Weight Constrained Path Finding with Bidirectional A*

Saman Ahmadi,[1] Guido Tack,[2] Daniel Harabor,[2] Philip Kilby[3]

[1] Department of Electrical and Biomedical Engineering, RMIT University, Australia
[2] Department of Data Science and Artificial Intelligence, Monash University, Australia
[3] CSIRO Data61, Australia
saman.ahmadi@rmit.edu.au, {guido.tack, daniel.harabor}@monash.edu, philip.kilby@data61.csiro.au

Abstract

Weight constrained path finding, known as a challenging variant of the classic shortest path problem, aims to plan cost optimum paths whose weight/resource usage is limited by a side constraint. Given the bi-criteria nature of the problem (i.e., the presence of cost and weight), solutions to the Weight Constrained Shortest Path Problem (WCSPP) have some properties in common with bi-objective search. This paper leverages the state-of-the-art bi-objective search algorithm BOBA* and presents WC-BA*, an exact A*-based WCSPP method that explores the search space in different objective orderings bidirectionally. We also enrich WC-BA* with two novel heuristic tuning approaches that can significantly reduce the number of node expansions in the exhaustive search of A*. The results of our experiments on a large set of realistic problem instances show that our new algorithm solves all instances and outperforms the state-of-the-art WCSPP algorithms in various scenarios.

Introduction

The Weight Constrained Shortest Path Problem (WCSPP) is well known as a technically challenging variant of the classical shortest path problem. The objective in the point-to-point WCSPP is to find a minimum-*cost* (shortest) path between two points in a graph such that the total *weight* (or resource consumption) of the path is limited. The WCSPP, as a core problem or a subroutine in larger problems, can be seen in various real-world applications in diverse areas such as transportation, robotics and game development. Typical examples can be route planning for bicycles where height difference between end points is to be limited (Storandt 2012), or solving the WCSPP as a sub-problem in the context of column generation (Zhu and Wilhelm 2012) or vehicle routing problems (Ahmadi et al. 2021b). The problem has been shown to be NP-complete (Handler and Zang 1980).

Constrained path finding is a difficult task, technically more difficult than the classic single-objective shortest path problem. This is because the search space becomes larger when exploring two (or more) dimensions and the search may have to deal with a significant number of paths even after rigorous pruning of unpromising ones. The WCSPP and its extended version with more than one side constraint, known as the Resource Constrained Shortest Path Problem (RCSPP), are well-studied topics in AI. Pugliese and Guerriero (2013) presented a summary of traditional exact solution approaches to the WCSPP and RCSPP by classifying them into three categories: path ranking, dynamic programming and branch-and-bound (B&B) approaches. Looking into the solution methods presented for constrained path finding over the last decade, we can still find effective methods from each category of solutions. Sedeño-Noda and Alonso-Rodríguez (2015) developed an enhanced path ranking approach called *CSP*, which was able to exploit pruning strategies of the B&B-like *Pulse* algorithm in (Lozano and Medaglia 2013). Thomas, Calogiuri, and Hewitt (2019) presented a dynamic programming approach to solve the RCSPP using bidirectional A* search. In their *RC-BDA** algorithm, the resource budget is equally divided between the forward and backward searches, allowing the searches to meet at 50% resource half-way points. They evaluated RC-BDA* on some of the instances of Sedeño-Noda and Alonso-Rodríguez (2015) for the WCSPP and reported 99% of the total instances solved within five hours. Cabrera et al. (2020) developed a parallel framework to execute Pulse of Lozano and Medaglia (2013) bidirectionally. To prevent the search falling into unpromising deep branches, bidirectional Pulse (*BiPulse*) limits the depth of the Pulse search and employs an adapted form of the queuing strategy proposed by Bolívar, Lozano, and Medaglia (2014). Their results show that BiPulse delivers better performance than both Pulse and RC-BDA* on medium-size instances, while leaving 3% of the instances unsolved after four hours of runtime. More recently, we improved RC-BDA* of Thomas, Calogiuri, and Hewitt (2019) for the WCSPP in (Ahmadi et al. 2021c) and proposed a dynamic programming framework called WC-EBBA* that can solve all instances of Sedeño-Noda and Alonso-Rodríguez (2015) within 10 minutes. In contrast to RC-BDA*, WC-EBBA* allows the search frontiers to meet at any fraction of the resource budget (not just at the 50% half-way point).

Given the success of A* in more effectively addressing difficult WCSPP instances, there are still some challenges that require attention if bidirectional A* is being used. Recent A*-based WCSPP methods (RC-BDA* and WC-EBBA*) work based on the traditional bidirectional search scheme where both searches explore the search space

in the same objective ordering (cost as the primary objective and weight for pruning) so that *storing* and *joining* partial paths are necessary to achieve the optimal path, which is known as collision of search frontiers. The frontier collision in such algorithms is space and time demanding as the number of paths grows exponentially. Further, there is a limited opportunity for improving the quality of heuristic functions used in such methods as both searches are working on the same dimension.

To address the shortcomings above, this paper leverages one of the recent time and space efficient bi-objective search algorithms in the literature and presents a new A*-based solution to the WCSPP called WC-BA*. Our WC-BA* algorithm avoids frontier collision by running individual (bidirectional) searches in different objective orderings. There are two main advantages: (i) the search can become more memory-efficient, as maintaining partial paths is no longer a necessity; (ii) the search space can shrink faster, as each search can improve the lower bounds for the other. We then enrich WC-BA* with two new heuristic tuning methods and empirically evaluate its algorithmic performance against the state of the art. The results show that our new algorithm is very effective in reducing the computation time of constrained search with A* in various scenarios. As our second contribution, we design a new set of 2000 challenging WCSPP instances, some of which require substantial computation time with the current technologies.

Problem Definition and Notation

Consider a directed graph $G = (S, E)$ with a finite set of states S and a set of edges $E \subseteq S \times S$. Every edge $e \in E$ has two non-negative attributes that can be accessed via the cost function $\mathbf{cost} : E \rightarrow \mathbb{R}^+ \times \mathbb{R}^+$. For the sake of simplicity in our algorithmic description, we replace the conventional $(cost, weight)$ attribute representation with $(cost_1, cost_2)$. Further, in our notation, every boldface function returns a tuple, so for the edge cost function we have $\mathbf{cost} = (cost_1, cost_2)$. A path is a sequence of states $u_i \in S$ with $i \in \{1, \dots, n\}$. The **cost** of path $\pi = \{u_1, u_2, u_3, \dots, u_n\}$ is then the sum of corresponding attributes on all the edges constituting the path, namely $\mathbf{cost}(\pi) = \sum_{i=1}^{n-1} \mathbf{cost}(u_i, u_{i+1})$. The Weight Constrained Shortest Path Problem (WCSPP) aims to find a $cost_1$-optimal path from $start \in S$ to $goal \in S$ such that the $cost_2$ of the optimum path is within the weight limit W.

We follow the standard notation in the heuristic search literature and define our search objects to be *nodes* (equivalent to partial paths). A node x is a tuple that contains the main information of the partial path to state $s(x) \in S$. Node x traditionally stores a value pair $\mathbf{g}(x)$ which measures the **cost** of a concrete path from an initial state to state $s(x)$, a value pair $\mathbf{f}(x)$ which is an estimate of the **cost** of a complete path from $start$ to $goal$ via $s(x)$; and also a reference $parent(x)$ which indicates the parent node of x.

We define two global upper bounds for the search, namely $\overline{\mathbf{f}} = (\overline{f_1}, \overline{f_2})$. $\overline{f_2}$ is the global upper bound on $cost_2$ of paths and can be initialised with the weight limit W. However, the initial value of $\overline{f_1}$ is not known beforehand and is updated during the search. We now describe the *validity* conditions.

Definition A path/node/state x is valid if its estimated costs $\mathbf{f}(x)$ are within the search global upper bounds $(\overline{f_1}, \overline{f_2})$, i.e., x is invalid if $f_1(x) > \overline{f_1}$ or $f_2(x) > \overline{f_2}$. In addition, path π is $cost_p$-valid if $cost_p(\pi) \leq \overline{f_p}$ for $p = 1, 2$.

We consider all operations of the boldface costs to be done element-wise. We also use $(\prec, \succ)$ or $(\preceq, \succeq)$ symbols in direct comparisons of boldface values, e.g. $\mathbf{g}(x) \preceq \mathbf{g}(y)$ denotes $g_1(x) \leq g_1(y)$ and $g_2(x) \leq g_2(y)$. In our notation, we generalise both possible search directions by searching in direction $d \in \{$*forward*, *backward*$\}$ from an *initial* state to a *target* state. Therefore, the (*initial*, *target*) pair would be $(start, goal)$ in the forward search and $(goal, start)$ in the backward search. In addition, we define d' to always be the opposite direction of d. To keep our notation consistent in the bidirectional setting, we always use the reversed graph or $Reversed(G)$ if we search backwards. We now define *dominance* over nodes generated in the same search direction.

Definition For every pair of nodes (x, y) associated with the same state $s(x) = s(y)$, we say node y is dominated by x if we have $g_1(x) < g_1(y)$ and $g_2(x) \leq g_2(y)$ or if we have $g_1(x) \leq g_1(y)$ and $g_2(x) < g_2(y)$. Node x weakly dominates y if $\mathbf{g}(x) \preceq \mathbf{g}(y)$.

Bidirectional Constrained A* Search

This section investigates the relationship between the Bi-Objective Shortest Path Problem (BOSPP) and the WCSPP. BOSPP methods aim to find a representative set of Pareto-optimal solution paths, i.e., a set in which every individual (non-dominated) solution offers a path that minimises the bi-objetive problem in both $cost_1$ and $cost_2$ (Miettinen 1998). In the WCSPP, however, there is only one cost-optimal solution path. Figure 1 depicts the relationship between a sample set of Pareto-optimal solutions (nodes in black, grey and red) and a cost-optimal solution of the WCSPP within the weight limit $\overline{f_2}$ (the red node marked Sol). As we see in the figure, Sol can be retrieved from the BOSPP solutions.

In this section, we consider our bidirectional bi-objective search algorithm BOBA* (Ahmadi et al. 2021a). BOBA* employs two bi-objective A* searches (Ulloa et al. 2020) to explore the graph in both forward and backward directions in the (f_1, f_2) and (f_2, f_1) order respectively, and grows two subsets of cost-unique Pareto-optimal solutions concurrently. From the WCSPP's point of view, if a cost-optimal solution path exists, it will always be in one of the subsets of BOBA*. Figure 1 highlights that Sol can be reached via two possible objective orderings: Sol is the first valid solution in the (f_1, f_2) order, whereas it is the last valid solution in the (f_2, f_1) order. With this introduction, we now discuss how BOBA* can be adapted for the WCSPP, namely by introducing its weight constrained variant WC-BA*.

Constrained Path Finding with WC-BA*

Our constrained A* search in direction d is guided by the $start$-$goal$ cost estimates or $\mathbf{f}$-values, which are traditionally established based on a consistent and admissible heuristic function $\mathbf{h}^d : S \rightarrow \mathbb{R}^+ \times \mathbb{R}^+$ (Hart, Nilsson, and Raphael 1968). In other words, for every search node x generated

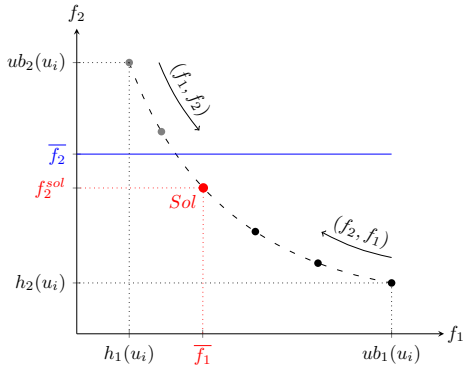

Figure 1: Schematic of a sample Pareto-optimal solution set (black, grey and red nodes) and also a solution Sol in the set (the red node) with the optimal cost pair of $(\overline{f_1}, f_2^{Sol})$ for the WCSPP with the weight limit $\overline{f_2}$. Nodes in grey are out-of-bounds for the WCSPP. h and ub-values are the lower and upper bounds on the **cost** of the initial state u_i respectively.

Algorithm 1: WC-BA* High-level

Input: The problem instance (G, **cost**, $start$, $goal$, W)
Output: A node corresponding with the cost-optimal feasible solution Sol

1 $\mathbf{h}, \mathbf{ub}, \overline{\mathbf{f}} \leftarrow$ Initialise(G, **cost**, $start$, $goal$, W) ▷ Alg. 2
2 $Sol \leftarrow \varnothing$
3 **do in parallel**
4 Run a WC-BA* search on (G, **cost**, $start$, $goal$) in the forward direction and (f_1, f_2) order with global upper bounds $\overline{\mathbf{f}}$, heuristic functions $(\mathbf{h}, \mathbf{ub})$ and initial solution Sol. Terminate the parallel search after the current search is complete. ▷ Alg. 3
5 Run a WC-BA* search on ($Rev(G)$, **cost**, $goal$, $start$) in the backward direction and (f_2, f_1) order with global upper bounds $\overline{\mathbf{f}}$, heuristic functions $(\mathbf{h}, \mathbf{ub})$ and initial solution Sol. Terminate the parallel search after the current search is complete. ▷ Alg. 3
6 **return** Sol

in direction d we have $\mathbf{f}(x) = \mathbf{g}(x) + \mathbf{h}^d(s(x))$ where $\mathbf{h}^d(s(x))$ estimates lower bounds on the **cost** of paths from state $s(x)$ to the target state of direction d. The heuristic function $\mathbf{h}^d$ can be established by conducting two simple unidirectional single-objective searches from target in the reverse direction d'. These unidirectional searches can also establish a set of upper bound functions $\mathbf{ub}^d : S \rightarrow \mathbb{R}^+ \times \mathbb{R}^+$. For every state $u \in S$, $\mathbf{ub}^d(u)$ denotes the upper bounds on **cost** of paths from state u to the target state in the search direction d, or equivalently, the upper bound on **cost** of paths from the initial state to state u in direction d'.

We now describe WC-BA*, the weight constrained search algorithm adapted from our bi-objective search algorithm BOBA* (Ahmadi et al. 2021a). WC-BA* is a bidirectional search algorithm that explores the search space in both objective orderings concurrently. Algorithm 1 shows the high-level design of WC-BA*. We follow BOBA*'s framework and propose a two-phase search, with parallel searches in

Algorithm 2: Initialisation Phase of WC-BA*

Input: The problem instance (G, **cost**, $start$, $goal$, W)
Output: Functions $\mathbf{h}$ and $\mathbf{ub}$ and global upper bounds $\overline{\mathbf{f}}$

1 Initialise global upper bounds: $\overline{f_1} \leftarrow \infty$ and $\overline{f_2} \leftarrow W$
2 **do in parallel**
3 $h_2^f, ub_1^f \leftarrow$ Run f_2-bounded backward A* on $cost_2$ (using an admissible heuristic), update $\overline{f_1}$ with $ub_1^f(start)$ when $start$ is going to get expanded and stop before expanding a state with $f_2 > \overline{f_2}$.
4 $h_1^b, ub_2^b \leftarrow$ Run f_1-bounded forward A* on $cost_1$ (using an admissible heuristic) and stop before expanding a state with $f_1 > \overline{f_1}$.
5 **do in parallel**
6 $h_2^b, ub_1^b \leftarrow$ Run f_2-bounded forward A* on $cost_2$ (using h_2^f as an admissible heuristic), ignore unexplored states in the previous round (lines 3-4), update $\overline{f_1}$ via paths matching if feasible and stop before expanding a state with $f_2 > \overline{f_2}$.
7 $h_1^f, ub_2^f \leftarrow$ Run f_1-bounded backward A* on $cost_1$ (using h_1^b an admissible heuristic), ignore unexplored states in the previous round (lines 3-4), update $\overline{f_1}$ via paths matching if feasible and stop before expanding a state with $f_1 > \overline{f_1}$.
8 **return** $(\mathbf{h}^f, \mathbf{h}^b)$, $(\mathbf{ub}^f, \mathbf{ub}^b)$, $\overline{\mathbf{f}}$

each phase. In the first phase, the algorithm establishes two sets of heuristic functions in both directions. It then initialises a solution node Sol and starts the second phase by running two parallel constrained searches in in different objective orderings. WC-BA* terminates as soon as one of the searches is complete. We explain each phase as follows.

Initialisation: Algorithm 2 shows the first phase of WC-BA* in two rounds in detail. The first round runs an f_1-bounded forward search and, in parallel, an f_2-bounded backward search to establish h_2^f and h_1^b respectively. This round includes initialising the global upper bound $\overline{f_1}$ using the $cost_2$-optimum path. Both searches have direct access to shared parameters, so the forward A* search on $cost_1$ will turn into a bounded search as soon as $\overline{f_1}$ gets updated by the concurrent backward search. The second round runs two complementary cost-bounded A* searches in parallel while benefiting from the lower bounds h_2^f and h_1^b obtained in the previous round as informed heuristics. The second round also performs two additional tasks: (i) it only explores the expanded states of round one, i.e., it disregards states already identified as out-of-bounds; (ii) it tries to update the global upper bound $\overline{f_1}$ by matching partial paths with their complementary optimum paths obtained via the first round.

Constrained Search: WC-BA* executes two concurrent constrained searches for its second phase. Following the search structure of BOBA*, we run one forward constrained A* search in the (f_1, f_2) order, and simultaneously, one backward constrained A* search in the (f_2, f_1) order, as shown in Algorithm 1. Both searches communi-

Algorithm 3: Constrained Search of WC-BA*

Input: Problem (G, **cost**, u_i, u_t), search direction d, objective ordering (f_p, f_s), global upper bounds $\overline{\mathbf{f}}$, heuristics $(\mathbf{h}, \mathbf{ub})$, and an initial solution Sol
Output: A node corresponding to an optimal solution

1 $Open^d \leftarrow \emptyset$
2 $g^d_{min}(u) \leftarrow \infty$ for each $u \in S$
3 $x \leftarrow$ new node with $s(x) = u_i$
4 $\mathbf{g}(x) \leftarrow (0,0)$, $\mathbf{f}(x) \leftarrow (h^d_1(u_i), h^d_2(u_i))$
5 $parent(x) \leftarrow \varnothing$
6 Add x to $Open^d$
7 $d' \leftarrow$ opposite direction of d
8 **while** $Open^d \neq \emptyset$ **do**
9 Remove from $Open^d$ node x with the lexicographically smallest (f_p, f_s) values
10 **if** $f_p(x) > \overline{f_p}$ **then break**
11 **if** $f_s(x) > \overline{f_s}$ **then continue**
12 **if** $g_s(x) \geq g^d_{min}(s(x))$ **then continue**
13 **if** $g^d_{min}(s(x)) = \infty$ **then**
14 $h^{d'}_p(s(x)) \leftarrow g_p(x)$
15 $g^d_{min}(s(x)) \leftarrow g_s(x)$
16 HTL/HTA(x, d, d') ▷ Procedure 4/5
17 **if** $g_s(x) + ub^d_s(s(x)) \leq \overline{f_s}$ **then**
18 **if** d= *forward* **then** $\overline{f_p} \leftarrow f_p(x)$
19 **else** $\overline{f_s} \leftarrow g_s(x) + ub^d_s(s(x))$
20 $Sol \leftarrow x$
21 **if** $h^d_p(s(x)) = ub^d_p(s(x))$ **then continue**
22 **for** *all* $v \in Succ(s(x))$ **do**
23 $y \leftarrow$ new node with $s(y) = v$
24 $\mathbf{g}(y) \leftarrow \mathbf{g}(x) + \mathbf{cost}(s(x), v)$
25 $\mathbf{f}(y) \leftarrow \mathbf{g}(y) + \mathbf{h}^d(v)$
26 $parent(y) \leftarrow x$
27 **if** $g_s(y) \geq g^d_{min}(v)$ **then continue**
28 **if** $\mathbf{g}(y) \not\preceq \mathbf{ub}^{d'}(v)$ **then continue**
29 **if** $\mathbf{f}(y) \not\preceq \overline{\mathbf{f}}$ **then continue**
30 Add y to $Open^d$
31 **return** Sol

cate with each other and have access to shared parameters of the search including the heuristic functions, global upper bounds, and best known solution. We abstract from the two possible attribute orderings $(1, 2)$ and $(2, 1)$ in our notation by using a pair (p, s) (for *primary* and *secondary*) with $p, s \in \{1, 2\}$ and $p \neq s$. Algorithm 3 presents the constrained search of WC-BA* in the generic (f_p, f_s) order.

Algorithm Description: Consider Algorithm 3 for a forward search in the (f_1, f_2) order. The algorithm first initialises essential data structures needed by the search, including the priority queue $Open^f$ and a search node associated with the *start* state. The algorithm also initialises for every $u \in S$ the scalar $g^f_{min}(u)$, a parameter that will keep track of the secondary cost of the last node successfully expanded for state u during the search in the forward direction, as in bi-objective A* (Ulloa et al. 2020). To be able to communicate with the concurrent search, the algorithm then sets d' to be the opposite direction *backward*. In every iteration of the forward search, the algorithm extracts the lexicographically smallest node from $Open^f$ in the (f_1, f_2) order. Let the extracted node be x. The search can terminate early if $f_1(x)$ is out-of-bounds. This is because A* guarantees that all future forward expansions will show a primary cost no smaller than $f_1(x)$ and will similarly not lead to a valid solution path. Otherwise, if $f_1(x) \leq \overline{f_1}$, there is still a chance for x to be an invalid node if it violates the global upper bound $\overline{f_2}$. In this case, x will be pruned via line 11.

Let x be a valid node in the forward search. In the next step, x is tested for dominance via line 12. The search compares x with the last node successfully expanded for state $s(x)$ in the forward direction. Since our forward A* explores nodes in the non-decreasing order of their f_1-values, we can observe that nodes expanded for $s(x)$ are also ordered based on their g_1-values. Therefore, if we keep track of the g_2-value of the last expanded node for $s(x)$ via $g^f_{min}(s(x))$, we can guarantee that node x is a weakly dominated node if $g_2(x) \geq g^f_{min}(s(x))$. Thus, x can be safely pruned. Otherwise, x is a non-dominated node and we can capture its secondary cost $g_2(x)$ via line 15. But before that, WC-BA* undertakes a strategy called Heuristic Tuning with First expansion (HTF) in lines 13-14 of Algorithm 3. HTF aims to improve the (secondary) heuristic function of the opposite direction upon the first expansion of the state, here $h^b_1(s(x))$. For the valid node x, if $g^f_{min}(s(x)) = \infty$, it means that there has not been any (successful) node expansion for $s(x)$ yet and x will represent the first valid path from $start$ to $s(x)$. Since the forward search explores nodes based on their f_1-value, A* guarantees that there would not be any valid path from $start$ to $s(x)$ with a better $cost_1$ than $g_1(x)$. Hence, it is always safe to use this shortest path to update heuristics of the opposite direction, in our case via $h^b_1(s(x)) \leftarrow g_1(x)$.

In the next step (line 17), the forward A* search tries to obtain a tentative solution by matching x with its complementary shortest path. Following the Early Solution Update (ESU) strategy in BOBA*, we join node x with the complementary shortest path on $cost_p$. The **cost** of this joined path is $(f_1(x), f'_2)$ where $f'_2 = g_2(x) + ub^f_2(s(x))$. Since x is already a valid node, we know that $f_1(x) \leq \overline{f_1}$. Therefore, the joined path is a tentative solution path if $f'_2 \leq \overline{f_2}$. In this case, the algorithm updates Sol with x and also the global upper bound $\overline{f_1}$ with $cost_1$ of the joined path ($f_1(x)$ in the forward search and f'_1 in the backward search). In the next step (line 21), the search skips expanding x if it is a terminal node, i.e., if we have $h^f_1(s(x)) = ub^f_1(s(x))$. This is because terminal nodes offer one complementary path which is optimum for both costs. Therefore, by joining x with such complementary shortest path we have $f_1(x) = g_1(x) + h^f_1(s(x))$ and $f'_2 = g_2(x) + ub^f_2(s(x)) = f_2(x)$. Since x is a valid node and has not been pruned by the global upper bounds, we can conclude that $f_1(x) \leq \overline{f_1}$ and $f_2(x) \leq \overline{f_2}$, which yields $f'_2 \leq \overline{f_2}$. Therefore, x is a tentative solution node and has already been captured by ESU.

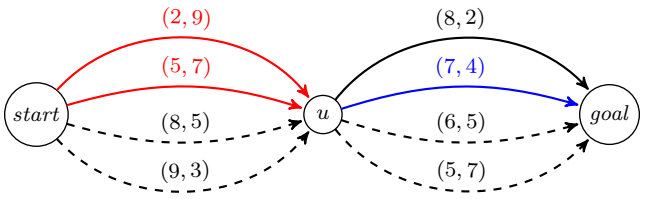

Figure 2: An example graph with four non-dominated paths to state u in each direction. Paths shown with dashed lines are future expansions. Given the global upper bound $\overline{f_2} = 10$, we can update $h_1^b(u) \leftarrow 6$ after expanding the second path of u in the backward direction (shown in blue). Tuples on the paths show $(cost_1, cost_2)$.

If x is not a terminal node, the algorithm extends x via the successor function $Succ(s(x))$ while performing validity and dominance tests via lines 23-30. If the extended path is valid, and if it is not weakly dominated by the last expanded node of the corresponding successor state, it will be added to $Open^f$ for further expansions. Finally, the forward search terminates if there is no node in $Open^f$ to explore.

New Heuristic Tuning Methods

We now describe our two new heuristic tuning methods for WC-BA*. These methods use previously expanded paths of states to improve secondary lower bounds. Our first new method HTA exploits **all** expanded paths whereas the second method HTL only uses the **last** expanded path. We explain these methods using an example.

Example: Figure 2 shows four non-dominated paths to state u in each direction. Forward (resp. backward) paths are ordered based on $cost_1$ (resp. $cost_2$). Assume WC-BA* has already expanded two paths in the forward direction (shown in solid red) and one path in the backward direction (shown in solid black) for state u. Paths shown with dashed lines are next (potential) expansions. In addition, after the first expansion of u in the forward direction, we have set $h_1^b(u) = 2$ via HTF. We now want to expand the second backward path of u (shown in blue with costs $(7, 4)$) where $\overline{f_2} = 10$.

The **HTA** method tries to match the backward path with **all** forward complementary paths reaching u. Here we have two paths and thus the actual costs of the joined paths will be $(9, 13)$ and $(12, 11)$. The joined paths are invalid as their $cost_2$ are out-of-bounds, i.e., $13 > 10$ and $11 > 10$. Since joining these two forward paths with the next backward paths of u would also violate the global upper bound $\overline{f_2}$, HTA picks the second forward path as the shortest feasible path on $cost_1$ and updates $h_1^b(u) \leftarrow 6$ accordingly.

The **HTL** method matches the new backward path with the **last** froward path only. Here, we can see that the joined path is invalid because its $cost_2$ is out-of-bounds. Hence, following HTA, HTL updates u's lower bound via $h_1^b(u) \leftarrow 6$.

HTA Method: Procedure 4 shows the pseudo-code of this tuning method in WC-BA*. We define for every state $u \in S$ in the search direction d a list $\chi^d(u)$ (initially empty) that stores expanded nodes with u in order. Let x be a valid

Procedure 4: HTA(x, d, d')

```
1 for all y ∈ χ^{d'}(s(x)) in order; do
2     f' ← g_p(x) + g_p(y)
3     if f' > f̄_p then
4         h_s^d(s(x)) ← g_s(y)
5         Remove y from χ^{d'}(s(x))
6     else
7         h_s^d(s(x)) ← g_s(y)
8         break
9 Add x to the end of χ^d(s(x))
```

Procedure 5: HTL(x, d, d')

```
1 f' ← g_p(x) + g^{d'}_{min}(s(x))
2 if f' > f̄_p then
3     h_s^d(s(x)) ← g^{d'}_{max}(s(x))
4 g^d_{max}(s(x)) ← g_p(s(x))
```

non-dominated node that the search aims to expand in direction d. The procedure HTA(x, d, d') tries to match x with all nodes previously expanded in the opposite direction d' with state $s(x)$ in order. These candidate nodes can be retrieved from the node list $\chi^{d'}(s(x))$. If the procedure finds a $cost_p$-valid joined path, it skips joining x with the remaining nodes in $\chi^{d'}(s(x))$. Otherwise, if the joined path is not $cost_p$-valid, HTA updates the $s(x)$'s lower bound and then removes the candidate node from $\chi^{d'}(s(x))$ since it will never be able to produce a $cost_p$-valid joined path. Finally, after the list is fully scanned, or after the first $cost_p$-valid joined path is found, the procedure stops the list scan and then adds x to the end of $\chi^d(s(x))$ to make it available for tuning the $s(x)$'s heuristic function in direction d'. Procedure 4 also shows a case where we can further improve lower bounds with the first $cost_p$-valid joined path (line 6). We discuss the correctness of this technique in Lemma 2.

HTL Method: Procedure 5 presents the pseudo-code of the HTL approach. The procedure in HTL is simpler than our first method and can be performed in constant time because we only have one candidate path. Therefore, we just need to track for every state the **cost** of its last expanded path. Since WC-BA* uses g^d_{min} to keep track of the secondary cost (g_s-value) of the last expanded path of each state, we introduce the cost array g^d_{max} to keep track of the primary costs (g_p-value) in direction d. For every state $u \in S$ we can initialise $g^d_{max}(u)$ to zero. The HTL(x, d, d') procedure first joins the current node x with the last expanded node in direction d' with state $s(x)$. Let f' be the primary cost of this joined path as shown in Procedure 5. If the joined path is not $cost_p$-valid, the procedure updates the secondary lower bound. In the last step, the procedure stores node x's primary cost $g_p(x)$ to keep the g^d_{max} array updated.

We now discuss the correctness of HTA and HTL.

Lemma 1 *Let x be the node just extracted from the priority queue of the WC-BA* search in direction d and in the*

(f_p, f_s) order. Also let y be a node already expanded in direction d' with $s(y) = s(x)$ in the (f_s, f_p) order. $g_s(y)$ is a lower bound on the $cost_s$ of complementary paths from $s(y)$ to the target state in direction d if $g_p(x) + g_p(y) > \overline{f_p}$.

Proof We need to show that there is no valid complementary path from $s(y)$ to the target state with a $cost_s$-value smaller than $g_s(y)$. We observe two cases:

(i) The primary heuristic functions always remain unchanged during our constrained searches and thus A* in direction d (with the consistent heuristic h_p^d) always expands nodes associated with the same state in a non-decreasing order of their g_p-values. Therefore, if we observe $g_p(x) + g_p(y) > \overline{f_p}$ upon extraction of x, we can guarantee that all future expansions of $s(x)$ in direction d would similarly satisfy the inequality above if we join them with y. In other words, neither x nor next expansions of $s(x)$ in direction d would yield a $cost_p$-valid joined path via y.

(ii) WC-BA* in direction d' enumerates all paths to $s(x)$ in a non-decreasing order of their g_s-value. Furthermore, since both searches rigorously prune weakly dominated nodes, we can also see that the g_s-value of the expanded nodes with $s(x)$ is monotonically decreasing. Therefore, if node y in direction d' shows $g_p(x) + g_p(y) > \overline{f_p}$, we can guarantee that all nodes expanded before y with $s(y)$ would similarly show a $cost_p$-value larger than $\overline{f_p}$ after being joined with x.

According to observations (i) and (ii) above, there does not exist a complementary path (from $s(x)$ to target) shorter than the sub-path represented by y (on $cost_s$) such that it can establish a $cost_p$-valid path between $start$ and $goal$. Therefore, we can guarantee the admissibility of our secondary heuristic function after updating $h_s^d(u)$ with $g_s(y)$. □

Given our tuning methods HTA and HTL from Procedures 4 and 5, we can see that the correctness of both methods is directly derived from Lemma 1.HTL only updates the lower bound if it does not find the $cost_p$ of the joined path within the global upper bound $\overline{f_p}$, whereas HTA successively checks previously expanded nodes in the opposite direction until it finds a $cost_p$-valid joined path. Furthermore, we can see that the correctness of the tuning condition does not depend on the existence of all nodes expanded in the opposite direction. Therefore, we can remove paths already scanned or even keep the last k expanded paths to terminate the likely lengthy list scan in HTA sooner.

We now prove the correctness of tuning with the first $cost_p$-valid path shown at line 6 of Procedure 4.

Lemma 2 *Assume the situation described in Lemma 1where we have $g_p(x) + g_p(y) > \overline{f_p}$. Also suppose that z is the first node expanded after y in direction d' with $s(y) = s(z)$. $g_s(z)$ is a lower bound on the $cost_s$ of complementary paths from $s(x)$ to the target in direction d if $g_p(x) + g_p(z) \not> \overline{f_p}$.*

Proof z is expanded after y, so we have $g_s(y) \leq g_s(z)$. In addition, the A* search in direction d' guarantees that there is no non-dominated node z' expanded with $s(y)$ for which we have $g_s(y) \leq g_s(z') < g_s(z)$. In addition, the proof in Lemma 1shows that y will never be able to establish a $cost_p$-valid joined path with future expansions of $s(y)$ in direction d. Therefore, z represents a minimum $cost_s$ complimentary path that is able to establish a $cost_p$-valid joined path via x and thus the heuristic function h_s^d remains admissible after updating $h_s^d(u)$ with $g_s(z)$. □

Note that Lemma 2requires HTA to keep at least the last two expanded nodes of each state, essentially because the correctness of tuning via the first $cost_p$-valid joined path depends on the previous joined path being invalid. We now show the correctness of constrained search with WC-BA*.

Theorem 1 *WC-BA* returns a node corresponding to a cost-optimal solution path for the WCSPP.*

Proof WC-BA* conducts two constrained searches in parallel. In each search, it enumerates all paths in best-first order and prunes dominated and invalid nodes. Dominated and out-of-bound nodes are not part of the cost-optimum solution path because such partial paths can always be replaced with a path that is shorter at least in one attribute. In addition, we already showed how early termination, ESU and tuning methods respectively preserve the correctness of A*, solution update and the admissibility of heuristic functions. Therefore, we just need to show that, if one of the constrained searches terminates, the other search can be stopped accordingly. This termination criterion in the high-level structure of WC-BA* is always correct because each individual search in WC-BA* is able find an optimal solution by itself. Therefore, when either of the searches terminates, it guarantees the optimality of the solution and thus the concurrent search can be stopped accordingly. □

Empirical Analysis

We compare our new algorithm with the state-of-the-art solution approaches designed for the WCSPP and RCSPP. The selected algorithms are the recent B&B method BiPulse (Cabrera et al. 2020), the path ranking method CSP (Sedeño-Noda and Alonso-Rodríguez 2015), and our WC-EBBA* algorithm in (Ahmadi et al. 2021c).

Benchmark Setup: Given the success of WC-EBBA* in solving all instances of the literature in under 10 minutes, we designed a larger benchmark set with 2000 easy-to-hard realistic instances to evaluate our new algorithm. Following the literature, we use road networks in the 9th DIMACS Implementation Challenge (DIMACS 2005) with the largest map (USA) containing around 24 million nodes and 57 million edges with distance and time as edge attributes (representing cost and weight respectively). We started with the bi-objective instances in (Sedeño-Noda and Colebrook 2019; Ahmadi et al. 2021a). We then used the competition's random pair generator to produce an additional set of 200 random instances for two other large maps of the DIMACS challenge: NW and USA (100 instances each). We solved all of the 1200 bi-objective instances and then sorted them based on their number of Pareto-optimal solution paths as a measure of difficulty. The number of Pareto-optimal solutions in the benchmark set ranges from 100 (in the NY map) to 70,000 (in the USA map). Next, we evenly sampled 10 easy-to-hard instances (out of 100) from each map, plus five additional instances from the USA to further challenge the algorithms in very large graphs. Thus, in total, we have 125 pairs over 12 maps. We then doubled the number of instances by adding reversed pairs to account for the impacts of search direction. Following the literature, we then

define the weight limit W based on the tightness of the constraint $\delta = (W - h_2)/(ub_2 - h_2)$ where h_2 and ub_2 are respectively lower and upper bounds on $cost_2$ of $start\text{-}goal$ paths. In this setup, high (resp. low) values of δ mean that the weight limit W is loose (resp. tight). Given the tightness levels $\delta \in \{10\%, 20\%, \ldots, 80\%\}$, we obtain 2000 cases.

Implementation: We implemented our WC-BA* in C++ and used the C implementation of the CSP algorithm and also the Java implementation of the BiPulse algorithm kindly provided to us by their authors (Sedeño-Noda and Alonso-Rodríguez 2015; Cabrera et al. 2020). For WC-EBBA*, we used its C++ implementation. For the sake of fairness, we implemented our WC-BA* with the same type of priority queue and backtracking approach used for WC-EBBA*. All C/C++ code was compiled with O3 optimisation settings using the GCC7.5 compiler. The Java code was compiled with OpenJDK version 1.8.0_292. We ran all experiments on an AMD EPYC 7543 processor running at 2.8 GHz and with 128 GB of RAM, under the SUSE Linux Server 15.2 environment and with a one-hour timeout. In addition, we allocated two CPU cores to all parallel algorithms. In order to achieve more consistent computation time, we perform three consecutive runs of each algorithm (per instance) and store the results of the run showing the median runtime. Our codes and benchmark instances are publicly available at https://bitbucket.org/s-ahmadi.

Performance Impact of New Tuning Methods

We first study the impacts of our tuning methods on the performance of WC-BA* by evaluating three variants of the algorithm. The first variant (our baseline) only employs HTF. The second and third variants utilise the new methods HTL and HTA on top of the HTF method respectively. For HTA, we use dynamic arrays to store for every state only the costs of its expanded paths during the search. Furthermore, we trade space for time and do not resize the dynamic array after every lower bound update. Instead, we keep track of the index of the last tested cost pair in each dynamic array using pointers. We found this around 15% faster but 20% less memory-efficient than HTA implemented with linked-lists.

Table 1 shows the runtime and total number of node expansions for the WC-BA* algorithm with HTF (baseline) and also for the baseline equipped with the HTA or HTL methods on a subset of maps. According to the results, we can see that both HTA and HTL methods have been successful in reducing the number of node expansions in WC-BA*. Comparing the average values across all maps, we observe 16-42% fewer node expansions when HTA is being used, and also 8-27% reduction in the total number of expansions when HTL is selected for heuristic tuning. In particular, the maximum number of node expansions drops by 50% in the CTR map after tuning lower bounds with HTA, reducing 152 million maximum node expansions of the baseline to 64 million expansions. We see the same pattern when comparing the runtime values and both methods contribute to faster computation time than solely using HTF. We can see WC-BA* with HTA as the best performer of all variants, showing up to 32% and 52% speedups (with respect to HTF) over the maps for the average and maximum runtimes respectively.

	Tuning	Runtime(s)		#Expansions	
Map	Method	Avg.	Max.	Avg.	Max.
CAL	HTF	1.93	23.48	10.5×10^6	151.8×10^6
	+HTL	1.60	18.79	7.7×10^6	115.0×10^6
	+HTA	1.54	11.31	6.1×10^6	63.6×10^6
W	HTF	41.86	382.72	197.5×10^6	1.5×10^9
	+HTL	36.04	318.35	167.3×10^6	1.3×10^9
	+HTA	34.68	311.15	148.1×10^6	1.3×10^9
CTR	HTF	79.40	636.64	359.8×10^6	2.4×10^9
	+HTL	68.33	549.08	297.9×10^6	2.0×10^9
	+HTA	62.43	414.02	251.9×10^6	1.6×10^9
USA	HTF	434.59	3594.13	1.8×10^9	13.9×10^9
	+HTL	367.39	3195.64	1.5×10^9	11.6×10^9
	+HTA	357.12	3098.93	1.3×10^9	10.3×10^9

Table 1: WC-BA*'s performance using heuristic tuning.

Memory: We also compared the memory requirement of WC-BA* over the instances before and after incorporating the new tuning methods. The detailed results illustrate that HTA consumes more space than HTL and HTF due to storing all paths. However, compared to the baseline, WC-BA* consumes around 19% less memory on average when we incorporate the HTL method. This is potentially because HTL contributes to fewer node expansions while only keeping track of the costs of the last expanded path of each state via fixed-size data structures. Therefore, we conclude that WC-BA* can take advantage of HTL to improve both runtime and memory requirement. Comparing the memory values across all instances, we see that the memory consumed by using the HTA method is around 62% larger than the memory consumption of the baseline on average. This highlights that the runtime saved by expanding fewer nodes via HTA in WC-BA* outweighs the time HTA needs to store and scan all paths during the search.

Algorithmic Performance

We now compare the performance of our WC-BA* against the state of the art. For this analysis, we consider WC-BA* equipped with the HTA method. We also realised that BiPulse is unable to handle our two largest maps CTR and USA due to some inefficiencies in its graph implementation. We report for each algorithm the number of solved cases and runtimes. We report the timeout (one hour) as the runtime of unsolved cases. Table 2 shows the detailed results.

Solved Cases: All of the algorithms have been able to fully solve all instances of five maps. For the BiPulse and CSP algorithms, however, we can see they both have been struggling with some of the instances of the NE, LKS, E and W maps. In the E map, for example, they cannot solve about 37% of instances within the timeout. In particular, CSP only solves less than half of the instances of the CTR map and just about 30% of the USA map. For the A*-based algorithms WC-EBBA* and WC-BA*, we can see far better performance as they both have been able to solve all instances.

Map	Algorithm	$\|S\|$	Runtime(s) Min	Avg.	Max	Map	Algorithm	$\|S\|$	Runtime(s) Min	Avg.	Max
NY	WC-BA*	160	**0.01**	**0.06**	0.18	BAY	WC-BA*	160	**0.01**	**0.10**	**0.33**
	WC-EBBA*	160	**0.01**	0.08	**0.16**		WC-EBBA*	160	**0.01**	0.13	0.46
	CSP	160	0.11	1.43	12.46		CSP	160	0.13	3.25	35.05
	BiPulse	160	0.46	0.85	2.35		BiPulse	160	0.54	1.08	4.05
COL	WC-BA*	160	**0.03**	**0.19**	**0.94**	FLA	WC-BA*	160	**0.17**	**1.08**	**4.28**
	WC-EBBA*	160	0.04	0.25	1.01		WC-EBBA*	160	0.22	1.29	4.99
	CSP	160	0.18	20.54	303.50		CSP	160	1.13	240.78	2925.19
	BiPulse	160	0.73	2.85	27.46		BiPulse	160	1.47	35.29	396.44
NW	WC-BA*	160	**0.10**	**1.87**	**15.25**	NE	WC-BA*	160	**0.15**	**1.96**	**27.50**
	WC-EBBA*	160	0.13	2.11	16.20		WC-EBBA*	160	0.21	2.67	32.00
	CSP	160	0.55	390.56	3181.55		CSP	156	0.83	388.98	3600.00
	BiPulse	160	1.58	163.50	1440.52		BiPulse	156	2.11	237.58	3600.00
CAL	WC-BA*	160	**0.11**	**1.54**	**11.31**	LKS	WC-BA*	160	0.08	**22.98**	**275.05**
	WC-EBBA*	160	0.12	1.93	11.71		WC-EBBA*	160	**0.07**	29.14	349.12
	CSP	150	0.97	573.27	3600.00		CSP	104	1.39	1691.62	3600.00
	BiPulse	160	2.26	154.10	2640.61		BiPulse	109	2.76	1439.84	3600.00
E	WC-BA*	160	**0.08**	**32.40**	**351.01**	W	WC-BA*	160	0.46	**34.68**	**311.15**
	WC-EBBA*	160	0.10	43.15	418.95		WC-EBBA*	160	**0.36**	40.62	367.12
	CSP	102	2.70	1876.37	3600.00		CSP	104	3.55	1891.69	3600.00
	BiPulse	99	3.66	1617.31	3600.00		BiPulse	101	7.76	1620.46	3600.00
CTR	WC-BA*	160	0.30	**62.43**	**414.02**	USA	WC-BA*	240	**0.15**	**357.12**	3098.93
	WC-EBBA*	160	**0.29**	99.39	758.78		WC-EBBA*	240	0.26	379.19	**2120.83**
	CSP	71	11.29	2285.69	3600.00		CSP	67	14.62	2782.51	3600.00

Table 2: Number of solved cases $|S|$ and runtime of the algorithms. Runtime of unsolved instances is assumed to be one hour.

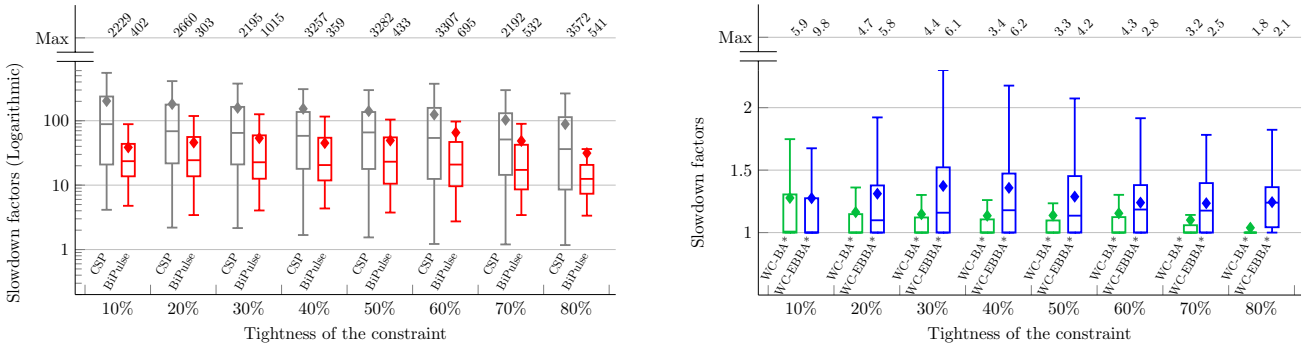

Figure 3: The distribution of the slowdown against the virtual best oracle in each level of tightness. Each box plot shows minimum, first quartile (25% data), median, third quartile (75% data), mean and maximum. Outliers are not shown.

Runtime: We report minimum, average and maximum runtime of the algorithms in each map. Boldface values in Table 2 denote the best runtimes. The results indicate that the runtime of all algorithms increases with the graph size and we have obtained larger values in larger graphs. Comparing the average runtime of BiPulse and CSP against the A*-based algorithms, we see they can be up to two orders of magnitude slower based on the average values. Among the A*-based methods, we observe that WC-BA* outperforms the state-of-the-art WC-EBBA* algorithm on all maps in terms of average runtime and in 10 maps (out of 12) in terms of maximum runtime. In particular, WC-BA* in the CTR map preforms approximately 60% faster than WC-EBBA* based on the average runtimes, which results in saving around one hour in the total time required for solving all CTR instances if we opt for the faster algorithm WC-BA*.

Memory: As we expected, our implementation of HTA with dynamic arrays turned out not to be space efficient as we did not resize path lists in HTA. However, the results show that WC-BA* with HTL can effectively reduce the memory requirement and shows up to 35% better memory usage compared to WC-EBBA* on average values.

Constraint Tightness: To study the strengths and weaknesses of the algorithms across various levels of tightness, we define our baseline to be the virtual best oracle, i.e., for every instance, the virtual oracle is given the best runtime of

all algorithms. Given the virtual best oracle as the baseline, we then calculate for every runtime (across all algorithms) a slowdown factor, i.e, algorithms with slowdown factors close to *one* are as good as the virtual best oracle. For the sake of better readability, Figure 3 shows the range of slowdown factors obtained for each algorithm across all levels of tightness in two plots. We do not show outliers in the box-plots, but the maximum slowdown for each constraint is presented above the corresponding plot separately.

The plots show that both CSP and BiPulse perform poorly in all levels of tightness with their median slowdown factors ranging from one to two orders of magnitude. For CSP, in particular, the maximum values show cases where this path ranking approach is up to three orders of magnitude slower than the virtual best oracle. For the A*-based approaches WC-BA* and WC-EBBA*, however, we see far smaller slowdown factors. Given their maximum slowdowns, we notice that they are all smaller than 10 but we see relatively larger values for WC-EBBA*. For the very tight 10% constraint (which can be seen as an edge case), both algorithms behave nearly the same but we see around 1.7 times smaller value when comparing them in terms of worst performance (max values). On the other constraint levels, which are more challenging and also more interesting in various practical settings, we can focus on the range that shows the middle 50% distribution and not just the corner cases. For slowdowns in this range, we see just up to 10% slower performance with WC-BA*, whereas WC-EBBA* is up to 50% slower than the virtual best oracle. Note that the most difficult WCSPP instances are normally located in the mid-range constraints (30-50% level) where we see far larger maximums (up to 2.5) with WC-EBBA*. This considerable performance difference denotes the success of WC-BA* in more efficiently solving large WCSPP instances than the existing (already fast) algorithm. In summary, WC-BA* is the best-performing algorithm in almost all levels of tightness.

Conclusion

This paper presented a new solution approach to the Weight Constrained Shortest Path Problem (WCSPP) called WC-BA*: a bidirectional A* search algorithm derived from the recent bi-objective search method BOBA*. WC-BA* explores the graph via different attribute orderings concurrently. We enriched our new algorithm with two novel heuristic tuning methods that can reduce the significant number of expansions in the exhaustive search of A* by up to 50%. We evaluated WC-BA* on very large graphs from a new set of 2000 challenging instances and compared its performance against the recent algorithms in the literature. The results show that WC-BA* improves the runtime over the state-of-the-art algorithms, outperforming the recent fast WCSPP algorithm by up to 60% in various constrained problem instances.

Acknowledgments

This research was partially supported by the Australian Research Council (ARC) grants DP190100013 and DP200100025 as well as a gift from Amazon.

References

Ahmadi, S.; Tack, G.; Harabor, D.; and Kilby, P. 2021a. Bi-Objective Search with Bi-Directional A*. In Mutzel, P.; Pagh, R.; and Herman, G., eds., *ESA 2021*, volume 204 of *LIPIcs*, 3:1–3:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Ahmadi, S.; Tack, G.; Harabor, D.; and Kilby, P. 2021b. Vehicle Dynamics in Pickup-And-Delivery Problems Using Electric Vehicles. In Michel, L. D., ed., *CP 2021*, volume 210 of *LIPIcs*, 11:1–11:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Ahmadi, S.; Tack, G.; Harabor, D. D.; and Kilby, P. 2021c. A Fast Exact Algorithm for the Resource Constrained Shortest Path Problem. In *AAAI 2021*, 12217–12224. AAAI Press.

Bolívar, M. A.; Lozano, L.; and Medaglia, A. L. 2014. Acceleration strategies for the weight constrained shortest path problem with replenishment. *Optim. Lett.*, 8(8): 2155–2172.

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.

DIMACS. 2005. 9th DIMACS Implementation Challenge - Shortest Paths. http://www.dis.uniroma1.it/~challenge9. Accessed: 2022-05-22.

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.

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

Miettinen, K. 1998. *Nonlinear multiobjective optimization*, volume 12 of *International series in operations research and management science*. Kluwer. ISBN 978-0-7923-8278-2.

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

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.

Sedeño-Noda, A.; and Colebrook, M. 2019. A biobjective Dijkstra algorithm. *Eur. J. Oper. Res.*, 276(1): 106–118.

Storandt, S. 2012. Route Planning for Bicycles - Exact Constrained Shortest Paths Made Practical via Contraction Hierarchy. In *ICAPS 2012*. AAAI.

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.

Ulloa, C. H.; Yeoh, W.; Baier, J. A.; Zhang, H.; Suazo, L.; and Koenig, S. 2020. A Simple and Fast Bi-Objective Search Algorithm. In *ICAPS 2020*, 143–151. AAAI Press.

Zhu, X.; and Wilhelm, W. E. 2012. A three-stage approach for the resource-constrained shortest path as a sub-problem in column generation. *Comput. Oper. Res.*, 39(2): 164–178.