A Fast Exact Algorithm for the Resource Constrained Shortest Path Problem

Saman Ahmadi,[1,2] **Guido Tack,**[1] **Daniel Harabor,**[1] **Philip Kilby**[2]
[1] Department of Data Science and Artificial Intelligence, Monash University, Australia
[2] CSIRO Data61, Australia
{saman.ahmadi, guido.tack, daniel.harabor}@monash.edu, philip.kilby@data61.csiro.au

Abstract

Resource constrained path finding is a well studied topic in AI, with real-world applications in different areas such as transportation and robotics. This paper introduces several heuristics in the resource constrained path finding context that significantly improve the algorithmic performance of the initialisation phase and the core search. We implement our heuristics on top of a bidirectional A* algorithm and evaluate them on a set of large instances. The experimental results show that, for the first time in the context of constrained path finding, our fast and enhanced algorithm can solve all of the benchmark instances to optimality, and compared to the state of the art algorithms, it can improve existing runtimes by up to four orders of magnitude on large-size network graphs.

Introduction

The Resource Constrained Shortest Path Problem (RCSPP) is a well-known NP-hard problem (Handler and Zang 1980). It has important real-world applications in diverse areas such as robotics, transportation and game development. The solution to a RCSPP is a minimum-cost path that consumes a limited amount of resources. Typical examples are finding a shortest distance path between two locations which can be traversed within a fixed time limit, or the quickest path between two locations within a certain energy budget. Formally, given a directed graph G with nodes V and arcs A, the task is to find a minimum-cost path p from $source \in V$ to $target \in V$ such that $\sum_{(i,j)\in p} r_{ij} \leq R$ where r_{ij} is the resource usage of the arc between node *i* and node *j* and R is the given upper bound on the total resource consumption.

A summary of works on traditional approaches to the RCSPP and its elementary counterpart (when cycles also need to be handled) such as path ranking (Santos, Coutinho-Rodrigues, and Current 2007) and dynamic programming (Righini and Salani 2008) was presented by Pugliese and Guerriero (2013). Extensions to these approaches can also be found in the recent literature. Lozano and Medaglia (2013) implemented a dynamic programming solution which explores the graph using a depth-first search (DFS) scheme. In their *Pulse* algorithm, the search is redirected if the current partial path looks unpromising by using three pruning strategies. The Pulse algorithm has been shown to outperform the *Lagrangian Relaxation* algorithm of Santos, Coutinho-Rodrigues, and Current (2007) and the *Label Setting* algorithm of Zhu and Wilhelm (2012) on small-size instances. Given the pruning strategies introduced by Lozano and Medaglia (2013), Sedeño-Noda and Alonso-Rodríguez (2015) presented an exact path ranking approach based on the *K-Shortest Paths (K-SP)* algorithm to solve the RCSPP in large networks. Their results show that K-SP outperforms Pulse on several instances.

Thomas, Calogiuri, and Hewitt (2019) adapted bidirectional A* search to the RCSPP (*RC-BDA**). Their algorithm integrated the pruning strategies of Lozano and Medaglia (2013) and was compared with Pulse and K-SP on the same instances of Sedeño-Noda and Alonso-Rodríguez (2015). The results show a runtime improvement over some difficult instances, but the algorithm is not competitive with Pulse and K-SP on many simple instances. Another recent work in the RCSPP context is the bidirectional version of the Pulse algorithm (Cabrera et al. 2020). Bidirectional Pulse (*Bi-Pulse*) is practically a combination of the DFS and the breadth-first search (BFS) strategy which recursively explores the graph in both directions. The experimental results of their study show that the parallel implementation of the Bi-Pulse algorithm delivers better performance compared to the normal Pulse algorithm and RC-BDA* on relatively small-size instances, but still fails to find optimal solutions for several large-size instances even with a large time limit.

Contribution: In this paper, we present an enhanced bidirectional A* algorithm for the RCSPP and introduce several heuristics for the initialisation phase and the main search of the algorithm that can together significantly improve its algorithmic performance. We design our Enhanced Biased Bidirectional A* (*RC-EBBA**) algorithm on top of the basic bidirectional A* search from the literature and evaluate it under a set of 440 benchmark instances. The experiments show that RC-EBBA* outperforms the state-of-the-art algorithms, which have already shown good performance on large-scale instances by one to four orders of magnitude.

Background

Bidirectional A* search was first presented by Pohl (1971). It runs two individual A* searches (Hart, Nilsson, and

Raphael 1968), one in the forward direction (from source to target) and the one other backwards (from target to source), using the corresponding heuristics in each direction. In the RCSPP context, these heuristics are normally obtained by finding the minimum cost path from *source* to every node $v \in V$, and from every v to *target*, as depicted in Figure 1. In this figure, $\underline{g}^f$ denotes the cost of the shortest path form the source node to node v in the forward direction where the resource usage of this path is denoted by $r^f_{\underline{g}}$. Analogously, the cost and resource usage of the shortest path from node v to the target node in the backward direction are denoted by $(\underline{g}^b, r^b_{\underline{g}})$. For every partial path p from source to node v in the direction $d \in \{f, b\}$ with the cost of g^d and resource consumption of r^d, the A* search establishes a lower bound on the cost of a complete path from source to target as $f^d = g^d + \underline{g}^{d'}$ where $\underline{g}^{d'}$ is the cost estimate of the complementary path in the reverse direction d'. Bidirectional A* search tries to find a feasible and complete minimum cost path by expanding partial paths showing the lowest overall cost from both directions using two priority queues. Each priority queue contains unexplored partial paths generated in one direction.

Initial solution: An initial solution for the RCSPP is a resource optimum path between source and target. The cost of this path is an initial upper bound C^*_0 on the cost. Therefore, when the cost of complete paths is concerned, the search can ignore partial paths with an overall cost greater than the initial upper bound, i.e. $f^d > C^*_0$.

Infeasible paths: A complete path is *infeasible* if its resource usage exceeds the resource limit. A common method to identify infeasible partial paths is by estimating the overall resource usage of the complete path based on a known lower bound on the resource consumption of the complementary path. In the bidirectional scheme, lower bounds on the resource consumption can be obtained by finding resource optimum paths in both directions as shown in Figure 1. In this figure, in every direction d, $\underline{r}^d$ denotes the resource usage of a resource optimum path from source to node v where the cost of this path is denoted by $g^d_{\underline{r}}$. Therefore, a partial path can be safely pruned if the estimated resource usage of its complete path is greater than the resource limit, i.e., $r^d + \underline{r}^{d'} > R$ where $\underline{r}^{d'}$ is the minimum resource usage in the reverse direction d'.

Dominated paths: In cases where there is more than one incoming partial path for a node, the search needs to ignore expanding partial paths for which there is at least one better partial path with a lower cost and resource consumption. To this end, we say the partial path p_1 is dominated by p_2 if $g^d_1 \geq g^d_2$ and $r^d_1 \geq r^d_2$.

Joined paths: In bidirectional search, every partial path to a node in one direction can be coupled with any of the partial paths to that node in the reverse direction to make a complete path between source and target. If the resulting joined path is feasible (its resource consumption is within the resource limit), it will be added into the corresponding priority queue as a complete path. Therefore, the algorithm can stop the search and return the minimum cost path as soon as a complete joined path is going to be expanded.

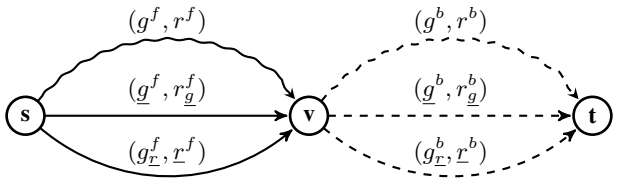

Figure 1: Partial paths in both directions.

Enhanced Biased Bidirectional A* for the Resource Constrained Shortest Path Problem

We now introduce our contributions to the RCSPP by presenting new heuristics and strategies which together make our Enhanced Biased Bidirectional A* (RC-EBBA*) the fastest available algorithm for solving the RCSPP. We first describe our improvements to the initialisation phase, followed by our contributions to the main search.

Initialisation of RC-EBBA*

Classic algorithms for the RCSPP need to know lower bounds on both cost and resource to be able to effectively prune unpromising partial paths by checking them against the corresponding upper limits. In this section, we show how to improve the search for lower bounds, and that there are several interesting heuristics that can be directly derived from this important phase.

Faster approach: A common method to obtain cost/resource lower bounds is running the one-to-all version of Dijkstra's algorithm on cost/resource prior to the main search of the RCSPP. Therefore, the execution time of the initialisation phase of the RCSPP is at least the time needed to run two one-to-all searches (one on cost and one on resource). For bidirectional search, initialisation phase would need to run an additional two one-to-all searches in the reverse direction. This approach can cause inefficiencies in practice, especially for easy instances on large graphs. We now show that the conventional initialisation phase of the RCSPP can be replaced with faster techniques such as a bounded version of Dijkstra's algorithm or bounded A* if a consistent and admissible heuristic exists.

Lemma 1. *The distance-bounded version of the Dijkstra's algorithm or distance-bounded A* with a consistent and admissible heuristic can be used to obtain the necessary lower bounds for the RCSPP.*

Proof Sketch: Given R and C^*_0 as resource and cost limits respectively, we know that the main search of the RCSPP will prune all paths with a cost/resource consumption greater than the corresponding upper bound. Using the natural property of Dijkstra's algorithm that nodes are expanded based on their minimum distance to the source, we can simply stop the search on a node for which the minimum distance is observed to be greater than the given cost/resource limit. In other words, nodes with a minimum distance greater than the upper limit can not be in a feasible solution path and, therefore, will not be explored in the RCSPP search. This analysis can be extended to A* with admissible heuristics as shown in Figure 2. Analogously, since A* is guaranteed to find an

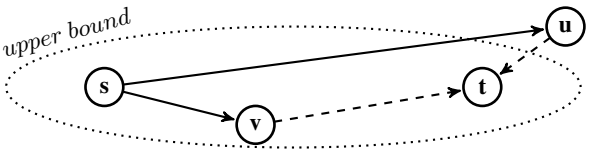

Figure 2: Nodes inside/outside of the upper bound.

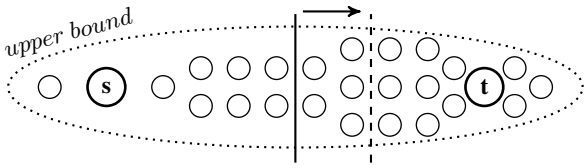

Figure 3: An unbalanced node distribution and budget share.

optimal solution and nodes are expanded based on their estimated distance to target, we can stop A* on the first node with an f-value larger than the given upper limit, knowing that the unexpanded nodes would violate the cost/resource limit if they were located on the solution path. □

Better quality heuristics: Our bounded search approach enables us to potentially remove unexpanded nodes from the RCSPP search space: Nodes that have not been expanded in any of our searches in the initialisation phase, either on cost or resource, are guaranteed not to be on the solution path. Now we present a new procedure which can significantly improve the quality of cost/resource lower bounds. Let us assume that we first perform a bounded search on *resource* and $V' \subseteq V$ is the subset of nodes expanded during this search. Using the main property of our bounded search, since the unexpanded nodes in $\{V - V'\}$ are not part of the solution path, we can now run our second search on *cost* just using the resource-valid nodes in V'. Therefore, as our second bounded search explores a smaller set of nodes, it eventually gives us better quality heuristics on *cost* compared to the base case where all of the nodes in V are explored. Furthermore, let us assume $V'' \subseteq V'$ is the subset of nodes our second bounded search expands. Our main search for the RCSPP now only needs to explore nodes in V'', which have already been shown to have a cost/resource heuristic within the upper bounds. If we prefer to improve the quality of the resource heuristic, rather than the cost heuristic, we can simply change the order of our searches on cost and resource.

More informed A*: When running the initialisation phase for the bidirectional search, one of our bounded A* can benefit from the result of its counterpart in the reverse direction. For example, if the first bounded A* search has been done on *cost* in the backward direction, our next bounded A* search on *cost* in the forward direction can use a more informed heuristic based on the minimum costs obtained in the backward direction. This technique will help us to speed up our complementary search in the reverse direction and also empowers our bounded search by invalidating more nodes in the corresponding cost/resource search.

Better initial solutions: In cases where the cost of the resource optimum path is known as C_0^*, the RCSPP search will try to close the gap between the minimum cost (feasible solution) and the upper bound. Now we argue that this gap can be shrunk or even closed altogether during the initialisation phase if a bidirectional search is performed. Let us assume that we have completed our first bounded search on resource in the backward direction. During our complementary search in the forward direction on resource, since the resource-optimum paths from every valid node to target are known (via the backward search), we can join resource-optimum paths to nodes in the forward direction with their counterpart in the backward direction to make feasible complete source-target paths. Then, if the cost of the resulting path is better than our initial C_0^*, we can simply update our initial solution with our new feasible path. Given the idea of coupling optimum partial paths, we can extend this strategy to our bounded searches in the initialisation phase. During a node's expansion in each bounded search, we try to find better initial solutions by creating complete paths using the node's information in the reverse direction. This means we can even join a cost-optimum partial path with a resource-optimum partial path in the reverse direction as shown in Figure 1. If the resulting path is feasible and its overall cost is less than the cost of our best known solution in C_0^*, we can update our cost upper bound accordingly.

Setting up the main search: Our final contribution to the RCSPP initialisation phase is related to its ability to set up the main search if the bidirectional approach is used. The last bounded search of our initialisation procedure produces information on the closeness of valid nodes to source and target, that is for every node $v \in V''$, the minimum costs to both source and target are known. Using this important observation, we will estimate the distribution of nodes around source and target, which allows us to *bias* our bidirectional search for the RCSPP. As an example, Figure 3 shows a case where there are more nodes around the target than the source. In this situation, we will bias our bidirectional search by setting a resource budget share in each direction. This means, instead of allocating an equal budget to searches in both directions (half of the budget, or $R/2$, for each direction), we shift the default budget border and allocate more budget to the search in the forward direction. We measure the closeness of nodes to source or target by comparing their minimum costs to the two endpoints, i.e., $\underline{g}^b$ and $\underline{g}^f$. For example, if nodes are generally closer to the target node, they have a smaller overall cost to the target than to the source. We therefore define our *budget factor* for direction d as

$$\beta^d = \min\left(1, 0.5 \times \sum_{v \in V''} \underline{g}^d(v) / \sum_{v \in V''} \underline{g}^{d'}(v)\right) \qquad (1)$$

where d is the direction for which we want to increase the budget and we have $\sum_{v \in V''} \underline{g}^d(v) > \sum_{v \in V''} \underline{g}^{d'}(v)$. To ensure the full budget remains accessible, we allocate the rest of the budget to the reverse direction d' via $\beta^{d'} = 1 - \beta^d$. The upper bound of our budget factors is limited to one, i.e., $0 \leq \beta^{f,d} \leq 1$. In the extreme case ($\beta^d = 1$), this setting may turn the bidirectional search into a unidirectional search. Therefore, our proposed approach provides the RCSPP search with a greater degree of flexibility by biasing the

Algorithm 1 Initialisation Phase of RC-EBBA*

Procedure Initialise (*source, target, budget*)

1: Backward bounded A* on *resource*, find the initial C_0^*
2: Forward bounded A* on *resource*, update C_0^* if possible
3: Set V'= {expanded nodes in the bounded search of step 2}
4: Backward bounded A* on *cost* and V', update C_0^* if possible
5: Forward bounded A* on *cost* and V', update C_0^* if possible
6: Set V''= {expanded nodes in the bounded search of step 5}
7: Calculate budget factors β^f and β^b using nodes in V''
8: **return** $(C_0^*, \beta^f, \beta^b)$

bidirectional search based on the distribution of nodes.

In summary, we present the main steps of our new initialisation approach for RC-EBBA* in Algorithm 1.

Our Enhanced Biased Bidirectional A*

This section describes our contributions to the main search of the RCSPP using an A*-based bidirectional search scheme as described by Thomas, Calogiuri, and Hewitt (2019). The pseudocode of our RC-EBBA* search is given in Algorithm 2. The algorithm starts by calling the initialisation function and initialising the priority queues in both directions ($Q^{f,b}$). It then uses the minimum cost path between source and target (here as a heuristic) to build the first set of partial paths at both ends and insert them into the corresponding priority queue for backward (Q^b) or forward (Q^f) partial paths. We store the main information of each partial path in a unique *label*. This information mainly includes the lower bound on source-target cost (denoted by f), cost (g), the resource consumption (r), and the last node (u) of the partial path. The priority queues process the labels based on their f-value. The algorithm then tries to expand the most promising partial path by choosing a direction which offers the smaller f-value among the top labels of Q^f and Q^b (denoted by d). After expanding every partial path, newly generated partial paths are inserted into the corresponding priority queue if they meet the feasibility criteria. Finally, the search successfully terminates when a partial path with a lower bound greater than the best found solution C^* is extracted from any of the priority queues. Now we explain our contributions to the bidirectional A* search for the RCSPP by highlighting the new features of RC-EBBA*.

Biased bidirectional search: In the RCSPP, as the resource consumption of the cost-optimum path is limited, the bidirectional search can place a limit on the maximum resource usage in each direction. This idea was used by Righini and Salani (2006), and Thomas, Calogiuri, and Hewitt (2019) adapted this constraint to their bidirectional search by allocating an equal resource budget ($R/2$) to the searches in both directions. We now extend this strategy and claim that the resource usage of directions can be limited by any fraction of the budget R.

Lemma 2. *The resource budget in each direction of bidirectional A* for the RCSPP can be any fraction of the budget as long as they together cover the whole range of R.*

Proof Sketch: The bidirectional search matches the partial paths in both directions in search of the solution path. Therefore, we just need to show that the solution path is still discoverable using any fraction. Let us assume that we are on one segment of the solution path with the resource usage R in direction d and the search decides to not further expand that partial path at the fraction β^d. Since there is no limit on the cost of partial paths in each direction (the solution has not been found yet), the search in the opposite direction d' is allowed to expand partial paths up to the budget fraction of $1 - \beta^d$, which also includes the second segment of the solution path if exists. Therefore, both partial paths of the solution can meet at the fraction β. □

Algorithm 2 Enhanced Biased Bidirectional A* for RCSPP

Procedure RC-EBBA* (*source, target, budget*)

1: $s \leftarrow$ *source*, $t \leftarrow$ *target*, $R \leftarrow$ *budget*
2: $(C^*, \beta^f, \beta^b) \leftarrow$ Initialise (s, t, R)
3: $Q^{f,b} \leftarrow \emptyset$, $r_{min}^{f,b}(v) \leftarrow \infty$ for each $v \in V''$
4: $label_s \leftarrow \{f : \underline{g}^b(s), g : 0, r : 0, u : s\}$, $Q^f.push(label_s)$
5: $label_t \leftarrow \{f : \underline{g}^f(t), g : 0, r : 0, u : t\}$, $Q^b.push(label_t)$
6: **while** $Q^f \cup Q^b \neq \emptyset$ **do**
7: $d \leftarrow$ direction of the *min-f label* from $Q^f \cup Q^b$
8: $\{f, g, r, u\} \leftarrow label \leftarrow Q^d.pop()$, $d' \leftarrow reverse\ (d)$
9: **if** $f \geq C^*$ **then**
10: **break**
11: **else if** $r \geq r_{min}^d(u)$ **then**
12: **continue**
13: **else**
14: $r_{min}^d(u) \leftarrow r$
15: **end if**
16: EarlyC*Update($label, d'$)
17: **if** $r \leq \beta^d R$ **then**
18: **for** *all* $v \in succ^d(u)$ **do**
19: $g' \leftarrow g + cost(u, v)$
20: $r' \leftarrow r + resource(u, v)$
21: $label' \leftarrow \{g' + \underline{g}^{d'}(v), g', r', v\}$
22: **if** Feasible($label', d$) **then**
23: $Q^d.push(label')$
24: **end if**
25: **end for**
26: **end if**
27: **if** $\underline{r}^{d'}(u) > \beta^{d'} R$ **then**
28: **continue**
29: **else**
30: **for** all $label' \in expanded^{d'}(u)$ **do**
31: $\{f', g', r', v'\} \leftarrow label'$
32: **if** $g + g' \geq C^*$ **then**
33: **break**
34: **else if** $r + r' \leq R$ **then**
35: $C^* \leftarrow g + g'$, **break**
36: **end if**
37: **end for**
38: $expanded^d(u).pushback(label)$
39: **end if**
40: **end while**
41: **return** C^*

We use the budget from the initialisation phase ($\beta^{f,d}$) to bias our search as shown at line 17 of Algorithm 2.

Efficient dominance checking: The common approach for dominance checking is to compare new partial paths against all of the previous partial paths during the expansion (Thomas, Calogiuri, and Hewitt 2019; Cabrera et al.

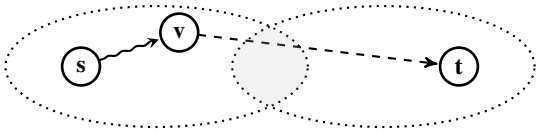

Figure 4: A partial path outside of the coupling area.

2020). In this paper, we extend the fast dominance checking approach of Ulloa et al. (2020) to our RC-BBDA*. In A* search, since the f-value of partial paths is monotonically increasing, partial paths can be checked for dominance *before* expansion. Let us assume two partial paths of a node with $f_2(v) \geq f_1(v)$ are sequentially extracted from the priority queue (so $f_1(v)$ is extracted first). Since both partial paths use the same heuristic to establish their f-values, we have $g_2(v) \geq g_1(v)$. Therefore, the second partial path will be dominated by the first partial path if $r_2(v) \geq r_1(v)$. In other words, to check a partial path for dominance, we just need to compare its resource usage against the resource consumption of the node's most recently expanded partial path. This means, every time we expand a non-dominated partial path in every direction, we keep track of the (always decreasing) changes in the node's minimum resource consumption denoted by $r^d_{min}(v)$ as shown at line 14 of Algorithm 2.

Efficient partial paths coupling: An essential step in the bidirectional search is to check whether the expanded partial path can be matched with any of the partial paths in the opposite direction to create a feasible solution. Now we present two techniques to improve the efficiency of this vital step.

The first technique is equipping the partial paths coupling strategy with the early termination criteria as shown at lines 32, 34 of Algorithm 2. We use our argument for the efficient dominance checking and claim that the f-value (and correspondingly cost g) of the expanded partial paths is guaranteed to be increasing. That is, if we store expanded partial paths associated with node v in order, the first/last added label has the minimum/maximum cost among the partial paths in the $expanded(v)$ set in direction d. Therefore, the path coupling procedure is allowed to terminate early as soon as it finds a feasible solution path (successful matching) or when it creates a path which is not as good as the best known solution path (unsuccessful matching) since as it goes further in the labels of the $expanded(v)$ set, the situation gets worse and the cost of the reverse partial paths increases.

Our second technique helps us to save time and space by not trying to store/match partial paths for which we are sure that there will not be a counterpart in the reverse direction (line 27 of Algorithm 2). Figure 4 depicts a sample scenario with this situation where the partial path from s to v needs to have a complementary path (from v to t) with a minimum resource usage greater than its allowed budget in that direction and the node is located outside of the expected matching area (highlighted). Therefore, there is no need to even have an $expanded$ set for node v, and the path coupling procedure for the incoming labels with this property can be ignored. We can also remove such labels after the expansion since the search will no longer need their information except for backtracking to construct the solution path.

Algorithm 3 Auxiliary Functions for RC-EBBA*

Procedure Feasible $(label', d)$
1: $\{f', g', r', v\} \leftarrow label'$
2: **if** $v \notin V''$ **then**
3: **return False**
4: **else if** $g' > g^d_{\underline{r}}(v)$ **or** $r' > r^d_{\underline{g}}(v)$ **then**
5: **return False**
6: **else if** $f' \geq C^*$ **or** $r' + \underline{r}^{d'}(v) > R$ **then**
7: **return False**
8: **else if** $r \geq r^d_{min}(v)$ **then**
9: **return False**
10: **else**
11: **return True**
12: **end if**

Procedure EarlyC*Update$(label, d')$
1: $\{f, g, r, u\} \leftarrow label$
2: **if** $f < C^*$ **and** $r + r^{d'}_{\underline{g}}(u) \leq R$ **then**
3: $C^* \leftarrow f$
4: **else if** $g + g^{d'}_{\underline{r}} < C^*$ **and** $r + \underline{r}^{d'} \leq R$ **then**
5: $C^* \leftarrow g + g^{d'}_{\underline{r}}$
6: **end if**
7: **return**

Stronger feasibility criteria: Our enhanced bidirectional search is equipped with new pruning strategies as presented in Algorithm 3. First, labels that are pointing to nodes outside of the valid domain V'' will be pruned. And second, we prune labels carrying a cost/resource usage above the node's cost/resource upper bound. This is because there always exists a dominant path with a lower cost and resource usage for which we know $\underline{g}^d \leq g^d$ or $\underline{r}^d \leq r^d$. Thirdly, if the resource usage of the partial path is greater than the resource usage of a previously expanded partial path, the label can be pruned. The idea of this strategy is related to our efficient dominance checking, but now we never process a label that is guaranteed to be dominated by violating $r^d_{min}(v)$.

Early solution update: As an additional heuristic, we use our improved lower bounds from the initialisation phase to shrink the gap between the minimum cost and the optimal solution by searching for a solution path before reaching the path's coupling area. This techniques matches every partial path in direction d with the minimum cost/resource partial path in the reverse direction as described in Algorithm 3. If the resulting path is feasible and improves the solution cost, the search will then update C^* with the cost of the new solution path. This procedure helps the algorithm to prune more infeasible paths by continually updating C^* even before coupling partial paths from the $expanded$ set.

Faster queue operations: Our final contribution to the RC-EBBA* algorithm is to use an efficient priority queue. RCSPP algorithms typically use a heap data structure to implement the required priority queues. Since the size of the queue in the RCSPP instances can become very large (sometimes in the order of V including dominated labels), updates on priority queues may be inevitably costly. Based on this observation, we propose to use a fixed-size bucket priority queue instead, which provides a fast push operation (without any tie breaking). We can use a fixed number of buckets

since the lower and upper bounds of the solution path are known prior to the main search ($\underline{g}(t)$ and C^*). As the search deals with a large number of labels, which all fall within a limited f-range, we expect the search to see almost all of the buckets filled and not waste too much of its time going through empty buckets. Therefore, the overall cost of scanning each bucket queue for pop operations in the worst case will be $O(C^* - \underline{g}(t))$ which we believe will outperform the binary heap with $O(log(V))$ in practice. We will investigate the performance of both structures in the experiment section.

Experimental Results and Analysis

We compare our RC-EBBA* with existing algorithms that have shown a good performance on large networks. The selected algorithms are K-SP (Sedeño-Noda and Alonso-Rodríguez 2015), Pulse (Lozano and Medaglia 2013), RC-BDA* (Thomas, Calogiuri, and Hewitt 2019) and the recent Bi-Pulse search (Cabrera et al. 2020).

Benchmark instances: Following the recent studies, we use the benchmark instances of Sedeño-Noda and Alonso-Rodríguez (2015) which have been designed based on a particular order of nodes over USA road networks in the 9th DIMACS challenge (DIMACS 2005) as shown in Table 1 in detail. The constrained problem in these instances is to find the shortest feasible path for 440 pairs of nodes, subject to varying time limits as a *resource* constraint.

Resource budget: Following the literature, we define the resource budget R based on the tightness of the constraint $p = (R - \underline{r}(t))/(r_{\underline{g}}(t) - \underline{r}(t))$ and $p \in \{0.1, 0.2, ..., 0.8\}$.

Implementation: We used the C implementation of the K-SP and Pulse algorithms kindly provided to us by Sedeño-Noda and Alonso-Rodríguez (2015). For the RC-BDA* and Bi-Pulse algorithms, we were unable to obtain the original implementations. We therefore implemented both algorithms based on their descriptions. For RC-BDA*, since its structure is similar to our bidirectional A* search, we simply disabled the new features of our RC-EBBA* search to obtain the basic RC-BDA* search. We also fixed a potential inefficiency in RC-BDA* which is related to its weak label matching strategy: Instead of inserting all of the expanded partial paths of a direction in a large pool and then searching for the best match inside the pool (extra effort for finding the same ending node), we allocated a set (in a form of linked list) for each node that keeps the partial path corresponding to that node, which makes the partial path coupling much more efficient than the original approach in the algorithm. For the Bi-Pulse algorithm, our single thread implementation with the proposed setting in the paper showed a very weak performance and ran out of memory in several cases. Therefore, we decided to report the timings from the original paper. For that paper, the authors implemented Bi-Pulse based on a parallel framework in Java and reported the runtimes using a machine with an Intel Core i7-4610M at 3.00 GHz and with 8GB of RAM and a four-hour timeout.

We implemented RC-BDA* and RC-EBBA* algorithms in C++ and compiled all of the algorithms (including K-SP and Pulse) using the GCC7.4 compiler with O3 optimisation settings. Our codes are publicly available [1]. We ran all of our experiments once on a single core of an Intel Xeon E5-2660V3 processor running at 2.6 GHz and with 16GB of RAM, under the SUSE Linux 12.4 environment and a four-hour timeout. In contrast to Pulse, Bi-Pulse and KS-P, our implementations of RC-BDA* and RC-EBBA* do not take any advantage of low-cost successors, i.e., we do not set any order on expanding nodes' successors.

Initialisation phase analysis: We assume that the budget is not known and should be calculated by the constraint p. Therefore, we first calculate the resource budget R via two backward A* searches. We then initialise our proposed bounded searches based on the tightness of the constraint p. That is, if $p < 0.5$, we next find resource heuristics and then reestablish our cost heuristics, otherwise, we rely on our initial (less informed) cost heuristics obtained during the budget search and resume the initialisation steps as explained in Algorithm 1. Furthermore, we use Euclidean distance as a consistent heuristic for our A* searches on DIMACS instances. Table 2 illustrates the performance of our initialisation approach against the existing approaches. First, we compare the average runtime of each strategy on the benchmark instances. For the unidirectional approaches (K-SP and Pulse), the initialisation phase requires to run one round of the Dijkstra's algorithm (one-to-all search from target, column Uni-Dij.). Bidirectional approaches (RC-BDA* and Bi-Pulse) though perform two one-to-all searches, from source and target (column Bi-Dij.). In cases where the shortest path is feasible (i.e., where $\underline{g}^b = g^b_{\underline{r}}$), they may terminate early without running the second set. The average runtime of our proposed initialisation approach is given in the third column of Table 2. Although our approach may perform a full bidirectional search in the worst case, our bounded A* search strategy is even faster than the uni-directional approach in practice. Comparing the average of total instances, we can see our approach completes the initialisation phase three times faster than the unidirectional scheme and six times faster than the bidirectional scheme. The importance of this phase is revealed when we look into the runtime of individual cases. For example, initialising easy cases of the full USA road network using the normal unidirectional one-to-all search requires at least 15 seconds on our machine, but our approach just needs 0.001 seconds to initialise those instances, a speed up of four orders of magnitude.

Table 2 also compares the quality of initial solutions using different approaches. Since both unidirectional and bidirectional strategies find the same initial solution, we just report the maximum distance of the initial solution C^*_0 from the optimal solution C^* for the unidirectional scheme, in the form of $\varphi(C^*_0) = (C^*_0 - C^*)/C^*$. The results in Table 2 show that the initial solutions using existing approaches can be as far as 38.69% off the optimal solution, but as seen in the last column of Table 2, our proposed strategy to update the initial solution significantly reduces that gap to at most 4.64% and even closes the gap in several cases. A good initial solution will directly help the RCSPP search, since it will be able to prune more suboptimal paths.

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

	Road Network Details		RC-EBBA*		K-SP		RC-BDA*		Pulse		Bi-Pulse	
Inst.	Nodes	Arcs	S.	Avg.(s)	S.	Avg.(s)	S.	Avg.(s)	S.	Avg.(s)	S.	Avg.(s)
NY	264,346	730,100	40	**0.066**	40	0.419	40	0.576	40	15.539	40	0.23
BAY	321,270	794,830	40	**0.078**	40	2.361	40	3.941	40	14.857	40	0.30
COL	435,666	1,042,400	40	**0.112**	40	7.732	40	15.066	36	1446.771	40	1.67
FLA	1,070,376	2,687,902	40	**0.423**	40	60.801	40	33.428	31	3743.906	40	2.64
NE	1,524,453	3,868,020	40	**0.116**	40	1.977	40	2.262	38	978.915	40	1.24
CAL	1,890,815	4,630,444	40	**11.490**	35	2430.943	32	3145.198	28	4330.042	35	1896.19
LKS	2,758,119	6,794,808	40	**6.239**	40	1354.912	36	1840.397	24	5761.238	36	1727.33
E	3,598,623	8,708,058	40	**0.691**	40	46.875	38	739.256	30	3652.112	40	13.63
W	6,262,104	15,119,284	40	**2.996**	40	858.041	31	3410.237	20	7479.830	38	852.61
CTR	14,081,816	33,866,826	40	**5.507**	40	1155.848	39	981.446	18	8051.855	-	-
USA	23,947,347	57,708,624	40	**89.993**	28	5009.795	24	6293.759	16	8645.876	-	-
		Overall	440	**10.701**	423	993.609	400	1496.869	321	4010.995	-	-

Table 1: Network details, number of solved cases and average runtimes (Bi-Pulse runtimes are only for solved cases)

	Avg. initialisation time (s)			Max. $\varphi(C_0^*)(\%)$	
Inst.	Uni-Dij.	Bi-Dij.	Ours	Uni-Dij.	Ours
NY	0.129	0.268	**0.057**	38.69	**4.64**
BAY	0.151	0.309	**0.074**	12.45	**3.40**
COL	0.202	0.374	**0.085**	9.16	**1.99**
FLA	0.521	1.267	**0.324**	22.52	**1.27**
NE	0.859	1.909	**0.106**	30.83	**3.19**
CAL	0.969	1.712	**0.839**	10.49	**3.43**
LKS	1.432	2.909	**0.679**	25.99	**4.41**
E	2.060	4.731	**0.480**	24.76	**2.93**
W	3.668	8.712	**1.172**	25.88	**3.42**
CTR	11.126	20.825	**1.571**	11.53	**2.85**
USA	14.686	38.582	**6.175**	10.93	**3.98**

Table 2: Initialisation time and initial solution quality

Algorithmic performance: Table 1 presents the experimental results for all of the algorithms in this study. This table shows the number of instances solved to optimality (denoted by S.) and average runtime in *seconds* including the initialisation time and the time needed to calculate the resource budget R from the constraint p. For unsolved instances (except for Bi-Pulse), we generously assume a runtime of four hours (the time limit). The detailed results can be found in our source-code repository. Among the competitors, K-SP shows the best performance and solves more cases compared to RC-BDA* and Pulse, but still fails to solve 17 instances within the time limit. In contrast, our RC-EBBA* algorithm needs less than 11 seconds on average to solve all of the instances to optimality. Our enhanced algorithm outperforms its well-designed competitors in almost all of the test cases, showing a minimum of one (BAY map) and a maximum of four orders of magnitude (CTR map) faster run-times on average. Compared to the Bi-Pulse runtimes, although they have been reported based on a parallel implementation using a machine with a notionally faster processor, our single thread RC-EBBA* is still superior to Bi-Pulse in term of the number of solved cases and the average runtimes (note that we can only report the published runtimes for Bi-Pulse, which do not count unsolved instances).

Near-optimal solutions: Our RC-EBBA* algorithm shows fast performance on easy-to-medium instances and is able to find the optimal solutions in less than a second on average. Among all of the large instances, RC-EBBA* solves the most difficult problem (in the USA instance) in less than 9 minutes. As RC-EBBA* is equipped with an efficient solution update strategy, if near-optimal solutions are also accepted, the algorithm can return a feasible solution within one per cent of the optimal solution of the most difficult problem in the benchmark instances after 4.5 minutes, two times faster than its 9-minute search for the optimal solution.

Bucket vs Heap: We also compared the effect of using bucket queues versus heaps for implementing RC-EBBA* on the same benchmark set. Using heaps, RC-EBBA* is still able to solve all of the instances to optimality in 37 seconds on average (compared to 10.7 seconds for bucket queues). Detailed results show that the proposed bucket-based RC-EBBA* shows a comparable performance (nearly equal) on easy instances but outperforms heap-based RC-EBBA* on all of the medium-to-difficult instances. Over all of the test cases, the RCSPP search is proceeded on average two times faster if the bucket structure is used. We also noticed bucket queues required less memory in practice, due to the fact that labels in buckets do not need to contain the f-values, as required by heap data structures. The overhead of the fixed number of buckets is therefore outweighed by the reduced size in memory of the labels.

Conclusion

This paper presents several improvements for the resource constrained shortest path problem (RCSPP) by introducing an enhanced biased bidirectional A* search RC-EBBA*. Our first set of improvements speed up the initialisation phase of RC-EBBA* and assist its search engine by providing better quality heuristics. The second set of improvements increase the search efficiency and algorithmic performance of the main search of RC-EBBA*. The results of our experiments on the largest set of instances in the literature show that, for the first time in the context of the RCSPP, the proposed RC-EBBA* algorithm can solve all of the 440 benchmark instances while outperforming the runtime of the state-of-the-art algorithms by one to four orders of magnitude.

References

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. doi:10.1002/net.21960. URL https://doi.org/10.1002/net.21960.

DIMACS. 2005. 9th DIMACS Implementation Challenge - Shortest Paths. URL http://users.diag.uniroma1.it/challenge9.

Handler, G. Y.; and Zang, I. 1980. A dual algorithm for the constrained shortest path problem. *Networks* 10(4): 293–309. doi:10.1002/net.3230100403. URL https://doi.org/10.1002/net.3230100403.

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. doi:10.1109/TSSC.1968.300136. URL https://doi.org/10.1109/TSSC.1968.300136.

Lozano, L.; and Medaglia, A. L. 2013. On an exact method for the constrained shortest path problem. *Comput. Oper. Res.* 40(1): 378–384. doi:10.1016/j.cor.2012.07.008. URL https://doi.org/10.1016/j.cor.2012.07.008.

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

Pugliese, L. D. P.; and Guerriero, F. 2013. A survey of resource constrained shortest path problems: Exact solution approaches. *Networks* 62(3): 183–200. doi:10.1002/net.21511. URL https://doi.org/10.1002/net.21511.

Righini, G.; and Salani, M. 2006. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discret. Optim.* 3(3): 255–273. doi:10.1016/j.disopt.2006.05.007. URL https://doi.org/10.1016/j.disopt.2006.05.007.

Righini, G.; and Salani, M. 2008. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks* 51(3): 155–170. doi:10.1002/net.20212. URL https://doi.org/10.1002/net.20212.

Santos, L.; Coutinho-Rodrigues, J.; and Current, J. R. 2007. An improved solution algorithm for the constrained shortest path problem. *Transportation Research Part B: Methodological* 41(7): 756 – 771. ISSN 0191-2615. doi:https://doi.org/10.1016/j.trb.2006.12.001. URL http://www.sciencedirect.com/science/article/pii/S0191261507000124.

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. doi:10.1016/j.amc.2015.05.109. URL https://doi.org/10.1016/j.amc.2015.05.109.

Thomas, B. W.; Calogiuri, T.; and Hewitt, M. 2019. An exact bidirectional A$\star$ approach for solving resource-constrained shortest path problems. *Networks* 73(2): 187–205. doi:10.1002/net.21856. URL https://doi.org/10.1002/net.21856.

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 Beck, J. C.; Buffet, O.; Hoffmann, J.; Karpas, E.; and Sohrabi, S., eds., *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020*, 143–151. AAAI Press. URL https://aaai.org/ojs/index.php/ICAPS/article/view/6655.

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. doi:10.1016/j.cor.2011.03.008. URL https://doi.org/10.1016/j.cor.2011.03.008.