

S*: A Heuristic Information-Based Approximation Framework for Multi-Goal Path Finding

Kenny Chour,¹ Sivakumar Rathinam,¹ Ramamoorthi Ravi²

¹ Mechanical Engineering, Texas A & M University, College Station

² Tepper School of Business, Carnegie Mellon University, Pittsburgh
ckennyc@tamu.edu, srathinam@tamu.edu, ravi@cmu.edu

Abstract

We combine ideas from uni-directional and bi-directional heuristic search, and approximation algorithms for the Traveling Salesman Problem, to develop a novel framework for a Multi-Goal Path Finding (MGPF) problem that provides a 2-approximation guarantee. MGPF aims to find a least-cost path from an origin to a destination such that each node in a given set of goals is visited at least once along the path. We present numerical results to illustrate the advantages of our framework over conventional alternates in terms of the number of expanded nodes and run time.

Introduction

Multi-Goal Path Finding (MGPF) aims to find a least-cost path in a graph $G = (V, E)$ with non-negative edge costs such that the path starts from an origin ($s \in V$) and ends at a destination ($d \in V$), and each node in a given set of goals ($\bar{T} \subseteq V$) is visited at least once along the path. In the special case when the goal set is empty ($\bar{T} = \emptyset$), MGPF reduces to the least-cost path problem and is polynomial time solvable (Dijkstra 1959; Lawler 2001). For the general case, we must also determine the sequence in which the goals must be visited, and therefore, MGPF is a generalization of the Steiner¹ Traveling Salesman Problem (Rodriguez-Pereira et al. 2019) and is NP-Hard. MGPF arises in numerous aerial robot and logistics applications as discussed in recent surveys (Otto et al. 2018; Macharet and Campos 2018).

Existing 2-approximation algorithms for the MGPF and its variants (Kou, Markowsky, and Berman 1981; Mehlhorn 1988) rely on three steps (Fig. 1): (i) find a suitable Steiner tree spanning the origin, goals and the destination, (ii) double the edges in the Steiner tree to obtain an Eulerian graph, and then finally (iii) find a path in the Eulerian graph that is feasible for the MGPF problem. The 2-approximation ratio and the computational complexity of these algorithms primarily relies on the Steiner tree construction in step (i); this construction must be done so that the cost of the Steiner tree constructed is at most the optimal cost of the MGPF. Well

Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

¹Any node that is *not* required to be visited is referred to as a *Steiner node*. A path may choose to visit a Steiner node if it helps in either finding feasible solutions or reducing the cost of travel.

known primal-dual (Ravi 1994; Agrawal, Klein, and Ravi 1995; Goemans and Williamson 1997) or minimum spanning tree based algorithms (Kou, Markowsky, and Berman 1981; Mehlhorn 1988) can be used to find such a Steiner tree.

The objective of this article is to propose a new approximation framework that fuses existing methods for Steiner tree construction with heuristic information to develop new algorithms for step (i) of the approximation algorithm for MGPF (Fig. 1). As a consequence, this framework provides new efficient 2-approximation algorithms for MGPF. Our work follows the spirit of the A* (Hart, Nilsson, and Raphael 1968) (or the bi-directional (Pohl 1969)) heuristic search methods where a best-first search procedure from the origin (or the origin and the destination) was combined with heuristic information to develop new algorithms for the least-cost path problem. In fact, in the special case when the goal set is empty, the primal-dual algorithm (Agrawal, Klein, and Ravi 1995) for the Steiner tree problem, depending on how it is applied, reduces to either the uni-directional search (Dijkstra 1959) or bi-directional search (Nicholson 1966) algorithm available for the least-cost path problem. Therefore, one can view our work in this article as a direct generalization of the A* and the bi-directional heuristic search procedures to Steiner tree computation and to MGPF.

We refer to the proposed framework as Steiner* (S*) and present its two variants. In the first variant, we use A* to grow closed and open sets from each node in $T := \{s, d\} \cup \bar{T}$ and simultaneously construct a Steiner tree when relevant bounding conditions are satisfied. We refer to this variant as S*-unmerged since we do not merge the closed sets corresponding to distinct nodes in T even when they overlap with each other. The second variant is referred to as S*-merged where the closed sets are merged when appropriate bounding conditions are satisfied akin to what happens in the Kruskal's minimum spanning tree algorithm (Kruskal 1956). The S*-merged framework is agnostic to the underlying optimality conditions used for the least-cost path computations during the search process; specifically, one can use the optimality conditions from A* (Hart, Nilsson, and Raphael 1968) or bi-directional search (Nicholson 1966) or Meet in the Middle (MM) (Holte et al. 2017) algorithms in the S*-merged framework and guarantee the required properties. We note here that while there are several bi-

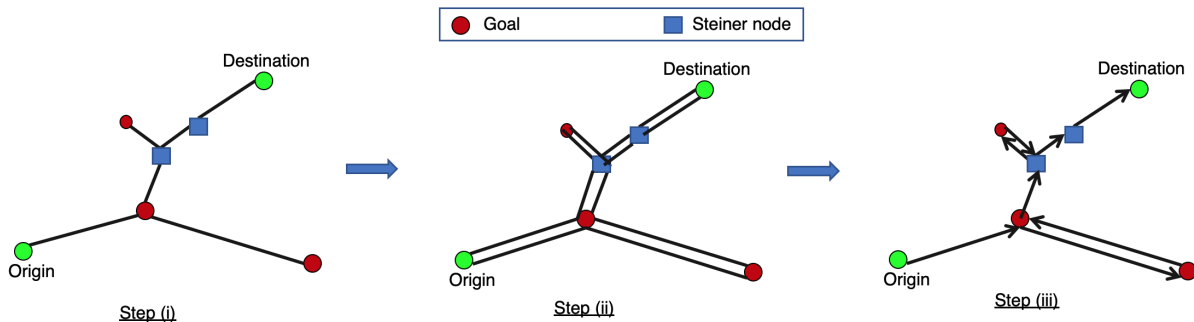


Figure 1: An approximation algorithm for MGPF: (i) Construct a suitable Steiner tree, (ii) Double the edges in the Steiner tree to form an Eulerian graph and (iii) Find a path in the Eulerian graph from the origin to destination that visits each goal at least once, while discarding remaining edges.

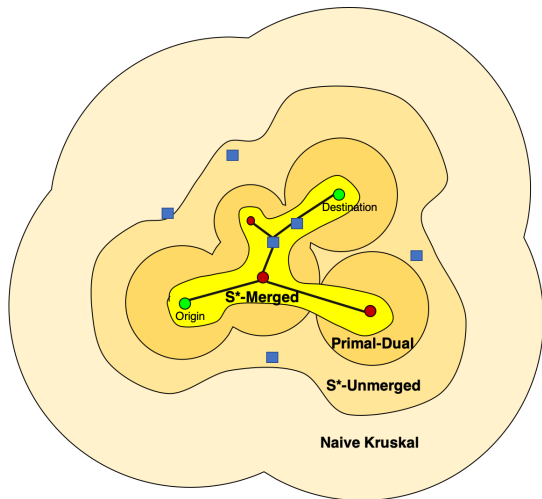


Figure 2: Each shaded region shows our expectation on the set of expanded nodes for conventional solvers and the proposed framework. Here, naive Kruskal is the popular approach (Kou, Markowsky, and Berman 1981) described in the Background and Preliminaries section.

directional heuristic search methods (De Champeaux 1983; Kwa 1989; Eckerle 1994; Kaindl and Kainz 1997; Barker and Korf 2015; Holte et al. 2017; Chen et al. 2017), we use MM (Holte et al. 2017) as a representative bi-directional search method for least-cost path computations in our framework as our goal is to address the MGPF; other methods will be considered in the future. Like uni-directional and bi-directional heuristic search, the expectation here is that combining existing algorithms with heuristic information will reduce the number of expanded nodes and possibly the computation time (Fig. 2).²

After describing the new algorithms with theoretical performance guarantees for the Steiner tree construction, we provide extensive computational results on the performance on the proposed framework for instances derived from the

²Counterexamples to this expectation is presented in the appendix of (Chour, Rathinam, and Ravi 2021).

Multi-Agent Path Finding (MAPF) library³ (Stern et al. 2019). While these numerical results clearly illustrate the benefits of the proposed framework in several scenarios, we do not claim that the proposed framework is superior to conventional solvers for each and every instance of the MGPF problem. Nevertheless, the proposed framework is the first of its kind for MGPF and provides a new line of research for related problems.

Background and Preliminaries

Let $c(u, v) \geq 0$ denote the cost of the edge joining two distinct vertices u and v in $G = (V, E)$. The cost of a path is defined as the sum of the edges in the path. Let $cost^*(u, t)$ denote the cost of the least-cost path from u to t in G . Let $\bar{h}_t(u)$ be an underestimate on $cost^*(u, t)$. To simplify our presentation and proofs, we assume $\bar{h}_t(u)$ is obtained using a consistent heuristic (Hart, Nilsson, and Raphael 1968).

A Steiner tree is a connected subgraph of edges that spans a subset of relevant nodes $T = \{s, d\} \cup \bar{T} \subseteq V$ also commonly referred to as terminals. Finding a Steiner tree which minimizes the sum of the cost of the edges in the tree is NP-Hard (Karp 1972). Therefore, a popular approach for finding a suitable Steiner tree in step (i) of the approximation algorithm (Fig. 1) is to find a Minimum Spanning Tree (MST) in the metric completion⁴ of the terminals, and then replace each edge in the MST by the corresponding least-cost path in G . There are several implementations of this approach (Kou, Markowsky, and Berman 1981; Mehlhorn 1988; Ravi 1994; Goemans and Williamson 1997). Irrespective of the specific implementation used, the Steiner tree construction relies on satisfying the following key properties:

- (SP) Ensure that when a path between a pair of terminals in T is confirmed by a path finding algorithm, it is indeed a least-cost path in G between them.
- (K) When a path P between two terminals is included in the Steiner tree, it obeys Kruskal’s condition (Kruskal

³<https://movingai.com/benchmarks>

⁴The metric completion of the terminals is a complete weighted graph on the terminals T where the weight of an edge between a pair of terminals is the minimum cost of a path between them in G .

Algorithm 1: S^* -unmerged

```
1 Inputs:
2  $G = (V, E), c(u, v) \forall u, v \in V, T \subseteq V$ 
3  $\bar{h}_t(u) \forall t \in T, u \in V$  // consistent lower
   bounds on  $c^*(u, t)$ 
4 Output:
5  $S_T$  // Steiner tree spanning  $T$ 
6 Initialization:
7  $C_t := \{t\} \forall t \in T$  // Closed sets
8  $O_t := \{u : (u, t) \in E\} \forall t \in T$  // Open sets
9  $D_t := T \setminus \{t\} \forall t \in T$  // Destination sets
10  $g_t(u) := c(t, u) \forall u \in V, t \in T$ 
11  $f_t(u) := g_t(u) + h(u, D_t), \forall u \in V, t \in T$ 
12  $Q = \emptyset$  // Paths eligible for  $S_T$ 
13  $S_T := \emptyset$ 
14 Main Loop:
15 while all the terminals are not connected in  $S_T$  do
16    $u_t := \arg \min_{u \in O_t} f_t(u) \forall t \in T$  //  $t$ 
   nominates best node
17    $t^* = \arg \min\{f_t(u_t) : t \in T\}$  // Choose
   nominator with least  $f$  cost
18    $C_{t^*} := C_{t^*} \cup \{u_{t^*}\}; O_{t^*} := O_{t^*} \setminus \{u_{t^*}\}$ 
19   for  $v \in \{v : (v, u_{t^*}) \in E, v \notin C_{t^*}\}$  do
20      $O_{t^*} := O_{t^*} \cup \{v\}$ 
21      $g_{t^*}(v) := \min(g_{t^*}(v), g_{t^*}(u_{t^*}) + c(u_{t^*}, v))$ 
22      $f_{t^*}(v) := g_{t^*}(v) + h(v, D_{t^*}), \forall v \in O_{t^*}$ 
23   end
24   if  $u_{t^*} \in D_{t^*}$  then
25      $Q := Q \cup \{PATH(t^*, u_{t^*})\}$ 
26      $D_{t^*} := D_{t^*} \setminus \{u_{t^*}\}$ 
27      $f_{t^*}(v) := g_{t^*}(v) + h(v, D_{t^*}), \forall v \in O_{t^*}$ 
28      $D_{u_{t^*}} := D_{u_{t^*}} \setminus \{t^*\}$ 
29      $f_{u_{t^*}}(v) := g_{u_{t^*}}(v) + h(v, D_{u_{t^*}}), \forall v \in O_{u_{t^*}}$ 
30   end
31    $S_T := UpdateSteinerTree(S_T, Q, f)$ 
32 end
33 return  $S_T$ 
```

1956) for inclusion in the MST of the metric completion of T . This requires that all paths between any pair of terminals with costs lower than the cost of P have been considered, and P does not create any cycles when added to the current tree.

If a Steiner tree algorithm satisfies the above key properties, then it is known that the cost of the Steiner tree obtained using the algorithm is at most equal to the optimal MGPF cost which leads to a 2-approximation algorithm for MGPF (Kou, Markowsky, and Berman 1981; Mehlhorn 1988). The variants of the S^* framework ensures that these two key properties are maintained thus proving the approximation guarantee of the final MGPF solution obtained using this approach.

S^* -unmerged

S^* -unmerged (Algorithm 1) uses A^* to build closed and

Algorithm 2: $UpdateSteinerTree(S_T, Q, f)$

```
1  $f^* := \min_t \min_{u \in O_t} f_t(u)$ 
2  $Q' = Q$  // Process paths locally
3 while  $Q'$  is nonempty do
4   Choose a path  $p \in Q'$  with the cheapest cost
   joining components  $C_1, C_2 \in S_T$ 
5   if adding  $p$  to  $S_T$  does not form a cycle AND
    $cost(p) \leq f^*$  then
6     Add  $p$  to  $S_T$ 
7      $\bar{C} := C_1 \cup C_2$ 
8     for  $t \in \bar{C} \cap T$  do
9        $D_t := T \setminus \bar{C}$ 
10       $f_t(v) := g_t(v) + h(v, D_t), \forall v \in O_t$ 
11    end
12  end
13  Delete  $p$  from  $Q'$ 
14 end
15 return  $S_T$ 
```

open sets from each terminal in T . We borrow the usual definitions of f, g and h costs from A^* ; however, in this framework, each terminal maintains its own version of the f, g and h costs for each of the nodes in its closed and open sets. Each terminal $t \in T$ maintains a destination list D_t which includes all the terminals not yet connected to t in the Steiner tree. In Algorithm 1, for any $S \subset V, h(u, S)$ is defined as the underestimate from node u to reach any terminal in S , i.e., $h(u, S) := \min\{h_t(u) : t \in T \cap S\}$.

After the initialization step, during each iteration of the the main loop, S^* -unmerged proceeds to let each terminal nominate a node with the least f -cost from its open set. The best nominated node u_{t^*} with the smallest f value is then moved to the corresponding terminal(t^*)'s closed set. Next, u_{t^*} is expanded and the corresponding g and f costs of its neighbors are updated (lines 19-23 in Algorithm 1). If paths are confirmed between two distinct terminals, they are also added to Q (line 25 in Algorithm 1) which maintains a list of paths eligible for the Steiner tree construction. Lines 26-29 in Algorithm 1 are referred as *re-prioritization* steps and are not mandatory; however, they may help in reducing the total number of expanded nodes during the search process at the expense of additional computation time. Finally, S^* -unmerged checks (line 31 in Algorithm 1) if the Steiner tree (S_T) needs to be updated based on the changes in Q or the bounding function f .

The procedure in Algorithm 2 ensures paths are added to S_T only if they satisfy the property (K). First, we consider the minimum f -value (f^*) among all the nodes in the open sets of all the terminals. Note that this will be f -cost of the next best nominated node in the algorithm. Therefore, all confirmed paths with costs at most equal to f^* from all terminals have been explored by now. We then process the confirmed paths in Q locally in *increasing order of cost* in the following way:

- If a path is between two terminals that are not connected in S_T and its cost is at most f^* (line 5 of Algorithm 2),

we include it in S_T . We also update the destination sets of the terminals due to changes in S_T and correspondingly change the f -costs for nodes in the open sets of these terminals (lines 8-11 of Algorithm 2).

- If a path doesn't satisfy the conditions in line 5 of Algorithm 2, we ignore and delete it locally since it does not obey property (K).

Correctness of S^* -unmerged

First, we observe that the open and closed sets of terminals are updated in the same manner as the A^* algorithm. Furthermore, as the heuristic costs are consistent, $g_t(u)$ for any node $u \in C_t$ is equal to $cost^*(t, u)$. Moreover, the shortest paths found between t and any node in C_t , and their corresponding g costs remain valid even after the changes in D_t since we recompute the lower bounds h to remain consistent, and use them in updating the bounds on the open sets. This shows that the paths finalized by the algorithm obey the (SP) property. As a result of the condition used in Algorithm 2, S^* -unmerged also satisfies the (K) property. Hence, we have the following theorem.

Theorem 1. *S^* -unmerged finds a Steiner tree of cost at most equal to the optimal MGPF cost. This Steiner tree can then be used to obtain a 2-approximation algorithm for MGPF.*

S^* -merged

Rather than carry out search from single terminals, S^* -merged (Algorithm 3) keeps track of the connectivity structure among the terminals in S_T using a set \mathbb{C} of components which are initialized to singleton terminals. When two components merge, we simply merge the set of terminals in these components in \mathbb{C} . We also carefully extend the definition of open and closed sets to subsets of terminals and ensure we update them so that they obey the conditions that the g -values of nodes in the closed sets give optimal paths from the node to some terminal in the component, and that the f -values remain lower bounds on reaching a terminal in another component. For this we will also need to ensure that the destination set of the merged components are updated appropriately.

Unlike S^* -unmerged where each terminal nominates its best node, in S^* -merged, each component nominates a node with the least f -cost from its open set. The best nominated node u_{A^*} with the least f value is then moved to the corresponding components (A^*) closed set. Next, u_{A^*} is expanded and the corresponding g and f costs of its neighbors are updated (lines 20-24 in Algorithm 3). If paths are confirmed between terminals in two distinct components, they are also added to Q (line 27 in Algorithm 3). Similar to S^* -unmerged, the re-prioritization steps in lines 28-31 of Algorithm 3 are not mandatory and can be used to speed up the implementation as needed. Finally, S^* -merged checks (line 35 in Algorithm 3) if the Steiner tree (S_T) and the component structure needs to be updated based on changes in Q or the bounding functions of these components.

We derive three versions of S^* -merged based on the method used to confirm the least-cost paths between components (line 26 in Algorithm 3). These methods are drawn

Algorithm 3: S^* -merged

```

1 Input:
2  $G = (V, E), c(u, v) \forall u, v, \in V, T \subseteq V$ 
3  $\bar{h}_t(u) \forall t \in T, u \in V$  // consistent lower
   bounds on  $c^*(u, t)$ 
4 Output:
5  $S_T$  // Steiner tree spanning  $T$ 
6 Initialization:
7  $\mathbb{C} := \{\{t\}, \forall t \in T\}$ 
8  $C_A := \mathcal{A}, \forall A \in \mathbb{C}$  // Closed sets of  $\mathcal{A}$ 
9  $O_A := \{u : (u, t) \in E, t \in \mathcal{A}\}, \forall A \in \mathbb{C}$  // Open
   sets of  $\mathcal{A}$ 
10  $D_A := T \setminus \mathcal{A}, \forall A \in \mathbb{C}$  // Destination sets
   of  $\mathcal{A} \in \mathbb{C}$ 
11  $g_A(u) := \min\{c(t, u) : t \in \mathcal{A}, t \in T\}, \forall u \in
   V, \mathcal{A} \in \mathbb{C}$ 
12  $f_A(u) := g_A(u) + h(u, D_A), \forall u \in V, \mathcal{A} \in \mathbb{C}$ 
   // Lower bound on  $cost^*(u, D_A)$ 
13  $Q = \emptyset$  // Paths eligible for  $S_T$ 
14  $S_T = \emptyset$ 

15 Main Loop:
16 while all the terminals are not connected in  $S_T$  do
17    $u_{\mathcal{A}} = \arg \min_{u \in O_{\mathcal{A}}} f_{\mathcal{A}}(u) \forall \mathcal{A} \in \mathbb{C}$  //  $\mathcal{A}$ 
   nominates best node
18    $\mathcal{A}^* = \arg \min\{f_{\mathcal{A}}(u_{\mathcal{A}}) : \mathcal{A} \in \mathbb{C}\}$  // Choose
   nominator with least  $f$  cost
19    $C_{\mathcal{A}^*} = C_{\mathcal{A}^*} \cup \{u_{\mathcal{A}^*}\}; O_{\mathcal{A}^*} = O_{\mathcal{A}^*} \setminus \{u_{\mathcal{A}^*}\}$ 
20   for  $v \in \{v : (v, u_{\mathcal{A}^*}) \in E, v \notin C_{\mathcal{A}^*}\}$  do
21      $O_{\mathcal{A}^*} = O_{\mathcal{A}^*} \cup \{v\},$ 
22      $g_{\mathcal{A}^*}(v) =$ 
        $\min\{g_{\mathcal{A}^*}(v), g_{\mathcal{A}^*}(u_{\mathcal{A}^*}) + c(u_{\mathcal{A}^*}, v)\}$ 
23      $f_{\mathcal{A}^*}(v) = g_{\mathcal{A}^*}(v) + h(v, D_{\mathcal{A}^*})$ 
24   end
25   for  $A \in \mathbb{C}, A \neq \mathcal{A}^*$  do
26     if Path Confirmation Condition between  $\mathcal{A}^*$ 
       and  $A$  is satisfied then
27       If  $terminal_{\mathcal{A}}(u)$  denotes a terminal in  $\mathcal{A}$ 
       that is nearest to  $u, Q := Q \cup$ 
        $PATH^*(terminal_{\mathcal{A}}(u), terminal_{\mathcal{A}^*}(u))$ 
28        $D_{\mathcal{A}^*} := D_{\mathcal{A}^*} \setminus \{t : t \in \mathcal{A} \cap T\}$ 
29        $\forall v \in O_{\mathcal{A}^*}, f_{\mathcal{A}^*}(v) = g_{\mathcal{A}^*}(v) + h(v, D_{\mathcal{A}^*})$ 
30        $D_{\mathcal{A}} := D_{\mathcal{A}} \setminus \{t : t \in \mathcal{A}^* \cap T\}$ 
31        $\forall v \in O_{\mathcal{A}}, f_{\mathcal{A}}(v) = g_{\mathcal{A}}(v) + h(v, D_{\mathcal{A}})$ 
32     end
33   end
34   Let  $\bar{\mathbb{C}}$  denote all the info pertaining to  $\mathbb{C}$ 
35    $[S_T, \bar{\mathbb{C}}] =$ 
      $UpdateSteinerTreeMerge(S_T, Q, \bar{\mathbb{C}})$ 
36 end

```

Algorithm 4: *UpdateSteinerTree_Merge*(S_T, Q, \bar{C})

```
1  $f^* =$   
    $\max\{\min_{\mathcal{A}}(\min_{u \in O_{\mathcal{A}}} f_{\mathcal{A}}(u)), \min_{\mathcal{A} \neq \mathcal{A}'}(rmin_{\mathcal{A}} +$   
      $rmin_{\mathcal{A}'})\}$   
2  $Q' = Q$  // Process paths locally  
3 while  $Q'$  is nonempty do  
4   Choose a path  $p \in Q'$  with the cheapest cost  
   joining components  $C_1, C_2 \in S_T$   
5   if adding  $p$  to  $S_T$  does not form a cycle AND  
    $cost(p) \leq f^*$  then  
6     Suppose  $p$  connects components  $\mathcal{A}_1, \mathcal{A}_2 \in \bar{C}$   
7      $\bar{C} = ComponentMerge(\mathcal{A}_1, \mathcal{A}_2, \bar{C})$   
8   end  
9   Delete  $p$  from  $Q'$   
10 end  
11 return [ $S_T, \bar{C}$ ]
```

Algorithm 5: *ComponentMerge*($\mathcal{A}_1, \mathcal{A}_2, \bar{C}$)

```
1  $\mathcal{A}_{12} = \mathcal{A}_1 \cup \mathcal{A}_2$  // merge components  
2  $D_{\mathcal{A}_{12}} = D_{\mathcal{A}_1} \cup D_{\mathcal{A}_2} \setminus \{t : t \in \mathcal{A}_{12} \cap T\}$   
3  $OC_1 := O_{\mathcal{A}_1} \cup C_{\mathcal{A}_1}, OC_2 := O_{\mathcal{A}_2} \cup C_{\mathcal{A}_2}$   
4  $g_{\mathcal{A}_{12}}(u) = \min\{g_{\mathcal{A}_1}(u), g_{\mathcal{A}_2}(u)\}$  for all  
    $u \in OC_1 \cup OC_2$  // merge g costs  
   depending on the set it is in  
5  $C_{\mathcal{A}_{12}} = (C_{\mathcal{A}_1} \cup C_{\mathcal{A}_2}) \setminus \{u : g_{\mathcal{A}_1}(u) < g_{\mathcal{A}_2}(u), u \in$   
    $O_{\mathcal{A}_1} \cap C_{\mathcal{A}_2} \vee g_{\mathcal{A}_2}(u) < g_{\mathcal{A}_1}(u), u \in$   
    $O_{\mathcal{A}_2} \cap C_{\mathcal{A}_1}\}$  // Remove nodes from  
   the closed set if the g cost is  
   lower in the open sets  
6  $O_{\mathcal{A}_{12}} = (OC_1 \cup OC_2) \setminus C_{\mathcal{A}_{12}}$   
7  $f_{\mathcal{A}_{12}}(u) = g_{\mathcal{A}_{12}}(u) + h(u, D_{\mathcal{A}_{12}}) \forall u \in O_{\mathcal{A}_{12}}$   
   // update fcosts  
8 Remove  $\bar{\mathcal{A}}_1, \bar{\mathcal{A}}_2$  from  $\bar{C}$  and add  $\bar{\mathcal{A}}_{12}$  to  $\bar{C}$   
9 return  $\bar{C}$ 
```

from three well-known variants, namely the bidirectional Heuristic Search (HS) with the *fmin* rule (Pohl 1969), bidirectional Best-first Search (BS) with the *gmin*-based rule (Nicholson 1966) which also mimics the classic primal-dual algorithms (Agrawal, Klein, and Ravi 1995; Goemans and Williamson 1997), and Meet-in-the-Middle (MM) Search (Holte et al. 2017). These versions are correspondingly referred to as S*-HS, S*-BS and S*-MM. The following discussion presents the path criterion used in each of them.

- **Path confirmation criterion for S*-HS:** In this version, we check if there is a node u such that the sum of the g -values of the shortest paths to u from two different components \mathcal{A} and \mathcal{A}^* is at most the larger of the lower bounds for reaching any terminal in the destination sets for \mathcal{A} and \mathcal{A}^* ; in other words, we test if $\min_{u \in V}(g_{\mathcal{A}^*}(u) + g_{\mathcal{A}}(u)) \leq \max(f_{\mathcal{A}^*}(u_{\mathcal{A}^*}), f_{\mathcal{A}}(u_{\mathcal{A}}))$. If this is the case, the path we have found via u represents a least-cost path between \mathcal{A} and \mathcal{A}^* . This stopping

condition is also commonly referred to as the “*fmin condition*” for bidirectional heuristic search (Bi-HS) (Pohl 1969; Sturtevant and Felner 2018).

- **Path confirmation criterion for S*-BS:** Let $gmin_{\mathcal{A}}$ denote the smallest g -value among the nodes in the open set of \mathcal{A} . Note that this is the node in the open set that can be confirmed next according to Dijkstra’s algorithm. We can then use the sum of the values of $gmin_{\mathcal{A}}$ and $gmin_{\mathcal{A}^*}$ to check if there is a least-cost path between any terminal in \mathcal{A} and any terminal in \mathcal{A}^* : $\min_{u \in V}(g_{\mathcal{A}^*}(u) + g_{\mathcal{A}}(u)) \leq gmin_{\mathcal{A}^*} + gmin_{\mathcal{A}}$. This is exactly the stopping condition to confirm a path in bidirectional best-first search (Nicholson 1966; Sturtevant and Felner 2018) which ensures that property (SP) holds for paths confirmed using this rule. Using this criterion also reduces S*-BS to the conventional primal-dual algorithm (Agrawal, Klein, and Ravi 1995) for the Steiner tree problem.
- **Path confirmation criterion for S*-MM:** For MM, we need more definitions. Define c_{min} to be the minimum cost of any edge in the graph. Let the priority of a node u for component \mathcal{A} be defined as $pr_{\mathcal{A}}(u) = \max\{f_{\mathcal{A}}(u), 2g_{\mathcal{A}}(u)\}$ where the first term denotes a lower bound on the cost to any other component and the second is twice the confirmed cost of connecting a terminal in the component to node u . Now, let $prmin_{\mathcal{A}} = \min_{u \in O_{\mathcal{A}}} pr_{\mathcal{A}}(u)$ for any \mathcal{A} . When a pair of components \mathcal{A} and \mathcal{A}^* are evaluated for a path between them, we define $C = \min\{prmin_{\mathcal{A}}, prmin_{\mathcal{A}^*}\}$. We can now use the path criterion from MM (Holte et al. 2017) to confirm a least-cost path between any terminal in \mathcal{A} and any terminal in \mathcal{A}^* as follows: $\min_{u \in V}(g_{\mathcal{A}^*}(u) + g_{\mathcal{A}}(u)) \leq \max\{C, f_{\mathcal{A}^*}(u_{\mathcal{A}^*}), f_{\mathcal{A}}(u_{\mathcal{A}}), gmin_{\mathcal{A}^*} + gmin_{\mathcal{A}} + c_{min}\}$. This ensures property (SP) holds for paths confirmed using this version.

The procedure in Algorithm 4, similar to the *UpdateSteinerTree* procedure in Algorithm 2, ensures paths are added to S_T only if they satisfy the property (K). A key difference in Algorithm 4 is the addition of new bounds to f^* to ensure different versions of S*-merged can be handled efficiently. To do this, for a component \mathcal{A} , we first define $rmin_{\mathcal{A}}$ as the minimum g -value over all nodes in the boundary of \mathcal{A} , namely those nodes in its closed set with a neighbor in its open set. Intuitively, if we draw a ball of this radius around the terminals in \mathcal{A} , every boundary node will occur only at this distance or later, so if we drew such balls around two different components \mathcal{A} and \mathcal{A}' , they would be disjoint. We then generalize the definition of f^* used in the Steiner Tree updating algorithm as follows: $f^* = \max\{\min_{\mathcal{A}}(\min_{u \in O_{\mathcal{A}}} f_{\mathcal{A}}(u)), \min_{\mathcal{A} \neq \mathcal{A}'}(rmin_{\mathcal{A}} + rmin_{\mathcal{A}'})\}$. By the disjointness of these two balls represented by the last term, we can see that using this definition to pick paths satisfies property (K).

Correctness of S*-merged

Since the path confirmation criteria for these three algorithms are directly drawn from the stopping conditions in the corresponding Bi-HS, Bi-BS and MM algorithms, it fol-

lows that the three algorithms obey the (SP) property when they confirm paths between components.

The main point of difference in the merged methods from regular source-destination path-finding algorithms is the definition of open and closed sets since they are now for components rather than just the source or destination. But this is precisely what is handled in the careful redefinition of these sets for a merged component in Algorithm 5. In particular, when components \mathcal{A} and \mathcal{A}' merge, if a node is present in the current closed sets of both \mathcal{A} and \mathcal{A}' , we use the smaller of the two confirmed g -estimates for the shortest path to it. However, if it is present in the closed set of \mathcal{A} and the open set of \mathcal{A}' but the g -estimate is smaller to \mathcal{A}' , then we remove it from the closed set of the merged component since we have a potentially better path from \mathcal{A}' and since it is still in the open set and not confirmed for its shortest path to \mathcal{A}' . The open set of the merged component is simply those nodes in the union of the open sets of both merging components that are not retained in the closed set. Once the f and g costs of the merged components are updated correctly, it also follows that the update Steiner tree method in Algorithm 4 ensures all the three versions of S^* -merged satisfy property (K). This leads to the following theorem.

Theorem 2. *The S^* -merged framework when specialized to any of the three path confirmation criteria (S^* -HS, S^* -BS, S^* -MM) finds a Steiner tree of cost at most equal to the optimal MGPF cost. This Steiner tree can then be used to obtain a 2-approximation algorithm for MGPF.*

Numerical Results

Setup: Computational experiments were conducted on a computer with a 2.80 GHz Intel Core i7-7700HQ processor. All algorithms were implemented in Python 3.6 under Ubuntu 18.04. We compared the number of expanded nodes and runtimes of the proposed algorithms, namely S^* -unmerged, S^* -HS, and S^* -MM, against two conventional solvers, the primal-dual (or S^* -BS) and the naive Kruskal’s approach⁵. Each of the algorithms was evaluated on five separate 8-neighbor type grid maps, obtained via the MAPF benchmark library. These maps (see Table 1) were chosen based on the shape of the obstacles (maze or randomized) or their absence. Within each map, a varying number of terminals ($N = 10, 20, 30, 40, 50$) was randomly generated and placed. For each map and N , 10 problem instances were generated. Comparisons were also made with respect to factors such as merging, reprioritization, and heuristic strengths. Due to space constraints, we first present the results for the “den312d” map in the MAPF library with and without the re-prioritization steps; later, in Table 1, we present results for all the maps for a fixed number of terminals with no re-prioritization.

Heuristics via landmarks: Each map was pre-processed to provide a fast look-up table for heuristic lower-bound estimates between any pair of nodes in the map. Each of these

⁵Here, we implement the approach described in the Background and Preliminaries section. First, we compute the least-cost paths between any pair of terminals to find the metric completion. Then use Kruskal’s algorithm to find a MST for the metric completion.

heuristic estimates was then scaled by a weighting factor w to understand its impact on the overall performance of the algorithms. For small maps, heuristic estimates were obtained by computing the least-costs between any pair of nodes in the map using Dijkstra’s algorithm. However, for moderately-sized maps, estimates were obtained using the ALT method (Goldberg and Harrelson 2005) in combination with the octile distance. To implement the ALT method, one hundred “landmarks” were randomly chosen throughout the maps such that the landmarks were “border nodes” in the graph (with node degree < 8). Dijkstra’s algorithm was then used to find the least-cost from each landmark to the remaining nodes in the map; these least-costs were in turn used to compute a lower bound on the least-cost between any pair of nodes in the map.

Comparisons Based on Expanded Nodes and Time

Fig. 3 shows the average number of expanded nodes and computational time (in secs) as a function of the number of terminals for all the algorithms. The weighting factor w for the heuristics in these results was set to 1. The three heuristic-based algorithms (S^* -unmerged, S^* -HS, and S^* -MM), expanded fewer nodes on average than the conventional solvers which did not use any heuristic information. The merged algorithms (S^* -HS, S^* -MM) outperformed the others in terms of expanded nodes. This trend was consistent across all maps and weighting factors (see Table 1). On the other hand, with respect to average computation times, the primal-dual algorithm (S^* -BS) was competitive in comparison to the other merged versions and S^* -unmerged on the tested instances (this can also be observed in Table 1). These runtimes were also dependent on whether the re-prioritization steps (both in S^* -unmerged and S^* -merged) were switched on or off. This will be examined in the next subsection.

Impact of Re-prioritization

The results in Fig. 3 show that the number of expanded nodes, on average, reduced by nearly 50% with re-prioritization for algorithms (S^* -unmerged, S^* -HS, and S^* -MM) at the expense of some additional computation time; these reductions also become more pronounced as the number of terminals increased. While re-prioritization did not significantly affect the computation times, the trends show that this will be a factor for a larger number of terminals. This overhead is likely linked to the data structures used for the open sets in the algorithms. Presently, each open-set is implemented using a binary heap based priority queue. More efficient data structures will be investigated in future work.

Impact of Quality of Heuristics

Results are reported here for problem instances with 50 terminals. Fig. 4 shows the average number of expanded nodes and computational times for each algorithm as a function of the weighting factor (w) used for the heuristics. $w = 0$ is equivalent to using no heuristic estimates and $w = 1$ corresponds to using the best possible estimates (computed using the landmark based algorithms described earlier). In general, we observed that S^* -MM expanded the least number of

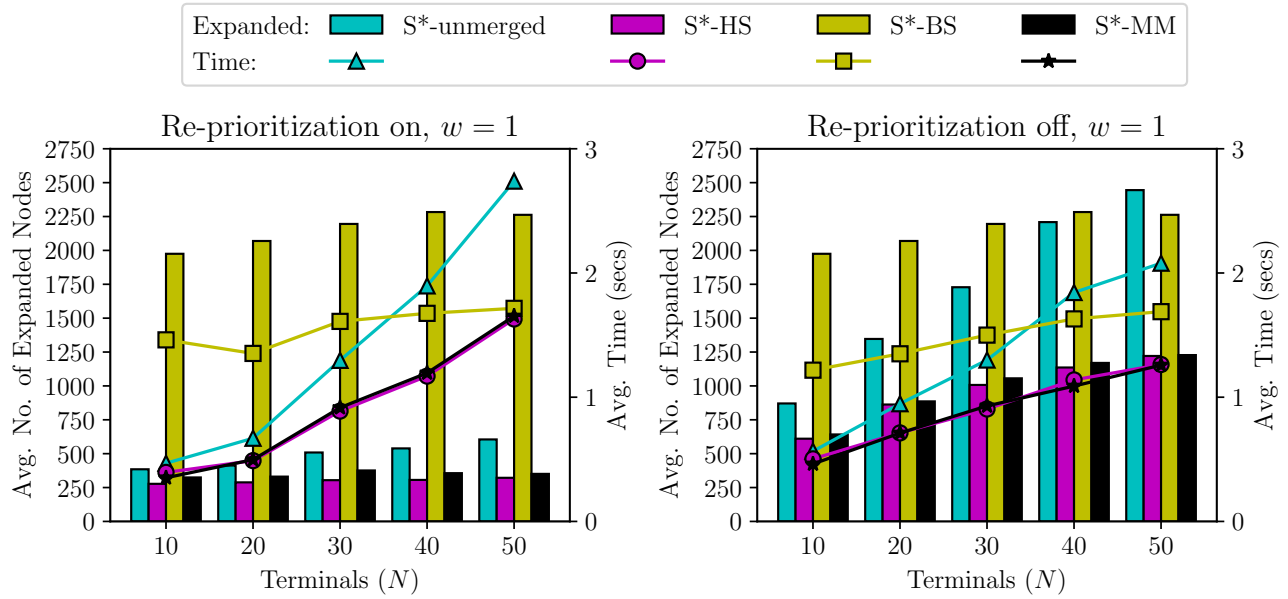


Figure 3: “den312d” map: Average number of expanded nodes and computational times as a function of number of terminals and re-prioritization. w is fixed at 1. For these instances, Naive Kruskal expands between 22005 – 119805 nodes with runtimes in 4.01 – 21.27 secs as N varies from 10 to 50 terminals.

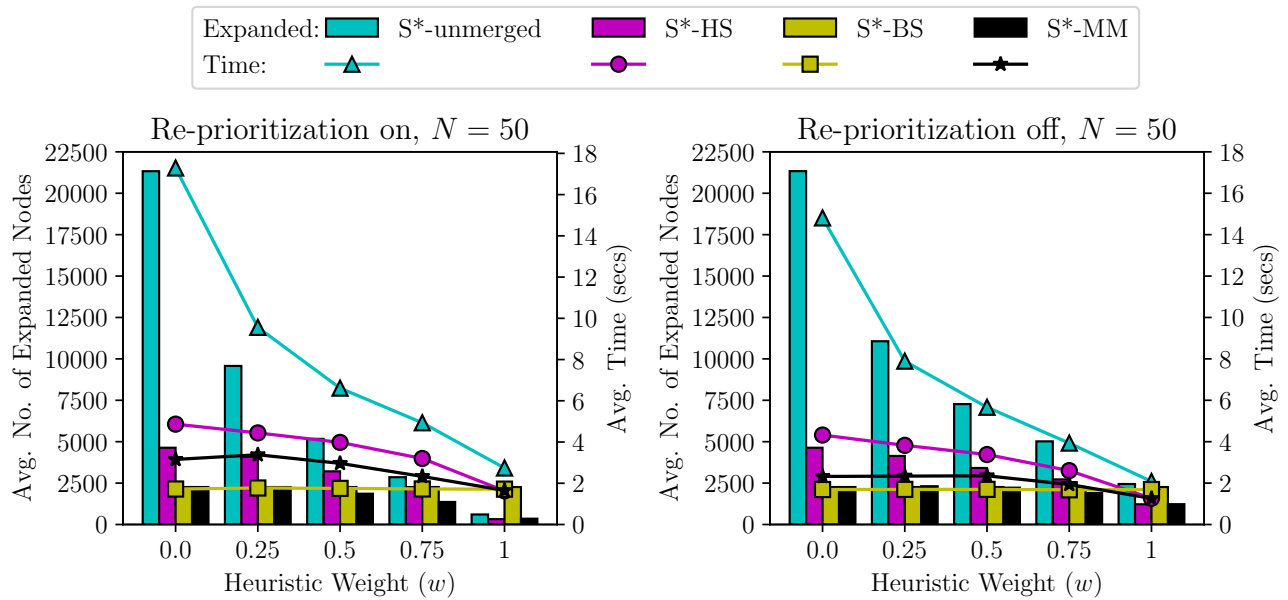


Figure 4: “den312d” map: Average number of expanded nodes and computation times as a function of heuristic strength and re-prioritization. N is fixed at 50.

nodes with lower computational times compared to all the other algorithms (this can also be inferred in Table 1). While the algorithms (S*-unmerged, S*-HS, and S*-MM) expanded significantly a fewer number of nodes in comparison

to S*-BS when $w = 1$ (particularly when re-prioritization is turned on), S*-BS performed better than other algorithms when $w = 0$.

There are also subtle differences between the two merged


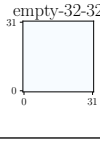
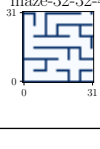
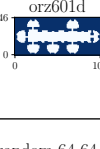
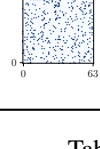
Map	Algorithm	Avg. No. of Expanded Nodes (Avg. runtime in secs)				
		$w = 0$	$w = 0.25$	$w = 0.50$	$w = 0.75$	$w = 1$
 den312d	Kruskal	119805 (21.29)	119805 (21.26)	119805 (21.26)	119805 (21.26)	119805 (21.26)
	S*-unmerged	21334.6 (14.80)	11064.8 (7.89)	7269.6 (5.66)	5019.7 (21.26)	2444.9 (2.07)
	S*-HS	4635.2 (4.32)	4139.7 (3.82)	3412.2 (3.37)	2723.2 (2.60)	1221.0 (1.26)
	S*-BS	2262.6 (1.68)	2262.6 (1.69)	2262.6 (1.69)	2262.6 (1.67)	2262.6 (1.68)
	S*-MM	2262.6 (2.32)	2310.6 (2.33)	2223.8 (2.35)	1906.7 (1.93)	1227.6 (1.25)
 empty-32-32	Kruskal	50176 (9.08)	50176 (9.08)	50176 (9.08)	50176 (9.08)	50176 (9.08)
	S*-unmerged	6201.6 (4.52)	4022.6 (3.01)	2805.6 (2.27)	2026.6 (1.69)	1159.2 (1.05)
	S*-HS	1881.8 (1.84)	1657.4 (1.62)	1231.2 (1.28)	939.9 (0.99)	489.2 (0.55)
	S*-BS	702.0 (0.55)	702.0 (0.57)	702.0 (0.56)	702.0 (0.55)	702.0 (0.56)
	S*-MM	702.0 (0.78)	714.2 (0.80)	776.4 (0.85)	702.3 (0.78)	492.4 (0.54)
 maze-32-32-4	Kruskal	38710 (6.70)	38710 (6.70)	38710 (6.70)	38710 (6.70)	38710 (6.70)
	S*-unmerged	8808.8 (6.17)	4740.3 (3.38)	3178 (2.40)	2361.8 (1.84)	1751.9 (1.44)
	S*-HS	1461.7 (1.39)	1379.7 (1.32)	1138.3 (1.14)	980.7 (0.97)	584.3 (0.59)
	S*-BS	768.9 (0.58)	768.9 (0.58)	768.9 (0.58)	768.9 (0.58)	768.9 (0.59)
	S*-MM	768.9 (0.79)	775.8 (0.80)	759.8 (0.80)	710.3 (0.73)	583.0 (0.59)
 orz601d	Kruskal	92610 (16.48)	92610 (16.48)	92610 (16.48)	92610 (16.48)	92610 (16.48)
	S*-unmerged	14639.7 (10.42)	8503.5 (6.26)	5745.9 (4.54)	3828.9 (3.10)	1994.1 (1.76)
	S*-HS	3569.2 (3.35)	3152.9 (2.99)	2515.6 (2.54)	1949.4 (1.96)	927.3 (0.99)
	S*-BS	1672.7 (1.25)	1672.7 (1.25)	1672.7 (1.26)	1672.7 (1.24)	1672.7 (1.26)
	S*-MM	1672.7 (1.68)	1723.5 (1.74)	1684.3 (1.26)	1424.9 (1.48)	931.7 (0.97)
 random-64-64-10	Kruskal	180663 (31.31)	180663 (31.31)	180663 (31.31)	180663 (31.31)	180663 (31.31)
	S*-unmerged	24270.9 (17.41)	12976.3 (9.57)	8055.3 (6.36)	4951.1 (2.25)	1981 (1.78)
	S*-HS	6869.7 (6.45)	5779.4 (5.49)	4366.9 (4.38)	2981.1 (3.01)	1111.3 (1.25)
	S*-BS	2792.7 (2.07)	2792.7 (2.08)	2792.7 (2.10)	2792.7 (2.09)	2792.7 (2.09)
	S*-MM	2792.7 (2.87)	2899.5 (3.03)	2794.2 (2.97)	2171.2 (2.25)	1137.2 (1.24)

Table 1: Summary of results for $N = 50$ terminals and varying heuristic weights with no re-prioritization.

Map	Min	Avg	Max
den312d	1.780	1.872	1.966
empty-32-32	1.788	1.900	1.976
maze-32-32-4	1.680	1.846	1.977
orz601d	1.693	1.835	1.962
random-64-64-10	1.815	1.882	1.939

Table 2: Minimum, average and maximum *a-posteriori* guarantees obtained for all the test instances.

heuristic-based algorithms. Figure 4 shows that using stronger heuristics have a greater effect on S*-HS than with S*-MM. This is because in general, we observe that MM confirms paths more aggressively than HS especially for less accurate heuristics. When $w = 0$, S*-MM behaves identically to S*-BS, and outperforms S*-HS. When $w = 1$, the performance of both S*-HS and S*-MM are quite similar.

A-posteriori Guarantees of Proposed Algorithms

The quality of the solutions obtained by any of the proposed algorithms for MGPF can be inferred by computing the *a-posteriori* guarantee, *i.e.*, for a given instance, the *a-posteriori* guarantee is defined as the ratio of the cost of

the feasible solution obtained by an algorithm and a lower bound to the optimal cost. The minimum, average and maximum *a-posteriori* guarantees obtained for the tested instances is shown in Table 2. These guarantees are generally lower than the approximation ratio which is a (worst-case) theoretical bound for any instance of the problem. A feasible path is constructed by following the procedure in Fig. 1. The lower bound to the optimal cost used here is simply the cost of the Steiner tree obtained using any of the proposed algorithms.

Conclusions

In this article, a framework called S* was presented for developing a suite of efficient 2-approximation algorithms for MGPF. Additionally, numerical results were also presented to compare the algorithms from the proposed framework with the conventional solvers in terms of the number of expanded nodes and computation time. Overall, the results show that the version of the proposed framework which uses the MM algorithm (Holte et al. 2017) performed the best. Future work can explore decentralized implementations and alternate data structures for faster implementations of S*.

Acknowledgements

This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-20-1-0080, and the Army Research Office under Cooperative Agreement Number W911NF-19-2-0243. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Air Force Office, Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

- Agrawal, A.; Klein, P.; and Ravi, R. 1995. When trees collide: an approximation algorithm for the generalized Steiner problem on networks. *SIAM J. Comput.* 24(3): 440–456.
- Barker, J. K.; and Korf, R. E. 2015. Limitations of Front-to-End Bidirectional Heuristic Search. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI’15, 10861092. AAAI Press. ISBN 0262511290.
- Chen, J.; Holte, R. C.; Zilles, S.; and Sturtevant, N. R. 2017. Front-to-End Bidirectional Heuristic Search with Near-Optimal Node Expansions. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 489–495.
- Chour, K.; Rathinam, S.; and Ravi, R. 2021. S*: A Heuristic Information-Based Approximation Framework for Multi-Goal Path Finding. arXiv 2103.08155.
- De Champeaux, D. 1983. Bidirectional Heuristic Search Again. *J. ACM* 30(1): 2232.
- Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1(1): 269–271.
- Eckerle, J. 1994. An optimal bidirectional search algorithm. In Nebel, B.; and Dreschler-Fischer, L., eds., *KI-94: Advances in Artificial Intelligence*, 394–394. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-48979-5.
- Goemans, M. X.; and Williamson, D. P. 1997. The primal-dual method for approximation algorithms and its application to network design problems. *Approximation algorithms for NP-hard problems* 144–191.
- Goldberg, A. V.; and Harrelson, C. 2005. Computing the shortest path: A search meets graph theory. In *SODA*, volume 5, 156–165.
- Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* SSC-4(2): 100–107.
- Holte, R. C.; Felner, A.; Sharon, G.; Sturtevant, N. R.; and Chen, J. 2017. MM: A bidirectional search algorithm that is guaranteed to meet in the middle. *Artificial Intelligence* 252: 232 – 266.
- Kaindl, H.; and Kainz, G. 1997. Bidirectional Heuristic Search Reconsidered. *J. Artif. Int. Res.* 7(1): 283317. ISSN 1076-9757.
- Karp, R. M. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, 85–103.
- Kou, L.; Markowsky, G.; and Berman, L. 1981. A fast algorithm for Steiner trees. *Acta Informatica* 15(2): 141–145.
- Kruskal, J. B. 1956. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society* 7(1): 48–50.
- Kwa, J. B. 1989. BS*: An admissible bidirectional staged heuristic search algorithm. *Artificial Intelligence* 38(1): 95 – 109. ISSN 0004-3702.
- Lawler, E. L. 2001. *Combinatorial optimization: networks and matroids*. Dover Pubns. ISBN 0486414531.
- Macharet, D. G.; and Campos, M. F. M. 2018. A survey on routing problems and robotic systems. *Robotica* 36(12): 17811803. doi:10.1017/S0263574718000735.
- Mehlhorn, K. 1988. A faster approximation algorithm for the Steiner problem in graphs. *Information Processing Letters* 27(3): 125–128.
- Nicholson, T. A. J. 1966. Finding the Shortest Route between Two Points in a Network. *The Computer Journal* 9(3): 275–280.
- Otto, A.; Agatz, N.; Campbell, J.; Golden, B.; and Pesch, E. 2018. Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey. *Networks* 72(4): 411–458.
- Pohl, I. 1969. Bidirectional and heuristic search in path problems. *Technical Report 104, Stanford Linear Accelerator Center*.
- Ravi, R. 1994. A primal-dual approximation algorithm for the Steiner forest problem. *Information Processing Letters* 50(4): 185–189.
- Rodriguez-Pereira, J.; Fernandez, E.; Laporte, G.; Benavent, E.; and Martinez-Sykora, A. 2019. The Steiner Traveling Salesman Problem and its extensions. *European Journal of Operational Research* 278(2): 615 – 628.
- Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; et al. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. *arXiv preprint arXiv:1906.08291*.
- Sturtevant, N. R.; and Felner, A. 2018. A Brief History and Recent Achievements in Bidirectional Search. In AAAI, 8000–8007.