Search-Based Planning with Extend Operator

Allen Cheng

CMU-RI-TR-20-35
August 2020

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Abstract

Sampling-based approaches are often favored in robotics for high-dimensional motion planning for their fast coverage of the search space. However, at best they offer asymptotic guarantees on completeness and solution quality, and returned paths are typically unpredictable due to their inherent stochasticity. By reasoning through the state space in a procedural fashion, heuristic search-based planners offer high-quality solutions with strong theoretical guarantees but struggle with high dimensionality. In complex domains that demand larger branching factors, this exhaustive behavior can result in considerably longer planning times, known as the "curse of dimensionality." The key factor to fast planning times by sampling-based approaches can largely be attributed to the extend operator, which greedily grows the search tree without regard for the problem's complexity. In this work, we exploit this observation and introduce an extend operator in heuristic search to enable considerable speedups in practice. We explore how this operator directly addresses many difficulties of bidirectional heuristic search and how it naturally extends to the multi-tree setting. We validate our simple approach on high-dimensional manipulation tasks, demonstrating significantly reduced search effort when compared against both search- and sampling-based algorithms. While doing so, we maintain theoretical guarantees on suboptimality and completeness.

Acknowledgements

First, I would like to express my gratitude to my advisor Professor Maxim Likhachev for his wisdom, support, and guidance along this journey. I would also like to thank Professor Jean Oh and Dhruv Mauria Saxena for their valuable insights as members of my thesis committee.

I am grateful for my fellow labmates in the Search-Based Planning Lab whose exciting work serves as continued inspiration. Special thanks to Sandip Aine, Andrew Dornbush, Wei Du, Yash Oza, Tushar Kusnur, and Raghav Sood for their research advice and encouragement. I would also like to thank B.J. Fecich and Peggy Martin for helping me chart my path into robotics.

Lastly, I would like to thank my parents and my sister whose sacrifices and support made it all possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Motion planning is a fundamental problem in robotics to enable collision-free navigation in an environment. Non-optimization-based approaches can be generally categorized as either sampling-based or search-based. Each of the two methodologies has their own advantages and drawbacks.

Search-based methods systematically explore a dense representation of the state space. In doing so, they are able to guarantee completeness for the given resolution and provide suboptimality bounds for the costs of returned paths. By being able to explicitly optimize solutions for criteria such as path length, clearance to obstacles, or smoothness, search-based approaches are an attractive option for robotic motion planning.

Sampling-based approaches capture the connectivity of the configuration space by sampling it. Random exploration allows for fast planning times, even in complex, high-dimensional spaces where search-based methods especially fall short. At the same time, however, such a strategy only allows for asymptotic guarantees at best for completeness and solution quality.

The rise of high-dimensional robots tasked with real-time planning in complex environments demands fast motion planning algorithms that leverage the strengths of both these classes.

1.2 Proposed Approach

Motivated by this need, our work brings the extend operator, a key mechanism of sampling-based approaches, to heuristic search. The extend operator enables fast coverage of the search space and inter-tree connection. In this work, we consider how it can be exploited to connect search frontiers in multi-directional settings for fast, high-dimensional motion planning, while retaining theoretical properties core to heuristic search.

This thesis makes the following contributions:

- Introduction of the extend operator to the context of heuristic search, both for bidirectional and multi-directional settings
- Theoretical analysis of the proposed approach to maintain bounds on suboptimality and completeness
- Experimental evaluations of planning in high-dimensional motion planning domains, such as single-arm planning and mobile manipulation

1.3 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 defines the single-query motion planning problem and provides a brief introduction to popular heuristic search algorithms A* and Weighted A*. Chapter 3 presents how the extend operator can be applied to bidirectional heuristic search as a greedy connect heuristic to connect search fronts in the spirit of RRT-Connect. We provide theoretical guarantees and their proofs for using extensions in bidirectional WA* (without re-expansions), which are also applicable to other A* variants. Borrowing again from sampling-based approaches, we explore the multi-tree extension for heuristic search in Chapter 4, where we try to split the search space further. Chapter 5 summarizes the contributions of this thesis and discusses directions for future work.

Chapter 2

Background

In this chapter, we formally define the single-query motion planning problem and provide necessary background information.

2.1 Problem Definition

This work focuses on single-query planning problems, where we wish to find a valid sequence of actions that transforms an initial configuration to a goal configuration in a goal set. Let S_{free} denote the free state space. $\mathcal{G}(S_{\text{free}}, E)$ represents the directed search graph. Edges E consist of transitions between state configurations, where these transitional costs serve as the edge costs. For a given initial configuration $s_{\text{start}} \in S_{\text{free}}$ and goal set $S_{\text{goal}} \subseteq S_{\text{free}}$, the problem at hand is to solve for a feasible path $\pi = \{s_0, s_1, \ldots, s_k\}$ such that,

$$\begin{aligned} s_0 &= s_{\text{start}} \\ s_k &\in S_{\text{goal}} \\ s &\in S_{\text{free}} \,\forall\, s \in \pi \\ (s_i, s_{i+1}) &\in E \,\forall\, (s_i, s_{i+1}) \in \pi \end{aligned}$$

The redundancy allows for a set of states to satisfy the goal constraint. However, for multi-directional planners that may conduct search in the opposite direction, an explicit $s_{\text{goal}} \in S_{\text{goal}}$ is used.

2.2 A* Search

A* [1] is one of the most widely used graph search algorithms and can be applied to solve the planning problem of finding a minimum cost path from s_{start} to s_{goal}. Search-based approaches require discretization of the free space S_{free} and instead consider $S \subseteq S_{\text{free}}$ for construction of the search graph $\mathcal{G}(S, E)$. An edge connects a pair of states and a cost function $c(s, s')$ denotes the edge cost between states s and s'. If no such edge exists, $c(s, s') = \infty$. The path found by A* will have the minimum sum of the edge costs. For each state $s \in S$, three values are maintained: the cost to reach the state from the initial state, $g(s)$; the estimated cost-to-go, $h(s)$; and the estimated cost of a path from s_{start} to s_{goal} that contains s, $f(s)$. The priority of a state is determined as $f(s) = g(s) + h(s)$ (line 1, Alg. 1). The g-values of all states are initialized to ∞, and the algorithm begins by setting $g(s_{\text{start}}) = 0$ and placing it in the OPEN list (lines 4-8, Alg. 1). The OPEN list is a minimum priority queue ordered by $f(s)$ and contains frontier states that have been discovered but not yet expanded. On each iteration of the main loop, A* selects state s_{min} with minimum priority from OPEN for expansion (line 10, Alg. 1). Successor function SUCC finds the set of reachable successors for a given state such that $\text{SUCC}(s_{\text{min}}) = \{s' \in S | c(s, s') \neq \infty\}$. A check is made to see if the g-value for successor state s' can be decreased (line 15, Alg. 1). If so, s' is either inserted into Open or has its Priority value updated if it has already been inserted (line 17, Alg. 1). Search either terminates when the s_{goal} is selected for expansion on line 11, Alg. 1 or from exhaustion on line 9, Alg. 1.

The heuristic serves as a guide during search, and planning performance greatly depends on how informative it is. Many theoretical properties of A* are also determined by the choice of heuristic. If $h(s)$ never overestimates the path cost to the goal, then it is *admissible*. This, coupled with the ordering of Open, ensures that expanded states are the most promising states discovered thus far during search. A* terminates when expanding the goal state, ensuring there are no other vertex that leads through a cheaper path. A heuristic is *consistent* if the triangle inequality is satisfied, $h(s) \leq c(s, s') + h(s')$ for state s and its arbitrary successor s', and $h(s_{\text{goal}}) = 0$. Consistency guarantees admissibility.

Planning can also proceed in the reverse direction from s_{goal} to s_{start}. This strategy is known as Backwards A*. In this setting, an explicit goal state $s_{\text{goal}} \in S_{\text{goal}}$ is used.

Algorithm 1 A* Algorithm

```
 1: procedure Priority($s$)
 2:     return $g(s) + h(s)$
 3: procedure main
 4:     for $s \in S$ do
 5:         $g(s) \leftarrow \infty$
 6:     Open $\leftarrow \emptyset$
 7:     $g(s_{\text{start}}) \leftarrow 0$
 8:     Insert $s_{\text{start}}$ into Open with Priority($s_{\text{start}}$)
 9:     while not Open.Empty() do
10:         $s_{\text{min}} \leftarrow$ Open.Top()
11:         if $s_{\text{min}} = s_{\text{goal}}$ then
12:             return ExtractPath($s_{\text{goal}}$)
13:         Remove $s_{\text{min}}$ from Open
14:         for $s' \in$ Succ($s_{\text{min}}$) do
15:             if $g(s') > g(s) + c(s, s')$ then
16:                 $g(s') \leftarrow g(s) + c(s, s')$
17:                 Insert/Update $s$ into Open with Priority($s$)
18:     return null
```

2.3 Weighted A*

A simple but effective extension to A* is Weighted A* (WA*) [2]. The original priority function is adapted to be $f(s) = g(s) + \omega h(s)$, where $\omega \geq 1$ is used as an inflation factor. Increasing ω biases the search to follow the guidance of the heuristic more strongly. While the introduction of ω also makes the heuristic term inadmissible, Likhachev et al. have shown that found solutions in this setting are bounded suboptimal, meaning the found path cost is no more than ω times the optimal solution cost [3]. A CLOSED list is introduced here to track states that have been expanded by search so they are not again re-expanded.

Chapter 3

Bidirectional Heuristic Search with Extend Operator for Motion Planning

This chapter formalizes the extend operator to a bidirectional heuristic search framework, proves its theoretical properties, and demonstrates speedups in approach in high-dimensional motion planning experiments. This chapter is adapted from our earlier work [4]. Compared to the previous version, the algorithm presented here bears several modifications: the priority function is changed to that of WA* to maintain consistency; a generalized bidirectional stopping condition is used, terminating the search according to the least-cost path discovered thus far; and re-expansions of states closed by the opposite search are prohibited (and will terminate the main search).

3.1 Introduction

While planning, the majority of effort is often spent near the start and goal configurations, with a large amount of free space in between. Consider the manipulation problem of pick-and-place in cluttered environments for a high degree-of-freedom (DoF) robotic arm, a challenge commonly faced in industrial settings. Successful grasping often requires tight and minimal tolerances due to clutter around the start and goal configurations. This demands a large amount of search effort around the start and goal region, but the majority of the intermediate path is typically collision-free. Naive search-based planners require procedural reasoning over the possible arm configurations when planning through this free space to maintain optimality guarantees.

Nicholson first proposed bidirectional search as two separate searches initialized from both the start and goal [5]. In theory, these planners can exponentially reduce the number of expanded states, making them an attractive class of algorithms. It is well known that informative heuristics significantly improve search efficiency for unidirectional search, so using them for both searches in a bidirectional algorithm is an intuitive extension. In practice, however, informed bidirectional search can end up expending significantly more effort in the case of missing frontiers or in time spent proving optimality of a found solution. This problem is exacerbated in high-dimensional state spaces, which are commonly found in robotic motion planning problems. Bidirectional sampling-based approaches on the other hand, such as the RRT-Connect algorithm [6], offer fast planning performance and scale well with high-dimensionality but at the cost of deterministic theoretical guarantees on solution quality and completeness.

In our work, we present a simple but effective extension that aims to overcome the known shortcomings of bidirectional heuristic search with the use of an extend operator. The operator was originally proposed as a greedy heuristic for connecting forward and backward searches in bidirectional sampling-based planners. Motivated by the example described earlier, we adopt this operator in the heuristic search setting to efficiently reason through free space in high DoF motion planning. By incorporating the extend operator, we maintain consistency and deterministic bounds on the suboptimality of found solutions, for a given resolution of action and state space.

3.2 Related Work

3.2.1 Bidirectional Planning Algorithms

The RRT-Connect algorithm [6] first introduced the extend function in the context of motion planning without kinodynamic constraints. The sampling-based planner dynamically constructs two rapidly-exploring random trees (RRTs) [7] from the start and goal configurations. It attempts to connect both trees using an extend operator. This operator takes a single, fixed-size step towards the nearest neighbor in the opposite tree. A Connect function repeatedly executes Extend until a node in the opposite tree is reached or an obstacle blocks any further extension attempts. The greedy nature of RRT-Connect leads to fast performance, and its sampling behavior allows it to translate well to high-dimensional motion planning problems. However, the algorithm has no guarantees on optimality or bounded suboptimality, and solution quality can vary greatly. Recent work [8, 9] has attempted to address this by adopting the local rewiring procedure from RRT* [10], but these approaches can only guarantee asymptotic optimality.

Pohl first explored the combination of bidirectional and heuristic search with the classical front-to-back bidirectional heuristic search algorithm BHPA [11]. However, the theoretical benefits were never consistently reflected in experiments. It was later noted that the frontiers for forward and backward search often missed or crossed through each other in practice [12]. This observation, considered to be crux of bidirectional heuristic search, motivated Champeaux and Sint to develop the Bidirectional Heuristic Front-to-Front Algorithm (BHFFA) [13]. BHFFA achieves reduced expansions compared to BHPA at the cost of having to exhaustively compute a front-to-front distance for entire frontiers. For high-dimensional motion planning, this is prohibitively expensive.

Recent research has focused on ensuring the two search efforts "meet in the middle" through the use of a unique priority function [14]. However, this property has no clear relevance to the context of complex motion planning, where frontier intersection may not always be in the middle.

The recently introduced A*-Connect algorithm [15] leverages a Multi-Heuristic A* (MHA*) [16] framework to guide search bidirectionally towards both the opposite root and frontier in parallel. MHA* provides structure for enabling the use of multiple inadmissible heuristic functions, while still providing guarantees on completeness and bounds on suboptimality via a consistent anchor heuristic. MHA* maintains separate priority queues containing the frontier states for each heuristic for each search. Specifically, A*-Connect runs two Improved MHA* searches [17]. For each, the consistent anchor heuristic serves as a front-to-back estimate. Additional heuristics, referred to as *connect heuristics*, are used to estimate the front-to-front cost to the opposite frontier. However, instead of exhaustively computing the front-to-front heuristic between all pairs of states in the frontier of the opposite search, A*-Connect computes the connect heuristic for selected pivot states. Pivot states are the last expanded states of the opposite search for each heuristic, and serve as an estimate for the "most promising" state for frontier connection. Effectively, the connect heuristic is a fast-to-compute estimate of the cost to the opposite frontier. A*-Connect produces bounded suboptimal solutions by addressing the issue of crossing frontiers in an efficient, structured manner. Our proposed algorithm similarly uses front-to-end heuristics for structured bidirectional search, but instead of an additional connect heuristic to guide front-to-front search, an extend operator is used to greedily extend to the opposite tree.

3.2.2 Tree Extension

The extend operator responsible for connecting pairs of states has been widely explored in robotic motion planning. The extend operator's job is two-fold: it must select a state as a local goal using a distance metric and perform the extension itself. For bidirectional sampling-based methods, nearest-neighbors (NN) are selected as extension targets. For problems with kinematic or kinodynamic constraints, the extension must solve a two-point boundary value problem (BVP). Lavalle proposes motion primitives that describe a predefined set of discretized control actions [18], but this does not fully solve the BVP. Prior research has offered solutions that include the use of optimal controllers [19] and splines [20]. Since we use the same nearest-neighbors extend operator in our approach, these methodologies and adaptations to various domains can be easily adopted into our algorithm.

Cohen et al. use adaptive motion primitives that are generated on the fly during search [21]. In the pick-and-place scenario we introduced previously, they are used to connect the search frontier to the goal configuration. The goal may be underdefined vis-á-vis degrees-of-freedom of the robot and thus may differ from the dimensionality of the search space. For example, we may specify a 6 DoF goal pose in $SE(3)$ while planning for a 7 DoF robot arm. The redundant joint allows for a set of states that satisfy the goal constraint. An analytical inverse kinematics (IK) solver is used to generate a valid state when the search is sufficiently close to the goal. A dynamically constructed motion primitive is used to reach the goal state. For the robotic arm, this translates into a linearly interpolated path to the generated, valid goal state for the target pose. The use of adaptive motion primitives is shown to improve search efficiency while preserving the planner's theoretical guarantees. The extend operator can be viewed as an adaptive motion primitive to connect more generally to any state in the opposite frontier.

3.3 Approach Overview

3.3.1 Notation and Assumptions

We assume the planning problem can be formulated as a graph search problem, where S refers to the finite set of states for the planning domain. The start and goal states are denoted as s_{start} and s_{goal}, respectively. The cost function $c(s, s')$ denotes the cost of the edge between states s and s'. If no such edge exists, $c(s, s') = \infty$. We further assume that $c(s, s') \geq 0 \; \forall \; s, s'$ pairs. Let $g(s)$ denote the current best path cost from s_{start} to s. The successor function, $\textsc{Succ}(s) := \{s' \in S | c(s, s') \neq \infty\}$, denotes the set of reachable successors for state s. The optimal path between s and s' has cost $c^*(s, s')$. The optimal path from s_{start} to s has cost $g^*(s)$.

The heuristic for state s is denoted as $h(s)$ and is used to estimate the best path cost from s to s_{goal}. $h(s)$ is considered admissible if it never overestimates the path cost to s_{goal}; that is, $h(s) \leq c^*(s, s_{\text{goal}}) \; \forall \; s \in S$. $h(s)$ is considered consistent if it satisfies $h(s_{\text{goal}}) = 0$ and $h(s) \leq h(s') + c(s, s') \; \forall \; s, s'$, where $s' \in \textsc{Succ}(s)$, $s \in S$, and $s \neq s_{\text{goal}}$. OPEN denotes a priority queue typically ordered with priority $f(s) = g(s) + h(s)$, or $f(s) = g(s) + \omega h(s)$ given an inflation factor $\omega \geq 1$. This is used to store frontier states, and CLOSED denotes states that have been expanded. We assume OPEN has function MinKey() that returns the minimum priority value among contained states. Similarly, function Top() returns the state corresponding to this value. If the queue is Empty(), MinKey() returns ∞.

We use dir and $\overline{dir}$ to indicate the current and opposite search directions, respectively. If $s \notin (\textsc{Open}_{\text{dir}} \cup \textsc{Closed}_{\text{dir}})$, $g_{\text{dir}}(s) = \infty$. An extension function, $\textsc{NewState}(s, s_{\text{NN}}, s')$ (line 7, Alg. 2), takes in search state s and the nearest neighbor in the opposite tree s_{NN}. It greedily moves from s towards s_{NN} by taking a step of ϵ-distance (or smaller if the distance from s to s_{NN} is smaller) and is successful if resulting state s' is reachable and collision-free.

3.3.2 Algorithm

At a high level, our algorithm runs two A*-like searches from both start and goal configurations, while using an extend operator to guide front-to-front connection. We choose to analyze our strategy using weighted A* (WA*) [2] to run the forward and backward searches. In effect, an inflation factor $\omega \geq 1$ is used to bias the front-to-back heuristic. Since our approach is bidirectional, we initialize separate OPEN and CLOSED sets for each search direction. s_{start} is initialized as the start configuration for the forward search, and the goal configuration for the backward search. s_{goal} follows in a complementary manner. We also maintain the cost of the least-cost path from s_{start} to s_{goal} throughout search, as paths might be found quickly but have not yet been shown to be within suboptimal bounds. The cost of this path is denoted as u and is initialized to ∞ at the start of search (line 20, Alg. 3). The main loop iteratively runs WA* where CONNECT (line 20, Alg. 2) is called after every expansion. The checks on line 9, Alg. 3 and line 21, Alg. 2 ensure that the considered states for expansion and connection, respectively, have not been closed by the opposite search. If connection is successful, the connecting state in the opposite frontier is added to the current OPEN list. After each iteration, the search direction is swapped (line 35, Alg. 3). Search stops when u satisfies the bidirectional TERMINATIONCRITERION (line 1, Alg. 3). This occurs when the cost of the best path found so far is less than or equal to the estimated costs from either search direction (line 2, Alg. 3). At this point, the search terminates, and the solution path is reconstructed by recursively tracing the parents of the connecting state in both directions.

3.3.3 Extend Operator

The connection process, depicted in Fig. 3.1, is a slightly modified version of the one used in RRT-Connect and can be viewed as a greedy front-to-front heuristic to address the problem of missing frontiers. Given a state and the current search direction, CONNECT first selects the nearest neighbor s_{NN} in the opposite search tree using a domain-dependent distance function, $\triangle$ (line 4, Alg. 2). It then attempts to connect to the chosen state by repeatedly applying EXTEND until either s_{NN} is reached or no further extensions are possible. Feasible extension states from NEWSTATE are treated similarly to successor states in A* expansion and added to $\textsc{Open}_{\text{dir}}$ with PRIORITY if s' is not in $\textsc{Closed}_{\text{dir}}$. The algorithm can vary depending on the chosen frequency at which valid intermediate states

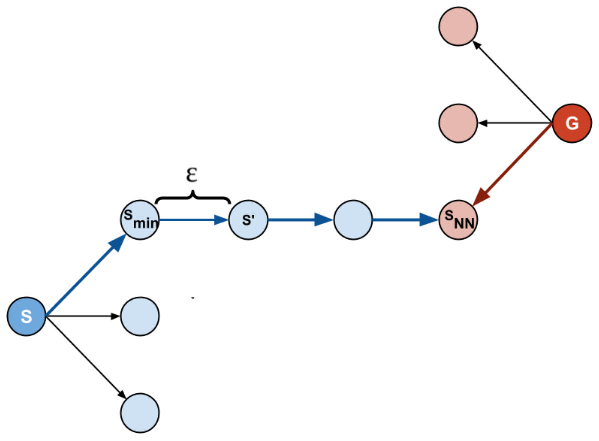


Figure 3.1: An illustration of the extend operator. On expansion, $s_{\min}$ takes ϵ-size step to successor states s' towards the nearest-neighbor state s_{NN}. If a connection is made s_{NN} has the root-to-state cost for forward and backward search, as shown with blue and red edges, respectively.

s' are added to the Open list. In the context of heuristic search, intermediate states added from extend iterations may not lie on the discretized graph induced by the Succ function. For this reason, we focus on the variation that does not add these intermediate successors, effectively requiring that ExtendStatus = *Reached* before adding to $\mathrm{Open}_{\mathrm{dir}}$ on line 17 in Alg. 2.

Algorithm 2 Extend Operator

```
procedure PRIORITY($s$, dir)
    return $g_{\text{dir}}(s) + \omega \cdot h_{\text{dir}}(s)$
procedure NEARESTNEIGHBOR($s_{\text{cur}}$, dir)
    return $\arg\min_{s \in \text{OPEN}_{\overline{\text{dir}}} \cup \text{CLOSED}_{\overline{\text{dir}}}} \triangle(s, s_{\text{cur}})$
procedure EXTEND($s, s_{\text{NN}}$, dir)
    EXTENDSTATUS ← *Trapped*
    if NEWSTATE($s, s_{\text{NN}}, s'$) then
        if $s' = s_{\text{NN}}$ then
            EXTENDSTATUS ← *Reached*
        else
            EXTENDSTATUS ← *Advanced*
        if $s' \notin \text{OPEN}_{\text{dir}} \cup \text{CLOSED}_{\text{dir}}$ then
            $g_{\text{dir}}(s') \leftarrow \infty$
        if $g_{\text{dir}}(s') > g_{\text{dir}}(s) + c(s, s')$ then
            $g_{\text{dir}}(s') \leftarrow g_{\text{dir}}(s) + c(s, s')$
            if $s' \notin \text{CLOSED}_{\text{dir}}$ then
                Insert/Update $s'$ in $\text{OPEN}_{\text{dir}}$ with PRIORITY($s'$, dir)
            UPDATELEASTCOSTPATH($s'$, dir)
    return EXTENDSTATUS
procedure CONNECT($s$, dir)
    if $s \notin \text{CLOSED}_{\overline{\text{dir}}}$ then
        $s_{\text{NN}} \leftarrow$ NEARESTNEIGHBOR($s$, dir)
        repeat
            EXTENDSTATUS ← EXTEND($s, s_{\text{NN}}$, dir)
        until not EXTENDSTATUS = *Advanced*
```

Algorithm 3 Bidirectional WA* with Extend Operator

```
procedure TerminationCriterion($u$)
    return $u \leq \max\big(\text{Open}_{\text{dir}}.\text{MinKey}(), \text{Open}_{\overline{\text{dir}}}.\text{MinKey}()\big)$
procedure UpdateLeastCostPath($s, \text{dir}$)
    if $g_{\text{dir}}(s) + g_{\overline{\text{dir}}}(s) < u$ then
        $u \leftarrow g_{\text{dir}}(s) + g_{\overline{\text{dir}}}(s)$
        $s_{\text{c}} \leftarrow s$
procedure Expand($s, \text{dir}$)
    Remove $s$ from $\text{Open}_{\text{dir}}$
    if $s \notin \text{Closed}_{\overline{\text{dir}}}$ then
        $\text{Closed}_{\text{dir}} \leftarrow \text{Closed}_{\text{dir}} \cup \{s\}$
        for $s' \in \text{Succ}_{\text{dir}}(s)$ do
            if $s' \notin \text{Open}_{\text{dir}} \cup \text{Closed}_{\text{dir}}$ then
                $g_{\text{dir}}(s') \leftarrow \infty$
            if $g_{\text{dir}}(s') > g_{\text{dir}}(s) + c(s, s')$ then
                $g_{\text{dir}}(s') \leftarrow g_{\text{dir}}(s) + c(s, s')$
                if $s' \notin \text{Closed}_{\text{dir}}$ then
                    Insert/Update $s'$ in $\text{Open}_{\text{dir}}$ with Priority($s', \text{dir}$)
                UpdateLeastCostPath($s', \text{dir}$)
procedure main()
    $u \leftarrow \infty$
    $s_{\text{c}} \leftarrow \emptyset$
    $\text{Open}_f \leftarrow \emptyset, \text{Open}_b \leftarrow \emptyset$
    $\text{Closed}_f \leftarrow \emptyset, \text{Closed}_b \leftarrow \emptyset$
    $g_{\text{f}}(s_{\text{start}}) \leftarrow 0, g_{\text{f}}(s_{\text{goal}}) \leftarrow \infty$
    $g_{\text{b}}(s_{\text{start}}) \leftarrow \infty, g_{\text{b}}(s_{\text{goal}}) \leftarrow 0$
    Insert $s_{\text{start}}$ into $\text{Open}_f$ with Priority($s_{\text{start}}, \text{f}$)
    Insert $s_{\text{goal}}$ into $\text{Open}_b$ with Priority($s_{\text{goal}}, \text{b}$)
    $\text{dir} \leftarrow \text{f}$
    while not $\text{Open}_{\text{dir}}$.Empty() do
        $s_{\text{min}} \leftarrow \text{Open}_{\text{dir}}.\text{Top}()$
        if TerminationCriterion($u$) then
            return ExtractPath($s_{\text{c}}$)
        Expand($s_{\text{min}}, \text{dir}$)
        Connect($s_{\text{min}}, \text{dir}$)
        $\text{dir} \leftarrow \overline{\text{dir}}$
    return null
```

3.3.4 Theoretical Properties

The following theoretical guarantees hold for bidirectional heuristic search using any A* variant for forward and backward search and maintain any unidirectional invariants. For the analysis of this section, we consider the case of using WA* (*without* re-expansions) and follow its proofs [3]. For the full proofs of Theorems 1 and 2, please refer to the Appendix.

Theorem 1. *On expansion for any search* dir *and state* $s = \arg\min_{v \in \textsc{Open}_{\text{dir}}} \textsc{Priority}(v, \text{dir})$*, it holds that* $g_{\text{dir}}(s) \leq \omega g^*_{\text{dir}}(s)$.

Proof. (Sketch) The proof for this theorem follows closely to the proof in WA* without re-expansions [3]. The main difference is the introduction of a Connect operator, which allows for additional successors to be added into $\textsc{Open}_{\text{dir}}$. These states may then contribute to the least-cost path to s, but the check on line 14, Alg. 2 ensures g_{dir}-values for these states are only lowered. Thus, the path cost to s will never increase as a result of adding these additional successor states from Connect. □

Lemma 1. *Suppose state s is selected for expansion and connection at line 30, Alg. 3. If* $s \in \textsc{Closed}_{\overline{\text{dir}}}$*, the main search terminates.*

Proof. State s is added into $\textsc{Open}_{\text{dir}}$ with $\textsc{Priority}(s, \text{dir}) = g_{\text{dir}}(s) + \omega h_{\text{dir}}(s)$ either from expansion or extension on line 17, Alg. 3 or line 17, Alg. 2, respectively. In the following line for both cases, u is updated such that $g_{\text{dir}}(s) + g_{\overline{\text{dir}}}(s)$ serves as the finite upper bound and can only further decrease from the check on line 4, Alg. 3. Since $s \in \textsc{Closed}_{\overline{\text{dir}}}$, it follows that $g_{\overline{\text{dir}}}(s) \leq \omega g^*_{\overline{\text{dir}}}(s)$ from Theorem 1. Thus, $u \leq g_{\text{dir}}(s) + \omega g^*_{\overline{\text{dir}}}(s)$. Rewriting the TerminationCriterion, we have,

$$u \leq \max\left(g_{\text{dir}}(s) + \omega h_{\text{dir}}(s), \textsc{Open}_{\overline{\text{dir}}}.\text{MinKey}()\right)$$

The stopping condition has a lower-bound cost of $g_{\text{dir}}(s) + \omega h_{\text{dir}}(s)$, where $u \leq g_{\text{dir}}(s) + \omega g^*_{\overline{\text{dir}}}(s) \leq g_{\text{dir}}(s) + \omega h_{\text{dir}}(s)$ as h is admissible. The TerminationCriterion is always satisfied, terminating the main search when expanding any state s that has been closed by the opposite search. □

Theorem 2. *On expansion for any search* dir *and state* $s = \arg\min_{u \in \textsc{Open}_{\text{dir}}} \textsc{Priority}(u, \text{dir})$*, it holds that* $\textsc{Priority}(s, \text{dir}) \leq \omega g^*_{\text{dir}}(s_{\text{goal}})$.

Proof. (Sketch) We assume $g^*_{\text{dir}}(s_{\text{goal}}) < \infty$; otherwise the theorem trivially holds. Let $\widetilde{\mathcal{G}}$ denote the subgraph of the original planning graph $\mathcal{G}$ where extensions between nodes of opposite search efforts are not allowed. We will first prove this theorem holds in this setting and use the result to generalize to $\mathcal{G}$.

The proof for this theorem over $\widetilde{\mathcal{G}}$ follows Theorem 2 in [16]. The difference in this setting is that bidirectional searching prevents the current search from expanding a state closed by the opposite search by Lemma 1, in addition to the goal state.

For $\mathcal{G}$, where extensions are possible, additional successor states from Connect may be included in a least-cost path to the current search's respective goal, accounted for in Theorem

1. Now, the least-cost path can be comprised of subpaths corresponding to search efforts from each direction. It may be possible that one search expands all states in its subpath, while the connecting state has yet to be realized by the opposite search. This case, however, reduces to planning over $\widetilde{\mathcal{G}}$. □

Theorem 3 (Bounded suboptimality). *When the main search exits, the returned solution (if one exists) has a cost which is at most ω-suboptimal, or $g(s_{\text{goal}}) \leq \omega g^*(s_{\text{goal}})$.*

Proof. The main search terminates when the criterion is met (line 31, Alg. 3) or when an Open list is exhausted (line 29, Alg. 3).

In the case that the stopping condition is satisfied, let u denote the cost of the least-cost path such that TerminationCriterion(u) is satisfied. Let s_c be the associated shared common state for this path such that $u = g_{\text{dir}}(s_c) + g_{\overline{\text{dir}}}(s_c) = g_{(}s_{\text{goal}})$. From Theorem 2, the minimum Priority values in each Open queue never exceed $\omega g^*(s_{\text{goal}})$. The main search only terminates when $u = g(s_{\text{goal}}) \leq \max(\omega g^*(s_{\text{goal}}), \omega g^*(s_{\text{goal}})) \leq g^*(s_{\text{goal}})$, proving the theorem for finite-cost solutions. □

Theorem 4 (Bounded re-expansions). *No state is ever re-expanded.*

Proof. When a state is expanded in a given search direction *dir*, it is added to Closed$_{\text{dir}}$ (line 9, Alg. 3). It is never added back to Open$_{\text{dir}}$ for re-expansions from the checks on line 16, Alg. 2 and line 16, Alg. 3. This guarantees that search *dir* will never re-expand a state previously expanded by itself. By Lemma 1, search *dir* will also never re-expand a state expanded by opposite search $\overline{dir}$. □

Theorem 5 (Completeness). *The main search terminates and returns a solution if a path from s_{start} to s_{goal} exists in the original finite, planning graph $\mathcal{G}$.*

Proof. No vertices or edges are removed from the original graph $\mathcal{G}$. They are only added during the extension steps. Search is conducted via two A*-like planners, which guarantees completeness in the unidirectional setting. Even in the case of non-intersecting searches, if there is a solution path, a search is guaranteed to find it. □

3.4 Evaluation

3.4.1 Experimental Setup

We compare our approach to search-based and sampling-based planners in the context of high-dimensional, single-arm manipulation. All experiments use a simulated model of a PR2 robot's 7 DoF arm. Each test involves generating collision-free trajectories to a specified goal. Start configurations are randomly generated such that the end-effector positions lie above the cluttered table top. Similarly, goal configurations are generated with the constraint that their end-effector positions are within a feasible region in the shelf. The setup, presented in Fig. 3.2, reproduces manipulation tasks that are common in industrial settings, where the majority of search effort is spent near start and goal regions with large amounts of free space in between. In the presence of high clutter all throughout the workspace, the search less likely is able to utilize the extend operator, leading to performance degradation.

Since any graph search algorithm may be used in our framework, we choose to run our algorithm, weighted A* without re-expansions, bidirectionally with an extend operator (WA*-Extend). We evaluate WA*-Extend's performance against unidirectional WA*, A*-Connect, BHPA$_w$ [22], RRT-Connect, and RRT*. These include unidirectional, bidirectional, search-based, and sampling-based planners.

Heuristics

For high-dimensional planning, it is well known that solutions to simplified, lower-dimensional problems can be used as informative heuristics for the original problem [23]. In the context of single-arm planning, a 6 DoF goal constraint describes the desired end-effector position and orientation in $SE(3)$. The solution to the relaxed problem that only considers the goal position $(x, y, z) \in \mathbb{R}^3$ performs well as a proxy heuristic for the full-dimensional representation. Cohen et al. discretize the environment using 3D voxels and find the shortest feasible path to the simplified goal using breadth-first search, referred to as h_{BFS} [24]. In practice, this heuristic proves to outperform other commonly used estimates such as h_{EUC}, the Euclidean distance between states, when applied to obstacles in cluttered workspaces.

Motion Primitives

Motion primitives are used to encode kinematic constraints of a robot and ensure that a valid transition between adjacent states exists. In search-based planning, the robot's action space is discretized to control the branching factor of search. For the 7 DoF arm, we use a base set of 14 static motion primitives, corresponding to moving each joint by a small, fixed amount in both directions.

Extend Operator

The extend operator used in this experimental setup selects the nearest-neighbor state s_{NN} in joint space and attempts a direct extension using a linearly interpolated path. For all our experiments, the bidirectional search we implement executes the extend operator until either a connection to s_{NN} is made, or until the extension is trapped due to an obstacle, unable to go any farther. Another option would be to directly interpolate between end-effector poses,

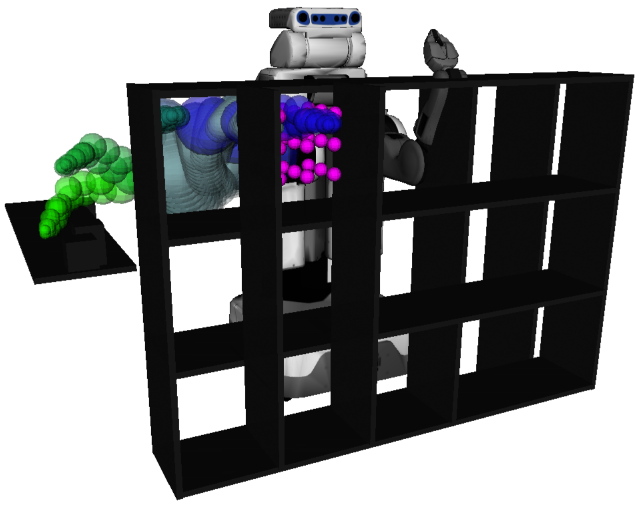


Figure 3.2: Visualization of WA*-Extend for the single-arm planning domain. The found trajectory is colored to denote the different regions of search. Green and blue denote the forward and backward search, respectively, and gray highlights the connection between frontiers. Pink spheres are goal states used in consistency tests.

similar to the approach of Cohen et al. for adaptive snapping to goal configuration [21]. However, this method calls for repeatedly computing inverse kinematics solutions, which can be computationally costly.

Implementation Details

Since our strategy requires extensions to the opposite frontier, we use the k-d tree library nanoflann [25] to dynamically add states and perform efficient NN queries. We also assume an undirected search graph and perform extensions relative to the current search direction. If the graph were directed, extensions from the backwards search would have to be computed in the reverse direction. The search-based planners in our tests use an inflation factor $\omega = 100$. h_{BFS} serves as the front-to-back, anchoring heuristic for A*-Connect as well as the main informative heuristics for remaining heuristic search-based planners. For the connect heuristic, we use the fast-to-compute h_{euc}, as Islam suggests [15]. For bidirectional planners, we swap directions after expanding 10 times [15], but we attempt to connect frontiers on each iteration of WA*-Extend. Our baseline sampling-based planners, RRT-Connect and RRT*, use implementations provided by the Open Motion Planning Library (OMPL) [26]. RRT* is conditioned to terminate after the first valid solution is found. Search-based approaches and RRT* use the distance traveled in joint space as their cost function. We post-process all solutions using existing OMPL implementations to simplify and smooth paths.

3.4.2 Experimental Results

Table 3.1 compares our approach against a suite of planners. We consider 100 different planning problems with randomly generated start and goal pairs such that all planners find a solution for them given a large timeout. We then say a planner *failed* if that solution took

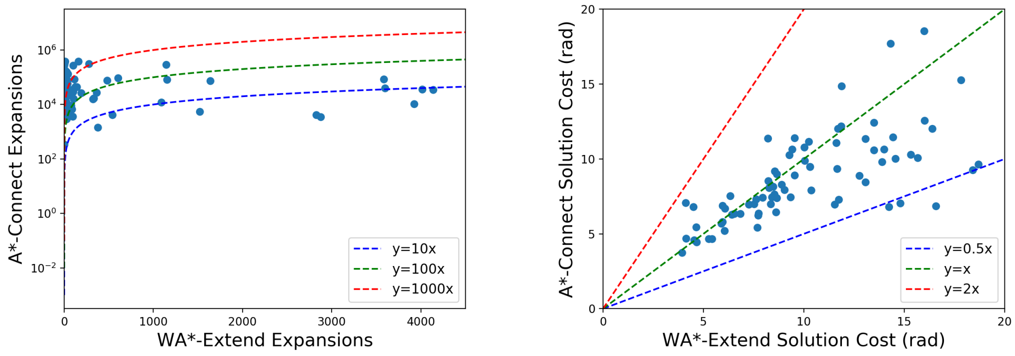


Figure 3.3: A comparison of the WA*-Extend state expansions (left) and post-processed solution costs (right) against A*-Connect for a single run.

longer than 120 seconds to find. Table 3.1 presents the averaged results of trials that were successfully completed across all planners.

Table 3.1: Performance for Single-Arm Motion Planning with Original Algorithm

	WA*-Extend	WA*	A*-Connect	BHPA	RRT-Connect	RRT*
Success (%)	99	58	100	98	100	98
State Expansions	**2027.24**	119284.95	21364.71	43638.84	-	-
Solution Cost (rad)	14.43	**11.17**	11.59	12.77	22.49	16.55
Total Plan Time (s)	3.47	31.58	6.44	17.51	**0.09**	7.09
Post Processing Time (ms)	6.95	7.97	6.63	6.07	5.54	**4.44**
Processed Solution Cost (rad)	11.68	9.59	**9.04**	10.88	12.89	10.41

Comparison with Search-Based Planners

Bidirectional variants are faster at finding solutions compared to unidirectional WA* for the high DoF planning problems we consider. Compared to the other search-based planners, our approach has the largest averaged solution cost but outperforms its search-based counterparts in terms of speed and reduced state expansions. In Fig. 3.3, we compare our approach to A*-Connect, which is more competitive compared to the other bidirectional heuristic search, BHPA$_w$. The left plot shows how WA*-Extend commonly expands 100 to 1000 times fewer states than A*-Connect on each run. The right plot illustrates that comparable solution cost is achieved after post-processing. Ultimately, this highlights the extend operator's effectiveness in efficiently connecting frontiers. In our example domain, this translates to interpolated paths in free regions as shown in Fig. 3.2, where the backward search is able to extend to the opposite frontier after planning out of the constrained cubby region near the goal configuration.

Comparison with Sampling-Based Planners

As observed in Table 3.1, RRT-Connect attains the fastest planning time. WA*-Extend is the second fastest and about twice as fast on average compared to RRT*. Without post-processing, solutions from sampling-based planners, especially RRT-Connect, are of poorer

Table 3.2: Evaluating Solution Consistency over Single Problem

	WA*-Extend	WA*	A*-Connect	BHPA$_w$	RRT-Connect	RRT*
μ_c	12.79	8.81	9.30	15.92	27.43	12.71
σ_c	-	-	-	-	8.06	2.11

Table 3.3: Evaluating Solution Consistency over Similar Problems

	WA*-Extend	RRT-Connect	RRT*
μ_c	14.21	26.92	14.41
σ_c	3.11	15.06	5.91

quality. In highly cluttered environments, post-processing methods might not always be able to converge to optimal solutions.

We also consider solution consistency, the notion that similar problems ideally result in similar solutions. To compare consistency across planners, we borrow the distance metric used by Cohen et al. [21], defined as the ratio between the unprocessed solution path length and Euclidean distance between start and goal end-effector poses,

$$d_c = \frac{|\pi_{\text{planner}}|}{dist(x_{\text{ee, start}}, x_{\text{ee, goal}})}$$

For the first set of tests, we execute 50 runs for each planner with the same start and goal configuration each time. The mean and standard deviation for this consistency metric, μ_c and σ_c, are recorded in Table 3.2. Due to their deterministic behavior, search-based planners do not see variation for the same problem query. On the other hand, solutions from sampling-based planners, especially RRT-Connect, exhibit high variability for the same test.

We repeat the experiment but for similar problem queries. The start is fixed and the 27 goal configurations have corresponding end-effector positions that lie on equidistant points on a 10-cm cube, visualized as pink spheres in Fig. 3.2. Each problem is run once, and results are reported in Table 3.3. We observe that WA*-Extend has much lower variability compared to sampling-based solutions. Inconsistent solutions are undesirable in applications that require high repeatability such as automated manufacturing or predictability when working collaboratively with humans. Grounded in a search-based framework, our approach manages to find consistent solutions.

Comparison with Original Version

Table 3.4 compares the performance of the original version of WA*-Extend presented in [4] to the modified one presented here. These tests are a result of 40 runs, conducted in a similar but more complex environment using the same timeout critera. The reported results are averaged across commonly passed tests. The modified version, which uses a more flexible termination condition, attains faster planning times and reduced expansions. Pre- and post-processed solution costs are very comparable. The modified algorithm also achieves a success rate of 100%, compared to the orginal version's 90%. From these results, it is clear the changes in the modified version yield considerable improvements.

Table 3.4: Performance Comparison using Modified Version

	Original	Modified
Success (%)	90	**100**
State Expansions	43684	**40351**
Solution Cost (rad)	**11.53**	11.56
Total Plan Time (s)	9.454	**8.097**
Post Processing Time (ms)	6.3221	6.0449
Processed Solution Cost (rad)	**11.02**	11.11

3.5 Summary

This chapter presents a simple extension to bidirectional heuristic search that enables fast and bounded suboptimal solutions. Bidirectional heuristic search faces the trade-off between ensuring intersecting frontiers and maintaining competitive planning efficiency. Our strategy capitalizes on large free space in high DoF motion planning to connect frontiers using an extend operator popular in sampling-based planners. By running WA* bidirectionally with an informative front-to-end heuristic, we maintain deterministic, theoretical guarantees that sampling-based planners lack. We also provide consistent solution paths as a result of using deterministic, search-based methods. Our simulated experiments show that our approach achieves significantly faster search efficiency compared with other heuristic-based strategies.

Chapter 4

Multi-Tree Heuristic Search

In this chapter, we extend the bidirectional framework established in the previous section to the multi-tree setting in order to further split the search space. We will detail the algorithm, provide additional optimizations, and evaluate our approach for mobile manipulation in a challenging environment.

4.1 Introduction

The past few decades have seen much research into the practicality of bidirectional approaches in search-based planning. Pohl first related frontiers to passing missiles that never intersect [11]. Kaindl and Kainz instead claimed that the majority of search time is spent proving that a found bidirectional solution is optimal [27]. Barker and Korf refuted this claim with results showing that optimal solutions are often found late in search [28]. In Chapter 3, we show how bidirectional heuristic search can be effective when search complexity is focused at the start and goal regions. Unfortunately, this assumption is unrealistic in many robotic domains. Fig. 4.1. highlights the limits of this assumption, displaying example results from bidirectional search-based approaches for the 3 DoF navigation domain and 10 DoF mobile manipulation domain. In the navigation example, individual search efforts fail to meet due to the cluttered environments and kinematic limitations of the extend operator in this domain. In the full-body mobile manipulation scenario, backward Dijkstra searches over a 2D grid serve as heuristics. They guide each search direction towards the narrow walkway formed by the kitchen island and the right wall, but this area is infeasible due to the open cabinet door. The uninformed heuristics will lead to exhaustive exploration of this region and significant increases in planning times.

These scenarios result in low connectivity between search fronts. To combat similar problems, sampling-based strategies have employed multi-directional search where numerous disjointed trees are grown simultaneously. Growing multiple RRTs has proven to be effective in highly constrained environments where local exploration may be stalled [29]. Accordingly, we propose a multi-tree framework for search-based planning to enable a similar kind of robustness in challenging scenarios.

4.2 Related Work

4.2.1 Multi-Tree Search

Search-based planning methods primarily employ simultaneous search from multiple roots to split the original problem into smaller, short-range searches that can be solved in parallel. PBA* [30] augments bidirectional search by selecting additional search roots termed *X-nodes*. Two additional A* instances are conducted for each X-node, which search to the

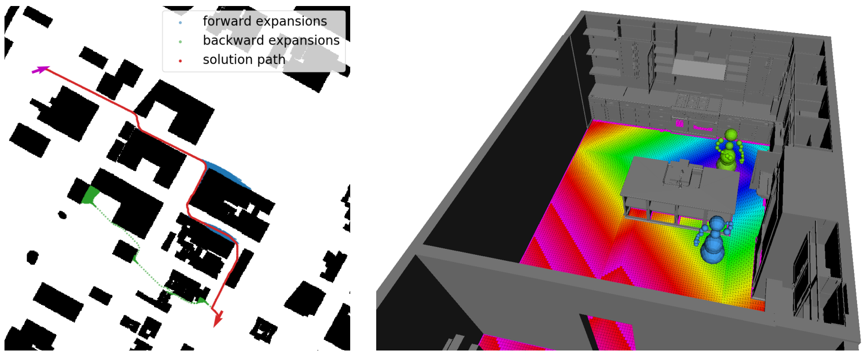


Figure 4.1: (a) An example where bi-frontal search efforts (blue) fail to meet in (x, y, θ) planning. (b) Backwards Dijkstra heuristic guides both searches to a walkway that is too narrow for the mobile robot. The color gradient visualizes costs of this heuristic considering the green configuration as the goal (where blue indicates smaller values than green).

original start and goal states. Ideally, these roots would lie near or on the optimal path between the start and goal states for potentially exponential reductions in each search's efforts. This process is naturally parallelized and a solution is returned when a valid path is first found, considering all search spaces. Variants to PBA* have enhanced the two main components driving the algorithm's efficiency: the identification [31] and encouragement of intersections [32] between search spaces and the selection of X-nodes [33]. Unlike PBA* and its variants, our method is nonparallel in implementation but guarantees bounded suboptimal solutions. Though our approach exploits bidirectionalism in a similar fashion as PBA*, we leverage the bidirectional framework with the extensions presented in Chapter 3 to establish fast connections between searches with theoretical guarantees.

The use of multiple trees to enable broader exploration in sampling-based search has been the subject of extensive research. Some approaches construct a roadmap of RRTs that is maintained between queries and incrementally updated to handle multi-query and dynamic scenarios [34–36]. However, our work is most similar to single-query approaches such as those presented by [37–39], which grow additional RRTs rooted in key regions in the configuration space. These works vary in fundamental multi-tree design decisions, like how additional search roots are selected, when connections between searches are made, and how such connections are made. [37] dynamically grows local trees from samples that failed to connect to a global tree and uses a fixed parameter to control the growth of these local trees. On the other hand, [38] and [39] use a specialized *bridge test* to identify a fixed number of roots located in narrow passage regions. The latter also uses an online meta-algorithm to schedule the order in which searches are expanded. Our proposed method draws on the higher-level design choices made by these works, but we incorporate additional enhancements to the base algorithm, including a more informed sampling method for additional search starts. Our multi-tree approach is inherently search-based and thus offers guarantees on solution suboptimality.

4.3 Approach Overview

4.3.1 Notations and Assumptions

We build from the notation established in Section 3.3. $S_{\text{sampled}} \subseteq S_{\text{free}}$ refers to the set of sampled states used as additional roots in the multi-tree setting. Prior to search, k states are sampled as additional bidirectional search roots. These searches are denoted as T_{f}^{i} and T_{b}^{i} for $i = 1, \dots, k$, and T_{f}^{0} and T_{b}^{0} correspond to the main bidirectional searches between s_{start} and s_{goal}. All of these searches are contained in $\mathcal{T} = \{T_{\text{f}}^{\text{i}} | 0 \leq i \leq k\} \cup \{T_{\text{b}}^{\text{i}} | 0 \leq i \leq k\}$. We assume each search supports its own search functions, properties, and containers, such as $T.g(s), T.s_{\text{start}}$, and $T.$Open.

4.3.2 Algorithm

At a high level, we initialize bidirectional local searches rooted from k samples, in addition to the main, global bidirectional search between s_{start} and s_{goal}. Supplementary searches are performed forwards and backwards from each sample for a total of $2k + 2$ searches (line 7, Alg. 4), each maintaining separate Open and Closed lists.

At each iteration, a non-terminated search is selected for expansion and connection steps in round-robin fashion (lines 9-14, Alg. 4). These processes follow closely to the bidirectional cases in Alg. 3 but are modified for the multi-tree setting. The checks to see if considered state s_{min} has been closed in the opposite search on line 9, Alg. 3 and line 21, Alg. 2 apply only to main bidirectional searches. Connect instead selects the nearest-neighboring state s_{NN} belonging to a tree on a different connected component, which may induce multiple nearest-neighbor calls. Because of this additional criterion, only a single extension will be realized for each pair of searches, resulting in the need to maximize information sharing. For a successful connection between s_{min} and s_{NN}, s_{min} is additionally inserted or updated to the Open list corresponding to s_{NN}'s search. The main search keeps track of the cost of the best solution so far, u, and terminates under the global stopping condition (line 1, Alg. 4). The other $2k$ searches terminate under the unidirectional criterion (line 3, Alg. 4). As we operate and terminate within our bidirectional framework, we maintain the guarantees established in Section 3.3.4.

Algorithm 4 Multi-Tree Heuristic Search with Extend Operator

```
1: procedure GlobalTerminationCriterion($u$)
2:     return $u \leq \max\left(T^0_{\mathrm{f}}.\textsc{Open}.\mathrm{MinKey}(), T^0_{\mathrm{b}}.\textsc{Open}.\mathrm{MinKey}()\right)$
3: procedure LocalTerminationCriterion($T^{\mathrm{i}}_{\mathrm{dir}}$)
4:     return $T^{\mathrm{i}}_{\mathrm{dir}}.g(T^{\mathrm{i}}_{\mathrm{dir}}.s_{\mathrm{goal}}) \leq T^{\mathrm{i}}_{\mathrm{dir}}.\textsc{Open}.\mathrm{MinKey}()$
5: procedure main($S_{\mathrm{sampled}}$)
6:     $u \leftarrow \infty$
7:     $\mathcal{T} \leftarrow$ InitializeSearches($s_{\mathrm{start}}, s_{\mathrm{goal}}, S_{\mathrm{sampled}}$)
8:     while not GlobalTerminationCriterion($u$) do
9:         $T^{\mathrm{i}}_{\mathrm{dir}} \leftarrow$ SelectSearch()
10:        if $T^0_{\mathrm{f}}.\textsc{Open}.\mathrm{Empty}()$ or $T^0_{\mathrm{b}}.\textsc{Open}.\mathrm{Empty}()$ then
11:            return null
12:        $s_{\min} \leftarrow T^{\mathrm{i}}_{\mathrm{dir}}.\textsc{Open}.\mathrm{Top}()$
13:        Expand($s_{\min}, T^{\mathrm{i}}_{\mathrm{dir}}$)
14:        Connect($s_{\min}, T^{\mathrm{i}}_{\mathrm{dir}}$)
15:    ExtractPath()
```

4.3.3 Additional Optimizations

Shortcut Successors

We add additional shortcut successors [40] when a connection between two searches is made (through either expansion or extension). A shortcut state corresponds to the best state in a connected component according to a search's heuristic. Shortcuts help progress a local planner by leveraging efforts from other searches and by reducing the amount of re-expansions and retracings of previously expanded states. They also enable local searches to quickly terminate in the case of a successful extension to a connected component containing its respective goal. As a result, this method offers an alternative to explicitly merging searches when a connection is established, allowing searches to reason through partial paths already generated by other efforts.

To determine the costs of jump successor states, we maintain an abstracted graph $\widetilde{\mathcal{G}}(\mathcal{T}, E_{\text{local}})$. Edges represent connectivity between searches. Edge costs correspond to the current least-cost paths between root pairs, determined and updated by search in the original planning graph $\mathcal{G}$. Vertices that correspond to searches with the same root have edge cost 0, and edges between unconnected searches are initialized as infinity. When a state s' is determined as a jump successor to s, we compute cost $c(s, s')$ in $\mathcal{G}$ using $\widetilde{\mathcal{G}}$. s and s' are temporarily added as vertices to $\widetilde{\mathcal{G}}$ and are connected to searches that additionally contain them. The associated edge costs are the g-values for the state in the respective search. The cost $c(s, s')$ in the original planning graph $\mathcal{G}$ is then the cost of the least-cost path between s and s' in $\widetilde{\mathcal{G}}$. In [40], paths between vertex pairs in the shortcut graph can be computed prior to search as the graph is static.

Critical States as Search Roots

While any number of additional searches could be employed, it is ideal to use them sparingly to reduce overhead costs associated with maintaining multiple searches. With this in mind, we propose to identify states critical for a given environment in a pre-processing phase following [41] (line 1, Alg. 6) and sample k critical states as additional roots prior to search (line 4, Alg. 6). While resulting critical states are not directly associated with local minima in contrast to [42], they are often located in areas of high connectivity and effective when used in conjunction with the extend operator. Preliminary experiments have also shown that this approach is superior to explicitly using the samples as extension targets.

The criticality metric for a state is measured using the notion of *betweenness centrality* [43] and depends on the number of shortest paths that pass through a given state. To measure centrality scores for states in a static environment, a standard probabilistic roadmap (PRM) [44] is first constructed in S_{free} denoted as $\mathcal{G}_{\text{PRM}}$ (line 2, Alg. 6). A database maintains each vertex's criticality score, all of which are initialized as 0 (line 2, Alg. 5). Then, many one-to-all shortest path problems are solved over this PRM, where a shortcutting process, ReduceVertices, removes any noncritical solution vertices that can be skipped without invalidating the found path (lines 6-7, Alg. 5). This step aims to remove any states that may lie in large, open spaces and are therefore not necessarily helpful in capturing the environment's complexity. Each surviving, non-endpoint vertex then has its centrality score incremented (line 9, Alg. 5). Once a database of vertices and their respective criticality scores has been computed, states can then be sampled in direct proportion to their scores. An example of identified critical states for a kitchen environment appears in Fig. 4.3.

In the original work [41], this process is used to generate training data to predict a sample's criticality scores conditioned on local environment features. We solely use this approach to construct a database of critical states and their scores for a given environment. Since roots are sampled from the continuous space, we first SNAP them to the center of a discrete cell before use (line 13, Alg. 5). In practice, the benefits of this selection method are three-fold: the number of roots can be minimized for greater search efficiency; regions that often induce local minima, such as narrow passage regions, are successfully captured; and the approach can be scaled to similar, new environments.

Algorithm 5 Critical Sampling

```
 1: procedure BUILDDATABASE($\mathcal{G}_{\text{PRM}}, m$)
 2:     $\text{DB}_{\text{crit}} : V_{\text{PRM}} \to \{0 | v \in V_{\text{PRM}}\}$
 3:     for $i = 0, \ldots, m$ do
 4:         $v_{\text{start}} \leftarrow$ SAMPLE($V_{\text{PRM}}$)
 5:         for each $v_{\text{goal}} \in V_{\text{PRM}} \setminus \{v_{\text{start}}\}$ do
 6:             $P \leftarrow$ SOLVE($\mathcal{G}_{\text{PRM}}, v_{\text{start}}, v_{\text{goal}}$)
 7:             $P \leftarrow$ REDUCEVERTICES($P$)
 8:             for each $v \in P \setminus \{v_{\text{start}}, v_{\text{goal}}\}$ do
 9:                 $\text{DB}_{\text{crit}}(v) \leftarrow \text{DB}_{\text{crit}}(v) + 1$
10:     return $\text{DB}_{\text{crit}}$
11: procedure INITIALIZECRITICALSTATES($k, \text{DB}_{\text{crit}}$)
12:     $I \leftarrow$ Sample $k$ critical states from $\text{DB}_{\text{crit}}$ proportional to criticality scores
13:     $I \leftarrow$ SNAP($I$)
14:     return $I$
```

Algorithm 6 Multi-Tree Heuristic Search with Critical Samples

```
1: procedure PREPROCESSINGPHASE($S_{\text{free}}, m$)
2:     $\mathcal{G}_{\text{PRM}} \leftarrow$ CONSTRUCTPRM($S_{\text{free}}$)
3:     $\text{DB}_{\text{crit}} \leftarrow$ BUILDDATABASE($\mathcal{G}_{\text{PRM}}, m$)
4: procedure QUERYPHASE($k, \text{DB}_{\text{crit}}$)
5:     $S_{\text{crit}} \leftarrow$ INITIALIZECRITICALSTATES($k, \text{DB}_{\text{crit}}$)
6:     RUNMULTITREESEARCH($S_{\text{crit}}$)
```

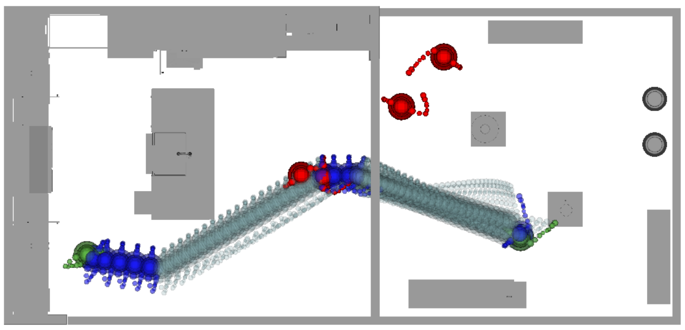


Figure 4.2: Visualization of MT-WA*-Extend for the mobile manipulation planning domain, featuring a kitchen (left) and living room (right). A narrow doorway exists to connect the two rooms, as implied by the found solution. The query start and goal configurations are denoted in green, and critical states are shown in red. The found solution is visualized (at differing resolutions for improved visibility) with blue and gray states indicating states found from procedural expansion and extension, respectively.

4.4 Evaluation

4.4.1 Experimental Setup

We evaluate our proposed multi-tree approach for high-dimensional mobile manipulation using the environment shown in Fig. 4.2. We simulate the UBTECH Walker robot with a mobile base represented with a 10 DoF state space: (x, y, θ) for the 3 DoF base and $(j_1, \dots, j_7)$ for the 7 DoF arm. Each planning problem involves generating a collision-free path for start-goal configurations within a fixed time budget. Start configurations are randomly generated such that their base positions lie within the left room, biased to be closer to the kitchen island. Goal configurations are similarly generated, with the constraint that their end-effector is positioned over one of the tables in the living room. The tested environment features two main sources of local minima: the narrow walkway between the kitchen island and the north wall, and the doorway connecting the two rooms.

We compare our multi-tree planner (MT-WA*-Extend) with the previously discussed optimizations against the bidirectional version WA*-Extend. We use weighted A* without re-expansions as the base planners for each algorithm. In addition, we benchmark MT-WA*-Extend against RRT-Connect, a multi-tree variant of RRT-Connect (MT-RRT-Connect), and PRM ran in the single-query setting. For a fair comparison, sampling-based planners are also augmented to best utilize a priori information of sampled critical states used as roots in our approach.

Mobile Manipulation

For the mobile manipulation domain, we use the maximum of admissible heuristics corresponding to the arm's end-effector and the robot's base. h_{BFS}, as described in Section 3.4, is used to guide the end-effector. Similarly, a backwards 2D Dijkstra is used to guide the mobile base.

Motion primitives are small feasible motions that are used to discretize the robot's action space in search-based planning. During search, an implicit graph is constructed, where the number of motion primitives decides the branching factor. For 10 DoF mobile manipulation, we use 36 primitives consisting of small joint movements in both directions and at varying magnitudes.

Implementation Details

The PRM used to construct a database of critical samples is allotted 90 seconds and adopts a 2:1 ratio for growing and expanding the roadmap. Each vertex attempts to connect to its 10 nearest neighbors (default OMPL settings). The constructed PRM $\mathcal{G}_{\text{PRM}}(V_{\text{prm}}, E_{\text{prm}})$ where $|V_{\text{prm}}| = 17607$ and $|E_{\text{prm}}| = 458328$. To determine criticality scores for these states, we sample (without replacement) 50 start vertices and plan to 2000 sampled goal vertices for each start. We post-process the 100k solution paths such that only key states remain and see their respective centrality scores increment. A fixed number of samples are drawn in proportion to their criticality scores prior to starting search.

The sampling-based planners are extended from OMPL implementations and modified to utilize pre-processed environmental information. PRM is run in the single-query setting, where each vertex in the roadmap has an assumed edge to all critical samples. To encourage growth towards critical states, RRT-Connect is given a 25% chance to select a root as its sample, akin to goal biasing. The MT-RRT-Connect follows design choices similar to those of our multi-tree implementation, where expansions are done in round-robin fashion and a single connection is attempted from the nearest state within a different tree. When connections are made, search trees are merged and removed from selection for expansion.

Search-based planners use an inflation factor $\omega = 100$. MT-WA*-Extend expands searches with round-robin scheduling and WA*-Extend changes search direction after each iteration. Both planners attempt to connect frontiers on each iteration. Search-based approaches use the distance traveled in joint space as their cost function. The L1 and L2 norms are used for angular and linear components, respectively. We post-process all solutions with OMPL to simplify and smooth paths.

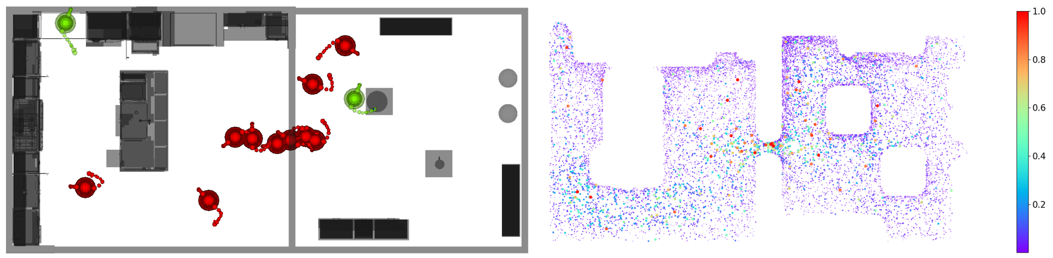
4.4.2 Experimental Results

Our experimental results demonstrate that the selection of additional starts in multi-tree search significantly affects planning performance for both search-based and sampling-based methods. We first evaluate critical samples as search roots compared to those that have been randomly drawn or selected by a user.

We then evaluate MT-WA*-Extend against other planners for 50 planning problems with randomly generated start-goal pairs and a fixed timeout of 60 seconds. Planners with randomized components, such as sampled additional search roots, are run 3 times for each problem. Table 4.1 compares planning performances across all planners and only considers pairwise common successes with presented approach MT-WA*-Extend. Results are presented as ratios with respect to those of MT-WA*-Extend.

Evaluating Critical Samples

To better understand multi-tree performance in a search-based setting, we compare performance when using varied amounts of additional bidirectional searches as well as different



(a) (b)

Figure 4.3: (a) An example problem for the kitchen environment with start and goal configurations (green) and 10 critical sampled roots (red). (b) The base positions of critical states and their respective normalized centrality scores are visualized, with their marker size and color denoting relative magnitude.

sampling strategies for choosing their roots (random, critical, and hand-picked). We fix the start and goal states, such that the scenario's key complexity is the doorway connecting the two rooms. Samples near or in this passageway aid in search, while others prove to be much less informative. Randomized sampling methods are repeated 5 times. Since hand-picking roots is deterministic, there are no standard deviation error bars. Fig. 4.4 presents the results.

Hand-picked roots result in the fewest expansions as the user is able to directly identify key states to enable near-instant connections with search fronts. However, user guidance is not always an available option and can be time-consuming to provide. Random sampling has the largest variance of the three and the worst performance due to the inability to target key regions. Critical sampling strikes a favorable balance with lower variability and comparable performance with the hand-picked strategy.

Across all the strategies, the required number of expansions initially decreases with additional search fronts, but performance appears to degrade after some point. This can be primarily attributed to the increased overhead costs in maintaining many active searches. For instance, using 12 roots significantly increased the required planning time with the critical strategy. Numerous roots are unnecessary as the largest source of complexity was a doorway, and the hand-picked strategy confirms this. At 4-8 samples, critical sampling could reliably draw samples at the doorway, as highlighted in Fig. 4.3a.

To escape local minimas induced by the entire map, roots ideally would be placed near the south end of the kitchen island and between the doorway. We found that 8 samples had the best empirical performance in producing states belonging to these regions.

Comparison to WA*-Extend

With a success rate almost double that of WA*-Extend, it is clear how a multi-tree approach can be effective in search-based planning in the presence of local minima where connectivity between search fronts is low. While WA*-Extend is prone to being pulled into local minima region in the kitchen and timing out, MT-WA*-Extend uses multiple search efforts that can offer alternative paths. By reducing search fronts, we also see greater efficiency in commonly passed tests with respect to state expansions and planning time. By having roots

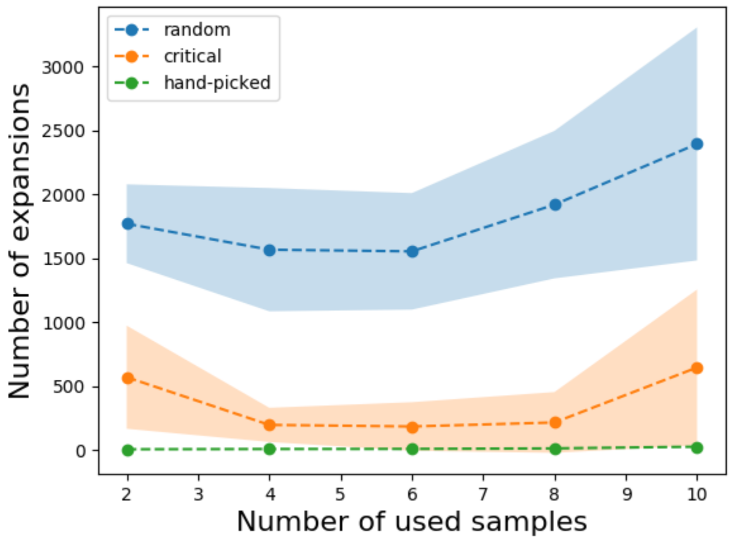


Figure 4.4: Performance of MT-WA* using varied number of additional starting roots and sampling strategies.

Table 4.1: Performance for Mobile Manipulation Planning with 8 Critical Roots

	MT-WA*-Extend	WA*-Extend	MT-RRT-Connect	RRT-Connect	PRM
Success (%)	92	48	100	100	100
Mean State Expansions	1.0	30.5	-	-	-
Mean Base Cost	1.0	0.83	1.32	1.43	1.42
Mean Arm Cost	1.0	0.74	1.76	1.71	1.62
Mean Total Plan Time	1.0	11.5	0.36	0.47	0.91
Mean Processed Base Cost	1.0	0.86	1.29	1.33	1.32
Mean Processed Arm Cost	1.0	0.70	1.74	1.65	1.51

initialized in central areas, connectivity in the configuration space is improved. Solution quality, however, is slightly lower in this case, given that searches are rooted in sampled configurations.

Comparison to Sampling-based searches

All sampling-based searches achieve a success rate of 100% largely owing to their robustness to deep local minima induced by the kitchen passageway. Comparable in that regard, MT-WA*-Extend offers much higher-quality paths before post-processing, since it is primarily a search-based approach with suboptimality bounds.

The multi-tree variant of RRT-Connect also outperforms the original version. This is due to the ability of MT-RRT-Connect to conduct search directly from a critical root, enabling much more frequent connectivity. The augmented PRM has slower planning times compared to its other sampling-based planners, as every grown vertex must consider additional critical roots as neighbors.

4.5 Summary

In this chapter, we present the natural extension to bidirectional search that provides heuristic search with robustness in the case that the heuristic fails to provide guidance. Multi-rooted searches split the search space, while the extend operator works to connect search efforts. Within a bidirectional framework, theoretical guarantees for bounded suboptimality and completeness are preserved. We also demonstrate how informed sampling techniques can be coupled with search-based planning to capitalize on the advantages of both philosophies. Simulated experimental results in the mobile manipulation domain demonstrate the efficacy of this approach in a difficult setting that would typically trap search-based planners.

Chapter 5

Conclusion

In this thesis, we consider the problem of speeding up search-based planning for single-query, high-dimensional motion planning. To address these challenges, we outline a framework that exploits the extend operator in the context of search-based planning. In Chapter 3, we use the extend operator in bidirectional heuristic search to greedily seek out connections between search fronts. We show that guarantees for completeness and bounded suboptimality are preserved in this setting. When evaluated for single-arm planning, the proposed algorithm produces high-quality solutions with great search efficiency, striking a successful balance. By efficiently reasoning through large regions of free space, our approach proves to be faster than other state-of-the-art bidirectional heuristic searches, while yielding solutions of lower cost and higher consistency compared to those of sampling-based strategies.

The bidirectional framework lends itself to a multi-tree implementation, which has been shown to improve connectivity in constrained and cluttered environments for similar sampling-based variants. In Chapter 4, we formalize multi-directional heuristic search for single-query motion planning with greater robustness to local minima and lesser dependence on the quality of the heuristic. We propose further improvements such as the adoption of shortcut successors to leverage other search efforts quickly and the use of critical states to serve as more informative search roots. When tested in a high-dimensional mobile manipulation domain for a challenging environment, our augmented approach sees a significantly higher success rate than its bidirectional counterpart, and found solutions are of better quality compared to those returned by sampling-based planners.

5.1 Future Work

This work establishes the use of an extend operator for search-based planning in the bidirectional and multi-tree settings, with the primary goal of capturing the milestone success of RRT-Connect for heuristic search. These extend operators may need to be adapted to account for the kinodynamic constraints of robots and to take advantage of redundant degrees-of-freedom of complicated robots. This leaves open the possibility of further research into more advanced operators and extension strategies.

In practice, collision checking often becomes the computational bottleneck in planning, motivating more efficient methods for extending to frontiers. These could entail increasing the number of extensions in each iteration or learning when an extension is likely based on environmental or graph-based features from other ongoing searches. Differing distance metrics in nearest-neighbor queries could also be used to exploit the planning domain or to diversify extension targets in other search frontiers.

The use of the extend operator in unidirectional search may also be worth further exploration, as predicting when bidirectional heuristic search will be the more effective method remains an active research problem [45].

All these directions for future work promise to make additional strides towards closing the performance gap between heuristic-based and sampling-based planners in high DoF motion planning.

Appendix A

Full Proofs for WA*-Extend

A.1 Pseudocode of WA*-Extend

The following pseudocode of the extend operator and WA*-Extend in Algorithms 7 and 8 slightly differs from that presented in the main text. Every state s maintains an additional variable $v(s)$ for each direction similar to $g(s)$. All v_{dir}-values are initially set to ∞ and then set to $g_{\text{dir}}(s)$ upon expansion of s in search *dir*. These modifications to the pseudocode simplify the proofs and do not affect the behavior or performance of WA*-Extend. For the following proofs, we follow the notation and assumptions stated in Section 3.3.1.

Algorithm 7 Extend Operator

```
procedure PRIORITY($s$, dir)
    return $g_{\text{dir}}(s) + \omega \cdot h_{\text{dir}}(s)$
procedure NEARESTNEIGHBOR($s_{\text{cur}}$, dir)
    return $\arg\min_{s \in \text{OPEN}_{\overline{\text{dir}}} \cup \text{CLOSED}_{\overline{\text{dir}}}} \triangle(s, s_{\text{cur}})$
procedure EXTEND($s$, $s_{\text{NN}}$, dir)
    EXTENDSTATUS ← *Trapped*
    if NEWSTATE($s$, $s_{\text{NN}}$, $s'$) then
        if $s' = s_{\text{NN}}$ then
            EXTENDSTATUS ← *Reached*
        else
            EXTENDSTATUS ← *Advanced*
        if $s' \notin \text{OPEN}_{\text{dir}} \cup \text{CLOSED}_{\text{dir}}$ then
            $g_{\text{dir}}(s') \leftarrow \infty$
            $v_{\text{dir}}(s') \leftarrow \infty$                    ▷ For proofs only
        if $g_{\text{dir}}(s') > g_{\text{dir}}(s) + c(s, s')$ then
            $g_{\text{dir}}(s') \leftarrow g_{\text{dir}}(s) + c(s, s')$
            if $s' \notin \text{CLOSED}_{\text{dir}}$ then
                Insert/Update $s'$ in $\text{OPEN}_{\text{dir}}$ with PRIORITY($s'$, dir)
            UPDATELEASTCOSTPATH($s'$, dir)
    return EXTENDSTATUS
procedure CONNECT($s$, dir)
    if $s \notin \text{CLOSED}_{\overline{\text{dir}}}$ then
        $s_{\text{NN}}$ ← NEARESTNEIGHBOR($s$, dir)
        repeat
            EXTENDSTATUS ← EXTEND($s$, $s_{\text{NN}}$, dir)
        until not EXTENDSTATUS = *Advanced*
```

Algorithm 8 Bidirectional WA* with Extend Operator

```
1:  procedure TerminationCriterion($u$)
2:      return $u \leq \max\left(\text{Open}_{\text{dir}}.\text{MinKey}(), \text{Open}_{\overline{\text{dir}}}.\text{MinKey}()\right)$
3:  procedure UpdateLeastCostPath($s, \text{dir}$)
4:      if $g_{\text{dir}}(s) + g_{\overline{\text{dir}}}(s) < u$ then
5:          $u \leftarrow g_{\text{dir}}(s) + g_{\overline{\text{dir}}}(s)$
6:          $s_c \leftarrow s$
7:  procedure Expand($s, \text{dir}$)
8:      Remove $s$ from $\text{Open}_{\text{dir}}$
9:      if $s \notin \text{Closed}_{\overline{\text{dir}}}$ then
10:         $v_{\text{dir}}(s) \leftarrow g_{\text{dir}}(s)$                                      ▷ For proofs only
11:         $\text{Closed}_{\text{dir}} \leftarrow \text{Closed}_{\text{dir}} \cup \{s\}$
12:         for $s' \in \text{Succ}_{\text{dir}}(s)$ do
13:             if $s' \notin \text{Open}_{\text{dir}} \cup \text{Closed}_{\text{dir}}$ then
14:                 $g_{\text{dir}}(s') \leftarrow \infty$
15:                 $v_{\text{dir}}(s') \leftarrow \infty$                                     ▷ For proofs only
16:             if $g_{\text{dir}}(s') > g_{\text{dir}}(s) + c(s, s')$ then
17:                 $g_{\text{dir}}(s') \leftarrow g_{\text{dir}}(s) + c(s, s')$
18:                 if $s' \notin \text{Closed}_{\text{dir}}$ then
19:                     Insert/Update $s'$ in $\text{Open}_{\text{dir}}$ with Priority($s', \text{dir}$)
20:                 UpdateLeastCostPath($s', \text{dir}$)
21: procedure main()
22:     $u \leftarrow \infty$
23:     $s_{\text{c}} \leftarrow \emptyset$
24:     $\text{Open}_f \leftarrow \emptyset, \text{Open}_b \leftarrow \emptyset$
25:     $\text{Closed}_f \leftarrow \emptyset, \text{Closed}_b \leftarrow \emptyset$
26:     $g_{\text{f}}(s_{\text{start}}) \leftarrow 0, g_{\text{f}}(s_{\text{goal}}) \leftarrow \infty$
27:     $g_{\text{b}}(s_{\text{start}}) \leftarrow \infty, g_{\text{b}}(s_{\text{goal}}) \leftarrow 0$
28:     $v_{\text{f}}(s_{start}) \leftarrow \infty, v_{\text{f}}(s_{goal}) \leftarrow \infty$                   ▷ For proofs only
29:     $v_{\text{b}}(s_{start}) \leftarrow \infty, v_{\text{b}}(s_{goal}) \leftarrow \infty$                   ▷ For proofs only
30:     Insert $s_{\text{start}}$ into $\text{Open}_f$ with Priority($s_{\text{start}}, \text{f}$)
31:     Insert $s_{\text{goal}}$ into $\text{Open}_b$ with Priority($s_{\text{goal}}, \text{b}$)
32:     $\text{dir} \leftarrow \text{f}$
33:     while not $\text{Open}_{\text{dir}}$.Empty() do
34:         $s_{\text{min}} \leftarrow \text{Open}_{\text{dir}}.\text{Top}()$
35:         if TerminationCriterion($u$) then
36:             return ExtractPath($s_{\text{c}}$)
37:         Expand($s_{\text{min}}, \text{dir}$)
38:         Connect($s_{\text{min}}, \text{dir}$)
39:         $\text{dir} \leftarrow \overline{\text{dir}}$
40:     return null
```

A.2 Proofs

In this section, we provide the full proofs for Theorems 1 and 2 introduced in the main text. The extend operator of WA*-Extend allows for additional successors during each iteration of the main search loop, unlike WA* without re-expansions. Therefore, we must first re-establish the correctness of g_{dir}-values. We use these results to prove lower-bound properties established by Theorems 1 and 2. Theorem 3 then relies on these properties in Section 3.3.4 to prove bounded suboptimality for WA*-Extend.

Lemma 2. *At any point of time and for any state s in search* dir*, we have $v_{\text{dir}}(s) \geq g_{\text{dir}}(s)$.*

Proof. The lemma clearly holds after initialization (before line 33, Alg. 8) since v_{dir}-values are infinite. Afterwards, g_{dir}-values can only decrease (line 16, Alg. 7 and line 17, Alg. 8). For any state s, $v_{\text{dir}}(s)$ is initialized to ∞ (line 14, Alg. 7 and line 15, Alg. 8) and only changes on line 10 in Alg. 8 when it is set to $g_{\text{dir}}(s)$. Thus, it is always true that $v_{\text{dir}}(s) \geq g_{\text{dir}}(s)$. □

Lemma 3. *On line 33 in Alg. 8 and for any search* dir*, v_{dir}- and g_{dir}- values are non-negative, $g_{\text{dir}}(s_{\text{start}}) = 0$, and $\forall\, s \neq s_{\text{start}}, g_{\text{dir}}(s) = \min\limits_{u \in \textsc{Pred}_{\text{dir}}(s)} (v_{\text{dir}}(u) + c(u, s))$.*

Proof. The lemma holds after initialization, where $g_{\text{dir}}(s_{\text{start}}) = 0$ and the remaining g_{dir}-values and all v_{dir}-values are infinite. The only places where g_{dir}- and v_{dir}- values are changed afterwards are on line 16 in Alg. 7 and lines 10 and 17 in Alg. 8. If $v_{\text{dir}}(s)$ changes in line 10, then it is decreased according to Lemma 2. Thus, it may only decrease the g_{dir}-values of its successors. The tests on line 15 in Alg. 7 and line 16 in Alg. 8 check this and update g_{dir}-values if necessary. Since all costs are non-negative and never change, $g_{\text{dir}}(s_{\text{start}})$ can never be changed: it will never pass the aforementioned tests and is therefore always 0. □

Theorem 6. *Suppose state s is selected for expansion and connection on line 34, Alg. 8 for any search* dir*. Then, the next time line 33, Alg. 8 is executed, $v_{\text{dir}}(s) = g_{\text{dir}}(s)$, where $g_{\text{dir}}(s)$ before and after the expansion of s remains the same.*

Proof. Suppose s is selected for expansion and connection in search *dir*. Then on line 10, Alg. 8, $v_{\text{dir}}(s) = g_{\text{dir}}(s)$, and it is the only place where v_{dir}-value changes. We therefore only need to show that $g_{\text{dir}}(s)$ does not change during these procedures. For expansion, $g_{\text{dir}}(s)$ could only change if $s \in \textsc{Succ}_{\text{dir}}(s)$ and $g_{\text{dir}}(s) > v_{\text{dir}}(s)+c(s, s)$. The second test, however, implies that $c(s, s) < 0$ since we have just set $v_{\text{dir}}(s) = g_{\text{dir}}(s)$. This contradicts our assumption that costs are non-negative. For connection to search $\overline{dir}$, $g_{\text{dir}}(s)$ similarly could only change if s is generated from extension function NewState and $g_{\text{dir}}(s) > v_{\text{dir}}(s) + c(s, s)$. The first condition cannot be met as NewState is assumed to take a positive ϵ-distance step from s to s_{NN}, as defined on line 23 in Alg. 7. The latter again contradicts our non-negative cost assumption. □

We will now prove Theorems 1 and 2, which help establish the bounded suboptimality property of WA*-Extend. Both these theorems refer to states corresponding to the minimum Priority-value or lower bound for search *dir*. Theorem 1 guarantees that the path costs to these states upon their expansion are no more than a factor of ω greater than the optimal cost. Theorem 2 provides similar bounds on the estimated path costs from respective s_{start} to s_{goal} through these states.

We expect the first lower-bound result to follow from WA* without re-expansions. We can intuitively view an $\textsc{Open}_{\text{dir}}$ in WA*-Extend as a superset of $\textsc{Open}$ in WA* without re-expansions, at any point during main search. This is because successors can be added from additional extension steps on line 18, Alg. 7 in addition to the typical expansion performed on line 19, Alg. 8. We show that adding these states to $\textsc{Open}_{\text{dir}}$ will never increase respective path lengths. We then extend this result to prove Theorem 2.

To formally analyze these results, we introduce Q_{dir}, the set of all states that have been generated but not yet expanded for search *dir*, as follows,

Definition 1. $\text{Q}_{\text{dir}} = \{u | v_{\text{dir}}(u) > g_{\text{dir}}(u) \wedge v_{\text{dir}}(u) > \omega g^*_{\text{dir}}(u)\}$

Theorem 7. *On line 33 in Alg. 8, let* Q_{dir} *be defined according to Definition 1. Then, for any state* $s = \arg\min_{u \in \text{Q}_{\text{dir}}} \textsc{Priority}(u, \text{dir})$, $g_{\text{dir}}(s) \leq \omega g^*_{\text{dir}}(s)$.

Proof. We assume $g^*_{\text{dir}}(s) < \infty$; otherwise, the theorem trivially holds. We prove by contradiction. Suppose there exists an s such that $\textsc{Priority}(s, \text{dir}) \leq \textsc{Priority}(u, \text{dir}) \forall u \in \text{Q}_{\text{dir}}$, but assume, instead, that $g_{\text{dir}}(s) > \omega g^*_{\text{dir}}(s)$. We also assume that $s \neq s_{\text{start}}$ since otherwise $g_{\text{dir}}(s) = 0 = g^*_{\text{dir}}(s)$ by Lemma 3. We will now show that there exists some $s_i \in \text{Q}_{\text{dir}}$ such that $\textsc{Priority}(s, \text{dir}) > \textsc{Priority}(s_i, \text{dir})$ for all settings, arriving at a contradiction. Consider the least-cost path from s_{start} to s, $\pi^*_{\text{dir}}(s_0 = s_{\text{start}}, \ldots, s_k = s)$. The cost of this path is $g^*_{\text{dir}}(s)$. Our assumption that $g_{\text{dir}}(s) > \omega g^*_{\text{dir}}(s)$ means that there exists at least one $s_i \in \pi^*_{\text{dir}}(s_0, \ldots, s_{k-1})$ whose $v_{\text{dir}}(s_i) > \omega g^*_{\text{dir}}(s_i)$. Otherwise,

$$\begin{aligned}
g_{\text{dir}}(s) = g_{\text{dir}}(s_k) &= \min_{s' \in \textsc{Pred}_{\text{dir}}(s_k)} (v_{\text{dir}}(s) + c(s', s_k)) \\
&\leq v_{\text{dir}}(s_{k-1}) + c(s_{k-1}, s_k) \\
&\leq \omega g^*_{\text{dir}}(s_{k-1}) + c(s_{k-1}, s_k) \\
&\leq \omega (g^*_{\text{dir}}(s_{k-1}) + c(s_{k-1}, s_k)) \\
&\leq \omega g^*_{\text{dir}}(s_k)
\end{aligned}$$

Let us now consider $s_i \in \pi^*$ with smallest index $i \geq 0$ (that is, closest to s_{start}) such that $v_{\text{dir}}(s_i) > \omega g_{\text{dir}}(s_i)$. We will now show that $s_i \in \text{Q}_{\text{dir}}$. If $i = 0$, then $g_{\text{dir}}(s_i) = g_{\text{dir}}(s_{\text{start}}) = 0$ and $v_{\text{dir}}(s_{\text{start}})$ is non-negative, as initialized on lines 28-29, Alg. 8. Thus, $v_{\text{dir}}(s_i) > \omega g^*_{\text{dir}}(s_i) = 0 = g_{\text{dir}}(s_i)$, and $s_i \in \text{Q}_{\text{dir}}$.

We will now show that $g_{\text{dir}}(s_i) \leq \omega g^*_{\text{dir}}(s_i)$. For $i > 0$, we have $v_{\text{dir}}(s_i) > \omega g^*_{\text{dir}}(s_i)$ and by the choice of s_i,

$$\begin{aligned}
g_{\text{dir}}(s_i) &= \min_{s' \in \textsc{Pred}_{\text{dir}}(s_i)} (v_{\text{dir}}(s') + c(s', s_i)) \\
&\leq v_{\text{dir}}(s_{i-1}) + c(s_{i-1}, s_i) \\
&\leq \omega g^*_{\text{dir}}(s_{i-1}) + c(s_{i-1}, s_i) \\
&\leq \omega g^*_{\text{dir}}(s_{i-1}) + c(s_{i-1}, s_i) \\
&\leq \omega (g^*_{\text{dir}}(s_{i-1}) + c(s_{i-1}, s_i)) \\
&\leq \omega g^*_{\text{dir}}(s_i)
\end{aligned}$$

Thus, $v_{\text{dir}}(s_i) > \omega g^*_{\text{dir}}(s_i) \geq g_{\text{dir}}(s_i)$, implying $s_i \in \text{Q}_{\text{dir}}$. Using this result, we will show $\textsc{Priority}(s, \text{dir}) > \textsc{Priority}(s_i, \text{dir})$ and finally arrive at a contradiction. According to our assumptions,

$$\begin{aligned}
\textsc{Priority}(s, \text{dir}) &= g_{\text{dir}}(s) + \omega h_{\text{dir}}(s) \\
&> \omega g^*_{\text{dir}}(s) + \omega h_{\text{dir}}(s) \\
&> \omega(g^*_{\text{dir}}(s_i) + c^*(s_i, s) + h_{\text{dir}}(s)) \\
&> \omega g^*_{\text{dir}}(s_i) + \omega h_{\text{dir}}(s) && h_{\text{dir}} \text{ is consistent} \\
&> g^*_{\text{dir}}(s) + \omega h_{\text{dir}}(s) \\
&> \textsc{Priority}(s_i, \text{dir})
\end{aligned}$$

Now, as $s_i \in \text{Q}_{\text{dir}}$ and $\textsc{Priority}(s_i, \text{dir}) < \textsc{Priority}(s, \text{dir})$, we encounter a contradiction to our assumption that $\textsc{Priority}(s, \text{dir}) \leq \textsc{Priority}(u, \text{dir})\ \forall\, u \in \text{Q}_{\text{dir}}$. □

Theorem 8. *On line 33 in Alg. 8, let* Q_{dir} *be defined according to Definition 1. Then* $\text{Q}_{\text{dir}} \subseteq \textsc{Open}_{\text{dir}}$.

Proof. We prove this by induction. At the very start, $\textsc{Open}_{\text{dir}}$ contains its respective starting state s_{start}, $v_{\text{dir}}(s_{\text{start}}) = \infty$, and $g_{\text{dir}}(s_{\text{start}}) = 0$. Thus, $s_{\text{start}} \in \text{Q}_{\text{dir}}$, and clearly $s_{\text{start}} \in \textsc{Open}_{\text{dir}}$. Also, for all other states s', $g_{\text{dir}}(s') = v_{\text{dir}}(s') = \infty$; thus, the statement holds.

Now, we consider the states that are part of $\textsc{Closed}_{\text{dir}}$. $\textsc{Closed}_{\text{dir}}$ holds all the states that are expanded in a given search *dir* on line 37, Alg. 8 (insertion in line 11, Alg. 8). Such a state s is removed from $\textsc{Open}_{\text{dir}}$ on line 8, Alg. 8 (before it is inserted in $\textsc{Closed}_{\text{dir}}$). It follows that $s \in \textsc{Closed}_{\text{dir}}$ implies $s \neq \textsc{Open}_{\text{dir}}$, as a state s expanded in search *dir* will never be put back into its $\textsc{Open}$ list due to the checks on line 18, Alg. 8 and line 17, Alg. 7.

Next, we show that the statement denoted below holds for the first time line 33, Alg. 8 is executed,

$$v_{\text{dir}}(s) \leq \omega g^*_{\text{dir}}(s)\ \forall\, s \in \textsc{Closed}_{\text{dir}} \tag{$*$}$$

This is obviously true as before the first execution of line 33, Alg. 8, no state is expanded in search *dir* and thus $\textsc{Closed}_{\text{dir}}$ is empty. We will now show by induction that the statement and the theorem continue to hold for subsequent executions of line 33, Alg. 8. Suppose the theorem and statement $(*)$ hold during all previous executions of line 33, Alg. 8. We need to show that the theorem holds the next time line 33, Alg. 8 is executed.

We first prove that the statement $(*)$ still holds during the next execution of line 33, Alg. 8. Considering the current iteration of the main search loop, we observe the following possibilities: either the search terminates, in which case the theorem is proved trivially (as there will be no more calls to line 33, Alg. 8) or state s will undergo expansion in search *dir* and connection to search $\overline{dir}$.

When state s is selected for expansion and connection, it has the minimum $\textsc{Priority}$-value such that $s = \arg\min_{u \in \textsc{Open}_{\text{dir}}} \textsc{Priority}(u, \text{dir})$. According to our induction hypothesis, $\text{Q}_{\text{dir}} \subseteq \textsc{Open}_{\text{dir}}$, selection of s therefore guarantees $g_{\text{dir}}(s) \leq \omega g^*_{\text{dir}}(s)$ by Theorem 7. From Theorem

6, it then also follows that the next time line 33, Alg. 8 is executed, $v_{\text{dir}}(s) \leq \omega g^*_{\text{dir}}(s)$, and hence statement $(*)$ still holds.

We now prove that after s is expanded in *dir* and an attempt to connect to $\overline{dir}$ is made, the theorem itself also holds. We prove this by showing that $\text{Q}_{\text{dir}} \subseteq \textsc{Open}_{\text{dir}}$ the next time line 33, Alg. 8 is executed.

Any state s that is generated for the first time from Expand or Connect will be initialized with $v_{\text{dir}}(s) = g_{\text{dir}}(s) = \infty$. Now, if the g_{dir}-value is lowered during expansion, then $v_{\text{dir}}(s) = \infty > g_{\text{dir}}(s)$. Similarly, for any other state s', if the g_{dir}-value is lowered, then $v_{\text{dir}}(s') > g_{\text{dir}}(s')$, as either $v_{\text{dir}}(s') = \infty$ or $v_{\text{dir}}(s')$ holds the value $g_{\text{dir}}(s')$ when s' was last expanded.

Every such state will be inserted/updated in $\textsc{Open}_{\text{dir}}$ on line 19, Alg. 8 for expansion and on line 18, Alg. 7 for connection. The only exception is if either of the prior checks on line 18, Alg. 8 or line 17, Alg. 7 are satisfied. In both cases, s has been expanded earlier in search *dir*, in which case $s \in \textsc{Closed}_{\text{dir}}$, and thus $v_{\text{dir}}(s) \leq \omega g^*_{\text{dir}}(s)$ from statement $(*)$.

All states that have $v_{\text{dir}}(s) > g_{\text{dir}}(s)$, are either in $\textsc{Open}_{\text{dir}}$ or $\textsc{Closed}_{\text{dir}}$. From statement $(*)$, any state $s \in \textsc{Closed}_{\text{dir}}$ satisfies $v_{\text{dir}}(s) \leq \omega g^*_{\text{dir}}(s)$. Therefore, for any state s such that $v_{\text{dir}}(s) > g_{\text{dir}}(s)$ and $v_{\text{dir}}(s) > \omega g^*_{\text{dir}}(s)$, it follows that $s \in \textsc{Open}_{\text{dir}}$. Thus, $\text{Q}_{\text{dir}} \subseteq \textsc{Open}_{\text{dir}}$. □

Theorem 1. *On expansion for any search* dir *and state* $s = \arg\min_{v \in \textsc{Open}_{\text{dir}}} \textsc{Priority}(v, \text{dir})$, *it holds that* $g_{\text{dir}}(s) \leq \omega g^*_{\text{dir}}(s)$.

Proof. From Theorem 8, $\text{Q}_{\text{dir}} \subseteq \textsc{Open}_{\text{dir}}$, so any state s that satisfies $\textsc{Priority}(s, \text{dir}) \leq \textsc{Priority}(u, \text{dir})$ for all $u \in \text{Q}_{\text{dir}}$ also satisfies $\textsc{Priority}(s, \text{dir}) \leq \textsc{Priority}(u, \text{dir})$ for all $u \in \textsc{Open}_{\text{dir}}$. From Theorem 7, it follows that $g_{\text{dir}}(s) \leq \omega g^*_{\text{dir}}(s)$. □

We recall the following statement proven in Section 3.3.4 of the main text.

Lemma 1. *Suppose state s is selected for expansion and connection at line 30, Alg. 3. If* $s \in \textsc{Closed}_{\overline{\text{dir}}}$, *the main search terminates.*

Lemma 4. *When search for a given direction* dir *selects its respective goal state* s_{goal} *for expansion and connection on line 34 in Alg. 8, the main search terminates.*

Proof. We assume $s_{\text{start}} \neq s_{\text{goal}}$ and search does not terminate after a single expansion. Then, s_{goal} is guaranteed to be expanded since goal states are initially added to respective Open lists (lines 31-32, Alg. 8) and search direction changes on line 33, Alg. 8. The lemma then holds as a direct result of Lemma 1. □

Theorem 2. *On expansion for any search* dir *and state* $s = \arg\min_{u \in \textsc{Open}_{\text{dir}}} \textsc{Priority}(u, \text{dir})$, *it holds that* $\textsc{Priority}(s, \text{dir}) \leq \omega g^*_{\text{dir}}(s_{\text{goal}})$.

Proof. We assume $g^*_{\text{dir}}(s_{\text{goal}}) < \infty$; otherwise, the theorem trivially holds. Let $\widetilde{\mathcal{G}}$ denote the subgraph of $\mathcal{G}$, where extensions between nodes of opposite search efforts are not allowed. We will first prove this theorem holds in this setting and use the result to generalize to $\mathcal{G}$.

We prove by contradiction. Suppose there exists an s such that $\textsc{Priority}(s, \text{dir}) \leq \textsc{Priority}(u, \text{dir})$ for all $u \in \textsc{Open}_{\text{dir}}$, but assume, instead, that $\textsc{Priority}(s, \text{dir}) > \omega g^*_{\text{dir}}(s_{\text{goal}})$. In this case, we will show that there exists some $s_i \in \textsc{Open}_{\text{dir}}$ such that $\textsc{Priority}(s_i, \text{dir}) < \textsc{Priority}(s, \text{dir})$, arriving at a contradiction.

For $\widetilde{\mathcal{G}}$, consider the least-cost path $\widetilde{\pi}^*(s_0 = s_{\text{start}}, \ldots s_k = s_{\text{goal}})$. From this path, we choose the state s_i to be the shallowest state in $\textsc{Open}_{\text{dir}}$ that has yet to be expanded by search dir. Such a state is guaranteed to exist because a) $\textsc{Open}_{\text{dir}}$ is initialized with s_{start}; b) when any $s_j \in \widetilde{\pi}^*$ undergoes expansion, its successor s_{j+1} is added to $\textsc{Open}_{dir}$; c) any $s_j \in (\textsc{Closed}_{\overline{\text{dir}}} \cap \widetilde{\pi}^*)$ is never expanded, according to Lemma 1; and d) $s_k = s_{\text{goal}}$ is never expanded as s_{goal} always satisfies the termination criterion and ends the main search under Lemma 4.

We will now show that for such s_i, it holds that $g_{\text{dir}}(s_i) \leq \omega g^*_{\text{dir}}(s_i)$. For i = 0, we have $g_{\text{dir}}(s_0) = g_{\text{dir}}(s_{\text{start}}) = 0 \leq \omega g^*_{\text{dir}}(s_0) = 0$. If $i \neq 0$, by our choice of s_i, we know that $s_{i-1} \in \widetilde{\pi}^*$ has already been expanded. Then, $g_{\text{dir}}(s_{i-1}) \leq \omega g^*_{\text{dir}}(s_{i-1})$ by Theorem 1. We have,

$$\begin{aligned} g_{\text{dir}}(s_i) &\leq g_{\text{dir}}(s_{i-1}) + c(s_{i-1}, s_i) && \text{(line 16, Alg. 8)} \\ &\leq \omega g^*_{\text{dir}}(s_{i-1}) + c(s_{i-1}, s_i) \\ &\leq \omega(g^*_{\text{dir}}(s_{i-1}) + c(s_{i-1}, s_i)) \\ &\leq \omega g^*_{\text{dir}}(s_i) \end{aligned}$$

We can now use this result to obtain,

$$\begin{aligned} \textsc{Priority}(s_i, \text{dir}) &= g_{\text{dir}}(s_i) + \omega h_{\text{dir}}(s_i) \\ &\leq \omega g^*_{\text{dir}}(s_i) + \omega h_{\text{dir}}(s_i) \\ &\leq \omega(g^*_{\text{dir}}(s_i) + h_{\text{dir}}(s_i)) \\ &\leq \omega(g^*_{\text{dir}}(s_i) + c^*(s_i, s_{\text{goal}}) && h_{\text{dir}} \text{ is admissible} \\ &\leq \omega g^*_{\text{dir}}(s_{\text{goal}}) \end{aligned}$$

As $s_i \in \textsc{Open}_{\text{dir}}$ and $\textsc{Priority}(s_i, \text{dir}) \leq \omega g^*_{\text{dir}}(s_{\text{goal}}) < \textsc{Priority}(s, \text{dir})$, state s no longer has the minimum $\textsc{Priority}$-value in $\textsc{Open}_{\text{dir}}$, arriving at a contradiction. Thus, the assumption that $\textsc{Priority}(s, \text{dir}) > \omega g^*_{\text{dir}}(s_{\text{goal}})$ must be incorrect, proving the theorem in the case of no extensions.

Now, for $\mathcal{G}$, where extensions are possible, the proof follows in an identical manner with one key difference. Consider the least-cost path $\pi^*(s_0 = s_{\text{start}}, \ldots, s_c, \ldots, s_k = s_{\text{goal}})$ and denote subpaths corresponding to each search as $\pi^*_{dir}(s_0, \ldots, s_c)$ and $\pi^*_{\overline{dir}}(s_c, \ldots, s_k)$. The difference between the proof for $\mathcal{G}$ and subgraph $\widetilde{\mathcal{G}}$ is that it is now possible for search dir to have expanded all states $s \in \pi^*_{dir}$, including s_c if s_c has yet to be realized by search $\overline{dir}$. However, this case then reduces to planning over subgraph $\widetilde{G}$, which we have already proven for. $\square$

Bibliography

[1] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[2] Ira Pohl. First results on the effect of error in heuristic search. *Machine Intelligence*, 5:219–236, 1970.

[3] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. Ara: formal analysis. 2003.

[4] Allen Cheng, Dhruv Saxena, and Maxim Likhachev. Bidirectional heuristic search for motion planning with an extend operator. In *Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[5] T Alastair J Nicholson. Finding the shortest route between two points in a network. *The computer journal*, 9(3):275–280, 1966.

[6] James J Kuffner Jr and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *ICRA*, volume 2, 2000.

[7] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[8] Sebastian Klemm, Jan Oberländer, Andreas Hermann, Arne Roennau, Thomas Schamm, J Marius Zollner, and Rüdiger Dillmann. Rrt∗-connect: Faster, asymptotically optimal motion planning. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1670–1677. IEEE, 2015.

[9] Matthew Jordan and Alejandro Perez. Optimal bidirectional rapidly-exploring random trees. 2013.

[10] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[11] Ira Pohl. Bidirectional and heuristic search in path problems. Technical report, 1969.

[12] James BH Kwa. Bs∗: An admissible bidirectional staged heuristic search algorithm. *Artificial Intelligence*, 38(1):95–109, 1989.

[13] Dennis de Champeaux and Lenie Sint. An improved bidirectional heuristic search algorithm. *J. ACM*, 24(2):177–191, 1977.

[14] Robert C Holte, Ariel Felner, Guni Sharon, and Nathan R Sturtevant. Bidirectional search that is guaranteed to meet in the middle. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[15] Fahad Islam, Venkatraman Narayanan, and Maxim Likhachev. A*-connect: Bounded suboptimal bidirectional heuristic search. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2752–2758. IEEE, 2016.

[16] Sandip Aine, Siddharth Swaminathan, Venkatraman Narayanan, Victor Hwang, and Maxim Likhachev. Multi-heuristic a. *The International Journal of Robotics Research*, 35(1-3):224–243, 2016.

[17] Venkatraman Narayanan, Sandip Aine, and Maxim Likhachev. Improved multi-heuristic a* for searching with uncalibrated heuristics. In *Eighth Annual Symposium on Combinatorial Search*, 2015.

[18] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.

[19] Dustin J Webb and Jur van den Berg. Kinodynamic rrt*: Optimal motion planning for systems with linear differential constraints. *arXiv preprint arXiv:1205.5088*, 2012.

[20] Kwangjin Yang, Sangwoo Moon, Seunghoon Yoo, Jaehyeon Kang, Nakju Lett Doh, Hong Bong Kim, and Sanghyun Joo. Spline-based rrt path planner for non-holonomic robots. *Journal of Intelligent & Robotic Systems*, 73(1-4):763–782, 2014.

[21] Benjamin J Cohen, Gokul Subramania, Sachin Chitta, and Maxim Likhachev. Planning for manipulation with adaptive motion primitives. In *2011 IEEE International Conference on Robotics and Automation*, pages 5478–5485. IEEE, 2011.

[22] Andreas L Koll and Hermann Kaindl. Bidirectional best-first search with bounded error: Summary of results. In *Proceedings of the 13th international joint conference on Artifical intelligence-Volume 1*, pages 217–223. Morgan Kaufmann Publishers Inc., 1993.

[23] Blai Bonet, Gábor Loerincs, and Héctor Geffner. A robust and fast action selection mechanism for planning. In *AAAI/IAAI*, pages 714–719, 1997.

[24] Benjamin J Cohen, Sachin Chitta, and Maxim Likhachev. Search-based planning for manipulation with motion primitives. In *2010 IEEE International Conference on Robotics and Automation*, pages 2902–2908. IEEE, 2010.

[25] Jose Luis Blanco and Pranjal Kumar Rai. nanoflann: a c++ header-only fork of flann, a library for nearest neighbor (nn) wih kd-trees, 2014.

[26] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.

[27] Hermann Kaindl and Gerhard Kainz. Bidirectional heuristic search reconsidered. *Journal of Artificial Intelligence Research*, 7:283–317, 1997.

[28] Joseph K Barker and Richard E Korf. Limitations of front-to-end bidirectional heuristic search. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[29] Matthew Clifton, Gavin Paul, Ngai Kwok, Dikai Liu, and Da-Long Wang. Evaluating performance of multiple rrts. In *2008 IEEE/ASME International Conference on Mechtronic and Embedded Systems and Applications*, pages 564–569. IEEE, 2008.

[30] PC Nelson and AA Toptsis. Superlinear speedup using bidirectionalism and islands. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)-Workshop on Parallel Processing in AI, Sydney, Australia*, pages 129–134, 1991.

[31] CP Nelson. Parallel bidirectional search using multidimensional heuristics. 1988.

[32] Peter C Nelson and Anestis A Toptsis. Wave-shaping in multiprocessor bidirectional heuristic state space search. In *Portuguese Conference on Artificial Intelligence*, pages 92–104. Springer, 1991.

[33] Anestis A Toptsis and Rahul A Chaturvedi. Voronoi-assisted parallel bidirectional heuristic search. In *International Conference on Future Generation Communication and Networking*, pages 73–84. Springer, 2008.

[34] Tsai-Yen Li and Yang-Chuan Shie. An incremental learning approach to motion planning with roadmap management. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 4, pages 3411–3416. IEEE, 2002.

[35] Matt Zucker, James Kuffner, and Michael Branicky. Multipartite rrts for rapid replanning in dynamic environments. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1603–1609. IEEE, 2007.

[36] Russell Gayle, Kristopher R Klingler, and Patrick G Xavier. Lazy reconfiguration forest (lrf)-an approach for motion planning with multiple tasks in dynamic environments. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1316–1323. IEEE, 2007.

[37] Morten Strandberg. Augmenting rrt-planners with local trees. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 4, pages 3258–3262. IEEE, 2004.

[38] Wei Wang, Xin Xu, Yan Li, Jinze Song, and Hangen He. Triple rrts: an effective method for path planning in narrow passages. *Advanced Robotics*, 24(7):943–962, 2010.

[39] Wei Wang, Yan Li, Xin Xu, and Simon X Yang. An adaptive roadmap guided multi-rrts strategy for single query path planning. In *2010 IEEE International Conference on Robotics and Automation*, pages 2871–2876. IEEE, 2010.

[40] Mike Phillips, Benjamin J Cohen, Sachin Chitta, and Maxim Likhachev. E-graphs: Bootstrapping planning with experience graphs. In *Robotics: Science and Systems*, volume 5, page 110, 2012.

[41] Brian Ichter, Edward Schmerling, Tsang-Wei Edward Lee, and Aleksandra Faust. Learned critical probabilistic roadmaps for robotic motion planning. *arXiv preprint arXiv:1910.03701*, 2019.

[42] Shivam Vats, Venkatraman Narayanan, and Maxim Likhachev. Learning to avoid local minima in planning for static environments. 2017.

[43] Linton C Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977.

[44] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[45] Nathan R Sturtevant, Shahaf Shperberg, Ariel Felner, and Jingwei Chen. Predicting the effectiveness of bidirectional heuristic search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 281–290, 2020.