An Asymptotically-Optimal Sampling-Based Algorithm for Bi-directional Motion Planning

Joseph A. Starek*, Javier V. Gomez†, Edward Schmerling‡, Lucas Janson§, Luis Moreno†, Marco Pavone*

***Abstract*— Bi-directional search is a widely used strategy to increase the success and convergence rates of sampling-based motion planning algorithms. Yet, few results are available that merge both bi-directional search and asymptotic optimality into existing optimal planners, such as PRM*, RRT*, and FMT*. The objective of this paper is to fill this gap. Specifically, this paper presents a bi-directional, sampling-based, asymptotically-optimal algorithm named Bi-directional FMT* (BFMT*) that extends the Fast Marching Tree (FMT*) algorithm to bi-directional search while preserving its key properties, chiefly lazy search and asymptotic optimality through convergence in probability. BFMT* performs a two-source, *lazy* dynamic programming recursion over a set of randomly-drawn samples, correspondingly generating two search trees: one in cost-to-come space from the initial configuration and another in cost-to-go space from the goal configuration. Numerical experiments illustrate the advantages of BFMT* over its unidirectional counterpart, as well as a number of other state-of-the-art planners.**

I Introduction

Motion planning is the computation of paths that guide systems from an initial configuration to a set of goal configuration (s) around nearby obstacles, while possibly optimizing an objective function. The problem has a long and rich history in the field of robotics, and many algorithmic tools have been developed; we refer the interested reader to [1] and references therein. Arguably, *sampling-based algorithms* are among the most pervasive, widespread planners available in robotics, including the Probabilistic Roadmap algorithm (PRM) [2], the Expansive Space Trees algorithm (EST) [3], [4], and the Rapidly-Exploring Random Tree algorithm (RRT) [5]. Since their development, efforts to improve the "quality" of paths led to asymptotically-optimal (AO) variants of RRT and PRM, named RRT* and PRM*, respectively, whereby the cost of the returned solution converges almost surely to the optimum as the number of samples approaches infinity [6], [7]. Many other planners followed, including BIT* [8] and RRT# [9] to name a few. Recently, a conceptually different asymptotically-optimal, sampling-based motion planning algorithm, called the Fast Marching Tree (FMT*) algorithm, has been presented in [10], [11]. Numerical experiments suggested that FMT* converges to an optimal solution faster than PRM* or RRT*, especially in high-dimensional configuration spaces and in scenarios where collision-checking is expensive.

*Dept. of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305, {jstarek, pavone}@stanford.edu.

†Dept. of Systems Engineering & Automation, Carlos III University of Madrid, Madrid, Spain, 28911, {jvgomez, moreno}@ing.uc3m.es.

‡Inst. for Computational & Mathematical Engineering, Stanford University, Stanford, CA 94305, schmrlng@stanford.edu.

§Dept. of Statistics, Stanford University, Stanford, CA 94305, ljanson@stanford.edu.

This work was supported by NASA under the Space Technology Research Grants Program, Grant NNX12AQ43G.

It is a well-known fact that *bi-directional* search can dramatically increase the convergence rate of planning algorithms, prompting some authors [12] to advocate its use for accelerating essentially any motion planning query. This was first rigorously studied in [13] and later investigated, for example, in [14], [15]. Collectively, the algorithms presented in [12]–[15] belong to the family of *non*-sampling-based approaches and are more or less closely related to a bi-directional implementation of the Dijkstra Method. More recently, and not surprisingly in light of these performance gains, bi-directional search has been merged with the sampling-based approach, with RRT-Connect and SBL representing the most notable examples [16], [17].

Though such bi-directional versions of RRT and PRM are probabilistically complete, they do not enjoy optimality guarantees. The next logical step in the quest for fast planning algorithms is the design of *bi-directional*, sampling-based, asymptotically-optimal algorithms. To the best of our knowledge, the only available results in this context are [18] and the unpublished work [19], both of which discuss bi-directional implementations of RRT*. Neither work, however, provides a mathematically-rigorous proof of asymptotic optimality starting from first principles. Accordingly, the objective of this paper is to propose and *rigorously* analyze such an algorithm.

Statement of Contributions: This paper introduces the Bi-directional Fast Marching Tree (BFMT*) algorithm.[1] To the best of the authors' knowledge, this is the first tree-based, asymptotically-optimal bi-directional sampling-based planner. BFMT* extends FMT* to bi-directional search and essentially performs a "lazy," bi-directional dynamic programming recursion over a set of probabilistically-drawn samples in the free configuration space. The contribution of this paper is threefold. First, we present the BFMT* algorithm in Section III. Second, we rigorously prove the asymptotic optimality of BFMT* (under the notion of convergence in probability) and characterize its convergence rate in Section IV. We note that the convergence rate of FMT* in [11] is proved only for obstacle-free configuration spaces, while we generalize that result to allow for the presence of obstacles. Finally, we perform numerical experiments in Section V across a number of planning spaces that suggest BFMT* converges to an *optimal* solution at least as fast as FMT*, PRM*, and RRT*, and sometimes significantly faster.

II Problem Definition

Let $\mathcal{X}$ be a d-dimensional configuration space, and let $\mathcal{X}_{\text{obs}}$ be the obstacle region, such that $\mathcal{X} \setminus \mathcal{X}_{\text{obs}}$ is an

[1]The asterisk *, pronounced "star", is intended to represent asymptotic optimality much like for the RRT* and PRM* algorithms.

open set (we consider $\partial\mathcal{X} \subset \mathcal{X}_{\text{obs}}$). Denote the obstacle-free space as $\mathcal{X}_{\text{free}} = \text{cl}(\mathcal{X} \setminus \mathcal{X}_{\text{obs}})$, where $\text{cl}(\cdot)$ denotes the closure of a set. A path planning problem, denoted by a triplet $(\mathcal{X}_{\text{free}}, \mathbf{x}_{\text{init}}, \mathbf{x}_{\text{goal}})$, seeks to maneuver from an initial configuration $\mathbf{x}_{\text{init}}$ to a goal configuration $\mathbf{x}_{\text{goal}}$ through $\mathcal{X}_{\text{free}}$. Let a continuous function of *bounded variation* $\sigma : [0, 1] \to \mathcal{X}$, called a *path*, be *collision-free* if $\sigma(\tau) \in \mathcal{X}_{\text{free}}$ for all $\tau \in [0, 1]$. A path is called a *feasible* solution to the planning problem $(\mathcal{X}_{\text{free}}, \mathbf{x}_{\text{init}}, \mathbf{x}_{\text{goal}})$ if it is collision-free, $\sigma(0) = \mathbf{x}_{\text{init}}$, and $\sigma(1) = \mathbf{x}_{\text{goal}}$.

Let Σ be the set of all paths. A cost function for the planning problem $(\mathcal{X}_{\text{free}}, \mathbf{x}_{\text{init}}, \mathbf{x}_{\text{goal}})$ is a function $J : \Sigma \to \mathbb{R}_{\geq 0}$ from Σ to the nonnegative real numbers; in this paper, we consider as $J(\sigma)$ the *arc length* of σ with respect to the Euclidean metric in $\mathcal{X}$ (the extension to general cost functions will be briefly discussed in Section IV-C).

> **Optimal path planning problem**: Given a path planning problem $(\mathcal{X}_{\text{free}}, \mathbf{x}_{\text{init}}, \mathbf{x}_{\text{goal}})$ and an arc length function $J : \Sigma \to \mathbb{R}_{\geq 0}$, find a feasible path σ^* such that $J(\sigma^*) = \min\{J(\sigma) \mid \sigma \text{ is feasible}\}$. If no such path exists, report failure.

Finally, we introduce some definitions concerning the *clearance* of a path, *i.e.*, its "distance" from $\mathcal{X}_{\text{obs}}$ [11]. For a given $\delta > 0$, the δ-interior of $\mathcal{X}_{\text{free}}$ is defined as the set of all points that are at least a distance δ away from any point in $\mathcal{X}_{\text{obs}}$. A collision-free path σ is said to have strong δ-clearance if it lies entirely inside the δ-interior of $\mathcal{X}_{\text{free}}$. A path planning problem with optimal path cost J^* is called *δ-robustly feasible* if there exists a strictly positive sequence $\delta_n \to 0$, with $\delta_n \leq \delta \ \forall n \in \mathbb{N}$, and a sequence $\{\sigma_n\}_{n=1}^{\infty}$ of feasible paths such that $\lim_{n\to\infty} J(\sigma_n) = J^*$ and for all $n \in \mathbb{N}$, σ_n has strong δ_n-clearance, $\sigma_n(1) = \mathbf{x}_{\text{goal}}$, $\sigma_n(\tau) \neq \mathbf{x}_{\text{goal}}$ for all $\tau \in (0, 1)$, and $\sigma_n(0) = \mathbf{x}_{\text{init}}$.

III THE BFMT* ALGORITHM

In this section, we present the Bi-Directional Fast Marching Tree algorithm, BFMT*, represented in pseudocode as Algorithm 1. To begin, we provide a high-level description of FMT* in Section III-A, on which BFMT* is based. We follow in Section III-B with BFMT*'s own high-level description, and then provide additional details in Section III-C.

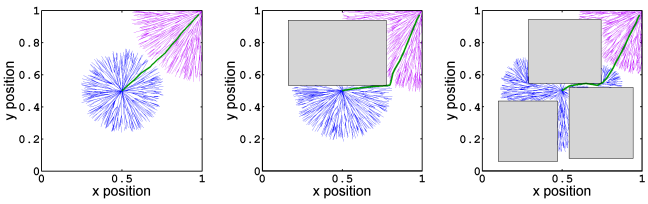
III-A FMT – High-level description*

The FMT* algorithm, introduced in [10], [11], is a unidirectional algorithm that essentially performs a forward dynamic programming recursion over a set of sampled points and correspondingly generates a *tree of paths* that grow steadily outward in cost-to-come space. The recursion performed by FMT* is characterized by three key features: (1) It is tailored to disk-connected graphs, where two samples are considered *neighbors* (hence connectable) if their distance is below a given bound, referred to as the *connection radius*; (2) It performs graph construction and graph search *concurrently*; and (3) For the evaluation of the immediate cost in the dynamic programming recursion, one "lazily" ignores the presence of obstacles, and whenever a locally-optimal (assuming no obstacles) connection to a new sample intersects an obstacle, that sample is simply skipped and left for later (as opposed to looking for other locally-optimal connections in the neighborhood).

The last feature, which makes the algorithm "lazy," may cause *suboptimal* connections. A central property of FMT* is that the cases where a suboptimal connection is made become vanishingly rare as the number of samples goes to infinity, which helps maintain the algorithm's asymptotically optimality. This manifests itself into a key computational advantage—by restricting collision detection to only locally-optimal connections, FMT* (as opposed to, *e.g.*, PRM* [6]) avoids a large number of costly collision-check computations, at the price of a vanishingly small "degree" of suboptimality. We refer the reader to [10], [11] for a detailed description of the algorithm and its advantages.

III-B BFMT – High-level description*

At its core, BFMT* implements a *bi-directional* version of the FMT* algorithm by simultaneously propagating two wavefronts (henceforth, the leaves of an expanding tree will be referred to as the wavefront of the tree) through the free configuration space. BFMT*, therefore, performs a *two-source* dynamic programming recursion over a set of sampled points, and correspondingly generates a *pair* of search trees: one in cost-to-come space from the initial configuration and another in cost-to-go space from the goal configuration (see Fig. 1). Throughout the remainder of the paper, we refer to the former as the *forward tree*, and to the latter as the *backward tree*.



(a) 0% Coverage (b) 25% Coverage (c) 50% Coverage

Fig. 1: The BFMT* algorithm generates a *pair* of search trees: one in cost-to-come space from the initial configuration (blue) and another in cost-to-go space from the goal configuration (purple). The path found by the algorithm is in green color.

The dynamic programming recursion performed by BFMT* is characterized by the same lazy feature of FMT* (see Section III-A). However, the time it takes to run BFMT* on a given number of samples can be substantially smaller than for FMT*. Indeed, for uncluttered configuration spaces, the search trees grow hyperspherically, and hence BFMT* only has to expand about half as far (in both trees) as FMT* in order to return a solution. This is made clear in Fig. 1(a), in which FMT* would have to expand the forward tree twice as far to find a solution. Since runtime scales approximately with edge number, which scales as the linear distance covered by the tree raised to the dimension of the state space, we may expect in loosely cluttered configuration spaces an approximate speed-up of a factor 2^{d-1} over FMT* in d-dimensional space (the -1 in the exponent is because BFMT* has to expand 2 trees, so it loses one factor of 2 advantage).

III-C BFMT – Detailed description*

To understand the BFMT* algorithm, some background notation must first be introduced. Let $\mathcal{S}$ be a set of points

sampled independently and identically from the uniform distribution on $\mathcal{X}_{\text{free}}$, to which $\mathbf{x}_{\text{init}}$ and $\mathbf{x}_{\text{goal}}$ are added. (The extension to non-uniform sampling distributions is addressed in Section IV-C.) Let tree $\mathcal{T}$ be the quadruple $(\mathcal{V}, \mathcal{E}, \mathcal{V}_{\text{unvisited}}, \mathcal{V}_{\text{open}})$, where $\mathcal{V}$ is the set of tree nodes, $\mathcal{E}$ is the set of tree edges, and $\mathcal{V}_{\text{unvisited}}$ and $\mathcal{V}_{\text{open}}$ are mutually exclusive sets containing the *unvisited* samples in $\mathcal{S}$ and the *wavefront* nodes in $\mathcal{V}$, correspondingly. To be precise, the unvisited set $\mathcal{V}_{\text{unvisited}}$ stores all samples in the sample set $\mathcal{S}$ that have not yet been considered for addition to the tree of paths. The wavefront set $\mathcal{V}_{\text{open}}$, on the other hand, tracks in sorted order (by cost from the root) only those nodes which have already been added to the tree that are near enough to tree leaves to actually form better connections. These sets play the same role as their counterparts in FMT*, see [10], [11]. However, in this case BFMT* "grows" two such trees, referred to as $\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{V}_{\text{unvisited}}, \mathcal{V}_{\text{open}})$ and $\mathcal{T}' = (\mathcal{V}', \mathcal{E}', \mathcal{V}'_{\text{unvisited}}, \mathcal{V}'_{\text{open}})$. Initially, $\mathcal{T}$ is the tree rooted at $\mathbf{x}_{\text{init}}$, while $\mathcal{T}'$ is the tree rooted at $\mathbf{x}_{\text{goal}}$. Note, however, that the trees are exchanged during the execution of BFMT*, so $\mathcal{T}$ in Algorithm 1 is not always the tree that contains $\mathbf{x}_{\text{init}}$.

The BFMT* algorithm is represented in Algorithm 1. Before describing BFMT* in detail, we list briefly the basic planning functions employed by the algorithm. Let SAMPLEFREE(n) be a function that returns a set of $n \in \mathbb{N}$ points sampled independently and identically from the uniform distribution on $\mathcal{X}_{\text{free}}$. Let COST($\overline{\tilde{\mathbf{x}}\mathbf{x}}$) be the cost of the straight-line path between configurations $\tilde{\mathbf{x}}$ and $\mathbf{x}$. Let PATH($\mathbf{z}, \mathcal{T}$) return the unique path in tree $\mathcal{T}$ from its root to node $\mathbf{z}$. Also, with a slight abuse of notation, let COST($\mathbf{x}$, $\mathcal{T}$) return the cost of the unique path in tree $\mathcal{T}$ from its root to node $\mathbf{x}$, and let COLLISIONFREE($\mathbf{x}$, $\mathbf{y}$) be a boolean function returning true if the straight-line path between configurations $\mathbf{x}$ and $\mathbf{y}$ is collision free. Given a set of samples $\mathcal{A}$, let NEAR($\mathcal{A}$, $\mathbf{z}$, r) return the subset of $\mathcal{A}$ within a ball of radius r centered at sample $\mathbf{z}$ (*i.e.*, the set $\{\mathbf{x} \in \mathcal{A} \mid ||\mathbf{x} - \mathbf{z}|| < r\}$). Let the TERMINATE function represent an external termination criterion (*i.e.*, timeout, maximum number of samples, etc.) which can be used to force early termination (or prevent infinite runtime for infeasible problems). Finally, regarding tree expansion, let SWAP($\mathcal{T}$, $\mathcal{T}'$) be a function that swaps the two trees $\mathcal{T}$ and $\mathcal{T}'$. and let COMPANION($\mathcal{T}$) return the companion tree $\mathcal{T}'$ to $\mathcal{T}$ (or vice versa).

We are now in position to describe the BFMT* algorithm. First, a set of n configurations in $\mathcal{X}_{\text{free}}$ is determined by drawing samples uniformly. Two trees are then initialized using the INITIALIZE subfunction at the bottom of Algorithm 1, with a forward tree rooted at $\mathbf{x}_{\text{init}}$ and a reverse tree rooted at $\mathbf{x}_{\text{goal}}$. Once complete, tree expansion begins starting with tree $\mathcal{T}$ rooted at $\mathbf{x}_{\text{init}}$ using the EXPAND procedure in Algorithm 2. In the following, the node selected for expansion will be consistently denoted by $\mathbf{z}$, while $\mathbf{x}_{\text{meet}}$ will denote the lowest-cost candidate node for tree connection (*i.e.*, for joining the two trees). The EXPAND procedure requires the specification of a *connection radius* parameter, r_n, whose selection will be discussed in Section IV. EXPAND implements the "lazy" dynamic programming recursion described (at a high level) in Section III-B, making locally-optimal collision-free connections *from* nodes $\mathbf{x}$ near $\mathbf{z}$ unvisited by tree $\mathcal{T}$ (those in set $\mathcal{V}_{\text{unvisited}}$ within search radius r_n of $\mathbf{z}$) *to* wavefront nodes $\mathbf{x}'$ near each $\mathbf{x}$ (those in set $\mathcal{V}_{\text{open}}$ within search radius r_n of $\mathbf{x}$). Any collision-free edges and newly-connected nodes found are then added to $\mathcal{T}$, the connection candidate node $\mathbf{x}_{\text{meet}}$ is updated, and $\mathbf{z}$ is dropped from the list of wavefront nodes. The key feature of the EXPAND function is that in the execution of the dynamic programming recursion it "lazily" ignores the presence of obstacles (see line 6) – as discussed in Section IV this comes at no loss of (asymptotic) optimality (see also [10], [11]). Note the EXPAND function is identical to that of unidirectional FMT*, with the exception here of additional lines for tracking connection candidate $\mathbf{x}_{\text{meet}}$.

After expansion, the algorithm checks whether a feasible path is found on line 7. If unsuccessful so far, TERMINATE (which reports failure upon early termination) is checked before proceeding. If the algorithm has not terminated, it checks whether the wavefront of the companion tree is empty (line 11). If this is the case, the INSERT function shown in Algorithm 3 samples a new configuration $\mathbf{s}$ uniformly from $\mathcal{X}_{\text{free}}$ and tries to connect it to a nearest neighbor in the companion tree within radius r_n. This way, the expanding tree is ensured to have at least one configuration in its wavefront available for expansion on subsequent iterations (the alternative would be to report failure). This mimics anytime behavior, and by forcing samples to lie close to tree nodes we effectively "reopen" closed nodes for expansion again. Uniform resampling may require many attempts before finding a configuration $\mathbf{s}$ which can be successfully connected to $\mathcal{V}'_{\text{open}}$, though this appeared to have a negligible impact on running time for our path planning studies. On the other hand, a more effective strategy might bias resampling towards areas requiring expansion (*e.g.*, bottlenecks, traps) rather than uniformly within tree coverage.

The algorithm then proceeds on lines 12–13 with the selection of the next node (and corresponding tree) for expansion. As shown, BFMT* "swaps" the forward and backward trees on each iteration, each being expanded in turns. As INSERT ensures the companion tree $\mathcal{T}'$ always has at least one node in its frontier $\mathcal{V}'_{\text{open}}$, a node is always available for subsequent expansion as the next $\mathbf{z}$. After selection, the entire process is iterated.

III-C.1 BFMT – Variations:* As for any bi-directional planner, the correctness and computational efficiency of BFMT* hinge upon two key aspects: (i) how computation is interleaved among the two trees (in other words, which wavefront at each step should be chosen for expansion), and (ii) when the algorithm should terminate. For instance, as an alternative tree expansion strategy (*i.e.*, item (i)), one could replace lines 12–13 with the "balanced trees" condition which enforces more of a balanced search, maintaining equal costs from the root within each wavefront such that the two wavefronts propagate and meet roughly equidistantly in cost-to-go from their roots:

12: $\mathbf{z}_1 \leftarrow \underset{\mathbf{x} \in \mathcal{V}_{\text{open}}}{\arg\min} \{\text{COST}(\mathbf{x}, \mathcal{T})\}$
13: $\mathbf{z}_2 \leftarrow \underset{\mathbf{x}' \in \mathcal{V}'_{\text{open}}}{\arg\min} \{\text{COST}(\mathbf{x}', \mathcal{T}')\}$
14: $(\mathbf{z}, \mathcal{T}) \leftarrow \underset{(\mathbf{z}_1, \mathcal{T}), (\mathbf{z}_2, \mathcal{T}')}{\arg\min} \{\text{COST}(\mathbf{z}_i, \mathcal{T}_i)\}$
15: $\mathcal{T}' = \text{COMPANION}(\mathcal{T})$

Similarly, as an alternative termination condition (*i.e.*, item

Algorithm 1 The Bi-directional Fast Marching Tree Algorithm (BFMT*)

Require: Query $(\mathbf{x}_{\text{init}}, \mathbf{x}_{\text{goal}})$, Search radius r_n, Sample count n
1: $\mathcal{S} \leftarrow \{\mathbf{x}_{\text{init}}, \mathbf{x}_{\text{goal}}\} \cup \text{SAMPLEFREE}(n)$
2: $\mathcal{T} \leftarrow \text{INITIALIZE}(\mathcal{S}, \mathbf{x}_{\text{init}})$
3: $\mathcal{T}' \leftarrow \text{INITIALIZE}(\mathcal{S}, \mathbf{x}_{\text{goal}})$
4: $\mathbf{z} \leftarrow \mathbf{x}_{\text{init}}$, $\mathbf{x}_{\text{meet}} \leftarrow \varnothing$, $\sigma^* \leftarrow \varnothing$
5: **while** $\sigma^* = \varnothing$
6: $\quad \{\mathbf{x}_{\text{meet}}, \mathcal{T}\} \leftarrow \text{EXPAND}(\mathcal{T}, \mathbf{z}, r_n, \mathbf{x}_{\text{meet}})$
7: $\quad$ **if** $\mathbf{x}_{\text{meet}} \neq \varnothing$
8: $\quad\quad \sigma^* \leftarrow \text{PATH}(\mathbf{x}_{\text{meet}}, \mathcal{T}) \cup \text{PATH}(\mathbf{x}_{\text{meet}}, \mathcal{T}')$
9: $\quad\quad$ **break**
10: $\quad$ **else if** TERMINATE() **then return** Failure
11: $\quad$ **else if** $\mathcal{V}'_{\text{open}} = \varnothing$ **then** $\mathcal{T}' \leftarrow \text{INSERT}(\mathcal{T}', r_n)$
12: $\quad \mathbf{z} \leftarrow \underset{\mathbf{x}' \in \mathcal{V}'_{\text{open}}}{\arg\min} \{\text{COST}(\mathbf{x}', \mathcal{T}')\}$
13: $\quad \text{SWAP}(\mathcal{T}, \mathcal{T}')$
14: **return** σ^*

function INITIALIZE($\mathcal{S}$, $\mathbf{x}_0$)
$\quad \mathcal{V} \leftarrow \varnothing$, $\mathcal{E} \leftarrow \varnothing$, $\mathcal{V}_{\text{unvisited}} \leftarrow \mathcal{S}$, $\mathcal{V}_{\text{open}} \leftarrow \varnothing$
$\quad$ **return** $\mathcal{T} \leftarrow \text{ADDNODE}((\mathcal{V}, \mathcal{E}, \mathcal{V}_{\text{unvisited}}, \mathcal{V}_{\text{open}}), \mathbf{x}_0)$

function ADDNODE($\mathcal{T}, \mathbf{x}$)
$\quad \mathcal{V} \leftarrow \mathcal{V} \cup \{\mathbf{x}\}$ ▷ Add $\mathbf{x}$ to tree
$\quad \mathcal{E} \leftarrow \mathcal{E} \cup \{(\mathbf{x}_{\min}, \mathbf{x})\}$ ▷ Add edge to tree
$\quad \mathcal{V}_{\text{unvisited}} \leftarrow \mathcal{V}_{\text{unvisited}} \setminus \{\mathbf{x}\}$ ▷ Mark $\mathbf{x}$ as visited
$\quad \mathcal{V}_{\text{open}} \leftarrow \mathcal{V}_{\text{open}} \cup \{\mathbf{x}\}$ ▷ Add $\mathbf{x}$ to wavefront
$\quad$ **return** $\mathcal{T} \leftarrow (\mathcal{V}, \mathcal{E}, \mathcal{V}_{\text{unvisited}}, \mathcal{V}_{\text{open}})$

(ii)), one might replace line 7 with the "best path" criterion:

7: $\mathbf{z} \in \left(\mathcal{V}' \setminus \mathcal{V}'_{\text{open}}\right)$

Currently line 7 returns the first available path discovered, at the moment that the two wavefronts touch at $\mathbf{x}_{\text{meet}}$ (which is not, in general, the lowest cost path). This alternative condition, on the other hand, returns the *exact optimal path* from $\mathbf{x}_{\text{init}}$ to $\mathbf{x}_{\text{goal}}$ through the given set $\mathcal{S}$ of n samples. This change terminates BFMT* when the two wavefronts have propagated sufficiently far through each other that no better solution can be discovered. Intuitively-speaking, this occurs at the first moment where the two trees have both selected, at the current iteration or previously, the same node as the minimum cost node $\mathbf{z}$ from their respective roots.

Though seemingly promising ideas, no appreciable differences in performance were found using the above criteria in combination or otherwise; hence we report only the simplest version of our planner as Algorithm 1.

IV Asymptotic Optimality of BFMT*

In this section, we prove the asymptotic optimality of BFMT*. We begin with a result called *probabilistic exhaustivity* that essentially states that any path in $\mathcal{X}_{\text{free}}$ may be "traced" arbitrarily well by connecting points from a sufficiently large randomly-distributed sample set covering $\mathcal{X}_{\text{free}}$. We then prove the (asymptotic) optimality of BFMT* by showing that it returns solutions with costs no greater than that of any tracing path. The claim is proven assuming BFMT* acts without the INSERT procedure (Algorithm 3), in place of which "Failure" is reported instead. The proof for the full algorithm then follows immediately by *a fortiori* argument.

Algorithm 2 Fast Marching Tree Expansion Step

function EXPAND($\mathcal{T}$, $\mathbf{z}$, r_n, $\mathbf{x}_{\text{meet}}$)
$\quad \mathcal{V}_{\text{open,new}} \leftarrow \varnothing$
$\quad Z_{\text{near}} \leftarrow \text{NEAR}(\mathcal{V}_{\text{unvisited}}, \mathbf{z}, r_n)$
$\quad$ **for** $\mathbf{x} \in Z_{\text{near}}$
$\quad\quad X_{\text{near}} \leftarrow \text{NEAR}(\mathcal{V}_{\text{open}}, \mathbf{x}, r_n)$
$\quad\quad \mathbf{x}_{\min} \leftarrow \underset{\tilde{\mathbf{x}} \in X_{\text{near}}}{\arg\min} \{\text{COST}(\tilde{\mathbf{x}}, \mathcal{T}) + \text{COST}(\overline{\tilde{\mathbf{x}}\mathbf{x}})\}$
$\quad\quad$ **if** COLLISIONFREE($\mathbf{x}_{\min}, \mathbf{x}$)
$\quad\quad\quad (\mathcal{V}, \mathcal{E}, \mathcal{V}_{\text{unvisited}}, \mathcal{V}_{\text{open,new}}) \leftarrow$
$\quad\quad\quad\quad \text{ADDNODE}((\mathcal{V}, \mathcal{E}, \mathcal{V}_{\text{unvisited}}, \mathcal{V}_{\text{open,new}}), \mathbf{x})$
$\quad\quad\quad$ **if** $\{\mathbf{x} \in \mathcal{V}'$ and $\text{COST}(\mathbf{x}, \mathcal{T}) + \text{COST}(\mathbf{x}, \mathcal{T}') <$
$\quad\quad\quad\quad \text{COST}(\mathbf{x}_{\text{meet}}, \mathcal{T}) + \text{COST}(\mathbf{x}_{\text{meet}}, \mathcal{T}')\}$
$\quad\quad\quad\quad \mathbf{x}_{\text{meet}} \leftarrow \mathbf{x}$ ▷ Save $\mathbf{x}$ as best connection
$\quad \mathcal{V}_{\text{open}} \leftarrow (\mathcal{V}_{\text{open}} \cup \mathcal{V}_{\text{open,new}}) \setminus \{\mathbf{z}\}$ ▷ Add new nodes to the wavefront; drop $\mathbf{z}$ from the wavefront
$\quad$ **return** $\{\mathbf{x}_{\text{meet}}, \mathcal{T} \leftarrow (\mathcal{V}, \mathcal{E}, \mathcal{V}_{\text{unvisited}}, \mathcal{V}_{\text{open}})\}$

Algorithm 3 Insertion of New Samples

function INSERT($\mathcal{T}$, r_n)
$\quad$ **while** $\mathcal{V}_{\text{open}} = \varnothing$ **and not** TERMINATE()
$\quad\quad \mathbf{s} \leftarrow \text{SAMPLEFREE}(1)$
$\quad\quad V_{\text{near}} \leftarrow \text{NEAR}(\mathcal{V}, \mathbf{s}, r_n)$
$\quad\quad$ **while** $V_{\text{near}} \neq \varnothing$
$\quad\quad\quad \mathbf{x}_{\min} \leftarrow \underset{\mathbf{x} \in V_{\text{near}}}{\arg\min} \{\text{COST}(\mathbf{x}, \mathcal{T}) + \text{COST}(\overline{\mathbf{x}\mathbf{s}})\}$
$\quad\quad\quad$ **if** COLLISIONFREE($\mathbf{x}_{\min}$, $\mathbf{s}$)
$\quad\quad\quad\quad \mathcal{T} \leftarrow \text{ADDNODE}(\mathcal{T}, \mathbf{s})$
$\quad\quad\quad\quad$ **break**
$\quad\quad\quad$ **else then** $V_{\text{near}} \leftarrow V_{\text{near}} \setminus \{\mathbf{x}_{\min}\}$
$\quad$ **return** $\mathcal{T} \leftarrow (\mathcal{V}, \mathcal{E}, \mathcal{V}_{\text{unvisited}}, \mathcal{V}_{\text{open}})$

IV-A Probabilistic exhaustivity

Let $\sigma : [0, 1] \to \mathcal{X}$ be a path. Given a set of samples (referred to as waypoints) $\{\mathbf{y}_m\}_{m=1}^M \subset \mathcal{X}$, we associate a path $y : [0, 1] \to \mathcal{X}$ that sequentially connects the nodes $\mathbf{y}_1, \ldots, \mathbf{y}_M$ with line segments. We consider the waypoints $\{\mathbf{y}_m\}$ to (ϵ, r)-*trace* the path σ if: (i) $\left|\left|\mathbf{y}_m - \mathbf{y}_{m+1}\right|\right| \leq r$ for all m, (ii) the cost of y is bounded as $J(y) \leq (1+\epsilon)J(\sigma)$, and (iii) the distance from any point of y to σ is no more than r, *i.e.*, $\min_{t \in [0,1]} ||y(s) - \sigma(t)|| \leq r$ for all $s \in [0, 1]$. In the context of sampling-based motion planning, we may expect to find closely-tracing $\{\mathbf{y}_m\}$ as a subset of the sampled points, provided the sample size is large. This notion is formalized below (Theorem 4.1, proven as Theorem IV.5 in [20]).

Theorem 4.1 (Probabilistic exhaustivity): Define path planning problem $(\mathcal{X}_{\text{free}}, \mathbf{x}_{\text{init}}, \mathbf{x}_{\text{goal}})$ and let $\sigma : [0, 1] \to \mathcal{X}_{\text{free}}$ be a feasible path. Denote the volume of the d-dimensional Euclidean unit ball by ζ_d. Finally, let $\mathcal{S} = \{\mathbf{x}_{\text{init}}, \mathbf{x}_{\text{goal}}\} \cup \text{SAMPLEFREE}(n)$, $\epsilon > 0$, and for fixed n consider the event $\mathcal{A}_n$ that there exist $\{\mathbf{y}_m\}_{m=1}^M \subset \mathcal{S}$, $y_1 = \mathbf{x}_{\text{init}}$, $y_M = \mathbf{x}_{\text{goal}}$ which (ϵ, r_n)-trace σ, where

$$r_n = 4\,(1+\eta)^{\frac{1}{d}} \left(\frac{1}{d}\right)^{\frac{1}{d}} \left(\frac{\mu(\mathcal{X}_{\text{free}})}{\zeta_d}\right)^{\frac{1}{d}} \left(\frac{\log n}{n}\right)^{\frac{1}{d}} \quad (1)$$

for a parameter $\eta \geq 0$. Then, as $n \to \infty$, the probability that $\mathcal{A}_n$ does not occur (denoted by its complement $\mathcal{A}_n^c$) is asymptotically bounded as $\mathrm{P}[\mathcal{A}_n^c] = \mathrm{O}\left(n^{-\frac{\eta}{d}} \log^{-\frac{1}{d}} n\right)$.

IV-B Asymptotic optimality (AO)

We are now in a position to prove the asymptotic optimality of BFMT*, which represents the main result of this section. We start with an important lemma, which relates the cost of the path returned by BFMT* to that of *any* feasible path.

Lemma 4.2 (Bi-directional FMT cost comparison):* Let $\sigma : [0, 1] \to \mathcal{X}_{\text{free}}$ be a feasible path with strong δ-clearance. Consider running BFMT* to completion with n samples and a connection radius r_n given by Eq. (1) with $\eta \geq 0$. Let J_n denote the cost of the path returned by BFMT*. Then for fixed $\epsilon > 0$:

$$\mathrm{P}[J_n > (1+\epsilon)J(\sigma)] = \mathrm{O}\Big(n^{-\frac{\eta}{d}} \log^{-\frac{1}{d}} n\Big).$$

Proof: In the interest of brevity, we provide a brief sketch of the proof here; for full details, however, we refer the interested reader to the extended version of the paper [21]. The proof begins by considering a regime of n sufficiently large so that the connection radius r_n is "small" with respect to both the obstacle clearance δ and the path cost $J(\sigma)$. Running BFMT* to completion generates one cost-to-come tree $\mathcal{T}_i$ and one cost-to-go tree $\mathcal{T}_g$ rooted at $\mathbf{x}_{\text{init}}$ and $\mathbf{x}_{\text{goal}}$, respectively. Provided tracing waypoints $\{\mathbf{y}_m\}$ exist within the sample set (this occurs with increasingly high probability for large n, by Theorem 4.1), the choice of r_n small with respect to δ ensures that these waypoints will represent a feasible sequence of connections for BFMT*. Moreover it may be shown that BFMT*, like FMT*, will recover a path with cost no worse than $J(y)$ — up to an additional additive factor of r_n which arises when the two trees $\mathcal{T}_i$ and $\mathcal{T}_g$ meet, motivating the second choice of r_n small with respect to $J(\sigma)$. As $J(y)$ is, by definition, close to $J(\sigma)$, we achieve the desired result. ■

Remark 4.3 (Alternative termination criteria): The proof holds as well for the different expansion and termination criteria discussed in Section III-C.1. However, due to space constraints the details are omitted.

We are now ready to show that BFMT* is asymptotically-optimal. The next theorem defines this formally.

Theorem 4.4 (BFMT asymptotic optimality):* Assume a δ-robustly feasible path planning problem as defined in Section II with optimal path σ^* of cost J^*. Then BFMT* converges *in probability* to σ^* as the number of samples $n \to \infty$. Specifically, for any $\epsilon > 0$,

$$\lim_{n\to\infty} \mathrm{P}[J_n > (1+\epsilon)J^*] = 0$$

Proof: We provide a brief sketch of the proof here but refer the interested reader to [21] for full details. The proof essentially follows as a corollary to Lemma 4.2. The δ-robustly feasible property ensures that we may approach the cost of σ^* arbitrarily well by approximant paths with strong clearance, which satisfy the requirement of Lemma 4.2. Lemma 4.2 implies that the output of BFMT* may in turn approach (with high probability as $n \to \infty$) the cost of any of these approximants arbitrarily well, which consequently implies the final result. ■

IV-C Sampling and cost generalizations

It is worth mentioning that the asymptotic optimality (AO) properties of BFMT* are not limited to uniform sampling and arc-length cost functions. For non-uniform sampling, so long as sample density is lower-bounded by a *positive* number over the configuration space, one need only increase r_n by a constant factor to ensure BFMT* stays AO. Furthermore, to handle other metric or line integral costs, the Euclidean balls used when making node connections need only be replaced by *cost* balls. Details on adjusting the algorithm and why the AO proof still holds can be derived from [11].

V Simulations

In this section, we provide numerical path-planning experiments that compare the performance of BFMT* with other sampling-based, asymptotically-optimal planning algorithms (namely, FMT*, RRT*, and PRM*)[2]. Given a planning workspace and query, we aim to observe the quality of the solution returned as a function of the execution time allotted to the algorithm. Here dynamic constraints are neglected and arc-length is used as path cost. As a basis for quality comparison between incremental or "anytime" planners (such as RRT*) and non-incremental planners (such as BFMT*, which generate solutions via sample batches), we vary the number of samples drawn by the planners during the planning process (which in essence serves as a proxy to execution time). Note *sample count* has a different connotation depending on the planner that will not necessarily be the number of nodes stored in the constructed solution graph – for RRT* (with one sample drawn per iteration), this is the number of iterations, while for FMT*, PRM*, and BFMT*, this is the number of free space samples taken during initialization.

V-A Simulation Setup

To generate simulation data for a given experiment, we queried the planning algorithms once each for a series of sample counts, recorded the cost of the solution returned, the planner execution time[3], and whether the planner succeeded or not, then repeated this process over 50 trials. To ensure a fair comparison, each planning algorithm was tested using the Open Motion Planning Library (OMPL) v1.0.0 [22], which provides high-quality implementations of many state-of-the-art planners and a common framework for executing motion plans. In this way, we could ensure that all algorithms employed the *exact same* primitive routines (*e.g.*, nearest-neighbor search, collision-checking, data handling, *etc.*), and measure their performances fairly. Regarding implementation, BFMT*, FMT*, and PRM* used $\eta = 0$ from Lemma 4.2 for the nearest-neighbor radius r_n in order to satisfy the theoretical bounds provided in Section IV and [6]. For RRT*, we used the default OMPL settings; namely, a 5% goal bias and a steering parameter equal to 20% of the maximum extent of the configuration space (except for the α-puzzle, in which case a value of 1.1 was found to work much better). For FMT*, we included the same INSERT routine as BFMT* for configuration resampling upon failure. For all algorithms, early termination (*e.g.*, using TERMINATE for BFMT*) was

[2]Existing state-of-the-art sampling-based, bi-directional algorithms (namely, RRT-Connect and SBL) were initially also included. However, average costs for RRT-Connect and SBL were roughly 2-4x greater, which occluded the details of other curves; they were thus omitted for clarity

[3]Code for all experiments was written in C++. Corresponding programs were compiled and run on a Linux-operated PC, clocked at 2.4 GHz and equipped with 7.5 GB of RAM.

suppressed by defining a 1000 second time limit, well above each planner's worst-case execution time.

Before proceeding, note that each marker shown on the plots throughout this section represents a single simulation at a fixed sample count. The points on the curves, however, represent the mean cost/time of *successful* algorithm runs *only* for a particular sample count, with error bars corresponding to one standard deviation of the 50 run sample mean.[4] Sample counts varied from the order of 200 to 2000 points for 2D problems, from 1000 to 30000 points for 3D problems, and 500 to 4000 points for the hypercube examples.

V-B Results and Discussion

Here we present benchmarking results (average solution cost versus average execution times and success rates) comparing BFMT* to other state-of-the-art sampling-based planners. Three benchmarking test scenarios were considered: (1) a 2D "bug trap" and (2) a 2D "maze" problem for a convex polyhedral robot in the $\mathbb{SE}(2)$ configuration space, as well as (3) a challenging 3D problem called the "α-puzzle" in which we seek to untangle two loops of metal (non-convex) in the $\mathbb{SE}(3)$ configuration space. All problems were drawn directly from OMPL's bank of tests, and are illustrated in Fig. 2. In each case, collision-checks relied on OMPL's built-in collision-checking library, FCL. Additionally, to tease out the performance of BFMT* relative to FMT* in high-dimensional environments, we also studied a point mass robot moving in cluttered unit hypercubes of 5 and 10 dimensions.[5]

(a) $\mathbb{SE}(2)$ bug trap (b) $\mathbb{SE}(2)$ maze (c) $\mathbb{SE}(3)$ α-puzzle

Fig. 2: Depictions of the OMPL rigid-body planning problems

Figure 3 shows the results for each BFMT*, FMT*, RRT*, and PRM*. Performance here is measured by execution time on the x-axis and solution cost on the y-axis—high quality data points are therefore located in the lower-left corner (low-cost solutions obtained quickly). The plots reveal that both FMT* and BFMT* for the most part outperform RRT* as well as PRM*. In particular, BFMT* and FMT* achieve higher success rates[6] (always a flat 100% for the cases studied) in shorter time. To extract further information, we need to examine each test in detail.

In the Bug Trap and Maze problems, BFMT* notably generates the same cost-time curve as FMT* (meaning they return solutions of very similar cost for a given sample count), but with data points shifted to the left (indicating they were obtained in shorter execution time). Though not shown due to slow running times for PRM* (whose results had to be truncated to clarify detail), all planners appear to tend towards similar low-cost solutions as more execution time was allocated. However BFMT* and FMT* seem to converge to an optimum much faster, particularly for the Maze problem (on the order of 1.5 and 2.0 seconds respectively, compared to 3-4 seconds for RRT* and 5-7 seconds for PRM*). This contrast becomes even more evident for the α-puzzle. Here we see an unusual spread of solutions – one in a band at around 500 cost and another at around 275. These indicate the presence of two solution types, or *homotopy classes*: one corresponding to the true α-puzzle solution, and another less-efficient path. This appears to have yielded a "bump" in the BFMT* cost-curve, where increasing the sample count momentarily gives an increased average cost. We believe this is a result of how BFMT* trees interconnect; at this count, by unlucky circumstance, the longer homotopy seems to be found first more often than usual. But as proved in Section IV, the behavior disappears as $n \to \infty$. Note RRT* seems to avoid this issue through goal biasing. Despite the difficult problem structure, BFMT* finds the cheaper homotopy faster than other planners, with many more of its data points clustered in the lower-left corner, generally at lower costs and times than RRT* and of equal quality but faster times than FMT*.

These results suggest that BFMT* tends to an optimal cost at least as fast as the other planners, and sometimes much faster. To shed light on the relative performance of FMT* and BFMT* further, we compare them in higher dimensions. Results for the 5D and 10D hypercube are shown in Fig. 4 (success rates were again at 100%, and were thus omitted). Here BFMT* substantially outperforms FMT*, particularly as dimension increases, with convergence in roughly 0.5 and 1.4 seconds (5D), and 5 and 20 seconds (10D) on average. This suggests that *reachable volumes* play a significant role in their execution time. The relatively small volume of reachable configurations around the goal at the corner implies that the reverse tree of BFMT* expands its wavefront through many fewer states than the forward tree of FMT* (which in fact needlessly expands towards the zero-vector); tree interconnection in the bi-directional case prevents its forward tree from growing too large compared to unidirectional search. This is pronounced exponentially as the dimension increases. In trap or maze-like scenarios, however, bi-directionality does not seem to change significantly the number of states explored by the marching trees, leading to comparable performance for the $\mathbb{SE}(2)$ bug-trap and maze. Note we expect a greater contrast in execution times in favor of BFMT* as the cost of collision-checking increases, such as with many non-convex obstacles or in time-varying environments.

VI Conclusion

In this paper, we presented a bi-directional, sampling-based, asymptotically-optimal motion planning algorithm named BFMT*, for which we rigorously proved its optimality and characterized its convergence rate – arguably firsts in the field of bi-directional sampling-based planning. Numerical experiments in $\mathbb{R}^d$, $\mathbb{SE}(2)$, and $\mathbb{SE}(3)$ revealed that BFMT*

[4]Standard deviation of the mean indicates where we expect with one-σ confidence the distribution mean to lie based on the 50-run sample mean, and is related to the standard deviation of the distribution by $\sigma_\mu = \sigma/\sqrt{50}$.

[5]We populated the space to 50% obstacle coverage with randomly-sized, axis-oriented hyperrectangles. $\mathbf{x}_{\text{init}}$ was set to the center at $[0.5, \ldots, 0.5]$, with the goal $\mathbf{x}_{\text{goal}}$ at the ones-vector (*i.e.*, $[1, \ldots, 1]$).

[6]Note we achieve even better success rates than in the original FMT paper [11] due to the introduction of resampling from the INSERT algorithm.

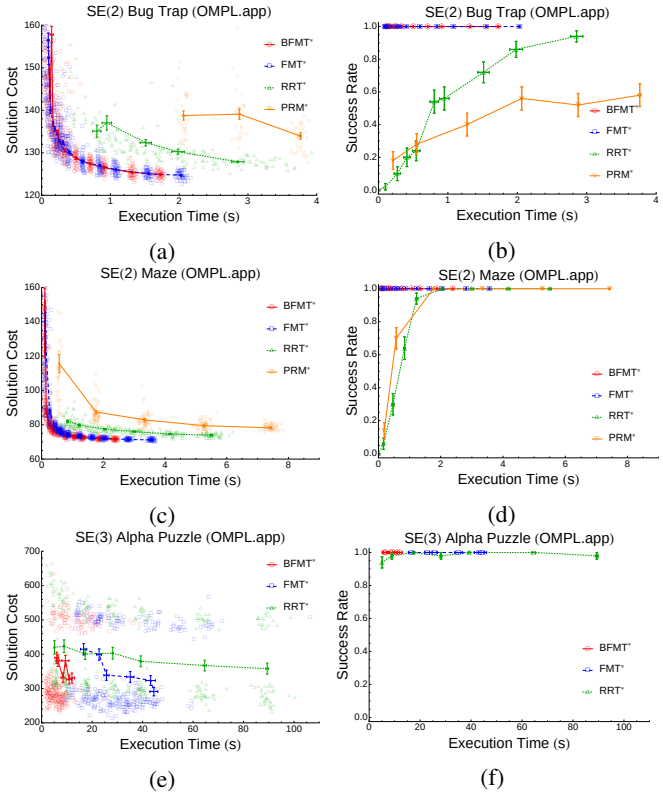


Fig. 3: Simulation results for the three OMPL scenarios.

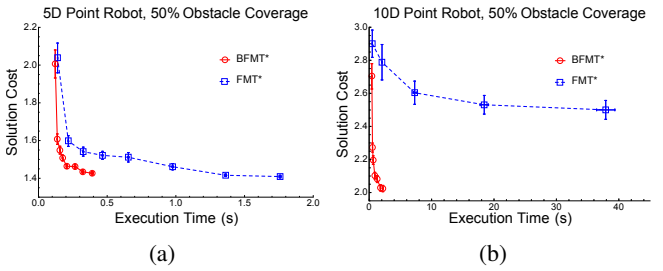


Fig. 4: FMT* and BFMT* results for 5D and 10D cluttered hypercubes (50% coverage; all success rates were 100%).

tends to an optimal solution at least as fast as its state-of-the-art counterparts, and in some cases significantly faster. Convergence rates are expected to improve with parallelization, in which each tree is grown using a separate CPU.

Future research will examine BFMT*'s interaction with more advanced techniques, such as adaptive sampling near narrow passages or sample biasing in INSERT (Algorithm 3) towards failed wavefronts. We also plan to extend BFMT* to dynamic environments through lazy re-evaluation (leveraging its tree-like forward and reverse path structures) in a way that reuses previous results as much as possible. Maintaining bounds on run-time performance and solution quality in this new context will be the greatest challenges. Ultimately, we hope that BFMT* will enable fast, easy-to-implement planning and re-planning in a wide range of time-varying scenarios, much as we have shown here for the static case.

REFERENCES

[1] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[2] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[3] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," *International Journal of Computational Geometry & Applications*, vol. 9, no. 4, pp. 495–512, 1999.

[4] J. M. Phillips, N. Bedrossian, and L. E. Kavraki, "Guided expansive spaces trees: A search strategy for motion- and cost-constrained state spaces," in *Proc. IEEE Conf. on Robotics and Automation*, vol. 4, 2004, pp. 3968–3973.

[5] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[6] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[7] J. Luo and K. Hauser, "An empirical study of optimal motion planning," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, Sep. 2014, pp. 1761–1768.

[8] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Bit*: Batch informed trees for optimal sampling-based planning via dynamic programming on implicit random geometric graphs," 2014, available at http://arxiv.org/abs/1405.5848.

[9] O. Arslan and P. Tsiotras, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *Proc. IEEE Conf. on Robotics and Automation*, Karlsruhe, Germany, May 2013, pp. 2421–2428.

[10] L. Janson and M. Pavone, "Fast Marching Trees: A fast marching sampling-based method for optimal motion planning in many dimensions," in *International Symposium on Robotics Research*, 2013.

[11] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *International Journal of Robotics Research*, vol. 34, no. 7, pp. 883–921, 2015.

[12] Y. K. Hwang and N. Ahuja, "Gross Motion Planning – a Survey," *ACM Comput. Surv.*, vol. 24, no. 3, pp. 219–291, Sep. 1992.

[13] I. Pohl, *Bi-directional and heuristic search in path problems*. Department of Computer Science, Stanford University., 1969, no. 104.

[14] M. Luby and P. Ragde, "A bidirectional shortest-path algorithm with good average-case behavior," *Algorithmica*, vol. 4, no. 1-4, pp. 551–567, 1989.

[15] A. Goldberg, H. Kaplan, and R. Werneck, *Reach for A*: Efficient Point-to-Point Shortest Path Algorithms*. Springer, 2006, ch. 12, pp. 129–143.

[16] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *Proc. IEEE Conf. on Robotics and Automation*, San Francisco, CA, Apr. 2000, pp. 995–1001.

[17] G. Sánchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," in *International Journal of Robotics Research*. Springer, 2003, pp. 403–417.

[18] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*. IEEE, 2011, pp. 2640–2645.

[19] M. Jordan and A. Perez, "Optimal Bidirectional Rapidly-Exploring Random Trees," Aug. 2013, http://people.csail.mit.edu/aperez/obirrt/csailtech.pdf.

[20] E. Schmerling, L. Janson, and M. Pavone, "Optimal sampling-based motion planning under differential constraints: the driftless case," in *Proc. IEEE Conf. on Robotics and Automation*, 2015, extended version available at http://arxiv.org/abs/1403.2483/.

[21] J. A. Starek, J. V. Gomez, E. Schmerling, L. Janson, L. Moreno, and M. Pavone, "An asymptotically-optimal sampling-based algorithm for bi-directional motion planning (extended version)," 2015, pp. 1–8, available at http://arxiv.org/abs/1507.07602/.

[22] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics and Automation Magazine*, vol. 19, no. 4, pp. 72–82, Dec. 2012.