BBoE: Leveraging Bundle of Edges for Kinodynamic Bidirectional Motion Planning

Srikrishna Bangalore Raghu and Alessandro Roncone

***Abstract*—In this work, we introduce *BBoE*, a bidirectional, kinodynamic, sampling-based motion planner that consistently and quickly finds solutions in environments with varying obstacle clutter. The algorithm combines exploration and exploitation while relying on precomputed robot motions, resulting in efficient convergence towards the goal. Our key contributions include: i) a strategy to quickly find solutions in obstacle-rich spaces by leveraging a bundle of precomputed robot motions (i.e., edges); and ii) BBoE, a robust bidirectional kinodynamic planner that utilizes this strategy to produce fast and feasible solutions. Across four systems spanning 3D to 6D state spaces, BBoE is the only planner to achieve $\geq 96\%$ success on every system while attaining the lowest planning time on three of four.**

***Index Terms*—Motion and Path Planning, Constrained Motion Planning, Nonholonomic Motion Planning**

I. INTRODUCTION

As robots are increasingly being deployed around us, they must operate in obstacle-rich scenarios (i.e., drones navigating disaster sites, mobile robots threading through a crowd of humans, and vehicles maneuvering in dense traffic), where they need to quickly find feasible and collision-free paths. This falls under the problem of motion planning: identifying a series of valid configurations that a robot can assume to transition from an initial state to a desired goal state. Among the popular techniques, sampling-based motion planning is a graph-based approach used to generate robot motions by sampling discrete states and establishing connections between them via edges [1]. Traditionally, these techniques employ a unidirectional tree that grows from the start state and expands towards the goal region [2], [3]. A faster alternative uses a bidirectional approach, where two trees rooted at the start and goal converge towards each other [4], [5]. In either case, these planners need to account for the constraints on the robots' kinematics (position, velocity) and dynamics (acceleration, forces, torques), also known as *kinodynamic constraints* [6].

Generating motions that respect these constraints is inherently difficult: unlike geometric planning, where any two nearby states can be connected by a straight line, kinodynamic systems can only move in ways their dynamics permit. This becomes especially challenging as the number of obstacles in the environment increases, resulting in very few possible solutions and the consequently low probability of finding them. A common approach for kinodynamic motion generation is forward propagation, which executes random control inputs for random durations in a Monte-Carlo attempt to reach a desired state [5], [7]. In cluttered spaces, it results in high planning times when accurate connections are preferred, high-cost paths when the connections are less accurate, and low success rates within a stipulated time limit. The Bundle of Edges (BoE) algorithm [8] removes this online cost by precomputing a large set of feasible motions (i.e., edges) offline and reusing them as a roadmap. However, BoE expands its tree by propagating every edge near the current state, resulting in repeated obstacle collisions and wasted propagation effort.

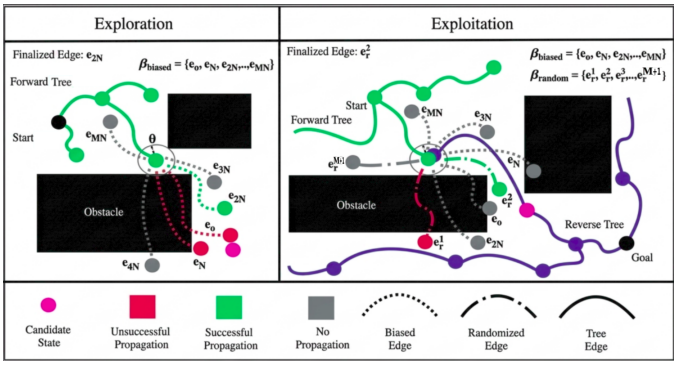


Fig. 1. The functionality of BBoE spans two phases, exploration and exploitation. During exploration, the forward tree primarily uses the biased edges (β_{biased}) to expand towards the candidate state, while the reverse tree relies on propagating random edges backward in time. During exploitation, either the biased edges or the randomized edges (β_{random}) are used while allowing the forward tree to accurately trace the reverse tree towards the goal.

Thus, whether motions are generated online or drawn from a precomputed bundle, current kinodynamic methods cannot reliably find solutions in highly cluttered and obstacle-dense environments without sacrificing speed. Overcoming this barrier requires a planner that reuses precomputed motions to avoid online propagation cost while handling them intelligently so that scarce solutions are found quickly even as clutter increases. Motivated by this, we present the Bidirectional Bundle of Edges (BBoE) algorithm which makes precomputed-motion planning viable in heavy clutter by reliably finding dynamically feasible solutions within practical time budgets across systems ranging from ground vehicles to aerial robots. This entails two contributions:

- A strategy to quickly find solutions in obstacle-rich spaces by sorting and sequencing a bundle of precomputed robot motions (i.e., edges).

Manuscript received: 2 July 2026, Revised: 19 September 2026, Accepted: 14 October 2026.

This paper was recommended for publication by Editor Lucia Pallottino upon evaluation of the Associate Editor and Reviewers' comments.

The authors are with the Department of Computer Science, University of Colorado Boulder, Boulder, CO, USA (Email: srba2850@colorado.edu). Code and supplementary material are available at https://brsrikrishna.github.io/BBoE_RAL/.

Digital Object Identifier (DOI): see top of this page.

- A bidirectional sampling-based motion planner that leverages this strategy to provide feasible solutions with reduced runtime and a high success rate amidst varying obstacle clutter, including higher-dimensional systems where existing precomputed-motion planners fail.

II. BACKGROUND AND RELATED WORK

A. Kinodynamic Motion Planning

To generate robot motions that follow kinematic and dynamic constraints, kinodynamic planners use two methods: Steering Functions and Forward Propagation. Steering functions use robot dynamics to analytically or numerically connect pairwise states, and have been used by mainstream planners like RRT [6] and RRT* [9]. However, they are computationally expensive and challenging to define for complex systems [10], [11]. To address the complexities of steering functions, planners such as SST [12] and DIRT [13] use forward propagation, a simpler and cheaper alternative. These planners sample several control inputs per expansion and keep the propagation ending closest to the desired state, thus treating the dynamics as a black box. This black-box treatment also extends forward propagation to contact-rich tasks, e.g., non-prehensile poking [14]. An alternative is to precompute robot motions offline and reuse them online. Motion primitive (MP) methods build a library of short, dynamically feasible trajectories that define an implicit graph over reachable states. These are usually constructed lazily during an A*-like search with edges ranked by a heuristic [15]. A related line of work, iDb-RRT [16] relaxes exact connections by allowing a bounded discontinuity, repaired afterward by trajectory optimization. BoE [8], LazyBoE [17], KiTE-Extend [18], and NPM-LMJ [19] apply the precomputed-motion idea in the sampling-based setting without discontinuities, using a bundle of edges as a probabilistic roadmap [20].

However, these methods do not focus on quickly finding solutions amidst a clutter of obstacles. Planners that have tried to tackle this contain drawbacks. RRT*-Smart [21] uses RRT* to generate an initial plan while relying on steering functions, while OracleNet [22] requires training for every new environment to find narrow pathways between states. Spill-free liquid transport in clutter [23] plans a geometric path with RRT* and enforces dynamics only afterwards through time parameterization, while CAT-RRT [24] traverses dense clutter by admitting contact, which the robot and its surroundings must tolerate. Furthermore, these algorithms only comprise unidirectional trees, which are slower at finding solutions compared to bidirectional alternatives.

B. Bidirectional Motion Planning

Bidirectionality in sampling-based motion planning was introduced by RRT-Connect [4], where two trees grow towards each other until they're connected, resulting in lower planning times. This benefit is seen in subsequent bidirectional planners, such as AIT* [25] and RRT*-Connect [1]. Bidirectional searches have previously attempted to quickly find solutions in cluttered spaces, but they contain drawbacks. B-Spline RRT [26] is tailor-made specifically for cars, while BI^{2}RRT* [27] banks on basic online propagation for traversal. In the kinodynamic domain, these algorithms traditionally depend on steering functions to connect the trees. Conversely, the Generalized Bidirectional Rapidly Exploring Random Tree (GBRRT) [5] avoids the complex computation of connecting the trees by enabling the reverse tree to guide the forward tree towards the goal. This spans over two phases. During the exploration phase, both trees explore the state space until they're close enough. The exploitation phase allows the forward tree to use the reverse tree as a heuristic and advance towards the goal. However, both phases rely on online Monte-Carlo propagations while targeting desired states, resulting in significant computation time and low success rates in heavily cluttered environments. Next, we present how BBoE combines precomputed edge handling with bidirectional search to enable fast navigation through obstacle clutter.

III. METHODS

A. Problem Formulation

In the kinodynamic domain, the state space comprises the robot's configuration (e.g., positions and angles) alongside its higher-order derivatives (e.g., velocities and accelerations). We define the robot state space $Q \subset \mathbb{R}^m$ containing m-dimensional states $x \in Q$ alongside the control space $U \subset \mathbb{R}^n$ consisting of n-dimensional controls $u \in U$. Given time t, the forward dynamics model of the robot is defined by:

$$\dot{x} = f(x(t), u(t)) \qquad x \in Q,\, u \in U$$

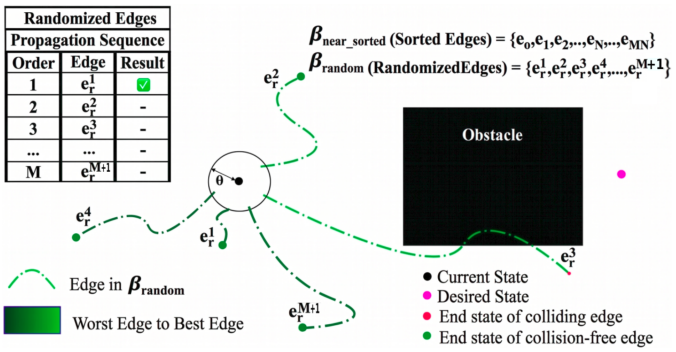
Given a start state x_0 and a goal region $Q_{goal} \subseteq Q$, BBoE solves the *feasibility* problem: find a collision-free, dynamically feasible trajectory π,

$$\pi(t) : [0,1] \to Q_{free} \subseteq Q, \quad \pi(0) = x_0,\, \pi(1) \in Q_{goal},$$

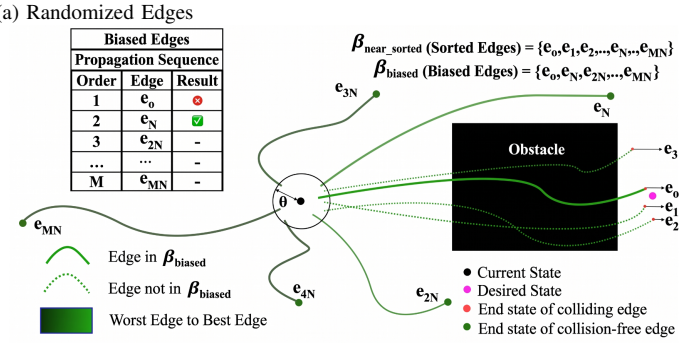
where every segment of π results from integrating f under an admissible control. We measure a planner by its time to the first solution and by its success rate within a fixed budget. BBoE claims no optimality with respect to path cost. Like GBRRT [5], BBoE grows a forward tree from x_0 and a reverse tree from Q_{goal} in two phases. During *exploration*, both trees expand toward randomly sampled states until they arrive within a threshold distance of each other. During *exploitation*, the forward tree expands toward reverse-tree nodes, which act as a heuristic leading it into the goal region.

B. Bundle of Edges

To enable accurate propagation from one state to the other, we leverage a bundle of precomputed robot motions (i.e., a bundle of edges) similar to the BoE [8] planner. An edge bundle consists of an expansive number of forward propagations randomly generated in the robot's state space. The bundle is computed offline and stored for future use. Once a motion planning query is initiated, the bundle is retrieved, and the edges that initiate in the vicinity of the robot's current state are used as a reference/map to guide the robot to the desired state. The computation performed offline reduces the burden on the planner during the online (i.e.,



(a) Randomized Edges



(b) Biased Edges

Fig. 2. Given the robot's current state, we gather all edges initiating within a radius of θ from it and store them in $\boldsymbol{\beta}_{\text{near_sorted}}$. From this set, we extract relevant edges and store them in sets $\boldsymbol{\beta}_{\text{random}}$ (a) and $\boldsymbol{\beta}_{\text{biased}}$ (b). We show the exact order of edge propagation in each scenario with the result.

planning) phase.

The edge bundle ε is formed when a set of random start states $x_0 \in Q$ undergoes forward propagation by applying random control inputs $u \in U$ for random time durations Δt, resulting in the robot reaching final states $x_f \in Q$:

$$\varepsilon = \{(x_0, u, \Delta t, x_f) | x_0, x_f \in Q, u \in U\}$$

where $x_f = x(\Delta t)$ is the solution of $\dot{x} = f(x, u)$ with $x(0) = x_0$ under the edge's control u over $[0, \Delta t]$. We write a generic edge as $\mathrm{e} = (x_0, u, \Delta t, x_f) \in \varepsilon$. Subscripts (e.g., e_i) are used only after the edges are sorted in Section III-C, where i denotes the position in the sorted list. To expand a tree node x_{near} toward a desired state x_{des}, the planner considers the neighboring edges (i.e., those whose start state x_0 lies within a radius θ of x_{near}). After a rigid-body transformation of each such edge's pose components onto x_{near}, the stored end state x_f serves only to *rank* the candidates. The stored control u and duration Δt are then propagated from the actual node x_{near} (i.e., the true dynamics are integrated online). Consequently, a mismatch between x_0 and x_{near}, which θ keeps bounded, degrades only the ranking heuristic and never the feasibility of the added trajectory. This is because every tree edge is exact and free of discontinuities. Because the pose components are transformed away, the bundle only needs to cover the kinodynamic subspace (velocities, steering and hitch angles) rather than the full state space. However, adding obstacles to the state space after the edge generation phase results in several edges colliding with these obstacles during the planning phase. Next, we try to overcome this issue with a strategic approach.

C. Edge Selection Strategy

An expansive and dense edge bundle can guide the robot from its current state to the desired state. For this purpose, BoE [8] propagates all the edges in the vicinity of the current state, resulting in excessive computation when new obstacles are introduced. As seen in Fig. 2 (b), once we gather all the edges that initiate within a neighborhood of radius θ from the robot's current state, we don't have to spend redundant computation by propagating all of them while aiming to reach the desired state. A more computationally efficient approach is to sort them based on the distance between their end state and the desired state, resulting in $\boldsymbol{\beta}_{\text{near_sorted}}$ $(\{\mathrm{e}_0, \mathrm{e}_1, \mathrm{e}_2, \ldots\})$. They can then be propagated one by one until a collision-free trajectory is obtained. However, if the best edge e_0, that gets the closest to the desired state, collides with an obstacle that was not present during edge bundle generation, the next best edges e_1, e_2 and e_3 have a high chance of colliding with the same obstacle, due to the extensive number of edges.

Our strategy maximizes the reach towards the desired state while minimizing the number of propagation attempts. To do so, we choose a user-defined integer $\mathrm{N} \geq 1$ and let $\mathrm{M} = \lfloor (|\boldsymbol{\beta}_{\text{near_sorted}}| - 1)/\mathrm{N} \rfloor$, so that MN is the largest multiple of N that indexes a sorted edge. We pick the best edge (i.e., e_0) and every Nth edge from the beginning of $\boldsymbol{\beta}_{\text{near_sorted}}$(i.e., $\mathrm{e}_\mathrm{N}, \mathrm{e}_{2\mathrm{N}}, \mathrm{e}_{3\mathrm{N}}, \ldots, \mathrm{e}_{\mathrm{MN}}$) and add them to a list $\boldsymbol{\beta}_{\text{biased}}$, giving rise to M+1 biased edges $(\{\mathrm{e}_0, \mathrm{e}_\mathrm{N}, \mathrm{e}_{2\mathrm{N}}, \mathrm{e}_{3\mathrm{N}}, \ldots, \mathrm{e}_{\mathrm{MN}}\})$. Essentially, if the best edge e_0 undergoes a collision, we are skipping $N - 1$ edges that have a high chance of colliding with the same obstacle, thus avoiding redundant computation. Until a collision-free trajectory is obtained, we attempt to propagate every Nth edge until we reach the last biased edge (i.e., e_{MN}), which is the worst-ranked edge we consider. In heavily cluttered spaces, there is a high probability that most of the edges in $\boldsymbol{\beta}_{\text{biased}}$ lead to collisions, resulting in very few successful expansions. The trajectories that are successful could either traverse away from the desired state or converge very little towards it. This is problematic whenever the current and desired states remain fixed, as in the exploitation phase of Fig. 1.

To tackle this issue, we refer to Fig. 2 (a), where M+1 edges (i.e., $\mathrm{e}_\mathrm{r}^1, \mathrm{e}_\mathrm{r}^2, \ldots, \mathrm{e}_\mathrm{r}^{\mathrm{M+1}}$), as many as in $\boldsymbol{\beta}_{\text{biased}}$, are drawn uniformly without replacement from $\boldsymbol{\beta}_{\text{near_sorted}}$ and stored in $\boldsymbol{\beta}_{\text{random}}$. This is a better alternative in cluttered spaces for a fixed pair of current and desired states. Due to the inherent stochasticity, using the randomized edges (i.e., $\boldsymbol{\beta}_{\text{random}}$) might find feasible narrow paths absent in $\boldsymbol{\beta}_{\text{biased}}$ that eventually converge towards the goal. This strategy of choosing either $\boldsymbol{\beta}_{\text{biased}}$ or $\boldsymbol{\beta}_{\text{random}}$ helps guide the robot to efficiently converge towards the goal amidst clutter. Next, we discuss our planner, BBoE, which incorporates this strategy.

D. Bidirectional Bundle of Edges

1) Inputs (Alg. 1): *BBoE* takes the robot's start state x_{start}, any goal state within the goal region ($x_{\text{goal}} \in Q_{goal}$), and

Algorithm 1: BBoE($x_{\text{start}}, x_{\text{goal}}, \delta_{\text{hr}}, k_{\text{bias}}, k_{\text{exploit}}, \text{N}$)

1 $G_{\text{for}} \leftarrow \{\mathbf{V}_{\text{for}} \leftarrow \{x_{\text{start}}\}, \mathbf{E}_{\text{for}} \leftarrow \{\}\}$
2 $G_{\text{rev}} \leftarrow \{\mathbf{V}_{\text{rev}} \leftarrow \{x_{\text{goal}}\}, \mathbf{E}_{\text{rev}} \leftarrow \{\}\}$
3 $\mathbf{Q} \leftarrow \{\}$, $k_{\text{exploration}} \leftarrow 1$
4 $\boldsymbol{\beta} \leftarrow$ LoadEdgeBundle()
5 **for** $k \leftarrow 1$ **to** M_{iter} **do**
6 $r_k \leftarrow \min(\gamma(\log(|\mathbf{V}_{\text{rev}}|)/|\mathbf{V}_{\text{rev}}|)^{1/d+1}, \delta_{\text{hr}})$
7 $x_{\text{rand}} \leftarrow$ RandomState()
8 $x_{\text{near}} \leftarrow$ NearestNeighbour(x_{rand}, G_{rev})
9 $\varepsilon_{\text{rev}} \leftarrow$ RevPropUsingEdgeBundle($\boldsymbol{\beta}$, x_{near}, x_{rand}, N)
10 **if** $\varepsilon_{rev} \neq$ NULL **then**
11 $x_{\text{rev}} \leftarrow \varepsilon_{\text{rev}}$.finalState()
12 $\mathbf{E}_{\text{rev}} \leftarrow \mathbf{E}_{\text{rev}} \cup \{\varepsilon_{\text{rev}}\}$
13 $\mathbf{V}_{\text{rev}} \leftarrow \mathbf{V}_{\text{rev}} \cup \{x_{\text{rev}}\}$
14 updatePriorityQueue(G_{for}, $\mathbf{Q}$, x_{rev}, r_k)
15 $q \leftarrow k_{\text{exploit}}$
16 $c_{\text{rand}} \sim U([0, 1])$
17 **if** $c_{rand} < q$ **then**
18 $\varepsilon_{\text{for}} =$ NULL
19 $x_{\text{pop}} \leftarrow$ Pop($\mathbf{Q}$)
20 **if** $x_{pop} \neq$ NULL **then**
21 $\mathbf{V}_{\text{rev_near}} \leftarrow \{x \in G_{\text{rev}}.\text{Nodes}() \mid d_{\text{X}}(x_{\text{pop},x}) \leq r_k\}$
22 $x_{\text{best}} \leftarrow \arg\min_{x \in \mathbf{V}_{\text{rev_near}}}(g(x_{\text{pop}}) + d_{\text{X}}(x_{\text{pop},x}) + h(x))$
23 $\varepsilon_{\text{for}} \leftarrow$ BestPropUsingEdgeBundle($\boldsymbol{\beta}$, x_{pop}, x_{best}, k_{bias}, N)
24 **if** $\varepsilon_{for} =$ NULL **then**
25 $x_{\text{rand}} \leftarrow$ RandomState()
26 $x_{\text{near}} \leftarrow$ NearestNeighbour(x_{rand}, G_{for})
27 $\varepsilon_{\text{for}} \leftarrow$ BestPropUsingEdgeBundle($\boldsymbol{\beta}$, x_{near}, x_{rand}, $k_{\text{exploration}}$, N)
28 **if** $c_{rand} \geq q$ **or** $\varepsilon_{for} =$ NULL **then**
29 $x_{\text{rand}} \leftarrow$ RandomState()
30 $x_{\text{near}} \leftarrow$ NearestNeighbour(x_{rand}, G_{for})
31 $\varepsilon_{\text{for}} \leftarrow$ RandomPropUsingEdgeBundle($\boldsymbol{\beta}$, x_{near})
32 **if** $\varepsilon_{for} \neq$ NULL **then**
33 $x_{\text{for}} \leftarrow \varepsilon_{\text{for}}$.finalNode()
34 $\mathbf{E}_{\text{for}} \leftarrow \mathbf{E}_{\text{for}} \cup \{\varepsilon_{\text{for}}\}$
35 $\mathbf{V}_{\text{for}} \leftarrow \mathbf{V}_{\text{for}} \cup \{x_{\text{for}}\}$
36 **if** goalRegionReached(x_{goal}, x_{for}) **then**
37 **return** Path(G_{for}, x_{start}, x_{for})
38 InsertToPriorityQueue(G_{rev}, $\mathbf{Q}$, x_{for}, r_k)
39 **return** NULL

Algorithm 2: BestPropUsingEdgeBundle($\boldsymbol{\beta}, x_{\text{near}}, x_{\text{des}}, k_{\text{bias}}, \text{N}$)

1 $\boldsymbol{\beta}_{\text{near}} \leftarrow \{\text{e} \in \boldsymbol{\beta} \mid \text{d}(x_0, x_{\text{near}}) \leq \theta,$ e untried at $x_{\text{near}}\}$
2 $\boldsymbol{\beta}_{\text{near_sorted}} \leftarrow \boldsymbol{\beta}_{\text{near}}$.SortEdges($x_{\text{des}}$)
3 $\boldsymbol{\beta}_{\text{biased}} \leftarrow \{e_{\text{i}} \in \boldsymbol{\beta}_{\text{near_sorted}} \mid i = 0, \text{N}, 2\text{N}, \ldots, \text{MN}\}$
4 $\boldsymbol{\beta}_{\text{random}} \leftarrow \{e_{\text{i}} \in \boldsymbol{\beta}_{\text{near_sorted}} \mid i \sim U(\{0, \ldots, |\boldsymbol{\beta}_{\text{near_sorted}}| - 1\})\}$, $|\boldsymbol{\beta}_{\text{random}}| =$ M+1, no repeats
5 $q \leftarrow k_{\text{bias}}$
6 $c_{\text{rand}} \sim U([0, 1])$
7 **if** $c_{rand} \leq q$ **then**
8 $\boldsymbol{\beta}_{\text{final}} \leftarrow \boldsymbol{\beta}_{\text{biased}}$
9 **else**
10 $\boldsymbol{\beta}_{\text{final}} \leftarrow \boldsymbol{\beta}_{\text{random}}$
11 **while** $\boldsymbol{\beta}_{final} \neq \boldsymbol{\phi}$ **do**
12 $e \leftarrow$ Pop($\boldsymbol{\beta}_{\text{final}}$)
13 $\varepsilon_{\text{new}} \leftarrow e$.Propagate($x_{\text{near}}$)
14 **if** ε_{new}.Success() **then**
15 **return** ε_{new}
16 **return** NULL

Algorithm 3: RandomPropUsingEdgeBundle($\boldsymbol{\beta}, x_{\text{near}}$)

1 $\boldsymbol{\epsilon}_{\text{near}} \leftarrow \{\text{e} \in \boldsymbol{\beta} \mid \text{d}(x_0, x_{\text{near}}) \leq \theta,$ e untried at $x_{\text{near}}\}$
2 $e \sim U(\boldsymbol{\epsilon}_{\text{near}})$
3 $\varepsilon_{\text{new}} \leftarrow e$.Propagate($x_{\text{near}}$)
4 **if** ε_{new}.Success() **then**
5 **return** ε_{new}
6 **return** NULL

four user-defined parameters, δ_{hr}, k_{bias}, k_{exploit} and N as input. δ_{hr} is the maximum distance within which nodes from the reverse tree have a focusing effect on the growth of the forward tree, k_{bias} is the probabilistic threshold used to decide between $\boldsymbol{\beta}_{\text{biased}}$ and $\boldsymbol{\beta}_{\text{random}}$, and N is the interval for choosing edges for $\boldsymbol{\beta}_{\text{biased}}$. k_{exploit} is the constant probability with which an iteration attempts exploitation (Line 15).

2) Initialization (Alg. 1, Lines 1–4): We use the node set $\mathbf{V}_{\text{for}}$ (initialized to x_{start}) and the edge set $\mathbf{E}_{\text{for}}$ to initialize the forward search tree G_{for} (line 1). Similarly, we use the node set $\mathbf{V}_{\text{rev}}$ (initialized to x_{goal}) and the edge set $\mathbf{E}_{\text{rev}}$ to initialize the reverse tree G_{rev} (line 2). We create a priority queue $\mathbf{Q}$ to maintain a list of prospective nodes in the forward tree to use for tree expansion. We also initialize $k_{\text{exploration}}$ to 1, which ensures that BestPropUsingEdgeBundle only uses $\boldsymbol{\beta}_{\text{biased}}$ during exploration (line 3). We conclude the initialization segment by loading a precomputed bundle of edges $\boldsymbol{\beta}$ (line 4).

3) Shrinking Neighborhood Radius r_k (Alg. 1, Line 6): Radius r_k has two objectives. It is used to determine if the trees are close enough to shift from exploration to exploitation (line 21) and to influence the priority queue update and insertion operations (lines 14 and 38). Here, d is the dimension of the state space and γ is a scenario-specific parameter.

4) Reverse Search (Alg. 1, Lines 7-14): We use the edge bundle $\boldsymbol{\beta}$ for expanding the reverse tree. The aim is to quickly explore the state space and provide a heuristic to the forward tree for goal convergence. First, we randomly sample a state (Line 7) and identify the nearest node in the reverse tree (Line 8). Then, RevPropUsingEdgeBundle propagates the nearest node towards the random state using the bundle of edges (Line 9). If the returned edge ε_{rev} is not NULL (Line 10), then we add ε_{rev} and its corresponding final state x_{rev} (Line 11) to the corresponding sets (Lines 12 and 13). We utilize x_{rev} to perform a heuristic update in $\mathbf{Q}$ using updatePriorityQueue (Line 14).

5) Forward Search (Alg. 1, Lines 15-39): The forward search is an active part of both the exploration phase (Lines 24-27 and 28-31) and exploitation phase (Lines 20-23). The exploitation probability q is the constant k_{exploit} (line 15). The value of q decides if the exploration phase or the exploitation phase is triggered (Line 17). If the exploitation phase is initiated, we check if the priority queue is empty (Line 19). If so, then exploitation isn't possible and we perform exploration instead (Lines 24-27) by calling BestPropUsingEdgeBundle (Alg. 2), where we leverage the strategy on the edge bundle $\boldsymbol{\beta}$ to expand the tree towards the desired state. If exploitation wasn't chosen in Line 17, we perform exploration in a random direction (Lines 29-31) via RandomPropUsingEdgeBundle (Alg. 3). If the priority queue isn't empty and exploitation was preferred (Line 20), we begin the exploitation process by gathering all the reverse tree nodes that are within radius r_k from x_{pop} and storing them in $\mathbf{V}_{\text{rev_near}}$ (Line 21). Then, we pick the best node amongst $\mathbf{V}_{\text{rev_near}}$(Line 22) and the edge bundle $\boldsymbol{\beta}$ aids the propagation from x_{pop} to x_{best} (Line 23). Once we obtain the new edge-node pair (Line 33), we add them to the tree (Lines 34 and 35) and check if we have reached the goal (Lines 36 and 37). If not, we add the new forward node to the priority queue $\mathbf{Q}$ if it is within r_k distance from any of the reverse tree nodes (Line 38).

6) Propagation using Edge Bundles (Alg. 2): Propagate (Alg. 2, Line 13) integrates the edge's stored control for its stored duration from the tree node x_{near} using the true

dynamics f, collision-checks the resulting trajectory, and returns it with a success flag. For the reverse tree, Propagate integrates backward in time (the duration is negated), so that a reverse node is a state from which its parent is reached under the stored control. The cost stored on a reverse node is the forward-time cost to the goal. Because the stored end state x_f describes forward propagation, it does not rank edges for backward propagation. The reverse tree therefore calls RevPropUsingEdgeBundle($\boldsymbol{\beta}, x_{\text{near}}, x_{\text{rand}}, \text{N}$) (Alg. 1, Line 9), which neither sorts nor builds $\boldsymbol{\beta}_{\text{biased}}$. It draws $\lfloor |\boldsymbol{\beta}_{\text{near}}^{\text{untried}}|/\text{N} \rfloor$ (at least one) of the node's untried neighboring edges at random for $\text{N} \geq 2$, or $\lfloor |\boldsymbol{\beta}_{\text{near}}|/5 \rfloor$ of them when $\text{N} = 1$, where 5 is a fixed constant. Here, it propagates each edge backward and keeps the one whose end state is closest to x_{rand} under d_X. These propagations per reverse expansion are included in all reported planning times. BestPropUsingEdgeBundle utilizes the edge bundle $\boldsymbol{\beta}$, user-defined values k_{bias} and N to propagate from the current state (x_{near}) to the desired state (x_{des}). k_{bias} is a number between 0 and 1 providing a probabilistic threshold for the planner to choose $\boldsymbol{\beta}_{\text{biased}}$ over $\boldsymbol{\beta}_{\text{random}}$. During exploration, k_{bias} will be hardcoded to 1 in order to impose the usage of $\boldsymbol{\beta}_{\text{biased}}$ and during exploitation, it will hold a user-defined value. N refers to the interval between sorted edges in $\boldsymbol{\beta}_{\text{biased}}$. We begin by focusing on $\boldsymbol{\beta}_{\text{near}}$, in which we store all the edges e whose start states x_0 lie within radius θ from x_{near} (Line 1). Then, we sort $\boldsymbol{\beta}_{\text{near}}$ based on the distance between each edge's end state and x_{des}, resulting in $\boldsymbol{\beta}_{\text{near_sorted}}$ (Line 2). Afterwards, we retrieve the first and every Nth element in $\boldsymbol{\beta}_{\text{near_sorted}}$ and store them in $\boldsymbol{\beta}_{\text{biased}}$ (Line 3). Furthermore, we draw M+1 edges (as many as in $\boldsymbol{\beta}_{\text{biased}}$) uniformly without replacement from $\boldsymbol{\beta}_{\text{near_sorted}}$ and store them in $\boldsymbol{\beta}_{\text{random}}$ (Line 4). We set $q = k_{\text{bias}}$ (Line 5) and use it to determine if $\boldsymbol{\beta}_{\text{random}}$ or $\boldsymbol{\beta}_{\text{biased}}$ will be used for propagation (Lines 5-10). Until a successful propagation occurs or the chosen list $\boldsymbol{\beta}_{\text{final}}$ becomes empty, we pop edges from $\boldsymbol{\beta}_{\text{final}}$ and propagate them from x_{near} (Lines 11-14), eventually returning the successful edge (Line 15). If none of the edges lead to a successful propagation, the result is NULL (Line 16).

7) Random Propagation using Edge Bundles (Alg. 3): RandomPropUsingEdgeBundle is used to quickly expand the forward tree by propagating a random neighboring edge. We store all the edges e whose start states x_0 lie within radius θ from x_{near} in $\boldsymbol{\epsilon}_{\text{near}}$ (Line 1). We randomly pick an edge from $\boldsymbol{\epsilon}_{\text{near}}$ (Line 2) and propagate it from x_{near} (Line 3). If we obtain a collision-free trajectory ε_{new}, we return it (Line 4 and 5). Otherwise, the result is NULL (Line 6).

IV. EVALUATION

A. Experimental Setup

As seen in Fig. 3, we evaluate BBoE with $k_{\text{bias}} = 0.85$ on four systems of increasing complexity: a Unicycle, a Treaded Vehicle, a Car With Trailer (CWT), and a Quadrotor. The specifications of each system are shown in Table I. The unicycle is kinematic with controls (v, ω). The treaded vehicle and the CWT are the second-order models of [5], the CWT state being $(x, y, \theta, v, \phi, \theta_1)$ (position, heading, speed, steering and trailer angle) with controls $(a, \dot{\phi})$. The quadrotor is a translational double integrator with attitude abstracted [28], i.e., state $(\mathbf{p}, \mathbf{v})$ and controls $\mathbf{a}$ with $|v_i| \leq 0.5$, $|a_i| \leq 2$. We compare BBoE against RRT [2], SST [12], GBRRT [5], BoE [8], and Db-RRT [16]. We evaluate Db-RRT-F and Db-RRT-C without their subsequent trajectory optimization step. Db-RRT's trajectory-optimization step adds hundreds of milliseconds to several seconds [16]. Since our focus is rapid initial-solution time and BBoE produces collision-free, dynamically feasible trajectories directly from search, we compare against the search component of Db-RRT in isolation. For the unicycle, we additionally evaluate Monte-Carlo (MC) and Edge-Bundle (EB) variants of RRT, SST, and GBRRT. The MC variants propagate online by generating 40 random control extensions per expansion and selecting the collision-free trajectory that terminates closest to the desired state. Likewise, the EB variants sample 40 edges from the precomputed bundle, propagate them from the current state, and select the expansion terminating closest to the desired state. Table I lists the number of edges actually used at planning time. For the unicycle, we further analyze planner performance across four difficulty levels (easy, medium, hard, and very hard) defined by obstacle count. Each level comprises 10 randomly generated environments evaluated across 50 seeds. For all systems, we generate a bundle of feasible edges, as seen in Fig. 4. Each edge is produced by sampling admissible control inputs and propagating the system dynamics over a random duration. For the unicycle, we use the same maneuver library as the GBRRT paper [5], generated by discretely sampling actions and time durations, to ensure a fair comparison. For all other systems, edges are sampled from continuous action and duration ranges. During planning, edges are transformed according to the robot's pose before propagation. Crucially, this transformation is applied only to the positional and orientation components of the state (e.g., x, y, z, θ), while the kinodynamic components (velocities, accelerations, angular rates, etc.) are preserved as-is, since rigid-body transformations are well-defined on the configuration manifold but do not extend consistently to the control space. To ensure fairness, BoE, Db-RRT, and BBoE leverage the same bundle of edges. All planners are evaluated over 50 random seeds using three metrics: i) Initial Solution Time, ii) Initial Solution Cost (Path Length), and iii) Success Rate. Each planner is assigned a fixed planning budget of 5 seconds, after which the trial is counted as a failure. On the treaded vehicle, BBoE and GBRRT both exclude reverse-tree nodes already reached by the forward tree (within 1 unit of d_X) from later exploitation targets to avoid retargeting nodes.

TABLE I

SPECIFICATIONS OF THE SYSTEMS USED IN THE EXPERIMENTS. FOR THE DISTANCE FUNCTIONS, $\mathbf{p}$ AND $\mathbf{v}$ DENOTE THE POSITION AND VELOCITY VECTORS OF THE RESPECTIVE SYSTEMS. # EDGES IS THE NUMBER OF BUNDLE EDGES USED AT PLANNING TIME.

System	State Dim.	Distance Function (d_X)	# Edges	Ref.
Unicycle	3	$\|\Delta\mathbf{p}\|$	190	[5]
Treaded	5	$\left(\|\Delta\mathbf{p}\|^2 + 0.25^2\|\Delta\mathbf{v}\|^2\right)^{1/2}$	50000	[5]
CWT	6	$\left(\|\Delta\mathbf{p}\|^2 + 0.25^2\Delta v^2\right)^{1/2}$	50000	[5]
Quadrotor	6	$\left(\|\Delta\mathbf{p}\|^2 + \|\Delta\mathbf{v}\|^2\right)^{1/2}$	100000	[28]

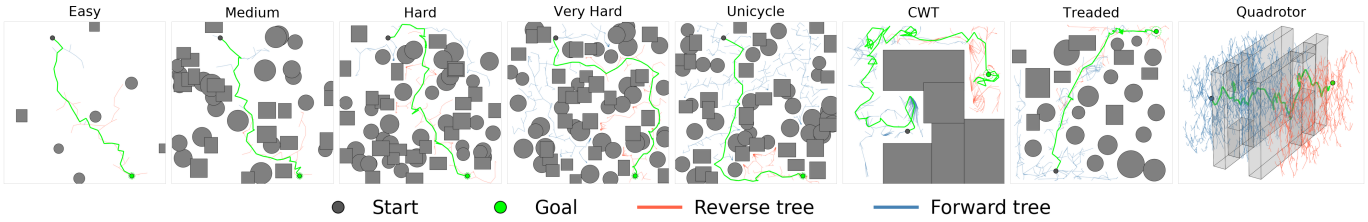


Fig. 3. Final BBoE solutions showing unicycle environments across four difficulty tiers (seed 15795) and four different dynamical systems. The forward tree (blue) traces the reverse tree (red) into the goal region. The solution path (green) consists of forward-tree edges.

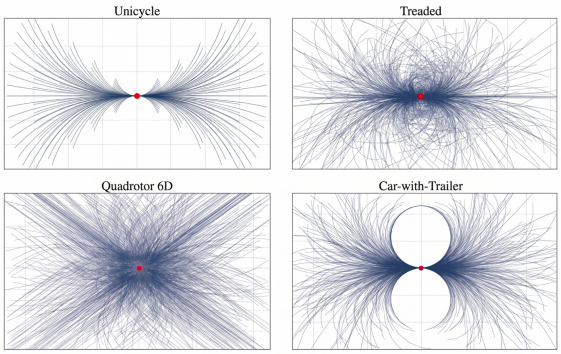


Fig. 4. The 2-dimensional projection of a subset of the edges generated for each system. Every edge starts at the origin of the XY plane. The unicycle edges are essentially the maneuver libraries used in the GBRRT paper.

B. Results

Across all systems, BBoE consistently achieves the highest/ near-highest success rates and the lowest/ near-lowest planning times. BoE and Db-RRT, despite operating on the same edge bundles, do not achieve comparable speed in most settings. Substituting online Monte-Carlo propagation with offline edge reuse yields consistent improvements for RRT, SST, and GBRRT. However, the EB variants alone do not match BBoE's performance, suggesting that the observed gains stem from the interaction between edge reuse, bidirectional search, and BBoE's edge sequencing strategy rather than from edge reuse in isolation.

TABLE II
CUMULATIVE SUCCESS RATE (%).

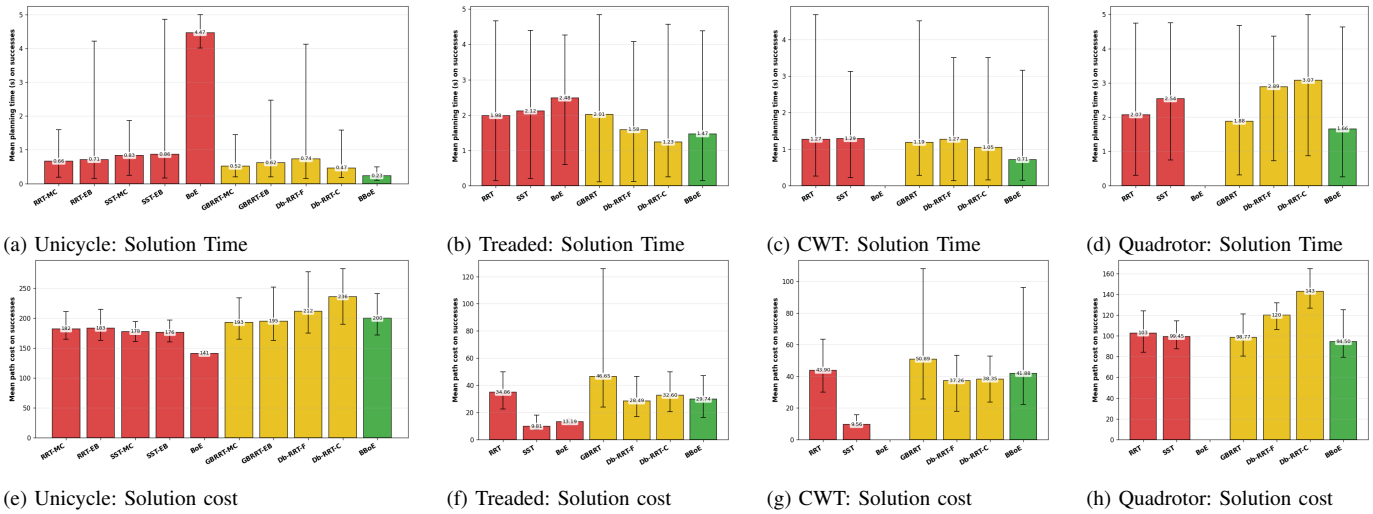
Planner	E	M	H	VH	Uni	CWT	Tr	Qd
RRT	–	–	–	–	–	96	80	90
RRT-MC	100	100	92	88	100	–	–	–
RRT-EB	100	100	100	90	100	–	–	–
SST	–	–	–	–	–	100	82	52
SST-MC	100	100	100	82	100	–	–	–
SST-EB	100	100	100	88	100	–	–	–
BoE	100	62	60	18	48	0	96	0
GBRRT	–	–	–	–	–	100	88	88
GBRRT-MC	100	100	98	100	100	–	–	–
GBRRT-EB	100	100	100	100	100	–	–	–
Db-RRT-F	100	100	90	52	100	86	78	24
Db-RRT-C	100	100	96	70	100	98	100	48
BBoE	**100**	**100**	**100**	**100**	**100**	**100**	**96**	**96**

1) Cross-System Evaluation: As seen in Fig. 5 and Table II, BBoE achieves the lowest mean planning time on three of the four systems and is competitive on the fourth. It reaches 0.23 s on the unicycle ($\sim$2$\times$ faster than the next-best Db-RRT-C at 0.47 s), 0.71 s on the CWT (vs. 1.05 s for Db-RRT-C), and 1.66 s on the 6D quadrotor in a narrow-passage environment (vs. 2.07 s for RRT and 3.07 s for Db-RRT-C). On the treaded vehicle, Db-RRT-C is faster (1.23 s vs. 1.47 s for BBoE), though all bidirectional planners on this system cluster in the 1.2–2.0 s range. Success rates follow a similar pattern: BBoE reaches 100% on the unicycle and CWT, and 96% on the treaded vehicle and quadrotor. The original BoE planner fails entirely on the CWT and quadrotor and reaches only 48% on the unicycle, demonstrating that edge reuse alone does not generalize to higher-dimensional or more constrained dynamics. BBoE produces the lowest-cost paths on the quadrotor (94.50, vs. 98.77–143 for the baselines) and competitive costs on the unicycle and treaded vehicle. Taken together, BBoE is the only planner that simultaneously achieves $\geq 96\%$ success on every system tested and the lowest planning time on three of four systems, spanning 3D to 6D state spaces and a range of dynamic constraints.

2) Four-Tier Difficulty Environments: To examine how planner performance scales with obstacle clutter, we evaluate all baselines on the unicycle system across four difficulty tiers (easy, medium, hard, very hard), with 50 random seeds per tier (Fig. 6 and Table II). On easy environments, all planners reach 100% success, and BBoE is competitive with the fastest baselines (0.10 s, vs. 0.04 s for Db-RRT-F and 0.05 s for Db-RRT-C). The picture shifts as difficulty increases. BBoE becomes the fastest planner from medium onward, achieving 0.13 s on medium, 0.18 s on hard, and 0.35 s on very-hard, while maintaining 100% success across all three tiers. The advantage widens with clutter: on very-hard environments, BBoE is roughly $1.4\times$ faster than the next-best GBRRT-EB at 0.48 s and substantially more reliable than the Db-RRT family, whose success rates collapse from 100% on easy to 52% (Db-RRT-F) and 70% (Db-RRT-C) on very-hard. Excluding BoE, path costs remain within 10–15% of the lowest-cost feasible planner across tiers. These results indicate that BBoE's relative advantage grows with environmental difficulty.

C. Ablation Studies

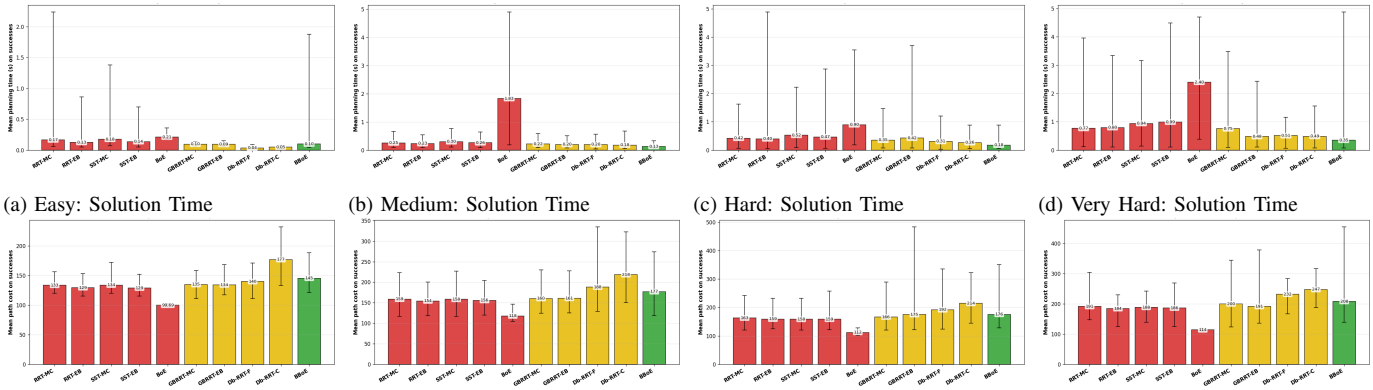
We isolate BBoE's components on the unicycle, aggregating 50 seeds across all four difficulty tiers (Table III). The edge-skipping parameter N (Table IIIa) is robust across two orders of magnitude: success is 50/50 for $N = 5$, 20 and 100 and 48/50 for $N = 1$, while planning time is lowest for $N = 20$ and 100 (0.46 s and 0.49 s) and about 0.7 s for $N = 1$ and $N = 5$. $k_{\text{bias}} = 0.85$ and θ were fixed across systems. θ is a radius on the kinodynamic components only, set to one tenth of each component's bound on every system (0.05 m/s for the quadrotor), while for the unicycle, the kinodynamic subspace is empty and every edge is a neighbor. $(\delta_{\text{hr}}, \gamma, k_{\text{exploit}})$ are inherited from GBRRT [5] and shared



(a) Unicycle: Solution Time (b) Treaded: Solution Time (c) CWT: Solution Time (d) Quadrotor: Solution Time

(e) Unicycle: Solution cost (f) Treaded: Solution cost (g) CWT: Solution cost (h) Quadrotor: Solution cost

Fig. 5. Cross-system planner evaluation on the unicycle, treaded vehicle, car-with-trailer (CWT), and quadrotor (50 seeds each). **Top row:** mean initial solution time on successes per system. **Bottom row:** mean solution cost on successes per system. BBoE achieves the lowest planning time on three of four systems and $\geq 96\%$ success on every system.



(a) Easy: Solution Time (b) Medium: Solution Time (c) Hard: Solution Time (d) Very Hard: Solution Time

(e) Easy: Solution Cost (f) Medium: Solution Cost (g) Hard: Solution Cost (h) Very Hard: Solution Cost

Fig. 6. Unicycle planner evaluation across 40 environments (10 per difficulty tier, 50 seeds each). **Top row:** Mean initial solution time on successes per difficulty. **Bottom row:** Mean solution cost on successes per difficulty.

with that baseline on each system: $(7, 14, 0.8)$, $(8, 7, 0.7)$, $(10, 8, 0.7)$ and $(6, 6, 0.95)$ for the unicycle, treaded vehicle, CWT and quadrotor. N was not fixed. The benchmarks of Sec. IV-B use $N = 5$ for the unicycle, treaded vehicle and CWT (50/50 in Table IIIa, though slower than $N = 20$) and $N = 1$ for the quadrotor, where a node has only about 40 neighboring edges against 190 for the unicycle. The biased-sampling parameter k_{bias} (Table IIIb) is flat above 0.25 (50/50 throughout). The only drop is at $k_{\text{bias}} = 0$, where pure random sampling achieves 42/50 at 2.53 s versus 50/50 at 0.65 s for $k_{\text{bias}} = 0.75$, a $\sim 4\times$ speedup confirming that goal-directed edge selection, not edge reuse alone, drives BBoE's performance. Finally, a unidirectional BBoE variant (Table IIIc) is very fast on easy–hard environments (0.12 s–0.18 s, 100% success) but degrades to 44/50 on very-hard, whereas bidirectional BBoE maintains 100% success across all tiers. Thus, bidirectional search contributes negligibly in low clutter but becomes essential as obstacle density forces single-tree expansion through narrow passages.

D. Discussion and Limitations

BBoE is not asymptotically optimal. For a fixed edge bundle, the states reachable from x_0 by sequences of neighboring edges form a countable, bundle-induced graph, and BBoE is probabilistically complete with respect to it. Every node and each of its untried neighboring edges is selected with nonzero probability, and no edge is retried from a node. Completeness over the state space holds only in the bundle-density limit [8]. BBoE performs no iterative cost descent and cannot be trapped in a local minimum in the optimization sense. Biased edges can stall when all of $\boldsymbol{\beta}_{\text{biased}}$ collides, which is why $\boldsymbol{\beta}_{\text{random}}$ exists. It fails on a *coverage gap* (i.e., when no sequence of edges threads the free space), an ill-chosen θ, dynamics not invariant under the rigid-body edge transformation, or bundle memory and neighbor-query cost, which grow with the kinodynamic dimension (Table I holds 190 to 100 000 edges, and a node has 190, 330, 22, and 40 neighbors for the unicycle, treaded vehicle, CWT and quadrotor). We confirmed the coverage-gap case on a unicycle environment whose only passage requires a 90° turn within 0.12 m of free width, while

TABLE III
BBoE ABLATIONS ON THE UNICYCLE ENVIRONMENTS. TIME/COST: MEAN [MIN, MAX] OVER SUCCESSFUL TRIALS.

(a) `num_skip_edges`

skip	S	T (s)	Cost
1	48/50	0.69 [0.26, 2.37]	184 [158, 213]
5	50/50	0.72 [0.31, 2.23]	198 [169, 246]
20	50/50	0.46 [0.14, 1.48]	201 [168, 284]
100	50/50	0.49 [0.16, 1.26]	205 [160, 246]

(b) k_{bias}

k_{bias}	S	T (s)	Cost
0.00	42/50	2.53 [0.75, 4.97]	453 [320, 712]
0.25	50/50	0.81 [0.44, 1.85]	280 [199, 441]
0.50	50/50	0.69 [0.31, 1.55]	222 [182, 329]
0.75	50/50	0.65 [0.28, 1.21]	205 [169, 260]
1.00	50/50	0.67 [0.26, 1.43]	192 [168, 222]

(c) Unidirectional BBoE

Difficulty	S	T (s)	Cost
Easy	50/50	0.16 [0.03, 0.46]	143 [123, 169]
Medium	50/50	0.12 [0.03, 0.41]	166 [134, 225]
Hard	50/50	0.18 [0.04, 0.83]	167 [132, 253]
Very Hard	44/50	0.28 [0.03, 1.45]	193 [147, 297]

the tightest turning radius in the bundle is 0.64 m. Here, BBoE fails on all 50 seeds at 5 s and on all 10 seeds at 30 s, although a continuous-control solution exists.

V. CONCLUSION AND FUTURE WORK

In this work, we presented BBoE, a bidirectional kinodynamic motion planner that leverages a precomputed bundle of edges to outperform existing baselines across various systems and environments. BBoE is the only evaluated planner to achieve $\geq$96% success across every system tested while attaining the lowest planning time on three of four, with its advantage growing alongside obstacle density. Using lazy search tactics with the edge sequencing strategy to optimize propagation in clutter, optimizing travel time, and replacing the hand-tuned k_{bias} and N with adaptive or learning-based selection, represent promising directions for future work.

REFERENCES

[1] A. Orthey, C. Chamzas, and L. E. Kavraki, "Sampling-based motion planning: A comparative review," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 7, 2023.

[2] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.

[3] W. Thomason, Z. Kingston, and L. E. Kavraki, "Motions in microseconds via vectorized sampling-based planning," in *2024 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2024, pp. 8749–8756.

[4] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*, vol. 2. IEEE, 2000, pp. 995–1001.

[5] S. Nayak and M. W. Otte, "Bidirectional sampling-based motion planning without two-point boundary value solution," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3636–3654, 2022.

[6] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

[7] J. Barraquand and J.-C. Latombe, "A monte-carlo algorithm for path planning with many degrees of freedom," in *Proceedings., IEEE International Conference on Robotics and Automation*. IEEE, 1990, pp. 1712–1717.

[8] R. Shome and L. E. Kavraki, "Asymptotically optimal kinodynamic planning using bundles of edges," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 9988–9994.

[9] S. Karaman and E. Frazzoli, "Sampling-based optimal motion planning for non-holonomic dynamical systems," in *2013 IEEE international conference on robotics and automation*. IEEE, 2013, pp. 5041–5047.

[10] D. J. Webb and J. Van Den Berg, "Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics," in *2013 IEEE international conference on robotics and automation*. IEEE, 2013, pp. 5054–5061.

[11] C. Xie, J. Van Den Berg, S. Patil, and P. Abbeel, "Toward asymptotically optimal motion planning for kinodynamic systems using a two-point boundary value problem solver," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 4187–4194.

[12] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *The International Journal of Robotics Research*, vol. 35, no. 5, pp. 528–564, 2016.

[13] Z. Littlefield and K. E. Bekris, "Efficient and asymptotically optimal kinodynamic motion planning via dominance-informed regions," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–9.

[14] A. Pasricha, Y.-S. Tung, B. Hayes, and A. Roncone, "Pokerrt: Poking as a skill and failure recovery tactic for planar non-prehensile manipulation," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4480–4487, 2022.

[15] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, 2009.

[16] J. Ortiz-Haro, W. Hönig, V. N. Hartmann, M. Toussaint, and L. Righetti, "idb-rrt: Sampling-based kinodynamic motion planning with motion primitives and trajectory optimization," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 10 702–10 709.

[17] A. Pasricha and A. Roncone, "The virtues of laziness: Multi-query kinodynamic motion planning with lazy methods," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 14 286–14 292.

[18] H. Gupta, P. Motter, A. Chakrabarty, R. Sodani, S. B. Raghu, A. Roncone, B. Hayes, and Z. Sunberg, "Efficient multi-robot motion planning with precomputed translation-invariant edge bundles," *arXiv preprint arXiv:2605.09801*, 2026.

[19] G. Briscoe-Martinez, A. Pasricha, A. Abderezaei, S. Chaganti, S. C. Vajrala, S. K. Popuri, and A. Roncone, "Exploring how non-prehensile manipulation expands capability in robots experiencing multi-joint failure," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 2943–2950.

[20] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 2002.

[21] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan, "Rrt∗-smart: Rapid convergence implementation of rrt∗ towards optimal solution," in *2012 IEEE international conference on mechatronics and automation*. IEEE, 2012, pp. 1651–1656.

[22] M. J. Bency, A. H. Qureshi, and M. C. Yip, "Neural path planning: Fixed time, near-optimal path generation via oracle imitation," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 3965–3972.

[23] A. Abderezaei, A. Pasricha, A. Klausenstock, and A. Roncone, "Clutter-aware spill-free liquid transport via learned dynamics," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024, pp. 9908–9915.

[24] N. Nechyporenko, C. Escobedo, S. Kadekodi, and A. Roncone, "Cat-rrt: Motion planning that admits contact one link at a time," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 4849–4856.

[25] M. P. Strub and J. D. Gammell, "Adaptively informed trees (ait*): Fast asymptotically optimal path planning through adaptive heuristics," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 3191–3198.

[26] M. Elbanhawi, M. Simic, and R. Jazar, "Randomized bidirectional b-spline parameterization motion planning," *IEEE Transactions on intelligent transportation systems*, vol. 17, no. 2, pp. 406–419, 2015.

[27] F. Burget, M. Bennewitz, and W. Burgard, "Bi 2 rrt*: An efficient sampling-based path planning framework for task-constrained mobile manipulation," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 3714–3721.

[28] S. Liu, N. Atanasov, K. Mohta, and V. Kumar, "Search-based motion planning for quadrotors using linear quadratic minimum time control," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2017, pp. 2872–2879.