Article

The International Journal of Robotics Research
2020, Vol. 39(5) 543–567

DOI: 10.1177/0278364919890396
journals.sagepub.com/home/ijr
SAGE

Batch Informed Trees (BIT*): Informed asymptotically optimal anytime search

Jonathan D Gammell[1], Timothy D Barfoot[2] and Siddhartha S Srinivasa[3]

Abstract

Path planning in robotics often requires finding high-quality solutions to continuously valued and/or high-dimensional problems. These problems are challenging and most planning algorithms instead solve simplified approximations. Popular approximations include graphs and random samples, as used by informed graph-based searches and anytime sampling-based planners, respectively.

Informed graph-based searches, such as A, traditionally use heuristics to search* a priori *graphs in order of potential solution quality. This makes their search efficient, but leaves their performance dependent on the chosen approximation. If the resolution of the chosen approximation is too low, then they may not find a (suitable) solution, but if it is too high, then they may take a prohibitively long time to do so.*

Anytime sampling-based planners, such as RRT, traditionally use random sampling to approximate the problem domain incrementally. This allows them to increase resolution until a suitable solution is found, but makes their search dependent on the order of approximation. Arbitrary sequences of random samples approximate the problem domain in every direction simultaneously, but may be prohibitively inefficient at containing a solution.*

This article unifies and extends these two approaches to develop Batch Informed Trees (BIT), an informed, anytime sampling-based planner. BIT* solves continuous path planning problems efficiently by using sampling and heuristics to alternately approximate and search the problem domain. Its search is ordered by potential solution quality, as in A*, and its approximation improves indefinitely with additional computational time, as in RRT*. It is shown analytically to be almost-surely asymptotically optimal and experimentally to outperform existing sampling-based planners, especially on high-dimensional planning problems.*

Keywords

Path planning, sampling-based planning, optimal path planning, heuristic search, informed search

1. Introduction

A fundamental task of any robotic system operating autonomously in dynamic and/or unknown environments is the ability to navigate between positions. The difficulty of this path planning problem depends on the number of possible robot positions (i.e., *states*) that could belong to a solution (i.e., the size of the *search space*). This search space is often continuous because robots can be infinitesimally repositioned and also often unbounded (e.g., planning outdoors) and/or high dimensional (e.g., planning for manipulation).

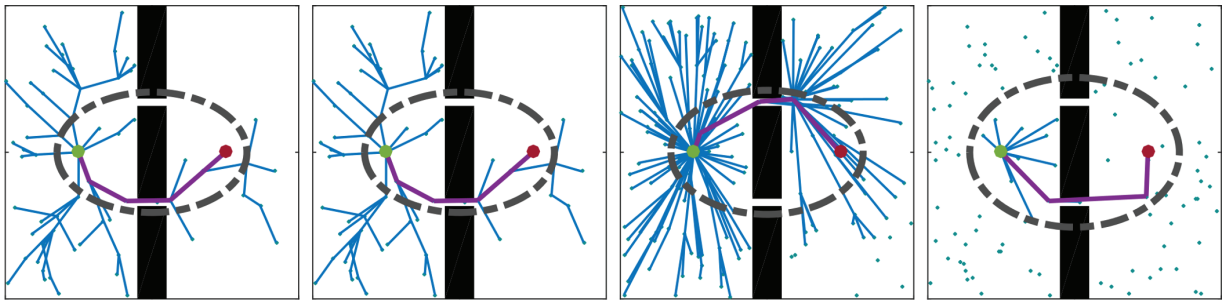
Most global path planning algorithms reduce this search space by considering a countable subset of the possible states (i.e., a *discrete approximation*). This simplifies the problem but limits formal algorithm performance to the chosen discretization. Popular discretizations in robotics include *a priori* graph- and anytime sampling-based approximations.

A priori graph-based approximations can be searched efficiently with informed algorithms, such as A* (Hart et al., 1968). These informed graph-based searches not only find the optimal solution to a representation (i.e., they are *resolution optimal*) but do so efficiently. For a chosen heuristic, A* is the *optimally efficient* search of a given graph (Hart et al., 1968).

This efficient search makes informed graph-based algorithms effective on many continuous path planning problems

[1]University of Oxford, Oxford, UK
[2]University of Toronto, Toronto, ON, Canada
[3]University of Washington, Seattle, WA, USA

Corresponding author:
Jonathan D Gammell, University of Oxford, Oxford OX2 6NN, UK.
Email: gammell@robots.ox.ac.uk



(a) RRT* (b) RRT # (c) FMT* (d) BIT*

Fig. 1. The relative *efficiency* of the searches performed by RRT* (a), RRT# (b), FMT* (c), and BIT* (d) as illustrated by their initial solution and its resulting informed set. Each planner was run until an initial solution was found and then stopped. The resulting set of states that could provide a better solution is shown as a dark gray dashed line. Search outside this informed set is provably unnecessary for the solution found by each planner and illustrates the inefficiency of the initial search of RRT*, RRT#, and FMT*. Note that RRT# finds an initial solution from the same samples as RRT* as the heuristics presented in Arslan and Tsiotras (2015) do not alter the search until a solution is found. Also note that by ordering its search on potential solution quality, BIT* does not consider any samples that cannot provide the best solution in its current approximation (i.e., batch of samples). The start and goal states of the problem are shown in green and red, respectively, while the graph built by the planner is shown in blue and the initial solution is highlighted in purple.
Note: Please refer to the online version for colour figure.

despite a dependence on the chosen approximation. Sparse approximations can be searched quickly but may only contain low-quality *continuous* solutions (if they contain any). Dense approximations alternatively contain high-quality continuous solutions (Bertsekas, 1975) but may be prohibitively expensive to search. Choosing the correct resolution *a priori* to the search is difficult and is exacerbated by the exponential growth of graph size with state dimension (i.e., the *curse of dimensionality*; Bellman, 1954, 1957).

Anytime sampling-based approximations are instead built and searched simultaneously by sampling-based algorithms such as probabilistic roadmaps (PRM; Kavraki et al., 1996), Rapidly exploring Random Trees (RRT; LaValle and Kuffner Jr., 2001), and RRT* (Karaman and Frazzoli, 2011). These sampling-based planners incrementally improve their approximation of the problem domain until a (suitable) solution is found (i.e., they have *anytime resolution*) and have probabilistic performance. The probability that they find a solution, if one exists, goes to unity with an infinite number of samples (i.e., they are *probabilistically complete*). The probability that algorithms such as RRT* and versions of PRM (e.g., s-PRM; Kavraki et al., 1998 and PRM*; Karaman and Frazzoli, 2011) asymptotically converge towards the optimum, if one exists, also goes to unity with an infinite number of samples (i.e., they are *almost-surely asymptotically optimal*; Karaman and Frazzoli, 2011).

This anytime representation makes sampling-based planners effective on many continuous path planning problems despite a dependence on sampling. Incremental planners, such as RRT*, generally search the problem domain in the order given by their random samples (Figure 1). This simultaneously expands the search into the entire sampling domain (i.e., it is *space filling*) and almost-surely converges asymptotically to the optimal solution by asymptotically finding optimal paths to *every* state. This random order is inefficient and wastes computational effort on states that are not used to find a solution. This may be prohibitively expensive in many real problems, including unbounded or high-dimensional environments.

Previous work combining the complementary advantages of these two approaches has been incomplete. Sampling concepts have been added to informed graph-based search by either sacrificing anytime resolution or by decreasing efficiency by ordering the search on metrics other than solution cost. Heuristic concepts have been added to sampling-based planners by either also sacrificing anytime resolution or by decreasing efficiency by only applying heuristics to some aspects of the search.

This article demonstrates how informed graph-based search and sampling-based planning can be directly unified and extended without compromise. It presents Batch Informed Trees (BIT*) as an example of an informed, anytime sampling-based planner that incrementally approximates continuous planning problems while searching in order of potential solution quality (Figure 2). This efficient approach avoids unnecessary computational costs while still almost-surely converging asymptotically to the optimum and finding better solutions faster than existing algorithms, especially in high state dimensions.

BIT* approximates continuous search spaces with an *edge-implicit* random geometric graph (RGG; Penrose, 2003). This RGG is defined by a set of random samples and an appropriate connection condition. The accuracy of this approximation improves as the number of samples increases and almost-surely converges towards containing the optimal solution as the number of samples approaches infinity, similar to RRT*.

This improving approximation is maintained and searched using heuristics. The initial approximation is searched in order of potential solution quality, as in A*.

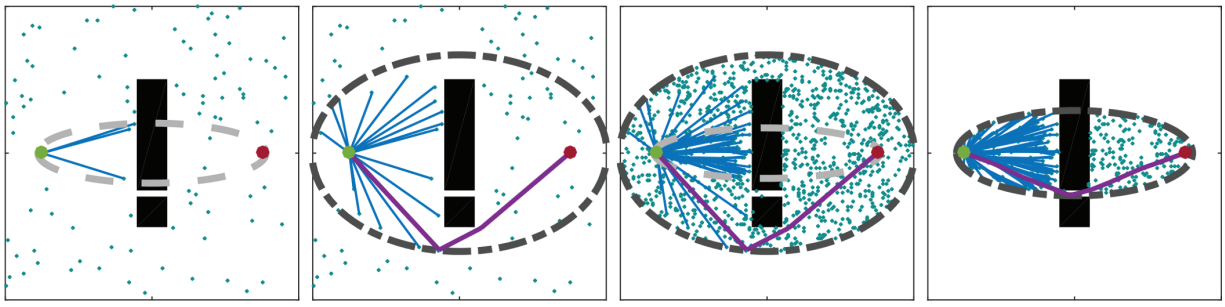


Fig. 2. A simple example of the ordered sampling-based search performed by BIT*. The start and goal states are shown in green and red, respectively, while the current solution is highlighted in purple. The set of states that can provide a better solution (the *informed* set) is shown as a dark gray dashed line, while the progress of the current batch is shown as a light gray dashed line illustrating the informed set that the current edge would define. The search of the first batch grows outwards as guided by a heuristic (a) ending when a solution is found (b). Note that ordering the search finds a solution without considering any states outside the informed set it defines. The search continues after pruning the representation and increasing its accuracy with more samples (c). This second search stops when an improved solution is found (d).
Note: Please refer to the online version for colour figure.

When it is improved the search is updated efficiently by reusing previous information, as in incremental search techniques such as Lifelong Planning A* (LPA*; Koenig et al., 2004) and Truncated LPA* (TLPA*; Aine and Likhachev, 2016). The improving approximation is focused to the *informed set* of states that could provide a better solution, as in Informed RRT* (Gammell et al., 2018, 2014c).

BIT* only requires three user-defined options, (i) a RGG-connection parameter, (ii) the heuristic function, and (iii) the number of samples per batch, and can generalize existing algorithms (Figure 3). With a single batch of samples and no heuristic, it is a search of a static approximation ordered by cost-to-come. This is exactly Dijkstra's algorithm (Dijkstra, 1959) on a RGG or a version of Fast Marching Tree (FMT*; Janson et al., 2015). With a single batch of samples and a heuristic, it is a search of a static approximation ordered by estimated solution quality. This is exactly a lazy version of A* (e.g., Cohen et al., 2014) on a RGG. With multiple batches and a heuristic, it is a truncated incremental search of a changing approximation ordered by estimated solution quality. This is equivalent to a lazy version of TLPA* on a RGG where replanning is always truncated after one propagation. With multiple batches of one sample and no heuristic, it is a construction of a tree through incremental sampling. This is equivalent to a "`steer`"-free version of RRT* where unsuccessful samples are maintained.

BIT* can also be extended to create new planners. Sorted RRT* (SORRT*) is presented as an extension of heuristically ordered search concepts to the incremental sampling of RRT* algorithms. A version of both BIT* and SORRT* are publicly available in the Open Motion Planning Library (OMPL; Şucan et al., 2012).[1]

The benefits of performing an ordered anytime search of the problem domain are demonstrated on both abstract problems and experiments for the Personal Robotic Lab's Home Exploring Robot Butler (HERB), a 14-DOF mobile manipulation platform (Srinivasa et al., 2012). The results show that BIT* finds better solutions faster than existing almost-surely asymptotically optimal planners and also RRT, especially in high dimensions. The only tested planner that found (worse) solutions faster was RRT-Connect (Kuffner Jr. and LaValle, 2000), a bidirectional version of RRT that is *not* almost-surely asymptotically optimal.

The remainder of this article is organized as follows. Section 2 presents a review of previous work combining aspects of informed search and sampling-based planning. Section 3 presents BIT* as an efficient search of an increasingly dense approximation of a continuous planning problem. Section 4 proves that BIT* is probabilistic complete and almost-surely asymptotically optimal and demonstrates its relationship to LPA* and TLPA*. Section 5 presents simple extensions of BIT* that may further improve performance in some planning situations, including SORRT*. Section 6 demonstrates the benefits of BIT* on both abstract problems and experiments for HERB. Section 7 finally presents a closing discussion and thoughts on future work.

1.1. Relationship to previous publications

This article is distinct from our prior work analyzing the necessary conditions for anytime sampling-based planners to improve an existing solution ("Informed sampling for asymptotically optimal path planning"; Gammell et al., 2018). This other publication investigates focusing the search for improvements in anytime sampling-based planners. It presents both analytical concepts that are applicable to any cost function (e.g., informed sets) and algorithmic approaches specific to problems seeking to minimize

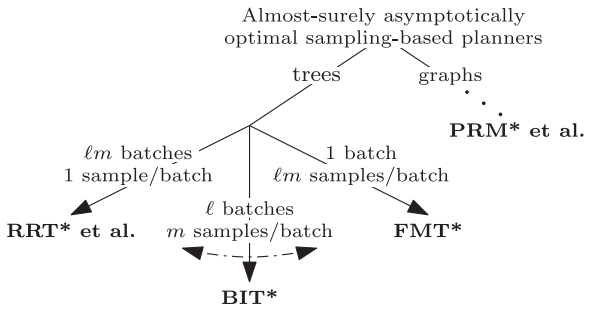


Fig. 3. A simplified taxonomy of almost-surely asymptotically optimal sampling-based planners that demonstrates the relationship between RRT*, FMT*, and BIT*. When using a batch size of a single sample, BIT* is a version of RRT*. When using a single batch consisting of multiple-samples, BIT* is a version of FMT*.

path length (e.g., direct informed sampling of the L^2 informed set).

In comparison, this article presents algorithmic approaches to order the search for both an initial solution and subsequent improvements. These techniques are applicable to any cost function, similarly to A*, and make use of the appropriate analytical results from Gammell et al. (2018). The performance of these techniques are illustrated on a number of problems seeking to minimize path length (Section 6) where we make use of the appropriate algorithmic results of Gammell et al. (2018).

1.2. Statement of contributions

This article is a continuation of ideas that were first presented at a RSS workshop (Gammell et al., 2014b) along with a supporting technical report (Gammell et al., 2014a) and were then published in Gammell et al. (2015) and Gammell (2017). We make the following specific contributions.

- We present an updated version of BIT* that avoids repeated consideration of failed edges and is a direct unification of informed graph-based search and sampling-based planning (Algorithms 1–3).
- We develop new extensions to BIT*, including including just-in-time (JIT) sampling for the direct search of unbounded problems (Algorithm 5) and SORRT* (Algorithm 6) as an application of ordered search concepts to the algorithmic simplicity of RRT*.
- We provide an expanded proof that BIT* is probabilistically complete and almost-surely asymptotically optimal (Theorems 2 and 3) and has a queue ordering equivalent to a lazy TLPA* (Lemma 4).
- We demonstrate experimentally the benefits of using ordered search concepts to combat the curse of dimensionality.

2. Prior work ordering sampling-based planners

A formal definition of the optimal path planning problem is presented to motivate a review of existing literature on the combination of informed graph-based and sampling-based concepts (Section 2.1). This prior work can be loosely classified as either incorporating sampling into informed A*-style searches (Section 2.2) or adding heuristics to incremental RRT/RRT*-style searches (Section 2.3).

2.1. Optimal path planning problem

The two most common path planning problems in robotics are those of *feasible* and *optimal* planning. Feasible path planning seeks a path that connects a start to a goal in a search space while avoiding obstacles and obeying the differential constraints of the robot. Optimal path planning seeks the feasible path that minimizes a chosen cost function (e.g., path length). By definition, solving an optimal planning problem requires solving the underlying feasible problem.

The optimal path planning problem is defined in Definition 1 similarly to Karaman and Frazzoli (2011). This definition is expressed in the state space of a robot but can be posed in other representations, including configuration space (Lozano-Pérez, 1983). The goal of these problems may be a single goal state (e.g., a pose for a mobile robot) or any state in a goal *region* (e.g., the set of joint angles that give a redundant manipulator a desired end-effector position).

Definition 1 (Optimal path planning problem). *Let $X \subseteq \mathbb{R}^n$ be the state space of the planning problem, $X_{\text{obs}} \subset X$ be the states in collision with obstacles, and $X_{\text{free}} = \text{cl}(X \backslash X_{\text{obs}})$ be the resulting set of permissible states, where $\text{cl}(\cdot)$ represents the closure of a set. Let $\mathbf{x}_{\text{start}} \in X_{\text{free}}$ be the initial state and $X_{\text{goal}} \subset X_{\text{free}}$ be the set of desired goal states. Let $\sigma : [0, 1] \to X_{\text{free}}$ be a continuous map to a sequence of states through collision-free space of bounded variation that can be executed by the robot (i.e., a collision-free, feasible path) and Σ be the set of all such non-trivial paths.*

The optimal path planning problem is then formally defined as the search for a path, $\sigma^* \in \Sigma$, that minimizes a given cost function, $c : \Sigma \to \mathbb{R}_{\geqslant 0}$, while connecting $\mathbf{x}_{\text{start}}$ to $\mathbf{x}_{\text{goal}} \in X_{\text{goal}}$,

$$\sigma^* = \underset{\sigma \in \Sigma}{\text{argmin}} \{ c(\sigma) \mid \sigma(0) = \mathbf{x}_{\text{start}}, \sigma(1) \in X_{\text{goal}} \}$$

where $\mathbb{R}_{\geqslant 0}$ is the set of non-negative real numbers.

2.2. A*-based approaches

A* is a popular planning algorithm because it is the optimally efficient search of a graph. Any other algorithm guaranteed to find the resolution-optimal solution will expand at least as many vertices when using the same heuristic

(Hart et al., 1968). This efficiency is achieved by ordering the search by potential solution quality such that the optimal solution is found by only considering states that could have provided a better solution. Applying A* to continuous path planning problems requires discretizing the search domain and significant work has incorporated sampling into this process, often to avoid the need to do so *a priori*.

Sallaberger and D'Eleuterio (1995) demonstrate the advantages of including stochasticity in the representation of a continuous state space by adding random perturbations to a regular discretization. They solve path planning problems for spacecraft and multilink robotic arms using dynamic programming and show that their technique finds better solutions than regular discretizations alone; however, the approximation is still defined *a priori* to the search.

Randomized A* (RA*; Diankov and Kuffner Jr., 2007) uses random sampling to apply A* directly to continuous planning problems. Vertices are expanded a user-specified number of times by randomly sampling possible nearby descendants. If a sample is sufficiently different than the existing states in the tree and can be reached without collision then it is added as a child of the expanded vertex. The resulting sampling-based search expands outwards from the start in order of potential solution quality but is not anytime. If a (suitable) solution is not found then the search must be restarted with a different number of samples per vertex.

Hybrid Randomized A* (HRA*; Teniente and Andrade-Cetto, 2013) performs a similar search for systems with differential constraints by sampling control inputs instead of states. Vertices are expanded a user-specified number of times by generating random control inputs and propagating them forward in time with a motion model. These new states are used both to expand a tree and rewire it when they are sufficiently similar to existing states. The vertex to expand is selected with a hybrid cost policy that considers path cost, the number of nearby vertices, and the distance to obstacles. This biases sampling into unexplored regions of the problem domain but may waste computational effort on states that are unnecessary to find the eventual solution.

Sampling-based A* (SBA*; Persson and Sharf, 2014) performs an ordered search by *iteratively* expanding vertices in a heuristically weighted version of Expansive Space Tree (EST; Hsu et al., 1999). At each iteration, a *single* random sample is generated near the vertex at the front of a priority queue. This queue is ordered on the potential solution quality of vertices and the likelihood of sampling a *unique* and *useful* sample in their neighborhoods. These likelihoods are required to bias sampling into unexplored space since vertices are never removed from the queue. They are estimated from previous collision checks and estimates of the sample density around vertices. Again, this search order on metrics other than the optimization objective may waste computational effort on states that are unnecessary to find the eventual solution.

Unlike these methods to incorporate sampling into informed graph-based search, BIT* maintains anytime performance while ordering its search only on potential solution quality. This allows it to be run indefinitely until a suitable solution is found, return incrementally better solutions, and almost-surely asymptotically converge to the optimum. This also limits its search of each representation to the set of states that were believed to be capable of providing a better solution than the one eventually found.

2.3. RRT-based approaches

RRT and RRT* solve continuous path planning problems by using incremental sampling to build a tree through obstacle-free space. This avoids *a priori* discretizations of the problem domain and allows them to be run indefinitely until a (suitable) solution is found. This also makes the search dependent on the sequence of samples (i.e., makes it random) and significant work has sought ways to use heuristics to order the search. Heuristics can also be used to focus the search, as in Anytime RRTs (Ferguson and Stentz, 2006) and Informed RRT* (Gammell et al., 2018, 2014c) but this does not change the order of the search.

Heuristically Guided RRT (hRRT; Urmson and Simmons, 2003) probabilistically includes heuristics in the RRT expansion step. Randomly sampled states are probabilistically added in proportion to their heuristic value relative to the tree. This balances the Voronoi bias of the RRT search with a preference towards expanding potentially high-quality paths. This improves performance, especially in problems with continuous cost functions (e.g., path length; Urmson and Simmons, 2003), but maintains a non-zero probability of wasting computational effort on states that are unnecessary for the eventual solution.

Karaman et al. (2011) and Akgun and Stilman (2011) both use heuristics to accelerate the convergence of RRT*. Karaman et al. (2011) remove vertices whose *current* cost-to-come plus a heuristic estimate of cost-to-go is higher than the current solution. Akgun and Stilman (2011) reject samples that are heuristically estimated to be unable to provide a better solution. These approaches both avoid unnecessary computational effort but do not alter the initial search as heuristics are not applied until a solution is found.

Kiesel et al. (2012) use a two-stage process to order sampling for RRT* in their *f-biasing* technique. A heuristic is first calculated by solving a coarse discretization of the planning problem with Dijkstra's algorithm. This heuristic is then used to bias RRT* sampling to the regions of the problem domain where solutions were found. This increases the likelihood of finding solutions quickly but maintains a non-zero sampling probability over the entire problem domain for the whole search and allows search effort to be wasted on states unnecessary for the eventual solution.

RRT# (Arslan and Tsiotras, 2013, 2015) uses heuristics to find and update a tree in the graph built incrementally by Rapidly exploring Random Graph (RRG; Karaman and Frazzoli, 2011). It efficiently maintains the optimal connection to each vertex by using LPA* to propagate changes

through the *entire* graph. This can also be implemented as policy iteration (Arslan and Tsiotras, 2016).

RRT# uses heuristics to limit this graph to regions of the problem domain that can improve an *existing* solution. This focuses the search for an improvement but does not alter the *order* in which the graph itself is constructed. As in RRT*, this graph is built in the order given by the sampling sequence using RRT-style incremental techniques. This may cause important, difficult-to-sample states to be discarded simply because they cannot currently be connected to the tree. It may also waste computational effort, especially during the search for an initial solution (Figure 1).

RRTX (Otte and Frazzoli, 2014, 2016) extends propagated rewiring to dynamic environments by limiting propagation to changes that improve the graph by more than a user-specified threshold. This ϵ *-consistency* balances the local rewiring performed by RRT* and the cascaded rewiring performed by RRT# to improve performance. As in RRT* and RRT#, it does not order the construction of the graph itself.

FMT* (Janson and Pavone, 2013; Janson et al., 2015) uses a marching method to search a batch of samples in order of increasing cost-to-come, similar to Dijkstra's algorithm. This separation of approximation and search makes the search order independent of the sampling order, but sacrifices anytime performance. Solutions are not returned until the search is finished and the search must be restarted if a (suitable) solution is not found, as with any other *a priori* discretization.

The Motion Planning Using Lower Bounds (MPLB) algorithm (Salzman and Halperin, 2015) extends FMT* with a heuristic and quasi-anytime resolution. Quasi-anytime performance is achieved by *independently* searching increasingly large batches of samples and returning the improved solutions when each individual search finishes. It was stated that this can be done efficiently by reusing information but no specific methods were presented.

Unlike these methods to incorporate informed search concepts into single-query sampling-based algorithms, BIT* orders its entire search only on potential solution quality and still maintains anytime performance. This avoids wasting computational effort by only considering states that are believed to be capable of providing the best solution given the current representation. This also returns suboptimal solutions in an anytime manner while almost-surely converging asymptotically to the optimum.

3. Batch Informed Trees (BIT*)

Any discrete set of states distributed in the state space of a planning problem, $X_{\text{samples}} \subset X$, can be viewed as a graph whose edges are given algorithmically by a transition function (an *edge-implicit* graph). When these states are sampled randomly, $X_{\text{samples}} = \{\mathbf{x} \sim \mathcal{U}(X)\}$, this graph is known as a RGG and has studied probabilistic properties (Penrose, 2003).

The connections (edges) between states (vertices) in a RGG depend on their relative geometric position. Common RGGs have edges to a specific number of each vertex's nearest neighbors (a *k*-nearest graph; Eppstein et al., 1997) or to all neighbors within a specific distance (an *r*-disc graph; Gilbert, 1961). RGG theory provides probabilistic relationships between the number and distribution of vertices, the k or r defining the graph, and specific properties such as connectivity or relative cost through the graph (Muthukrishnan and Pandurangan, 2005; Penrose, 2003).

Anytime sampling-based planners can be viewed as algorithms that construct a graph in the problem domain and search it. Some algorithms perform this construction and search simultaneously (e.g., RRT*) and others separately (e.g., FMT*) but, as in graph-based search, their performance always depends on both the accuracy of their approximation and the quality of their search. RRT* uses RGG theory to limit graph complexity while maintaining probabilistic bounds on approximation accuracy but incompletely searches the graph in the order it is constructed (i.e., performs a *random* search). RRT# and RRTX exploit the constructed graphs more thoroughly than RRT* but do not alter the order of its construction (i.e., they depend on the same *random* search of the underlying problem domain). FMT* performs a complete ordered search but uses RGG theory to define an *a priori* approximation of the problem domain (i.e., it is not anytime).

BIT* uses RGG theory to limit graph complexity while simultaneously building the graph in an anytime manner and searching it in order of potential solution quality. This is made possible by using *batches* of random samples to build an increasingly dense *edge-implicit* RGG in the informed set and using incremental search techniques to update the search (Figure 4). The anytime approximation allows BIT* to run indefinitely until a (suitable) solution is found. The ordered search avoids unnecessary computational cost by only considering states when they represent the best possible solution to a given approximation. The incremental search allows it to update its changing representation efficiently by reusing previous information.

The complete BIT* algorithm is presented in Algorithms 1–3 and Sections 3.1–3.7 with a discussion on some practical considerations presented in Section 3.8. For simplicity, the discussion is limited to a search from a single start state to a finite set of goal states with a constant batch size. This formulation can be directly extended to searches from a start or goal region, and/or with variable batch sizes. This version is publicly available in OMPL.

BIT* builds an explicit spanning tree of the implicit RGG defined by a batch of samples. The graph initially consists of only the start and goal (Algorithm 1, lines 1–5; Section 3.2) but is incrementally grown with batches of new samples during the search (Algorithm 1, lines 7–12; Section 3.3). The graph is searched in order of potential solution quality by selecting the best possible edge from a queue ordered by potential solution cost (Algorithm 1, lines 13–15; Section 3.4) and considering whether this edge improves the cost-to-come of its target vertex and could improve the current solution (Algorithm 1, lines 16–34; Section 3.5).

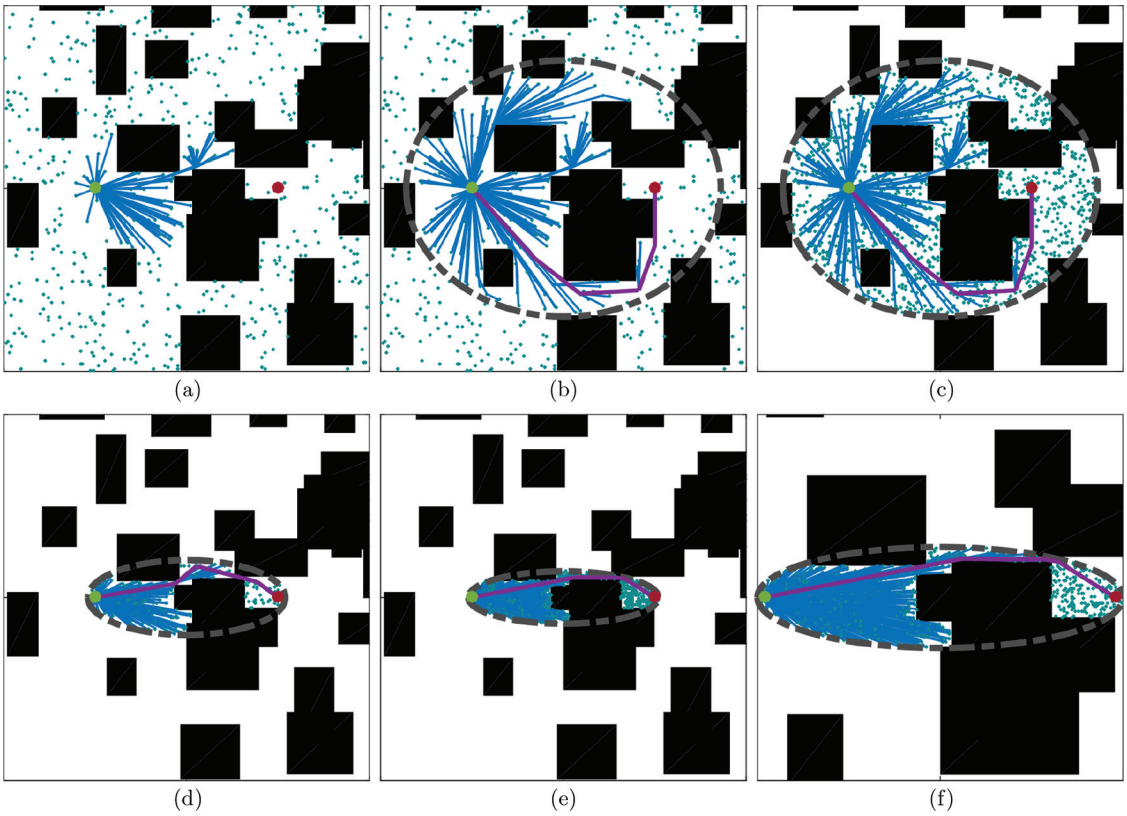


Fig. 4. An example of how BIT* uses incremental techniques to efficiently search batches of samples in order of potential solution quality. The search starts with a batch of samples uniformly distributed in the planning domain. This batch is searched outward from the start in order of potential solution quality (a). The search continues until the batch is exhausted or a solution is found that cannot be improved with the current samples (b). New samples are then added to the informed set and incremental techniques are used to continue the search (c)–(e). This results in an algorithm that performs an ordered anytime search that almost-surely asymptotically converges to the optimal solution, shown enlarged in (f). Note that BIT* orders all aspects of the search and never considers states in a batch that cannot provide the best solution (i.e., searches only inside the informed set defined by the eventual solution in the current graph).

The search continues until no edges in the implicit RGG could provide a better solution, at which point the accuracy of the approximation is increased by adding a batch of new samples and the search is resumed. This process continues indefinitely until a suitable solution is found.

3.1. Notation

The functions $\widehat{g}(\mathbf{x})$ and $\widehat{h}(\mathbf{x})$ represent admissible estimates of the cost-to-come to a state, $\mathbf{x} \in X$, from the start and the cost-to-go from a state to the goal, respectively (i.e., they bound the true costs from below). The function, $\widehat{f}(\mathbf{x})$, represents an admissible estimate of the cost of a path from $\mathbf{x}_{\text{start}}$ to X_{goal} constrained to pass through $\mathbf{x}$, i.e., $\widehat{f}(\mathbf{x}) := \widehat{g}(\mathbf{x}) + \widehat{h}(\mathbf{x})$. This estimate defines an informed set of states, $X_{\widehat{f}} := \left\{\mathbf{x} \in X \mid \widehat{f}(\mathbf{x}) < c_i\right\}$, that could provide a solution better than the current best solution cost, c_i (Gammell et al., 2018, 2014c).

Let $\mathcal{T} := (V, E)$ be an *explicit* tree with a set of vertices, $V \subset X_{\text{free}}$, and edges, $E = \{(\mathbf{v}, \mathbf{w})\}$ for some $\mathbf{v}, \mathbf{w} \in V$. The function $g_{\mathcal{T}}(\mathbf{x})$ then represents the cost-to-come to a state $\mathbf{x} \in X$ from the start vertex given the current tree, $\mathcal{T}$. A state not in the tree, or otherwise unreachable from the start, is assumed to have a cost-to-come of infinity. It is important to recognize that these two functions will always bound the unknown true optimal cost to a state, $g(\cdot)$, i.e., $\forall \mathbf{x} \in X$, $\widehat{g}(\mathbf{x}) \leqslant g(\mathbf{x}) \leqslant g_{\mathcal{T}}(\mathbf{x})$.

The functions $\widehat{c}(\mathbf{x}, \mathbf{y})$ and $c(\mathbf{x}, \mathbf{y})$ represent an admissible estimate of the cost of an edge and the true cost of an edge between states $\mathbf{x}, \mathbf{y} \in X$, respectively. Edges that intersect an obstacle are assumed to have a cost of infinity, and therefore $\forall \mathbf{x}, \mathbf{y} \in X$, $\widehat{c}(\mathbf{x}, \mathbf{y}) \leqslant c(\mathbf{x}, \mathbf{y}) \leqslant \infty$.

The notation $X \xleftarrow{+} \{\mathbf{x}\}$ and $X \xleftarrow{-} \{\mathbf{x}\}$ is used to compactly represent the set compounding operations $X \leftarrow X \cup \{\mathbf{x}\}$ and $X \leftarrow X \setminus \{\mathbf{x}\}$, respectively. As is customary, the minimum of an empty set is taken to be infinity.

3.2. Initialization (Algorithm 1, lines 1–5)

BIT* begins searching a planning problem with the start, $\mathbf{x}_{\text{start}}$, in the spanning tree, $\mathcal{T} := (V, E)$, and the goal states, X_{goal}, in the set of unconnected states, X_{unconn} (Algorithm 1, lines 1–2). This defines an implicit RGG whose vertices consist of all states (i.e., $V \cup X_{\text{unconn}}$) and whose edges are defined by a distance function and an appropriate connection limit. When the goal is a continuous region of the

Algorithm 1: BIT*$\left(\mathbf{x}_{\text{start}} \in X_{\text{free}},\ X_{\text{goal}} \subset X_{\text{free}}\right)$

```
1  V ← {x_start};  E ← ∅;  T = (V, E);                                   ┐
2  X_unconn ← X_goal;                                                    │
3  Q_V ← V;  Q_E ← ∅;                                                    │ §3.2
4  V_sol'n ← V ∩ X_goal;  V_unexpnd ← V;  X_new ← X_unconn;              │
5  c_i ← min_{v_goal ∈ V_sol'n} {g_T(v_goal)} ;                          ┘
6  repeat
7      if Q_E ≡ ∅ and Q_V ≡ ∅ then                                       ┐
8          X_reuse ← Prune(T, X_unconn, c_i);                            │
9          X_sampling ← Sample(m, x_start, X_goal, c_i);                 │ §3.3
10         X_new ← X_reuse ∪ X_sampling;                                 │
11         X_unconn ←+ X_new;                                            │
12         Q_V ← V;                                                      ┘
13     while BestQueueValue(Q_V) ≤                                       ┐
14             BestQueueValue(Q_E) do                                    │ §3.4
           ExpandNextVertex(Q_V, Q_E, c_i);                              │
15     (v_min, x_min) ← PopBestInQueue(Q_E);                             ┘
16     if g_T(v_min) + ĉ(v_min, x_min) + ĥ(x_min) < c_i then             ┐
17         if g_T(v_min) + ĉ(v_min, x_min) < g_T(x_min) then             │
18             c_edge ← c(v_min, x_min);                                 │
19             if g_T(v_min) + c_edge + ĥ(x_min) < c_i then              │
20                 if g_T(v_min) + c_edge < g_T(x_min) then              │
21                     if x_min ∈ V then                                 │
22                         v_parent ← Parent(x_min);                     │
23                         E ←− {(v_parent, x_min)};                     │
24                     else                                              │
25                         X_unconn ←− {x_min};                          │
26                         V ←+ {x_min};                                 │ §3.5
27                         Q_V ←+ {x_min};                               │
28                         V_unexpnd ←+ {x_min};                         │
29                         if x_min ∈ X_goal then                        │
30                             V_sol'n ←+ {x_min};                       │
31                     E ←+ {(v_min, x_min)};                            │
32                     c_i ← min_{v_goal ∈ V_sol'n} {g_T(v_goal)} ;      │
33     else                                                              │
34         Q_E ← ∅;  Q_V ← ∅;                                            ┘
35 until STOP;
36 return T;
```

problem domain it will need to be discretized (e.g., sampled) before adding to the set of unconnected states.

The explicit spanning tree of this edge-implicit RGG is built using two queues, a vertex expansion queue, $\mathcal{Q}_V$, and an edge evaluation queue, $\mathcal{Q}_E$. These queues are sorted in order of potential solution quality through the current tree. Vertices in the vertex queue, $\mathbf{v} \in \mathcal{Q}_V$, are ordered by the sum of their current cost-to-come and an estimate of their cost-to-go, $g_{\mathcal{T}}(\mathbf{v}) + \widehat{h}(\mathbf{v})$. Edges in the edge queue, $(\mathbf{v}, \mathbf{x}) \in \mathcal{Q}_E$, are sorted by the sum of the current-cost-to-come of their source vertex, an estimate of the edge cost, and an estimate of the cost-to-go of their target vertex, $g_{\mathcal{T}}(\mathbf{v}) + \widehat{c}(\mathbf{v}, \mathbf{x}) + \widehat{h}(\mathbf{x})$. Ties are broken in the vertex queue in favor of entries with the lowest cost-to-come through the current tree, $g_{\mathcal{T}}(\mathbf{v})$, and in the edge queue in favor of the lowest cost-to-come through the current tree and estimated edge cost, $g_{\mathcal{T}}(\mathbf{v}) + \widehat{c}(\mathbf{v}, \mathbf{x})$, and then the cost-to-come through the current tree, $g_{\mathcal{T}}(\mathbf{v})$. These queues are initialized to contain all the vertices in the tree and an empty queue, respectively (Algorithm 1, line 3).

To improve search efficiency, BIT* tracks the vertices in the goal region, $V_{\text{sol'n}}$, the vertices that have never been expanded, V_{unexpnd}, the samples newly created during this batch, X_{new}, and the current best solution, c_i. These are initialized to any vertices already in the solution (empty in all but the most trivial planning problems), the existing vertices, the existing samples, and the current best solution, respectively (Algorithm 1, lines 4–5).

Initialized, BIT* now searches the continuous planning problem by alternately building increasingly accurate implicit RGG approximations (Section 3.3) and searching these representations for explicit solutions in order of potential solution quality (Sections 3.4 and 3.5).

3.3. Batch addition (Algorithm 1, lines 7–12)

BIT* alternates between building an increasingly dense approximation of the continuous planning problem and searching this representation for a solution. The approximation is updated whenever it has been completely searched (i.e., both queues are empty; Algorithm 1, line 7) by removing unnecessary states and adding a batch of new samples. This avoids the computational cost of representing regions of the problem domain that cannot provide a better solution while increasing the accuracy of approximating the regions that can (i.e., the informed set). This improving approximation allows BIT* to almost-surely converge asymptotically to the optimal solution.

The approximation is pruned to the informed set by removing any states or edges that cannot improve the *current* solution (Algorithm 1, line 8; Section 3.7). This reduces unnecessary complexity but may disconnect vertices in the informed set that cannot improve the solution solely because of their current connections. These vertices are recycled as additional "new" samples in the batch so that they may be reconnected later if better connections are found.

The approximation is improved by adding m new randomly generated samples from the informed set (Algorithm 1, line 9). This can be accomplished with direct informed sampling (Gammell et al., 2018, 2014c) or advanced rejection sampling (e.g., Kunz et al., 2016) as appropriate.

Each batch of states are labeled as X_{new} for the duration of that batch's search (Algorithm 1, line 10). This set is used to improve search efficiency and consists of both the newly generated samples and the recycled disconnected vertices. BIT* adds these new states to the set of unconnected states and initializes the vertex queue with all the vertices in the tree (Algorithm 1, lines 11–12) to restart the search (Sections 3.4 and 3.5).

3.4. Edge selection (Algorithm 1, lines 13–15)

Graph-based search techniques often assume that finding and evaluating vertex connections is computationally

inexpensive (e.g., given explicitly). This is not true in sampling-based planning since finding vertex connections (e.g., the edges in the edge-implicit RGG) requires performing a nearest-neighbor search and evaluating them requires checking for collisions and solving two-point boundary-value problems (two-point BVPs), such as differential constraints. BIT* avoids these computational costs until required by using a lazy search procedure that delays both finding and evaluating connections in the RGG. Similar lazy techniques can be found in both advanced graph-based search and sampling-based planners (Bohlin and Kavraki, 2000; Branicky et al., 2001; Cohen et al., 2014; Hauser, 2015; Helmert, 2006; Salzman and Halperin, 2016; Sànchez and Latombe, 2002).

Connections are found by using a vertex queue, $\mathcal{Q}_V$, ordered by potential solution quality. This vertex queue delays processing a vertex (i.e., performing a nearest-neighbor search) until its outgoing connections *could* be part of the best solution to the current graph. Connections from all vertices are evaluated by using an edge queue, $\mathcal{Q}_E$, also ordered by potential solution quality. This edge queue delays evaluating an edge (i.e., performing collision checks and solving two-point BVPs) until it *could* be part of the best solution to the current graph.

A vertex in the vertex queue could be part of the best solution when it could provide an outgoing edge better than the best edge in the edge queue. When the heuristic is consistent (e.g., the L^2 norm) the queue value of a vertex, $\mathbf{v} \in \mathcal{Q}_V$, is a lower-bounding estimate of the queue value of its outgoing edges,

$$\forall \mathbf{x} \in X,\ g_T(\mathbf{v}) + \widehat{h}(\mathbf{v}) \leqslant g_T(\mathbf{v}) + \widehat{c}(\mathbf{v}, \mathbf{x}) + \widehat{h}(\mathbf{x})$$

The best edge at any iteration can therefore be found by processing the vertex queue until it is worse than the edge queue (Algorithm 1, line 13). This process of removing a vertex from the vertex queue and placing its outgoing edges in the edge queue is referred to as *expanding* a vertex (Algorithm 1, line 14; Section 3.6). Once all necessary vertices are expanded, the best edge in the queue, $(\mathbf{v}_{\min}, \mathbf{x}_{\min})$, is removed (Algorithm 1, line 15) and used for this iteration of the search (Section 3.5).

The functions `BestQueueValue(·)` and `PopBestInQueue(·)` return the value of the element at the front of a queue and pop the element off the front of a queue, respectively.

3.5. Edge processing (Algorithm 1, lines 16–34)

BIT* also uses heuristics to avoid expensive calculations when evaluating the best edge, $(\mathbf{v}_{\min}, \mathbf{x}_{\min})$. An edge is added to the spanning tree if and only if:

1. an *estimate* of its cost *could* provide a better *solution*, given the current tree (Algorithm 1, line 16),

$$g_T(\mathbf{v}_{\min}) + \widehat{c}(\mathbf{v}_{\min}, \mathbf{x}_{\min}) + \widehat{h}(\mathbf{x}_{\min}) < c_i$$

2. an *estimate* of its cost *could* improve the *current tree* (Algorithm 1, line 17),

$$g_T(\mathbf{v}_{\min}) + \widehat{c}(\mathbf{v}_{\min}, \mathbf{x}_{\min}) < g_T(\mathbf{x}_{\min})$$

3. its *real* cost *could* provide a better *solution*, given the current tree (Algorithm 1, line 19),

$$g_T(\mathbf{v}_{\min}) + c(\mathbf{v}_{\min}, \mathbf{x}_{\min}) + \widehat{h}(\mathbf{x}_{\min}) < c_i$$

4. its *real* cost *will* improve the *current tree* (Algorithm 1, line 20),

$$g_T(\mathbf{v}_{\min}) + c(\mathbf{v}_{\min}, \mathbf{x}_{\min}) < g_T(\mathbf{x}_{\min}).$$

For collision-free edges, conditions 1 and 3 are always true in the absence of a solution, while conditions 2 and 4 are always true when the target of the edge, $\mathbf{x}_{\min}$, is not in the spanning tree.

Checking if the edge could ever provide a better solution or improve the current tree (conditions 1 and 2) allows BIT* to reject edges without calculating their true cost (Algorithm 1, lines 16–17). Condition 1 also provides a stopping condition for searching the current RGG. When an edge fails this condition so does the entire queue and both queues can be cleared to start a new batch (Algorithm 1, line 34). If the edge fails condition 2 it is discarded and the iteration finishes. If the edge passes both these conditions its true cost is calculated by performing a collision check and solving any two-point BVPs (Algorithm 1, line 18). Note that edges in collision are considered to have infinite cost.

Checking if the real edge could provide a better solution given the current tree (condition 3), allows BIT* to reduce tree complexity by rejecting edges that could never improve the current solution (Algorithm 1, line 19). Checking if the real edge improves the current tree (condition 4), assures the cost-to-come of the explicit tree decreases monotonically (Algorithm 1, line 20). If the edge fails either of these conditions it is discarded and the iteration finishes.

An edge passing all of these conditions is added to the spanning tree. If the target vertex is already connected (Algorithm 1, line 21), then the edge represents a *rewiring* and the current edge must be removed (Algorithm 1, lines 22–23). Otherwise, the edge represents an *expansion* of the tree and the target vertex must be moved from the set of unconnected states to the set of vertices, inserted into the vertex queue for future expansion, and marked as a never-expanded vertex (Algorithm 1, lines 25–28). The new vertex is also added to the set of vertices in the goal region if appropriate (Algorithm 1, lines 29–30).

The new edge is then finally added to the tree (Algorithm 1, line 31) and the current best solution is updated as necessary (Algorithm 1, line 32). The search then continues by selecting the next edge in the queue (Section 3.4) or increasing the approximation accuracy if the current RGG cannot provide a better solution (Section 3.3).

Algorithm 2: `ExpandNextVertex`$(\mathcal{Q}_V \subseteq V,\ \mathcal{Q}_E \subseteq V \times (V \cup X),\ c_i \in \mathbb{R}_{\geqslant 0})$

1 $\mathbf{v}_{\min} \leftarrow \texttt{PopBestInQueue}\,(\mathcal{Q}_V)$;
2 **if** $\mathbf{v}_{\min} \in V_{\text{unexpnd}}$ **then**
3 $\quad X_{\text{near}} \leftarrow \texttt{Near}\,(X_{\text{unconn}}, \mathbf{v}_{\min}, r_{\text{BIT}^*})$;
4 **else**
5 $\quad X_{\text{near}} \leftarrow \texttt{Near}\,(X_{\text{new}} \cap X_{\text{unconn}}, \mathbf{v}_{\min}, r_{\text{BIT}^*})$;
6 $\mathcal{Q}_E \xleftarrow{+} \left\{(\mathbf{v}_{\min}, \mathbf{x}) \in V \times X_{\text{near}} \;\middle|\; \widehat{g}\,(\mathbf{v}_{\min}) + \widehat{c}\,(\mathbf{v}_{\min}, \mathbf{x}) + \widehat{h}\,(\mathbf{x}) < c_i\right\}$;
7 **if** $\mathbf{v}_{\min} \in V_{\text{unexpnd}}$ **then**
8 $\quad V_{\text{near}} \leftarrow \texttt{Near}\,(V, \mathbf{v}_{\min}, r_{\text{BIT}^*})$;
9 $\quad \mathcal{Q}_E \xleftarrow{+} \left\{(\mathbf{v}_{\min}, \mathbf{w}) \in V \times V_{\text{near}} \;\middle|\; (\mathbf{v}_{\min}, \mathbf{w}) \notin E,\ \widehat{g}\,(\mathbf{v}_{\min}) + \widehat{c}\,(\mathbf{v}_{\min}, \mathbf{w}) + \widehat{h}\,(\mathbf{w}) < c_i,\ \widehat{g}\,(\mathbf{v}_{\min}) + \widehat{c}\,(\mathbf{v}_{\min}, \mathbf{w}) < g_{\mathcal{T}}\,(\mathbf{w})\right\}$;
10 $\quad V_{\text{unexpnd}} \xleftarrow{-} \{\mathbf{v}_{\min}\}$;

3.6. Vertex expansion (Algorithm 1, line 14; Algorithm 2)

The function `ExpandNextVertex`$(\mathcal{Q}_V, \mathcal{Q}_E, c_i)$ removes the front of the vertex queue (Algorithm 2, line 1) and adds its outgoing edges in the RGG to the edge queue. The RGG is defined using the results of Karaman and Frazzoli (2011) to limit graph complexity while maintaining almost-sure asymptotic convergence to the optimum. Edges exist between a vertex and the k_{BIT^*}-closest states or all states within a distance of r_{BIT^*}, with

$$r_{\text{BIT}^*} > r^*_{\text{BIT}^*},$$
$$r^*_{\text{BIT}^*} := \left(2\left(1+\frac{1}{n}\right)\left(\frac{\min\left\{\lambda(X), \lambda\left(X_f^{\widehat{}}\right)\right\}}{\zeta_n}\right)\left(\frac{\log(|V|+|X_{\text{unconn}}|-m)}{|V|+|X_{\text{unconn}}|-m}\right)\right)^{\frac{1}{n}} \tag{1}$$

and

$$k_{\text{BIT}^*} > k^*_{\text{BIT}^*}$$
$$k^*_{\text{BIT}^*} := e\left(1+\frac{1}{n}\right)\log(|V|+|X_{\text{unconn}}|-m) \tag{2}$$

where $|\cdot|$ is the cardinality of a set, m is the number of samples added in the last batch, $\lambda(\cdot)$ is the Lebesgue measure of a set (e.g., the *volume*), and ζ_n is the Lebesgue measure of an n-dimensional unit ball. Recent work has presented different expressions (Janson et al., 2015) and expressions for non-Euclidean spaces (Kleinbort et al., 2016).

This connection limit is calculated from the cardinality of the graph *minus* the m new samples to simplify proving almost-sure asymptotic optimality (Section 4). This lower bound will be large for the initial sparse batches but it can be thresholded with a maximum edge length, as is done by RRT*, i.e., $r'_{\text{BIT}^*} := \min\{r_{\max}, r_{\text{BIT}^*}\}$. The function `Near` returns the states that meet the selected RGG connection criterion for a given vertex.

Every vertex in the tree is either expanded or pruned in every batch. Processing all the outgoing edges from vertices would result in BIT* repeatedly considering the same previously rejected edges. This can be avoided by using the sets of never-expanded vertices, V_{unexpnd}, and new samples, X_{new}, to add only *previously unconsidered* edges to the edge queue.

Whether edges to unconnected samples are new depends on whether the source vertex has previously been expanded (Algorithm 2, line 2). If it has not been expanded then none of its outgoing connections have been considered and all nearby unconnected samples are potential descendants (Algorithm 2, line 3). If it has been expanded then any connections to "old" unconnected samples have already been considered and rejected and only the "new" samples are considered as potential descendants (Algorithm 2, line 5). The subset of these potential edges that could improve the current solution are added to the queue in both situations (Algorithm 2, line 6).

Whether edges to connected samples (i.e., *rewirings*) are new also depends on whether the source vertex has been expanded (Algorithm 2, line 7). If it has not been expanded, then all nearby connected vertices are considered as potential descendants (Algorithm 2, line 8). The subset of these potential edges that could improve the current solution *and* the current tree are added to the queue (Algorithm 2, line 9) and the vertex is then marked as expanded (Algorithm 2, line 10).

If a vertex has previously been expanded, then no rewirings are considered. Improvements in the tree may now allow a previously considered edge to improve connected vertices but considering these connections would require repeatedly reconsidering infeasible edges. As in RRT*, this lack of *propagated rewiring* has no effect on almost-sure asymptotic optimality.

3.7. Graph pruning (Algorithm 1, line 8; Algorithm 3)

The function, `Prune`$(\mathcal{T}, X_{\text{unconn}}, c_i)$, reduces the complexity of both the approximation of the continuous planning problem (i.e., the implicit RGG) and its search (i.e., the explicit spanning tree) by limiting them to the informed set. It removes any states that can *never* provide a better solution and disconnects any vertices that cannot provide a better solution *given the current tree*. Disconnected vertices that *could* improve the solution with a better connection are reused as new samples in the next batch to maintain uniform sample density in the informed set. This assures that every vertex is either expanded or pruned in each batch as assumed by `ExpandNextVertex` to avoid reconsidering edges (Section 3.6).

The set of recycled vertices is initialized as an empty set (Algorithm 3, line 1) and all unconnected states that cannot provide a better solution (i.e., are not members of the

Algorithm 3: Prune($\mathcal{T} = (V, E)$, $X_{\text{unconn}} \subset X, c_i \in \mathbb{R}_{\geqslant 0}$)

1 $X_{\text{reuse}} \leftarrow \emptyset$;
2 $X_{\text{unconn}} \xleftarrow{-} \left\{ \mathbf{x} \in X_{\text{unconn}} \;\middle|\; \widehat{f}(\mathbf{x}) \geq c_i \right\}$;
3 **forall** $\mathbf{v} \in V$ **in order of increasing** $g_{\mathcal{T}}(\mathbf{v})$ **do**
4 **if** $\widehat{f}(\mathbf{v}) > c_i$ **or** $g_{\mathcal{T}}(\mathbf{v}) + \widehat{h}(\mathbf{v}) > c_i$ **then**
5 $V \xleftarrow{-} \{\mathbf{v}\}$; $V_{\text{sol'n}} \xleftarrow{-} \{\mathbf{v}\}$; $V_{\text{unexpnd}} \xleftarrow{-} \{\mathbf{v}\}$;
6 $\mathbf{v}_{\text{parent}} \leftarrow \texttt{Parent}(\mathbf{v})$;
7 $E \xleftarrow{-} \{(\mathbf{v}_{\text{parent}}, \mathbf{v})\}$;
8 **if** $\widehat{f}(\mathbf{v}) < c_i$ **then**
9 $X_{\text{reuse}} \xleftarrow{+} \{\mathbf{v}\}$;
10 **return** X_{reuse};

informed set) are removed (Algorithm 3, line 2). The connected vertices are then incrementally pruned in order of increasing cost-to-come (Algorithm 3, line 3) by identifying those that can never provide a better solution or improve the solution given the *current* tree (Algorithm 3, line 4). Vertices that fail either condition are removed from the tree by disconnecting their incoming edge and removing them from the vertex set and any labeling sets (Algorithm 3, lines 5–7).

Any disconnected vertex that could provide a better solution (i.e., is a member of the informed set) is reused as a sample in the next batch (Algorithm 3, lines 8–9). This maintains uniform sample density in the informed set and assures that vertices will be reconnected if future improvements allow them to provide a better solution. This set of disconnected vertices is returned to BIT* at the end of the pruning procedure (Algorithm 3, line 10).

3.8. Practical considerations

Algorithms 1–3 describe a generic version of BIT* and leave room for a number of practical improvements depending on the specific implementation.

Searches (e.g., Algorithm 2, line 3) can be implemented efficiently with appropriate datastructures that do not require an exhaustive global search (e.g., *k*-d trees or randomly transformed grids; Kleinbort et al., 2015). Pruning (Algorithm 1, line 8; Algorithm 3) is computationally expensive and should only occur when a new solution has been found or limited to *significant* changes in solution cost.

In an object-oriented programming language, many of the sets (e.g., X_{new}, V_{unexpnd}) can be implemented more efficiently as labels. The cost-to-come to a state in the current tree, $g_{\mathcal{T}}(\cdot)$, can also be implemented efficiently using back pointers.

While queues can be implemented efficiently by using containers that sort on insertion, the value of elements in the vertex and edge queues will change when vertices are rewired. There appears to be little practical difference between efficiently re-sorting the affected elements in these queues and only lazily re-sorting the queue before finishing to assure no elements have been missed.

Depending on the datastructure used for the edge queue, it may be beneficial to remove unnecessary entries when a new edge is added to the spanning tree, i.e., by adding

$$\mathcal{Q}_E \xleftarrow{-} \{(\mathbf{v}, \mathbf{x}_{\min}) \in \mathcal{Q}_E \mid \widehat{g}(\mathbf{v}) + \widehat{c}(\mathbf{v}, \mathbf{x}_{\min}) \geqslant g_{\mathcal{T}}(\mathbf{x}_{\min})\}$$

after Algorithm 1, line 31.

4. Analysis

BIT* performance is analyzed theoretically using the results of Karaman and Frazzoli (2011). It is shown to be probabilistically complete (Theorem 2) and almost-surely asymptotically optimal (Theorem 3). Its search ordering is also shown to be equivalent to a lazy version of the ordering used in LPA* and TLPA* (Lemma 4).

Theorem 2 (Probabilistic completeness of BIT*). *The probability that* BIT* *finds a feasible solution to a given path planning problem, if one exists, when given infinite samples is unity*,

$$\liminf_{q\to\infty} P\big(\sigma_{q,\,\text{BIT}^*} \in \Sigma,\, \sigma_{q,\,\text{BIT}^*}(0) = \mathbf{x}_{\text{start}},\, \sigma_{q,\,\text{BIT}^*}(1) \in X_{\text{goal}}\big) = 1$$

where q is the number of samples, $\sigma_{q,\,\text{BIT}^*}$ is the path found by BIT from those samples, and Σ is the set of all feasible, collision-free paths.*

Proof. Proof of Theorem 2 follows from the proof of almost-sure asymptotic optimality (Theorem 3). □

Theorem 3 (Almost-sure asymptotic optimality of BIT*). *The probability that BIT* converges asymptotically towards the optimal solution of a given path planning problem, if one exists, when given infinite samples is unity*,

$$P\left(\limsup_{q\to\infty} c\big(\sigma_{q,\,\text{BIT}^*}\big) = c(\sigma^*)\right) = 1$$

where q is the number of samples, $\sigma_{q,\,\text{BIT}^*}$ is the path found by BIT from q samples and σ^* is optimal solution to the planning problem.*

Proof. Theorem 3 is proven by showing that BIT* considers at least the same edges as RRT* for a sequence of states, $X_{\text{samples}} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_q)$, and connection limit, $r_{\text{BIT}^*} \geqslant r_{\text{RRT}^*}$.

RRT* incrementally builds a tree from a sequence of samples. For each state in the sequence, $\mathbf{x}_k \in X_{\text{samples}}$, it considers the neighborhood of earlier states that are within the connection limit,

$$X_{\text{near},\,k} := \left\{ \mathbf{x}_j \in X_{\text{samples}} \mid j < k, \left\|\mathbf{x}_k - \mathbf{x}_j\right\|_2 \leqslant r_{\text{RRT}^*} \right\}$$

It selects the connection from this neighborhood that minimizes the cost-to-come of the state and then evaluates the ability of connections from this state to reduce the cost-to-come of the other states in the neighborhood.

Given the same sequence of states, BIT* groups them into batches of samples, $X_{\text{samples}} = (Y_1, Y_2, \ldots, Y_\ell)$, where each batch is a set of $m < q$ samples, e.g., $Y_1 := \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$. It incrementally builds a tree by processing this batched sequence of samples. For each state in the sequence, $\mathbf{y} \in Y_k$, it considers the neighborhood of states from the same or earlier batches within the connection limit,

$$X_{\text{near},k} := \left\{\mathbf{x} \in Y_j \mid j \leqslant k,\ \|\mathbf{y} - \mathbf{x}\|_2 \leqslant r_{\text{BIT}^*}\right\}$$

It adds the edge in this neighborhood to the tree that minimizes the cost-to-come of the state and considers all the outgoing edges to connect its neighbors. This set of edges contains all those considered by RRT* for an equivalent connection limit, $r_{\text{BIT}^*} \geqslant r_{\text{RRT}^*}$, given that $m \geqslant 1$.

As (1) uses the same connection radius for a batch that RRT* would use for the first sample in the batch and the connection radius of both are monotonically decreasing, this shows that BIT* considers at least the same edges as RRT*. From these edges, BIT* selects those that improve the cost-to-come of the target state and could currently provide a better solution as in Karaman et al. (2011). It is therefore almost-surely asymptotically optimal as stated in Theorem 3. □

BIT* searches the RGG in order of potential solution quality using an edge queue. This is shown to be equivalent to the vertex queue ordering used by LPA*/TLPA* with a *lazy* approximation of incoming edge costs (Lemma 4). The search itself is not equivalent to LPA* as BIT* does not reconsider outgoing connections from rewired vertices (i.e., it does not propagate rewirings). It is instead a version of TLPA*.

Lemma 4 (Equivalent queue ordering of BIT* and LPA*/TLPA*). *The edge ordering in* BIT* *that uses first the sum of a vertex's estimated cost-to-go, estimated incoming edge cost, and current cost-to-come of its parent*,

$$g_{\mathcal{T}}(\mathbf{u}) + \widehat{c}(\mathbf{u}, \mathbf{v}) + \widehat{h}(\mathbf{v})$$

then the estimated cost-to-come of the vertex,

$$g_{\mathcal{T}}(\mathbf{u}) + \widehat{c}(\mathbf{u}, \mathbf{v})$$

and then the cost-to-come of its parent,

$$g_{\mathcal{T}}(\mathbf{u})$$

is equivalent to the vertex ordering in LPA (Koenig et al., 2004) and TLPA* (Aine and Likhachev, 2016).*

Proof. LPA* and TLPA* use a queue of vertices ordered lexicographically first on the solution cost constrained to go through the vertex and then the cost-to-come of the vertex. Both these terms are calculated for a vertex, $\mathbf{v} \in V$, considering all its possible incoming edges (referred to as the *rhs-value* in LPA*), i.e.,

$$\min_{(\mathbf{u},\mathbf{v}) \in E} \{g_{\mathcal{T}}(\mathbf{u}) + c(\mathbf{u}, \mathbf{v})\} + \widehat{h}(\mathbf{v})$$

and

$$\min_{(\mathbf{u},\mathbf{v}) \in E} \{g_{\mathcal{T}}(\mathbf{u}) + c(\mathbf{u}, \mathbf{v})\}$$

This minimum requires calculating the true edge cost between a vertex and all of its possible parents. This calculation is expensive in sampling-based planning (e.g., collision checking, differential constraints, etc.) and reducing its calculation is desirable. This can be achieved by incrementally calculating the minimum in the order given by an admissible heuristic estimate of edge cost. Considering edges into the vertex in order of increasing *estimated* cost calculates a running minimum that can be stopped when the estimated cost through the next edge to consider is higher than the current minimum.

BIT* combines the minimum calculations for individual vertices into a single edge queue. This simultaneously calculates the minimum cost-to-come for each vertex in the current graph while expanding vertices in order of increasing estimated solution cost. □

5. Modifications and extensions

The basic version of BIT* presented in Algorithms 1–3 can be modified and extended to include features that may improve performance for some planning applications. Section 5.1 presents a method to delay rewiring the tree until an initial solution is found. This prioritizes exploring the RGG to find solutions and may be beneficial in time-constrained applications. Section 5.2 presents a method to delay sampling until necessary. This avoids approximating regions of the planning problem that are never searched, improves performance in large planning problems, and avoids the need to define *a priori* limits in unbounded problems.

Section 5.3 presents a method for BIT* to occasionally remove unconnected samples while maintaining almost-sure asymptotic optimality. This avoids repeated connection attempts to infeasible states and may be beneficial in problems where many regions of the free space are unreachable.

Section 5.4 extends the idea of reducing the number of connections attempted per sample during an ordered search to develop SORRT*. This version of RRT* uses batches of samples to order its search of the problem domain by potential solution quality, as in BIT*, but uses a `steer` function and only makes one connection attempt per sample, as in RRT*.

5.1. Delayed rewiring

Many robotic systems have a finite amount of computational time available to solve planning problems. In these

situations, rewiring the existing tree before an initial solution is found reduces the likelihood of BIT* solving the given problem. A method to delay rewirings until a solution is found is presented in Algorithm 4 as simple modifications to `ExpandNextVertex`, with changes highlighted in red (cf. Algorithm 2). The rewirings are still performed once a solution is found and this method does not affect almost-sure asymptotic optimality.

Rewirings are delayed by separately tracking whether vertices are unexpanded to nearby unconnected samples, V_{unexpnd}, and unexpanded to nearby connected vertices, V_{delayed}. This allows BIT* to prioritize finding a solution by only considering edges to new samples until a solution is found and then improving the graph by considering rewirings from the existing vertices.

A vertex is moved from the never-expanded set to the delayed set when edges to its potential unconnected descendants are added to the edge queue (Algorithm 4, lines 4–5). This allows future expansions of the vertex to avoid old unconnected samples while tracking that the vertex's outgoing rewirings have not yet been considered. Vertices in the delayed set are expanded as potential rewirings of other connected vertices once a solution is found (Algorithm 4, line 9) and the delayed label is removed (Algorithm 4, line 12).

This extension requires initializing and resetting V_{delayed} along with the other labeling sets (e.g., Algorithm 1, line 4 and Algorithm 3, line 5). This extension is included in the publicly available OMPL implementation of BIT*.

Algorithm 4: `ExpandNextVertex`$(\mathcal{Q}_V \subseteq V,\ \mathcal{Q}_E \subseteq V \times (V \cup X),\ c_i \in \mathbb{R}_{\geqslant 0})$

1 $\mathbf{v}_{\min} \leftarrow \texttt{PopBestInQueue}\,(\mathcal{Q}_V)$;
2 **if** $\mathbf{v}_{\min} \in V_{\text{unexpnd}}$ **then**
3 $\quad X_{\text{near}} \leftarrow \texttt{Near}\,(X_{\text{unconn}}, \mathbf{v}_{\min}, r_{\text{BIT}^*})$;
4 $\quad V_{\text{unexpnd}} \xleftarrow{-} \{\mathbf{v}_{\min}\}$;
5 $\quad V_{\text{delayed}} \xleftarrow{+} \{\mathbf{v}_{\min}\}$;
6 **else**
7 $\quad X_{\text{near}} \leftarrow \texttt{Near}\,(X_{\text{new}} \cap X_{\text{unconn}}, \mathbf{v}_{\min}, r_{\text{BIT}^*})$;
8 $\mathcal{Q}_E \xleftarrow{+} \left\{(\mathbf{v}_{\min}, \mathbf{x}) \in V \times X_{\text{near}} \;\middle|\; \widehat{g}\,(\mathbf{v}_{\min}) + \widehat{c}\,(\mathbf{v}_{\min}, \mathbf{x}) + \widehat{h}\,(\mathbf{x}) < c_i\right\}$;
9 **if** $\mathbf{v}_{\min} \in V_{\text{delayed}}$ **and** $c_i < \infty$ **then**
10 $\quad V_{\text{near}} \leftarrow \texttt{Near}\,(V, \mathbf{v}_{\min}, r_{\text{BIT}^*})$;
11 $\quad \mathcal{Q}_E \xleftarrow{+} \left\{(\mathbf{v}_{\min}, \mathbf{w}) \in V \times V_{\text{near}} \;\middle|\; (\mathbf{v}_{\min}, \mathbf{w}) \notin E,\ \widehat{g}\,(\mathbf{v}_{\min}) + \widehat{c}\,(\mathbf{v}_{\min}, \mathbf{w}) + \widehat{h}\,(\mathbf{w}) < c_i,\ \widehat{g}\,(\mathbf{v}_{\min}) + \widehat{c}\,(\mathbf{v}_{\min}, \mathbf{w}) < g_{\mathcal{T}}\,(\mathbf{w})\right\}$;
12 $\quad V_{\text{delayed}} \xleftarrow{-} \{\mathbf{v}_{\min}\}$;

5.2. Just-inTime (JIT) Sampling

Many robotic systems operate in environments that are unbounded (e.g., the outdoors). These problems have commonly required using *a priori* search limits to make the problem domain tractable. Selecting these limits can be difficult and may prevent finding a solution (e.g., defining a domain that does not contain a solution) or reduce performance (e.g., defining a domain too large to search sufficiently). A method to avoid these problems in BIT* by generating samples just-in-time (JIT) is presented in Algorithm 5 and accompanying modifications to the main algorithm. This modification generates samples only when needed by BIT*'s search while still maintaining uniform sample density and almost-sure asymptotic convergence to the optimum. This avoids approximating regions of the problem that are not used to find a solution and allows BIT* to operate directly on large or unbounded planning problems.

BIT* searches a planning problem by constructing and searching an implicit RGG defined by a number of uniformly distributed samples (vertices) and their relative distances (edges). These samples are given explicitly in Algorithms 1–3 but are not used until they could be a descendant of a vertex in the tree. This occurs when the samples are within the local neighborhood of the vertex (Figure 5). The cost associated with the informed set containing the neighborhood of a vertex, X_{near}, for planning problems seeking to minimize path length,

$$c_{\text{req'd}} := \max_{\mathbf{x} \in X_{\text{near}}} \left\{\widehat{f}(\mathbf{x})\right\}$$

is bounded from above as

$$c_{\text{req'd}} \leqslant \widehat{f}(\mathbf{v}) + 2r_{\text{BIT}^*}.$$

JIT sampling only generates samples when necessary to expand vertices by incrementally sampling this growing informed set. It does this while maintaining uniform sample density by tracking the previously sampled size of the set, c_{sampled}, and only generating the new samples necessary to increase it. The function $\texttt{UpdateSamples}\left(\mathbf{v}, c_{\text{sampled}}, c_i\right)$ generates JIT samples for vertex expansion in problems seeking to minimize path length (Algorithm 5). The required size of the sampled informed set, $c_{\text{req'd}}$, is a function of the neighborhood and the maximum size of the informed set (Algorithm 5, line 1). If it is higher than the previously sampled cost, c_{sampled}, then the local neighborhood has not been completely sampled and new samples are generated (Algorithm 5, line 2). If it is lower, then the neighborhood has already been sampled and no new samples are generated.

The number of required new samples, m', can be calculated from the chosen batch sample density, ρ, and the volume of the shell being sampled (Algorithm 5, lines 3–4). These samples are added to the set of new states and the set of unconnected states (Algorithm 5, lines 5–6). Finally, the sampled cost is updated to reflect the new size of the sampled informed set (Algorithm 5, line 7).

The function $\lambda_{\text{PHS}}(\cdot)$ calculates the measure of the prolate hyperspheroid defined by the start and goal with the given cost,

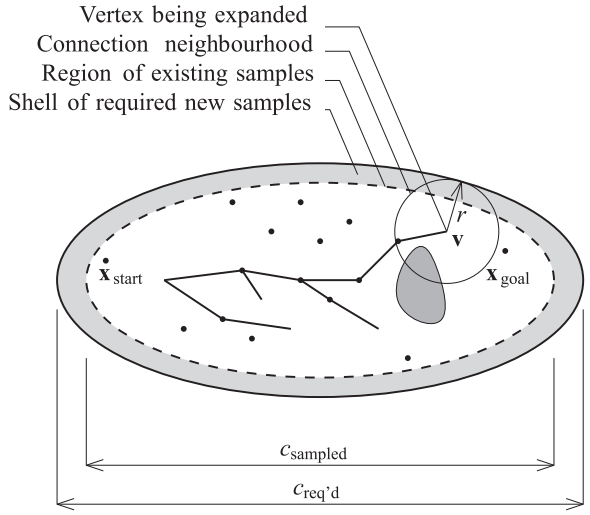


Fig. 5. An illustration of JIT sampling. Samples are generated only when they are necessary for the expansion of a vertex, **v**, into the edge queue. This is accomplished while maintaining uniform sample density by an expanding informed set. The informed set is expanded by adding uniformly distributed samples in the prolate hyperspheroidal shell defined by the difference between the maximum heuristic value of the neighborhood, $c_{\text{req'd}}$, and the currently sampled informed set, c_{sampled}.

$$\lambda\left(X_{\widehat{f}}\right) \leqslant \lambda(X_{\text{PHS}}) = \frac{c_i\left(c_i^2 - c_{\min}^2\right)^{\frac{n-1}{2}}\zeta_n}{2^n}$$

where ζ_n is the Lebesgue measure of an n-dimensional unit ball,

$$\zeta_n := \frac{\pi^{\frac{n}{2}}}{\Gamma\left(\frac{n}{2}+1\right)}$$

and $\Gamma(\cdot)$ is the gamma function, an extension of factorials to real numbers (Euler, 1738). The function $\texttt{Sample}\left(m', c_{\text{sampled}}, c_{\text{req'd}}\right)$ generates informed samples within the cost interval $\left[c_{\text{sampled}}, c_{\text{req'd}}\right)$ and may be implemented with rejection sampling.

Using Algorithm 5 requires modifying Algorithms 1 and 2 . The `Sample` function of a batch (Algorithm 1, line 9) is replaced with an initialization of the sampled cost variable, $c_{\text{sampled}} \leftarrow 0$, and $\texttt{UpdateSamples}\left(\mathbf{v}, c_{\text{sampled}}, c_i\right)$ is added to the front of `ExpandNextVertex` (Algorithm 2). This extension is included in the publicly available OMPL implementation of BIT*.

5.3. Sample removal

BIT* approximates continuous planning problems with an implicit RGG. It efficiently increases the accuracy of this approximation by focusing it to the informed set. It alternately increases density by adding new samples and shrinks the set by searching the existing samples for better solutions. This builds an explicit spanning tree that contains all states that could *currently* provide a better solution but may not use every state in the RGG.

Algorithm 5: $\texttt{UpdateSamples}\left(\mathbf{v} \in V,\ c_{\text{sampled}} \leqslant c_i, c_i \in \mathbb{R}_{\geqslant 0}\right)$

```
1 c_req'd ← min{ f̂(v) + 2r_BIT*, c_i };
2 if c_req'd > c_sampled then
3     λ_sample ← λ_PHS(c_req'd) − λ_PHS(c_sampled);
4     m' ← ρ λ_sample;
5     X_new ←+ Sample(m', c_sampled, c_req'd);
6     X_unconn ←+ X_new;
7     c_sampled ← c_req'd;
```

States in an informed set may not be able to improve a solution for many reasons. The approximation may be insufficiently accurate (i.e., low sample density) to capture difficult features (e.g., narrow passages) or represent sufficiently optimal paths. The informed set may also include regions of the problem domain that cannot improve the solution due to unconsidered problem features (e.g., barriers separating free space) or because it is otherwise poorly chosen (i.e., low precision). Unconnected samples in the first situation may later be beneficial to the search but samples in the second represent unnecessary computational cost. Periodically removing these samples would reduce the complexity of the implicit RGG and avoid repeatedly attempting to connect them to new vertices in the tree.

Unconnected samples can be removed while maintaining the requirements for almost-sure asymptotic optimality by modifying the RGG connection limits to consider only uniformly distributed samples. This can be accomplished by using the number of uniformly distributed samples added since the last removal of unconnected states in (1) and (2). This simple extension is included in the publicly available OMPL implementation of BIT*.

5.4. Sorted RRT* (SORRT*)

Approaching sampling-based planning as the search of an implicit RGG motivates BIT* to consider multiple connections to each sample. Section 5.3 presents a method to limit this number of attempts by periodically removing samples. The natural extension of this idea is to consider only a single connection attempt per sample, as in RRT*. This motivates the development of SORRT*, a version of RRT* that orders its search by potential solution quality by sorting batches of samples.

SORRT* is presented in Algorithm 6 as simple modifications of Informed RRT*, with changes highlighted in red (cf. Gammell et al., 2018, 2014c). Instead of expanding the tree towards a randomly generated sample at each iteration, SORRT* extends the tree towards the best unconsidered sample in its current batch. It accomplishes this by using a queue of samples, $\mathcal{Q}_{\text{Samples}}$, ordered by potential solution cost, $\widehat{f}(\cdot)$. This queue is filled with m samples (Algorithm 6, lines 5–6) and the search proceeds by expanding the tree

Algorithm 6: SORRT*$(\mathbf{x}_{\text{start}} \in X_{\text{free}},\ X_{\text{goal}} \subset X)$

```
1  V ← {x_start}; E ← ∅; T = (V, E);
2  V_sol'n ← ∅;  Q_Samples ← ∅;
3  for i = 1 ... q do
4      c_i ← min_{v_goal ∈ V_sol'n} {g_T(v_goal)};
5      if Q_Samples ≡ ∅ then
6          Q_Samples ← Sample(m, x_start, X_goal, c_i);
7      x_rand ← PopBestInQueue(Q_Samples);
8      v_nearest ← Nearest(V, x_rand);
9      x_new ← Steer(v_nearest, x_rand);
10     if CollisionFree(v_nearest, x_new) then
11         if x_new ∈ X_goal then
12             V_sol'n +← {x_new};
13         V +← {x_new};
14         V_near ← Near(V, x_new, r_rewire);
15         v_min ← v_nearest;
16         forall v_near ∈ V_near do
17             c_new ← g_T(v_near) + ĉ(v_near, x_new);
18             if c_new < g_T(v_min) + c(v_min, x_new) then
19                 if CollisionFree(v_near, x_new) then
20                     v_min ← v_near;
21         E +← {(v_min, x_new)};
22         forall v_near ∈ V_near do
23             c_near ← g_T(x_new) + ĉ(x_new, v_near);
24             if c_near < g_T(v_near) then
25                 if CollisionFree(x_new, v_near) then
26                     v_parent ← Parent(v_near);
27                     E −← {(v_parent, v_near)};
28                     E +← {(x_new, v_near)};
29         Prune(V, E, c_i);
30 return T;
```

towards the best sample in the queue (Algorithm 6, line 7). This orders the search for the m samples in a batch, at which point a new batch of samples is generated and the search continues.

The function PopBestInQueue$(\cdot)$ pops the best element off the front of a queue given its ordering. A goal bias may be implemented in SORRT* by adding a small probability of sampling the goal instead of removing the best sample from the queue. This algorithm is publicly available in OMPL.

SORRT* can be viewed as a simplified version of BIT* that only considers the best-possible edges. Attempting to connect each sample once avoids the computational cost of repeated connection attempts to infeasible samples but still maintains some dependence on the sampling order. High-utility samples (e.g., samples near the optimum) may be underutilized depending on the state of the tree when they are found. This can become problematic if these samples have a low sampling probability (e.g., samples in narrow passages). Making multiple connections attempts per sample and retaining samples for multiple batches allows BIT* to exploit these useful samples more than algorithms such as SORRT*. As seen in Section 6, this results in different performance, especially in high state dimensions.

6. Experiments

The benefits of ordering the search of continuous planning problems are demonstrated on simulated problems in $\mathbb{R}^2$, $\mathbb{R}^4$, $\mathbb{R}^8$, and $\mathbb{R}^{16}$ (Section 6.1) and one- and two-armed problems for HERB (Section 6.2) using OMPL.[2] BIT* is compared with the OMPL versions of RRT, RRT-Connect, RRT*, RRT$^\#$ (i.e., RRT^X with $\epsilon = 0$), FMT*, Informed RRT*, and SORRT*.

All planners used the same tuning parameters and configurations where possible. Planning time was limited to 1, 10, 30, and 100 seconds in $\mathbb{R}^2$, $\mathbb{R}^4$, $\mathbb{R}^8$, and $\mathbb{R}^{16}$, and 20 and 600 seconds for HERB ($\mathbb{R}^7$ and $\mathbb{R}^{14}$), respectively. RRT-style planners used a goal-sampling bias of 5% and a maximum edge length of $\eta = 0.3$, 0.5, 0.9, and 1.7 on the abstract problems ($\mathbb{R}^2$, $\mathbb{R}^4$, $\mathbb{R}^8$, and $\mathbb{R}^{16}$) and 0.7 and 1.3 on HERB, respectively. These values were selected experimentally to reduce the time required to find an initial solution on simple training problems.

The RRT* planners, FMT*, and BIT* all used a connection radius equal to twice their lower bound (e.g., $r_{\text{RRT}^*} = 2r^*_{\text{RRT}^*}$) and approximated the Lebesgue measure of the free space with the measure of the entire planning problem. The RRT* planners also used the ordered rewiring technique presented in Perez et al. (2011). Informed RRT*, SORRT*, and BIT* used the L^2 norm as estimates of cost-to-come and cost-to-go, direct informed sampling, and delayed pruning the graph until solution cost changed by more than 5%. SORRT* and BIT* both used $m = 100$ samples per batch for all problems. BIT* also thresholded its initial connection radius by using the same radius for both the first and second batches.

6.1. Simulated planning problems

The algorithms were tested on simulated problems in $\mathbb{R}^2$, $\mathbb{R}^4$, $\mathbb{R}^8$, and $\mathbb{R}^{16}$ on problems consisting of dual enclosures (Section 6.1.1), many different homotopy classes (Section 6.1.2), and randomly generated obstacles (Section 6.1.3). The planners were tested with 100 different pseudo-random seeds on each problem and state dimension. The solution cost of each planner was recorded every 10^{-4} seconds by a separate thread and the median was calculated from the 100 trials by interpolating each trial at a period of 10^{-4} seconds, except for the problems in $\mathbb{R}^{16}$ where 10^{-3} seconds was used for both times. The absence of a solution was considered an infinite cost for the purpose of calculating the median and infinite values were not plotted.

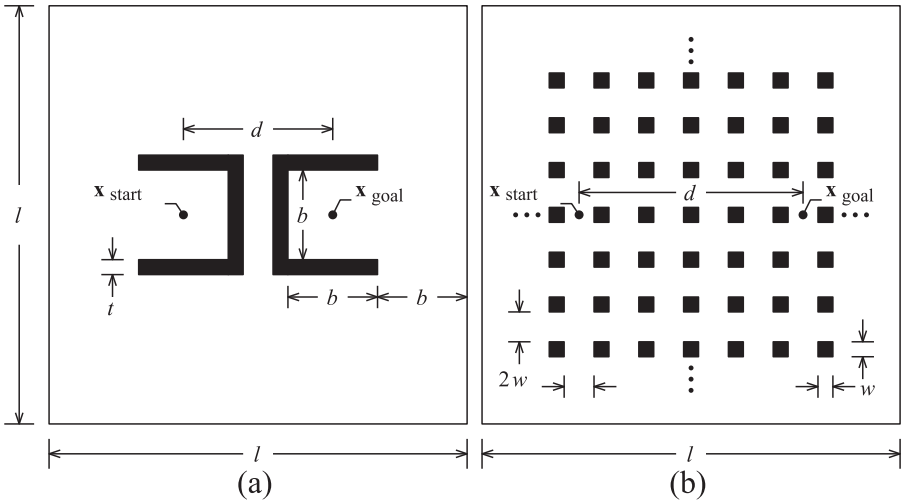


Fig. 6. Illustration of the planning problems used in Sections 6.1.1 and 6.1.2. They are used to investigate algorithm performance on complex problems containing dual enclosures (a) and many homotopy classes (b) across state dimension. The problem dimensions in (a) were chosen to make the gaps symmetric, i.e., $b = 0.6$, and $l = 2.8$, for the chosen wall thickness, $t = 0.1$. The width of the individual obstacles in (b) are chosen such that the start and goal states are 5 "columns" apart in a problem domain of size $l = 4$. For both problems, the distance between the start and goal, d, is 1.

6.1.1. Dual-enclosure problems. The algorithms were tested on problems with two enclosures in $\mathbb{R}^2$, $\mathbb{R}^4$, $\mathbb{R}^8$, and $\mathbb{R}^{16}$ (Figure 6(a)). The problems consisted of a (hyper)cube of width $l = 2.8$ with the start and goal located at $[-0.5, 0, \ldots, 0]^{\mathrm{T}}$ and $[0.5, 0, \ldots, 0]^{\mathrm{T}}$, respectively. The enclosures were symmetric around the start and goal with a thickness of $t = 0.1$ and openings of width $b = 0.6$. As all almost-surely asymptotically optimal planners struggled to solve the problem in $\mathbb{R}^{16}$, it was run for 1,000 seconds with recording and interpolation periods of 2×10^{-3} seconds.

The results are presented in Figure 7 with the percent of trials solved and the median solution cost plotted versus run time. The results demonstrate the advantages of ordering the approximation and search of difficult planning problems. BIT* is competitive with other almost-surely asymptotically optimal planning algorithms in $\mathbb{R}^2$ and outperforms all algorithms other than RRT-Connect in higher state dimensions. In all dimensions, BIT* finds a solution in every trial (i.e., attains 100% success) sooner than the other anytime almost-surely asymptotically optimal planners.

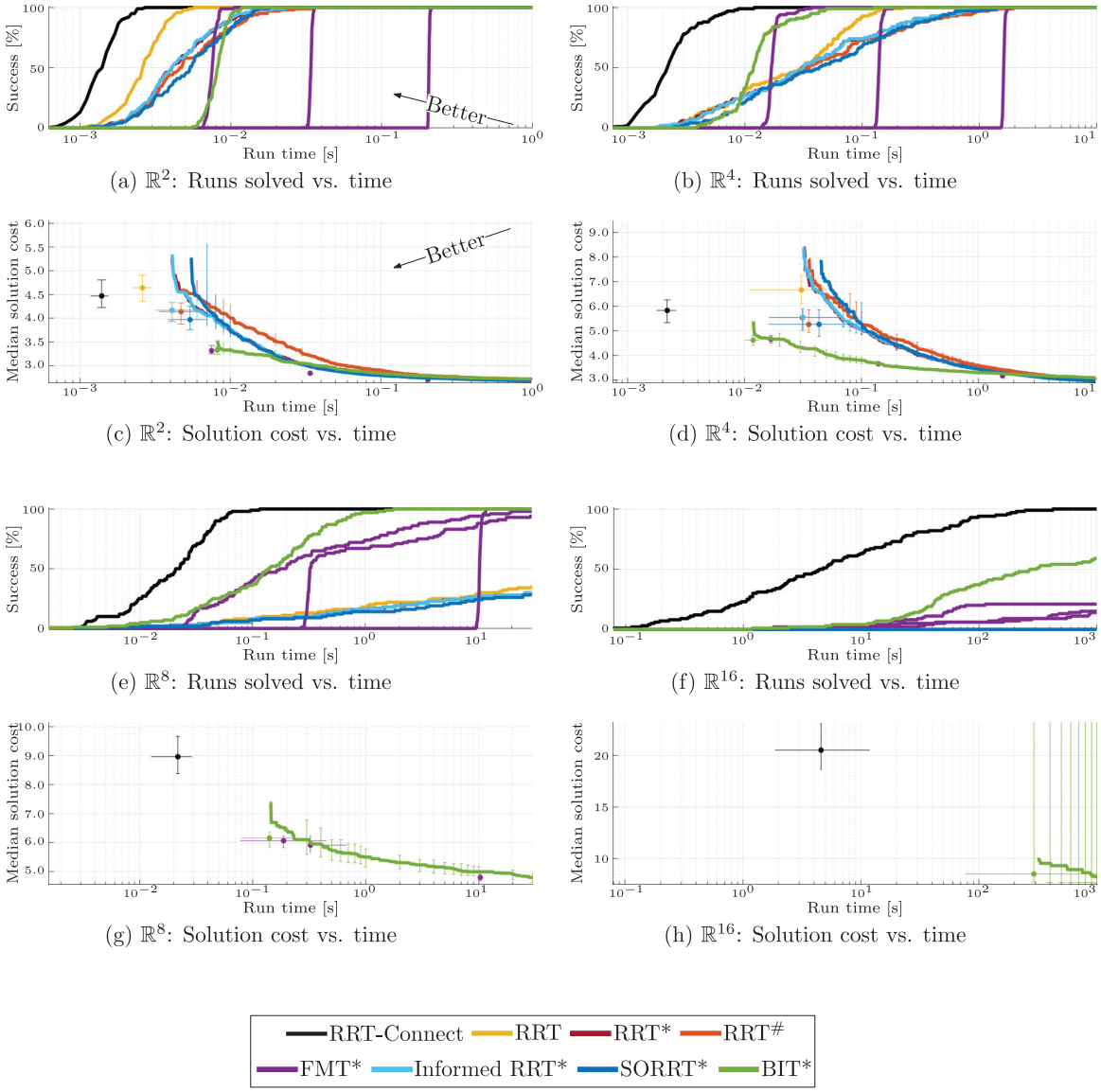
Specifically in $\mathbb{R}^2$, the median time required for BIT* to find an initial solution is more than that of the RRT*-based planners (Figure 7(c)); however, once a solution is found, BIT* finds better or equivalent solutions than the best performing RRT* planners at any given time. FMT* slightly outperforms the other almost-surely asymptotically optimal planners but is not anytime.

The performance of RRT*-based planners decreases more rapidly with increasing state dimension on this problem than planners that process multiple samples such as BIT* and FMT*. BIT* outperforms all planners other than RRT-Connect in terms of success rate and median solution cost in $\mathbb{R}^4$ (Figures 7(b) and (d)). This difference increases in $\mathbb{R}^8$ where the RRT*-based planners only find a solution in the available time in 35% of the trials or less (Figures 7(e) and (g)). All almost-surely asymptotically optimal planners struggle to solve the problem in $\mathbb{R}^{16}$ but BIT* is the only one that finds a solution in more than 50% of the trials (Figures 7(f) and (h)).

6.1.2. Problems with many homotopy classes. The algorithms were tested on problems with many homotopy classes in $\mathbb{R}^2$, $\mathbb{R}^4$, $\mathbb{R}^8$, and $\mathbb{R}^{16}$ (Figure 6(b)). The problems consisted of a (hyper)cube of width $l = 4$ with the start and goal located at $[-0.5, 0, \ldots, 0]^{\mathrm{T}}$ and $[0.5, 0, \ldots, 0]^{\mathrm{T}}$, respectively. The problem domain was filled with a regular pattern of axis-aligned (hyper)rectangular obstacles with a width such that the start and goal were 5 "columns" apart.

The results are presented in Figure 8 with the percentage of trials solved and the median solution cost plotted versus run time. These results demonstrate both the advantages and disadvantages of attempting multiple connections per sample in problems with many disconnected obstacles. In lower dimensions, informed algorithms that only make a single connection attempt per sample (e.g., Informed RRT* and SORRT*) have better median solution costs at any given time. In higher dimensions, only the bidirectional RRT-Connect and algorithms that make multiple connection attempts per sample (e.g., BIT* and FMT*) find solutions in a reasonable amount of time. In all dimensions, BIT* finds a solution in every trial (i.e., attains 100% success) sooner than every planner tested other than RRT-Connect.

Specifically, BIT* is the most likely almost-surely asymptotically optimal planning algorithm to solve the problem in $\mathbb{R}^2$, $\mathbb{R}^4$, and $\mathbb{R}^8$ (Figures 8(a), (b), and (e)) but the informed, RRT*-based algorithms have better median solution costs (Figures 8(c), (d), and (g)). This advantage disappears in $\mathbb{R}^{16}$ where BIT* is the only anytime almost-surely asymptotically optimal planner to always solve the planning problem in the given time (Figures 8(f) and (h)).



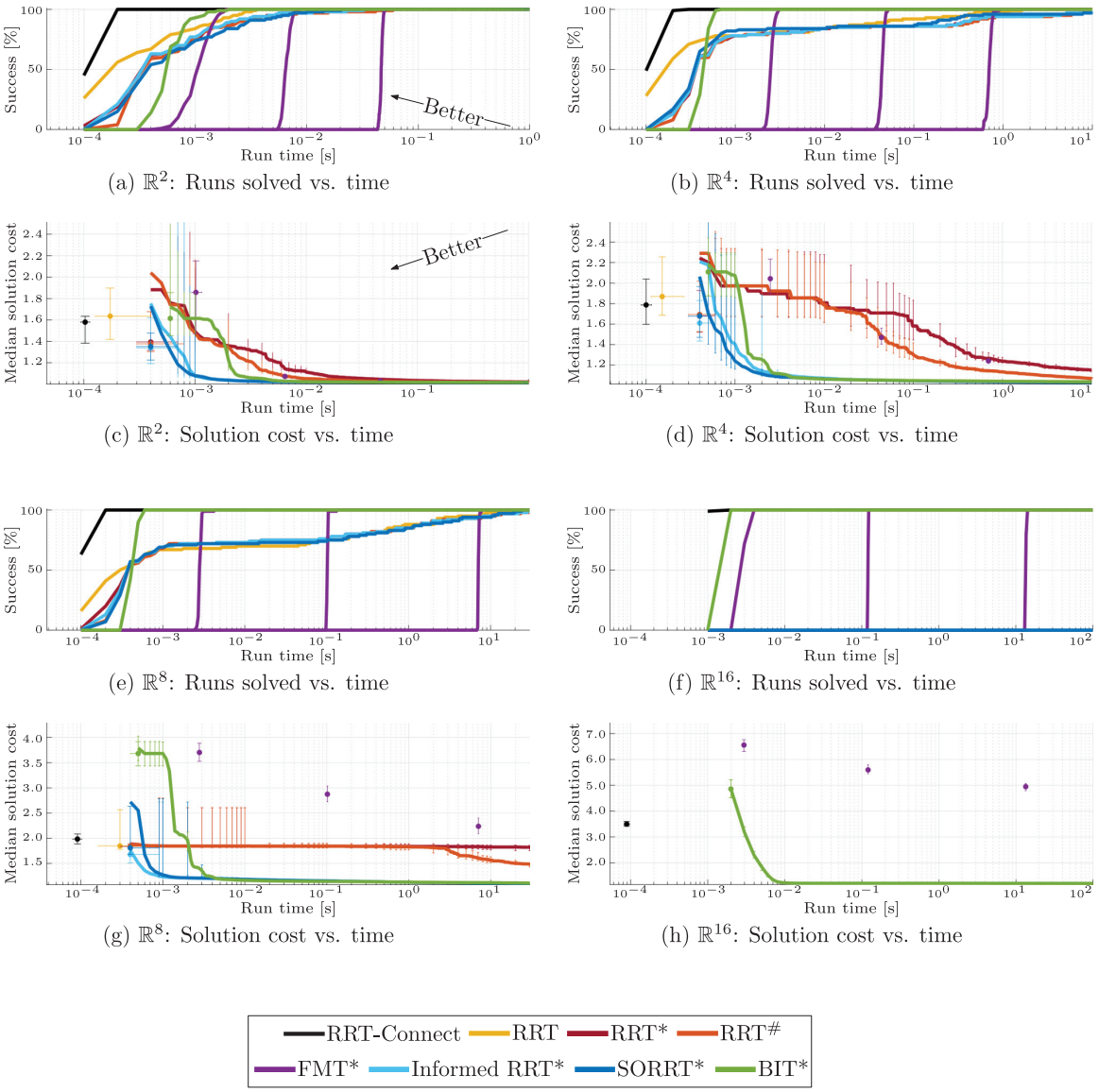
(a) $\mathbb{R}^2$: Runs solved vs. time
(b) $\mathbb{R}^4$: Runs solved vs. time
(c) $\mathbb{R}^2$: Solution cost vs. time
(d) $\mathbb{R}^4$: Solution cost vs. time
(e) $\mathbb{R}^8$: Runs solved vs. time
(f) $\mathbb{R}^{16}$: Runs solved vs. time
(g) $\mathbb{R}^8$: Solution cost vs. time
(h) $\mathbb{R}^{16}$: Solution cost vs. time

Fig. 7. Planner performance versus time for the problem illustrated in Figure 6(a). Each planner was run 100 different times in $\mathbb{R}^2$, $\mathbb{R}^4$, $\mathbb{R}^8$, and $\mathbb{R}^{16}$ with run times limited to 1, 10, 30, and 1,000 seconds, respectively. The percentage of trials solved is plotted versus run time for each planner and presented in (a), (b), (e), and (f). The median path length is plotted versus run time for each planner and presented in (c), (d), (g), and (h), with unsuccessful trials assigned infinite cost. The error bars denote a non-parametric 99% confidence interval on the median. The results show that BIT* is competitive to other almost-surely asymptotically optimal planners in $\mathbb{R}^2$ and outperforms them in all other tested state dimensions. Note the difficulty of solving this problem in $\mathbb{R}^8$ and $\mathbb{R}^{16}$ in the available time.

This difference in performance may arise from the relative difficulty of planning tasks in different state dimensions. It appears that in lower state dimensions the main challenge of this problem is avoiding obstacles. In these situations, informed, RRT*-based algorithms will outperform BIT* as they only make one connection attempt per sample. In higher state dimensions, it appears that the exponential increase in problem measure (i.e., the curse of dimensionality) makes navigating towards the goal an equal challenge and BIT* outperforms these other algorithms by considering multiple connection attempts per sample in an ordered fashion.

6.1.3. Random problems. The planners were tested on randomly generated problems in $\mathbb{R}^2$, $\mathbb{R}^4$, $\mathbb{R}^8$, and $\mathbb{R}^{16}$. The worlds consisted of a (hyper)cube of width $l = 2$ populated with approximately 75 random axis-aligned (hyper)rectangular obstacles that obstruct at most one third of the environment.

For each state dimension, 10 different random worlds were generated and the planners were tested on each with 100 different pseudo-random seeds. The true optima for these 10 problems are different and unknown and there is no meaningful way to compare the results across problems. Results from a representative problem are instead presented



(a) $\mathbb{R}^2$: Runs solved vs. time
(b) $\mathbb{R}^4$: Runs solved vs. time
(c) $\mathbb{R}^2$: Solution cost vs. time
(d) $\mathbb{R}^4$: Solution cost vs. time
(e) $\mathbb{R}^8$: Runs solved vs. time
(f) $\mathbb{R}^{16}$: Runs solved vs. time
(g) $\mathbb{R}^8$: Solution cost vs. time
(h) $\mathbb{R}^{16}$: Solution cost vs. time

Fig. 8. Planner performance versus time for the problem illustrated in Figure 6(b). Each planner was run 100 different times in $\mathbb{R}^2$, $\mathbb{R}^4$, $\mathbb{R}^8$, and $\mathbb{R}^{16}$ with run times limited to 1, 10, 30, and 100 seconds, respectively. The percentage of trials solved is plotted versus run time for each planner and presented in (a), (b), (e), and (f). The median path length is plotted versus run time for each planner and presented in (c), (d), (g), and (h), with unsuccessful trials assigned infinite cost. The error bars denote a non-parametric 99% confidence interval on the median. The results show that on this problem BIT* finds a solution to all trials faster than all tested planners other than RRT-Connect and outperforms other almost-surely asymptotically optimal planners in $\mathbb{R}^{16}$. Note that RRT*-based planners are not able to find solutions for this high state dimension in the time available.

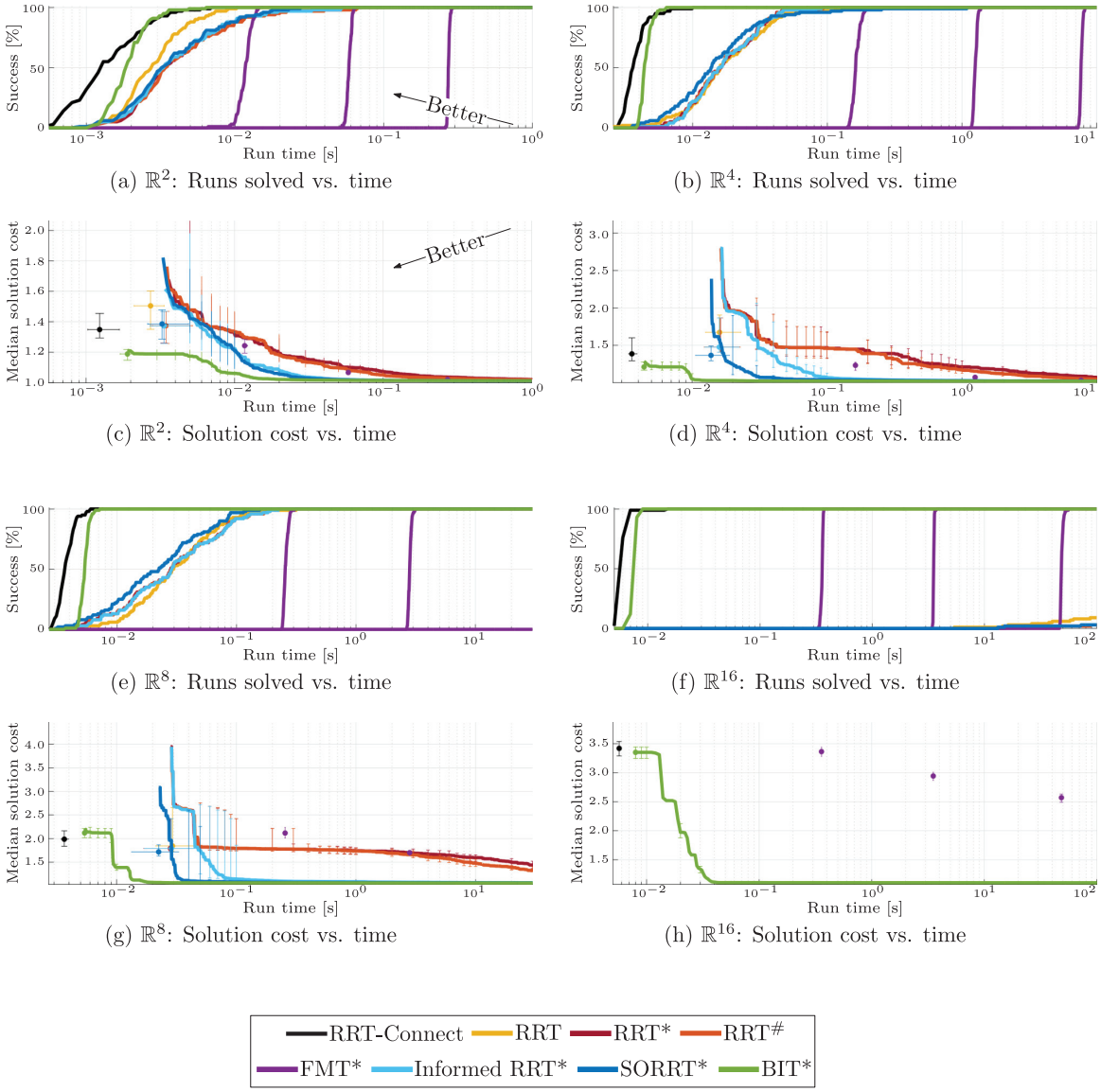
in Figure 9 with the percentage of trials solved and the median solution cost plotted versus computational time.

These experiments show that BIT* generally finds better solutions faster than other sampling-based optimal planners and RRT on these types of problems regardless of the state dimension. It has a higher likelihood of having found a solution at a given computational time (Figures 9(a), (b), (e), and (f)), and converges faster towards the optimum (Figures 9(c), (d), (g), and (h)), with the relative improvement increasing with state dimension. The only tested planner that found solutions faster than BIT* was RRT-Connect, a non-anytime planner that cannot converge to the optimum.

6.2. Path planning for HERB

It is difficult to capture the challenges of actual high-dimensional planning in abstract worlds. Two planning problems inspired by manipulation scenarios were created for HERB, a 14-DOF mobile manipulation platform.

Start and goal poses were chosen for one arm (7 DOFs, Section 6.2.1) and two arms (14 DOFs, Section 6.2.2) to define planning problems with the objective of minimizing path length through configuration space. They were used to compare the OMPL versions of RRT, RRT-Connect, FMT*, RRT# (i.e., RRT^X with $\epsilon = 0$), Informed RRT*, SORRT*,



(a) $\mathbb{R}^2$: Runs solved vs. time
(b) $\mathbb{R}^4$: Runs solved vs. time
(c) $\mathbb{R}^2$: Solution cost vs. time
(d) $\mathbb{R}^4$: Solution cost vs. time
(e) $\mathbb{R}^8$: Runs solved vs. time
(f) $\mathbb{R}^{16}$: Runs solved vs. time
(g) $\mathbb{R}^8$: Solution cost vs. time
(h) $\mathbb{R}^{16}$: Solution cost vs. time

Fig. 9. Planner performance versus time for a randomly generated problem. Each planner was run 100 different times in $\mathbb{R}^2$, $\mathbb{R}^4$, $\mathbb{R}^8$, and $\mathbb{R}^{16}$ with run times limited to 1, 10, 30, and 100 seconds, respectively. The percentage of trials solved is plotted versus run time for each planner and presented in (a), (b), (e), and (f). The median path length is plotted versus run time for each planner and presented in (c), (d), (g), and (h), with unsuccessful trials assigned infinite cost. The error bars denote a non-parametric 99% confidence interval on the median. The results show that BIT* outperforms other almost-surely asymptotically optimal planners in both ability to solve the problem and median solution cost. Note the increase in this difference with higher state dimension and the difficulty of solving the problem in $\mathbb{R}^{16}$ in the available time with RRT*-based planners.

and BIT*. The planners were run on each problem 50 times while recording success rate, initial solution time and cost, and final cost. Trials that did not find a solution were considered to have taken infinite time and have infinite path length, respectively, for the purpose of calculating medians. The number of FMT* samples for both problems was chosen to use the majority of the available computational time.

6.2.1. A one-armed planning problem. A planning problem was defined for HERB's left arm around a cluttered table (Figure 10). The arm starts folded at the elbow and held at approximately the level of the table (Figures 10(a) and (b)) and must be moved into position to grasp a box (Figures 10(c) and (d)). The planners were given 20 seconds of computational time to solve this 7-DOF problem with the objective of minimizing path length in configuration space. FMT* used $m = 30$ samples.

The percentage of trials that successfully found a solution (Figure 11(a)), the median time and cost of the initial solution (Figures 11(b) and (c)) and the final cost (Figure 11(d)) were plotted for each planner. Infinite values are not plotted.

The results show BIT* finds solutions more often than other almost-sure asymptotically optimal planners on this

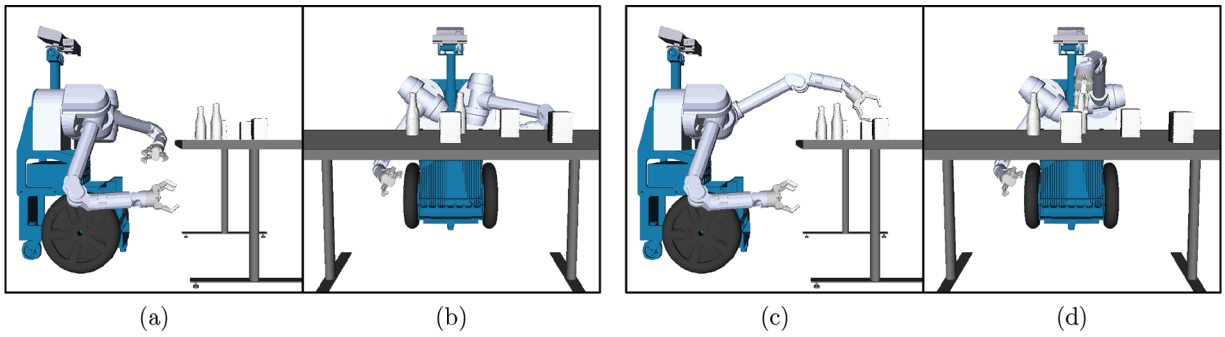


Fig. 10. A one-armed motion planning problem for HERB in $\mathbb{R}^7$. Starting at a position level with the table, (a) and (b), HERB's left arm must be moved in preparation for grasping a box on the far side of the table, (c) and (d).

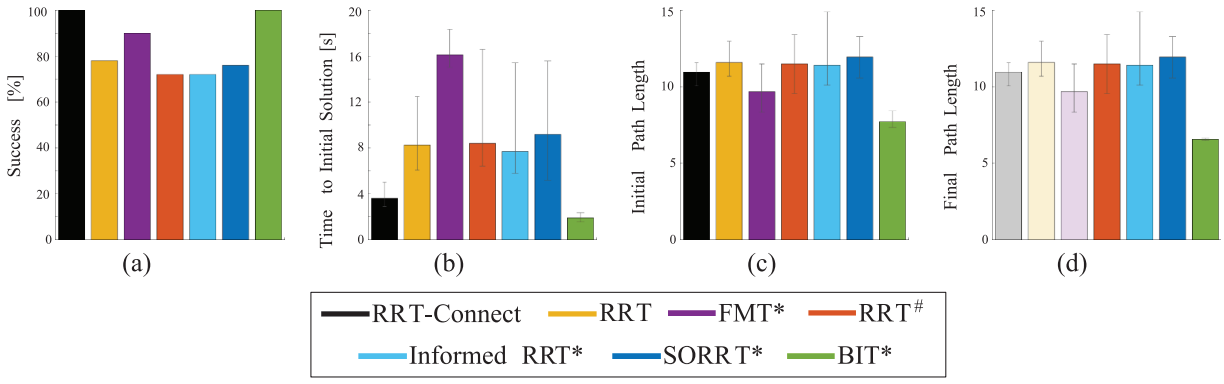


Fig. 11. Results from 50, 20 second trials on the one-armed HERB planning problem shown in Figure 10. The percentage of solutions solved (a), the median time to an initial solution (b), the median initial path length (c), and the median final path length (d) are presented with 99% confidence intervals for each planner. Unsuccessful trials were assigned infinite time and cost. The inability of non-anytime planners (e.g., RRT, RRT-Connect, and FMT*) to use the remaining available time to improve their initial solution is denoted with diminished color in (d), where present. BIT* is the only almost-surely asymptotically optimal planner to solve all 50 trials and does so in a time comparable to RRT-Connect. It also finds significantly lower cost paths than all the other planners.

problem (Figure 11(a)) and also finds better solutions faster than all tested algorithms, including RRT-Connect (Figures 11(b)–(d)). Figure 12 presents a composite photograph of HERB executing a path found by BIT* for a similar problem.

6.2.2. A two-armed planning problem. A second planning problem was defined for both of HERB's arms moving around a cluttered table (Figure 13). The arms start at a neutral position with their forearms extended under the table (Figures 13(a) and (b)) and must be moved into position to open a bottle (Figures 13(c) and (d)). The planners were given 600 seconds of computational time to solve this 14-DOF problem with the objective of minimizing path length in configuration space. FMT* used $m = 1{,}750$ samples.

The percentage of trials that successfully found a solution (Figure 14(a)), the median time and cost of the initial solution (Figures 14(b) and (c)) and the final cost (Figure 14(d)) were plotted for each planner. Infinite values are not plotted.

Fig. 12. A composite figure of HERB executing a path found by BIT* on a one-armed planning problem similar to Figure 10.

The results show that even when more computational time is available, BIT* still finds solutions more often than other almost-surely asymptotically optimal planners (Figure 14(a)) and also finds initial solutions faster than all tested

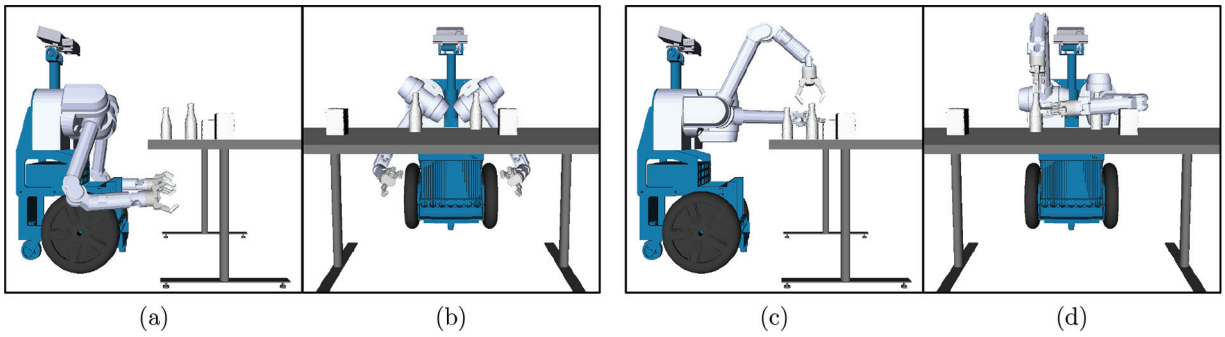


Fig. 13. A two-armed motion planning problem for HERB in $\mathbb{R}^{14}$. Starting under the table, (a) and (b), HERB's arms must be moved in preparation for opening a bottle, (c) and (d).

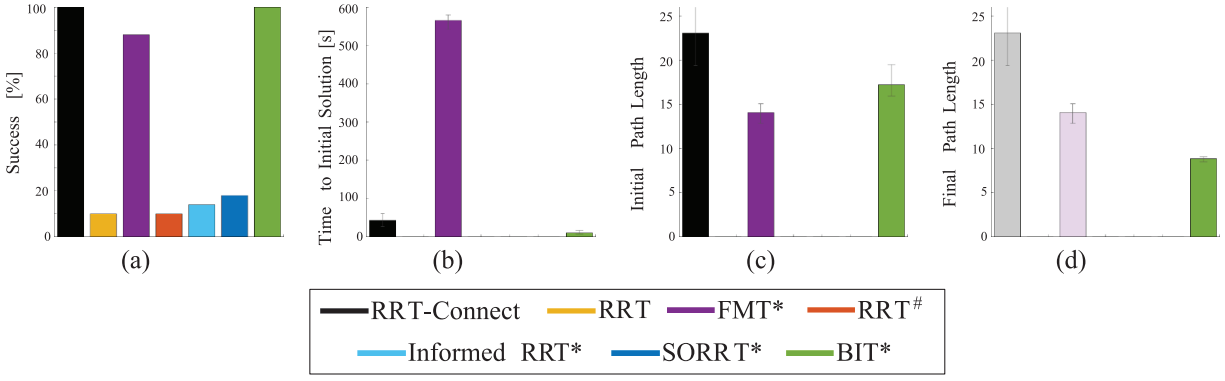


Fig. 14. Results from 50, 600 second trials on the two-armed HERB planning problem shown in Figure 13. The percentage of solutions solved (a), the median time to an initial solution (b), the median initial path length (c), and the median final path length (d) are presented with 99% confidence intervals for each planner. Unsuccessful trials were assigned infinite time and cost. The inability of non-anytime planners (e.g., RRT, RRT-Connect, and FMT*) to use the remaining available time to improve their initial solution is denoted with diminished color in (d), when present. BIT* is the only anytime planner to solve all 50 trials and does so in a time comparable to RRT-Connect. It focuses the search to the informed set once a solution is found and the resulting increasingly dense RGG allows it to find lower cost paths than all the other planners.

algorithms, including RRT-Connect (Figure 14(b)). As a non-anytime algorithm, FMT* is tuned to use the majority of the available time and finds a better initial solution (Figure 14(c)) but as BIT* is able to improve its solution it still finds a better final path after approximately the same amount of time (Figure 14(d)). Figure 15 presents a composite photograph of HERB executing a path found by BIT* for a similar problem.

7. Discussion and conclusion

Most planning algorithms discretize the search space of continuous path planning problems. Popular approaches in robotics include *a priori* graph- or anytime sampling-based approximations. Both of these approaches are successful but have important limitations.

A priori graphs approximate a problem before it is searched. Doing so "correctly" is challenging since the relationship between resolution and search performance depends on the specific features of a planning problem (e.g., the size and arrangement of obstacles). If the chosen approximation is insufficient (e.g., a sparse graph) it may preclude finding a (suitable) solution, but if it is excessive (e.g., a dense graph), it may make finding a solution prohibitively expensive.

Graph-based searches are effective path planning techniques despite these limitations. This is because informed algorithms, such as A*, use heuristics to order their search by potential solution quality. This not only finds the optimal solution to the given representation (i.e., it is *resolution optimal*) but does so by expanding the minimum number of vertices for the chosen heuristic (i.e., it is *optimally effcient*; Hart et al., 1968).

Anytime sampling-based planners alternatively build approximations that increase in resolution. This avoids the need to select a representation *a priori* and allows them to be run indefinitely until a (suitable) solution is found. RRT* has a unity probability of finding a solution, if one exists, with an infinite number of samples (i.e., it is *probabilistically complete*) and finds continuously improving solutions (i.e., it is *almost-surely asymptotically optimal*; Karaman and Frazzoli, 2011).

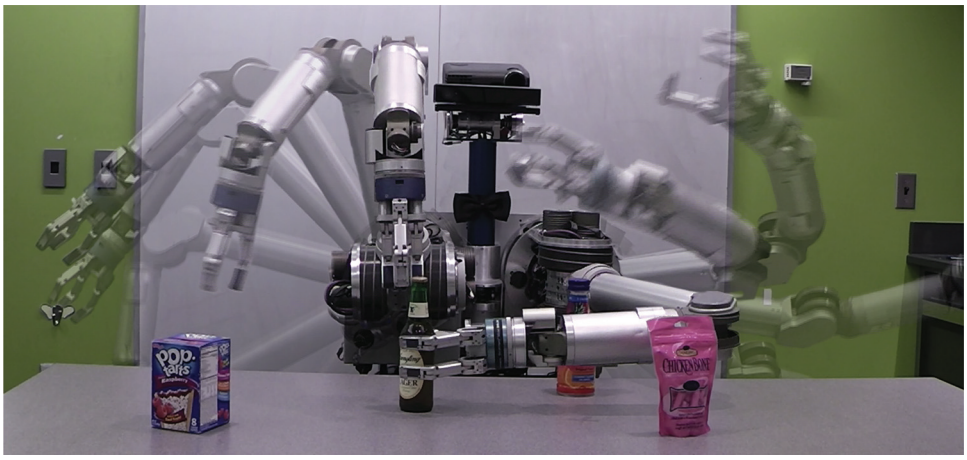


Fig. 15. A composite figure of HERB executing a path found by BIT* on a two-armed planning problem similar to Figure 13.

This has made them effective planning techniques despite the fact their search is often inefficient. Incremental sampling-based planners tend to explore the entire problem domain equally (i.e., they are *space filling*). This wastes computational effort on regions of the problem domain that are not needed to find the solution and can be prohibitively expensive in large planning problems and/or high state dimensions.

Previous attempts to unify these two approaches have been incomplete. They either sacrifice anytime resolution (e.g., RA* and FMT*), order the search on metrics other than solution cost and waste computational effort (e.g., SBA*), or do not order all aspects of the problem domain search (e.g., RRT# and RRTX). This article has demonstrated that these tradeoffs are unnecessary and that anytime sampling-based planners can be directly and solely ordered by a heuristic estimate of solution cost.

BIT* directly unifies informed graph-based search and sampling-based planning (Section 3). It uses heuristics and batches of random samples to simultaneously build anytime approximations of continuous planning problems and search these approximations in order of potential solution quality. This avoids the computational costs of both searching an improper approximation (graph-based search) and performing an unordered search of the problem domain (sampling-based planning). A version of BIT* is publicly available in OMPL.

BIT* approximates the search space by using *batches* of samples to define increasingly dense implicit RGGs. Building this approximation with batches of *multiple samples* allows each search to be ordered by potential solution quality, as in A*. Building this approximation from *multiple batches* of samples allows it to be improved indefinitely until it contains a suitable solution, as in RRT*. As a result, BIT* is probabilistically complete and almost-surely asymptotically optimal (Section 4).

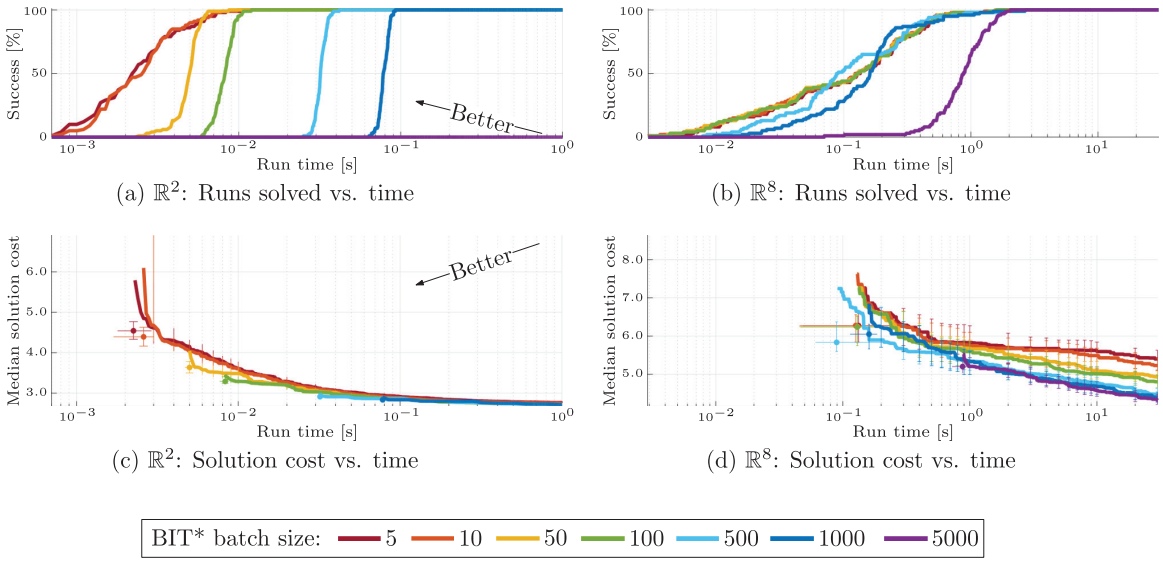
This simultaneous approximation and search is done efficiently by using heuristics. The approximation is focused to the regions of the planning problem that could provide better solutions, as in Informed RRT*. The denser approximations are searched efficiently by reusing previous information, as in TLPA*. Edge calculations (e.g., two-point BVPs and collisions checks) are delayed until necessary, as in *lazy* versions of both graph-based searches and sampling-based planners (e.g., Bohlin and Kavraki, 2000; Branicky et al., 2001; Cohen et al., 2014; Hauser, 2015; Helmert, 2006; Salzman and Halperin, 2016; Sànchez and Latombe, 2002).

A brief set of extensions to BIT* are presented (Section 5). These include prioritizing an initial solution, avoiding the need to define *a priori* search limits in unbounded problems, and avoiding unreachable areas of the problem domain. These ideas also motivate the development of SORRT* as an extension of batch-ordered search to the algorithmic simplicity of RRT* (Algorithm 6). A version of SORRT* is publicly available in OMPL.

The benefits of BIT* are demonstrated experimentally on abstract planning problems and simulated experiments for HERB (Section 6). The results highlight the advantages and disadvantages of both using an ordered search and considering multiple connections per sample. As state dimension increases, BIT* becomes more likely to have found a solution and generally finds better solutions faster than the other almost-surely asymptotically optimal planners.

The experiments also highlight the relative sensitivity of anytime planners to their tuning parameters. The performance of RRT-style planners depends heavily on the maximum edge length, η, and achieving the best performance requires tuning it for the problem size, dimension, and even obstacle characteristics. Alternatively, the same batch size was used for BIT* on all the tested problems even though further tuning on specific problems could provide better performance (Figure 16). This result should motivate future research on more advanced sample addition procedures, including variable and adaptive batch sizes.

Using heuristics to avoid unnecessary edge evaluations allows BIT* to spend more computational effort on the edges that are evaluated. Xie et al. (2015) showed that a two-point BVP solver can be used to calculate edges for BIT* for problems with differential constraints. They find



(a) $\mathbb{R}^2$: Runs solved vs. time
(b) $\mathbb{R}^8$: Runs solved vs. time
(c) $\mathbb{R}^2$: Solution cost vs. time
(d) $\mathbb{R}^8$: Solution cost vs. time

Fig. 16. Planner performance versus time of BIT* with various batch sizes on the problem illustrated in Figure 6(a). Each configuration was run 100 different times in $\mathbb{R}^2$ and $\mathbb{R}^8$ with run times limited to 1 and 30 seconds, respectively. The percentage of trials solved is plotted versus run time and presented in (a) and (b). The median path length is plotted versus run time and presented in (c) and (d) with unsuccessful trials assigned infinite cost. The error bars denote a non-parametric 99% confidence interval on the median. It appears that decreasing the batch size decreases the median solution time towards a problem-specific threshold. The median initial solution time was the same in $\mathbb{R}^2$ for batch sizes of 5 and 10 samples, but progressively higher for larger batches (a). It was equivalent for batches of 5, 10, 50, and 100 samples in $\mathbb{R}4$ (not shown) and for all tested batch sizes other than $5,000$ in $\mathbb{R}^8$ (b). Decreasing the batch size also appears to decrease the rate of convergence towards the optimum, with the effect becoming more pronounced in higher state dimensions (d). It is not clear how universal these relationships are between obstacle configurations.

that doing so is competitive to state-of-the-art optimal sampling-based techniques that are explicitly designed to avoid solving two-point BVPs. Choudhury et al. (2016) showed that a path optimizer (i.e., CHOMP; Zucker et al., 2013) can be used on potential edges in BIT*. This provides a method to exploit local problem information (i.e., cost gradients) to propose higher-quality edges and improve performance.

BIT* is described as using a LPA**ordering* to efficiently search an incrementally built (i.e., changing) RGG embedded in a continuous planning problem. While the search order is the same, it is important to note a key difference in how these two algorithms reuse information. When LPA* updates the cost-to-come of a vertex it reconsiders the cost-to-come of all possibly descendent vertices. This can be prohibitively expensive in large graphs and the results of RRT* demonstrate that this propagation is unnecessary for a planner to almost-surely converge asymptotically to the optimum. By not propagating these changes, BIT* performs a truncated rewiring similar to TLPA*.

This article has demonstrated the benefits of unifying informed graph-based search and sampling-based planning. Using incremental search techniques to efficiently search an increasingly dense RGG allows BIT* to outperform existing anytime almost-surely asymptotically optimal planners. These results will hopefully motivate further research into combining graph-based search and sampling-based planning. Of particular interest would be probabilistic statements about search efficiency analogous to the formal statements for A*.

There is also a clear opportunity to consider different anytime approximations, such as deterministic sampling (Janson et al., 2018) or adaptive meshes (Yershov and Frazzoli, 2016), and more advanced graph-based-search techniques, such as Anytime Repairing A* (ARA*; Likhachev et al., 2008) and Multi-Heuristic A* (MHA*; Aine et al., 2015), to further accelerate the search performance of BIT*.

Acknowledgements

This work was performed while Jonathan D Gammell was at the University of Toronto, Toronto, Canada. We would like to thank the editorial board for considering this manuscript and the reviewers for their detailed comments and dedicated efforts to improve it. We would also like to thank Christopher Dellin, Michael Koval, and Jennifer King for help running the HERB experiments.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This research was supported by contributions from the Natural Sciences and Engineering Research Council of Canada (NSERC) through the NSERC Canadian Field Robotics Network (NCFRN), the Ontario Ministry of Research and Innovation's Early Researcher Award Program, and the Office of Naval Research (ONR) Young Investigator Program.

Notes

1. Library available from http://ompl.kavrakilab.org/.
2. The experiments were run on a laptop with 16 GB of RAM and an Intel i7-4810MQ processor running Ubuntu 14.04 (64-bit).

ORCID iD

Jonathan D Gammell https://orcid.org/0000-0002-1034-3889

References

Aine S and Likhachev M (2016) Truncated incremental search. *Artificial Intelligence* 234: 49–77.

Aine S, Swaminathan S, Narayanan V, Hwang V and Likhachev M (2015) Multi-heuristic A*. *The International Journal of Robotics Research* 35(1–3): 224–243.

Akgun B and Stilman M (2011) Sampling heuristics for optimal motion planning in high dimensions. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2640–2645.

Arslan O and Tsiotras P (2013) Use of relaxation methods in sampling-based algorithms for optimal motion planning. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2421–2428.

Arslan O and Tsiotras P (2015) Dynamic programming guided exploration for sampling-based motion planning algorithms. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4819–4826.

Arslan O and Tsiotras P (2016) Incremental sampling-based motion planners using policy iteration methods. In: *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pp. 5004–5009.

Bellman RE (1954) The theory of dynamic programming. *Bulletin of the American Mathematical Society (AMS)* 60(6): 503–516.

Bellman RE (1957) *Dynamic Programming*. Princeton, NJ: Princeton University Press.

Bertsekas DP (1975) Convergence of discretization procedures in dynamic programming. *IEEE Transactions on Automatic Control* 20(3): 415–419.

Bohlin R and Kavraki LE (2000) Path planning using lazy PRM. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Vol. 1, pp. 521–528.

Branicky MS, LaValle SM, Olson K and Yang L (2001) Quasi-randomized path planning. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Vol. 2, pp. 1481–1487.

Choudhury S, Gammell JD, Barfoot TD, Srinivasa SS and Scherer S (2016) Regionally Accelerated Batch Informed Trees (RABIT*): A framework to integrate local information into optimal path planning. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4207–4214.

Cohen B, Phillips M and Likhachev M (2014) Planning single-arm manipulations with *n*-arm robots. In: *Proceedings of Robotics: Science and Systems (RSS)*, Berkeley, CA.

Diankov R and Kuffner JJ Jr (2007) Randomized statistical path planning. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Dijkstra EW (1959) A note on two problems in connexion with graphs. *Numerische Mathematik* 1(1): 269–271.

Eppstein D, Paterson MS and Yao FF (1997) On nearest-neighbor graphs. *Discrete and Computational Geometry* 17(3): 263–282.

Euler L (1738) De progressionibus transcendentibus seu quarum termini generales algebraice dari nequeunt. *Commentarii academiae scientiarum Petropolitanae* 5: 36–57.

Ferguson D and Stentz A (2006) Anytime RRTs. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5369–5375.

Gammell JD (2017) *Informed Anytime Search for Continuous Planning Problems*. PhD Thesis, University of Toronto.

Gammell JD, Barfoot TD and Srinivasa SS (2018) Informed sampling for asymptotically optimal path planning. *IEEE Transactions on Robotics* 34(4): 966–984.

Gammell JD, Srinivasa SS and Barfoot TD (2014a) BIT*: Batch informed trees for optimal sampling-based planning via dynamic programming on implicit random geometric graphs. Technical Report TR-2014-JDG006, Autonomous Space Robotics Lab, University of Toronto.

Gammell JD, Srinivasa SS and Barfoot TD (2014b) BIT*: Sampling-based optimal planning via batch informed trees. In: *The Information-based Grasp and Manipulation Planning Workshop, Robotics: Science and Systems (RSS)*, Berkeley, CA.

Gammell JD, Srinivasa SS and Barfoot TD (2014c) Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2997–3004.

Gammell JD, Srinivasa SS and Barfoot TD (2015) Batch Informed Trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3067–3074.

Gilbert EN (1961) Random plane networks. *Journal of the Society for Industrial and Applied Mathematics* 9(4): 533–543.

Hart PE, Nilsson NJ and Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2): 100–107.

Hauser K (2015) Lazy collision checking in asymptotically-optimal motion planning. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2951–2957.

Helmert M (2006) The fast downward planning system. *Journal of Artificial Intelligence Research* 26: 191–246.

Hsu D, Latombe JC and Motwani R (1999) Path planning in expansive configuration spaces. *International Journal of Computational Geometry and Applications* 9(4–5): 495–512.

Janson L, Ichter B and Pavone M (2018) Deterministic sampling-based motion planning: Optimality, complexity, and performance. *The International Journal of Robotics Research* 37(1): 46–61.

Janson L and Pavone M (2013) Fast marching trees: A fast marching sampling-based method for optimal motion planning in many dimensions. In: *Proceedings of the International Symposium on Robotics Research (ISRR)*.

Janson L, Schmerling E, Clark A and Pavone M (2015) Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International Journal of Robotics Research* 34(7): 883–921.

Karaman S and Frazzoli E (2011) Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research* 30(7): 846–894.

Karaman S, Walter MR, Perez A, Frazzoli E and Teller S (2011) Anytime motion planning using the RRT*. In: *Proceedings of*

the IEEE International Conference on Robotics and Automation (ICRA), pp. 1478–1483.

Kavraki LE, Kolountzakis MN and Latombe JC (1998) Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation* 14(1): 166–171.

Kavraki LE, Švestka P, Latombe JC and Overmars MH (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4): 566–580.

Kiesel S, Burns E and Ruml W (2012) Abstraction-guided sampling for motion planning. In: *Proceedings of the Fifth Annual Symposium on Combinatorial Search (SoCS)*.

Kleinbort M, Salzman O and Halperin D (2015) Efficient high-quality motion planning by fast all-pairs r-nearest-neighbors. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2985–2990.

Kleinbort M, Salzman O and Halperin D (2016) Collision detection or nearest-neighbor search? On the computational bottleneck in sampling-based motion planning. In: *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR)*.

Koenig S, Likhachev M and Furcy D (2004) Lifelong planning A*. *Artificial Intelligence* 155(1–2): 93–146.

Kuffner JJ Jr and LaValle SM (2000) RRT-Connect: An efficient approach to single-query path planning. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Vol. 2, pp. 995–1001.

Kunz T, Thomaz A and Christensen H (2016) Hierarchical rejection sampling for informed kinodynamic planning in high-dimensional spaces. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 89–96.

LaValle SM and Kuffner JJ Jr (2001) Randomized kinodynamic planning. *The International Journal of Robotics Research* 20(5): 378–400.

Likhachev M, Ferguson D, Gordon G, Stentz A and Thrun S (2008) Anytime search in dynamic graphs. *Artificial Intelligence* 172(14): 1613–1643.

Lozano-Pérez T (1983) Spatial planning: A configuration space approach. *IEEE Transactions on Computers* 32(2): 108–120.

Muthukrishnan S and Pandurangan G (2005) The bin-covering technique for thresholding random geometric graph properties. In: *Proceedings of the Sixteenth Annual ACM–SIAM Symposium on Discrete Algorithms*, pp. 989–998.

Otte M and Frazzoli E (2014) RRTX: Real-time motion planning/replanning for environments with unpredictable obstacles. In: *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, Istanbul, Turkey.

Otte M and Frazzoli E (2016) RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning. *The International Journal of Robotics Research* 35(7): 797–822.

Penrose M (2003) *Random Geometric Graphs* (*Oxford Studies in Probability*, Vol. 5). Oxford: Oxford University Press.

Perez A, Karaman S, Shkolnik A, Frazzoli E, Teller S and Walter MR (2011) Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4307–4313.

Persson SM and Sharf I (2014) Sampling-based A* algorithm for robot path-planning. *The International Journal of Robotics Research* 33(13): 1683–1708.

Sallaberger CS and D'Eleuterio GM (1995) Optimal robotic path planning using dynamic programming and randomization. *Acta Astronautica* 35(2–3): 143–156.

Salzman O and Halperin D (2015) Asymptotically-optimal motion planning using lower bounds on cost. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4167–4172.

Salzman O and Halperin D (2016) Asymptotically near-optimal RRT for fast, high-quality motion planning. *IEEE Transactions on Robotics* 32(3): 473–483.

Sánchez G and Latombe JC (2002) On delaying collision checking in PRM planning: Application to multi-robot coordination. *The International Journal of Robotics Research* 21(1): 5–26.

Srinivasa S, Berenson D, Cakmak M, et al. (2012) HERB 2.0: Lessons learned from developing a mobile manipulator for the home. *Proceedings of the IEEE* 100(8): 1–19.

Şucan IA, Moll M and Kavraki LE (2012) The Open Motion Planning Library. *IEEE Robotics and Automation Magazine* 19(4): 72–82.

Teniente EH and Andrade-Cetto J (2013) HRA*: Hybrid randomized path planning for complex 3D environments. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1766–1771.

Urmson C and Simmons R (2003) Approaches for heuristically biasing RRT growth. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vol. 2, pp. 1178–1183.

Xie C, van den Berg J, Patil S and Abbeel P (2015) Toward asymptotically optimal motion planning for kinodynamic systems using a two-point boundary value problem solver. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4187–4194.

Yershov DS and Frazzoli E (2016) Asymptotically optimal feedback planning using a numerical Hamilton–Jacobi–Bellman solver and an adaptive mesh refinement. *The International Journal of Robotics Research* 35(5): 565–584.

Zucker M, Ratliff N, Dragan AD, Pivtoraiko M, Klingensmith M, Dellin CM, Bagnell JA and Srinivasa SS (2013) CHOMP: Covariant Hamiltonian optimization for motion planning. *The International Journal of Robotics Research* 32(9–10): 1164–1193.