2016 IEEE International Conference on Robotics and Automation (ICRA)
Stockholm, Sweden, May 16-21, 2016

Regionally Accelerated Batch Informed Trees (RABIT*): A Framework to Integrate Local Information into Optimal Path Planning

Sanjiban Choudhury[1], Jonathan D. Gammell[2],
Timothy D. Barfoot[2], Siddhartha S. Srinivasa[1], and Sebastian Scherer[1]

***Abstract*— Sampling-based optimal planners, such as RRT*, almost-surely converge asymptotically to the optimal solution, but have provably slow convergence *rates* in high dimensions. This is because their commitment to finding the global optimum compels them to prioritize *exploration* of the entire problem domain even as its size grows exponentially. Optimization techniques, such as CHOMP, have fast convergence on these problems but only to *local* optima. This is because they are *exploitative*, prioritizing the immediate improvement of a path even though this may not find the global optimum of nonconvex cost functions.**

In this paper, we present a hybrid technique that integrates the benefits of both methods into a single search. A key insight is that applying local optimization to a subset of edges likely to improve the solution avoids the prohibitive cost of optimizing every edge in a global search. This is made possible by Batch Informed Trees (BIT*), an informed global technique that orders its search by potential solution quality. In our algorithm, Regionally Accelerated BIT* (RABIT*), we extend BIT* by using optimization to exploit local domain information and find alternative connections for edges in collision and *accelerate* the search. This improves search performance in problems with difficult-to-sample homotopy classes (e.g., narrow passages) while maintaining almost-sure asymptotic convergence to the global optimum.

Our experiments on simulated random worlds and real data from an autonomous helicopter show that on certain difficult problems, RABIT* converges 1.8 times faster than BIT*. Qualitatively, in problems with difficult-to-sample homotopy classes, we show that RABIT* is able to efficiently transform paths to avoid obstacles.

I. Introduction

Sampling-based planners are popular for solving the motion-planning problem in robotics and are effective on a large range of applications. Algorithms such as Probabilistic Roadmaps (PRM) [1], Rapidly-exploring Random Trees (RRT) [2], and Expansive Space Trees (EST) [3] are *probabilistically complete*. These algorithms find a solution, if one exists, with probability one as the number of samples goes to infinity.

Recent research has developed *asymptotically optimal* planners, such as RRT* [4]. The solutions found by these algorithms converge asymptotically to the optimum, if one exists, with probability one as the number of samples goes to infinity (i.e., *almost surely*). With informed search techniques, such as Informed RRT*, this convergence can be linear in the absence of obstacles [5], but generally convergence is provably poor, especially in high dimensions [6]. This is because the algorithms prioritize *exploring* the domain despite its exponential growth in size with increased state dimension.

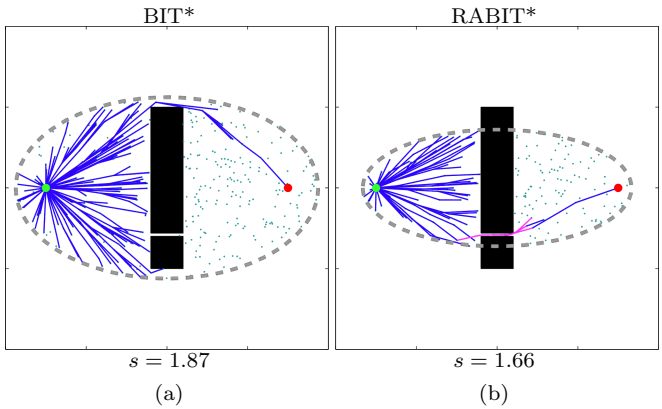


Fig. 1. Solutions found by BIT* and RABIT* after 0.1 seconds of computation time on a simulated 2-dimensional world. The world has an obstacle with a narrow gap that creates three homotopy classes, two of which flank the obstacle and one that passes through the narrow gap. BIT*'s ability to find a path through the narrow gap depends on the distribution of the random samples. RABIT* judiciously applies a local optimizer to potential edges (shown in magenta), allowing it to exploit additional problem information and find paths through difficult-to-sample homotopy classes (e.g., narrow passages) faster than sampling alone.

Local optimizers, such as Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [7], instead prioritize *exploiting* domain information, such as cost gradients, to modify and improve a path. This rapidly finds high-quality solutions in high-dimensional planning problems, but only provides guarantees on *local* optimality. This is because nonconvex cost functions have local optima that can entrap greedy optimization methods, such as gradient descent. This makes the relative suitability of local optimizers and global searches dependent on the specific planning problem [8].

In the field of nonconvex optimization, it is common to avoid these limitations by combining local optimization with a global search [9]–[14]. This provides the rapid convergence of local optimizers (e.g., gradient descent) with the insensitivity to local optima of a global search (e.g., stochastic search).

In this paper, we use this existing work as motivation for a framework to *integrate* local and global planning methods into a hybrid search. We do so by using Batch Informed Trees (BIT*) [15], a global method that uses heuristics to search in order of potential solution quality. This provides the efficiency necessary for a local optimizer to generate potential edges from domain information (e.g., cost gradients). This

[1] S. Choudhury, S. S. Srinivasa and S. Scherer are with The Robotics Institute at Carnegie Mellon University, Pittsburgh, Pennsylvania, USA. Email: {sanjibac, basti, siddh}@cs.cmu.edu

[2] J. D. Gammell and T. D. Barfoot are with the Autonomous Space Robotics Lab at the University of Toronto, Toronto, Ontario, Canada. Email: {jon.gammell, tim.barfoot}@utoronto.ca

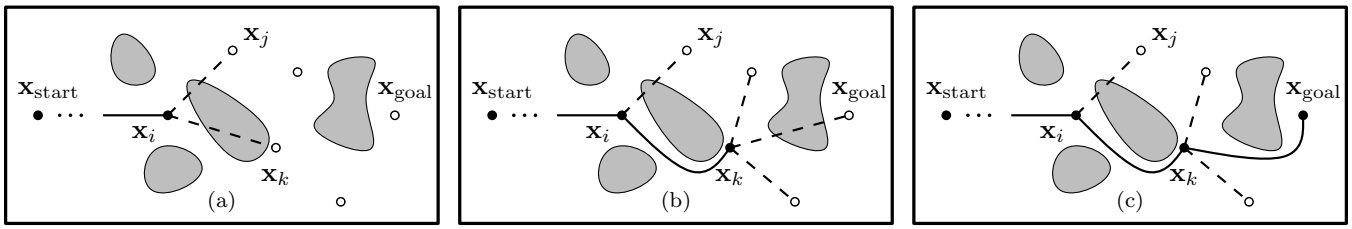


Fig. 2. An illustration of how the RABIT* algorithm uses a local optimizer to exploit obstacle information and improve a global search. The global search is performed, as in BIT*, by incrementally processing an edge queue (dashed lines) into a tree (a). Using heuristics, the potential edge from $\mathbf{x}_i$ to $\mathbf{x}_k$ is processed first as it could provide a better solution than an edge from $\mathbf{x}_i$ to $\mathbf{x}_j$. The initial straight-line edge is given to a local optimizer which uses information about obstacles to find a local optima between the specified states (b). If this edge is collision free, it is added to the tree and its potential outgoing edges are added to the queue. The next-best edge in the queue is then processed in the same fashion, using the local optimizer to once again propose a better edge than a straight-line (c).

extra information helps the algorithm find difficult-to-sample homotopy classes (e.g., narrow passages, Fig. 1). The resulting algorithm, Regionally Accelerated BIT* (RABIT*), maintains almost-sure asymptotic convergence to the global optimum.

We demonstrate this technique with CHOMP [7] as a gradient-based local optimizer (Fig. 2). The algorithm is tested in various state dimensions on simulated random worlds with narrow passages and an example problem from an autonomous helicopter flight. These experiments show that RABIT* is 'CHOMPing at the BIT*' [*sic*] to find solutions to challenging high-dimensional problems with difficult-to-sample homotopy classes, outperforming both existing asymptotically optimal algorithms and local optimization.

The remainder of this paper is organized as follows. Section II is a review of relevant literature, while Section III contains formal definitions and notation. The algorithm is presented in Section IV and Section V presents experimental results. Finally, Sections VI and VII present a discussion with thoughts on future work and a conclusion.

II. Background

Substantial work exists on improving the solution quality of sampling-based planners, including adaptations to search techniques, local optimization methods, and hybrid searches.

A. Adapted Search Techniques

Many adaptations exist to the RRT search procedure. Urmson and Simmons [16] use a heuristic to bias sample generation in an RRT while Ferguson and Stentz [17] use a series of independent RRTs in their Anytime RRTs algorithm. Jaillet et al. [18] combine RRT with stochastic optimization techniques in their Transition-based RRT (T-RRT) algorithm, while Rickert et al. [19] attempt to balance exploration and exploitation through gradient information in their Exploring/Exploiting Tree (EET) algorithm. Though these techniques are effective, their asymptotic optimality is limited by the underlying RRT.

Karaman and Frazzoli [4] incrementally rewire the RRT graph using random geometric graph (RGG) theory to achieve asymptotic optimality in their algorithm, RRT*. Recent work has focused on improving the convergence rate of asymptotically optimal planners.

Alterovitz et al. [20], Akgun and Stilman [21], and Nasir et al. [22] all use path-biased sampling in their algorithms. This increases the likelihood of sampling near the current solution and the convergence to a local optimum, but results in a nonuniform distribution that can decrease the likelihood of finding solutions in *other* homotopy classes.

Gammell et al. [5] improve convergence for problems seeking to minimize path length by directly sampling a (upper-bound) heuristic in their Informed RRT* algorithm. Although this has linear convergence to the optimum in the absence of obstacles, the presence of obstacles in practical problems prevents the subproblem from shrinking indefinitely.

Arslan and Tsiotras [23], [24] use dynamic programming [25] and Lifelong Planning A* (LPA*) techniques [26] in their RRT$^{\#}$ algorithm to improve RRT* rewiring. This improves convergence but does not directly focus the search.

Janson et al. [6] use a marching method on a set of samples in their Fast Marching Tree (FMT*) algorithm. The search expands outward from the start in order of increasing cost-to-come; however, it is not anytime and must be restarted if more samples are needed to find a solution. Salzman and Halperin [27] extend FMT* to quasi-anytime performance with their Motion Planning Using Lower Bounds (MPLB) algorithm. Denser sets of samples are searched using lower-bounding estimates of the solution cost through states, with improved solutions being found only when a search finishes. They state that this can be done efficiently by reusing information, but they do not provide specific methods to do so.

Gammell et al. [15] combine incremental graph-search techniques with RGG theory in their BIT* algorithm to create an anytime asymptotically optimal search that checks potential solutions in order of estimated cost. This is done efficiently by using heuristics to search batches of samples and has proven effective even for problems with differential constraints that require solving two-point boundary-value problems (2-pt BVPs) [28]. This paper builds on Gammell et al. [15] by incorporating local search to improve convergence.

B. Local Optimizers

Local optimization methods focus on improving an initial suboptimal path towards a local optimum. All these methods can be used to post-process results from global searches and some can be used to solve a problem directly. While optimization can occasionally switch between topologically close classes, these methods are generally limited to the homotopy class of the initial path.

Basic techniques seek to simplify the initial path by removing redundant states through path pruning or path

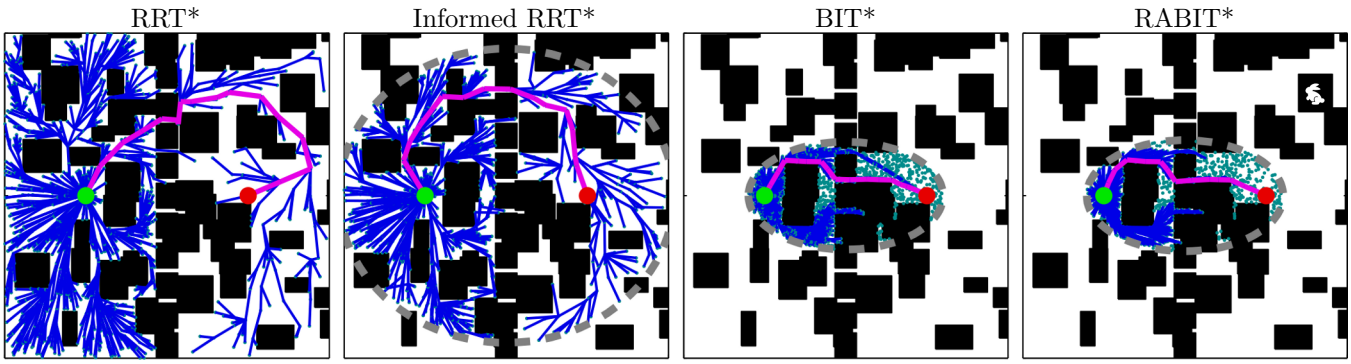


Fig. 3. The results after 0.2 seconds of RRT*, Informed RRT*, BIT*, and RABIT* on the random $\mathbb{R}^2$ world analyzed in Fig. 4. Note how RABIT* uses local information to find a path through the optimum narrow gap with fewer samples than BIT*. While in easier low-dimensional problems this does not result in better performance than BIT*, it does on analogous problems in $\mathbb{R}^8$.

shortcutting [29]–[33]. Path pruning iteratively improves a discrete path by considering new connections between existing vertices, while path shortcutting performs a similar procedure but also considers interpolating edges to create new vertices. In both, when a feasible connection is found the path is simplified by using it and removing the intermediate vertices.

More advanced techniques seek to optimize an initial path, independent of its feasibility, by exploiting additional information about the problem domain (e.g., cost gradients). These can be used independent of a global search; however, they may become stuck in the local optima of nonconvex cost functions.

Zucker et al. [7] use gradient methods to optimize an initial solution in their CHOMP algorithm. Kalakrishnan et al. [34] use stochastic methods to replace analytical gradients in their Stochastic Trajectory Optimization for Motion Planning (STOMP) algorithm. Choudhury et al. [35] optimize systems with dynamic constraints in their Dynamics Projection Filter (DPF) algorithm. These techniques associate a cost to obstacles and perform optimization of an unconstrained cost function. This allows for the rapid discovery of local optima, but does not guarantee that the path will be feasible (i.e., collision-free). Schulman et al. [36] perform optimization constrained by obstacle avoidance in their trajectory optimization (TrajOpt) algorithm; however, it is still susceptible to local optima.

C. Hybridization

Hybrid search techniques combine the results of multiple, possibly different, search algorithms into a solution that is better than that of the individual inputs.

Basic techniques combine the results of completed searches [37], [38]. This allows for a wide variety of methods as inputs, including both global and local techniques, but keeps each search independent. This means information discovered by one search is not shared with others, a limitation that is especially problematic in domains with difficult-to-sample features such as narrow passages.

Otte and Correll [39] use a parallel hybrid method in their Coupled Forest of Random Engrafting Search Trees (C-FOREST) algorithm. Demonstrated with RRT*, the algorithm shares information between multiple sampling-based planners using heuristics and rejection sampling. Compared to other parallel algorithms [40], this results in a superlinear speedup in computation time; however, they only use global searches.

D. Regionally Accelerated BIT* (RABIT*)

In this paper, we present an anytime hybrid search that integrates local information into a global search. A local optimizer is used to exploit additional domain information (e.g., cost gradients) to divert edges around obstacles. When combined with BIT*'s ordered search, this efficiently exploits local information to find difficult-to-sample features while continuing to explore the entire problem domain and almost-surely converge asymptotically to the global optimum. This is analogous to the way 2-pt BVP-solvers can be used to plan with dynamic constraints [28].

III. Definitions

We define the asymptotically optimal planning problem as in [4] as well as some mathematical notation.

Definition 1 (Optimal Planning): 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 $X_{\text{free}} = X \setminus X_{\text{obs}}$ be the resulting set of permissible states. 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 $\sigma : [0,1] \to X$ be a sequence of states (a path) and Σ be the set of all nontrivial paths.

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

$$\sigma^* = \arg\min_{\sigma \in \Sigma} \{ s(\sigma) \mid \sigma(0) = \mathbf{x}_{\text{start}}, \sigma(1) \in X_{\text{goal}}, \forall t \in [0,1], \sigma(t) \in X_{\text{free}} \},$$

where $\mathbb{R}_{\geq 0}$ is the set of non-negative real numbers. We say a planner almost-surely converges asymptotically to this optimum if the probability of convergence is unity as the number of iterations, i, goes to infinity,

$$P\left(\limsup_{i \to \infty} s(\sigma_i) = s(\sigma^*) \right) = 1.$$

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

1 $V \leftarrow \{\mathbf{x}_{\text{start}}\}$; $E \leftarrow \emptyset$; $X_{\text{samples}} \leftarrow \{\mathbf{x}_{\text{goal}}\}$;
2 $\mathcal{Q}_V \leftarrow \emptyset$; $\mathcal{Q}_E \leftarrow \{(\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}})\}$; $r \leftarrow \infty$;
3 **repeat**
4 **if** $\mathcal{Q}_V \equiv \emptyset$ **and** $\mathcal{Q}_E \equiv \emptyset$ **then**
5 Prune $(g_{\mathcal{T}}(\mathbf{x}_{\text{goal}}))$;
6 $X_{\text{samples}} \xleftarrow{+} \text{Sample}(m, g_{\mathcal{T}}(\mathbf{x}_{\text{goal}}))$;
7 $r \leftarrow \text{radius}(|V| + |X_{\text{samples}}|)$;
8 $\mathcal{Q}_V \leftarrow V$;
9 **while** BestQueueValue$(\mathcal{Q}_V) \leq$ BestQueueValue$(\mathcal{Q}_E)$ **do**
10 ExpandVertex(BestInQueue$(\mathcal{Q}_V)$);
11 ProcessEdge(BestInQueue$(\mathcal{Q}_E)$);
12 **until** STOP;
13 **return** $\mathcal{T}$;

Algorithm 2: ProcessEdge$((\mathbf{v}_{\text{m}}, \mathbf{x}_{\text{m}}) \in \mathcal{Q}_E)$

1 $\mathcal{Q}_E \xleftarrow{-} \{(\mathbf{v}_{\text{m}}, \mathbf{x}_{\text{m}})\}$;
2 **if** $g_{\mathcal{T}}(\mathbf{v}_{\text{m}}) + \widehat{s}(\mathbf{v}_{\text{m}}, \mathbf{x}_{\text{m}}) + \widehat{h}(\mathbf{x}_{\text{m}}) < g_{\mathcal{T}}(\mathbf{x}_{\text{goal}})$ **then**
3 $\sigma_{(\mathbf{v}_{\text{m}}, \mathbf{x}_{\text{m}})} \leftarrow \text{OptimizeEdge}((\mathbf{v}_{\text{m}}, \mathbf{x}_{\text{m}}))$;
4 **if** $\widehat{g}(\mathbf{v}_{\text{m}}) + s(\sigma_{(\mathbf{v}_{\text{m}}, \mathbf{x}_{\text{m}})}) + \widehat{h}(\mathbf{x}_{\text{m}}) < g_{\mathcal{T}}(\mathbf{x}_{\text{goal}})$ **then**
5 **if** $g_{\mathcal{T}}(\mathbf{v}_{\text{m}}) + s(\sigma_{(\mathbf{v}_{\text{m}}, \mathbf{x}_{\text{m}})}) < g_{\mathcal{T}}(\mathbf{x}_{\text{m}})$ **then**
6 **if** $\mathbf{x}_{\text{m}} \in V$ **then**
7 $E \xleftarrow{-} \{(\mathbf{v}, \mathbf{x}_{\text{m}}) \in E\}$;
8 **else**
9 $X_{\text{samples}} \xleftarrow{-} \{\mathbf{x}_{\text{m}}\}$;
10 $V \xleftarrow{+} \{\mathbf{x}_{\text{m}}\}$; $\mathcal{Q}_V \xleftarrow{+} \{\mathbf{x}_{\text{m}}\}$;
11 $E \xleftarrow{+} \{(\mathbf{v}_{\text{m}}, \mathbf{x}_{\text{m}})\}$;
12 $\mathcal{Q}_E \xleftarrow{-} \{(\mathbf{v}, \mathbf{x}_{\text{m}}) \in \mathcal{Q}_E \mid g_{\mathcal{T}}(\mathbf{v}) + \widehat{s}(\mathbf{v}, \mathbf{x}_{\text{m}}) \geq g_{\mathcal{T}}(\mathbf{x}_{\text{m}})\}$;
13 **else**
14 $\mathcal{Q}_V \leftarrow \emptyset$; $\mathcal{Q}_E \leftarrow \emptyset$;

A. Notation

We define $\mathcal{T} := (V, E)$ to 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 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 $g_{\mathcal{T}}(\mathbf{x})$ represents the cost-to-come to a state, $\mathbf{x} \in X$, through the tree. We assume a state not in the tree has a cost-to-come of infinity. Note that these two functions will always bound the unknown true optimal cost to a state, $g(\cdot)$, i.e., $\forall \mathbf{x} \in X$, $\widehat{g}(\mathbf{x}) \leq g(\mathbf{x}) \leq g_{\mathcal{T}}(\mathbf{x})$.

There exists a path, $\sigma_{(\mathbf{v},\mathbf{w})} : [0, 1] \to X$, for every edge, $(\mathbf{v}, \mathbf{w})$, in the tree such that $\sigma_{(\mathbf{v},\mathbf{w})}(0) = \mathbf{v}$ and $\sigma_{(\mathbf{v},\mathbf{w})}(1) = \mathbf{w}$. The functions $\widehat{s}(\sigma_{(\mathbf{v},\mathbf{w})})$ and $s(\sigma_{(\mathbf{v},\mathbf{w})})$ represents an admissible estimate and the true cost of this path, respectively. With a slight abuse of notation, we denote the costs of straight-line edges as simply $\widehat{s}(\mathbf{v}, \mathbf{w})$ and $s(\mathbf{v}, \mathbf{w})$.

Finally, the function $\lambda(\cdot)$ represents the Lebesgue measure of a set (e.g., the *volume*), the cardinality of a set is denoted by $|\cdot|$, and we take the minimum of an empty set to be infinity, as is customary. We use the notation $X \xleftarrow{+} \{\mathbf{x}\}$ and $X \xleftarrow{-} \{\mathbf{x}\}$ to compactly represent the compounding set operations $X \leftarrow X \cup \{\mathbf{x}\}$ and $X \leftarrow X \setminus \{\mathbf{x}\}$, respectively. The trace of a matrix is given by $\text{tr}(\cdot)$ and the Euclidean norm of a vector or matrix is denoted by $||\cdot||_2$. We use ∇ and $\boldsymbol{\nabla}$ to denote the *vector* and *matrix* gradients of a scalar function, respectively.

IV. Algorithm

We present RABIT*, a hybrid search technique. It generates edges for a global search that avoid obstacles by using an optimizer to exploit local domain information. It does this while maintaining almost-sure asymptotic convergence to the global optimum.

The version presented uses CHOMP (Section IV-B) to exploit cost gradients inside BIT* (Section IV-A); however, other local optimization [34]–[36] or search techniques would also be appropriate. CHOMP is a local optimization algorithm that uses a quasi-Newton method to quickly improve a path by exploiting cost-gradient information. This not only improves the quality of the global search, but also its ability to find difficult-to-sample homotopy classes.

For simplicity, Algs. 1–4 present the search from a single start state to a single goal state. This formulation could easily be extended to a goal region by extending the heuristic, or to a search originating from the goal.

A. Global Search

RABIT* uses the principles developed in BIT*. Due to space constraints, we present only basic implementation details and direct the reader to [15] for more information.

BIT* is an asymptotically optimal planning algorithm that orders its search on estimated solution cost and uses the current solution to bound the search domain. This ordering provides the opportunity to apply a local optimizer efficiently in a global search. For clarity, we separate the algorithm into a main section that is unchanged (Alg. 1) and the processing of edges (Alg. 2) where we highlight the changes in RABIT*.

The algorithm starts with a given initial state, $\mathbf{x}_{\text{start}} \in X_{\text{free}}$, in an otherwise empty tree, $\mathcal{T}$. The goal state, $\mathbf{x}_{\text{goal}} \in X_{\text{goal}}$, is placed in the set of unconnected samples, X_{samples}, and the search queues are initialized (Alg. 1, Lines 1–2).

Whenever the queues are empty (Alg. 1, Line 4), RABIT* starts a new batch (Alg. 1, Lines 5–8). All vertices that cannot improve the solution are removed (Alg. 1, Line 5, see [15]) and more samples are added (Alg. 1, Line 6). This creates a new, denser *implicit* RGG from both the existing and new states, defined by a connection radius, r (Alg. 1, Line 7). The connected vertices are then inserted into the vertex-expansion queue to start the search (Alg. 1, Line 8).

The connection radius is calculated from bounds for asymptotic optimality [4],

$$\text{radius}(q) := 2\eta\left(1 + \tfrac{1}{n}\right)^{\frac{1}{n}} \left(\frac{\lambda(X_{\widehat{f}})}{\zeta_n}\right)^{\frac{1}{n}} \left(\frac{\log(q)}{q}\right)^{\frac{1}{n}}, \quad (1)$$

where q is the number of vertices, $X_{\widehat{f}}$ is the subset of states that can provide a better solution [5], ζ_n is the volume of an n-dimension unit ball, and $\eta > 1$ is a tuning parameter.

The underlying implicit RGG is searched in order of decreasing solution quality using a queue-based incremental graph search (Alg. 1, Lines 9–11) that expands vertices only when necessary (Alg. 1, Lines 9–10). Vertices are expanded

Algorithm 3: OptimizeEdge($(\mathbf{v}, \mathbf{w}) \in \mathcal{Q}_E$)

```
1 if ŝ(v, w) < s(v, w) then
2     σ' ← Optimize({v, w});
3     if s(σ') < s(v, w) then
4         return σ';
5     else
6         return {v, w};
7 else
8     return {v, w};
```

Algorithm 4: Optimize (σ)

```
1 S' ← σ;
2 if ||σ(1) − σ(o)||_2 < γ then
3     if tr(∇c(S')^T ∇c(S')) / c(S') ≥ ν then
4         for i = 0, 1, ..., i_max and ||∇c(S')||_2 ≥ ε do
5             ΔS ← −α_i^{-1} A^{-1} ∇c(S');
6             S' ← S' + ΔS;
7 σ' ← S';
8 return σ';
```

from a vertex queue, $\mathcal{Q}_V$, into an edge queue, $\mathcal{Q}_E$, when they could provide a better edge than the best of the edge queue (Alg. 1, Line 9). The functions `BestQueueValue` and `BestInQueue` return the value of the best element and the best element itself, respectively, for either the vertex or edge queues. The ordering value of a vertex, $\mathbf{v}$, or an edge, $(\mathbf{v}, \mathbf{x})$, in its queue is a lower bounding estimate of the cost of a solution constrained to pass through the vertex or edge given the current tree, $g_{\mathcal{T}}(\mathbf{v}) + \widehat{h}(\mathbf{v})$, and $g_{\mathcal{T}}(\mathbf{v}) + \widehat{s}(\mathbf{v}, \mathbf{x}) + \widehat{h}(\mathbf{x})$, respectively. A vertex is expanded into the edge queue by adding the potential edges from the vertex to all other vertices within the distance r (Alg. 1, Line 10, see [15]).

As time allows, RABIT* processes the edge queue (Alg. 1, Line 11), increasing the density of the RGG with new batches of samples when necessary.

Potential edges are processed by evaluating increasingly accurate cost calculations (Alg. 2). This allows optimizations and collision checks to be delayed until necessary, limiting computational effort. The cost of a solution through the edge is first estimated using an admissible estimate of the edge cost (Alg. 2, Line 2). If this cannot improve the current solution, then neither can the rest of the queue and the batch is finished (Alg. 2, Line 14). RABIT* then applies the local optimizer in an attempt to find a collision-free path for the proposed edge (Alg. 2, Line 3, Section IV-B). The solution cost through this path is then estimated using the actual path cost (Alg. 2, Line 4). If this cannot improve the solution or it is in collision, then the potential edge is discarded. Finally, the effect of the path on the existing graph is evaluated, if it does not improve the cost-to-come of the target vertex then it is discarded (Alg. 2, Line 5).

If the new path passes all these checks, it is added to the graph as an edge (Alg. 2, Lines 6–12), either improving the path to an existing state (a *rewiring*) or connecting a new state (an *expansion*). For rewirings, the existing edge in the graph is removed (Alg. 2, Line 7). For expansions, the vertex is moved from the set of unconnected samples and added to the graph (Alg. 2, Lines 9–10). Finally, the queue is pruned of any redundant edges incoming to the vertex that cannot provide a better path (Alg. 2, Line 12).

B. Local Optimization

Local optimization is used to exploit domain information to generate high-quality potential edges for the global search. Using obstacle information finds collision-free edges and helps the global search find paths through difficult-to-sample homotopy classes.

The integration of the local optimizer into the global search is presented in Alg. 3. Edges are only optimized if the heuristic predicts that a better path is possible (Alg. 3, Line 1), otherwise the original straight-line path is returned (Alg. 3, Line 8). If a better path may exist, the optimizer is applied to the straight-line edge between the two states (Alg. 3, Line 2). If the cost of the optimized path is less than the cost of the original, then the function returns the optimized path (Alg. 3, Lines 3–4). If not, it returns the straight-line path (Alg. 3, Line 6). This comparison allows for the integration of local optimization methods that do not guarantee obstacle avoidance or minimize a different cost function (e.g., CHOMP) into the global search. Note that by definition, calculating the path cost includes collision checks.

1) Covariant Hamiltonian Optimization for Motion Planning (CHOMP): In this version of RABIT*, we use CHOMP as a local optimizer to exploit cost gradients (Alg. 4). Due to space constraints, we present only basic implementation details and direct the reader to [7] for more information, including on path parameterizations. We use a discretized straight-line parameterization to represent paths as matrices, $\mathbf{S} \in \mathbb{R}^{z \times n}$, where z is a number of *intermediate* waypoints between the start and end of the path, a single tuning parameter. These waypoints are internal to CHOMP and are not considered vertices in the RABIT* graph.

To reduce optimization time, we skip paths that CHOMP cannot improve efficiently. To avoid low-resolution paths, we skip those longer than a user-tuned threshold (Alg. 4, Line 2). To avoid paths near local optima, we compare the trace of a paths cost gradient to its cost. If this ratio is insufficiently large, then the path is already near a local optima and is not optimized further (Alg. 4, Line 3). The iterative CHOMP procedure is then repeated for a specified number of iterations, $i_{\max}$, while the gradient is sufficiently large (Alg. 4, Line 4).

CHOMP minimizes a cost function, $c(\cdot) \in \mathbb{R}_{\geq 0}$, that combines path smoothness and obstacle avoidance,

$$c(\mathbf{S}) := \operatorname{tr}\left(0.5\, \mathbf{S}^T \mathbf{A} \mathbf{S} + \mathbf{S}^T \mathbf{B} + \mathbf{C}\right) + \lambda\, c_{\text{obs}}(\mathbf{S}),$$

where $\lambda \in \mathbb{R}_{\geq 0}$ is a user-tuned weighting parameter.

The smoothness matrix terms, $\mathbf{A} \in \mathbb{R}^{z \times z}$, $\mathbf{B} \in \mathbb{R}^{z \times n}$, and $\mathbf{C} \in \mathbb{R}^{n \times n}$ are,

$$\mathbf{A} := \begin{bmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{bmatrix},$$

$$\mathbf{B} := \begin{bmatrix} -\mathbf{v} & \mathbf{0} & \dots & \mathbf{0} & -\mathbf{w} \end{bmatrix}^T,$$

and

$$\mathbf{C} := 0.5\left(\mathbf{v}\mathbf{v}^T + \mathbf{w}\mathbf{w}^T\right),$$

where $\mathbf{v}$ and $\mathbf{w}$ are the start and end states of the path, respectively.

The obstacle cost, $c_{\text{obs}}(\cdot) \in \mathbb{R}_{\geq 0}$, is

$$c_{\text{obs}}(\mathbf{S}) := \sum_{j=1}^{z} w_{\text{obs}}(\mathbf{x}_j) ||\mathbf{x}_{j+1} - \mathbf{x}_j||_2,$$

where $\mathbf{x}_j$ is the j-th waypoint on the discretized path and the weight, $w_{\text{obs}}(\mathbf{x})$, is given by

$$w_{\text{obs}}(\mathbf{x}) := \begin{cases} 0 & \text{if } \delta(\mathbf{x}) > \epsilon \\ 0.5(\epsilon - \delta(\mathbf{x}))^2/\epsilon & \text{if } \epsilon \geq \delta(\mathbf{x}) \geq 0 \\ 0.5\epsilon - \delta(\mathbf{x}) & \text{if } \delta(\mathbf{x}) < 0, \end{cases}$$

where $\epsilon \in \mathbb{R}_{>0}$ is a tuning parameter defining obstacle clearance. The function $\delta(\mathbf{x})$ is the distance from the state to the nearest obstacle boundary and is negative when the state is inside an obstacle.

The form of this cost function allows CHOMP to use a quasi-Newton approach to iteratively optimize the path,

$$\mathbf{S}_{i+1} := \mathbf{S}_i + \mathbf{\Delta S},$$

where $\mathbf{\Delta S} \in \mathbb{R}^{z \times n}$,

$$\mathbf{\Delta S} := -\alpha_i^{-1}\mathbf{A}^{-1}\boldsymbol{\nabla} c(\mathbf{S}_i),$$

and $\forall i = 1, 2, \ldots, i_{\max},\ \alpha_i \in \mathbb{R}_{\geq 0}$ are tuning parameters that define the step size at each iteration, i, of the optimization.

The matrix gradient of the cost function, $\boldsymbol{\nabla} c(\mathbf{S}) \in \mathbb{R}^{z \times n}$, is given analytically by

$$\boldsymbol{\nabla} c(\mathbf{S}) = \mathbf{AS} + \mathbf{B} + \boldsymbol{\nabla} c_{\text{obs}}(\mathbf{S}),$$

where the j-th row of the obstacle cost, $\boldsymbol{\nabla} c_{\text{obs}}(\mathbf{S}) \in \mathbb{R}^{z \times n}$, is

$$\begin{aligned} \boldsymbol{\nabla} c_{\text{obs}}(\mathbf{S})_j = {} & \nabla w_{\text{obs}}(\mathbf{x}_j)^T ||\mathbf{x}_{j+1} - \mathbf{x}_j||_2 \\ & + w_{\text{obs}}(\mathbf{x}_{j-1}) \frac{\mathbf{x}_j^T - \mathbf{x}_{j-1}^T}{||\mathbf{x}_j - \mathbf{x}_{j-1}||_2} \\ & + w_{\text{obs}}(\mathbf{x}_j) \frac{\mathbf{x}_{j+1}^T - \mathbf{x}_j^T}{||\mathbf{x}_{j+1} - \mathbf{x}_j||_2}, \end{aligned}$$

and the vector gradient of the weight, $\nabla w_{\text{obs}}(\mathbf{x}) \in \mathbb{R}^n$, is

$$\nabla w_{\text{obs}}(\mathbf{x}) = \begin{cases} 0 & \text{if } \delta(\mathbf{x}) > \epsilon \\ -\nabla\delta(\mathbf{x})(\epsilon - \delta(\mathbf{x}))/\epsilon & \text{if } \epsilon \geq \delta(\mathbf{x}) \geq 0 \\ -\nabla\delta(\mathbf{x}) & \text{if } \delta(\mathbf{x}) < 0, \end{cases}$$

and $\nabla\delta(\cdot) \in \mathbb{R}^n$ is the vector gradient of the distance function with respect to a state. In some planning applications, this distance function and its gradient can be calculated *a priori*.

V. Experimental Results

An Open Motion Planning Library (OMPL) [41] implementation of RABIT* was evaluated on both simulated random worlds (Section V-A) and a real planning problem from an autonomous helicopter (Section V-B)[1].

A. Random Worlds

To systemically evaluate RABIT*, it was run on randomly generated worlds in $\mathbb{R}^2$ and $\mathbb{R}^8$. It was compared to publicly available OMPL implementations of RRT, RRT-Connect [42], RRT*, Informed RRT*, and BIT*.

Algorithms used an RGG constant (η in (1)) of 1.1 and approximated $\lambda(X_{\text{free}})$ with $\lambda(X)$. RRT-based algorithms used a goal bias of 5%, and a maximum edge length of 0.2 and 1.25 in $\mathbb{R}^2$ and $\mathbb{R}^8$, respectively. BIT*-based algorithms used 100 samples per batch, Euclidean distance for heuristics, and direct informed sampling [5]. Graph pruning was limited to changes in the solution cost greater than 1% and we used an approximately sorted queue as discussed in [15]. RABIT* uses CHOMP parameters of $\lambda = 100$, $\epsilon = 0.05$, $z = 8$, $\gamma = 0.05$ ($\mathbb{R}^2$) or 0.2 ($\mathbb{R}^8$), $\nu = 0.1$, $i_{\max} = 5$, $\varepsilon = 1 \times 10^{-3}$, and $\alpha_i = i^{-1/2} \times 10^{-3}$. The distance gradient was calculated analytically.

The worlds consisted of a (hyper)cube of width 2 divided in half by a wall with 10 narrow gaps. The world also contained random axis-aligned (hyper)rectangular obstacles such that at most one third of the environment was obstructed. The start and goal states were located on different sides of the wall at $[-0.5, 0.0, \ldots, 0.0]$ and $[0.5, 0.0, \ldots, 0.0]$, respectively (Fig. 3). This allowed us to randomly generate challenging planning problems that had an optimal solution passing through a difficult-to-sample narrow passage.

For each state dimension, 10 different random worlds were generated and the planners were tested with 100 different pseudo-random seeds on each world. The solution cost of each planner was recorded every 1 millisecond by a separate thread and the median was calculated by interpolating each trial at a period of 1 millisecond.

As the true optima for these problems are different and unknown, there is no meaningful way to compare the results across problems. Instead, results from a representative problem are presented in Fig. 4, where the percent of trials solved and the median solution cost are plotted versus computation time.

To quantify the results in Fig. 4, we calculate the time for each algorithm to reach 90% of its final value. In $\mathbb{R}^2$, RABIT* takes 0.215s compared to BIT*'s 0.179s. In $\mathbb{R}^8$, RABIT* takes 0.262s compared to BIT*'s 0.471s.

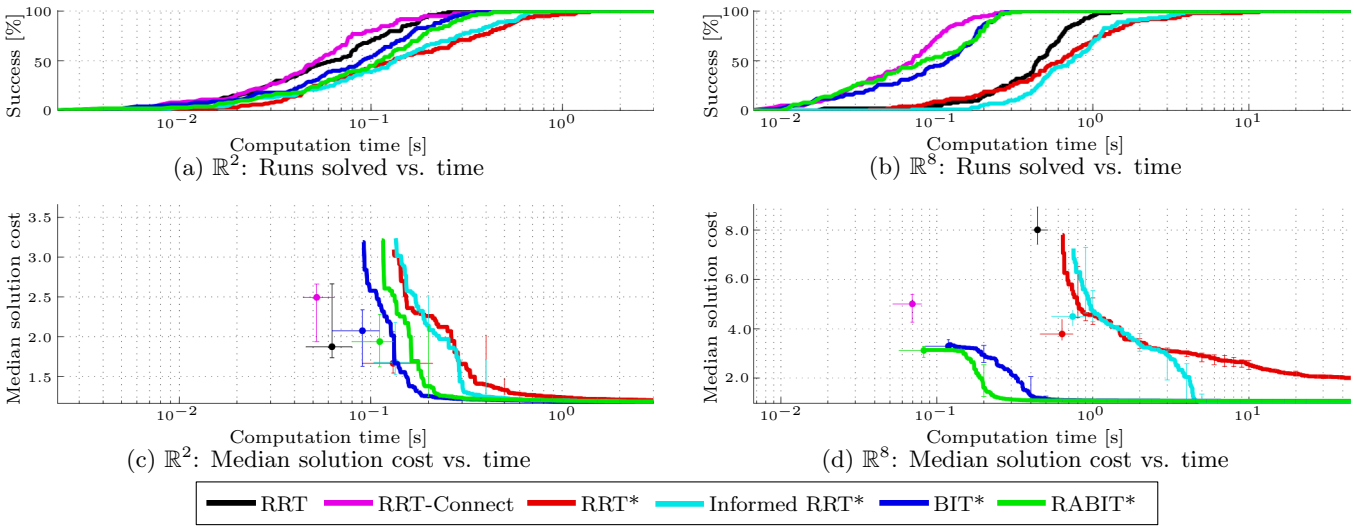
These results show that in $\mathbb{R}^2$ RABIT* performs similarly to BIT*; however, that in $\mathbb{R}^8$ RABIT* finds better solutions faster on these problems with difficult-to-sample homotopy classes.

B. Autonomous Helicopter

To evaluate RABIT* on real planning problems, it was run on a recorded flight mission of an autonomous helicopter. The autonomous helicopter operates at speeds of up to 50 meters/second in challenging environments that may contain difficult-to-sample features such as valleys. Plans must obey the dynamic and power constraints of the vehicle (including climb-rate limits) and completely avoid obstacles, a planning problem that is difficult to solve in *real time*.

Sensor data collected from test flights over Mesa, Arizona were used to propose a planning problem around mountains (Fig. 5). This problem is challenging because the helicopter's constraints create a large number of states from which no

[1]Experiments were run on a MacBook Pro with 16 GB of RAM and an Intel i7-4870HQ processor running a 64-bit version of Ubuntu 14.04.



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

(c) $\mathbb{R}^2$: Median solution cost vs. time (d) $\mathbb{R}^8$: Median solution cost vs. time

Fig. 4. The results from representative worlds in $\mathbb{R}^2$ and $\mathbb{R}^8$ for RRT, RRT-Connect, RRT*, Informed RRT*, BIT*, and RABIT*. For the chosen random worlds, (a) and (b) show the percentage of the 100 trials solved versus run time, while (c) and (d) show the median solution cost versus run time. Dots represent the median initial solution cost and time. We considered unsolved planners to have an infinite solution cost, and calculated the median from $> 50\%$ of solved trials, represented as solid lines with error bars denoting a non-parametric 95% confidence interval on median solution cost and time. For legibility, we did not plot confidence intervals with an infinite upper bound.

collision-free path can be found (inevitable collision states). These states reduce the connectivity of the free space and increase the inherently difficult problem of sampling a valid path through the narrow valley.

This problem was used to compare the ability of BIT*, CHOMP, and RABIT* to plan for a vehicle with restrictive constraints given limited computation time (2 seconds). These results demonstrate how RABIT* combines the benefits of global and local techniques. CHOMP uses cost-gradient information but can become stuck in a poor local minima when optimizing long paths, failing to find a feasible solution. BIT* almost-surely converges asymptotically to the global optimum but has difficulty sampling the valley in the available time, finding a path that goes around the mountain (6.17 kilometres). RABIT* uses the local optimizer on short paths to help find narrow passages and a global search to avoid infeasible local minima, finding a path through the valley (5.0 kilometres).

VI. Discussion & Future Work

RABIT* uses a local optimizer to find connections between states that cannot be connected by a straight line. This helps find paths in difficult-to-sample homotopy classes, such as through narrow passages, faster than an entirely global search. The simulated problems presented in Figs. 3–4 were designed to have such features. A column of obstacles with narrow gaps divides the start and goal, guaranteeing that the optimal solution belongs to a difficult-to-sample homotopy class. We found that in problems where global sampling is an effective method to find the optimal homotopy class (e.g., without these narrow passages), RABIT* performed similarly to BIT*.

For a local optimizer to be beneficial to the overall search, it must be applied judiciously. During development, we investigated hybrid searches based on other global methods and only found beneficial results with BIT*. This is because searching in order of decreasing solution quality limits the optimizer to edges that could improve the solution, helping to reduce the amount of time spent on optimization. This coincides with observations made by Xie et al. in personal communication regarding their use of a 2-pt BVP-solver with BIT* in [28]. The local optimizer can actually be considered a more general 2-pt BVP-solver that includes problem domain constraints (i.e., obstacles) in addition to dynamic constraints.

While we presented a version of RABIT* that uses CHOMP, any local optimizer or search can be used regardless of its cost function or optimality without affecting almost-sure asymptotic optimality. This is because RABIT* uses the *global* cost function to compare the straight-line and locally optimized paths. We are currently investigating replacing CHOMP with TrajOpt [36] as the local optimizer in RABIT*.

We had initially planned to use the local optimizer to also *rewire* the RABIT* tree. Using it to propose high-quality edges from the start to a vertex, generated by optimizing the existing path through the tree, appeared to be a promising method to increase the rate at which the solution converges towards the optimum. In our experiments, CHOMP proved to be an inefficient optimizer for collision-free, piecewise locally optimal paths. We are currently investigating whether another local optimizer would make this procedure beneficial.

VII. Conclusion

RABIT* integrates a local optimizer into a global search to create a hybrid technique with benefits from both methods. By using local optimization, it is able to *exploit* local information to quickly find paths in difficult-to-sample homotopy classes (e.g., narrow passages). By using an informed global search, it is able to efficiently *explore* the entire planning domain, considering all possibly better homotopy classes while maintaining almost-sure asymptotic convergence to the optimum.

We demonstrate the benefits of this technique on both simulated and real planning problems with narrow passages,

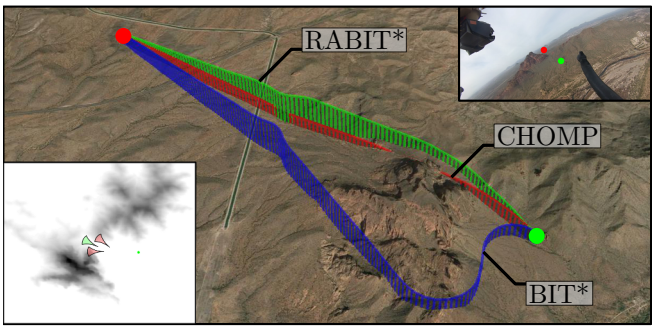


Fig. 5. The results of a planning problem encountered during flight tests of an autonomous helicopter in Mesa, Arizona from a start (green dot) to a goal located 4.91 kilometres away (red dot). An inset evidence map shows the terrain height above the helicopter, scaled such that the highest peak is black. Examples of inevitable collision states (red cones) and useful states (green cones) are shown on evidence map to illustrate the difficulty of navigating the through the valley. The three-dimensional map shows the results found in the 2 seconds of computation time available in flights conditions, with the view from the helicopter as inset. BIT* finds a flanking solution to the left of the mountains (blue, 6.17 kilometres), but is unable to navigate the valley in the available planning time. CHOMP finds an infeasible local optimum that strikes a mountain (red) from an initial condition connecting directly connecting the start and goal. RABIT* is able to use edges proposed by a local optimizer to find a path through the valley (green, 5.0 kilometres).

comparing it to existing asymptotically optimal planning algorithms and a local optimization technique. Randomly generated worlds show that on easy problems ($\mathbb{R}^2$), it is roughly equivalent to BIT*, while on harder problems ($\mathbb{R}^8$) it outperforms the other tested algorithms. Data from an autonomous helicopter demonstrate its suitability for time-constrained path planning in challenging environments for vehicles with dynamic constraints.

Acknowledgment

This research was funded 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. Sanjiban Choudhury and Sebastian Scherer acknowledge the support from ONR contract N0001412C0671 and ONR grant N000141310821.

References

[1] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *TRA*, 12(4): 566–580, 1996.
[2] S. M. LaValle and J. J. Kuffner Jr., "Randomized kinodynamic planning," *IJRR*, 20(5): 378–400, 2001.
[3] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *IJRR*, 21(3): 233–255, 2002.
[4] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *IJRR*, 30(7): 846–894, 2011.
[5] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *IROS* 2014, 2997–3004.
[6] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *IJRR*, 34(7): 883–921, 2015.
[7] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant hamiltonian optimization for motion planning," *IJRR*, 32(9–10): 1164–1193, 2013.
[8] S. Choudhury, S. Arora, and S. Scherer, "The planner ensemble: Motion planning by executing diverse algorithms," in *ICRA* 2015, 2389–2395.
[9] C. G. E. Boender, A. H. G. K. Rinnooy, G. T. Timmer, and L. Stougie, "A stochastic method for global optimization," *Mathematical Programming*, 22(1): 125–140, 1982.
[10] G. E. Hinton and S. J. Nowlan, "How learning can guide evolution," *Complex Systems*, 1(3): 495–502, 1987.
[11] C. G. Atkeson, "Using local trajectory optimizers to speed up global optimization in dynamic programming," in *NIPS* 1994, 663–670.
[12] D. Whitley, V. S. Gordon, and K. Mathias, "Lamarckian evolution, the baldwin effect and function optimization," in *PPSN III*, ser. Lecture Notes in Computer Science, 1994, vol. 866, pp. 5–15.
[13] R. Storn and K. Price, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces," *Jour. of Global Optimization*, 11(4): 341–359, 1997.
[14] W. Huyer and A. Neumaier, "Global optimization by multilevel coordinate search," *Jour. of Global Optimization*, 14(4): 331–355, 1999.
[15] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *ICRA* 2015, 3067–3074.
[16] C. Urmson and R. Simmons, "Approaches for heuristically biasing RRT growth," in *IROS* 2003, 1178–1183.
[17] D. Ferguson and A. Stentz, "Anytime RRTs," in *IROS* 2006, 5369–5375.
[18] L. Jaillet, J. Cortés, and T. Siméon, "Sampling-based path planning on configuration-space costmaps," *TRO*, 26(4): 635–646, 2010.
[19] M. Rickert, O. Brock, and A. Knoll, "Balancing exploration and exploitation in motion planning," in *IROS* 2008, 2812–2817.
[20] R. Alterovitz, S. Patil, and A. Derbakova, "Rapidly-exploring roadmaps: Weighing exploration vs. refinement in optimal motion planning," in *ICRA* 2011, 3706–3712.
[21] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions," in *IROS* 2011, 2640–2645.
[22] J. Nasir, F. Islam, U. Malik, Y. Ayaz, O. Hasan, M. Khan, and M. S. Muhammad, "RRT*-SMART: A rapid convergence implementation of RRT*," *Int. Jour. of Adv. Rob. Sys.*, 10: 299, 2013.
[23] O. Arslan and P. Tsiotras, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *ICRA* 2013, 2421–2428.
[24] ——, "Dynamic programming guided exploration for sampling-based motion planning algorithms," in *ICRA* 2015, 4819–4826.
[25] R. E. Bellman, "The theory of dynamic programming," *Bull. of the AMS*, 60(6): 503–516, 1954.
[26] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A*," *AI*, 155(1–2): 93–146, 2004.
[27] O. Salzman and D. Halperin, "Asymptotically-optimal motion planning using lower bounds on cost," in *ICRA* 2015, 4167–4172.
[28] C. Xie, J. van den Berg, S. Patil, and P. Abbeel, "Toward asymptotically optimal motion planning for kinodynamic systems using a two-point boundary value problem solver," in *ICRA* 2015, 4187–4194.
[29] S. Berchtold and B. Glavina, "A scalable optimizer for automatically generated manipulator motions," in *IROS* 1994, 1796–1802.
[30] S. Sekhavat, P. Švestka, J.-P. Laumond, and M. H. Overmars, "Multilevel path planning for nonholonomic robots using semiholonomic subsystems," *IJRR*, 17(8): 840–857, 1998.
[31] D. Hsu, J.-C. Latcombe, and S. Sorkin, "Placing a robot manipulator amid obstacles for optimized execution," in *ISATP* 1999, 280–285.
[32] R. Geraerts and M. H. Overmars, "Creating high-quality paths for motion planning," *IJRR*, 26(8): 845–863, 2007.
[33] K. Hauser and V. Ng-Thow-Hing, "Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts," in *ICRA* 2010, 2493–2498.
[34] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *ICRA* 2011, 4569–4574.
[35] S. Choudhury and S. Scherer, "The dynamics projection filter (DPF) - real-time nonlinear trajectory optimization using projection operators," in *ICRA* 2015, 644–649.
[36] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization," in *RSS* 2013.
[37] B. Raveh, A. Enosh, and D. Halperin, "A little more, a lot better: Improving path quality by a path-merging algorithm," *TRO*, 27(2): 365–371, 2011.
[38] R. Luna, I. A. Şucan Mark Moll, and L. E. Kavraki, "Anytime solution optimization for sampling-based motion planning," in *ICRA* 2013, 5068–5074.
[39] M. W. Otte and N. Correll, "C-FOREST: Parallel shortest-path planning with super linear speedup," *TRO*, 29(3): 798–806, 2013.
[40] J. Ichnowski and R. Alterovitz, "Parallel sampling-based motion planning with superlinear speedup," in *IROS* 2012, 1206–1212.
[41] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE R&A Mag.*, 19(4): 72–82, 2012.
[42] J. J. Kuffner Jr. and S. M. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *ICRA* 2000, 995–1001.