Asymptotically near-optimal RRT for fast, high-quality, motion planning

Oren Salzman* and Dan Halperin*

* Balvatnic School of Computer Science, Tel-Aviv University, Israel

Abstract—We present Lower Bound Tree-RRT (LBT-RRT), a single-query sampling-based algorithm that is asymptotically near-optimal. Namely, the solution extracted from LBT-RRT converges to a solution that is within an approximation factor of $1 + \varepsilon$ of the optimal solution. Our algorithm allows for a continuous interpolation between the fast RRT algorithm and the asymptotically optimal RRT* and RRG algorithms. When the approximation factor is 1 (i.e., no approximation is allowed), LBT-RRT behaves like the RRT* algorithm. When the approximation factor is unbounded, LBT-RRT behaves like the RRT algorithm. In between, LBT-RRT is shown to produce paths that have higher quality than RRT would produce and run faster than RRT* would run. This is done by maintaining a tree which is a sub-graph of the RRG roadmap and a second, auxiliary tree, which we call the lower-bound tree. The combination of the two trees, which is faster to maintain than the tree maintained by RRT*, efficiently guarantees asymptotic near-optimality. We suggest to use LBT-RRT for high-quality, anytime motion planning. We demonstrate the performance of the algorithm for scenarios ranging from 3 to 12 degrees of freedom and show that even for small approximation factors, the algorithm produces high-quality solutions (comparable to RRT*) with little runtime overhead when compared to RRT.

I. INTRODUCTION AND RELATED WORK

Motion planning is a fundamental research topic in robotics with applications in diverse domains such as graphical animation, surgical planning, computational biology, autonomous exploration, search-and-rescue, and warehouse management. Sampling-based planners such as PRM [1], RRT [2] and their many variants, introduced in the 1990s, enabled solving motion-planning problems that had been previously considered infeasible [3]. Following the exposition of sampling-based planners, the interest in the motion-planning community shifted from finding an arbitrary solution to the motion-planning problem to finding a high-quality solution. Quality can be measured in terms of, for example, length, clearance, smoothness, energy, to mention a few criteria, or some combination of the above.

A. High-quality planning with sampling-based algorithms

Unfortunately, common implementations of sampling-based planners such as RRT and PRM produce solutions that are sub-optimal or even far from optimal [4], [5]. Thus, many variants of these algorithms and heuristics were proposed in order to produce high-quality paths.

Post-processing existing paths: Post-processing a path produced by a motion-planning algorithm by applying short-cutting is a common, effective, approach to increase path quality [6]. Typically, two non-consecutive configurations are chosen randomly along the path. If the two configurations can be connected using a straight line and this connection improves the quality of the original path, the straight line replaces the original path that connected the two configurations. This process is continued iteratively until a termination condition holds.

Path hybridization: An inherent problem with path post-processing is that it is local in nature. A path that was post-processed using shortcutting often remains in the same homotopy class of the original one. Combining multiple different paths (that may be of low quality) enables the construction of high-quality paths, even when the number of paths combined is relatively low [7].

Online optimization: Changing the sampling strategy [8], [9], [10], [11], the connection scheme to a new milestone [10], [12] are examples of the heuristics proposed to create higher-quality solutions. Additional approaches include, among others, reachability [13] and random restarts [14].

Asymptotically optimal and near-optimal solutions: In a recent seminal work, Karaman and Frazzoli [4] give a rigorous analysis of the performance of the RRT and PRM algorithms. They show, that with probability one, the algorithms will not produce the optimal path. By modifying the connection scheme of a new sample to the existing data structure, they propose PRM* (a variant of the PRM algorithm) and RRG and RRT* (variants of the RRT algorithm) all of which are shown to be asymptotically optimal. In each algorithm, the number of nodes each new milestone is connected to increases proportionally to $\log(n)$ (where $n$ is the number of free samples).

As PRM* may produce prohibitively large graphs, recent work has focused on sparsifying these graphs. This can be done as a post-processing stage of the PRM* [15], [16], or as a modification of the PRM* [17], [18], [19], [20].

Recently, RRT* [21] was suggested as an alternative asymptotically optimal algorithm with a faster convergence rate when compared to RRT*. 
The performance of the RRT* was improved \cite{22} using several heuristics that bare resemblance to the lazy approach used in this work. Additional heuristics to speed up the convergence rate were presented in RRT*-SMART \cite{22}.

**Anytime and online solutions:** An interesting variant of the basic motion-planning problem is anytime motion-planning: In this problem, the time to plan is not known in advance, and the algorithm may be terminated at any stage. Clearly, any solution should be found as fast as possible and if time permits, it should be refined to yield a high-quality solution.

Ferguson and Stentz \cite{24} suggest iteratively running the RRT while considering only areas that may potentially improve the existing solution. Alterovitz et al. \cite{25} suggest the RRM algorithm that finds an initial path similar to the RRT algorithm. Once such a path is found, RRM uses a user-specified parameter to weigh whether to explore further the configuration space or to refine the explored space by adding edges to the current roadmap. Luna et al. \cite{26} suggest alternating between path shortcutting and path hybridization in an anytime fashion.

RRT* was recently adapted for a slightly different problem: online motion-planning \cite{27}. Here, an initial path is computed and the robot begins its execution. While the robot moves along this path, the algorithm attempts to refine the part that the robot has not moved along.

**B. Contribution**

We present LBT-RRT, a single-query sampling based algorithm that is asymptotically near-optimal. Namely, the solution extracted from LBT-RRT converges to a solution that is within a factor of \((1+\varepsilon)\) of the optimal solution. LBT-RRT allows for interpolating between the fast, yet sub-optimal, RRT algorithm and the asymptotically optimal RRT* algorithm. By choosing \(\varepsilon = 0\) no approximation is allowed and LBT-RRT maintains a tree identical to the tree maintained by RRT*. Choosing \(\varepsilon = \infty\) allows for any approximation and LBT-RRT maintains a tree identical to the tree maintained by RRT.

The asymptotic near-optimality of LBT-RRT is achieved by simultaneously maintaining two trees. Both trees are defined over the same set of vertices but each consists of a different set of edges. A path in the first tree may not be feasible, but its cost is always a lower bound on the cost of paths extracted from the RRT* (using the same sequence of random nodes). On the other hand, a path extracted from the second tree is always feasible and its cost is within a factor of \((1+\varepsilon)\) from the lower bound provided by the first tree.

We suggest to use LBT-RRT for high-quality motion planning under a limited time budget. We demonstrate the performance of the algorithm for scenarios ranging from 3 to 12 degrees of freedom (DoF) and show that even for small values of \(\varepsilon\), the algorithm produces high-quality solutions (comparable to RRT*) with little runtime overhead when compared to RRT.

**C. Outline**

We begin in Section [II] by reviewing the RRT, RRG and RRT* algorithms. In Section [III] we present our algorithm and its analysis and continue in Section [IV] to demonstrate its favorable characteristics on several scenarios. In Section [V] we list several applications where either RRT or RRT* were used and argue that LBT-RRT may serve as a superior alternative with no fundamental modification to the underlying algorithms using RRT or RRT*. Moreover, we discuss alternative implementations of LBT-RRT using tools developed for either RRT or RRT* that can enhance LBT-RRT. We conclude in Section [VI] by describing possible directions for future work regarding near-optimal motion planning.

**II. TERMINOLOGY AND ALGORITHMIC BACKGROUND**

We begin this section by formally stating the motion-planning problem and introducing several standard procedures used by sampling-based algorithms and continue by reviewing the RRT, RRG and RRT* algorithms.

**A. Problem definition and terminology**

We follow the formulation of the motion-planning problem as presented by Karaman and Frazzoli \cite{4}. Let \(X\) denote the configuration space (C-space), \(X_{\text{free}}\) and \(X_{\text{forb}}\) denote the free and forbidden spaces, respectively. We define the motion planning problem as \((X_{\text{free}}, x_{\text{init}}, x_{\text{goal}})\) where: \(x_{\text{init}} \in X_{\text{free}}\) is an initial free configuration and \(x_{\text{goal}} \subseteq X_{\text{free}}\) is the goal region. A path \(\sigma : [0, 1] \rightarrow X\) is a continuous mapping to the C-space. It is collision-free if \(\forall \tau \in [0, 1], \sigma(\tau) \in X_{\text{free}}\) and feasible if it is collision-free, \(\sigma(0) = x_{\text{init}}\) and \(\sigma(1) \in X_{\text{goal}}\).

We will make use of the following procedures throughout this paper: \text{sample\_free}, a procedure returning a free configuration at random, \text{nearest\_neighbor}(x, V) and \text{nearest\_neighbors}(x, V, k) are procedures returning the nearest neighbor and \(k\) nearest neighbors of \(x\) within the set \(V\), respectively. Let \text{steer}(x, y) return a configuration \(z\) that is closer to \(y\) than \(x\) is. \text{collision\_free}(x, y) tests if the straight line segment connecting \(x\) and \(y\) is contained in \(X_{\text{free}}\) and \text{cost}(x, y) be a procedure returning the cost of the straight-line path connecting \(x\) and \(y\). Let us denote by \text{cost}_{G}(x) the minimal cost of reaching a node \(x\) from \(x_{\text{init}}\) using a roadmap \(G\). These are standard procedures used by the RRT and RRT* algorithms. Finally, we use the (generic) predicate \text{build\_tree} to assess if a stopping criteria has been reached to terminate the algorithm. This can be, for example, reaching a certain number of samples, exceeding a fixed time budget or any other natural stopping criteria.

**B. Algorithmic background**

The RRT, RRG and RRT* algorithms share the same high-level structure. All three algorithms maintain a roadmap as the underlying data structure. The roadmap is a tree in the case of RRT and RRT* and a graph in the case of RRG. At each iteration a configuration \(x_{\text{rand}}\) is sampled at random, then, the nearest configuration in the roadmap \(x_{\text{nearest}}\) is found and extended in the direction of \(x_{\text{rand}}\) to a new configuration \(x_{\text{new}}\).
If the path between $x_{\text{nearest}}$ and $x_{\text{new}}$ is collision-free, then $x_{\text{new}}$ is added to the roadmap (see Algorithms 1, 2, 3 lines 3-9).

The algorithms differ with respect to the connections added to the roadmap. In the RRT algorithm, only the edge $(x_{\text{nearest}}, x_{\text{new}})$ is added. In the RRG and RRT* algorithms, a set $X_{\text{near}}$ of $k_{\text{RRG}}\log(|V|)$ nearest neighbors of $x_{\text{new}}$ is considered. Here, $k_{\text{RRG}}$ is a constant ensuring that the cost of paths produced by the RRG and RRT* algorithms indeed converge to the optimal cost as the number of samples grows. Choosing $k_{\text{RRG}} = 2e$ is a valid choice for all problem instances $\mathcal{H}$. For each neighbor $x_{\text{near}} \in X_{\text{near}}$ of $x_{\text{new}}$, the RRG algorithm checks if the path between $x_{\text{near}}$ and $x_{\text{new}}$ is collision-free and if so, $(x_{\text{near}}, x_{\text{new}})$ and $(x_{\text{new}}, x_{\text{near}})$ are added as additional edges to the roadmap (Algorithm 2 lines 10-13). The RRT* maintains a sub-graph of the RRG roadmap containing the best path found to each node. This is done by an additional rewiring procedure (Algorithm 4) which is invoked twice: The first time, it is used to find the node $x_{\text{new}} \in X_{\text{near}}$ which will minimize the cost to reach $x_{\text{new}}$ (Algorithm 3 lines 11-12). The second time, the procedure is used to attempt to minimize the cost to reach every node $x_{\text{near}} \in X_{\text{near}}$ by considering $x_{\text{new}}$ as its parent (Algorithm 4 lines 13-14).

### Algorithm 1 RRT ($x_{\text{init}}$)

1. $T.V \leftarrow \{x_{\text{init}}\}$
2. while build_tree() do
3.     $x_{\text{rand}} \leftarrow \text{sample_free}()$
4.     $x_{\text{nearest}} \leftarrow \text{nearest_neighbor}(x_{\text{rand}}, T.V)$
5.     $x_{\text{new}} \leftarrow \text{steer}(x_{\text{nearest}}, x_{\text{rand}})$
6.     if (!collision_free($x_{\text{nearest}}, x_{\text{new}}$)) then
7.         CONTINUE
8.     $T.V \leftarrow T.V \cup \{x_{\text{new}}\}$
9.     $T.parent(x_{\text{new}}) \leftarrow x_{\text{nearest}}$

### Algorithm 2 RRG ($x_{\text{init}}$)

1. $G.V \leftarrow \{x_{\text{init}}\}$, $G.E \leftarrow \emptyset$
2. while build_tree() do
3.     $x_{\text{rand}} \leftarrow \text{sample_free}()$
4.     $x_{\text{nearest}} \leftarrow \text{nearest_neighbor}(x_{\text{rand}}, T.V)$
5.     $x_{\text{new}} \leftarrow \text{steer}(x_{\text{nearest}}, x_{\text{rand}})$
6.     if (!collision_free($x_{\text{nearest}}, x_{\text{new}}$)) then
7.         CONTINUE
8.     $G.V \leftarrow G.V \cup \{x_{\text{new}}\}$
9.     $G.E \leftarrow \{(x_{\text{nearest}}, x_{\text{new}}), (x_{\text{new}}, x_{\text{nearest}})\}$
10. $X_{\text{near}} \leftarrow \text{nearest_neighbors}(x_{\text{new}}, G.V, k_{\text{RRG}}\log(|G.V|))$
11. for all $(x_{\text{near}}, X_{\text{ordered}})$ do
12.     if (collision_free($x_{\text{near}}, x_{\text{new}}$)) then
13.         $G.E \leftarrow \{(x_{\text{near}}, x_{\text{new}}), (x_{\text{new}}, x_{\text{near}})\}$

### Algorithm 3 RRT* ($x_{\text{init}}$)

1. $T.V \leftarrow \{x_{\text{init}}\}$
2. while build_tree() do
3.     $x_{\text{rand}} \leftarrow \text{sample_free}()$
4.     $x_{\text{nearest}} \leftarrow \text{nearest_neighbor}(x_{\text{rand}}, T.V)$
5.     $x_{\text{new}} \leftarrow \text{steer}(x_{\text{nearest}}, x_{\text{rand}})$
6.     if (!collision_free($x_{\text{nearest}}, x_{\text{new}}$)) then
7.         CONTINUE
8.     $T.V \leftarrow T.V \cup \{x_{\text{new}}\}$
9.     $T.parent(x_{\text{new}}) \leftarrow x_{\text{nearest}}$
10. $X_{\text{near}} \leftarrow \text{nearest_neighbors}(x_{\text{new}}, T.V, k_{\text{RRG}}\log(|T.V|))$
11. for all $(x_{\text{near}}, X_{\text{near}})$ do
12.     if (collision_free($x_{\text{near}}, x_{\text{new}}$)) then
13.     for all $(x_{\text{near}}, X_{\text{near}})$ do
14.         $\text{rewire_RRT}^*(x_{\text{near}}, x_{\text{new}})$

### Algorithm 4 rewire_RRT*($x_{\text{potential_parent}}, x_{\text{child}}$)

1. if (collision_free($x_{\text{potential_parent}}, x_{\text{child}}$)) then
2.     $c \leftarrow \text{cost($x_{\text{potential_parent}}, x_{\text{child}}$)}$
3.     if ($\text{cost}_T(x_{\text{potential_parent}}) + c < \text{cost}_T(x_{\text{child}})$) then
4.         $T.parent(x_{\text{child}}) \leftarrow x_{\text{potential_parent}}$

### III. ASYMPOTICALLY NEAR-OPTIMAL MOTION-PLANNING

Clearly the asymptotic optimality of the RRT* and RRG algorithms come at the cost of the additional $k_{\text{RRG}}\log(|V|) - 1$ calls to the local planner at each stage (and some additional overhead). If we are not concerned with asymptotically optimal solutions, we do not have to consider all of the $k_{\text{RRG}}\log(|V|)$ neighbors when a node is added. Our idea is to initially only assess the quality of each edge. We use the quality of the edge to decide if to discard it, use it without checking if it is collision-free or use it after validating that it is indeed collision-free. Thus, many calls to the local planner can be avoided, though we still need to estimate the quality of many edges. Our approach is viable in cases where such an assessment can be carried out efficiently, namely more efficient than deciding if an edge is collision-free. This condition holds naturally when the quality measure is path length.

#### A. Notation

Let $\Sigma$ be the set of all paths, $\Sigma_f \subset \Sigma$ the set of all feasible paths and $c : \Sigma \rightarrow \mathbb{R}^+$ a cost function. We denote by $c^*$ the optimal cost, namely $c^* = \min_{\sigma \in \Sigma_f}\{c(\sigma)\}$. In this paper we only consider path length as the cost function.
Let $\sigma_{\text{ALG}}(n)$ be the path produced by a sampling-based algorithm ALG using $n$ free samples. A sampling-based algorithm ALG is asymptotically optimal if $\lim_{n \to \infty} c(\sigma_{\text{ALG}}(n)) = c^*$ with probability one and is asymptotically $(1 + \varepsilon)$-optimal if $\lim_{n \to \infty} c(\sigma_{\text{ALG}}(n)) \leq (1 + \varepsilon) \cdot c^*$ with probability one. We will refer to an asymptotically $(1 + \varepsilon)$-optimal algorithm as asymptotically near-optimal and refer to $(1 + \varepsilon)$ as the approximation factor.

We introduce an additional procedure used by our algorithm, $\text{order}_G(X, x)$, which returns the elements of $X$ ordered according to the cost to reach $x$ from $x_{\text{init}}$ through an element $x' \in X$. Namely, after ordering, $x_1 \in X$ is before $x_2 \in X$ if $\text{cost}_G(x_1) + \text{cost}(x_1, x) \leq \text{cost}_G(x_2) + \text{cost}(x_2, x)$. The importance of this ordering will be discussed in the next subsection.

### B. LBT-RRT

We propose a simple modification to the RRT* algorithm allowing a continuous interpolation between the fast RRT algorithm and the asymptotically optimal RRT* algorithm. We maintain two roadmaps $T_{\text{lb}}, T_{\text{apx}}$ simultaneously, both consist of the same set of vertices but differ in their edge set.

Let us consider the roadmap $G_{\text{RRG}}$ constructed by the RRG algorithm if run on the same sequence of samples used for LBT-RRT. Now, $T_{\text{lb}}, T_{\text{apx}}$ maintain the following invariants throughout the algorithm:

**Lower bound invariant** - For every node $x \in T_{\text{apx}}, T_{\text{lb}}$, $\text{cost}_{T_{\text{lb}}}(x) \leq \text{cost}_{G_{\text{RRG}}}(x)$.

This is ensured by maintaining $T_{\text{lb}}$ as a subgraph of $G_{\text{RRG}}$, with possibly some additional edges that are not collision-free, containing only the best route to each node. Notice that the cost of an edge, regardless of whether it is collision-free or not, is the length of the straight line connecting its end configurations.

**Bounded approximation invariant** - For every node $x \in T_{\text{apx}}, T_{\text{lb}}$, $\text{cost}_{T_{\text{apx}}}(x) \leq (1 + \varepsilon) \cdot \text{cost}_{T_{\text{lb}}}(x)$.

The main body of the algorithm (see Algorithm 5) follows the structure of the RRT, RRT* and RRG algorithms with respect to adding a new milestone (lines 3-7) but differs in the connections added. If a path between the new node $x_{\text{apx}}$ and its nearest neighbor $x_{\text{apx nearest}}$ is indeed collision-free, it is added to both trees with an edge from $x_{\text{apx nearest}}$ to $x_{\text{apx}}$ (lines 8-10). This is demonstrated in Figure 1.

Similar to RRT*, LBT-RRT locates the set $X_{\text{near}}$ of $k_{\text{RRG}} \log(|V|)$ nearest neighbors of $x_{\text{new}}$ (line 11). Then, $^1$The subscript of $T_{\text{lb}}$ is an abbreviation for lower bound and the subscript of $T_{\text{apx}}$ is an abbreviation for approximation.

---

**Fig. 1.** Adding a new configuration $x_{\text{new}}$ to the roadmaps. Obstacles are depicted in pink, $T_{\text{apx}}$ is depicted by solid black arrows and $T_{\text{lb}}$ is depicted be dotted blue arrows. The new edge added is depicted by a dotted green arrow.
Algorithm 5 LBT-RRT ($x_{\text{init}}, \varepsilon$)

1: $T_{\text{lb}}, V \leftarrow \{x_{\text{init}}\}$
2: while build_tree() do
3:     $x_{\text{rand}} \leftarrow \text{sample_free}()$
4:     $x_{\text{nearest}} \leftarrow \text{nearest_neighbor}(x_{\text{rand}}, T_{\text{lb}}, V)$
5:     $x_{\text{new}} \leftarrow \text{steer}(x_{\text{nearest}}, x_{\text{rand}})$
6:     if $\text{(collision_free}(x_{\text{new}}))$ then
7:         CONTINUE
8:     $T_{\text{lb}}, V \leftarrow T_{\text{lb}}, V \cup \{x_{\text{new}}\}$
9:     $T_{\text{lb}}, \text{parent}(x_{\text{new}}) \leftarrow x_{\text{nearest}}$
10:    $T_{\text{apx}}, \text{parent}(x_{\text{new}}) \leftarrow x_{\text{nearest}}$
11:    $X_{\text{near}} \leftarrow \text{nearest_neighbors}(x_{\text{new}}, T_{\text{lb}}, V, k_{\text{RRG}} \log(|T_{\text{lb}}, V|))$
12:    $X_{\text{ordered}} \leftarrow \text{order}_{T_{\text{lb}}}(X_{\text{near}}, x_{\text{new}})$
13:    for all $(x_{\text{near}}, X_{\text{ordered}})$ do
14:        rewire_LBT-RRT($x_{\text{near}}, x_{\text{new}}$)
15:    for all $(x_{\text{near}}, X_{\text{near}})$ do
16:        rewire_LBT-RRT($x_{\text{new}}, x_{\text{near}}$)

Algorithm 6 rewire_LBT-RRT($x_{\text{potential parent}}, x_{\text{child}}$)

1: $c \leftarrow \text{cost}(x_{\text{potential parent}}, x_{\text{child}})$
2: $\text{potential} \_\text{cost}_{lb} \leftarrow \text{cost}_{lb}(x_{\text{potential parent}}) + c$
3: if $(\text{cost}_{lb}(x_{\text{child}})) \leq \text{potential} \_\text{cost}_{lb}$ then
4:    return
5: if $(\text{cost}_{apx}(x_{\text{child}})) > (1 + \varepsilon) \cdot \text{potential} \_\text{cost}_{lb}$ then
6:    if $(\text{collision_free}(x_{\text{potential parent}}, x_{\text{child}}))$ then
7:        $T_{\text{lb}}, \text{parent}(x_{\text{child}}) \leftarrow x_{\text{potential parent}}$
8:        $T_{\text{apx}}, \text{parent}(x_{\text{child}}) \leftarrow x_{\text{potential parent}}$
9:    else
10:       $T_{\text{lb}}, \text{parent}(x_{\text{child}}) \leftarrow x_{\text{potential parent}}$

We note the following (straightforward, yet helpful) observations comparing LBT-RRT, RRG and RRT* when run on the same set of random samples:

Observation III.2. A node is added to $T_{lb}$ and to $T_{apx}$ if and only if a node is added to $G_{RRG}$ (see Algorithm 2 and Algorithm 5, lines 3-8).

Observation III.3. Both LBT-RRT and RRG consider the same set of $k_{RRG} \log(|V|)$ nearest neighbors of $x_{\text{new}}$ (see Algorithm 2, line 10 and Algorithm 5, line 11).

Observation III.4. $T_{lb}$ undergoes the same rewiring process as RRT* (see Algorithm 6, lines 7, 10) with possibly some additional edges that are not collision-free (see Algorithm 6, line 10).

Observation III.5. An edge is added to $T_{apx}$ only if it is collision free (see Algorithms 5, line 10 and Algorithm 6, line 8).

Thus, the following corollary trivially holds:

Corollary III.6. For each node $x \in T_{lb}$, $\text{cost}_{lb}(x) \leq \text{cost}_{RRG}(x)$

Using Observations III.2 through III.5 and Corollary III.6

Lemma III.7. After every iteration of Algorithm 5 (lines 3-16) the bounded approximation invariant is maintained.

Proof: The edges of $T_{lb}, T_{apx}$ are updated in one of the subsequent nodes, Algorithm 6 will not contain any (computationally demanding) calls to the collision detector (lines 3-4). We note that this ordering can also be done to speed up the RRT* algorithm.

C. Analysis

In this section we show that Algorithm 5 maintains the lower bound invariant (Corollary III.6) and that after every iteration of the algorithm the bounded approximation invariant is maintained (Lemma III.7). A trivial corollary (using the asymptotic optimality of RRG) is:

Corollary III.1. LBT-RRT is asymptotically near-optimal with an approximation factor of $(1 + \varepsilon)$.

Namely the cost of the path computed by LBT-RRT converges to a cost lower than $(1 + \varepsilon)$ times the cost of the optimal path almost surely.
following cases:

**case (a):** When adding a new milestone \( x_{new} \) to the trees, it is initially connected in both trees to the same milestone \( x_{nearest} \) (see lines 8-10, Algorithm 5). Assume that the invariant was maintained prior to this step, namely:

\[
\text{cost}_{\text{apx}}(x_{nearest}) \leq (1 + \epsilon) \cdot \text{cost}_{lb}(x_{nearest}).
\]

Using the invariant,

\[
\text{cost}_{\text{apx}}(x_{new}) = \text{cost}_{\text{apx}}(x_{nearest}) + \text{cost}(x_{nearest}, x_{new}) \\
\leq (1 + \epsilon) \cdot \text{cost}_{lb}(x_{nearest}) + \text{cost}(x_{nearest}, x_{new}) \\
\leq (1 + \epsilon) \cdot \text{cost}_{lb}(x_{new}).
\]

**case (b):** Additional rewiring may occur in Algorithm 6 line 5. In this case both trees update the incoming edge of \( x_{child} \) to be \( x_{potential\_parent} \). Assuming that the invariant was maintained prior to this step then it is maintained after this update (see case (a), only that now we use \( x_{potential\_parent} \) instead of \( x_{nearest} \)).

**case (c):** Finally rewiring may also occur in Algorithm 6 line 10. This occurs when \( \text{cost}_{\text{apx}}(x_{child}) \leq (1 + \epsilon) \cdot \text{potential\_cost}_{lb} \). In this case only \( \text{lb} \) is updated and:

\[
\text{cost}_{\text{apx}}(x_{child}) \leq (1 + \epsilon) \cdot \text{potential\_cost}_{lb} = (1 + \epsilon) \cdot \text{cost}_{lb}(x_{child}).
\]

**D. Discussion**

Let \( T^\omega_\text{ALG} \) denote the time needed for an algorithm \( \text{ALG} \) to find a feasible solution on a set of random samples \( \omega \). Clearly, \( T^\omega_{RRT} \leq T^\omega_{RRT^*} \) (as the RRT* may require more calls to the collision detector than the RRT algorithm). Moreover for every \( \epsilon_1 \leq \epsilon_2 \) it holds that

\[
T^\omega_{RRT} \leq T^\omega_{LBT-RRT}(\epsilon_2) \leq T^\omega_{LBT-RRT}(\epsilon_1) \leq T^\omega_{RRT^*}.
\]

Thus, given a limited amount of time, the RRT* algorithm may fail to construct any solution. On the other hand, the RRT algorithm may find a solution fast but will not improve its quality (if the goal is a single configuration). The LBT-RRT allows to find a feasible path quickly while improving its quality.

**IV. Evaluation**

We present an experimental evaluation of the performance of LBT-RRT as an anytime algorithm on different scenarios consisting of 3, 6, and 12 DoFs (Figure 5). All experiments were run using the Open Motion Planning Library (OMPL 0.10.2) [28] on a 3.4GHz Intel Core i7 processor with 8GB of memory. The RRT* was implemented by using \( \epsilon = 0 \). This implementation outperforms a naive implementation of RRT* due to the use of the \textit{order} function (see Algorithm 5 line 12 and [22]).

The Maze scenario (Figure 3a) consists of a planar polygonal robot that can translate and rotate. The Alternating barriers scenario (Figure 3b) consists of a robot with three perpendicular rods free-flying in space. The robot needs to pass through a series of barriers each containing a large and a small hole. For an illustration of one such barrier, see Figure 4. The large holes are located at alternating sides of consecutive barriers. Thus, an easy path to find would be to cross each barrier through a large hole. A high-quality path would require passing through a small hole after each large hole. Finally, the cubicles scenario consists of two L-shaped robots free-flying in space that need to exchange locations amidst a sparse collection of obstacles.

We compare the performance of LBT-RRT with RRT and RRT* when a fixed time budget is given. We consider \( (1 + \epsilon) \) values of 1.2, 1.4, 1.8 and report on the success rate of each algorithm (Figure 5). Additionally, we report on the path length after applying shortcuts (Figure 6). All results are averaged over 100 different runs.

Figure 5 depicts similar behavior for all scenarios: As one would expect, the success rate for all algorithms has a monotonically increasing trend as the time budget increases. For a specific time budget, the success rate for the RRT algorithm is typically highest while that of the RRT* is lowest. The success rate for LBT-RRT for a specific time budget, typically increases as the value of \( \epsilon \) increases. Figure 6 also depicts similar behavior for all scenarios: the average path length decreases for all algorithms (except for the RRT algorithm). The RRT exhibits the highest average path length. The average path length for LBT-RRT typically decreases as the value of \( \epsilon \) decreases and is comparable to that of RRT* for low values of \( \epsilon \).

Thus, Figures 5, 6 should be looked at simultaneously as they encompass the tradeoff between speed to find \textit{any} solution and the quality of the solution found. Let us demonstrate this on the alternating barriers scenario: If we look at the success rate of each algorithm to find \textit{any} solution (Figure 5b), one can see that the RRT algorithm manages to achieve a success rate of 70% after 30 seconds. The RRT* on the other hand requires 70 seconds to achieve the same success rate (more than double the time). For all different values of \( \epsilon \), LBT-RRT manages to achieve a success rate of 70% after 50 seconds (around 50% overhead when compared to RRT). Now, considering the path length...
length at 50 seconds, typically the paths extracted from LBT-RRT yield the same quality when compared to RRT* while ensuring a high success rate.

The same behavior has been observed for both the Maze scenario and the Cubicles scenario. Results omitted in this text. For supplementary material the reader is referred to [http://acg.cs.tau.ac.il/projects/LBT-RRT/project-page](http://acg.cs.tau.ac.il/projects/LBT-RRT/project-page).

V. ADDITIONAL APPLICATIONS & VARIANTS

RRT has been used in numerous applications and various efficient implementations and heuristics have been suggested for it. Even the relatively recent RRT* has already gained many applications and various implementations. Typically, the applications rely on the efficiency of RRT or the asymptotic optimality of RRT*. We list several such applications and discuss the possible advantage of replacing the existing planner (either RRT or RRT*) with LBT-RRT.

Efficient implementations and heuristics typically take into account the primitive operations used by the RRT and the RRT* algorithms (such as collision detection, nearest neighbor computation, sampling procedure etc.). Thus, techniques suggested for efficient implementations of RRT and RRT* may be applied to LBT-RRT with little effort as the latter relies on the same primitive operations.

A. Re-planning using RRTs:

Many real-world applications involve a C-space that undergoes changes (such as moving obstacles or partial initial information of the workspace). A common approach to plan in such dynamic environments is to run the RRT algorithm, and re-plan when a change in the environment occurs. Re-planning may be done from scratch although this can be unnecessary and time consuming as the assumption is that only part of the environment changes. Ferguson et al. [24] suggest to (i) plan an initial path using the RRT algorithm, (ii) when a change
in the configuration space is detected, nodes in the existing tree may be invalidated and a “trimming” procedure is applied where invalid parts are removed and (iii) the trimmed tree is grown until a new solution is generated.

Obviously LBT-RRT can replace the RRT in the suggested scheme. If the overhead of running LBT-RRT when compared to RRT is acceptable (which may indeed be the case as the experimental results suggest), then the algorithm will be able to produce high-quality paths in dynamic environments.

B. High-quality planning on implicitly-defined manifolds:

Certain motion-planning problems, such as grasping with a multi-fingered hand, involve planning on implicitly-defined manifolds. Jaillet and Porta [29] address the central challenges of applying the RRT* algorithm to such cases. The challenges include sampling uniformly on the manifold, locating the nearest neighbors using the metric induced by the manifold, computing the shortest path between two points and more. They suggest AtlasRRT*, an adaptation of RRT* that operates on manifolds. It follows the same structure as the RRT* but maintains an atlas by iteratively adding charts to the atlas to facilitate the primitive operations of RRT* on the manifold (i.e., sampling, nearest neighbor queries, local planning etc.).

If one is concerned with fast convergence to a high quality solution, LBT-RRT can be used seamlessly, replacing the guarantee for optimality with a weaker near-optimality guarantee.

C. Sampling Heuristics:

Following the exposition of the RRT* algorithm, Akgun and Stilman [30] suggested a sampling bias for the RRT* algorithm. This bias accelerates cost decrease of the path to the goal in the RRT* tree. Additionally, they suggest a simple node-rejection criterion to increase efficiency. These heuristics may be applied to the LBT-RRT by simply changing the procedure sample_free (Algorithm 5, line 3).

D. Parallel RRTs:

In recent years, hardware allowing for parallel implementation of existing algorithms has become widespread both in the form of multi-core Central Processing Units (CPUs) and in the form of Graphics Processing Units (GPUs). Parallel implementations for sampling based algorithms have already been proposed in the late 1990s [31]. Since then, a multitude of such implementations emerged (see, e.g., [32], [33] for a detailed literature review).

We review two approaches for parallel implementation of RRT and RRT* and claim that both approaches may be used for parallel implementation of LBT-RRT. The first approach, by Ichnowski et al. [32] suggests parallel variants of RRT and RRT* on multi-core CPUs that achieve superlinear speedup. By using CPU-based implementation, their approach retains the ability to integrate the planners with existing CPU-based libraries and algorithms. They achieve superlinear speedup by: (i) lock-free parallelism using atomic operations to reduce slowdowns caused by contention, (ii) partition-based sampling to reduce the size of each processor core’s working data set and to improve cache efficiency and (iii) parallel backtracking in the rewiring phase of RRT*. As previously mentioned, LBT-RRT may benefit from all three key algorithmic features.

Bialkowski et al. [33] present a second approach for parallel implementation of RRT and RRT*. They suggest a massively parallel, GPU-based implementation of the collision-checking procedures of RRT and RRT*. Again, this approach may be applied to the collision-checking procedure of LBT-RRT without any need for modification.

E. Pruning via Branch and Bound:

Let $\text{cost}_T(x)$ be a lower bound on the cost to reach the target from $x \in \mathcal{X}_{free}$. A possible bound may be achieved by computing the Euclidean distance between $x$ and the target configuration (ignoring obstacles on the way).

Given a tree $T$ denote by $c_T^*$ the lowest-cost path in $T$ from the root to the goal region (if no such path exists then $c_T^* = \infty$). Now, recall that $\text{cost}_T(x)$ denotes the cost of the unique path that starts at the root and reaches $x$ through the edges of $T$. Thus, for every node $x$ in the tree, such that $\text{cost}_T(x) + \text{cost}_T(x) \geq c_T^*$, we define $x$ to be unuseful. One can keep track of all such nodes that are un-useful and periodically delete them from the tree.

We note that this is a technique frequently employed in AI that has been used for sampling-based planners (see, e.g. [24], [27]).

VI. Future work

We seek to suggest natural stopping criteria for LBT-RRT. Such criteria could possibly be related to the rate at which the quality is increased as additional samples are introduced. Once such a criterion is established, one can think of the following algorithmic framework: Run LBT-RRT with a large approximation factor (large $\varepsilon$), once the stopping criterion has been met, decrease the approximation factor and continue running. This framework may allow an even quicker convergence to find any feasible path while allowing for refinement as time permits (similar to [25]).

An interesting question to be further studied is can our framework be applied to different quality measures. For certain quality measures, such as bottleneck clearance of a path, this is unlikely (as bounding the quality of an edge already identifies if it is collision-free). However, for measures such as energy consumption, the framework may be used.

Additionally, we wish to explore the behavior of our algorithm given a fixed, limited time budget. This analysis should take into account the different paths or homotopy classes and the probability that the algorithm will find each one using a given fixed number of samples.

REFERENCES

[1] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high dimensional configuration spaces,” IEEE Transactions on Robotics, vol. 12, no. 4, pp. 566–580, 1996.

[2] J. J. Kuffner and S. M. LaValle, “RRT-Connect: An efficient approach to single-query path planning,” in ICRA, 2000, pp. 995–1001.
[3] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation*. MIT Press, June 2005.

[4] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *I. J. Robotic Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[5] O. Nechushtan, B. Raveh, and D. Halperin, “Sampling-diagram automata: A tool for analyzing path quality in tree planners,” in *WAFF*, 2010, pp. 285–301.

[6] R. Geraerts and M. H. Overmars, “Creating high-quality paths for motion planning,” *I. J. Robotic Res.*, vol. 26, no. 8, pp. 845–863, 2007.

[7] B. Raveh, A. Enosh, and D. Halperin, “A little more, a lot better: Improving path quality by a path-merging algorithm,” *IEEE Transactions on Robotics*, vol. 27, no. 2, pp. 365–371, 2011.

[8] R. Alterovitz, S. Patil, and A. Derbakova, “Rapidly-exploring roadmaps for optimal motion planning using incremental sampling-based algorithms,” in *ICRA*, 2011, pp. 4307–4313.

[9] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan, “RRT*-Smart: Rapid convergence implementation of RRT* towards optimal solution,” in *International Conference on Mechatronics and Automation (ICMA)*, 2012.

[10] D. Ferguson and A. T. Stentz, “Anytime RRTs,” in *IROS*, 2006, pp. 5369 – 5375.

[11] R. Alterovitz, S. Patil, and A. Derbakova, “Rapidly-exploring roadmaps: Weighing exploration vs. refinement in optimal motion planning,” in *ICRA*, 2013, pp. 2413–2420.

[12] R. Luna, I. A. Șucan, M. Moll, and L. E. Kavraki, “Anytime solution optimization for sampling-based motion planning,” in *ICRA*, 2013, pp. 5053–5059.

[13] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. J. Teller, “Anytime motion planning using the RRT,” in *ICRA*, 2011, pp. 1478–1483.

[14] I. A. Șucan, M. Moll, and L. E. Kavraki, “The Open Motion Planning Library,” *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[15] L. Jaillet and J. M. Porta, “Asymptotically-optimal path planning on manifolds,” in *RSS*, 2012.

[16] B. Akgun and M. Stilman, “Sampling heuristics for optimal motion planning in high dimensions,” in *IROS*, 2011, pp. 2640–2645.