2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

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

Oren Salzman[*] and Dan Halperin[*]
[*] Blavatnik 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 guarantee 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 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 enabled solving motion-planning problems that had been previously considered infeasible [3]. Gradually, 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 planners such as RRT and PRM produce solutions that may be far from optimal [4], [5]. Thus, many variants of these algorithms and heuristics were proposed in order to produce high-quality paths.

This work has been supported in part by the 7th Framework Programme for Research of the European Commission, under FET-Open grant number 255827 (CGL—Computational Geometry Learning), by the Israel Science Foundation (grant no. 1102/11), by the German-Israeli Foundation (grant no. 1150-82.6/2011), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

Post-processing existing paths: Post-processing an existing path by applying *shortcutting* 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. The process is continued iteratively until a termination condition holds.

Path hybridization: A path that was post-processed using shortcutting often remains in the homotopy class of the original path. Combining even a small number of different paths (that may be of low quality) enables the construction of higher-quality paths [7].

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

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 and propose the PRM* and the RRG and RRT* algorithms (variants of the PRM and RRT algorithms, respectively) all of which are shown to be *asymptotically optimal*. In each algorithm, the number of nodes each new sample is connected to is proportional to $\log(n)$ (here 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* [14], [15], or as a modification of PRM* (see, e.g., [16]).

The performance of RRT* was improved by suggesting different heuristics to speed up the convergence rate [17], [18]. Recently, RRT$^{\#}$ [19] was suggested as an alternative asymptotically optimal algorithm with a faster convergence rate when compared to RRT*.

Anytime and online solutions: In anytime motion-planning, 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 higher-quality solution.

Ferguson and Stentz [20] suggest iteratively running RRT while considering only areas that may potentially improve the existing solution. Alterovitz et al. [21] suggest the RRM algorithm, which finds an initial path similar to the RRT algorithm. Once such a path is found, RRM either explores further the configuration space or refines the explored space. Luna et al. [22] suggest alternating between path shortcutting and path hybridization in an anytime fashion.

RRT* was recently adapted for online motion-planning [23]. 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, anytime motion planning. 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 ε, the algorithm produces high-quality solutions (comparable to RRT*) with little runtime overhead when compared to RRT.

C. Outline

In Section II we review the RRT, RRG and RRT* algorithms. In Section III we present our algorithm and our an asymptotical near-optimality proof. We continue in Section IV to demonstrate its favorable characteristics on several scenarios and conclude our paper in Section V.

II. Terminology and algorithmic background

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

A. Problem definition and terminology

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

We will make use of the following procedures: `sample_free`, a procedure returning a random free configuration, `nearest_neighbor`(x, V) and `nearest_neighbors`(x, V, k) are procedures returning the nearest neighbor and k nearest neighbors of x within the set V, respectively. Let `steer`(x, y) return a configuration z that is closer to y than x is, `collision_free`(x, y) tests if the straight line segment connecting x and y is contained in $\mathcal{X}_{free}$ and `cost`(x, y) be a procedure returning the cost of the straight-line path connecting x and y. Let us denote by `cost`$_{\mathcal{G}}(x)$ the minimal cost of reaching a node x from x_{init} using a roadmap $\mathcal{G}$. These are standard procedures used by the RRT or RRT* algorithms. Finally, we use the (generic) predicate `build_tree` to assess if a stopping criterion has been reached to terminate the algorithm[1].

B. Algorithmic background

The RRT, RRG and RRT* algorithms share the same high-level structure. They maintain a roadmap which is a tree for RRT and RRT* and a graph for RRG. At each iteration a configuration x_{rand} is sampled at random. Then, the nearest configuration to x_{rand} in the roadmap $x_{nearest}$ is found and extended in the direction of x_{rand} to a new configuration x_{new}. If the path between $x_{nearest}$ and x_{new} is collision-free, then x_{new} is added to the roadmap.

The algorithms differ with respect to the connections added to the roadmap. In the RRT algorithm, only the edge $(x_{nearest}, x_{new})$ is added. In the RRG or RRT* algorithms, a set X_{near} of $k_{RRG} \log(|V|)$ nearest neighbors of x_{new} is considered[2]. For each neighbor $x_{near} \in X_{near}$ of x_{new}, the RRG algorithm checks if the path between x_{near} and x_{new} is collision-free and if so, (x_{near}, x_{new}) and (x_{new}, x_{near}) are added as additional edges to the roadmap. 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 which is invoked twice: The first time, it is used to find the node $x_{near} \in X_{near}$ which will minimize the cost to reach x_{new} . The second time, the procedure is used to attempt to minimize the cost to reach every node $x_{near} \in X_{near}$ by considering x_{new} as its parent.

III. Asymptotically near-optimal motion-planning

Clearly the asymptotic optimality of the RRT* and RRG algorithms come at the cost of the additional $k_{RRG} \log(|V|) - 1$ calls to the local planner at each stage (and some additional overhead). If we are not concerned with *asymptotically optimal*

[1]A stopping criterion can be, for example, reaching a certain number of samples or exceeding a fixed time budget.

[2]Here, k_{RRG} is a constant ensuring that the cost of paths produced by RRG and RRT* converge to the optimal cost as the number of samples grows. Choosing $k_{RRG} = 2e$ is a valid choice for all problem instances [4].

solutions, we do not have to consider all of the $k_{RRG}\log(|V|)$ 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 Σ be the set of all feasible paths, $c : \Sigma \rightarrow \mathbb{R}^+$ a cost function and c^* denote the *optimal cost*. Namely, $c^* = \min_{\sigma\in\Sigma}\{c(\sigma)\}$. In this paper we only consider *path length* as the cost function. Let $\sigma_{\text{ALG}}(n)$ be the path produced by an algorithm ALG using n free samples. It is *asymptotically optimal* if $\lim_{n\rightarrow\infty} c(\sigma_{\text{ALG}}(n)) = c^*$ with probability one and *asymptotically* $(1+\varepsilon)$*-optimal* if $\lim_{n\rightarrow\infty} c(\sigma_{\text{ALG}}(n)) \leq (1+\varepsilon)\cdot c^*$ with probability one. We refer to an algorithm that computes an asymptotically $(1+\varepsilon)$-optimal path as *asymptotically near-optimal* and to $(1+\varepsilon)$ as the *approximation factor*.

Let $\texttt{order}_{\mathcal{G}}(X, x)$ be a procedure which returns the elements of X ordered according to the cost to reach x from x_{init} through an element x' of X. Namely, after ordering, $x_1 \in X$ is before $x_2 \in X$ if $\texttt{cost}_{\mathcal{G}}(x_1) + \texttt{cost}(x_1, x) \leq \texttt{cost}_{\mathcal{G}}(x_2) + \texttt{cost}(x_2, x)$. The importance of this ordering will be discussed in the next subsection.

B. LBT-RRT

We propose a modification to the RRT* algorithm by maintaining two roadmaps $\mathcal{T}_{lb}, \mathcal{T}_{apx}$ simultaneously. Each roadmap is a tree rooted at x_{init} and consist of the same set of vertices but differ in their edge set[3].

Let $\mathcal{G}_{RRG}$ be the roadmap constructed by RRG if run on the same sequence of samples used for LBT-RRT. Now, $\mathcal{T}_{lb}, \mathcal{T}_{apx}$ maintain the following invariants throughout the algorithm:

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

and

Lower bound invariant - For every node $x \in \mathcal{T}_{apx}, \mathcal{T}_{lb}$, $\texttt{cost}_{\mathcal{T}_{lb}}(x) \leq \texttt{cost}_{\mathcal{G}_{RRG}}(x)$.

The lower bound invariant is maintained by constructing $\mathcal{T}_{lb}$ as a subgraph of $\mathcal{G}_{RRG}$, 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 segment connecting its end configurations.

[3]The subscript of $\mathcal{T}_{lb}$ is an abbreviation for lower bound and the subscript of $\mathcal{T}_{apx}$ is an abbreviation for approximation.

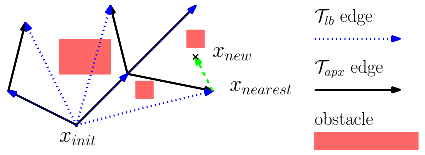


Fig. 1. Adding a new configuration x_{new} to the roadmaps. Obstacles are depicted in pink, $\mathcal{T}_{apx}$ is depicted by solid black arrows and $\mathcal{T}_{lb}$ is depicted be dotted blue arrows. The new edge added is depicted by a dotted green arrow.

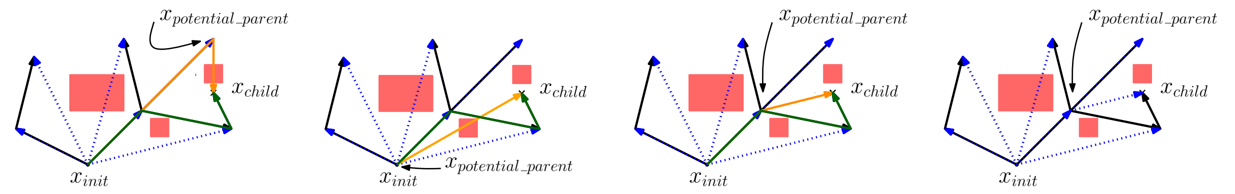
The main body of the algorithm (see Algorithm 1) follows the structure of the RRT, RRT* and RRG algorithms with respect to adding a new milestone (lines 3-7) but differs in the connections added. If a path between the new node x_{new} and its nearest neighbor $x_{nearest}$ is indeed collision-free, it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

Similar to RRT*, LBT-RRT locates the set X_{near} of $k_{RRG}\log(|V|)$ nearest neighbors of x_{new} (line 11). Then, it uses a rewiring procure `rewire` (Algorithm 2) to assess **(i)** which node $x_{near} \in X_{near}$ should be the parent of x_{new} in each tree (lines 13-14) and **(ii)** if x_{new} may improve the cost to reach any node $x_{near} \in X_{near}$ in each tree (lines 15-16). If so, the roadmaps are updated. In contrast to RRT* where rewiring is *always* performed, LBT-RRT only performs rewiring that is necessary to maintain the two invariants.

Given two nodes $x_{potential\_parent}$, x_{child}, the subroutine `rewire` (Algorithm 2) checks if an edge should be added between $x_{potential\_parent}$ and x_{child}. If so, the newly introduced edge replaces the existing edge between the parent of x_{child} and x_{child} (namely a rewiring occurs). Initially the cost $potential\_cost_{lb}$ to reach x_{child} in $\mathcal{T}_{lb}$ using $x_{potential\_parent}$ as the parent of x_{child} is computed (lines 1-2). The cost of the existing path to x_{child} in $\mathcal{T}_{lb}$ is either **(i)** smaller than $potential\_cost_{lb}$, **(ii)** larger than $potential\_cost_{lb}$ and rewiring would violate the bounded approximation invariant or **(iii)** larger than $potential\_cost_{lb}$ but rewiring would not violate the bounded approximation invariant.

Each of the above cases is handled by Algorithm 2: If the cost of the existing path to x_{child} in $\mathcal{T}_{lb}$ is lower than $potential\_cost_{lb}$ (case (i)), then no action is taken (lines 3-4 and Figure 2a). If not, then a potentially lower-cost path may exist (recall that the edge between $x_{potential\_parent}$ and x_{child} may not be collision-free). If the bounded approximation invariant is violated (case (ii), line 5 and Figure 2b), then one must explicitly check if the connection is collision-free (line 6). If this is indeed the case then both $\mathcal{T}_{lb}$ and $\mathcal{T}_{apx}$ are rewired. If the cost does not violate the bounded approximation invariant (case (iii), line 9 and Figure 2c) then only $\mathcal{T}_{lb}$ is rewired (line 10 and Figure 2d).

Now we can explain the advantage in ordering the nodes (Algorithm 1, line 12). The set X_{near} will be traversed from the node that may yield the lowest bound to reach x_{new} to the node that will yield the highest lower bound (Algorithm 1, lines 13-14). After the first rewiring, either with or without a call to the collision detector (Algorithm 2 lines 7,8 and line 10, respectively), no subsequent node can improve the cost



(a) Existing path is superior to potential path (Line 3)

(b) Potential path is superior to existing path and violates the bounded approximation invariant (Line 5)

(c) Potential path is superior to existing path but does not violate the bounded approximation invariant (Line 9)

(d) Potential path is added with no collision checking to $\mathcal{T}_{lb}$ (Line 10)

Fig. 2. Different cases handled by Algorithm 2. Obstacles are depicted in pink and the edges of $\mathcal{T}_{lb}$ and $\mathcal{T}_{apx}$ are depicted by dotted blue and solid black lines, respectively. Existing path from x_{init} to x_{child} is depicted in green while the potential path is depicted in orange.

Algorithm 1 LBT-RRT (x_{init}, ε)

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

Algorithm 2 rewire$(x_{potential_parent}, x_{child})$

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

to reach x_{new}. Indeed, after such rewiring, for each subsequent node, Algorithm 2 will not contain any (computationally demanding) calls to the collision detector. We note that this ordering is used in practice to speed up RRT*.

C. Analysis

We show that Algorithm 1 maintains the lower bound invariant (Corollary III.5) and that after every iteration the bounded approximation invariant is maintained (Lemma III.6).

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.1. *A node is added to $\mathcal{T}_{lb}$ and to $\mathcal{T}_{apx}$ if and only if a node is added to $\mathcal{G}_{RRG}$ (Algorithm 1, lines 3-8).*

Observation III.2. *Both LBT-RRT and RRG consider the same set of $k_{RRG}\log(|V|)$ nearest neighbors of x_{new} (Algorithm 1, line 11).*

Observation III.3. *$\mathcal{T}_{lb}$ undergoes the same rewiring process as RRT* (see Algorithm 2, lines 7, 10) with possibly some additional edges that are not collision-free (see Algorithm 2, line 10).*

Observation III.4. *Every edge of $\mathcal{T}_{apx}$ is collision free (see Algorithm 1, line 10 and Algorithm 2, line 8).*

Thus, the following corollary trivially holds:

Corollary III.5. *For each node $x \in \mathcal{T}_{lb}$, $\mathtt{cost}_{\mathcal{T}_{lb}}(x) \leq \mathtt{cost}_{\mathcal{G}_{RRG}}(x)$*

Using Observations III.1 through III.4 and Corollary III.5,

Lemma III.6. *After every iteration of Algorithm 1 (lines 3-16) the bounded approximation invariant is maintained.*

Proof: The edges of $\mathcal{T}_{lb}, \mathcal{T}_{apx}$ are updated in one of the 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 Algorithm 1, lines 8-10). Assume that the invariant was maintained prior to this step. Then,

$$\begin{aligned}\mathtt{cost}_{\mathcal{T}_{apx}}(x_{new}) &= \mathtt{cost}_{\mathcal{T}_{apx}}(x_{nearest}) + \\ &\quad\ \mathtt{cost}(x_{nearest}, x_{new}) \\ &\leq (1+\varepsilon)\cdot \mathtt{cost}_{\mathcal{T}_{lb}}(x_{nearest}) + \\ &\quad\ \mathtt{cost}(x_{nearest}, x_{new}) \\ &\leq (1+\varepsilon)\cdot \mathtt{cost}_{\mathcal{T}_{lb}}(x_{new}).\end{aligned}$$

case (b): Additional rewiring may occur in Algorithm 2, 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 2, line 10. This occurs when $(\texttt{cost}_{\mathcal{T}_{apx}}(x_{child}) \leq (1+\varepsilon) \cdot potential\_cost_{lb})$. In this case only $\mathcal{T}_{lb}$ is updated and:

$$\begin{aligned} \texttt{cost}_{\mathcal{T}_{apx}}(x_{child}) &\leq (1+\varepsilon) \cdot potential\_cost_{lb} \\ &= (1+\varepsilon) \cdot \texttt{cost}_{\mathcal{T}_{lb}}(x_{child}). \end{aligned}$$

■

From Corollary III.5, Lemma III.6 and using the asymptotic optimality of RRG:

Theorem III.7. *LBT-RRT is asymptotically near-optimal with an approximation factor of* $(1+\varepsilon)$.

D. Discussion

Let $\mathrm{T}^{\omega}_{ALG}$ denote the time needed for an algorithm ALG to find a feasible solution on a set of random samples ω. Clearly, $\mathrm{T}^{\omega}_{\mathrm{RRT}} \leq \mathrm{T}^{\omega}_{\mathrm{RRT}^*}$ (as the RRT* algorithm may require more calls to the collision detector than the RRT algorithm). Moreover for every $\varepsilon_1 \leq \varepsilon_2$ it holds that

$$\mathrm{T}^{\omega}_{\mathrm{RRT}} \leq \mathrm{T}^{\omega}_{\mathrm{LBT-RRT}(\varepsilon_2)} \leq \mathrm{T}^{\omega}_{\mathrm{LBT-RRT}(\varepsilon_1)} \leq \mathrm{T}^{\omega}_{\mathrm{RRT}^*}.$$

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

RRT and RRT* have been used in numerous applications and various implementations and heuristics have been suggested for them. Typically, the applications rely on the efficiency of RRT or the asymptotic optimality of RRT*.

We argue that LBT-RRT can replace RRT and RRT* in many applications. If the overhead of running LBT-RRT when compared to RRT is acceptable, then the algorithm will be able to produce *high-quality* paths in different settings. Additionally, if one is concerned with *fast convergence* to a high quality solution, LBT-RRT can replace RRT* seamlessly, trading the asymptotic optimality with a weaker near-optimality guarantee. For a partial list of examples where LBT-RRT can replace RRT or RRT*, the reader is referred to [24].

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. Again, for a partial list of examples, the reader is referred to [24].

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 3). All experiments were run using the Open Motion Planning Library (OMPL 0.10.2) [25] on a 3.4GHz Intel Core i7 processor with 8GB of memory. RRT* was implemented by using $\varepsilon = 0$ which outperforms a naive implementation of RRT* due to the use of the `order` function (see Algorithm 1 line 12 and [17]).

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. Finally, the cubicles scenario consists of two L-shaped robots free-flying in space that need to exchange locations amidst obstacles[4].

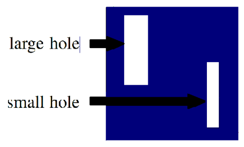


Fig. 4. One barrier of the Alternating barriers scenario.

We compare the performance of LBT-RRT with RRT and RRT* when a fixed time budget is given. We consider $(1+\varepsilon)$ 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). Each result is averaged over 100 different runs.

Figure 5 depicts similar behavior for all scenarios: 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 RRT 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 ε 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 ε decreases and is comparable to that of RRT* for low values of ε.

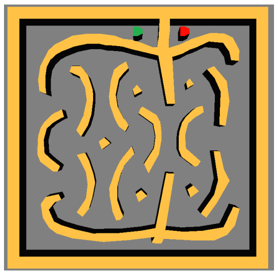
Thus, Figures 5, 6 should be looked at simultaneously as they encompass the tradeoff between speed to find *any* solution and the quality of the solution found. For supplementary material and in-depth analysis of the results, the reader is referred to the extended version of this paper [24] and to http://acg.cs.tau.ac.il/projects/LBT-RRT/project-page.

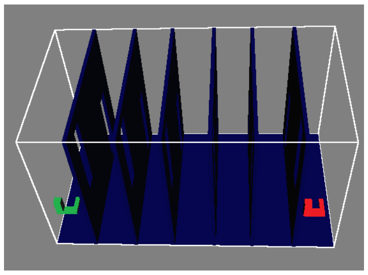
V. Future work

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 framework: Run LBT-RRT with a large approximation factor, once the stopping criterion has been met, decrease the approximation factor and continue running. This may allow an even quicker convergence to find any feasible path while allowing for refinement as time permits (similar to [21]).

Additionally, we wish to apply our framework to different quality measures. For certain 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.

[4]The Maze Scenario and the Cubicles Scenario are provided as part of the OMPL distribution.


(a) Maze scenario


(b) Alternating barriers scenario

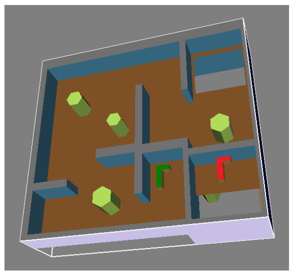
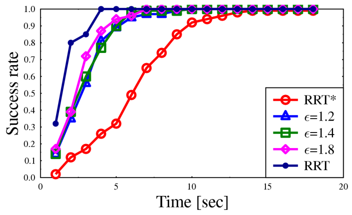
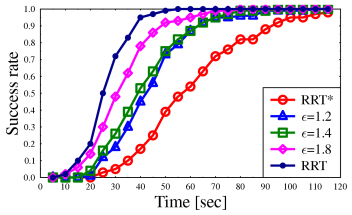

(c) Cubicles scenario (2-robots)

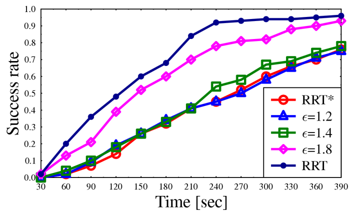
Fig. 3. Benchmark scenarios. The start and goal configuration are depicted in green and red, respectively.



(a) Maze Scenario

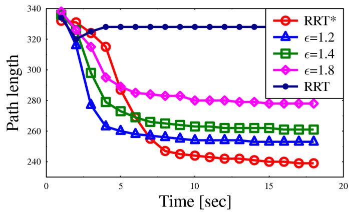


(b) Alternating barriers scenario

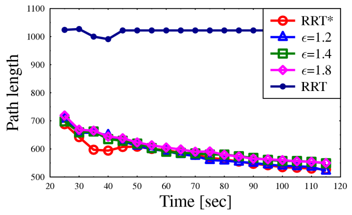


(c) Cubicles scenario

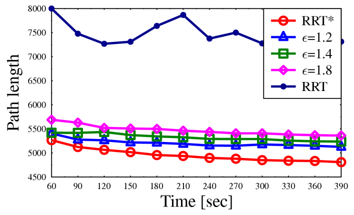
Fig. 5. Success rate for algorithms on different scenarios.



(a) Maze Scenario



(b) Alternating barriers scenario



(c) Cubicles scenario

Fig. 6. Path lengths for algorithms on different scenarios.

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 Trans. Robot.*, 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 *WAFR*, 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 Trans. on Robot.*, vol. 27, no. 2, pp. 365–371, 2011.

[8] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, "OBPRM: an obstacle-based PRM for 3d workspaces," in *WAFR*. Natick, MA, USA: A. K. Peters, Ltd., 1998, pp. 155–168.

[9] J.-M. Lien, S. L. Thomas, and N. M. Amato, "A general framework for sampling on the medial axis of the free space," in *ICRA*, 2003, pp. 4439–4444.

[10] C. Urmson and R. G. Simmons, "Approaches for heuristically biasing RRT growth," in *IROS*, 2003, pp. 1178–1183.

[11] T. Siméon, J.-P. Laumond, and C. Nissoux, "Visibility-based probabilistic roadmaps for motion planning," *Advanced Robotics*, vol. 14, no. 6, pp. 477–493, 2000.

[12] R. Geraerts and M. H. Overmars, "Creating high-quality roadmaps for motion planning in virtual environments," in *IROS*, 2006, pp. 4355–4361.

[13] N. A. Wedge and M. S. Branicky, "On heavy-tailed runtimes and restarts in rapidly-exploring random trees," in *AAAI*, 2008, pp. 127–133.

[14] D. Shaharabani, O. Salzman, P. K. Agarwal, and D. Halperin, "Sparsification of motion-planning roadmaps by edge contraction," in *ICRA*, 2013, pp. 4083–4090.

[15] J. D. Marble and K. E. Bekris, "Computing spanners of asymptotically optimal probabilistic roadmaps," in *IROS*, 2011, pp. 4292–4298.

[16] A. Dobson and K. E. Bekris, "Improving sparse roadmap spanners," in *ICRA*, 2013, pp. 4091–4096.

[17] A. Perez, S. Karaman, A. Shkolnik, E. Frazzoli, S. Teller, and M. Walter, "Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms," in *IROS*, 2011, pp. 4307–4313.

[18] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan, "RRT*-Smart: Rapid convergence implementation of RRT* towards optimal solution," in *ICMA*, 2012.

[19] O. Arslan and P. Tsiotras, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *ICRA*, 2013, pp. 2413–2420.

[20] D. Ferguson and A. Stentz, "Anytime RRTs," in *IROS*, 2006, pp. 5369 – 5375.

[21] R. Alterovitz, S. Patil, and A. Derbakova, "Rapidly-exploring roadmaps: Weighing exploration vs. refinement in optimal motion planning," in *ICRA*, 2011, pp. 3706–3712.

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

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

[24] O. Salzman and D. Halperin, "Asymptotically near-optimal RRT for fast, high-quality, motion planning," *CoRR*, vol. abs/1308.0189, abs/1308.0189, 2013.

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