Asymptotically Near-Optimal RRT for Fast, High-Quality Motion Planning

Oren Salzman and Dan Halperin

***Abstract*—We present *lower bound tree-RRT* (LBT-RRT), a single-query sampling-based motion-planning 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 cost function is the path length. When the approximation factor is 1 (i.e., no approximation is allowed), LBT-RRT behaves like RRG. When the approximation factor is unbounded, LBT-RRT behaves like RRT. 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 that is a subgraph of the RRG roadmap and a second, auxiliary graph, which we call the *lower-bound* graph. The combination of the two roadmaps, which is faster to maintain than the roadmap 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 with RRG and RRT*) with little running-time overhead when compared with RRT.**

***Index Terms*—Motion control, nonholonomic motion planning.**

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, ch. 7]. Recently, in the robotics community, there has been growing interest in finding *high-quality* paths, which turns out to be a nontrivial problem [4], [5]. Quality can be measured in terms of, for example, length, clearance, smoothness, and energy, to mention a few criteria, or some combination of the above.

A. High-Quality Planning With Sampling-Based Algorithms

Unfortunately, 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.

1) Postprocessing Existing Paths: Postprocessing an existing path by applying *shortcutting* is a common, effective, approach to increase path quality; see, e.g., [6]. Typically, two nonconsecutive configurations are chosen randomly along the path. If the two configurations can be connected using a straight-line segment in the configuration space and this connection improves the quality of the original path, the segment replaces the original path that connected the two configurations. The process is continued iteratively until a termination condition holds.

2) Path Hybridization: An inherent problem with path postprocessing is that it is local in nature. A path that was postprocessed using shortcutting often remains in the same homotopy class of the original path. Carefully combining even a small number of different paths (that may be of low quality) often enables the construction of a higher quality path [7].

3) Online Optimization: Changing the sampling strategy [8]–[11] or the connection scheme to a new milestone [10], [12] are examples of heuristics proposed to create higher quality solutions. Additional approaches include, among others, useful cycles [6] and random restarts [13].

4) Asymptotically Optimal and Near-Optimal Solutions: In their 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 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*. Namely, as the number of samples tends to infinity, the solution obtained by these algorithms converges to the optimal solution with probability one. To ensure asymptotic optimality, 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 postprocessing stage of the PRM* [14], [15], or as a modification of PRM* [16]–[18].

The performance of RRT* can be improved using several heuristics that bear a resemblance to the lazy approach used in this work [19]. Additional heuristics to speed up the

Manuscript received March 3, 2015; revised September 22, 2015 and February 16, 2016; accepted February 18, 2016. Date of publication April 14, 2016; date of current version June 3, 2016. This paper was recommended for publication by Associate Editor T. Bretl and Editor C. Torras upon evaluation of the reviewers' comments. This work was supported in part by the 7th Framework Programme for Research of the European Commission, under FET-Open Grant 255827 Computational Geometry Learning (CGL), by the Israel Science Foundation under Grant 1102/11, by the German–Israeli Foundation under Grant 1150-82.6/2011, and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University. A preliminary and partial version of this paper appeared at the 2014 IEEE International Conference on Robotics and Automation.

The authors are with the Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv 6997801, Israel (e-mail: orenzalz@post.tau.ac.il; danha@post.tau.ac.il).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2016.2539377

convergence rate of RRT* were presented in RRT*-SMART [20]. Recently, RRT$^{\#}$ [21] was suggested as an asymptotically optimal algorithm with a faster convergence rate when compared with RRT*. RRT$^{\#}$ extends its roadmap in a fashion similar to RRT* but adds a replanning procedure. This procedure ensures that the tree rooted at the initial state contains the lowest cost path information for vertices that have the potential to be part of the optimal solution. Thus, in contrast with RRT* that only performs *local* rewiring of the search tree, RRT$^{\#}$ efficiently propagates changes to *all* the relevant parts of the roadmap. Janson and Pavone [22] introduced the asymptotically optimal fast marching tree algorithm (FMT*). The single-query asymptotically optimal algorithm maintains a tree as its roadmap. Similarly to PRM*, FMT* samples n collision-free nodes. It then builds a minimum-cost spanning tree rooted at the initial configuration over this set of nodes. Lazy variants have been proposed both for PRM* and RRG [23] and for FMT* [24]. Finally, we note that using nearest-neighbor data structures tailored for these algorithms additional speedup in the running times may be obtained [25].

An alternative approach to improve the running times of these algorithms is to relax the asymptotic optimality to *asymptotic near-optimality*. An algorithm is said to be asymptotically near-optimal if, given an *approximation factor* ε, the solution obtained by the algorithm converges to within a factor of $(1+\varepsilon)$ of the optimal solution with probability one, as the number of samples tends to infinity. The sparse stable RRT (SST) [26], [27] is an example of an asymptotically near-optimal algorithm. SST is best suited for systems with dynamics as it does not require solving a two-point boundary value problem (BVP). This is important as RRT* and its variants require a BVP solver in order to connect two existing nodes in their roadmap, a requirement that is sometimes impractical. However, SST requires additional parameters that affect the running time of the algorithm as well as the quality of the solution. To tune the desired behavior of SST, one needs to know the minimal clearance of the optimal path in the configuration space. Thus, while SST is indeed an asymptotically near-optimal algorithm, it remains unknown how to choose the parameters that will ensure a $(1+\varepsilon)$-approximation for a given ε.

5) 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 quickly as possible and, if time permits, it should be refined to yield a higher quality solution.

Ferguson and Stentz [28] suggest iteratively running RRT while considering only areas that may potentially improve the existing solution. Alterovitz *et al.* [29] suggest the Rapidly exploring Roadmap Algorithm (RRM), which finds an initial path similar to RRT. Once such a path is found, RRM either explores further the configuration space or refines the explored space. Luna *et al.* [30] suggest alternating between path shortcutting and path hybridization in an anytime fashion.

RRT* was also adapted for online motion planning [31]. Here, an initial path is computed and the robot begins its execution. While the robot moves along this path, the algorithm refines the part that the robot has not yet moved along.

B. Contribution

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 a factor of $(1+\varepsilon)$ of the optimal solution. LBT-RRT allows for interpolating between the fast, yet suboptimal, RRT algorithm and the asymptotically optimal RRG algorithm when the cost function is path length. By choosing $\varepsilon = 0$, no approximation is allowed and LBT-RRT maintains a roadmap identical to the one maintained by RRG. 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 roadmaps. Both roadmaps are defined over the same set of vertices, but each consists of a different set of edges. On the one hand, a path in the first roadmap may not be feasible, but its cost is always a *lower bound* on the cost of paths extracted from RRG (using the same sequence of random nodes). On the other hand, a path extracted from the second roadmap is always feasible and its cost is within a factor of $(1+\varepsilon)$ from the lower bound provided by the first roadmap.

We suggest to use LBT-RRT for high-quality anytime motion planning. We demonstrate its performance on scenarios ranging from 3 to 12 degrees of freedom (DoF) and show that the algorithm produces high-quality solutions (comparable with RRG and RRT*) with little running-time overhead when compared with RRT.

This paper is a modified and extended version of a publication presented at the 2014 IEEE International Conference on Robotics and Automation [32]. In this paper, we present additional experiments and extensions of the original algorithmic framework. Finally, we note that the conference version of this paper contained an oversight with regard to the roadmap that is used for the lower bound. We explain the problem and its fix in detail in Section III after providing all the necessary technical background.

C. Outline

In Section II, we review the RRT, RRG, and RRT* algorithms. In Section III, we present our algorithm LBT-RRT and a proof of its asymptotic near-optimality. We continue in Section IV to demonstrate in simulations its favorable characteristics on several scenarios. In Section V, we discuss a modification of the framework to further speed up the convergence to high-quality solutions. We conclude in Section by describing possible directions for future work.

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. We continue by reviewing the RRT, RRG, and RRT* algorithms.

Algorithm 1: RRT (x_{init}).

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

Algorithm 2: RRG (x_{init}).

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

Algorithm 3: RRT* (x_{init}).

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

Algorithm 4: `rewire_RRT`$^*(x_{\text{potential\_parent}}, x_{\text{child}})$.

1: **if** (`collision_free`$(x_{\text{potential\_parent}}, x_{\text{child}})$) **then**
2: $c \leftarrow$ `cost`$(x_{\text{potential\_parent}}, x_{\text{child}})$
3: **if** (cost$_{\mathcal{T}}(x_{\text{potential_parent}}) + c <$ `cost`$_{\mathcal{T}}(x_{\text{child}})$) **then**
4: $\mathcal{T}$.parent$(x_{\text{child}}) \leftarrow x_{\text{potential_parent}}$

A. Problem Definition and Terminology

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

We will make use of the following procedures throughout the paper: `sample_free`, a procedure returning a random free configuration; `nearest_neighbor`(x, V) and `nearest_neighbors`(x, V, k), 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 $\mathcal{X}_{\text{free}}$, and let `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} by using a roadmap $\mathcal{G}$. These are standard procedures used by the RRT or RRT* algorithms. Finally, we use the (generic) predicate `construct_roadmap` 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 as the underlying data structure, which is a directed tree for RRT and RRT* and a directed graph for RRG. At each iteration, a configuration x_{rand} is sampled at random. Then, x_{nearest}, the nearest configuration to x_{rand} in the roadmap, 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, x_{new} is added to the roadmap (see Algorithms 1–3, lines 3–9).

The algorithms differ in the connections added to the roadmap. In RRT, only the edge $(x_{\text{nearest}}, x_{\text{new}})$ is added. In RRG and RRT*, a set X_{near} of $k_{\text{RRG}}\log(|V|)$ nearest neighbors of x_{new} is considered. Here, k_{RRG} is a constant ensuring that the cost of paths produced by RRG and RRT* indeed converges to the optimal cost almost surely as the number of samples grows. A valid choice for all problem instances is $k_{\text{RRG}} = 2e$ [4]. For each neighbor $x_{\text{near}} \in X_{\text{near}}$ of x_{new}, RRG checks if the path between x_{near} and x_{new} is collision-free and, if so, $(x_{\text{near}}, x_{\text{new}})$ and $(x_{\text{new}}, x_{\text{near}})$ are added to the roadmap (lines 10–13). RRT* maintains a subgraph of the RRG roadmap. This is done by an additional rewiring procedure (see Algorithm 4)

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

that is invoked twice. The first time, it is used to find the node $x_{\text{near}} \in X_{\text{near}}$, which will minimize the cost to reach x_{new} (see Algorithm 3, lines 11 and 12). The second time, the procedure is used to minimize the cost to reach every node $x_{\text{near}} \in X_{\text{near}}$ by considering x_{new} as its parent (see Algorithm 3, lines 13 and 14). Thus, at all times, RRT* maintains a tree that, as mentioned, is a subgraph of the RRG roadmap.

Given a sequence of n random samples, the cost of the path obtained using the RRG algorithm is a lower bound on the cost of the path obtained using the RRT* algorithm. However, RRG requires both additional memory (to explicitly store the set of $O(\log n)$ neighbors) and exhibits longer running times (due to the additional calls to the local planner). In practice, this excess in running time is far from negligible (see Section IV), making RRT* a more suitable algorithm for asymptotically optimal motion planning.

III. Asymptotically Near-Optimal Motion Planning

Clearly, the asymptotic optimality of the RRT* and RRG algorithms comes at the cost of the additional $O(k_{\text{RRG}} \log(|V|))$ 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 *estimate* the quality of each edge. We use this estimate of the quality of the edge to decide whether 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, although we still need to estimate the quality of many edges. Our approach is viable in cases in which such an assessment can be carried out efficiently. Namely, more efficiently than deciding if an edge is collision-free. This condition holds naturally when the quality measure is *path length*, which is the cost function considered in this paper. Note that this does not necessarily have to the be Euclidean distance. It may be a weighted combination of the translational and rotational motions of the robot (as is demonstrated in Section IV). For a discussion on different cost functions, see Section VI.

A. Single-Sink Shortest-Path Problem

As we will see, our algorithm needs to maintain the shortest path from x_{init} to any node in a graph. Moreover, this graph undergoes a series of edge insertions and edge deletions. This problem is referred to as the fully dynamic *single-source shortest-path problem* or SSSP for short. Efficient algorithms [33], [34] exist that can store the minimal cost to reach each node (and the corresponding path) in such settings from a source node. In our setting, this source node is x_{init}. We make use of the following procedures, which are provided by SSSP algorithms: $\texttt{delete\_edge}_{\texttt{SSSP}}(\mathcal{G}, (x_1, x_2))$ and $\texttt{insert\_edge}_{\texttt{SSSP}}(\mathcal{G}, (x_1, x_2))$ that delete and insert, respectively, the edge (x_1, x_2) from/into the graph $\mathcal{G}$ while maintaining $\text{cost}_{\mathcal{G}}$ for each node. We assume that these procedures return the set of nodes whose cost has changed due to the edge deletion or edge insertion. Furthermore, let $\texttt{parent}_{\texttt{SSSP}}(\mathcal{G}, x)$ be a procedure returning the parent of x in the shortest path from the source to x in $\mathcal{G}$.

Algorithm 5: LBT-RRT ($x_{\text{init}}, \varepsilon$).

```
1:  T_lb.G ← {x_init}   T_apx.V ← {x_init}
2:  while construct_roadmap() do
3:      x_rand ← sample_free()
4:      x_nearest ← nearest_neighbor(x_rand, T_lb.V)
5:      x_new ← steer(x_nearest, x_rand)
6:      if (!collision_free(x_nearest, x_new)) then
7:          CONTINUE
8:      T_apx.V ← T_apx.V ∪ {x_new}
9:      T_apx.parent(x_new) ← x_nearest
10:     G_lb.V ← G_lb.V ∪ {x_new}
11:     insert_edge_SSSP(G_lb, (x_nearest, x_new))
12:     X_near ← nearest_neighbors(x_new,
                G_lb.V, k_RRG log(|G_lb.V|))
13:     for all (x_near, X_near) do
14:         consider_edge (x_near, x_new)
15:     for all (x_near, X_near) do
16:         consider_edge (x_new, x_near)
```

B. Lower Bound Tree-RRT

We propose a modification to the RRG algorithm by maintaining two roadmaps $\mathcal{G}_{\text{lb}}, \mathcal{T}_{\text{apx}}$ simultaneously. Both roadmaps have the same set of vertices but differ in their edge set. $\mathcal{G}_{\text{lb}}$ is a graph and $\mathcal{T}_{\text{apx}}$ is a tree rooted at x_{init}.[2]

Let $\mathcal{G}_{\text{RRG}}$ be the roadmap constructed by RRG if run on the same sequence of samples used for LBT-RRT. The following invariants are maintained by the LBT-RRT algorithm:

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

and

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

The lower bound invariant is maintained by ensuring that the edges of $\mathcal{G}_{\text{RRG}}$ are a subset of the edges of $\mathcal{G}_{\text{lb}}$. As we will see, $\mathcal{G}_{\text{lb}}$ may possibly contain some edges that $\mathcal{G}_{\text{RRG}}$ considered but found to be in collision.

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_{new} and its nearest neighbor x_{nearest} is indeed collision-free, it is added to both roadmaps together with an edge from x_{nearest} to x_{new} (lines 8–11).

Similar to RRG and RRT*, LBT-RRT locates the set X_{near} of $k_{\text{RRG}} \log(|V|)$ nearest neighbors of x_{new} (line 12). Then, for

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

Algorithm 6: consider_edge(x_1, x_2).

1: $I \leftarrow \texttt{insert\_edge}_{\texttt{SSSP}}(\mathcal{G}_{\text{lb}}, (x_1, x_2))$
2: $Q \leftarrow \{x \in I \mid \text{cost}_{\mathcal{T}_{\text{apx}}}(x) > (1+\varepsilon) \cdot \text{cost}_{\mathcal{G}_{\text{lb}}}(x)\}$
3: **while** $Q \neq \emptyset$ **do**
4: $x \leftarrow Q.\texttt{top}()$;
5: **if** $\text{cost}_{\mathcal{T}_{\text{apx}}}(x) > (1+\varepsilon) \cdot \text{cost}_{\mathcal{G}_{\text{lb}}}(x)$ **then**
6: $x_{\text{parent}} \leftarrow \texttt{parent}_{\texttt{SSSP}}(\mathcal{G}_{\text{lb}}, x)$
7: **if** $(\texttt{collision\_free}\,(x_{\text{parent}}, x))$ **then**
8: $\mathcal{T}_{\text{apx}}.\text{parent}(x) \leftarrow x_{\text{parent}}$
9: $Q.\texttt{pop}()$
10: **else**
11: $D \leftarrow \texttt{delete\_edge}_{\texttt{SSSP}}(\mathcal{G}_{\text{lb}}, (x_{\text{parent}}, x))$
12: **for all** $y \in D \cap Q$ **do**
13: $Q.\texttt{update\_cost}(y)$
14: **else**
15: $Q.\texttt{pop}()$

each edge connecting a node from X_{near} to x_{new} and for each edge connecting x_{new} to a node from X_{near}, it uses a procedure consider_edge (see Algorithm 6) to assess if the edge should be inserted to either roadmap. The edge is first lazily inserted into $\mathcal{G}_{\text{lb}}$ without checking if it is collision-free. This *may* cause the bounded approximation invariant to be violated, which in turn will induce a call to the local planner for a set of edges. Each such edge might either be inserted into $\mathcal{T}_{\text{apx}}$ or removed from $\mathcal{G}_{\text{lb}}$.

This is done as follows. First, the edge considered is inserted to $\mathcal{G}_{\text{lb}}$ while updating the shortest path to reach each vertex in $\mathcal{G}_{\text{lb}}$ (see Algorithm 6, line 1). Denote by I the set of updated vertices after the edge insertion. Namely, for every $x \in I$, $\texttt{cost}_{\mathcal{G}_{\text{lb}}}(x)$ has decreased due to the edge insertion. This cost decrease may, in turn, cause the bounded approximation invariant to be violated for some nodes in U. All such nodes are collected and inserted into a priority queue Q (line 2) ordered according to $\texttt{cost}_{\mathcal{G}_{\text{lb}}}$ from low to high. Now, the algorithm proceeds in iterations until the queue is empty (lines 3–15). At each iteration, the head of the queue x is considered (line 4). If the bounded approximation invariant does not hold (line 5), the algorithm checks if the edge in $\mathcal{G}_{\text{lb}}$ connecting the node x to its parent along the shortest path to x_{init} is collision-free (lines 6–7). If this is the case, the approximation tree is updated (line 8), and the head of the queue is removed (line 9). If not, the edge is removed from $\mathcal{G}_{\text{lb}}$ (line 11). This causes an increase in $\text{cost}_{\mathcal{G}_{\text{lb}}}$ for a set D of nodes, some of which are already in the priority queue. Clearly, the bounded approximation invariant holds for the nodes $x \in D$ that are not in the priority queue. Thus, we take only the nodes $x \in D$ that are already in Q and update their location in Q according to their new cost (lines 12 and 13). Finally, if the bounded approximation invariant holds for x, then it is removed from the queue (line 15).

C. Analysis

In this section, we show that Algorithm 5 maintains the lower bound invariant (see Corollary III.5) and that after every iteration of the algorithm, the bounded approximation invariant is maintained (see Lemma III.8). We then report on the time complexity of the algorithm (see Corollary III.10).

We note the following straightforward yet helpful observations comparing LBT-RRT and RRG when run on the same sequence of random samples:

Observation III.1: A node x is added to $\mathcal{G}_{\text{lb}}$ and to $\mathcal{T}_{\text{apx}}$ if and only if x is added to $\mathcal{G}_{\text{RRG}}$ (see Algorithm 2, lines 3–8 and Algorithm 5, lines 3–11).

Observation III.2: Both LBT-RRT and RRG consider the same set of $k_{\text{RRG}} \log(|V|)$ nearest neighbors of x_{new} (see Algorithm 2, line 10, and Algorithm 5, line 12).

Observation III.3: Every edge added to the RRG roadmap (see Algorithm 2, line 13) is added to $\mathcal{G}_{\text{lb}}$ (see Algorithm 5, lines 14, 16, and Algorithm 6, line 1).

Note that some additional edges may be added to $\mathcal{G}_{\text{lb}}$ that are not added to the RRG roadmap as they are not collision-free.

Observation III.4: Every edge of $\mathcal{T}_{\text{apx}}$ is collision free (see Algorithm 5, line 9, and Algorithm 6, line 8).

Thus, the following corollary trivially holds.

Corollary III.5: After every iteration of LBT-RRT (see Algorithm 5, lines 3–16), the lower bound invariant is maintained.

We continue with the following observations relevant to the analysis of the procedure $\texttt{consider\_edge}(x_1, x_2)$:

Observation III.6: The only place where $\text{cost}_{\mathcal{G}_{\text{lb}}}$ is decreased is during a call to $\texttt{insert\_edge}_{\texttt{SSSP}}(\mathcal{G}_{\text{lb}}, (x_1, x_2))$ (see Algorithm 6, line 1).

Observation III.7: A node x is removed from the queue Q (see Algorithms 6, lines 9 and 15) only if the bounded approximation invariant holds for x.

Showing that the bounded approximation invariant is maintained is done by induction on the number of calls to $\texttt{consider\_edge}(x_1, x_2)$. Using Observation III.6, prior to the first call to $\texttt{consider\_edge}(x_1, x_2)$, the bounded approximation invariant is maintained. Thus, we need to show the following.

Lemma III.8: If the bounded approximation invariant holds prior to a call to the procedure $\texttt{consider\_edge}(x_1, x_2)$ (see Algorithm 6), then the procedure will terminate with the invariant maintained.

Proof: Assume that the bounded approximation invariant was maintained prior to a call to $\texttt{consider\_edge}(x_1, x_2)$. By Observation III.6, inserting a new edge (line 1) may cause the bounded approximation invariant to be violated for a set of nodes. Moreover, it is the *only* place where such an event can occur. Observation III.7 implies that the bounded approximation invariant holds for every vertex *not* in Q.

Recall that, in the priority queue, we order the nodes according to $\texttt{cost}_{\mathcal{G}_{\text{lb}}}$ (from low to high) and at each iteration of $\texttt{consider\_edge}(x_1, x_2)$ the top of the priority queue x is considered. The parent x_{parent} of x, which has a smaller cost value, cannot be in the priority queue. Thus, the bounded approximation invariant holds for x_{parent}. Namely

$$\texttt{cost}_{\mathcal{T}_{\text{apx}}}(x_{\text{parent}}) \leq (1+\varepsilon) \cdot \texttt{cost}_{\mathcal{G}_{\text{lb}}}(x_{\text{parent}}).$$

Now, if the edge between x_{parent} and x is found to be free (line 7), we update the approximation tree (line 8). It follows that after

such an event

$$\begin{aligned}\texttt{cost}_{\mathcal{T}_{\text{apx}}}(x) &= \texttt{cost}_{\mathcal{T}_{\text{apx}}}(x_{\text{parent}}) + \texttt{cost}(x_{\text{parent}}, x)\\ &\leq (1+\varepsilon)\cdot \texttt{cost}_{\mathcal{G}_{\text{lb}}}(x_{\text{parent}}) + \texttt{cost}(x_{\text{parent}}, x)\\ &\leq (1+\varepsilon)\cdot \texttt{cost}_{\mathcal{G}_{\text{lb}}}(x).\end{aligned}$$

Namely, after updating the approximation tree, the bounded approximation invariant holds for the node x.

To summarize, at each iteration of Algorithm 6 (lines 3–16), either: 1) we remove a node x from Q (line 9 or line 15) or 2) we remove an incoming edge to the node x from the lower bound graph (line 11). If the node x was removed from Q (case 1), the bounded approximation invariant holds—either it was not violated to begin with (line 15) or it holds after updating the approximation tree (lines 8 and 9).

To finish the proof, we need to show that the main loop (lines 3–15) in Algorithm 6 indeed terminates. Recall that the degree of each node is $O(\log n)$. Thus, a node x cannot be at the head of the queue more than $O(\log n)$ times (after each time, we either remove an incoming edge or remove x from the queue). This, in turn, implies that after at most $O(n \log n)$ iterations, Q is empty and the main loop terminates. ■

From Corollary III.5, Lemma III.8, and using the asymptotic optimality of RRG, we conclude the following.

Theorem III.9: 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 at most $(1+\varepsilon)$ times the cost of the optimal path almost surely.

We continue now to discuss the time complexity of the algorithm. If δ is the number of nodes updated during a call to an SSSP procedure[3] (namely, $\texttt{insert\_edge}_{\text{SSSP}}$ or $\texttt{delete\_edge}_{\text{SSSP}}$), then the complexity of the procedure is $O(\delta \log n)$ when using the algorithm of Ramalingam and Reps [34]. Set $\hat{\delta}$ to be the maximum value of δ over all calls to SSSP procedures (see Algorithm 5, line 11, and Algorithm 6, lines 1 and 11), and let n denote the final number of samples used by LBT-RRT.

We have $O(n \log n)$ edges and each edge will be inserted to $\mathcal{G}_{\text{lb}}$ once (see Algorithm 5, line 11, or Algorithm 6, line 1) and possibly be removed from $\mathcal{G}_{\text{lb}}$ once (Algorithm 6, line 11). Therefore, the total complexity due to the SSSP procedures is $O(\hat{\delta} \cdot n \log^2 n)$. The time complexity of all the other operations (nearest neighbors, collision detection, etc.) is similar to RRG, which runs in time $O(n \log n)$.

Corollary III.10: LBT-RRT runs in time $O(\hat{\delta} \cdot n \log^2 n)$, where n is the number of samples and $\hat{\delta}$ is the maximal number of nodes updated over all SSSP procedures.

While this running time may seem discouraging, we note that, in practice, the local planning dominates the actual running time of the algorithm in practice. As we demonstrate in Section IV through various simulations, LBT-RRT produces high-quality results in an efficient manner.

[3]The number of nodes δ updated during an SSSP procedure depends on the topology of the graph and the edge weights. Theoretically, in the worst case, $\delta = O(n)$ and a dynamic SSSP algorithm cannot perform better than recomputing shortest paths from scratch. However, in practice, this value is much smaller.

D. Implementation Details

We describe the following optimizations that we use in order to speed up the running time of the algorithm. The first is that the set X_{near} is ordered according to the cost to reach x_{new} from x_{init} through an element x of X_{near}. Hence, the set X_{near} will be traversed from the node that yields the smallest lower bound to reach x_{new} to the node that will yield the highest lower bound. After the first edge that does *not* violate the bounded approximation invariant, no subsequent node can improve the cost to reach x_{new} and $\texttt{insert\_edge}_{\text{SSSP}}$ will not need to perform any updates. This ordering was previously used to speed up RRT* (see, e.g., [19] and [35]).

The second optimization comes to avoid the situation in which $\texttt{insert\_edge}_{\text{SSSP}}$ is called and immediately afterward the same edge is removed. Hence, given an edge, we first check if the bounded approximation invariant will be violated had the edge been inserted. If this is indeed the case, the local planner is invoked and, only if the edge is collision free, $\texttt{insert\_edge}_{\text{SSSP}}$ is called. *Remark:* The conference version of this paper contained an oversight with regard to how the bounded approximation invariant was maintained. Specifically, instead of storing $\mathcal{G}_{\text{lb}}$ as a graph, a tree was stored, which was rewired locally. When the algorithm tested if the bounded approximation invariant was violated for a node x, it only considered the *children* of x in the tree. This local test did not take into account the fact that changing the cost of x in the tree could also change the cost of nodes y that are descendants of x (but not its children). The implications of the oversight is that the algorithm was not asymptotically near optimal. The experimental results presented in the conference version of this paper suggest that, in certain scenarios, this oversight did not have a significant effect on the convergence to high-quality solutions. Having said that, LBT-RRT as presented in this paper is both asymptotically near optimal and converges to high-quality solutions faster than the original algorithm.

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 (see Fig. 1). The algorithm was implemented using the Open Motion Planning Library (OMPL 0.10.2) [36] and our implementation is distributed with the latest OMPL release. All experiments were run on a 2.8-GHz Intel Core i7 processor with 8 GB of memory. We employ the OMPL definition of path length for our benchmarks, which uses a weighted combination between the Euclidean distance and angular distance; see [36] for details. Typical implementations of RRT-type algorithms have an additional parameter called the step size, which is the maximal distance between the new node added to the tree and its parent. In our implementation, we used the default value provided by OMPL as the step size, which is 0.2Δ. Here, Δ is the maximum possible distance between any pair of configurations in the underlying C-space.

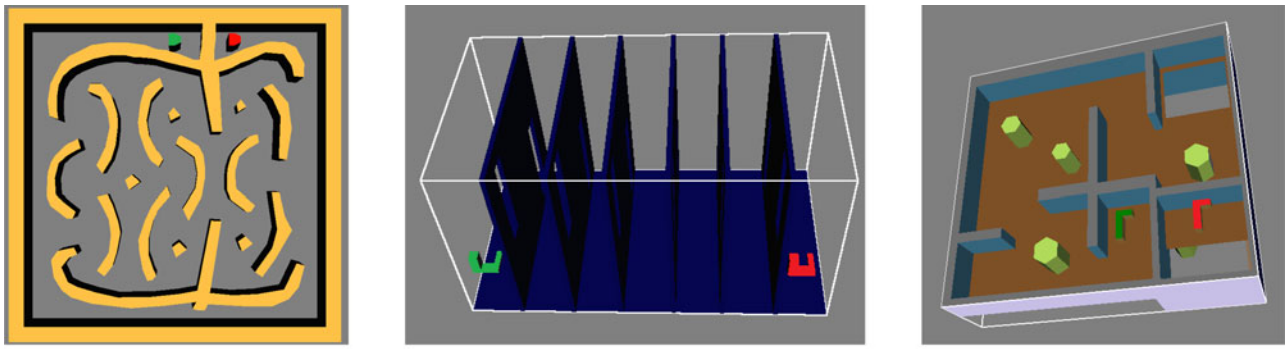

Fig. 1. Benchmark scenarios. The start and goal configuration are depicted in green and red, respectively. (a) Maze scenario. (b) Alternating barriers scenario. (c) Cubicles scenario (two robots).

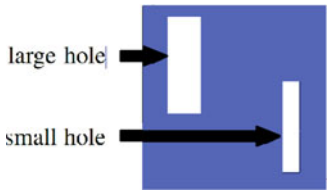

Fig. 2. One barrier of the alternating barriers scenario.

The Maze scenario [see Fig. 1(a)] consists of a planar polygonal robot that can translate and rotate. The Alternating barriers scenario [see Fig. 1(b)] 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 Fig. 2. 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.[4]

We compare the performance of LBT-RRT with SST, RRT, RRG, and RRT* when a fixed time budget is given. RRT* was implemented by using the ordering optimization described in Section III and [19]. The implementation of SST was adapted from the publicly available source code (http://pracsyslab.org/sst_software). SST requires two parameters δ_{BN} and δ_s that affect its approximation factor and its convergence rate. Following [27], we chose $\delta_{\mathrm{BN}} = 1.5 \cdot \delta_s$, thus having only one parameter to tune. We performed a binary search over a wide range of values of δ_s for each scenario and chose to plot the parameters that yielded results that were the most competitive with LBT-RRT. For LBT-RRT, we consider $(1+\varepsilon)$ values of $1.2, 1.4, 1.8$. We report on the success rate of each algorithm (see Fig. 3) and the path length after applying shortcuts (see Fig. 4). Each benchmark was run 100 different times; however, the path length reported is averaged over all the runs that have found a solution.

Fig. 3 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 RRT is typically highest, while that of the RRT* and RRG is lowest. The success rate for LBT-RRT for a specific time budget typically increases as the value of ε increases. The SST proved to have comparable success rates in finding a solution in the Maze scenario and in the Alternating barriers scenario. It failed to find a solution for any choice of parameters for the Cubicles scenario within the given time budget of 6 min. Fig. 4 also depicts similar behavior for all scenarios: The average path length decreases for all algorithms (except for RRT). The average path length for LBT-RRT typically decreases as the value of ε decreases and is comparable with that of RRT* for low values of ε. When SST found a solution, its rate of convergence to a high-quality one was significantly slower than LBT-RRT and RRT*. This is not very surprising as SST is tailored for problems with dynamics and it is in such scenarios where it typically is more computationally efficient than RRT*. We note that although RRG is asymptotically optimal, its overhead makes it a poor algorithm when one desires a *high-quality* solution very quickly.

Thus, Figs. 3 and 4 should be looked at simultaneously as they encompass the tradeoff between speed to find *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 *any* solution [see Fig. 3(b)], one can see that RRT manages to achieve a success rate of 70% after 45 s. RRT*, on the other hand, requires 90 s to achieve the same success rate (exactly double the time). For all different values of ε, LBT-RRT manages to achieve a success rate of 70% after 70 s (around 66% overhead when compared with RRT). Now, considering the path length at 70 s, typically the paths extracted from LBT-RRT yield the same quality when compared with RRT* while ensuring a high success rate.

The same behavior of finding paths of high quality (similar to the quality that RRT* produces) within the time frames that RRT requires in order to find *any* solution has been observed for both the Maze scenario and the Cubicles scenario. Results

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

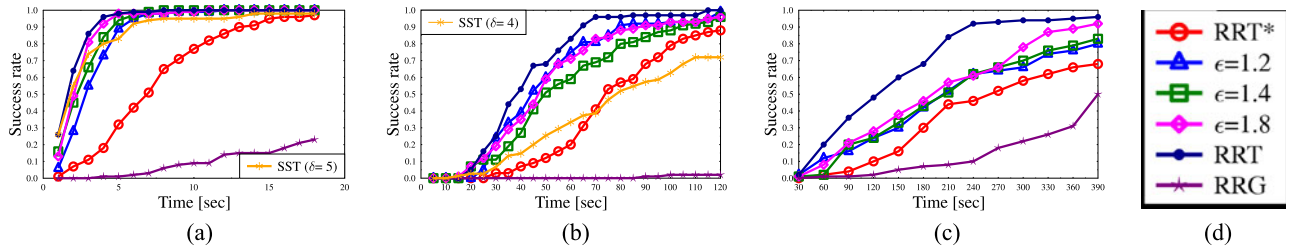

Fig. 3. Success rate for algorithms on different scenarios. (a) Maze scenario. (b) Alternating Barriers scenario. (c) Cubicles scenario. (d) Legend.

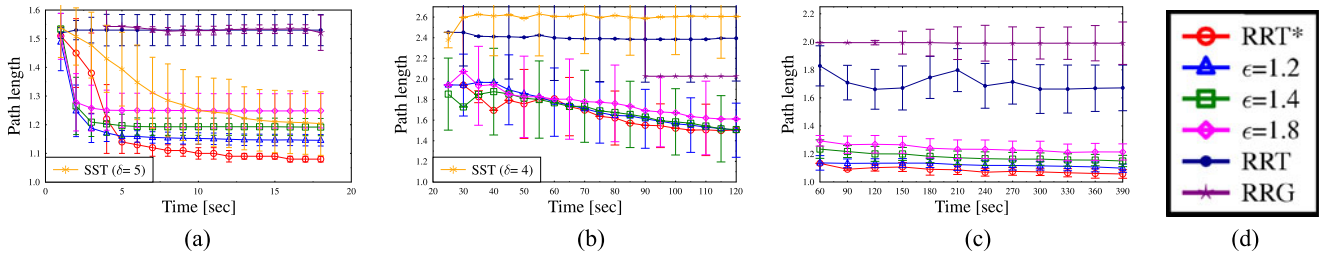

Fig. 4. Path lengths for algorithms on different scenarios. Length values are normalized such that a length of one represents the length of an optimal path. Error bars denote the standard deviation of path length. To avoid cluttering the graphs, the error bars are only depicted for a sample of the graph points. (a) Maze scenario. (b) Alternating Barriers scenario. (c) Cubicles scenario. (d) Legend.

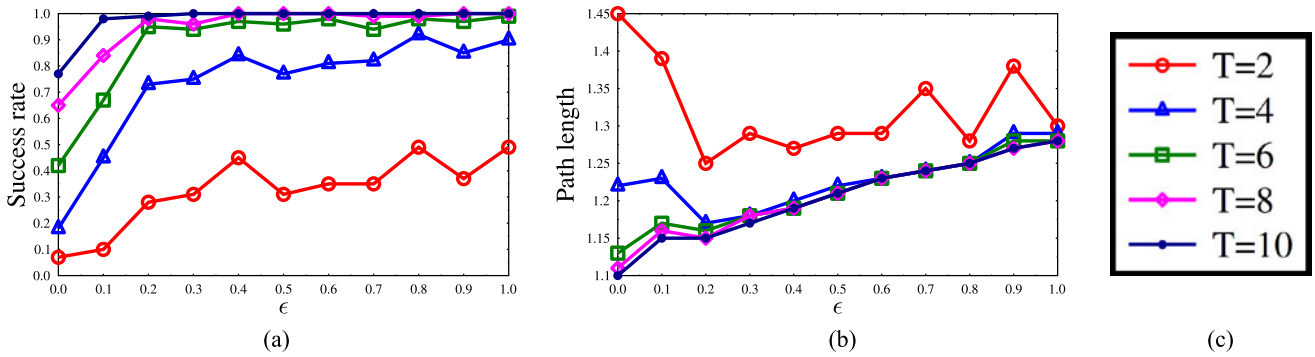

Fig. 5. Average success rate and smoothed path length as functions of the approximation factor for the Maze Scenario. Time values are in seconds. (a) Success rate. (b) Path length. (c) Legend.

are omitted in this text. For supplementary material, the reader is referred to http://acg.cs.tau.ac.il/projects/LBT-RRT.

Choosing the approximation factor: Choosing the best value of the approximation factor ε is both application and scenario dependent. However, we present an analysis on the effect of choosing different values of ε. We report both success rate [see Fig. 5(a)] and path length [see Fig. 5(b)] as a function of the approximation factor, for given planning times for the Maze scenario. This was done by sampling the approximation factor at intervals of 0.1 between 0 and 1.

As noted, the success rate of finding a solution increases monotonically with ε. The behavior of the solution's quality as a function of ε is slightly more complex; for low values of ε and small running times, the quality of the solution is poor as the algorithm did not have enough iterations to find high-quality solutions. For high values of ε (and all running times), the quality of the solution is poor as well. This is because the algorithm did not attempt to improve the solution found due to the high appropriation factor.

The value of $\varepsilon = 0.2$ seems to best capture the tradeoff between the success rate and the convergence to high-quality solutions for this specific scenario. The additional gain in success rate seems to be highest when increasing ε from 0.1 to 0.2. The path length seems to be shortest at this value when compared with all values of ε for the initial running time of the algorithm. Even for large running times, the path length obtained by lower values of ε is only marginally smaller.

V. Lazy Goal-Biased Lower Bound Tree-RRT

In this section, we show to further reduce the number of calls to the local planner by incorporating a lazy approach together with a goal bias.

LBT-RRT maintains the bounded approximation invariant to *every* node. This is desirable in settings in which a high-quality path to every point in the configuration space is required. However, when only a high-quality path to the goal is needed, this may lead to unnecessary time-consuming calls to the local planner.

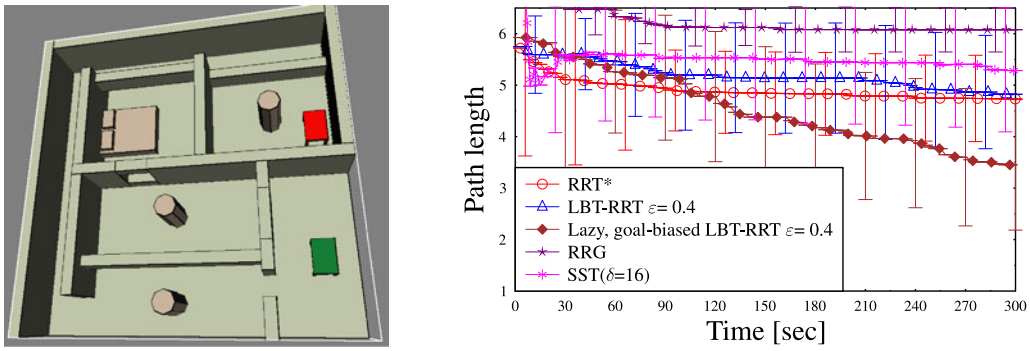

Fig. 6. Simulation results comparing lazy goal-biased LBT-RRT with LBT-RRT, SST, RRT*, and RRG. (a) Home scenario (provided by the OMPL distribution). Start and target table-shaped robots are depicted in green and red, respectively. (b) Path lengths as a function of computation time. Length values are normalized such that a length of one represents the length of an optimal path. Error bars denote the standard deviation of path length.

Therefore, we suggest the following variant of LBT-RRT in which we relax the bounded approximation invariant such that it holds only for nodes $x \in X_{\text{goal}}$. This variant is similar to LBT-RRT but differs with respect to the calls to the local planner and with respect to the dynamic shortest-path algorithm used. As we only maintain the bounded approximation invariant to the goal nodes, we do not need to continuously update the (approximate) shortest path to every node in $\mathcal{G}_{\text{lb}}$. We replace the SSSP algorithm, which allows us to compute the shortest paths to *every* node in a dynamic graph, with Lifelong Planning A* (LPA*) [37]. LPA* allows us to repeatedly find shortest paths from a given start to a given goal while allowing for edge insertions and deletions. Similar to A* [38], this is done by using a heuristic function h such that for every node x, $h(x)$ is an estimate of the cost to reach the goal from x.

Given a start vertex x_{init}, a goal region X_{goal}, we will use the following functions, which are provided when implementing LPA*: $\texttt{shortest\_path}_{\text{LPA*}}(G)$ recomputes the shortest path to reach X_{goal} from x_{init} on the graph G and returns the node $x \in X_{\text{goal}}$ such that $x \in X_{\text{goal}}$ and $\texttt{cost}_{\mathcal{G}_{\text{lb}}}(x)$ is minimal among all $x' \in X_{\text{goal}}$. Once the function has been called, the following functions take constant running time: $\texttt{cost}_{\text{LPA*}}(G)$ returns the minimal cost to reach X_{goal} from x_{init} on the graph G and for every node x lying on a shortest path to the goal, $\texttt{parent}_{\text{LPA*}}(G, x)$ returns the predecessor of the node x along this path. Additionally, $\texttt{insert\_edge}_{\text{LPA*}}(G, x, y)$ and $\texttt{delete\_edge}_{\text{LPA*}}(G, x, y)$ insert (delete) the edge (x, y) to (from) the graph G, respectively.

We are now ready to describe lazy goal-biased LBT-RRT, which is similar to LBT-RRT except for the way in which new edges are considered. Instead of the function `consider_edge` called in lines 14 and 16 of Algorithm 5, the function `consider_edge_goal_biased` is called.

`consider_edge_goal_biased` (x_1, x_2), outlined in Algorithm 7, begins by computing the cost to reach the goal in $\mathcal{T}_{\text{apx}}$ (line 1) and in $\mathcal{G}_{\text{lb}}$ after adding the edge (x_1, x_2) lazily to $\mathcal{G}_{\text{lb}}$ (lines 2–5). Namely, the edge is added with no call to the local planner and without checking if the bounded approximation invariant is violated. Note that the *relaxed* bounded approximation invariant is violated (line 6) only if a path to the goal is found. Clearly, if all edges along the shortest path to the goal are found to be collision free, then the invariant holds. Thus, the algorithm attempts to follow the edges along the path (starting at the last edge and backtracking toward x_{init}) one by one and we test to see if they are indeed collision free. If an edge is collision free (line 7), it is inserted to $\mathcal{T}_{\text{apx}}$ (line 8), and a path to the goal in $\mathcal{T}_{\text{apx}}$ is recomputed (line 9). This is repeated as long as the relaxed bounded approximation invariant is violated. If the edge is found to be in collision (line 12), it is removed from $\mathcal{G}_{\text{lb}}$ (line 13) and the process is repeated (line 14).

Algorithm 7: `consider_edge_goal_biased` (x_1, x_2).

1: $c_{\min}^{\text{apx}} \leftarrow \texttt{cost}_{\text{LPA*}}(\mathcal{T}_{\text{apx}})$
2: $\texttt{insert\_edge}_{\text{LPA*}}(\mathcal{G}_{\text{lb}}, x_1, x_2)$
3: $x \leftarrow \texttt{shortest\_path}_{\text{LPA*}}(\mathcal{G}_{\text{lb}})$
4: $x_{\text{parent}} \leftarrow \texttt{parent}_{\text{LPA*}}(\mathcal{G}_{\text{lb}}, x)$
5: $c_{\min}^{\text{lb}} \leftarrow \texttt{cost}_{\text{LPA*}}(\mathcal{G}_{\text{lb}})$
6: **while** $c_{\min}^{\text{apx}} > (1+\varepsilon) \cdot c_{\min}^{\text{lb}}$ **do**
7: **if** $(\texttt{collision\_free}\,(x_{\text{parent}}, x))$ **then**
8: $\texttt{insert\_edge}_{\text{LPA*}}(\mathcal{T}_{\text{apx}}, x_{\text{parent}}, x)$
9: $\texttt{shortest\_path}_{\text{LPA*}}(\mathcal{T}_{\text{apx}})$
10: $c_{\min}^{\text{apx}} \leftarrow \texttt{cost}_{\text{LPA*}}(\mathcal{T}_{\text{apx}})$
11: $x \leftarrow \texttt{parent}_{\text{LPA*}}(x)$
12: **else**
13: $\texttt{delete\_edge}_{\text{LPA*}}\,(\mathcal{G}_{\text{lb}}, x_{\text{parent}}, x)$
14: **GoTo** line 3

Following similar arguments as described in Section III, one can show the correctness of the algorithm. We note that as long as no path has been found, the algorithm performs no more calls to the local planner than RRT. Additionally, it is worth noting that the planner resembles Lazy-RRG* [39].

We compared lazy goal-biased LBT-RRT with LBT-RRT, SST, RRT*, and RRG on the Home scenario [see Fig. 6(a)]. In this scenario, a low-quality solution is typically easy to find and all algorithms (except RRG) find a solution with roughly

the same success rate as RRT (results omitted). Converging to the optimal solution requires longer running times as low-quality paths are easy to find yet high-quality ones pass through narrow passages. Fig. 6(b) depicts the path length obtained by the algorithms as a function of time. The convergence to the optimal solution of RRG is significantly slower than all other algorithms. LBT-RRT, SST, and RRT* all find a low-quality solution (between five and six times longer than the optimal solution) within the allowed time frame and manage to slightly improve upon its cost (with RRT* obtaining slightly shorter solutions than LBT-RRT and SST). When enhancing LBT-RRT with a lazy approach together with goal-biasing, one can observe that the convergence rate improves substantially.

VI. Conclusion and Future Work

In this work, we presented an asymptotically near-optimal motion-planning algorithm. Using an approximation factor allows the algorithm to avoid calling the computationally expensive local planner when no substantially better solution may be obtained. LBT-RRT, together with the lazy goal-biased variant, makes use of *dynamic shortest path algorithms*. This is an active research topic in many communities such as artificial intelligence and communication networks.

Hence, the algorithms we proposed in this work may benefit from any advances made for dynamic shortest path algorithms. For example, D'Andrea *et al.* [40] recently presented an algorithm that allows for dynamically maintaining shortest path trees under *batches* of updates, which can be used by LBT-RRT instead of the SSSP algorithm. In many applications in which either RRT or RRT* were used, we argue that LBT-RRT may serve as a superior alternative with no fundamental modification to the underlying algorithms, which originally use RRT or RRT*. Moreover, one may consider alternative implementations of LBT-RRT using tools developed for either RRT or RRT* that can enhance LBT-RRT. For further discussions on the subject, see the expanded arXiv version of this paper [41].

In [41] we show that the framework presented in this paper for relaxing the optimality of RRG can be used to have a similar effect on another asymptotically optimal sampling-based algorithm, FMT* [22]. It would be interesting to see if other algorithms can benefit from the approach we take in this work.

Looking to further extend our framework, we seek 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 (large ε), 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 [29]). While changing the approximation factor in LBT-RRT may possibly require a massive rewiring of $\mathcal{G}_{\mathrm{lb}}$ (to maintain the bounded approximation invariant), this is not the case in lazy goal-biased LBT-RRT. In this variant of LBT-RRT, the approximation factor can change at any stage of the algorithm without any modifications at all.

An interesting question to be further studied is whether our framework can be applied 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 whether it is collision free. However, for some other measures such as energy consumption, we believe that the framework could be effectively used.

Acknowledgment

The author would like to thank C. Zhong for feedback on the implementation of the algorithm, S. Chechik for advice regarding dynamic shortest path algorithms, and K. Bekris for advice regarding efficient implementation of the SST algorithm and tuning of its parameters.

References

[1] L. E. Kavraki, P. Švestka, 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, Aug. 1996.

[2] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 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*. Cambridge, MA, USA: MIT Press, 2005.

[4] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. 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 *Proc. 9th Int. Workshop Algorithmic Foundations Robot.*, 2010, pp. 285–301.

[6] R. Geraerts and M. H. Overmars, "Creating high-quality paths for motion planning," *Int. J. Robot. 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. Robot.*, vol. 27, no. 2, pp. 365–371, Apr. 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 *Proc. Int. Workshop Algorithmic Foundations Robot.*, 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 *Proc. IEEE Int. Conf. Robot. Autom.*, 2003, pp. 4439–4444.

[10] C. Urmson and R. G. Simmons, "Approaches for heuristically biasing RRT growth," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2003, pp. 1178–1183.

[11] A. C. Shkolnik, M. Walter, and R. Tedrake, "Reachability-guided sampling for planning under differential constraints," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2009, pp. 2859–2865.

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

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

[14] O. Salzman, D. Shaharabani, P. K. Agarwal, and D. Halperin, "Sparsification of motion-planning roadmaps by edge contraction," *Int. J. Robot. Res.*, vol. 33, no. 14, pp. 1711–1725, 2014.

[15] J. D. Marble and K. E. Bekris, "Computing spanners of asymptotically optimal probabilistic roadmaps," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 4292–4298.

[16] J. D. Marble and K. E. Bekris, "Asymptotically near-optimal is good enough for motion planning," presented at the 15th Int. Symp. Robot. Res., Flagstaff, AZ, USA, 2011.

[17] J. D. Marble and K. E. Bekris, "Towards small asymptotically near-optimal roadmaps," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 2557–2562.

[18] A. Dobson and K. E. Bekris, "Sparse roadmap spanners for asymptotically near-optimal motion planning," *Int. J. Robot. Res.*, vol. 33, no. 1, pp. 18–47, 2014.

[19] A. Perez, S. Karaman, A. Shkolnik, E. Frazzoli, S. Teller, and M. Walter, "Asymptotically-optimal path planning for manipulation using incremen-

tal sampling-based algorithms," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 4307–4313.
[20] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan, "RRT*-Smart: Rapid convergence implementation of RRT* towards optimal solution," *Int. J. Adv. Robot. Syst.*, vol. 10, pp. 1–12, 2013.
[21] O. Arslan and P. Tsiotras, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 2413–2420.
[22] L. Janson, E. Schmerling, A. A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *Int. J. Robotic Res.*, vol. 34, no. 7, pp. 883–921, 2015.
[23] J. Luo and K. Hauser, "An empirical study of optimal motion planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2014, pp. 1761–1768.
[24] O. Salzman and D. Halperin, "Asymptotically-optimal motion planning using lower bounds on cost," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 4167–4172.
[25] M. Kleinbort, O. Salzman, and D. Halperin, "Efficient high-quality motion planning by fast all-pairs *r*-nearest-neighbors," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 2985–2990.
[26] Z. Littlefield, Y. Li, and K. E. Bekris, "Efficient sampling-based motion planning with asymptotic near-optimality guarantees for systems with dynamics," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 1779–1785.
[27] Y. Li, Z. Littlefield, and K. E. Bekris, "Sparse methods for efficient asymptotically optimal kinodynamic planning," in *Proc. Int. Workshop Algorithmic Foundations Robot.*, 2014, pp. 263–282.
[28] D. Ferguson and A. Stentz, "Anytime RRTs," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2006, pp. 5369–5375.
[29] R. Alterovitz, S. Patil, and A. Derbakova, "Rapidly-exploring roadmaps: Weighing exploration vs. refinement in optimal motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 3706–3712.
[30] R. Luna, I. A. Şucan, M. Moll, and L. E. Kavraki, "Anytime solution optimization for sampling-based motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 5053–5059.
[31] S. Karaman, M. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 1478–1483.
[32] O. Salzman and D. Halperin, "Asymptotically near-optimal RRT for fast, high-quality, motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 4680–4685.
[33] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni, "Fully dynamic algorithms for maintaining shortest paths trees," *J. Algorithms*, vol. 34, no. 2, pp. 251–281, 2000.
[34] G. Ramalingam and T. W. Reps, "On the computational complexity of dynamic graph problems," *Theor. Comput. Sci.*, vol. 158, nos. 1/2, pp. 233–277, 1996.
[35] D. J. Webb and J. van den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 5054–5061.
[36] I. A. Şucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, Dec. 2012.
[37] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A," *Artif. Intell.*, vol. 155, no. 1, pp. 93–146, 2004.
[38] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, MA, USA: Addison-Wesley, 1984.
[39] K. Hauser, "Lazy collision checking in asymptotically-optimal motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, 2951–2957.
[40] A. D'Andrea, M. D'Emidio, D. Frigioni, S. Leucci, and G. Proietti, "Dynamically maintaining shortest path trees under batches of updates," in *Structural Information and Communication Complexity* (ser. Lecture Notes in Computer Science), vol. 8179, T. Moscibroda and A. Rescigno, Eds. New York, NY, USA: Springer, 2013, pp. 286–297.
[41] O. Salzman and D. Halperin, "Asymptotically near-optimal RRT for fast, high-quality, motion planning," *CoRR*, vol. abs/1308.0189, 2013.

Oren Salzman received the B.Sc. degree (honors) from Technion-Israel Institute of Technology, Haifa, Israel, and the M.Sc. degree (honors) from Tel Aviv University, Tel Aviv, Israel. He is working toward the Ph.D. degree with the School of Computer Science, Tel Aviv University, under the supervision of Prof. Dan Halperin.

Dan Halperin received the B.Sc. degree in mathematics and computer science, and the M.Sc. and Ph.D. degrees in computer science, all from Tel Aviv University, Tel Aviv, Israel.

After receiving the Ph.D. degree, he spent three years at the Computer Science Robotics Laboratory, Stanford University, Stanford, CA, USA. He is currently a Professor of Computer Science with Tel Aviv University. His main areas of research are computational geometry and its applications, robust geometric computing, robotics and automation. He is cofounder of the international workshop series entitled Algorithmic Foundations of Robotics (WAFR).