RRT$^{\mathrm{X}}$: Real-Time Motion Planning/Replanning for Environments with Unpredictable Obstacles

Michael Otte and Emilio Frazzoli

Massachusetts Institute of Technology, Cambridge MA 02139, USA
ottemw@mit.edu

Abstract. We present RRT$^{\mathrm{X}}$, the first asymptotically optimal sampling-based motion planning algorithm for real-time navigation in dynamic environments (containing obstacles that unpredictably appear, disappear, and move). Whenever obstacle changes are observed, e.g., by onboard sensors, a graph rewiring *cascade* quickly updates the search-graph and repairs its shortest-path-to-goal subtree. Both graph and tree are built directly in the robot's state space, respect the kinematics of the robot, and continue to improve during navigation. RRT$^{\mathrm{X}}$ is also competitive in static environments—where it has the same amortized per iteration runtime as RRT and RRT* $\Theta\left(\log n\right)$ and is faster than RRT$^{\#}$ $\omega\left(\log^2 n\right)$. In order to achieve $O\left(\log n\right)$ iteration time, each node maintains a set of $O\left(\log n\right)$ expected neighbors, and the search graph maintains ϵ-consistency for a predefined ϵ.

Keywords: real-time, asymptotically optimal, graph consistency, motion planning, replanning, dynamic environments, shortest-path

1 Introduction

Replanning algorithms find a motion plan and then repair that plan on-the-fly if/when changes to the obstacle set are detected during navigation. We present RRT$^{\mathrm{X}}$, the first asymptotically optimal sampling-based replanning algorithm. RRT$^{\mathrm{X}}$ enables real-time kinodynamic navigation in dynamic environments, i.e., in environments with obstacles that *unpredictably* appear, move, and vanish. RRT$^{\mathrm{X}}$ refines, updates, and remodels a single graph and its shortest-path subtree over the entire duration of navigation. Both graph and subtree exist in the robot's state space, and the tree is rooted at the goal state (allowing it to remain valid as the robot's state changes during navigation). Whenever obstacle changes are detected, e.g., via the robot's sensors, *rewiring* operations *cascade* down the affected branches of the tree in order to repair the graph and remodel the shortest-path tree.

Although RRT$^{\mathrm{X}}$ is designed for dynamic environments, it is also competitive in static environments—where it is asymptotically optimal and has an expected amortized per iteration runtime of $\Theta\left(\log n\right)$ for graphs with n nodes. This is similar to RRT and RRT* $\Theta\left(\log n\right)$ and faster than RRT$^{\#}$ $\Theta\left(\log^2 n\right)$.

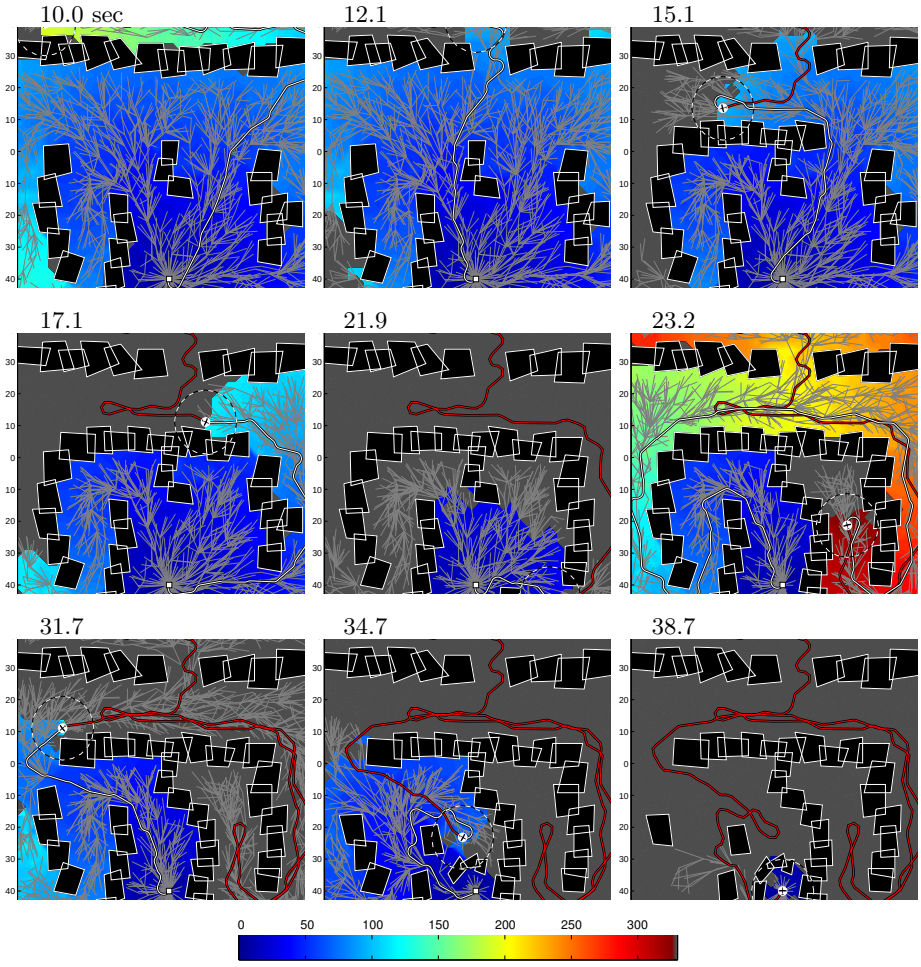

Fig. 1: Dubins robot (white circle) using RRT$^{\mathrm{X}}$ to move from start to goal (white square) while repairing its shortest-path tree (light-gray) vs. obstacle changes. Color is cost-to-goal. Planned/executed paths are white/red. Obstacles are black with white outlines. Time (in seconds) appears above each sub-figure. Tree *edges* are drawn (not Dubins trajectories). See http://tinyurl.com/l53gzgd for video.

The expected $\Theta(\log n)$ time is achieved, despite rewiring cascades, by using two new graph rewiring strategies: (1) Rewiring cascades are aborted once the graph becomes ϵ-consistent[1], for a predefined $\epsilon > 0$. (2) Graph connectivity information is maintained in local neighbor sets stored at each node, and the usual

[1] "ϵ-consistency" means that the cost-to-goal stored at each node is within ϵ of its look-ahead cost-to-goal, where the latter is the minimum sum of distance-to-neighbor plus neighbor's cost-to-goal.

edge symmetry is allowed to be broken, i.e., the directed edge (u, w) will eventually be forgotten by u but not by w or *vice versa*. In particular, node v always remembers the original neighbors that were calculated upon its insertion into the search-graph. However, each of those original neighbors will forget its connection to v once it is no longer within an RRT*-like shrinking D-ball centered at v (with the exception that connections within the shortest-path subtree are also remembered). This guarantees: (A) each node maintains expected $O(\log n)$ neighbors, (B) the RRT* solution is always a realizable sub-graph of the RRT$^{\mathrm{X}}$ graph—providing an upper-bound on path length, (C) all "edges" are remembered by at least one node. Although (1) and (2) have obvious side-effects[2], they significantly decrease reaction time (i.e, iteration time vs. RRT$^{\#}$ and cost propagation time vs. RRT*) without hindering asymptotic convergence to the optimal solution.

A YouTube play-list of RRT$^{\mathrm{X}}$ movies at http://tinyurl.com/l53gzgd shows RRT$^{\mathrm{X}}$ solving a variety of motion problems in different spaces [13].

1.1 Related Work

In general, RRT$^{\mathrm{X}}$ differs from previous work in that it is the first asymptotically optimal sampling-based replanning[3] algorithm.

Previous sampling-based replanning algorithms (e.g., ERRT [2], DRRT [3], multipartite RRT [19], LRF [4]) are concerned with finding a *feasible* path. Previous methods also delete nodes/edges whenever they are invalidated by dynamic obstacles (detached subtrees/nodes/edges not in collision may be checked for future reconnection). Besides the fact that RRT$^{\mathrm{X}}$ is a shortest-path planning algorithm, it also rewires the shortest-path subtree to *temporarily* exclude edges/nodes that are currently in collision (if the edges/nodes cease to be in collision, then RRT$^{\mathrm{X}}$ rewires them back into the shortest-path subtree).

RRT$^{\#}$ [1] is the only other sampling-based algorithm that uses a rewiring cascade; in particular, after the cost-to-goal of an old node is decreased by the addition of a new node. RRT$^{\#}$ is designed for static environments (obstacle appearances, in particular, break the algorithm). In Section 3 we prove that in static environments the asymptotic expected runtime to build a graph with n nodes is $\Theta(n \log n)$ for RRT$^{\mathrm{X}}$ and $\omega\left(n \log^2 n\right)$ for RRT$^{\#}$.

PRM [6] is the first asymptotically optimal sampling-based motion planning algorithm. PRM*/RRT* [5] are the first with $\Theta(\log n)$ expected per iteration time. PRM/PRM*/RRT* assume a static environment, and RRT* uses "Lazy-propagation" to spread information through an inconsistent graph (i.e., data is transferred only via new node insertions).

D* [17], Lifelong-A* [7], and D*-Lite [8] are discrete graph replanning algorithms designed to repair an A*-like solution after edge weights have changed.

---

[2] (1) Allows graph inconsistency. (2) Prevents the practical realization of some paths.

[3] Replanning algorithms find a sequence of solutions to the *same* goal state "on-the-fly" vs. an evolving obstacle configuration and start state, and are distinct from multi-query algorithms (e.g., PRM [6]) and single-query algorithms (e.g., RRT [10]).

These algorithms traditionally plan/replan over a grid embedded in the robot's *workspace*, and thus find geometric paths that are suboptimal with respect to the robot's state space—and potentially impossible to follow given its kinematics.

Any-Time SPRT [14] is an asymptotically optimal sampling-based motion planning algorithm that maintains a consistent graph; however, it assumes a static environment and requires $O(n)$ time per iteration.

LBT-RRT [16] is designed for static environments and maintains a "lower-bound" graph that returns asymptotically $1+\hat{\epsilon}$ "near-optimal" solutions. Note that tuning $\hat{\epsilon}$ changes the performance of LBT-RRT along the spectrum between RRT and RRT*, while tuning ϵ changes the graph consistency of RRT$^{\mathrm{X}}$ along the spectrum between that of RRT* and RRT$^{\#}$ (i.e., in static environments).

Recent work [12] prunes sampling-based roadmaps down to a sparse subgraph *spanner* that maintains near-optimality while using significantly fewer nodes. This is similar, in spirit, to how RRT$^{\mathrm{X}}$ limits each nodes neighbor set to $O(\log n)$.

Feedback planners generate a continuous control policy over the state space (i.e. instead of embedding a graph in the state space). Most feedback planners do not consider obstacles [18, 15], while those that do [11] assume that obstacles are both static and easily representable in the state space (sampling-based motion planning algorithms do not).

1.2 Preliminaries

Let $\mathcal{X}$ denote the robot's D-dimensional state space. $\mathcal{X}$ is a measurable metric space that has finite measure. Formally, $\mathscr{L}(\mathcal{X}) = c$, for some $c < \infty$ and $\mathscr{L}(\cdot)$ is the Lebesgue measure; assuming $\mathrm{d}(x_1, x_2)$ is a distance function on $\mathcal{X}$, then $\mathrm{d}(x_1, x_2) \geq 0$ and $\mathrm{d}(x_1, x_3) \leq \mathrm{d}(x_1, x_2) + \mathrm{d}(x_2, x_3)$ and $\mathrm{d}(x_1, x_2) = \mathrm{d}(x_2, x_1)$ for all $x_1, x_2, x_3 \in \mathcal{X}$. We assume the boundary of $\mathcal{X}$ is both locally Lipschitz-continuous and has finite measure. The obstacle space $\mathcal{X}_{\mathrm{obs}} \subset \mathcal{X}$ is the open subset of $\mathcal{X}$ in which the robot is "in collision" with obstacles or itself. The free space $\mathcal{X}_{free} = \mathcal{X} \setminus \mathcal{X}_{\mathrm{obs}}$ is the closed subset of $\mathcal{X}$ that the robot can reach. We assume $\mathcal{X}_{\mathrm{obs}}$ is defined by a set $\mathcal{O}$ of a finite number of obstacles O, each with a boundary that is both locally Lipschitz-continuous and has finite measure.

The robot's start and goal states are x_{start} and x_{goal}, respectively. At time t the location of the robot is $x_{\mathrm{bot}}(t)$, where $x_{\mathrm{bot}} : [t_0, t_{\mathrm{cur}}] \to \mathcal{X}$ is the traversed path of the robot from the start time t_0 to the current time t_{cur}, and is undefined for $t > t_{\mathrm{cur}}$. The obstacle space (and free space) may change as a function of time and/or robot location, i.e. $\Delta\mathcal{X}_{\mathrm{obs}} = f(t, x_{\mathrm{bot}})$. For example, if there are unpredictably moving obstacles, inaccuracies in *a priori* belief of $\mathcal{X}_{\mathrm{obs}}$, and/or a subset of $\mathcal{X}_{free}$ must be "discovered" via the robot's sensors.

A movement trajectory $\pi(x_1, x_2)$ is the curve defined by a continuous mapping $\pi : [0, 1] \to \mathcal{X}$ such that $0 \mapsto x_1$ and $1 \mapsto x_2$. A trajectory is *valid* iff both $\pi(x_1, x_2) \cap \mathcal{X}_{\mathrm{obs}} = \emptyset$ and it is possible for the robot to follow $\pi(x_1, x_2)$ given its kinodynamic and other constraints. $\mathrm{d}_\pi(x_1, x_2)$ is the length of $\pi(x_1, x_2)$.

1.3 Environments: Static vs. Dynamic and Related Assumptions

A *static* environment has an obstacle set that changes deterministically vs. t and x_{bot}, i.e., $\Delta\mathcal{X}_{\text{obs}} = f(t, x_{\text{bot}})$ for f known *a priori*. In the simplest case, $\Delta\mathcal{X}_{\text{obs}} \equiv \emptyset$. In contrast, a *dynamic*[4] environment has an unpredictably changing obstacle set, i.e., f is a "black-box" that cannot be known *a priori*. The assumption of incomplete prior knowledge of $\Delta\mathcal{X}_{\text{obs}}$ guarantees myopia; this assumption is the defining characteristic of replanning algorithms, in general. While nothing prevents us from estimating $\Delta\mathcal{X}_{\text{obs}}$ based on prior data and/or online observations, we cannot guarantee that any such estimate will be correct. Note that $\Delta\mathcal{X}_{\text{obs}} \neq \emptyset$ is not a sufficient condition for $\mathcal{X}$ to be dynamic[5].

1.4 Problem Statement of "Shortest-Path Replanning"

Given $\mathcal{X}$, $\mathcal{X}_{\text{obs}}$, x_{goal}, $x_{\text{bot}}(0) = x_{\text{start}}$, and unknown $\Delta\mathcal{X}_{\text{obs}} = f(t, x_{\text{bot}})$, find $\pi^*(x_{\text{bot}}, x_{\text{goal}})$ and, until $x_{\text{bot}}(t) = x_{\text{goal}}$, simultaneously update $x_{\text{bot}}(t)$ along $\pi^*(x_{\text{bot}}, x_{\text{goal}})$ while recalculating $\pi^*(x_{\text{bot}}, x_{\text{goal}})$ whenever $\Delta\mathcal{X}_{\text{obs}} \neq \emptyset$, where

$$\pi^*(x_{\text{bot}}, x_{\text{goal}}) = \underset{\pi(x_{\text{bot}}, x_{\text{goal}}) \in \mathcal{X}_{free}}{\arg\min} \mathrm{d}_\pi(x_{\text{bot}}, x_{\text{goal}})$$

1.5 Additional Notation Used for the Algorithm and its Analysis

RRT$^{\text{X}}$ constructs a graph $\mathcal{G} := (V, E)$ embedded in $\mathcal{X}$, where V is the node set and E is the edge set. With a slight abuse of notation we will allow $v \in V$ to be used in place of v's corresponding state $x \in \mathcal{X}$, e.g., as a direct input into distance functions. Thus, the robot starts at v_{start} and goes to v_{goal}. The "shortest-path" subtree of $\mathcal{G}$ is $\mathcal{T} := (V_\mathcal{T}, E_\mathcal{T})$, where $\mathcal{T}$ is rooted at v_{goal}, $V_\mathcal{T} \subset V$, and $E_\mathcal{T} \subset E$. The set of 'orphan nodes' is defined $V_\mathcal{T}^{\text{c}} = V \setminus V_\mathcal{T}$ and contains all nodes that have become disconnected from $\mathcal{T}$ due to $\Delta\mathcal{X}_{\text{obs}}$ (c denotes the set compliment of $V_\mathcal{T}$ with respect to V and *not* the set of nodes in the compliment graph of $\mathcal{T}$).

$\mathcal{G}$ is built, in part, by drawing nodes at random from a random sample sequence $S = \{v_1, v_2, \ldots\}$. We assume v_i is drawn i.i.d from a uniform distribution over $\mathcal{X}$; however, this can be relaxed to any distribution with a finite probability density for all $v \in \mathcal{X}_{free}$. We use V_n, $\mathcal{G}_n$, $\mathcal{T}_n$ to denote the node set, graph, and tree when the node set contains n nodes, e.g., $\mathcal{G}_n = \mathcal{G}$ s.t. $|V| = n$. Note $m_i = |V_{m_i}|$ at iteration i, but $m_i \neq i$ in general because samples may not always be connectable to $\mathcal{G}$. Indexing on m (and not i) simplifies the analysis.

$\mathbb{E}_n(\cdot)$ denotes the expected value of '$\cdot$' over the set $\mathcal{S}$ of all such sample sequences, conditioned on the event that $n = |V|$. The expectation $\mathbb{E}_{n,v_x}(\cdot)$ is conditioned on both $n = |V|$ and $V_n \setminus V_{n-1} = \{v_x\}$ for v_x at a particular $x \in \mathcal{X}$.

---

[4]The use of the term "dynamic" to indicate that an environment is "unpredictably changing" comes from the artificial intelligence literature. It should not be confused with the "dynamics" of classical mechanics.

[5]For example, if $\mathcal{X} \subset \mathbb{R}^d$ space, $\mathbb{T}$ is time, and obstacle movement is known *a priori*, obstacles are stationary with respect to $\hat{\mathcal{X}} \subset \left(\mathbb{R}^d \times \mathbb{T}\right)$ space-time.

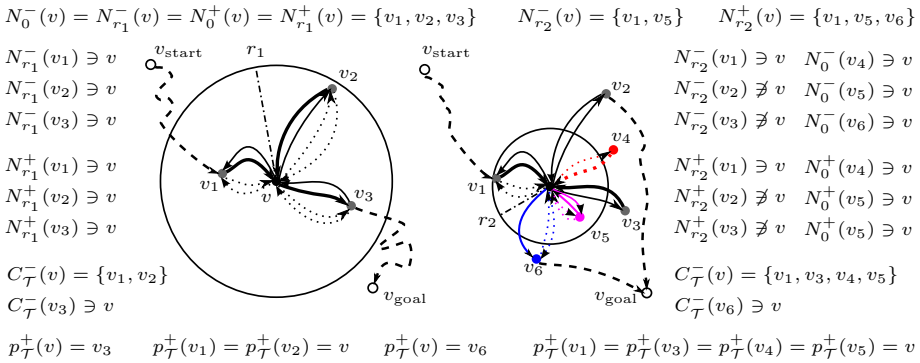

Fig. 2: Neighbor sets of/with node v. Left: v is inserted when $r = r_1$. Right: later $r = r_2 < r_1$. Solid: neighbors known to v. Black solid: original in- $N_0^-(v)$ and out-neighbors $N_0^+(v)$ of v. Colored solid: running in- $N_r^-(v)$ and out-neighbors $N_r^+(v)$ of v. Dotted: v is neighbor of $v_i \neq v$. Dotted colored: v is an original neighbor of v_i. Bold: edge in shortest-path subtree. Dashed: other sub-path.

RRT$^{\text{X}}$ uses a number of neighbor sets for each node v, see Figure 2. Edges are directed $(u, v) \neq (v, u)$, and we use a superscript '$-$' and '$+$' to denote association with incoming and outgoing edges, respectively. 'Incoming neighbors' of v is the set $N^-(v)$ s.t. v knows about (u, v). 'Outgoing neighbors' of v is the set $N^+(v)$ s.t. v knows about (v, u). At any instant $N^+(v) = N_0^+(v) \cup N_r^+(v)$ and $N^-(v) = N_0^-(v) \cup N_r^-(v)$, where $N_0^-(v)$ and $N_0^+(v)$ are the original PRM*-like in/out-neighbors (which v always keeps), and $N_r^-(v)$ and $N_r^+(v)$ are the 'running' in/out-neighbors (which v culls as r decreases). The set of all neighbors of v is $N(v) = N^+(v) \cup N^-(v)$. Because $\mathcal{T}$ is rooted at v_{goal}, the parent of v is denoted $p_{\mathcal{T}}^+(v)$ and the child set of v is denoted $C_{\mathcal{T}}^-(v)$.

$\mathtt{g}(v)$ is the (ϵ-consistent) cost-to-goal of reaching v_{goal} from v through $\mathcal{T}$. The look-ahead estimate of cost-to-goal is $\mathtt{lmc}(v)$. Note that the algorithm stores both $\mathtt{g}(v)$ and $\mathtt{lmc}(v)$ at each node, and updates $\mathtt{lmc}(v) \leftarrow \min_{u \in N^+(v)} \mathrm{d}_\pi(v, u) + \mathtt{lmc}(u)$ when appropriate conditions have been met. v is 'ϵ-consistent' iff $\mathtt{g}(v) - \mathtt{lmc}(v) < \epsilon$. $\mathrm{g}_m(v)$ is the cost-to-goal of v given $\mathcal{T}_m$. Recall that $\pi_{\mathcal{X}}^*(v, v_{\text{goal}})$ is the optimal path from v to v_{goal} through $\mathcal{X}$; the length of $\pi_{\mathcal{X}}^*(v, v_{\text{goal}})$ is $\mathrm{g}^*(v)$.

Q is the priority queue that is used to determine the order in which nodes become ϵ-consistent during the rewiring cascades. The key that is used for Q is the ordered pair $\big(\min(\mathtt{g}(v), \mathtt{lmc}(v)), \mathtt{g}(v)\big)$ nodes with smaller keys are popped from Q before nodes with larger keys, where $(a, b) < (c, d)$ iff $a < c \vee (a = c \wedge b < d)$.

2 The RRT$^{\text{X}}$ Algorithm

RRT$^{\text{X}}$ appears in Algorithm 1 and its major subroutines in Algorithms 2-6 (minor subroutines appear on the last page). The main control loop, lines 3-17, terminates once the robot reaches the goal state. Each pass begins by updating the RRT*-like neighborhood radius r (line 4), and then accounting for

Algorithm 1: $\texttt{RRT}^{\texttt{X}}(\mathcal{X}, S)$

```
1  V ← {v_goal}
2  v_bot ← v_start
3  while v_bot ≠ v_goal do
4      r ← shrinkingBallRadius()
5      if obstacles have changed then
6          updateObstacles()
7      if robot is moving then
8          v_bot ← updateRobot(v_bot)
9      v ← randomNode(S)
10     v_nearest ← nearest(v)
11     if d(v, v_nearest) > δ then
12         v ← saturate(v, v_nearest)
13     if v ∉ X_obs then
14         extend(v, r)
15     if v ∈ V then
16         rewireNeighbors(v)
17         reduceInconsistency()
```

Algorithm 2: $\texttt{extend}(v, r)$

```
1  V_near ← near(v, r)
2  findParent(v, V_near)
3  if p_T^+(v) = ∅ then
4      return
5  V ← V ∪ {v}
6  C_T^-(p_T^+(v)) ← C_T^-(p_T^+(v)) ∪ {v}
7  forall u ∈ V_near do
8      if π(v, u) ∩ X_obs = ∅ then
9          N_0^+(v) ← N_0^+(v) ∪ {u}
10         N_r^-(u) ← N_r^-(u) ∪ {v}
11     if π(u, v) ∩ X_obs = ∅ then
12         N_r^+(u) ← N_r^+(u) ∪ {v}
13         N_0^-(v) ← N_0^-(v) ∪ {u}
```

Algorithm 3: $\texttt{cullNeighbors}(v, r)$

```
1  forall u ∈ N_r^+(v) do
2      if r < d_π(v, u) and p_T^+(v) ≠ u then
3          N_r^+(v) ← N_r^+(v) \ {u}
4          N_r^-(u) ← N_r^-(u) \ {v}
```

Algorithm 4: $\texttt{rewireNeighbors}(v)$

```
1  if g(v) − lmc(v) > ε then
2      cullNeighbors(v, r)
3      forall u ∈ N^-(v) \ {p_T^+(v)} do
4          if lmc(u) > d_π(u, v) + lmc(v) then
5              lmc(u) ← d_π(u, v) + lmc(v)
6              makeParentOf(v, u)
7              if g(u) − lmc(u) > ε then
8                  verrifyQueue(u)
```

Algorithm 5: $\texttt{reduceInconsistency}()$

```
1  while size(Q) > 0 and
     (keyLess(top(Q), v_bot) or lmc(v_bot) ≠ g(v_bot)
       or g(v_bot) = ∞ or Q ∋ v_bot) do
2      v ← pop(Q)
3      if g(v) − lmc(v) > ε then
4          updateLMC(v)
5          rewireNeighbors(v)
6      g(v) ← lmc(v)
```

Algorithm 6: $\texttt{findParent}(v, U)$

```
1  forall u ∈ U do
2      π(v, u) ← computeTrajectory(X, v, u)
3      if d_π(v, u) ≤ r and
       lmc(v) > d_π(v, u) + lmc(u) and π(v, u) ≠ ∅
       and X_obs ∩ π(v, u) = ∅ then
4          p_T^+(v) ← u
5          lmc(v) ← d_π(v, u) + lmc(u)
```

obstacle and/or robot changes (lines 5-8). "Standard" sampling-based motion planning operations improve and refine the graph by drawing new samples and then connecting them to the graph if possible (lines 9-14). RRT*-like graph rewiring (line 16) guarantees asymptotic optimality, while rewiring cascades enforce ϵ-consistency (line 17, and also on line 6 as part of `updateObstacles`()). `saturate`(v, v_{nearest}), line 12, repositions v to be δ away from v_{nearest}.

`extend`(v, r) attempts to insert node v into $\mathcal{G}$ and $\mathcal{T}$ (line 5). If a connection is possible then v is added to its parent's child set (line 6). The edge sets of v and its neighbors are updated (lines 7-13). For each new neighbor u of v, u is added to v's initial neighbors sets $N_0^+(v)$ and $N_0^-(v)$, while v is added to u's running neighbor sets $N_r^+(u)$ and $N_r^-(u)$. This Differentiation allows RRT$^{\text{X}}$ to maintain $O(\log n)$ edges at each node, while ensuring $\mathcal{T}$ is no worse than the tree created by RRT* given the same sample sequence.

`cullNeighbors`(v, r) updates $N_r^-(v)$, and $N_r^+(v)$ to allow only edges that are shorter than r—with the exceptions that we do not remove edges that are part of $\mathcal{T}$. RRT^{X} inherits asymptotic optimality and probabilistic completeness from PRM*/RRT* by never culling $N_0^-(v)$ or $N_0^+(v)$.

`rewireNeighbors`(v) rewires v's in-neighbors $u \in N^-(v)$ to use v as their parent, if doing so results in a better cost-to-goal at u (lines 3-6). This rewiring is similar to RRT*'s rewiring, except that here we verify that ϵ-inconsistent neighbors are in the priority queue (lines 7-8) in order to set off a rewiring cascade during the next call to `reduceInconsistency`().

`reduceInconsistency`() manages the rewiring cascade that propagates cost-to-goal information and maintains ϵ-consistency in $\mathcal{G}$ (at least up to the level-set of `lmc`$(\cdot)$ containing v_{bot}). It is similar to its namesake from RRT$^{\#}$, except that in RRT$^{\mathrm{X}}$ the cascade only continues through v's neighbors if v is ϵ-inconsistent (lines 3-5). This is one reason why RRT$^{\mathrm{X}}$ is faster than RRT$^{\#}$. Note that v is always made locally 0-consistent (line 6).

`updateLMC`(v) updates `lmc`(v) based on v's out-neighbors $N^+(v)$ (as in RRT$^{\#}$). `findParent`(v, U) finds the best parent for v from the node set U.
`propogateDescendants`() performs a cost-to-goal *increase* cascade leaf-ward through $\mathcal{T}$ after an obstacle has been added; the cascade starts at nodes with edge trajectories made invalid by the obstacle and is necessary to ensure that the *decrease* cascade in `reduceInconsistency`() reaches all relevant portions of $\mathcal{T}$ (as in D*). `updateObstacles`() updates $\mathcal{G}$ given $\Delta\mathcal{X}_{\mathrm{obs}}$; affected by nodes are added to Q or $V_{\mathcal{T}}^{\mathrm{c}}$, respectively, and then `reduceInconsistency`() and/or `propogateDescendants`() are called to invoke rewiring cascade(s).

3 Runtime Analysis of RRT, RRT*, RRT$^{\mathrm{X}}$, and RRT$^{\#}$

In 3.1-3.3 we prove bounds on the time required by RRT, RRT*, RRT$^{\mathrm{X}}$, and RRT$^{\#}$ to build a search-graph containing n nodes in the static case $\Delta\mathcal{X}_{\mathrm{obs}} = \emptyset$. In 3.4 we discuss the extra operations required by RRT$^{\mathrm{X}}$ when $\Delta\mathcal{X}_{\mathrm{obs}} \neq \emptyset$.

Lemma 1. $\sum_{j=1}^{n} \log j = \Theta(n \log n)$.

Proof. $\log(j+1) - \log j \leq 1$ for all $j \geq 1$. Therefore, by construction:
$-n + \int_1^n \log x \, dx \leq \sum_{j=1}^{n} \log j \leq n + \int_1^n \log x \, dx$ for all $n \geq 1$. Calculus gives:
$-n + \frac{1-n}{\ln 2} + n \log n \leq \sum_{j=1}^{n} \log j \leq n + \frac{1-n}{\ln 2} + n \log n$ □

Slight modifications to the proof of Lemma 1 yield the following corollaries:

Corollary 1. $\sum_{j=1}^{n} \frac{\log j}{j} = \Theta\left(\log^2 n\right)$.

Corollary 2. $\sum_{j=1}^{n} \log^2 j = \Theta\left(n \log^2 n\right)$.

Let f_i^{RRT} and f_i^{RRT*} denote the runtime of the ith iteration of RRT and RRT*, respectively, assuming samples are drawn uniformly at random from $\mathcal{X}$ according to the sequence $S = \{v_1, v_2, \ldots\}$. Let $f^{RRT}(n)$, $f^{RRT*}(n)$, and $f^{RRT^X}(n)$ denote the *cumulative* time until $n = |V_n|$, i.e., the graph contains n nodes, using RRT, RRT*, and RRT$^{\mathrm{X}}$, respectively.

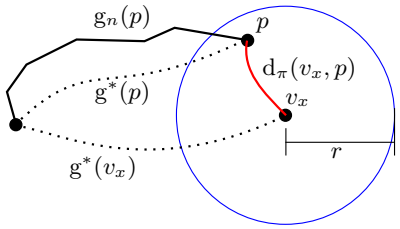

Fig. 3: Node v_x at x is inserted when $n = |V_n|$. The parent of v_x is p. $d_\pi(v_x, p)$ is the distance from v_x to p along the (red) trajectory. $g_n(v_x)$ and $g_n(p)$ are the cost-to-goals of v_x and p when $n = |V_n|$, while $g^*(v_x)$ and $g^*(p)$ are their optimal cost-to-goals, respectively. The neighbor ball (blue) has radius r. Obstacles are not drawn.

3.1 Expected Time until $|V| = n$ for RRT and RRT*

[10] and [5] give the following propositions, respectively:

Proposition 1. *$f_i^{RRT} = \Theta(\log m_i)$ for $i \geq 0$, where $m_i = |V_{m_i}|$ at iteration i.*

Proposition 2. *$\mathbb{E}\left(f_i^{RRT*}\right) = \Theta\left(f_i^{RRT}\right)$ for all $i \geq 0$.*

The dominating term in both RRT and RRT* is due to a nearest neighbor search. The following corollaries are straightforward to prove given Lemma 1 and Propositions 1 and 2 (proofs are omitted here due to space limitations).

Corollary 3. *$\mathbb{E}\left(f^{RRT}(n)\right) = \Theta(n \log n)$.*

Corollary 4. *$\mathbb{E}\left(f^{RRT*}(n)\right) = \Theta(n \log n)$.*

3.2 Expected Amortized Time until $|V| = n$ for RRT$^{\mathrm{X}}$ (Static $\mathcal{X}$)

In this section we prove: $\mathbb{E}_n\left(f^{RRT^X}(n)\right) = \Theta(n \log n)$. The proof involves a comparison to RRT* and proceeds in the following three steps:

1. RRT* cost-to-goal values approach optimality, in the limit as $n \to \infty$.
2. For RRT*, the summed total difference (i.e., over all nodes) between initial and optimal cost-to-goal values is $O(\epsilon n)$; thus, when $n = |V_n|$, RRT$^{\mathrm{X}}$ will have performed at most $O(n)$ cost propagations of size ϵ given the same S.
3. For RRT$^{\mathrm{X}}$, each propagation of size ϵ requires the same order amortized time as inserting a new node (which is the same order for RRT* and RRT$^{\mathrm{X}}$).

By construction RRT$^{\mathrm{X}}$ inherits the asymptotically optimal convergence of RRT* (we assume the planning problem, cost function, and ball parameter are defined appropriately). Theorem 38 from [5] has two relevant corollaries:

Corollary 5. *$\mathbb{P}\left(\{\limsup_{n\to\infty} g_n(v) = g^*(v)\}\right) = 1$ for all $v : v \in V_{n<\infty}$.*

Corollary 6. *$\lim_{n\to\infty} \mathbb{E}_n\left(g_n(v) - g^*(v)\right) = 0$ for all $v : v \in V_{n\leq\infty}$.*

Consider the case of adding v_x as the nth node in RRT* (Figure 3), where v_x is located at x. The RRT* parent of v_x is p and $\mathrm{d}(v_x, p)$ is the distance from v_x to p. The length of the trajectory from v_x to p is $\mathrm{d}_\pi(v_x, p)$. The radius of the shrinking neighborhood ball is r. By construction $\mathrm{d}(v_x, p) < r$ and $\mathrm{d}_\pi(v_x, p) < r$. Let $\hat{\mathrm{d}}(v_x, p)$ be a stand in for both $\mathrm{d}(v_x, p)$ and $\mathrm{d}_\pi(v_x, p)$. The following proposition comes from the fact that $\lim_{n\to\infty} r = 0$.

Proposition 3. $\lim_{n\to\infty} \mathbb{E}_{n,v_x}\left(\hat{\mathrm{d}}(v_x, p)\right) = 0$, *where p is RRT* parent of v_x.*

Lemma 2. $\lim_{n\to\infty} \mathbb{E}_{n,v_x}\left(\mathrm{g}_n(v_x) - \mathrm{g}^*(v_x)\right) = 0$.

Proof. By the triangle inequality $\mathrm{g}^*(v_x) + \mathrm{d}(p, v_x) \geq \mathrm{g}^*(p)$. Rearranging and then adding $\mathrm{g}_n(v_x)$ to either side:

$$\mathrm{g}_n(v_x) - \mathrm{g}^*(v_x) \leq \mathrm{g}_n(v_x) - \mathrm{g}^*(p) + \mathrm{d}(p, v_x). \tag{1}$$

(1) holds over all $S \in \{\hat{S} : V_n \setminus V_{n-1} = \{v_x\}\} \subset \mathcal{S}$, thus

$$\mathbb{E}_{n,v_x}\left(\mathrm{g}_n(v_x) - \mathrm{g}^*(v_x)\right) \leq \mathbb{E}_{n,v_x}\left(\mathrm{g}_n(v_x) - \mathrm{g}^*(p) + \mathrm{d}(p, v_x)\right). \tag{2}$$

By construction $\mathrm{g}_n(v_x) = \mathrm{g}_n(p) + \mathrm{d}_\pi(v_x, p)$. Substituting into (2), using the linearity of expectation, and taking the limit of either side:

$$\lim_{n\to\infty} \mathbb{E}_{n,v_x}\left(\mathrm{g}_n(v_x) - \mathrm{g}^*(v_x)\right) \leq$$
$$\lim_{n\to\infty} \mathbb{E}_{n,v_x}\left(\mathrm{g}_n(p) - \mathrm{g}^*(p)\right) + \lim_{n\to\infty} \mathbb{E}_{n,v_x}\left(\mathrm{d}_\pi(v_x, p)\right) + \lim_{n\to\infty} \mathbb{E}_{n,v_x}\left(\mathrm{d}(p, v_x)\right).$$

The law of large numbers guarantees that x (i.e., location of v_x) becomes uncorrelated with the cost-to-goal of p, in the limit as $n \to \infty$. Thus, consequently: $\lim_{n\to\infty} \mathbb{E}_{n,v_x}\left(\mathrm{g}_n(p) - \mathrm{g}^*(p)\right) = \lim_{n\to\infty} \mathbb{E}_n\left(\mathrm{g}_n(p) - \mathrm{g}^*(p)\right)$. Using Corollary 6 and Proposition 3 (twice) finishes the proof. □

Applying the law of total expectation yields the following corollary regarding the nth node added to V, and completes step 1 of the overall proof.

Corollary 7. $\lim_{n\to\infty} \mathbb{E}_n\left(\mathrm{g}_n(v) - \mathrm{g}^*(v)\right) = 0$, *where $V_n \setminus V_{n-1} = \{v\}$.*

Lemma 3. $\sum_{m=1}^{n} \mathbb{E}_m\left(\mathrm{g}_m(v_m) - \mathrm{g}^*(v_m)\right) = O\left(\epsilon n\right)$ *for all n such that $1 \leq n < \infty$ and where $V_m \setminus V_{m-1} = \{v_m\}$ for all m s.t. $1 \leq m \leq \infty$.*

Proof. Using the definition of a limit with Corollary 7 shows that for any $c_1 > 0$ there must exist some $n_c < \infty$ such that $\mathbb{E}_n\left(\mathrm{g}_n(v) - \mathrm{g}^*(v)\right) < c_1$ for all $n > n_c$. We choose $c_1 = \epsilon$ and define $c_2 = \sum_{m=1}^{n_c} \mathbb{E}_m\left(\mathrm{g}_m(v_m) - \mathrm{g}^*(v_m)\right)$ so that by construction $\sum_{m=1}^{n} \mathbb{E}_m\left(\mathrm{g}_m(v_n) - \mathrm{g}^*(v_n)\right) \leq c_2 + \epsilon n$ for all n s.t. $1 \leq n < \infty$. □

Let $f^{pr}(n)$ denote the total number of cost propagations that occur (i.e., through any and all nodes) in $\mathrm{RRT}^{\mathrm{X}}$ as a function of $n = |V_n|$.

Lemma 4. $\mathbb{E}_n\left(f^{pr}(n)\right) = O\left(n\right)$

Proof. When $m = |V_m|$ a propagation is possible only if there exists some node v such that $\mathrm{g}_m(v) - \mathrm{g}^*(v) > \epsilon$. Assuming RRT* and RRT$^{\mathrm{X}}$ use the same S, then by construction $\mathrm{g}_m(v)$ for RRT* is an upper bound on $\mathrm{g}_m(v)$ for RRT$^{\mathrm{X}}$ for all $v \in V_m$ and m such that $1 \leq m < \infty$. Thus, $f^{pr}(n) \leq (1/\epsilon) \sum_{m=1}^{n} \mathrm{g}_m(v_m) - \mathrm{g}^*(v_m)$, where $\mathrm{g}_m(v_m)$ is the RRT*-value of this quantity. Using the linearity of expectation to apply Lemma 3 we find that $\mathbb{E}_n\left(f^{pr}(n)\right) \leq (1/\epsilon) O\left(\epsilon n\right)$. □

Corollary 8. $\lim_{n\to\infty} \frac{\mathbb{E}_n(f^{pr}(n))}{cn} \leq 1$ *for some constant* $c < \infty$.

Corollary 8 concludes step two of the overall proof. The following Lemma 5 uses the notion of runtime amortization[6]. Let $\hat{f}^{single}(n)$ denote the *amortized* time to propagate an ϵ-cost reduction from node v to $N(v)$ when $n = |V_n|$.

Lemma 5. $\mathbb{P}\left(\{\lim_{n\to\infty} \frac{\hat{f}^{single}(n)}{c \log n} \leq 1\}\right) = 1$ *for some constant* $c < \infty$.

Proof. By construction, a single propagation through v requires interaction with $|N(v)|$ neighbors. Each interaction normally requires $\Theta\left(1\right)$ time–except when the interaction results in $u \in N(v)$ receiving an ϵ-cost decrease. In the latter case u is added/updated in the priority queue in $O\left(\log n\right)$ time; however, we add this $O\left(\log n\right)$ time to u's *next* propagation time, so that the current propagation from v only incurs $\Theta\left(1\right)$ *amortized* time per each $u \in N(v)$. To be fair, v must account for any similar $O\left(\log n\right)$ time that it has absorbed from each of the c_1 nodes that have given it an ϵ-cost reduction since the last propagation from v. But, for $c_1 \geq 1$ the current propagation from v is at least $c_1\epsilon$ and so we can count it as c_1 different ϵ-cost decreases from v to $N(v)$ (and v only touches each $u \in N(v)$ once). Hence, $c_1 \hat{f}^{single}(n) = |N(v)| + c_1 O\left(\log n\right)$. By the law of large numbers, $\mathbb{P}\left(\{\lim_{n\to\infty} |N(v)| = c_2 \log n)\right\} = 1$, for some constant c_2 s.t. $0 < c_2 < \infty$. Hence, $\mathbb{P}\left(\left\{\lim_{n\to\infty} \hat{f}^{single}(n) \leq (c_2/c_1) \log n + \log n\right\}\right) = 1$, setting $c = 1 + c_2/c_1$ finishes the proof. □

Corollary 9. $\lim_{n\to\infty} \frac{\mathbb{E}_n\left(\hat{f}^{single}(n)\right)}{c \log n} \leq 1$ *for some constant* $c < \infty$.

Let $f^{all}(n)$ denote the *total* runtime associated with cost propagations by the iteration that $n = |V_n|$, where $f^{all}(n) = \sum_{j=1}^{f^{pr}(n)} \hat{f}^{single}(m_j)$ for a particular run of RRT$^{\mathrm{X}}$ resulting in $n = |V_n|$ and $f^{pr}(n)$ individual ϵ-cost decreases.

Lemma 6. $\lim_{n\to\infty} \frac{\mathbb{E}_n\left(f^{all}(n)\right)}{cn \log n} < 1$ *for* $c \leq \infty$.

Proof. $\lim_{n\to\infty} \mathbb{E}_n\left(f^{pr}(n)\right) \neq 0$ and $\lim_{n\to\infty} \mathbb{E}_n\left(\hat{f}^{single}(n)\right) \neq 0$, so obviously $\lim_{n\to\infty} \frac{\mathbb{E}_n(f^{pr}(n))\mathbb{E}_n\left(\hat{f}^{single}(n)\right)}{\mathbb{E}_n(f^{pr}(n))\mathbb{E}_n\left(\hat{f}^{single}(n)\right)} = 1$. Although $\hat{f}^{single}(n)$ and $f^{pr}(n)$ are mutually

[6] In particular, if a node u receives an ϵ-cost decrease $> \epsilon$ via another node v, then u agrees to take responsibility for the runtime associated with that exchange (i.e., including it as part u's next propagation time).

dependent, in general, they become *independent*[7] in the limit as $n \to \infty$. Thus, $\lim_{n\to\infty} \frac{\mathbb{E}_n\left(f^{pr}(n)\hat{f}^{single}(n)\right)}{\mathbb{E}_n(f^{pr}(n))\mathbb{E}_n\left(\hat{f}^{single}(n)\right)} = \lim_{n\to\infty} \frac{\mathbb{E}_n(f^{pr}(n))\mathbb{E}_n\left(\hat{f}^{single}(n)\right)}{\mathbb{E}_n(f^{pr}(n))\mathbb{E}_n\left(\hat{f}^{single}(n)\right)} = 1$. Note that the previous step would not have been allowed *outside* the limit. Using algebra: $\lim_{n\to\infty} \frac{\mathbb{E}_n\left(\sum_{j=1}^{f^{pr}(n)} \hat{f}^{single}(n)\right)}{\mathbb{E}_n(f^{pr}(n))\mathbb{E}_n\left(\hat{f}^{single}(n)\right)} = 1$. Using Corollaries 8 and 9: $\lim_{n\to\infty} \frac{\mathbb{E}_n\left(\sum_{j=1}^{f^{pr}(n)} \hat{f}^{single}(n)\right)}{c_1 c_2 n \log n} \leq \lim_{n\to\infty} \frac{\mathbb{E}_n\left(\sum_{j=1}^{f^{pr}(n)} \hat{f}^{single}(n)\right)}{\mathbb{E}_n(f^{pr}(n))\mathbb{E}_n\left(\hat{f}^{single}(n)\right)} = 1$ for some $c_1, c_2 < \infty$. Using algebra and defining $c = c_1 c_2$ finishes the proof. □

Corollary 10. $\mathbb{E}_n\left(f^{all}(n)\right) = O\left(n \log n\right).$

Theorem 1. $\mathbb{E}_n\left(f^{RRT^X}(n)\right) = \Theta\left(n \log n\right).$

Proof. When $\Delta\mathcal{X}_{\text{obs}} = \emptyset$, RRT$^{\text{X}}$ differs from RRT* in 3 ways: (1) ϵ-cost propagation, (2) neighbor list storage, and (3) neighbor list culling[8]. RRT$^{\text{X}}$ runtime is found by adding the extra time of (1), (2), and (3) to that of RRT*. Corollary 10 gives the asymptotic time of (1). (2) and (3) are $O\left(|N(v)|\right)$, the same as finding a new node's neighbors in RRT*. Therefore, $\mathbb{E}_n\left(f^{RRT^X}(n)\right) = \mathbb{E}_n\left(f^{RRT*}(n)\right) + \mathbb{E}_n\left(f^{all}(n)\right)$. Trivially: $\mathbb{E}_n\left(f^{RRT*}(n)\right) = \Theta\left(\mathbb{E}_n\left(f^{RRT*}(n)\right)\right)$, and by Corollaries 4 and 10: $\mathbb{E}_n\left(f^{RRT^X}(n)\right) = \Theta\left(n \log n\right) + O\left(n \log n\right) = \Theta\left(n \log n\right)$ □

3.3 Expected Time until $|V| = n$ for RRT$^{\#}$

RRT$^{\#}$ does not cull neighbors (in contrast to RRT$^{\text{X}}$) and so all nodes continue to accumulate neighbors forever.

Lemma 7. $\mathbb{E}_n\left(|N(v)|\right) = \Theta\left(\log^2 n\right)$ *for all* v *s.t.* $v \in V_m$ *for some* $m < n < \infty$.

Proof. Assuming v is inserted when $m = |V_m|$, the expected value of $\mathbb{E}_n\left(|N(v)|\right)$, where $n = |V_n|$ for some $n > m$ is:
$\mathbb{E}_n\left(|N(v)|\right) = c \log m + \sum_{j=m+1}^{n} \frac{c \log j}{j} = c\left(\log m + \left(\sum_{j=1}^{n} \frac{\log j}{j}\right) - \left(\sum_{j=1}^{m} \frac{\log j}{j}\right)\right)$
where c is constant. Corollary 1 finishes the proof. □

RRT$^{\#}$ propagates all cost changes (in contrast, RRT$^{\text{X}}$ only propagates those larger than ϵ). Thus, *any* cost decrease at v is propagated to *all* descendants of v, *plus* any additional nodes that become new descendants of v due to the propagation. Let $f^{pr\#}(n)$ be the number of propagations (i.e., through a single node) that have occurred in RRT$^{\#}$ by the iteration that $n = |V_n|$.

Lemma 8. $\mathbb{P}(\lim_{n\to\infty} \frac{n}{f^{pr\#}(n)} = 0) = 1$ *with respect to* $\mathcal{S}$.

[7] i.e., because the number of neighbors of a node converges to the function $\log n$ with probability 1 (as explained in Lemma 5)

[8] Note that neighbors that are not removed during a cull are touched again during the RRT*-like rewiring operation that necessarily follows a cull operation.

Proof. By contradiction. Assume that $\mathbb{P}(\lim_{n\to\infty} \frac{n}{f^{pr\#}(n)} = 0) = c_1 < 1$. Then there exists some c_2 and c_3 such that $\mathbb{P}(\lim_{n\to\infty} \frac{n}{f^{pr\#}(n)} \geq c_2 > 0) = c_3 > 0$ and therefore $\mathbb{E}(\lim_{n\to\infty} \frac{n}{f^{pr\#}(n)}) \geq c_2 c_3 > 0$, and the expected number of cost propagations to each v s.t. $v \in V_n$ is $c_4 = \frac{1}{c_2 c_3} < \infty$, in the limit as $n \to \infty$. This is a contradiction because v experiences an infinite number of cost *decreases* with probability 1 as a result of RRT$^{\#}$'s asymptotic optimal convergence, and each decrease (at a non-leaf node) causes at least one propagation. □

The runtime of RRT$^{\#}$ can be expressed in terms of the runtime of RRT* plus the extra work required to keep the graph consistent (cost propagations):

$$f_{m_i}^{RRT\#} = \Theta\left(f_{m_i}^{RRT*}\right) + \sum_{j=1}^{f^{pr\#}(m_i)} f_j^{pr\#}. \tag{3}$$

Here, $f^{pr\#}(m_i)$ is the total number of cost propagations (i.e., through a single node) by iteration i when $m_i = |V|$, and $f_j^{pr\#}$ is the time required for the jth propagation (i.e., through a single node). Obviously $f_j^{pr\#} > c$ for all j, where $c > 0$. Also, for all $j \geq 1$ and all m the following holds, due to non-decreasing expected neighbor set size vs. j:

$$\mathbb{E}_m\left(f_j^{pr\#}\right) \leq \mathbb{E}_m\left(f_{j+1}^{pr\#}\right) \tag{4}$$

Lemma 9. $\lim_{n\to\infty} \frac{n \log^2 n}{\mathbb{E}_n\left(\sum_{j=1}^{f^{pr\#}(n)} f_j^{pr\#}\right)} = 0.$

Proof. $\lim_{n\to\infty} \frac{\mathbb{E}_n\left(\sum_{j=1}^{n} f_j^{pr\#}\right)}{\mathbb{E}_n\left(\sum_{j=n+1}^{f^{pr\#}(n)} f_j^{pr\#}\right)} = 0$ because the ratio between the number of terms in the numerator vs. denominator approaches 0, in the limit, by Lemma 8, and the smallest term in the denominator is no smaller than the largest term in the numerator, by (4). Obviously, $\lim_{n\to\infty} \frac{\mathbb{E}_n\left(\sum_{j=1}^{n} f_j^{pr\#}\right)}{\mathbb{E}_n\left(\sum_{j=1}^{f^{pr\#}(n)} f_j^{pr\#}\right)} \leq \lim_{n\to\infty} \frac{\mathbb{E}_n\left(\sum_{j=1}^{n} f_j^{pr\#}\right)}{\mathbb{E}_n\left(\sum_{j=n+1}^{f^{pr\#}(n)} f_j^{pr\#}\right)} = 0.$
Rearranging: $\lim_{n\to\infty} \frac{\sum_{j=1}^{n} \mathbb{E}_{m_j}\left(f_j^{pr\#}\right)}{\mathbb{E}_n\left(\sum_{j=1}^{f^{pr\#}(n)} f_j^{pr\#}\right)} = 0$. By Lemma 7: $\lim_{n\to\infty} \frac{\sum_{j=1}^{n} \mathbb{E}_{m_j}\left(c \log^2 m_j\right)}{\sum_{j=1}^{n} \mathbb{E}_{m_j}\left(f_j^{pr\#}\right)} = 1$
for some constant c such that $0 < c < \infty$. Hence,

$$\lim_{n\to\infty} \frac{\sum_{j=1}^{n} \mathbb{E}_{m_j}\left(f_j^{pr\#}\right)}{\mathbb{E}_n\left(\sum_{j=1}^{f^{pr\#}(n)} f_j^{pr\#}\right)} \quad \frac{\sum_{j=1}^{n} \mathbb{E}_{m_j}\left(c \log^2 m_j\right)}{\sum_{j=1}^{n} \mathbb{E}_{m_j}\left(f_j^{pr\#}\right)} = \frac{c \sum_{j=1}^{n} \mathbb{E}_{m_j}\left(\log^2 m_j\right)}{\mathbb{E}_n\left(\sum_{j=1}^{f^{pr\#}(n)} f_j^{pr\#}\right)} = 0$$

Corollary 2 with the linearity of expectation finishes the proof. □

Corollary 11. $\mathbb{E}_n\left(\sum_{j=1}^{f^{pr\#}(n)} f_j^{pr\#}\right) = \omega\left(n \log^2 n\right).$

In the above, $\omega\left(n \log^2 n\right)$ is a stronger statement than $\Omega\left(n \log^2 n\right)$. We are now ready to prove the asymptotic runtime of RRT$^{\#}$.

Theorem 2. $\mathbb{E}_n\left(f_n^{RRT\#}\right) = \omega\left(n \log^2 n\right)$

Proof. Taking the limit (as $m_i = n \to \infty$) of the expectation of either side of (3) and then using Corollaries 4 and 11, we see that the expected runtime of RRT$^{\#}$ is dominated by propagations: $\mathbb{E}_n\left(f_n^{RRT\#}\right) = \Theta\left(n \log n\right) + \omega\left(n \log^2 n\right)$. □

3.4 RRT$^{\mathrm{X}}$ Obstacle Addition/Removal in Dynamic Environments

The addition of an obstacle requires finding V_{obs}, the set of all nodes with trajectories through the obstacle, and takes expected $O\left(|D(V_{\text{obs}})| \log n\right)$ time. The resulting call to `reduceInconsistency()` interacts with each $u \in D(V_{\text{obs}})$, where $D(V_{\text{obs}})$ is the set of all descendants of all $u \in V_{\text{obs}}$, and each interaction takes expected time $O\left(\log n\right)$ due to neighbor sets and heap operations. Thus, adding an obstacle requires expected time $O\left(|D(V_{\text{obs}})| \log n\right)$. Removing an obstacle requires similar operations and thus the same order of expected time. In the special case that the obstacle has existed since t_0, then $D(V_{\text{obs}}) = \emptyset$ and time is $O\left(1\right)$.

4 Simulation: Dubins Vehicle in a Dynamic Environment

We have tested RRT$^{\mathrm{X}}$ on a variety of problems and state spaces, including unpredictably moving obstacles; we encourage readers to watch the videos we have posted online at [13]. However, due to space limitations, we constrain the focus of the current paper to how RRT$^{\mathrm{X}}$ can be used to solve a Dubins vehicle problem in a dynamic environment.

The state space is defined $\mathcal{X} \subset \mathbb{R}^2 \times \mathbb{S}^1$. The robot moves at a constant speed and has a predefined minimum turning radius $r_{\min}$ [9]. Distance between two points $x, y \in \mathcal{X}$ is defined $\mathrm{d}(x, y) = \sqrt{c_\theta (x_\theta - y_\theta)^2 + \sum_{i=1}^{2} (x_i - y_i)^2}$, where x_θ is heading and x_i is the coordinate of x with respect to the ith dimension of $\mathbb{R}^2$, and assuming the identity $\theta = \theta + 2\pi$ is obeyed. The constant c_θ determines the cost trade-off between a difference in location vs. heading. $\mathrm{d}(x, y)$ is the length of the geodesic between x and y through $\mathbb{R}^2 \times \mathbb{S}^1$. $\mathrm{d}_\pi(x, y)$ is the length of the Dubins trajectory that moves the robot from x to y. In general, $\mathrm{d}_\pi(x, y) \neq \mathrm{d}(x, y)$; however, by defining c_θ appropriately (e.g., $c_\theta = 1$) we can guarantee $\mathrm{d}_\pi(x, y) \geq \mathrm{d}(x, y)$ so that $\mathrm{d}(x, y)$ is an admissible heuristic on $\mathrm{d}_\pi(x, y)$.

Figure 1 shows a simulation. $r_{\min} = 2\text{m}$, speed $= 20$ m/s, sensor range $= 10\text{m}$. The robot plans for 10s before moving, but must react on-the-fly to $\Delta\mathcal{X}_{\text{obs}}$.

5 Discussion

RRT$^{\mathrm{X}}$ is the first asymptotically optimal algorithm designed for kinodynamic replanning in environments with unpredictably changing obstacles. Analysis and simulations suggest that it can be used for effective real-time navigation. That said, the myopia inherent in dynamic environments makes it impossible for any algorithm/agent to avoid collisions with obstacles that can overwhelm finite agility and/or information (e.g., appear at a location that cannot be avoided).

Analysis shows that RRT$^{\mathrm{X}}$ is competitive with all state-of-the-art motion planning algorithms in static environments. RRT$^{\mathrm{X}}$ has the same order expected runtime as RRT and RRT*, and is quicker than RRT$^{\#}$. RRT$^{\mathrm{X}}$ inherits probabilistic completeness and asymptotic optimality from RRT*. By maintaining a ϵ-consistent graph, RRT$^{\mathrm{X}}$ has similar behavior to RRT$^{\#}$ for cost changes larger than ϵ; which translates into faster convergence than RRT* in practice.

Algorithm 7: updateObstacles()

```
if ∃O : O ∈ 𝒪 ∧ O has vanished then
    forall O : O ∈ 𝒪 ∧ O has vanished do
        removeObstacle(O)
    reduceInconsistency()
if ∃O : O ∉ 𝒪 ∧ O has appeared then
    forall O : O ∉ 𝒪 ∧ O has appeared do
        addNewObstacle(O)
    propogateDescendants()
    verrifyQueue(v_bot)
    reduceInconsistency()
```

Algorithm 8: propogateDescendants()

```
forall v ∈ V_T^c do
    V_T^c ← V_T^c ∪ C_T^-(v)
forall v ∈ V_T^c do
    forall u ∈ (N^+(v) ∪ {p_T^+(v)}) \ V_T^c do
        g(u) ← ∞
        verrifyQueue(u)
forall v ∈ V_T^c do
    V_T^c ← V_T^c \ {v}
    g(v) ← ∞
    lmc(v) ← ∞
    if p_T^+(v) ≠ ∅ then
        C_T^-(p_T^+(v)) ← C_T^-(p_T^+(v)) \ {v}
        p_T^+(v) ← ∅
```

Algorithm 9: verrifyOrphan(v)

```
if v ∈ Q then remove(Q, v)
V_T^c ← V_T^c ∪ {v}
```

Algorithm 10: removeObstacle(O)

```
E_O ← {(v, u) ∈ E : π(v, u) ∩ O ≠ ∅}
𝒪 ← 𝒪 \ {O}
E_O ← E_O \ {(v, u) ∈ E : π(v, u) ∩ O' ≠ ∅
    for some O' ∈ 𝒪}
V_O ← {v : (v, u) ∈ E_O}
forall v ∈ V_O do
    forall u : (v, u) ∈ E_O do
        d_π(v, u) ← recalculate d_π(v, u)
    updateLMC(v)
    if lmc(v) ≠ g(v) then verrifyQueue(v)
```

Algorithm 11: addNewObstacle(O)

```
𝒪 ← 𝒪 ∪ {O}
E_O ← {(v, u) ∈ E : π(v, u) ∩ O ≠ ∅}
forall (v, u) ∈ E_O do
    d_π(v, u) ← ∞
    if p_T^+(v) = u then verrifyOrphan(v)
    if v_bot ∈ π(v, u) then π_bot = ∅
```

Algorithm 12: verrifyQueue(v)

```
if v ∈ Q then update(Q, v)
else add(Q, v)
```

Algorithm 13: updateLMC(v)

```
cullNeighbors(v, r)
forall u ∈ N^+(v) \ V_T^c : p_T^+(u) ≠ v do
    if lmc(v) > d_π(v, u) + lmc(u) then
        p' ← u
makeParentOf(p', v)
```

The consistency parameter ϵ must be greater than 0 to guarantee $\Theta(\log n)$ expected iteration time. It should be small enough such that a rewiring cascade is triggered whenever obstacle changes require a course correction by the robot. For example, in a Euclidean space (and assuming the robot's position is defined as its center point) we suggest using an ϵ no larger than 1/2 the robot width.

6 Conclusions

We present RRT$^{\mathrm{X}}$, the first asymptotically optimal sampling-based replanning algorithm. RRT$^{\mathrm{X}}$ facilitates real-time navigation in unpredictably changing environments by rewiring the same search tree for the duration of navigation, continually repairing it as changes to the state space are detected. Resulting motion plans are both valid, with respect to the dynamics of the robot, and asymptotically optimal in static environments. The robot is also able to improve its plan while it is in the process of executing it. Analysis and simulations show that RRT$^{\mathrm{X}}$ works well in both unpredictably changing and static environments. In static environments the runtime of RRT$^{\mathrm{X}}$ is competitive with RRT and RRT* and faster than RRT$^{\#}$.

Acknowledgments This work was supported by the Air Force Office of Scientific Research, grant #FA-8650-07-2-3744.

References

1. Arslan, O., Tsiotras, P.: Use of relaxation methods in sampling-based algorithms for optimal motion planning. In: Robotics and Automation (ICRA), 2013 IEEE International Conference on. pp. 2421–2428. IEEE (2013)
2. Bruce, J., Veloso, M.: Real-time randomized path planning for robot navigation. In: IEEE/RSJ International Conference on Intelligent Robots and Systems. vol. 3, pp. 2383–2388 vol.3 (2002)
3. Ferguson, D., Kalra, N., Stentz, A.: Replanning with rrts. In: IEEE International Conference on Robotics and Automation. pp. 1243–1248 (May 2006)
4. Gayle, R., Klingler, K., Xavier, P.: Lazy reconfiguration forest (lrf) - an approach for motion planning with multiple tasks in dynamic environments. In: IEEE International Conference on Robotics and Automation. pp. 1316–1323 (April 2007)
5. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. Int. Journal of Robotics Research 30(7), 846–894 (June 2011)
6. Kavraki, L., Svestka, P., Latombe, J., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE transactions on Robotics and Automation (Jan 1996)
7. Koenig, S., Likhachev, M., , Furcy, D.: Lifelong planning A*. Artificial Intelligence Journal 155(1-2), 93–146 (2004)
8. Koenig, S., Likhachev, M., Furcy, D.: D* lite. In: Proceedings of the Eighteenth National Conference on Artificial Intelligence. pp. 476–483 (2002)
9. LaValle, S.: Planning Algorithms. Cambridge University Press (2006)
10. LaValle, S., Kuffner, J.J.: Randomized kinodynamic planning. International Journal of Robotics Research 20(5), 378–400 (2001)
11. Lavalle, S.M., Lindemann, S.: Simple and efficient algorithms for computing smooth, collision-free feedback laws over given cell decompositions. The International Journal of Robotics Research 28(5), 600–621 (2009)
12. Marble, J.D., Bekris, K.E.: Asymptotically near-optimal planning with probabilistic roadmap spanners. IEEE Transactions on Robotics 29(2), 432–444 (2013)
13. Otte, M.: Videos of RRT^{X} simulations in various state spaces (March 2014), http://tinyurl.com/l53gzgd
14. Otte, M.: Any-Com Multi-Robot Path Planning. Ph.D. thesis, University of Colorado at Boulder (2011)
15. Rimon, E., Koditschek, D.E.: Exact robot navigation using artificial potential functions. Robotics and Automation, IEEE Transactions on 8(5), 501–518 (1992)
16. Salzman, O., Halperin, D.: Asymptotically near-optimal rrt for fast, high-quality, motion planning. arXiv arXiv:1308.0189v3 (2013)
17. Stentz, A.: The focussed D* algorithm for real-time replanning. In: Proceedings of the International Joint Conference on Artificial Intelligence (August 1995)
18. Tedrake, R., Manchester, I.R., Tobenkin, M., Roberts, J.W.: Lqr-trees: Feedback motion planning via sums-of-squares verification. The International Journal of Robotics Research 29(8), 1038–1052 (2010)
19. Zucker, M., Kuffner, J., Branicky, M.: Multipartite rrts for rapid replanning in dynamic environments. In: IEEE International Conference on Robotics and Automation. pp. 1603–1609 (April 2007)