The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

Extending Rapidly-Exploring Random Trees for Asymptotically Optimal Anytime Motion Planning

Yasin Abbasi-Yadkori and Joseph Modayil and Csaba Szepesvari

***Abstract*— We consider the problem of anytime planning in continuous state and action spaces with non-linear deterministic dynamics. We review the existing approaches to this problem and find no algorithms that both quickly find *feasible* solutions and also eventually approach *optimal* solutions with additional time. The state-of-the-art solution to this problem is the rapidly-exploring random tree (RRT) algorithm that quickly finds a feasible solution. However, the RRT algorithm does not return better results with additional time. We introduce RRT^{++}, an anytime extension of the basic RRT algorithm. We show that the new algorithm has desirable theoretical properties and experimentally show that it efficiently finds near optimal solutions.**

I. Introduction

Consider a motion planning problem for a wheeled robot with non-trivial dynamics navigating through an environment filled with benches. The rapidly exploring random tree (RRT) algorithm [5] is the most popular method for solving this planning problem, but in cluttered environments (Figure 1) the RRT algorithm will reliably fail to find a path that is anywhere near optimal. This problem raises the question: does there exist an algorithm that returns feasible solutions as quickly as RRT and also asymptotically approaches an optimal solution?

Several papers have explored related ideas. One approach uses heuristics to bias the search [8]. Another approach [1], runs RRT repeatedly with tighter constraints on each iteration to improve the solution quality. A third approach [2] examines the restricted class of problems that permit bidirectional extensions and lack differential constraints. Our proposed algorithm outperforms the first two methods, and is more general than the third.

First, we present a formal characterization of desirable properties for an anytime planning algorithm. We review popular approaches to planning algorithms and argue that they do not satisfy these properties. Then, we present RRT^{++}, an anytime extension to the RRT algorithm which satisfies some of these properties. Finally, we present experiments that demonstrate the ability of the new algorithm to efficiently approach optimal solutions in a cluttered environment.

II. Problem Formulation

We consider the problem of planning in continuous state and action spaces with non-linear deterministic dynamics. We are interested in anytime algorithms for a single query problem. The planning agent has a set of actions $\mathcal{A} \subset \mathbb{R}^n$ and the environment states are given by the set $\mathcal{X} \subset \mathbb{R}^d$.

Department of Computing Science, University of Alberta, Email: {abbasiya, jmodayil, szepesva}@ualberta.ca

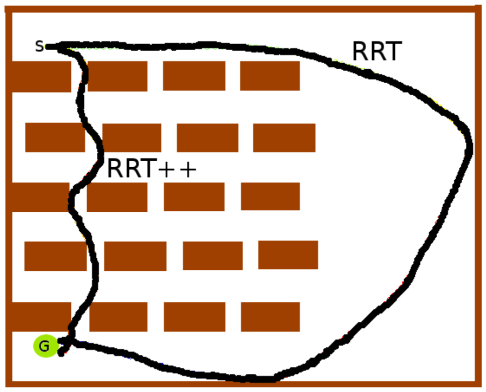

Fig. 1. A robot motion planning problem going from S to G with dynamical constraints. The robot must avoid several bench shaped obstacles in this cluttered environment. The RRT algorithm typically finds a substantially sub-optimal solution. Our new algorithm RRT^{++} eventually finds a nearly optimal path.

Let the set of planning problems be $\mathcal{P}$. Each problem

$$P = <x^s, g, f, m, c>$$

has a start state $x^s \in \mathcal{X}$, and *specification* functions g, f, m, and c. The goal function is $g : \mathcal{X} \to \{0, 1\}$, and the free space indicator is $f : \mathcal{X} \to \{0, 1\}$. The set of all states for which $f(x) = 1$ is called the free space (or the collision free space) and is denoted by $\mathcal{X}_1$. The function $m : \mathcal{X} \times \mathcal{A} \to \mathcal{X}$ is a black-box deterministic forward model of the system dynamics, where at time t the agent takes the action a_t and the agent goes to the next state $x_{t+1} = m(x_t, a_t)$. The cost function $c : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ associates a cost with every step. (A collision can be modelled by setting x_{t+1} to a collision free state and/or setting the cost to infinite). Although the problem is fully specified by the above formulation, solution methods may require auxiliary structures. In particular, with RRT we assume the existence of a setpoint controller $\kappa : \mathcal{X} \times \mathcal{X} \to \mathcal{A}$ that takes a current state and a target state as inputs and returns an action that brings the agent's state closer to the target.

A trajectory J is a connected sequence of state and action steps, $J = \{(x(t), a(t))\}_{t \in \{1,\ldots,L\}}$, where L is the trajectory length. For a non-feasible trajectory, namely one that does not terminate in a goal state, the cost of the trajectory is set to be infinity. The cost of a feasible trajectory is given by the sum of the costs of the steps.

An anytime planning algorithm A is a potentially non-terminating computation that takes a planning problem as an input and can be asked for a solution at any point in time.

We restrict our attention to anytime algorithms for which the cost is monotonically decreasing, as this can easily be satisfied. We consider time to be constrained by the number of calls to the specification functions of the problem. Let $i \in \mathbb{N}$ be the number of such function calls (interactions) when algorithm A is run on problem P. We shall denote the output J of A (a trajectory) at this time i on problem P by $J = A(P, i)$. Also note that we consider randomized anytime algorithms, namely given the same inputs, the output could vary randomly. We let $\mathcal{S}$ ($\mathcal{S}_{\mathrm{det}}$) denote the set of such randomized (deterministic) algorithms.

Define the cost of an optimal plan

$$c^*(P) = \min_{A \in \mathcal{S}_{\mathrm{det}},\ i \in \mathbb{N}} c(A(P, i)).$$

Define the expected regret of an algorithm after i interactions to be

$$R^A_{i,P} = \min(\mathbb{E}\left[c(A(P, i))\right] - c^*(P), R^*).$$

Thus, $R^* > 0$ is the upper bound on the regret. When R^* is finite, the meaning of R^* is that a trajectory with a cost above R^* is as useless as not knowing a feasible solution. Thus a finite value of R^* allows us to quantify what we mean by plans that are useless in practice.

We define the *feasibility time*, $F(A, P)$, as the expected time for an algorithm A to find a useful solution for the problem P:

$$F(A, P) = \mathbb{E}\left[I^A_P\right],$$

where

$$I^A_P = \min\{i \mid c(A(P, i) < R^*\}$$

is the (random) number of interactions that algorithm A needs on problem P to produce a trajectory with cost less than R^*.

We say an anytime planning algorithm is an *(asymptotic) no-regret* algorithm for a problem P if

$$\lim_{i \to \infty} R^A_{i,P} = 0.$$

This means for any problem, the returned solutions will eventually be arbitrarily close to optimal in expectation. Algorithms which do not have this property are *regretable*.

We say an anytime planning algorithm is a *uniform (asymptotic) no-regret* algorithm over problem class $\mathcal{P}$ if

$$\lim_{i \to \infty} \sup_{P \in \mathcal{P}} R^A_{i,P} = 0.$$

Note that $\sup_{P \in \mathcal{P}} R^A_{i,P}$ is the worst-case regret over $\mathcal{P}$ after i interactions, i.e., this value bounds the regret irrespectively of the problem. The interesting problem in connection to the uniform no-regret property is to characterize the largest set of problems (specific to a chosen algorithm) for which the uniform no-regret property holds.

Having a small feasibility time and being a no-regret algorithm are desired properties. However, these are crude concepts and only represent minimum requirements. In particular, that one of these desired properties holds for an algorithm does not necessarily translate into a good practical performance. However, not having one of these properties usually means that something is not right with the algorithm. In particular, in such cases we expect that the algorithm considered will have difficulties on some practical problems.

III. Existing Algorithms

We first introduce two problem features that affect popular algorithms. If the wall in Figure 2(a) is thin and blocks passage, then we call it a *narrow blocking wall* (NBW). If there is a freespace region which is thin (shown here with width ϵ) but passable, we call it a *narrow passage* (NP). These problem features naturally affect discrete state approaches and also affect sample based approaches. Assume an algorithm tests some actions in some states to find the solution. If it hits an obstacle and the algorithm is optimistic (i.e. assumes that the obstacle is not large and does not block the passage), then this algorithm will get stuck indefinitely in the presence of a narrow blocking wall. On the other hand, if the algorithm is pessimistic (i.e. assumes that the obstacle is large and blocks the passage), then this algorithm is not able to find a narrow passage. We place both of these features into a sample problem called $p(\epsilon)$, as shown in Figure 2. In this example a kinematic model is assumed.

First we review a common problem in the existing algorithms. Then, we review existing algorithms and show that they are subject to this problem.

A. Solving an abstract problem (SAP)

Many algorithms simplify the original problem by first building and solving an abstracted problem. For example, they may remove the obstacles in a map or learn a heuristic for the map. They may discretize the problem by placing a grid over the state space, find the shortest path over this grid and then search for a sequence of controls that keeps the robot near this shortest path. However, a solution to the abstract problem may not correspond to a feasible solution for the original problem, and conversely solutions to the original problem may not be feasible solutions for the abstract problem. Hence abstractions can lose soundness, completeness, or both.

Let us now review some existing approaches. We start with some stylized algorithms that attempt to work with an abstraction to illustrate some issues.

B. Iteratively Refined Action Spaces

One algorithm to find near optimal solutions is to use discrete action planning methods by incrementally refining the discretization of the continuous action space (which we shall call IRAS). This approach tries one level of action space discretization, and then refines the discretization when no solution is found up to a given trajectory length of L of action sequences. For example, we could define an action granularity of $1/\epsilon$ in each of d dimensions, and consider all paths up to length N/ϵ for some problem specific N. As the algorithm would eventually consider all possible paths, it would eventually consider trajectories with costs that are arbitrarily close to optimal. Thus this is an asymptotic no-regret algorithm:

Proposition 1: IRAS is a no-regret algorithm for $p(\epsilon)$
Unfortunately, it will often take an extremely long time to generate a feasible solution. As this approach considers all possible action sequences, it must search exponentially many

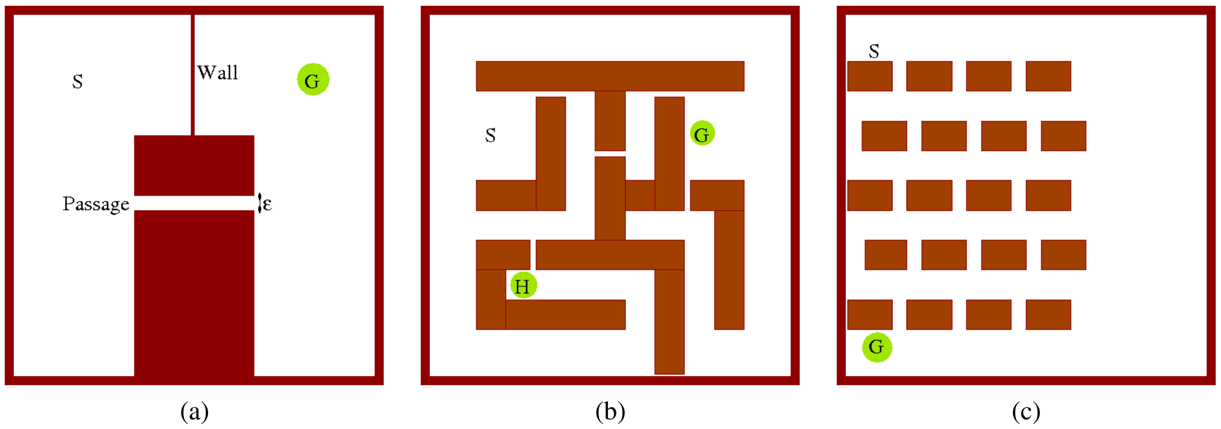

Fig. 2. The planning problems we consider in this paper. The letter S is at the start state, and the green circles are goal regions. The brown blocks are obstacles. a) The map $p(\epsilon)$, used for analysing common problems for anytime algorithms. b) A map with multiple narrow passages. The width of each wall is 10 and the size of each narrow passage is 1.6. c) An example of a realistic cluttered environment where RRT will often return a substantially suboptimal path.

leaf nodes until it finds the goal. For the $p(\epsilon)$ problem, the solution time is governed by $(1/\epsilon)^n$ where $n = L/S$ and $L > 1$ is the length of the shortest path between the start state and the goal, and $S = \epsilon$ is the step size.

Proposition 2: $F(\text{IRAS}, p(\epsilon)) = \Omega((1/\epsilon)^{1/\epsilon})$.

C. Incrementally refined state spaces

Given the difficulty of handling all possible action trajectories, one popular alternative is to perform a discretization of the state space. This approach has the benefit of potentially sparse state to state transitions. Unfortunately, when continuous dynamics are involved, it is nearly impossible to find an abstraction of the state space which fiducially captures exactly the state transitions that are possible in the original problem. Thus after generating an abstract state space, the options are to introduce transitions between abstract states A and B for which 1) there exists "some" original state a (that maps to A) that under some action will transition to some state b (that maps to B) 2) a "usually" version of the previous, or 3) "all" original states in A can transition under some action to all original states in B. Guaranteeing condition 3 is infeasible in practice, so some version of 1 or 2 is typically used.

The lack of a veridical representation causes the problems mentioned above with solving an abstract problem. In particular, the algorithms that are optimistic in defining the transitions have difficulties with narrow blocking walls that partition the reachable states within a state abstraction. Thus paths that appear feasible in the abstract problem turn out to be completely infeasible in the original problem. As a result these algorithm will fail to find an optimal solution.

Proposition 3: If A is an algorithm which solves an abstract problem with optimistic transition structure then $F(A, p(\epsilon)) = \infty$ and the feasibility time in general is inversely proportional to the width of the narrow blocking wall.
This problem affects many popular approaches for discrete state spaces including A^*.

The other possibility is making the algorithms pessimistic. In this case the algorithm will need to refine the discretization until a passage is found, which again may take a long time in $p(\epsilon)$.

D. Motion Planning

When using classic motion planning [4], [3] for a problem with dynamics, the common approach is to first solve an abstract problem without dynamics and then search for controls that move the robot along the solution path. This approach ignores the differential dynamics of the problem and can get stuck even with increased resolution. Hence, the proposed solution may not be feasible due to differential constraints of the problem. From this classical approach, the most popular for multiple query problems are Potential Field Methods [4] and Probabilistic Roadmaps [3].

The state-of-the-art algorithm that incorporates dynamics for a single query problem is the Rapidly exploring Random Trees (RRT) algorithm [5]. The main idea in RRT is to randomly pick states in the free space and extending the planning tree towards that state. The pseudo-code of RRT is shown in Figure 3.

We also assume that the given freespace sampling function f generates uniformly distributed states from the free space. We use rejection sampling to implement this sampler. However, sampling states from the free space can in general be expensive.

First, note that RRT is not constructing an abstract problem. Although RRT samples random states and tries to get to them, it *considers* a sub-goal for only one time-step (notice the difference between *considering* a sub-goal and *committing* to a sub-goal). Also, because RRT does not build the planning tree in an exhaustive fashion, it is able to efficiently handle the continuity in the action space. Compared to dynamic programming algorithms, a disadvantage of RRT is that the model is assumed to be deterministic.

Although RRT is guaranteed to find a feasible solution [6], the solution quality can be arbitrarily bad. Furthermore, the plain RRT algorithm can not use additional computational resources to improve the quality of its first solution.

```
Input: $x^s$ (starting state), $g$ (goal function), $m$ (a generative model), $\kappa$ (a setpoint controller), $\rho$ (a metric), $u(\mathcal{X}_1)$ (a sampler from the free space), $f$ (the free space indicator function).
Output: A solution trajectory.
Let $\mathcal{T}$ be a set of states, initialized by $\mathcal{T} = \emptyset$.
$\mathcal{T} = \mathcal{T} \bigcup \{x^s\}$.
while Goal Not Found do
  $X = u(\mathcal{X}_1)$.
  $Y = \text{argmin}_{Y' \in \mathcal{T}} \rho(Y', X)$.
  $A = \kappa(Y, X)$.
  $Z = m(Y, A)$.
  Add $Z$ as a child to $Y$.
  $\mathcal{T} = \mathcal{T} \bigcup \{Z\}$.
  if $g(Z) = 1$ then
    Goal is Found. Return the solution trajectory.
  end if
end while
```

Fig. 3. The RRT Algorithm

Proposition 4: Even with trajectory optimization, RRT is regretable for $p(\epsilon)$.

IV. RRT^{++}

Our proposed method, RRT^{++}, is an extension of RRT to an anytime algorithm that quickly finds a feasible solution, and incrementally improves this solution when given additional time. It will return solutions as quickly as the RRT algorithm. As time progresses, its solutions will approach the minimal cost.

The RRT^{++} algorithm is based on the observation that RRT is rapidly exploring the state space, but it needs to explore the path space. The planning tree has a number of branches. For optimality, one of these branches needs to be close to an optimal path. The difficulty is that a desirable branch may only be partially developed by the time the other branches are arriving at the goal.

For example, consider the problem in Figure 8(a) of moving from the starting state S to the goal G. There are two relevant branches in the map: b1 is following the optimal path and b2 is going through a sub-optimal route. During the execution of RRT, many attempts to extend b1 will fail because b1 needs to go through some narrow passages. Hence, b2 will develop and reach the goal faster than b1.

Consider the consequence of making RRT an anytime algorithm by continuing to grow the original search tree after the goal has been reached. If RRT tries to extend b1 by sampling more states, b2 will *absorb* samples near the goal and prevent b1 from growing towards the goal. In Figure 8(a), a sub-branch of b2 has blocked b1.

This argument suggests that it is futile to make RRT an anytime algorithm by continuing to add new samples to the tree. Another simple approach to improve solutions for an anytime algorithm is to run RRT repeatedly, starting with an empty tree for each run. We call this method Repeated RRT (R^{3}T). This algorithm is shown in Figure 4. Despite being simple, this approach has desirable properties. Unfortunately, this approach can be slow to converge on problems where RRT repeatedly finds poor paths.

```
Input: $x^s$ (starting state), $g$ (goal function), $m$ (a generative model), $\kappa$ (a setpoint controller), $\rho$ (a metric), $u(\mathcal{X}_1)$ (a sampler from the free space), $f$ (the free space indicator function), $c$ (a trajectory cost function), $i$ (computational interaction budget).
$C$ saves the total computation used.
$J^*$ saves the best trajectory found.
$b = \infty$.
while $C < i$ do
  Let $\mathcal{T}$ be a set of states, initialized by $\mathcal{T} = \emptyset$.
  $\mathcal{T} = \mathcal{T} \bigcup \{x^s\}$.
  $J = RRT(x^s, g, m, \kappa, \rho, u(\mathcal{X}_1), f)$.
  Update $C$.
  if $c(J) < b$ then
    $b = c(J)$.
    Store $J^* = J$ as the best solution.
  end if
end while
Return $J^*$.
```

Fig. 4. The Repeated RRT (R^{3}T) Algorithm

From the base RRT algorithm, R^{3}T also provides the fast feasible solutions. However, it is an asymptotic no-regret algorithm for the problem $p(\varepsilon)$, as there is some non-zero probability of selecting a sequence of points within an arbitrarily small window near the optimal path.

Proposition 5: R^{3}T is an asymptotic no-regret algorithm for $p(\epsilon)$.

Proposition 6: $\forall p,\ F(\text{R}^3\text{T}, p) = F(\text{RRT}, p)$

As a side note, despite similarities to R^{3}T , the Anytime RRT [1] algorithm is regretable for $p(\epsilon)$, as the algorithm can set an unattainable cost bound.

By destroying the entire search tree between runs, R^{3}T takes a crude approach to achieving improved plans. Another more delicate solution is to cut just some *bad* branch that reached the goal, and continue running RRT with the remainder of the search tree. The idea is to clear the space around the goal to let other branches grow. This approach raises the question of where the tree should be cut. Intuitively, we want to cut the tree at a node that forks into two or more large branches, such that one of these branches leads to the goal.

Assume there is a trajectory that connects the node x to the goal, and let $\mathcal{U}(\mathcal{T}, x)$ denote this trajectory. Let $\mathcal{Z}(\mathcal{T}, x)$ be all children of x that are not in $\mathcal{U}(\mathcal{T}, x)$ and let $D(\mathcal{T}, x)$ be the child of x that is in $\mathcal{U}(\mathcal{T}, x)$. Consider all trajectories $\mathcal{J} = \{x, x', \dots\}$, where $x' \in \mathcal{Z}(\mathcal{T}, x)$. Let $\mathcal{E}(\mathcal{T}, x)$ be the longest such trajectory.

For example, consider Figure 5. In this figure, some nodes ($D = D(\mathcal{T}, S)$, $D1$, $D2$, B, M, F) are shown with hollow circles and some trajectories ($\mathcal{U}(\mathcal{T}, S)$ and $\mathcal{E}(\mathcal{T}, S)$) are shown with dashed black lines. In this figure, $\mathcal{Z}(\mathcal{T}, S) = \{D1, D2\}$. Intuitively, we want to cut node $D(\mathcal{T}, S)$ and its descendants from the tree $\mathcal{T}$ and continue running RRT.

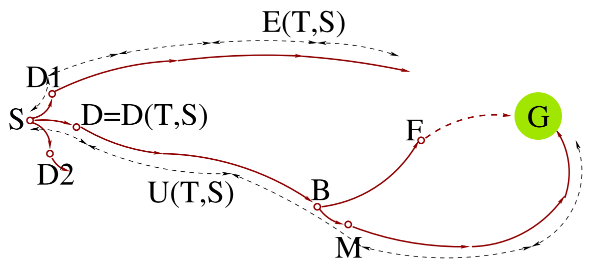

Fig. 5. A portion of the planning tree $\mathcal{T}$ after reaching a goal state. We indicate nodes on the tree with hollow circles, and trajectories with dashed lines. This simplified tree graph shows a child node $D(\mathcal{T}, S)$ of S, the trajectory that passes through this child to the goal $\mathcal{U}(\mathcal{T}, S)$, and the longest disjoint sibling trajectory $\mathcal{E}(\mathcal{T}, S)$.

We find a state x^* such that

$$x^* = \underset{x \in \mathcal{U}(\mathcal{T}, x^s)}{\operatorname{argmax}} |\mathcal{U}(\mathcal{T}, x)| + |\mathcal{E}(\mathcal{T}, x)|, \quad (1)$$

where $|\mathcal{J}|$ is the number of elements in trajectory $\mathcal{J}$. Then, we remove $D(\mathcal{T}, x^*)$ and its descendants from the tree $\mathcal{T}$ and continue running RRT.

This intuition is a heuristic and can fail. Consider Figure 5 again. Assume the optimal trajectory goes from S to B to F and then follows the dashed line. Then the right point to cut is M. But even if $|\mathcal{E}(\mathcal{T}, S)|$ was substantially smaller than what we have shown in this figure, the solution to Problem 1 is still $x^* = D(\mathcal{T}, S)$ (simply because $|\mathcal{U}(\mathcal{T}, S)|$ is large and dominates the optimization). Hence, if only a small improvement at the very end of $\mathcal{U}(\mathcal{T}, x^s)$ is possible, this heuristic probably cannot find it. Note that this problem doesn't exist in Figure 8(a).

We introduce a cyclic incremental fallback method to overcome this problem. Consider a cyclic counter $r = (\mod (c, N) + 1)/N$, where $N \in \mathbb{N}$ is a constant algorithm parameter and $c \in \mathbb{N}$ increases by one after each execution of RRT. Let $\mathcal{U}(\mathcal{T}, x^s) = \{x_0 = x^s, \dots, x_m\}$ be the solution trajectory found by running the last RRT algorithm. Then we consider only the descendants of $x' = x_{\lfloor (1-r)m \rfloor}$ (i.e. a $1-r$ fraction of the solution trajectory) for the cut operation:

$$x^* = \underset{x \in \mathcal{U}(\mathcal{T}, x')}{\operatorname{argmax}} |\mathcal{U}(\mathcal{T}, x)| + |\mathcal{E}(\mathcal{T}, x)|.$$

After cutting x^* and its descendants from $\mathcal{T}$, we increase c by one and run RRT again starting with the smaller tree. This procedure continues until our computational budget is ended. The pseudo-code for the RRT^{++} algorithm is shown in Figure 6.

Although RRT^{++} is designed to be better than R^{3}T in practice, it is not clear that the algorithm as described above has the no-regret property. We restore this property by including random restarts (with a tiny probability δ) to recover the no-regret property.

Proposition 7: $\forall p,\ F(\text{RRT}^{++}, p) = F(\text{RRT}, p)$

Proposition 8: RRT^{++} is a no-regret algorithm for $p(\epsilon)$.

The fact that random restarts allow RRT^{++} to trivially gain the no-regret property shows that the no-regret property can be considered weak. Note however, that this property is not so weak that it is easily achieved by all the algorithms considered earlier.

```
Input: $x^s$ (starting state), $g$ (goal function), $f$ (a generative model), $\kappa$ (a setpoint controller), $\rho$ (a metric), $N$ (number of cuts), $i$ (computational interaction budget), $\delta$ (reset probability).
$C$ saves the total computation used.
$c = 1$ is a counter.
$\mathcal{U}(\mathcal{T}, x) = \{x_0, x_2, \dots, x_L\}$, $E(\mathcal{T}, x)$, and $c^*(\mathcal{T}, x)$ are defined in the text.
Let $\mathcal{T}$ be the set of states, initialized by $\mathcal{T} = \emptyset$.
while $C < i$ do
    With probability $\delta$, $\mathcal{T} = \emptyset$.
    $\mathcal{T} = \text{RRT}(x^s, g, f, \kappa, \rho, \mathcal{T})$.
    Update $C$.
    Let $\mathcal{U}(\mathcal{T}, x^s) = \{x_0, \dots, x_m\}$ (so $m = |\mathcal{U}(\mathcal{T}, x^s)|$)
    $r = (\mod (c, N) + 1)/N$.
    $x' = x_{\lfloor (1-r)m \rfloor}$.
    $x^* = \text{argmax}_{x \in \mathcal{U}(\mathcal{T}, x^0)} |\mathcal{U}(\mathcal{T}, x)| + |\mathcal{E}(\mathcal{T}, x)|$.
    Cut $D(\mathcal{T}, x^*)$ and descendants from $\mathcal{T}$.
    $c = c + 1$.
end while
```

Fig. 6. The RRT^{++} Algorithm

V. Simulation Results

We use the RLAI Critterbot robot simulator [7]. We model a two wheeled robot situated in two 120×120 maps shown in Figure 2. The simulated robot has non-trivial dynamics including substantial inertia and limited torques. The mass is $1.1kg$ and the moment of inertia is $0.004217kg \times m^2$. For simplicity, we performed the collision detection only for the center of the robot, effectively transforming the robot's body to a point.

Let the top left corner of the map be the origin and the horizontal and vertical lines be the two coordinate axes. The state is a 4-dimensional vector $x = (x_1, x_2, x_3, x_4)$, where x_1 and x_2 are coordinates of the robot, and where $x_3 = \dot{x}_1$ and $x_4 = \dot{x}_2$ are velocities. The action is the torques applied to the robot wheels. Instead of using a setpoint controller to generate actions, we restricted action selection to one of three discrete actions. The metric ρ on the state space is the Euclidean distance.

For these experiments, the cost of a single step in a path is the two dimensional distance travelled by the robot. In the case of a collision, the cost is infinite (and the forward model returns the original state).

The starting state in the maps (marked with an S), and the green regions (marked with G) represent the goal regions. The maps are designed such that there are multiple paths from the start to a goal that are not homotopic. For each goal position, the shortest path has narrow passages that makes the planning problem difficult.

We compare the performance of RRT^{++} and R^{3}T on the problems in Figure 2. For the first map, we solve two separate planning problems with goal centers of G and H. In the second map, the start state is S and the center goal state is G. The radius of the goal region is $\epsilon = 4$ in both problems.

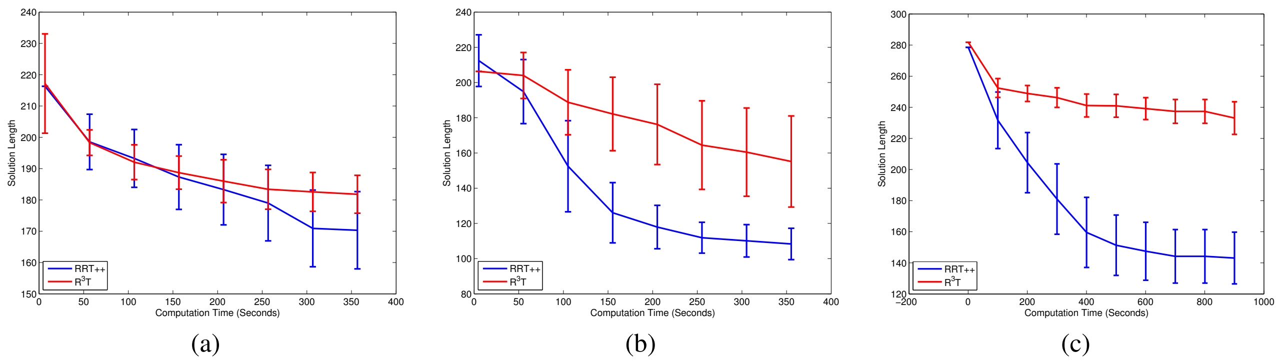

Fig. 7. In all these graphs, red is R^3T and blue is RRT^{++}. The vertical bars show 95% confidence intervals from ten runs. (a) The results for goal state G for the map in Figure 2(b). (b) The results for goal state H for the map in Figure 2(c). (c) The results for the cluttered map in Figure 2(c). These graphs show that RRT^{++} can find substantially improved plans.

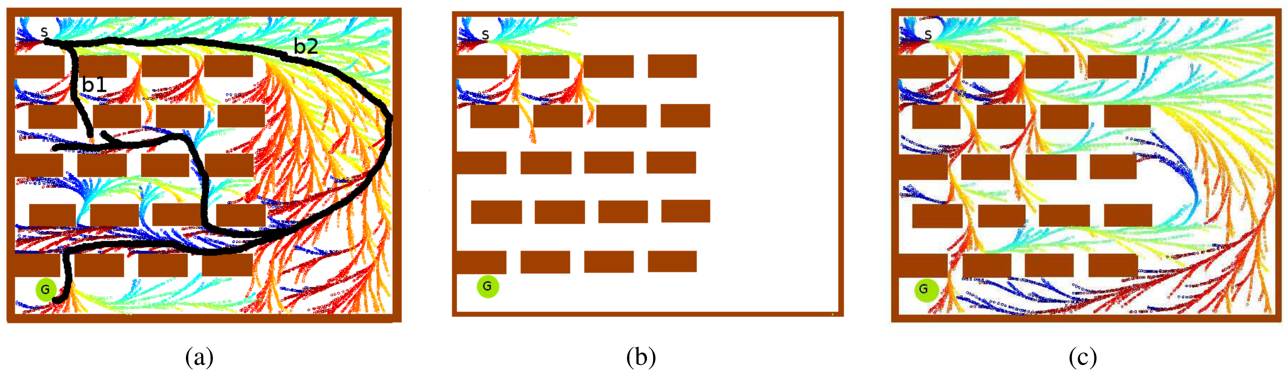

Fig. 8. The shape of the RRT^{++} planning tree (a) before a cut, (b) after a cut, and (c) on eventually finding a near-optimal solution. Different colors indicate different robot orientations. In (a) the solution is sub-optimal and the bad branch has blocked the good branch. In (b), the bad branch is removed from the tree. In (c), RRT^{++} has found a near-optimal solution because bad branches had been cleared away by earlier steps.

The cost of R^3T and RRT^{++} is the CPU time in seconds. The performance criteria is the solution length, which is the sum of the Euclidean distances between consecutive positions in the solution trajectory.

Figure 7 shows the performance of R^3T and RRT^{++} on the above problems. The results show that R^3T does not converge quickly towards the optimal solution, but RRT^{++} shows substantially better performance.

Figure 8 shows how RRT^{++} modifies the search tree. Figure 8(a) shows the search tree just before the cut operation. Trajectory b2 is the last solution trajectory and b1 is a near-optimal trajectory. Figure 8(b) shows the search tree after the cut operation. The cut point is very close to S and trajectory b2 is gone. Figure 8(c) shows the search tree when a solution after the cut operation is found. Trajectory b1 has reached to the goal. This trajectory is the same near-optimal trajectory that we have shown in Figure 1.

In separate experiments, we examined modifications suggested by other papers. We examined the use of heuristic biases to the search [8]. We also examined the idea of extending multiple nearby branches towards a node, as in the RRG algorithm [2]. In our experiments, both variants performed worse than the R^3T algorithm.

VI. Conclusion

We have introduced RRT^{++}, a new anytime extension of RRT that quickly generates feasible solutions, and asymptotically approaches optimal solutions. We introduced desirable properties for an anytime algorithm, and we showed that RRT^{++}has desirable properties. In addition to this theoretical structure, we provide empirical results that show RRT^{++} converges towards optimal solutions on problems where RRT reliably finds substantially suboptimal solutions. Moreover, we show that R^3T does not efficiently converge towards an optimal solution in a realistic cluttered environment. In comparison on the same problem, RRT^{++} is able to use additional computational resources to efficiently reduce costs to a near optimal solution.

References

[1] D. Ferguson and A. Stentz. Anytime RRTs. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.
[2] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Robotics: Science and Systems*, 2010.
[3] L. Kavraki, P. Svestka, J-C Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 1996.
[4] J-C Latombe. *Robot Motion Planning*. Kluwer Publishers, 1991.
[5] S.M. LaValle and J.J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.
[6] S.M. LaValle and J.J. Kuffner. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2001.
[7] RLAI. Critterbot simulator, 2009. http://critterbot.rl-community.org.
[8] C. Urmson and R. Simmons. Approaches for heuristically biasing RRT growth. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.