Proceedings of the 2006 IEEE/RSJ
International Conference on Intelligent Robots and Systems
October 9 - 15, 2006, Beijing, China

# Anytime RRTs

Dave Ferguson and Anthony Stentz
Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania
{dif,tony}@cmu.edu

***Abstract*— We present an anytime algorithm for planning paths through high-dimensional, non-uniform cost search spaces. Our approach works by generating a series of Rapidly-exploring Random Trees (RRTs), where each tree reuses information from previous trees to improve its growth and the quality of its resulting path. We also present a number of modifications to the RRT algorithm that we use to bias the search in favor of less costly solutions. The resulting approach is able to produce an initial solution very quickly, then improve the quality of this solution while deliberation time allows. It is also able to guarantee that subsequent solutions will be better than all previous ones by a user-defined improvement bound. We demonstrate the effectiveness of the algorithm on both single robot and multirobot planning domains.**

## I. Introduction

Rapidly-exploring Random Trees (RRTs) have been shown to be very effective for solving robotic path planning problems involving complex configuration spaces [1], [2], [3], [4], [5]. By combining random sampling of the configuration space with biased sampling around a desired goal configuration, RRTs efficiently provide solutions to problems involving vast, high-dimensional configuration spaces that would be intractable using discrete approaches.

However, while RRTs have been found to be extremely effective at generating feasible solutions, they provide no control over the quality of the solutions produced. In many situations, in particular when dealing with non-uniform cost search spaces, the difficulty of executing different solution paths may vary substantially. It is thus important that solution cost is taken into account during the search process.

Nevertheless, few efforts have been made to incorporate cost considerations into sampling-based planning algorithms. One notable exception is work by Urmson and Simmons [6], who developed a series of modified versions of the RRT algorithm that select tree nodes for expansion based on the cost of their current path from the initial node. Their approaches were able to produce less costly solutions through both uniform and non-uniform cost search spaces. However, as might be imagined, this improvement in solution quality usually came at a computational price: the approach that produced the best solutions required significantly more computation than the unmodified RRT algorithm.

For agents operating in the real world under time constraints, it is important both that high quality solutions are produced and that these solutions can be produced within the time available for planning. One class of planners well-suited to this task are anytime planners, which quickly find an initial, highly suboptimal plan, and then improve this plan until time runs out.

In this paper, we present a sampling-based anytime algorithm for generating solution paths through both uniform and non-uniform cost search spaces. Our approach works by generating a series of Rapidly-exploring Random Trees, where each tree reuses information from previous trees to improve its growth and the quality of its resulting path. The first tree is generated using an unmodified RRT algorithm to ensure that a valid solution is available in the minimum possible time. We also present a number of modifications to the Rapidly-exploring Random Tree algorithm that we use to bias the subsequent searches in favor of less costly solutions. The resulting approach is able to produce an initial solution very quickly, then improve the quality of this solution while deliberation time allows. It is also able to guarantee that subsequent solutions will be better than all previous ones by a user-defined improvement bound.

We begin by describing the basic RRT algorithm and extensions that have been made to this algorithm to improve the quality of the solutions produced. We then introduce Anytime Rapidly-exploring Random Trees and show how this approach can be used to generate less costly solutions through both uniform and non-uniform cost search spaces. We go on to present a number of results from a single robot navigation domain and a multirobot constrained exploration domain, and conclude with discussions and extensions.

## II. Rapidly-exploring Random Trees

The standard RRT algorithm for planning a path from some initial configuration $q_{start}$ to a goal configuration $q_{goal}$ is presented in Fig. 1. This algorithm initializes a search tree with the initial robot configuration as the root node, then incrementally grows the tree until the goal configuration is reached. To grow the tree, first a target configuration $q_{target}$ is randomly selected from the configuration space using the function *ChooseTarget*. Then, a *NearestNeighbor* function selects the node $q_{nearest}$ in the tree closest to $q_{target}$. Finally, a new node $q_{new}$ is created in an *Extend* function by growing the tree some distance from $q_{nearest}$ towards $q_{target}$. If extending the tree towards $q_{target}$ requires growing through an obstacle, no extension occurs. This process is repeated until the tree grows to within some user-defined threshold of the goal (line 3). A very nice property that follows from this method of

```
InitializeRRT(rrt T)
 1  T.add(q_start);
GrowRRT(rrt T)
 2  q_new = q_start;
 3  while (Distance(q_new, q_goal) > distance-threshold)
 4    q_target = ChooseTarget();
 5    q_nearest = NearestNeighbor(q_target, T);
 6    q_new = Extend(q_nearest, q_target, T);
 7    if (q_new ≠null)
 8      T.add(q_new)
ChooseTarget()
 9    p = RandomReal([0.0, 1.0]);
10    if (p < goal-sampling-prob)
11      return q_goal;
12    else
13      return RandomConfiguration();
Main()
14  InitializeRRT(tree);
15  GrowRRT(tree);
```

Fig. 1. The RRT Algorithm.

construction is that the tree growth is strongly biased towards unexplored areas of the configuration space. Consequently, exploration occurs very quickly. The RRT algorithm can also be made significantly more efficient if the search is focussed towards the goal. This can be achieved through using a goal bias (lines 9 to 11): with probability $1 - p$, $q_{target}$ is randomly selected; with probability $p$, $q_{target}$ is set to the goal configuration. As the value of $p$ is increased, the RRT behaves increasingly like best-first search.

As mentioned earlier, the original RRT algorithm does not take into account solution cost during the search. Thus, it can produce paths that are grossly suboptimal, particularly in non-uniform cost search spaces. To improve upon this, Urmson and Simmons presented three modified versions of the RRT algorithm that take cost into account when selecting nodes in the tree for extension [6]. They replaced the simple *NearestNeighbor* function (Fig. 1 line 5) with a function that found the $k$ nearest neighbors to the current $q_{target}$ point, then selected from these $k$ nodes either (1) the closest node $q_{nearest}$ to $q_{target}$, as long as an estimate of the cost of a path from $q_{start}$ through $q_{nearest}$ to $q_{goal}$ is less than some probabilistic threshold $r$, or (2) the first of the $k$ nodes (ordered by estimated path cost) whose current estimated path cost is less than $r$, or (3) the node with the minimum estimated path cost, as long as this cost was less than $r$. These 3 different selection methods resulted in 3 different algorithms. Of these, the third method produced the best overall solutions, but required more computation than the standard RRT algorithm.

We are interested in using RRTs for navigation of single agents and multi-agent teams in partially-known outdoor environments. Because our agents are acting in the real world, there may be situations where plans must be generated and executed extremely quickly. There may also be other situations where there is more time for deliberation and better plans are desired. The agents may not know a priori how much planning time is available and they certainly will not know how long it will take a particular algorithm to generate a solution. Thus, it is useful to generate a series of solutions and then employ the best of these solutions when an action has to be executed.

In the discrete planning community, there exist a number of efficient algorithms that can provide this performance [7], [8]. For example, the ARA* algorithm developed by Likhachev, Gordon, and Thrun works by creating an initial, highly-suboptimal solution by running an inflated A* search with a high suboptimality bound $\epsilon$, then improves this solution by repeatedly running new searches with decreasing values of $\epsilon$. After each search terminates, the cost of the most recent solution is guaranteed to be at most $\epsilon$ times the cost of an optimal solution.

In order to plan through complex, very high-dimensional configuration spaces (as might be encountered when planning for teams of agents), we would like a sampling-based analog to these discrete anytime algorithms. In the following section, we present a sampling-based anytime algorithm that efficiently constructs an initial solution using the standard RRT algorithm, then improves this result by generating a series of solutions, each guaranteed to be better than all the previous ones by some improvement factor $\epsilon_f$. These successive solutions are generated by using modified versions of the RRT algorithm that take into account cost—both of nodes in the current tree and of previous solutions—to influence the sampling of the search space, the selection of nodes in the tree for extension, and the extension operation itself.

## III. Anytime Rapidly-exploring Random Trees

Discrete anytime algorithms such as ARA* achieve their anytime performance by efficiently generating a series of solutions, where each solution is better than the previous ones. At any point in time the best solution found thus far can be returned, along with a suboptimality bound on the quality of this solution.

We can use this same basic idea to create a sampling-based anytime algorithm. We start out by generating an initial RRT without any cost considerations. We then record the cost $\mathcal{C}_s$ of the solution returned by this RRT. Next, we generate a new RRT and ensure that its solution is better than the previous solution by limiting the nodes added to the tree to only those that could possibly contribute to a solution with a lower overall cost than $\mathcal{C}_s$. We can also multiply our cost bound $\mathcal{C}_s$ by some factor $(1 - \epsilon_f)$, where $0 \leq \epsilon_f < 1$, to ensure that the next solution will be at least $\epsilon_f$ times less expensive than our previous solution. In such a case, $\epsilon_f$ is known as the solution improvement factor. We then update our cost bound $\mathcal{C}_s$ based on the cost of the new solution and repeat the process until time for planning runs out.

Such an approach guarantees that each solution produced will be better than all the previous solutions. However, it does not guarantee that new solutions will be able to be produced. In order for this approach to be truly effective we require dependable methods for generating each successive

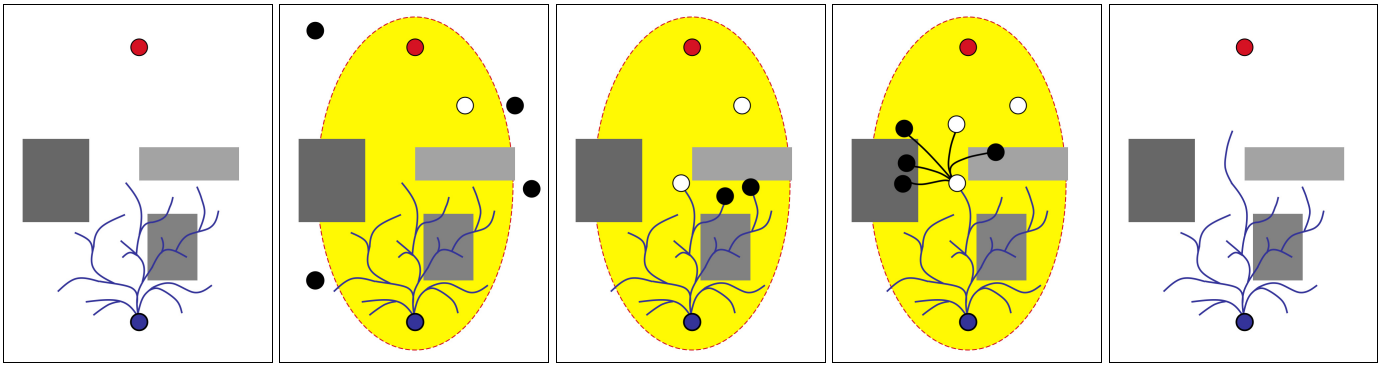



Fig. 2. Anytime RRT Planning. Given a partial RRT (shown in (a)) being grown from an initial configuration (the bottom filled circle) to a goal configuration (the top filled circle), this illustration shows how the Anytime RRT approach samples, selects, and extends the tree. To begin with, we assume some previous solution has already been generated. (b) When using the Anytime RRT approach to sample the configuration space, only areas that could potentially lead to an improved solution are considered (indicated by the shaded oval). Thus, the black nodes are rejected while the white node is accepted. (c) When selecting the next node in the tree to extend, the $k$ nodes closest to the sample point are found and ordered according to both their distance from the sample point and the cost of their path from the start node. Here, $k = 3$ and the white node and two black nodes are the closest; the white node is selected first since its path is less expensive than those of the two black nodes. (d) The tree node from (c) is then extended by generating a set of possible extensions and then probabilistically choosing the one that is least expensive. The cost of the extension to the white node is cheaper than the extensions to the black nodes, so the white node is chosen as the next element to be added to the tree. (e) After checking that the sum of the cost of the path from the start node through the tree to the new element and the heuristic cost of a path from the new element to the goal is less than the solution bound, the element is added to the tree.

low-cost solution. We rely upon novel node sampling, node selection, and node extension operations that incorporate cost considerations and variable bias factors to efficiently produce solutions satisfying a specified cost bound. We discuss each of these operations in turn.

### A. Node Sampling

If we are only interested in generating a solution that is cheaper than some upper bound value $\mathcal{C}_s$, then we can use this upper bound to influence the sampling process used by the RRT algorithm. Rather than randomly sampling the entire configuration space, we restrict our sampling to just those areas of the configuration space that could possible provide a solution satisfying the upper bound. Given a node $q_{target}$ in the configuration space, we can check whether $q_{target}$ could be part of such a solution by calculating a heuristic cost from the initial node $q_{start}$ to $q_{target}$, $h(q_{start}, q_{target})$, as well as a heuristic cost from $q_{target}$ to the goal node $q_{goal}$, $h(q_{target}, q_{goal})$. If these heuristic values do not overestimate the costs of optimal paths between these nodes then the combination of these heuristic values gives us a lower bound on the cost of any path from $q_{start}$ through $q_{target}$ to $q_{goal}$. If this lower bound cost is greater than our upper bound $\mathcal{C}_s$, then there is no way $q_{target}$ could be part of a solution satisfying our upper bound and so $q_{target}$ can be ignored.

However, depending on the complexity of the configuration space and what heuristics are used, this approach could make it very difficult for the RRT to find a solution. For example, if there are narrow passages in the configuration space between large obstacles, then it may be very difficult to sample nodes inside the passages. This is a well-known problem with the original RRT algorithm, but it could be exacerbated by disregarding any samples that fall inside configuration space obstacles. Depending on how the heuristic deals with such samples, the above approach could make it even more difficult for the tree to grow down any narrow passages. Further, by reducing our consideration of sample points to only those whose heuristic values are promising, we are in effect cutting off large chunks of the configuration space. This is entirely the point of restricting our sampling, but it can also introduce new narrow passages. For example, imagine an obstacle that resides near the edge of the promising configurations, as determined by our heuristic values. It may be possible to plan a path around this obstacle, but finding such a path may be difficult using our restricted sampling approach, as very few samples exist that will pull the tree towards this edge. Thus, it is important to use conservative heuristic estimates and not disregard points that reside in configuration space obstacles.

One way of implementing this restricted sampling idea is to continue generating random samples $q_{target}$ from the configuration space until we find one whose combined heuristic cost is less than our upper bound. This approach is illustrated in Fig. 2(b). Another method is to use the heuristic functions to do the sampling itself, so that every node sampled will satisfy the upper bound. On the whole, this restricted sampling technique saves us a lot of unnecessary computation spent on irrelevant areas of the configuration space.

### B. Node Selection

Once a sample node $q_{target}$ has been generated using the above technique, we then select a node from the tree $q_{tree}$ to extend out towards the sample node. In the original RRT algorithm, the closest node in the tree to $q_{target}$ is selected to be $q_{tree}$. However, as Urmson and Simmons show [6], much cheaper solutions can be obtained if we modify this selection process to incorporate cost considerations.

Our selection approach is based on their ideas but uses bias factors to vary over time the influence of cost, so that

```
ReinitializeRRT(rrt T)
  T.cleartree();
  T.add(q_start);
GrowRRT(rrt T)
  q_new = q_start; time = 0;
  while (Distance(q_new, q_goal) > distance-threshold)
    q_target = ChooseTarget(T);
    if (q_target ≠ null)
      q_new = ExtendToTarget(q_target, T);
      if (q_new ≠ null)
        T.add(q_new);
    UpdateTime(time);
    if (time > max-time-per-rrt)
      return null;
  return T.c(q_start, q_new);
ChooseTarget(rrt T)
  p = RandomReal([0.0, 1.0]);
  if (p < goal-sampling-prob)
    return q_goal;
  else
    q_new = RandomConfiguration();
    attempts = 0;
    while (h(q_start, q_new) + h(q_new, q_goal) > T.C_s)
      q_new = RandomConfiguration();
      attempts = attempts + 1;
      if (attempts > max-sample-attempts)
        return null;
    return q_new;
```

Fig. 3. The Anytime RRT Algorithm: GrowRRT and ChooseTarget Functions.

initially nodes are selected based purely on their distance from $q_{target}$, and in subsequent searches this gradually changes so that some combination of distance from $q_{target}$ and the cost of the node is considered, eventually leading to purely cost-based selection. We accomplish this by using a distance bias parameter $d_b$ and a cost bias parameter $c_b$. First, the $k$ nearest neighbor nodes in the tree to $q_{target}$ are computed. Next, these nodes are ordered in increasing node selection cost:

$$\textbf{SelCost}(q) = d_b \cdot \textbf{Distance}(q, q_{target}) + c_b \cdot c(q_{start}, q),$$

where $c(q_{start}, q)$ is the cost of the current path from $q_{start}$ to $q$ in the tree. We then process these nodes in order until one is found from which a valid extension can be made (the extension process is described below). Fig. 2(c) shows an illustration of the node selection process.

Initially, $d_b = 1$ and $c_b = 0$ and so our node selection operates exactly as a nearest neighbor lookup. Between each successful search, $d_b$ is reduced by some value $\delta_d$ and $c_b$ is increased by some value $\delta_c$, so that the cost of the tree nodes becomes increasingly important as time progresses. This has the effect of producing more costly solutions early on, when we are most concerned with ensuring that valid solutions are available, and then providing cheaper solutions if there is extra time available for planning.

```
SelCost(rrt T, configuration q, configuration q_target)
  return T.d_b · Distance(q, q_target) + T.c_b · T.c(q_start, q);
ExtendToTarget(rrt T, configuration q_target)
  Q_near = kNearestNeighbors(q_target, k, T);
  while Q_near is not empty
    remove q_tree with minimum SelCost(T, q_tree, q_target) from Q_near;
    Q_ext = GenerateExtensions(q_tree, q_target);
    q_new = argmin_{q∈Q_ext} c(q_tree, q);
    T.c(q_start, q_new) = T.c(q_start, q_tree) + c(q_tree, q_new);
    if (T.c(q_start, q_new) + h(q_new, q_goal) ≤ T.C_s)
      return q_new;
  return null;
Main()
  T.d_b = 1; T.c_b = 0; T.C_s = ∞;
  forever
    ReinitializeRRT(T);
    T.C_n = GrowRRT(T);
    if (T.C_n ≠ null)
      PostCurrentSolution(T);
      T.C_s = (1 − ε_f) · T.C_n;
      T.d_b = T.d_b − δ_d;
      if (T.d_b < 0)
        T.d_b = 0;
      T.c_b = T.c_b + δ_c;
      if (T.c_b > 1)
        T.c_b = 1;
```

Fig. 4. The Anytime RRT Algorithm: ExtendToTarget and Main Functions.

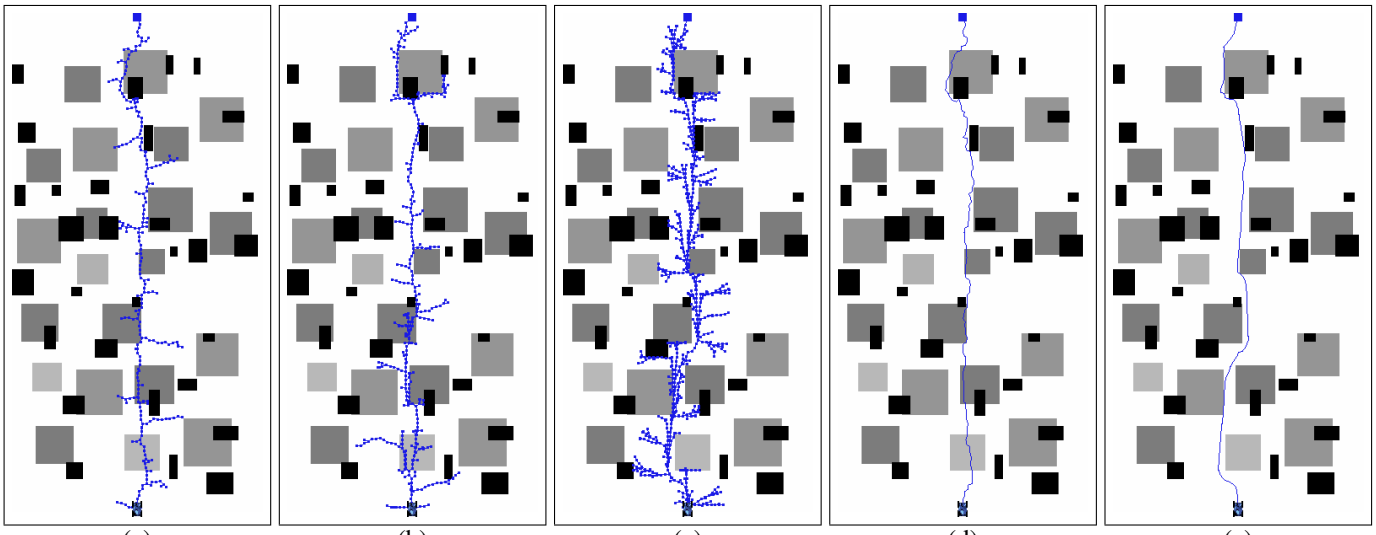
### C. Node Extension

When a node in the tree is selected for extension, we take into account the nature of the configuration space in its vicinity to produce a new, low-cost branch of the tree. There are two different approaches we use to do this. The first approach is to generate a set of possible extensions that lead from the tree node $q_{tree}$ in the general direction of the sample node $q_{target}$ and then take the cheapest of these extensions. This results in a new node $q_{new}$ which is added to the tree if it could potentially contribute to a solution satisfying our upper bound $\mathcal{C}_s$, i.e., if

$$c(q_{start}, q_{tree}) + c(q_{tree}, q_{new}) + h(q_{new}, q_{goal}) \leq \mathcal{C}_s,$$

where $c(q_{start}, q_{tree})$ is the cost of the current path from $q_{start}$ to $q_{tree}$ in the tree (as before), and $c(q_{tree}, q_{new})$ is the cost of the extension just constructed from $q_{tree}$ to $q_{new}$. If $q_{new}$ does not satisfy our solution bound then a new set of extensions that do not lead as directly to the sample node is considered. This process continues, with each subsequent set of extensions 'fanning out' further from the sample node, until either a $q_{new}$ is generated that satisfies our bound or some maximum number of attempts have been made.

The second approach is to use a large initial set of possible extensions and take the cheapest of these extensions. Again, we check that the corresponding new node $q_{new}$ satisfies our solution bound before adding it to the tree. This approach is



(a) (b) (c) (d) (e)

Fig. 5. Anytime RRTs used for single robot path planning. The start node is the vehicle at the bottom of the environment, the goal is the square at the top. Shaded regions represent higher-cost areas to traverse through; black regions represent obstacles. (a) Initial RRT generated without cost consideration. (b) Fifth RRT generated, using costs of previous solutions and nodes of the current tree to bias the growth of the current tree. (c) Final RRT generated. (d) Path corresponding to initial RRT from (a). (e) Path corresponding to final RRT from (c). These images correspond to the results presented in Fig. 7(a).

illustrated in Fig. 2(d). Because the first of these approaches is generally faster and the second produces less costly solutions, both are useful in our anytime framework: the first can be used to efficiently produce RRTs at early stages of our planning, while the second can be used to produce later trees with less costly solutions. It can also be beneficial to include some randomness in the extension operation: with some probability a random extension operation is selected and tested against our solution bound.

Pseudocode of the basic Anytime RRT approach is given in Figures 3 and 4. For space and simplicity we have not included all the features mentioned in the previous discussion. In particular, our actual implementation begins by generating a standard RRT and uses this to provide an initial solution and bound, and the **ExtendToTarget** function is modified over time to switch between our different extension approaches. In the pseudocode, **UpdateTime**($time$) increments an elapsed time counter, **kNearestNeighbors**($q_{target}, k, T$) returns the $k$ closest nodes in the tree $T$ to a sample node $q_{target}$, **GenerateExtensions** generates a set of extension operations (in the manner discussed earlier) and returns the nodes at the endpoints of these extensions, and **PostCurrentSolution** publishes the current solution so that it can be executed if deliberation time runs out. The plaintext identifiers (e.g. 'num-neighbors') are constants. Other identifiers (e.g. $d_b$) are as described earlier, prefixed with the tree identifier $T$ to illustrate that they are variables associated with the tree.

Because the Anytime RRT approach uses a solution bound to influence the growth of each RRT, it has a number of nice properties. We include the major ones here; proofs of these can be found in [9]. Firstly, because the algorithm restricts nodes added to the tree based on the solution bound $\mathcal{C}_s$, it is guaranteed not to produce any solution whose cost is greater than $\mathcal{C}_s$.

**Theorem 1.** *When the solution bound is $\mathcal{C}_s$, the cost of any solution generated by the Anytime RRT algorithm will be less than or equal to $\mathcal{C}_s$.*

By updating the solution bound each time a new path is generated, the algorithm is also able to guarantee that the next solution will be better than the previous one, by at least our user-defined solution improvement factor $\epsilon_f$.

**Theorem 2.** *Each time a solution is posted by the Anytime RRT algorithm, the cost of this solution will be less than $(1-\epsilon_f)$ times the cost of the previous solution posted.*

Together, these properties ensure that solutions produced by the Anytime RRT algorithm improve over time and that the rate of improvement is at least as good as the solution improvement factor $\epsilon_f$.

**Corollary 1.** *If $\epsilon_f > 0$ then the solutions posted by the Anytime RRT algorithm will have associated costs that are strictly decreasing. Moreover, these costs will decrease by at least a factor of $\epsilon_f$ between each successive solution.*

## IV. Experiments and Results

The primary motivation behind this work was efficient multirobot path planning in non-uniform cost environments. In particular, we are interested in constrained exploration, where a team of robots is tasked with exploring an environment while abiding by some constraints on their paths [10]. For instance, imagine an environment containing areas through which no communication is possible (due to eavesdropping adversaries or environmental characteristics), and the team must maintain line-of-sight communication at all times. RRTs are useful for

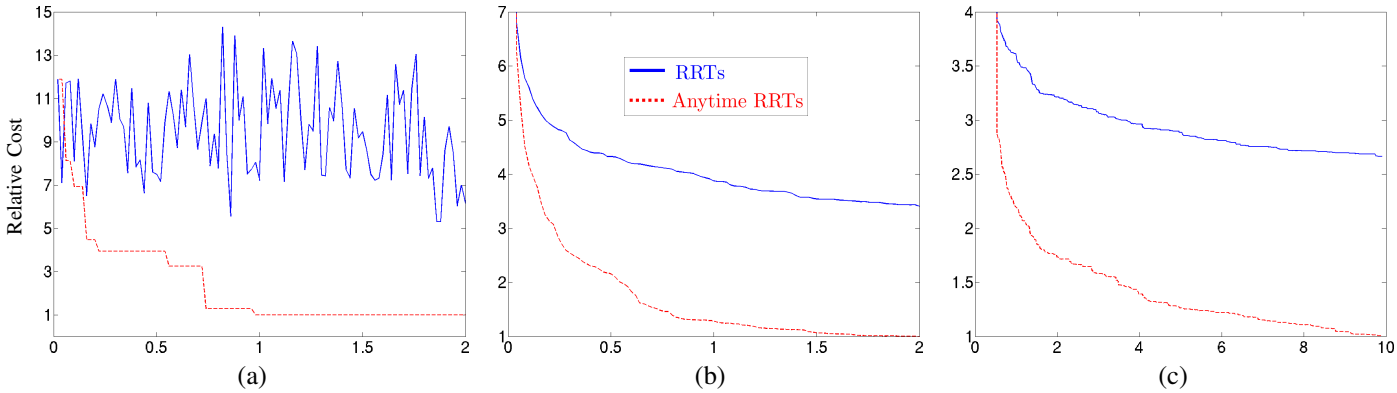



Fig. 7. Anytime RRT results. (a) Example results from a single run in one of the single robot environments. Shown are the RRT solution costs generated as time progresses - the regular RRT approach does not use previous solution costs to influence the growth of future trees. (b,c) Average relative solution cost as a function of time for our (b) single robot, and (c) three robot runs. The minimum-cost solution generated by each of the approaches (relative to the best overall solution generated) is presented at each point in time.

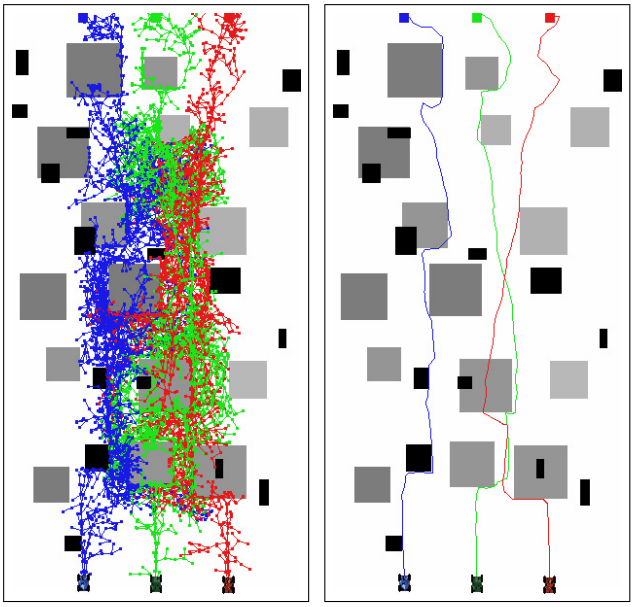


Fig. 6. Anytime RRTs used for multirobot constrained exploration. On the left is one of the trees produced by our anytime approach. On the right is the corresponding path.

solving this problem because they cope well with the very high dimensional configuration space.

To analyse the performance of Anytime RRTs, we generated two different sets of experiments. In the first, we planned paths for a single agent across a $300 \times 600$ non-uniform cost environment in which there was a set of randomly placed obstacles and a set of randomly placed, random cost areas (see Fig. 5 for an example such environment). We compared the solutions generated by Anytime RRTs to those generated by a series of standard RRTs. Each approach was allowed to run for a total time of 2 seconds (on a 1.5 GHz Powerbook G4), and each individual RRT was allowed to take up to $0.5$ seconds. We repeated this process for $100$ different environments. Fig. 7(b) plots the average cost of the best solutions generated versus planning time for these experiments. These results have been normalized: for each run, the best solution produced by the Anytime RRT approach was recorded and all solution costs were then computed relative to this cost.

Our second set of experiments had a similar setup, except that we planned joint paths for a team of 3 agents and maintained a line-of-sight communication constraint over the entire path. Again, we randomly generated $100$ different non-uniform cost environments, and the obstacles in the environment acted as both navigation obstacles (i.e., no agent could have its path intersect any of these obstacles) and communication obstacles (i.e., if the direct line between two agents intersected one of these obstacles at any point in the agents' paths, line-of-sight communication was broken between these agents). We allowed the planner $10$ seconds of planning time, and each individual tree was allowed up to $2$ seconds. Results from this second set of experiments are shown in Fig. 7(c); as with the first experiment, these results have been normalized.

For both these experiments, the Anytime RRT approach began by growing a standard RRT using the standard nearest neighbor and extension operators. Then, it slowly decreased $d_b$ by $\delta_d = 0.1$ each iteration and increased $c_b$ by $\delta_c = 0.1$. It switched its extension operator from the former approach mentioned in Section III-C to the latter approach after the third solution was found. $\epsilon_f$ was set to $0.1$ so that each successive solution was guaranteed to be $10\%$ less costly than the previous solution. For our heuristics we used Euclidean distance and our available extensions were straight-line segments for each agent.

There are a couple points worth noting from the results of these experiments. Firstly, both the regular RRT approach and the Anytime RRT approach both start with the same solution (the graphs start from the same point in the upper left), so that both are able to provide an initial, valid solution in the minimum possible time. This is important for occasions where the available planning time may turn out to be very small and an agent has to act quickly.

Secondly, the Anytime RRT approach produces much better solutions, and is able to improve upon initial solutions much more quickly than the regular RRT approach. Overall, the best solutions generated by the regular RRT approach at the end

of the maximum time allowed for planning were on average 3.6 and 2.8 times more expensive than the corresponding Anytime RRT solutions for the single agent and multi-agent cases, respectively. But the intermediate solutions produced by Anytime RRTs were also much better: Anytime RRTs were able to quickly and continually reduce the solution cost so that at any point in time they provided less costly solutions.

To provide a more detailed look at the behavior of each approach during a single run, Fig. 7(a) shows the results for a single environment in our single agent planning scenario. Here, we have plotted the most recent solution cost versus time over the course of planning. This graph shows the costs of the solutions generated by the Anytime RRT approach strictly decreasing over time, while the solutions produced by the regular RRT approach vary widely in cost and do not exhibit any general improvement. The environment from which these results came, along with some of the trees and solutions produced by the Anytime RRT approach, are illustrated in Fig. 5.

## V. Discussion

One of the most significant limitations of sampling-based planners to date has been their inability to provide high quality solutions or even bounds on the suboptimality of the solutions generated. In this work, we have presented techniques that are very effective at biasing sampling-based planners in favor of better solutions. These techniques are extremely useful for producing low cost solutions in non-uniform cost search spaces. Further, we have shown how these techniques can be incorporated into an anytime sampling-based planner that not only improves its solution over time, but can provide (and accept) bounds on the quality of this improvement.

Our anytime approach generates a series of RRTs, each producing a new solution that is guaranteed to be less expensive than the previous solution by a user-defined improvement factor $\epsilon_f$. Thus, a valid solution is returned as quickly as by the standard RRT algorithm, but the quality of this solution is then improved while deliberation time allows. The resulting algorithm provides similar benefits as recently-developed discrete anytime algorithms, but is able to plan over much larger, higher-dimensional search spaces. We have provided key properties of the algorithm including relative bounds on the quality of the solutions generated, and have demonstrated its effectiveness for both single agent navigation and multi-agent constrained exploration.

We are currently investigating a number of extensions to this work. Firstly, it may be possible to exploit even more information from previous solutions to aid in the generation of new ones, such as low cost branches of previous RRTs or extracted knowledge concerning the nature of the configuration space (e.g. as in [11] but with cost considerations).

It is also worth further investigating how heuristics can be most effectively used to focus the growth of the trees. For example, over the course of our experiments we have found that when the heuristic is not very informed, inflating the heuristic values of nodes close to the root can prevent the RRTs from growing into 'dead ends', where no new nodes can be added because the early nodes were too expensive. Using information from previous searches and the current search to improve the heuristic estimates may be an even more effective approach. Finally, we are implementing the approach on a team of autonomous John Deere E-Gator vehicles for outdoor constrained exploration.

## VI. Acknowledgements

This work was sponsored by the U.S. Army Research Laboratory, under contract "Robotics Collaborative Technology Alliance" (contract number DAAD19-01-2-0012). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies or endorsements of the U.S. Government. Dave Ferguson is supported in part by a National Science Foundation Graduate Research Fellowship.

## References

[1] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[2] ——, "Rapidly-exploring Random Trees: Progress and prospects," *Algorithmic and Computational Robotics: New Directions*, pp. 293–308, 2001.

[3] M. Kobilarov and G. Sukhatme, "Time Optimal Path Planning on Outdoor Terrain for Mobile Robots under Dynamic Constraints," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[4] J. Kim and J. Ostrowski, "Motion planning of aerial robots using Rapidly-exploring Random Trees with dynamic constraints," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[5] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Motion planning for humanoid robots," in *Proceedings of the International Symposium on Robotics Research (ISRR)*, 2003.

[6] C. Urmson and R. Simmons, "Approaches for heuristically biasing RRT growth," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2003.

[7] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems*. MIT Press, 2003.

[8] R. Zhou and E. Hansen, "Multiple sequence alignment using A*," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2002, Student abstract.

[9] D. Ferguson and A. Stentz, "Anytime RRTs: The Proofs," Carnegie Mellon School of Computer Science, Tech. Rep. CMU-RI-TR-06-07, 2006.

[10] N. Kalra, D. Ferguson, and A. Stentz, "Constrained Exploration for Studies in Multirobot Coordination," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[11] B. Burns and O. Brock, "Toward optimal configuration space sampling," in *Proceedings of Robotics: Science and Systems (RSS)*, 2005.