Proceedings of the 2003 IEEE/RSJ
Intl. Conference on Intelligent Robots and Systems
Las Vegas, Nevada · October 2003

Approaches for Heuristically Biasing RRT Growth

Chris Urmson & Reid Simmons

The Robotics Institute, Carnegie Mellon University, Pittsburgh, USA
{curmson, reids}@ri.cmu.edu

Abstract

This paper presents several modifications to the basic rapidly-exploring random tree (RRT) search algorithm. The fundamental idea is to utilize a heuristic quality function to guide the search. Results from a relevant simulation experiment illustrate the benefit and drawbacks of the developed algorithms. The paper concludes with several promising directions for future research.

1 Introduction

Rapidly-exploring random trees (RRT) have been shown to provide an efficient method for solving planning problems with kino-dynamic constraints [5]. Frazzoli adapted RRTs for real time planning for an autonomous helicopter operating in the presence of moving obstacles [3]. More recently, Bruce adapted RRTs for use with a robotic soccer team [2]. In both of these examples, there is only a binary evaluation of whether space is free or impassable. In many applications, however, it is important to consider a continuum of costs between completely free space and impassable obstacles. This is particularly relevant in the case of planning for outdoor mobile robots where terrain characteristics vary greatly. Algorithms such as D* [6] account for terrain variability but they require a discretization of the state and action space which may be infeasible for some problems.

This paper provides a method for biasing the growth of an RRT based on costs discovered through the exploration of the space. Doing so provides extra information to the algorithm that allows it to operate in more than just an exploratory manner. Our techniques show improvements over the basic RRT algorithm in the quality of solutions found in both binarized and continuous cost spaces.

Section 2 introduces the basic RRT algorithm. Section 3 describes a variety of enhancements that we have explored and section 4 provides an empirical comparison between these improved algorithms and the basic algorithm. The paper concludes with some insights and promising future research directions discovered through this work.

```
BUILD_RRT ( x_init )
T.init ( x_init );
While ( x_goal ∉ T ) do
        { x_rand, x_near } ← SELECT_NODE(T);
        EXTEND ( T, x_rand, x_near );
Return T;
```

```
SELECT_NODE ( T )
        x_rand ← RANDOM_STATE();
        x_near ← NEAREST_NEIGHBOR( x_rand, T);
return { x_rand, x_near };
```

```
EXTEND( T, x_rand, x_near)
if ( NEW_STATE( x_rand, x_near, x_new, u_new ) ) then
        T.add_vertex( x_new );
        T.add_edge( x_near, x_new, u_new);
return;
```

***Figure 1:** The RRT Algorithm.*

2 The Basic RRT algorithm

The rapidly-exploring random tree planner is an incremental search algorithm that provides benefits over conventional roadmap planners [4] due to the inherent feasibility of the solutions generated. An outline of the algorithm is shown in Figure 1. The RANDOM_STATE function draws its samples from a uniform distribution spread over the problem space. NEW_STATE takes this random sample (x_{sample}) and a node to be extend (x_{near}), and generates an action from the node towards the sample (u_{new}) and the resultant state (x_{new}). If u_{new} is not executable, NEW_STATE returns *false*.

The elegance of an RRT based search is that it uses randomization to implicitly compute the Voronoi regions of each of the nodes in a tree, and then selects a node to expand with preference proportional to the size of its Voronoi region. This technique causes the tree to rapidly explore the space. In essence, the tree is probabilistically pulled outward from its root.

In [5], LaValle and Kuffner mention that shaping the probability distribution used to generate random samples for growing the tree effects the rate of convergence of the algorithm. Adding the goal state to the distribution with a higher probability dramatically increases the rate of convergence of the planner. The

```
SELECT_NODE(T)
do
        x_rand ← RANDOM_STATE();
        x_near ← NEAREST_NEIGHBOR(x, T);
        m_quality ← 1- (x_near.cost –T.opt_cost ) /
              (T.max_cost-T.opt_cost);
        m_quality ← min ( m_quality, T.prob_floor );
        r ←RANDOM_VALUE();
 while ( r > m_quality);
return { x_rand, x_near };
```

```
EXTEND( T, x_rand, x_near )
if (NEW_STATE ( x_rand, x_near, x_new, u_new ) ) then
        T.add_vertex ( x_new );
        T.add_edge ( x_near, x_new, u_new);
        CALCULATE_COST ( x_new );
        T.max_cost ←max(x_new.cost, T.max_cost);
return;
```

***Figure 2:** The hRRT algorithm.*

principal effect is that goal biasing pulls the tree in the general direction of the goal state. A second and equally important effect is that when the tree nears the goal state, there is an increased likelihood of it extending to the goal state. Without goal biasing, the tree may come close to the goal, but through happenstance, fail to achieve it.

3 Improving the RRT

One weakness of the basic RRT algorithm is that it does not take path cost into account. This can lead to solutions that are far from optimal (in particular, in cases where the search space has a continuum of costs ranging from free to impassable). Our work is predicated on the idea that providing the algorithm with some knowledge of path cost, better solutions will be found.

To guide the growth of the randomized tree, the algorithm should preferentially extend lower cost paths heading towards the goal, while maintaining a reasonable bias towards exploration. To this end, we developed a modification to the RRT algorithm that provides a probabilistic implementation of heuristic search concepts. For convenience, this algorithm will be referred to as heuristically-guided RRT (hRRT) search.

The hRRT shapes the probability distribution based on the growth of the tree. The distribution is shaped to make the likelihood of selecting any particular node dependent on both the size of its Voronoi region (a bias toward exploration) and the quality of the path to that node (a bias toward exploiting known good parts of the space). To accomplish this, an additional measure, $m_{quality}$, is computed:

$$m_{quality} = 1 - \frac{(C_{vertex} - C_{opt})}{(C_{max} - C_{opt})}$$

C_{vertex} represents the total cost associated with a node (i.e. the sum of the integrated cost along the path to this node and the estimated cost from this node to the goal state). C_{opt} is the estimated total cost of the optimal path from the initial state to the goal state and is computed using the distance metric for the problem. C_{max} holds the value of the highest total cost for any node considered thus far. The numerator indicates how much worse a path through this node is expected to be, in comparison to the optimal path. The denominator provides a scaling coefficient to ensure that the weight falls within the range [0, 1].

The quality measure is used to weight the Voronoi region associated with each node. The weighting is applied by modifying the EXTEND and SELECT_NODE functions of the basic RRT algorithm as shown in Figure 2. This algorithm generates random states until a *good* nearest neighbor is probabilistically chosen. The likelihood of selecting a node for extension is proportional to both the size of its Voronoi region and its quality measure. A floor value is introduced to ensure that the search is not overly biased against exploration. If the floor value is set to 1, this algorithm becomes the basic RRT search algorithm.

Incorporating this quality measure results in a dramatic improvement in the cost of paths found when the cost function for the space is not binary. It even provides a modest improvement in the quality of the paths returned in the case of binary costs. A more detailed discussion of the performance improvements are provided with the experimental comparisons presented in section 3.

The hRRT algorithm generally works well, but can sometimes encounter situations that result in undesired behavior. In the case illustrated in Figure 3, vertex A is near an obstacle and has been extended into a high cost region. In this situation, the desired behavior is that A should expand into the narrow passageway between the obstacle and the high cost region, and exploration from B should cease. Unfortunately, this frequently does not happen because B's Voronoi region is much larger than A's. The probability of selecting B relative to A is given by:

$$\frac{\|vor(B)\| p_{Bquality}}{\|vor(A)\| p_{Aquality}}$$

This means that any time the ratio of the sizes of A and B's Voronoi regions is larger than the probability floor value, B will be more likely to be selected than A. This observation led to an investigation of algorithms that try to expand one of the k-nearest neighbors, not just the single nearest neighbor.

3.1 k-Nearest Algorithms

By considering the k-nearest neighbors, rather than only the single nearest neighbor, regions of the tree are chosen for expansion rather than particular nodes. In essence, the combined area of the Voronoi regions of a set of points is used to compute the probability that biases towards expansion. To implement these algorithms, only the SELECT_NODE function needs to be modified.

IkRRT

The *iterative* k-nearest neighbor algorithm (IkRRT) iterates over the k-neighbors in order of their quality measure (see Figure 4). Once a node is sufficiently good to pass the probabilistic quality test, it is selected and the extension process continues. Note that if k is equal to 1, this algorithm is equivalent to the hRRT algorithm. By considering a set of nodes, rather than the single closest node, the determining factor in cases similar to those illustrated in Figure 3 is the quality measure. Thus the likelihood of A being expanded (as desired) is increased.

BkRRT

A variant method is the *best* of k-nearest neighbors algorithm (BkRRT). Pseudo-code for this algorithm is shown in Figure 5. In this algorithm, only the node of the k-nearest neighbors with the best quality measure is considered. If this node fails the quality test, a new random sample is drawn and the process repeats. This algorithm is much more aggressive in searching for lower cost solutions since it only considers the best node in any region of the tree. The downside of this approach is that the algorithm tends to over-explore regions of the space.

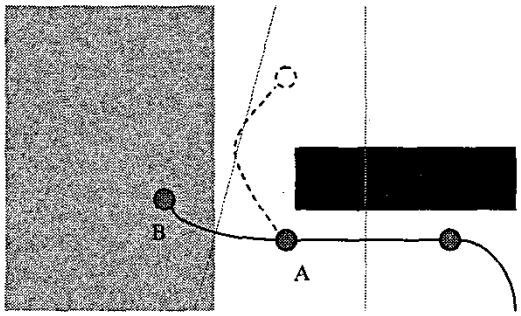

***Figure 3:** A difficult case for the hRRT algorithm to handle correctly.*

```
SELECT_NODE(T)
do
    x_rand ← RANDOM_STATE();
    K_near ← NEAREST_NEIGHBOR(x_rand, T, k);
    SORT_BY_QUALITY(K_near)
    for each x in K_near
        m_quality ← 1- (x.cost –T.opt_cost ) /
                    (T.max_cost-T.opt_cost);
        m_quality ← min(m_quality, T.prob_floor );
        r ← RANDOM_VALUE();
        if (r < m_quality )
            return { x_rand, x };
while true;
```

***Figure 4:** Select node function for iterative k-nearest algorithm*

```
SELECT_NODE (T)
do
    x_rand ← RANDOM_STATE();
    K_near ← NEAREST_NEIGHBOR(x_rand, T, k);
    x_near ← BEST_QUALITY( K_near )
    m_quality ← 1- (x_near.cost –T.opt_cost ) /
                (T.max_cost-T.opt_cost)
    m_quality ← min( m_quality, T.prob_floor );
    r ← RANDOM_VALUE();
while ( r>m_quality );
return { x_rand, x_near };
```

***Figure 5:** Select node function for best of k-nearest algorithm*

4 Experimental Results

This section provides some experimental results showing the relative performance of these algorithms in a variety of test cases. The long term goal of this work is to apply randomized planning to the field of outdoor mobile robot navigation in difficult terrain. To that end, a velocity controlled massive particle is used as a simplified, yet relevant, model for these experiments. The particle is actuated by magnitude-limited forces applied to it in any direction. The control input is a desired direction. This control input is fed to a PID controller that actuates the forces applied to the particle. This model contains constraints similar to that of a robot moving over terrain: high cost regions act as an analogy for sloped terrain, while the inertial properties of the

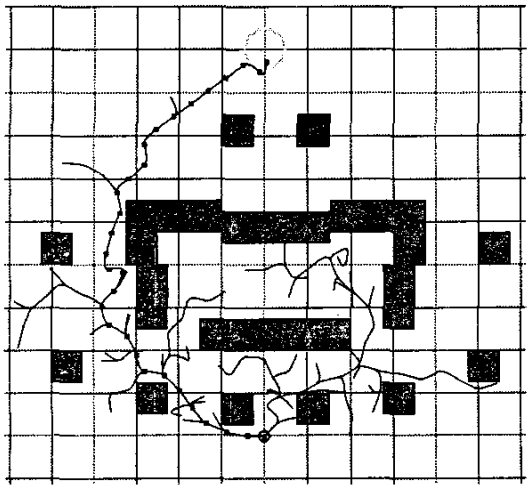

Figure 6: An hRRT grown in the first experiment.

particle and control space mirror those of a mobile robot.

The performance of the basic RRT algorithm is compared with the three algorithms presented here. All experiments were performed on a Pentium III 900 MHz computer with 256MB of memory with results representing an average over a set of 100 trials. The k-nearest neighbor algorithms were tested with a k of 5. In the first experiment, we consider an environment where the configuration space is binarized, i.e. the cost for moving through any part of the free space is the same.

4.1 Binary Space

Figure 6 shows the binary space in which this experiment was performed. Table 1 shows the performance of the various algorithms.

Algorithm	Avg. Cost	Avg. # of nodes in T	Avg. Time (s)
RRT	18.07	421	0.376
5% goal bias	17.28	174	0.138
10% goal bias	17.24	138	0.111
hRRT	16.84	372	0.528
5% goal bias	16.02	202	0.263
10% goal bias	15.66	167	0.228
IkRRT	16.28	962	1.392
5% goal bias	15.81	575	0.726
10% goal bias	15.74	521	0.670
BkRRT	14.34	882	2.544
5% goal bias	14.12	691	1.933
10% goal bias	14.14	644	1.831

***Table 1:** A comparison of various algorithms in a binary free space.*

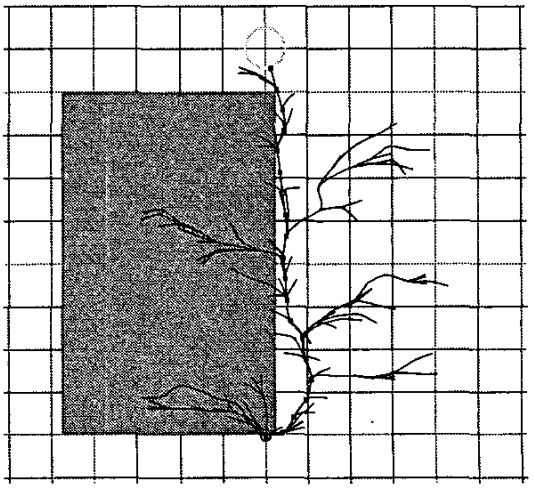

Figure 7: An IkRRT grown in the second experiment.

The basic hRRT algorithm shows a modest improvement in the quality of solution found with a minor increase in computational cost. Both of the k-nearest neighbor algorithms improve the solution but are more costly to compute. In all cases, the addition of a small amount of goal biasing reduces the computation time and in general, improves the solution.

4.2 Variable Cost Plane

The next experiment compares the algorithms in an open plane. A portion of the plane has an increased cost (the grey area is ten times more expensive to move through than the white area). Figure 7 shows the space considered in this experiment. Table 2 shows the performance of the various algorithms.

Algorithm	Avg. Cost	Avg. # of nodes in T	Avg. Time (s)
RRT	71.67	439	0.391
5% goal bias	63.12	113	0.074
10% goal bias	64.89	80	0.051
hRRT	23.89	292	0.276
5% goal bias	27.22	116	0.088
10% goal bias	25.49	85	0.061
IkRRT	16.30	562	0.555
5% goal bias	17.86	268	0.211
10% goal bias	17.42	193	0.171
BkRRT	13.25	432	0.454
5% goal bias	14.21	211	0.171
10% goal bias	14.39	179	0.140

***Table 2:** A Comparison of algorithms in a variable cost plane.*

When operating in a space with a non-uniform cost, the hRRT shows a dramatic improvement over the basic RRT algorithm in terms of path quality, while maintaining a similar computational cost. The *k*-nearest neighbor algorithms show a significant improvement over the hRRT algorithm but are more computationally expensive.

An important observation to make about this data is that the addition of a goal bias somewhat reduces the quality of the solution while dramatically improving the rate of convergence. The goal biasing essentially pulls the tree through the expensive regions, to some degree overriding the heuristic biasing.

4.3 Variable Cost Space with Obstacles

In this experiment the environments from experiments one and two are overlaid, constructing a variable cost space with obstacles embedded in it. The resulting environment is shown in Figure 8. Table 3 shows how the various algorithms compare in this space.

Algorithm	Avg. Cost	Avg. # of nodes in T	Avg. Time (s)
RRT	75.79	406	0.361
5% goal bias	79.93	163	0.129
10% goal bias	79.68	136	0.111
hRRT	25.21	243	0.250
5% goal bias	23.83	131	0.127
10% goal bias	24.82	124	0.126
I*k*RRT	18.93	732	0.910
5% goal bias	26.37	577	0.701
10% goal bias	29.79	511	0.614
B*k*RRT	18.56	569	0.910
5% goal bias	17.96	434	0.667
10% goal bias	18.18	429	0.689

***Table 3:** A comparison of the algorithms in a variable cost space with obstacles.*

Once again, the hRRT finds better paths than the basic RRT. An interesting observation is that the hRRT requires less computation time than the RRT, when no biasing is provided. This performance increase results because the hRRT generally avoids exploring the left (high cost) portion of the plane, thus reducing the size of the search space. This effect can also be seen in the dramatic reduction in computation time for the I*k*RRT and B*k*RRT algorithms (as compared to experiment 1).

4.4 Experimental Summary

From these experiments, it can be seen that heuristic guidance provides a significant improvement in the quality of paths produced through an RRT like search.

In all cases, utilizing the hRRT algorithm instead of the basic RRT algorithm produces statistically significant improvements in the quality of solutions found. In the binary obstacle case, the RRT requires less computation than the hRRT but in the other cases the computational cost of the hRRT is lower. The I*k*RRT and B*k*RRT algorithms improve further on the quality of results returned, but require a statistically significant increase in computational cost. Finally, the addition of a small amount of goal bias to the heuristically guided algorithms provides a dramatic improvement in the rate of convergence with only a small, often statistically insignificant decrease in the quality of the solution. Statistical significance was determined by testing the hypothesis that two means were equal, using a 5% significance level.

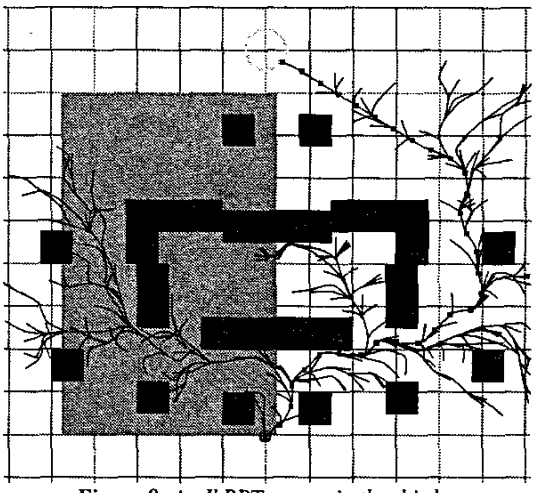

***Figure 8:** An I*k*RRT grown in the third experiment.*

5 Insights and Future Work

In working with the RRT algorithms, we have begun to develop an understanding of several important factors that must be considered when applying RRT-like algorithms.

Shape of the Probability Distribution

One of the most important components of an RRT based planner is the underlying probability distribution. For example, in the problem shown in Figure 9, if there is no probability mass to the right of the problem space, an

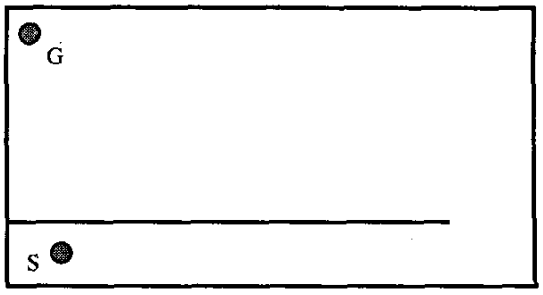

***Figure 9:** A problem where the distribution must have probability mass to the right of the space.*

RRT based algorithm will take a long time to find a solution. Similarly, without goal biasing, the likelihood of achieving the goal state will be very low unless there is additional probability mass above and to the left of the problem space. The reason for this is that without these additional probability masses, there is little, or no, area in which the random samples could be generated to pull the tree beyond the wall and towards the goal state.

Given this observation, it is important to carefully consider the shape and bounds of the probability distribution applied to solve any particular problem.

Exploitation vs. Exploration

A fundamental trade-off in any search problem is how much effort should be spent exploring the space vs. how much effort should be used to exploit, or greedily search the space. This work attempts to strike a balance between these two competing desires. Improvements may come from providing a better estimate for the true cost between the initial state and the goal state. Work has begun on a variation of these algorithms that propagates an expected cost back through the tree using an update function similar to:

$$C_i = P_i + \frac{1}{N_{children}+1}\left(\sum_{children}\left(P_{ij}+C_j-P_j\right)+G_i\right)$$

Where C_i is the total cost of a node, P_i is the cost of the path to the node, P_{ij} is the incremental path cost between node i and node j, and G_i is the optimistic estimate of cost to the goal for node i.

Each time a state is added to the tree, this function is applied recursively to the nodes in the path from the new state to the root. When this data reaches the root, C_{opt} is updated accordingly. Information about the problem space is thus used to adjust the heuristic cost guiding the search. At this point, the idea has shown limited success. The principal difficulty at this point is how to incorporate knowledge of failures to extend the tree. This information is important as it provides an understanding of the difficulty of the problem space.

Randomization vs. Determinism

In a recent paper [1], Branicky et al. argued convincingly for the use of pseudo random sampling in favor of randomized sampling in probabilistic roadmap planners [4]. The results of the Branicky work cannot be applied directly in the case of RRT-like planners, since the randomization is actually being used to perform an implicit computation. Even so, the arguments in this paper are cause for pause and force a careful consideration of the role of randomization in path planning. We are beginning investigation of a hybrid planner that will utilize a deterministic search to grow the initial components of a tree and then switch to an RRT like algorithm if the initial deterministic search stalls.

Planning in Unknown Environments

Instead of solving an entire planning problem, the RRT could be used to plan to a fixed horizon. An iterative planning strategy would use an RRT like algorithm to generate a feasible trajectory in the general direction determined by a low cost planner. This approach holds promise in domains where the search space is too large to search quickly at full resolution, or in cases where complete knowledge of the search space is unavailable. Initial experiments in this area have shown some promise. We will continue exploring this idea in experiments with an outdoor mobile robot moving through unknown difficult terrain.

6 Conclusions

In this paper we have presented extensions to the basic RRT algorithm that incorporate heuristic functions to bias the search towards low-cost solutions. These algorithms generate better paths in both binarized and continuous cost problem spaces. The heuristic guidance is applied by weighting the probability density function used to build the random tree. The hRRT algorithm shows significant improvements in the quality of paths found with little to no computational cost. The *k*-nearest neighbor variants show further improvement in the quality of paths found, but are more computationally expensive.

Acknowledgments

This work was partially funded by NASA under contract #1229340. The authors would like to thank James Kuffner, Steven LaValle and Alonzo Kelly for valuable discussions.

References

[1] M. Branicky et al. "Quasi-Randomized Path Planning", Proc. IEEE ICRA, Seoul, Korea, Mat 2001.

[2] J. Bruce & M. Veloso. "Real-Time Randomized Path Planning for Robot Navigation", Proc. IEEE/RSJ IROS, 2002.

[3] E. Frazzoli, et al. "Real-Time Motion Planning for Agile Autonomous Vehicles", AIAA Journal of Guidance, Control, and Dynamics, Volume 25, Issue 1, 2002.

[4] L. Kavraki et al. "Probabilistic Roadmaps for path planning in high dimensional configuration spaces", IEEE Transactions on Robotics & Automation, Vol. 12, Issue 4, August 1996.

[5] S. LaValle & J. Kuffner. "Randomized Kinodynamic Motion Planning", IJRR, Vol. 20, No. 5 pp 378-400, May 2001.

[6] A. Stentz, "Optimal and Efficient Path Planning for Partially-Known Environments", Proc. IEEE ICRA, May 1994.