Blind RRT: A probabilistically complete distributed RRT — Source link

BLIND RRT: A PROBABILISTICALLY COMPLETE, DISTRIBUTED RRT

An Undergraduate Research Scholars Thesis

by

CESAR ADOLFO RODRIGUEZ VILLANUEVA

Submitted to Honors and Undergraduate Research
Texas A&M University
in partial fulfillment of the requirements for the designation as

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Research Advisor: Nancy Amato

May 2013

Major: Computer Engineering

TABLE OF CONTENTS

ABSTRACT

Blind RRT: A Probabilistically Complete, Distributed RRT. (May 2013)

Cesar Adolfo Rodriguez Villanueva
Department of Computer Science
Texas A&M University

Research Advisor: Dr. Nancy Amato
Department of Computer Science

Rapidly-Exploring Random Trees (RRTs) have been successful at finding feasible solutions for high-dimensional problems. With motion planning becoming more computationally demanding, we turn to parallel motion planning for efficient solutions. Existing work on distributed RRTs has been limited by the overhead that global communication requires. A recent approach, Radial RRT, demonstrated a scalable algorithm that subdivides the space into regions to increase the locality of the computations. However, if an obstacle completely blocks RRT growth in a region, the planning space is not covered and thus planning problems cannot always be solved. We present a new algorithm, Blind RRT, which ignores obstacles during initial growth to efficiently explore the entire space. Because obstacles are ignored, free components of the tree become disconnected and fragmented. Thus, Blind RRT merges parts of the tree that have become disconnected from the root. We show how this algorithm can be applied to the Radial RRT framework allowing both scalability and usefulness in motion planning. We show this method to be a probabilistically complete approach to parallel RRTs. We show that our method not only scales, but also overcomes the motion planning limitations that Radial RRT has in a series of difficult motion planning tasks. The results show Blind RRT as a scalable strategy capable of effectively covering the space.

DEDICATION

I dedicate this thesis to my family and friends whose support was always invaluable for the completion of this work.

ACKNOWLEDGMENTS

This thesis would not have been possible without the support of many people. I would like to thank Dr. Nancy Amato for her valuable advice before, during and after the completion of the thesis; with her assistance I was always in the right path to success. I also wish to thank my mentors, Jory Denny and Sam Ade Jacbos, whose help was always valuable and without their guidance, the thesis would have faced many difficulties. Thank to Dr. Shawna Thomas for helping restructure and guiding the elaboration of this thesis.

This research supported in part by NSF awards CNS-0551685, CCF-0833199, CCF-0830753, IIS-0917266, IIS-0916053, EFRI-1240483, RI-1217991, by NSF/DNDO award 2008-DN-077-ARI018-02, by NIH NCI R25 CA090301-11, by DOE awards DE-FC52-08NA28616, DE-AC02-06CH11357, B575363, B575366, by THECB NHARP award 000512-0097-2009, by Samsung, Chevron, IBM, Intel, Oracle/Sun and by Award KUS-C1-016-04, made by King Abdullah University of Science and Technology (KAUST). This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

CHAPTER I
INTRODUCTION

Motion planning has been an active research area in the field of robotics. It also has many other applications including computer graphics [1], virtual reality [2], computational biology [3], and computer-aided design [4]. Problems in motion planning range from simple 2-D scenarios to high complexity robots with many degrees of freedom (DOF). Given that exact solutions are intractable for $DOF \geq 5$ [5], sampling-based motion planning has emerged as a promising technique for solving high-dimensional problems [6, 7].

Within sampling-based motion planning, we find two main approaches which most strategies follow: graph-based approaches, e.g., Probabilistic Roadmaps (PRM) [6], and tree-based approaches, e.g., Rapidly-Exploring Random Trees (RRT) [7]. PRMs perform two stages: the learning phase, which constructs a mapping (roadmap) of the planning space (configuration space ($\mathcal{C}_{space}$) [8]) encoding valid paths; and the query phase, where extracts a valid path from the roadmap using a simple graph search. RRTs iteratively grow one or more trees from a given configuration towards random configurations. In each iteration, a robot configuration q_{rand} is chosen as an expansion direction, and the nearest node in the tree q_{near} to q_{rand} is selected and expanded towards q_{rand}. Construction stops when the tree reaches the goal.

Even though sampling-based strategies are quite efficient, the computational costs, in terms of time and resources, of obtaining a solution are still expensive, especially in environments where the ratio of obstacle space, $\mathcal{C}_{obst}$, to the free space, $\mathcal{C}_{free}$, is high. Parallel motion planning embraces the notion that these kinds of problems can be solved with one or more processes collaboratively yielding a more efficient solution. One such method, Radial RRT [9], radially decomposes $\mathcal{C}_{space}$ into conical regions with the origin at the initial configuration and grows an RRT independently in each region. After this, branches are joined to adjacent regions through connected component connection techniques. Finally, cycles are removed so the resulting graph is a tree. This method scales well, but lacks the benefits of being probabilistically complete, which makes it impractical for many motion planning applications. Intuitively, an obstacle could entirely block an RRT's ability to grow in a region disallowing coverage of $\mathcal{C}_{free}$.

In this work, we present an RRT variant, Blind RRT, which ignores obstacles during the initial tree construction and retains all invalid and valid vertices and all valid edges. In this way, Blind RRT expands regardless of encountering obstacles or not. After initial construction, a post processing step is performed where invalid nodes are removed and unconnected pieces of the tree are merged back to the root. We apply this method in parallel to create a probabilistically complete parallel algorithm, both scalable and effective in covering the space.

Contributions of this work include:

- A novel method, Blind RRT, capable of exploring $\mathcal{C}_{free}$ regardless of the $\mathcal{C}_{obst}$ to $\mathcal{C}_{free}$ ratio.
- Application of Blind RRT with radially subdivided RRT to provide both scalability and usefulness in motion planning.
- Experimental analysis of the scalability of our strategy and the effectiveness in finding paths measured through tree coverage. We compare our results to RRT and Radial RRT.

The results in this thesis have been submitted for publication [10].

Thesis Organization

The chapter organization of this thesis is as follows:

- Chapter II contains the relevant previous work as well as the preliminary content of motion planning.

- Chapter III introduces a novel method Blind RRT, and explains its process in detail.
- Chapter IV presents Radial Blind RRT, the application of Blind RRT to Radial RRT to provide both scalability and usefulness in motion planning.
- Chapter V contains the experimental analysis of the scalability of our strategy and the effectiveness in finding paths measured through tree coverage.

CHAPTER II
PREVIOUS WORK

In this section, we present preliminary definitions and related motion planning techniques that are relevant to this thesis. We discuss strategies that use obstacle witnesses to increase the quality of the roadmap and present previous work in parallel motion planning focusing on the parallelization of RRTs.

Preliminaries

Motion planning is the problem of finding a valid path (e.g., collision-free) taking a movable object (e.g., a robot) from a start configuration to a goal configuration in an environment [11]. A configuration of a robot is described by its n *degrees of freedom* ($DOFs$), each corresponding to an object component (e.g., a position and orientation). The space of all possible configurations, feasible or not, is called the configuration space ($\mathcal{C}_{space}$) [8]. $\mathcal{C}_{space}$ is comprised into two sets: $\mathcal{C}_{free}$, all feasible configurations, and $\mathcal{C}_{obst}$, all infeasible configurations. Thus, motion planning becomes the problem of finding a continuous trajectory in $\mathcal{C}_{free}$ from a start configuration to a goal configuration. Exact solutions become intractable for high-dimensional spaces as it is difficult to explicitly compute $\mathcal{C}_{obst}$ boundaries [5]. However, the feasibility of configurations can be determined quite efficiently using collision detection (CD) tests in the workspace, the robot's natural space.

Sampling-based Motion Planning

Because of the intractability of motion planning, sampling-based methods, which find approximate solutions, are both successful and effective in practice [6, 7]. Probabilistic Roadmaps

(PRMs) [6] and Rapidly-Exploring Random Trees (RRTs) [7] are two common approaches to sampling-based motion planning. Whereas PRMs construct a graph representing the connectivity of $\mathcal{C}_{free}$, RRTs iteratively grow a tree rooted at a start configuration and towards unexplored areas of $\mathcal{C}_{space}$, described in Algorithm 1. In each iteration, RRT generates a random configuration, $q_{rand} \in \mathcal{C}_{space}$, and then it identifies the configuration q_{near} in the tree that is nearest to q_{rand}. q_{near} is expanded towards q_{rand} at a distance of Δq to generate a new configuration q_{new}. q_{new} is added to the tree as well as an edge between q_{near} and q_{new}. RRT iterates until a stopping criteria is met, e.g., a query is solved. RRT-Connect [12] is a variant that greedily grows two trees, usually rooted at the start and the goal, towards each other until a connection between them is found.

Algorithm 1 RRT

```
Input: Configuration $q_{root}$ and expansion step $\Delta q$
Output: Tree $\tau$
1: $\tau \leftarrow q_{root}$
2: while $\neg done$ do
3:     $q_{rand} \leftarrow$ RandomCfg()
4:     $q_{near} \leftarrow$ NearestNeighbor($\tau, q_{rand}$)
5:     $q_{new} \leftarrow$ Expand($q_{near}, q_{rand}, \Delta q$)
6:     UpdateTree($T, q_{near}, q_{new}$)
7: end while
8: return $\tau$
```

Often, narrow passages are difficult for sampling-based motion planning. Many approaches that solve this problem find samples on or near the $\mathcal{C}_{obst}$ boundary. Obstacle-based PRM (OBPRM) [13] samples configurations near the obstacles by pushing configurations from $\mathcal{C}_{obst}$ to $\mathcal{C}_{free}$. Uniform OBPRM performs the validation of random lines in $\mathcal{C}_{space}$ to find obstacle boundaries. Gaussian PRM [14] and Bridge Test PRM [15] attempt inexpensive tests to find configurations near obstacles or contained in narrow passages. Toggle PRM entirely maps both $\mathcal{C}_{obst}$ and $\mathcal{C}_{free}$, retaining witnesses from failed connections in one space to augment the quality of the opposite space's maps. These ideas inspire the algorithmic process presented here.

Parallel Motion Planning

As mentioned before, the computation required for motion planning problems has motivated work on parallel motion planning. Early parallel motion planning was aimed at the discretization of $\mathcal{C}_{space}$, but it was limited to low dimensional problems [16, 17]. PRM was first parallelized in [18]. A scalable Parallel PRM was presented in [19] where $\mathcal{C}_{space}$ is subdivided, each processor independently constructs a roadmap in a region, and the resulting roadmaps are connected in a global connection phase. One of the first parallel RRTs built multiple trees at the same time, concurrently exploring $\mathcal{C}_{space}$ and returning once a process finds a connection to the goal [20]. Three different distributed RRT algorithms are presented in [21]. The first is a message-passing implementation where a process sends an end-signal when the goal is found. The second allows processes to build the tree collaboratively by communicating when a new node and edge are found. The last parallelization uses the manager-worker pattern, where the workers perform collision detection on edges assigned by the manager. The bottle-neck of this approach is the unbalanced work that the manager(s) may need to perform while workers wait idly to receive their task.

Radial RRT [9] achieves scalability by introducing $\mathcal{C}_{space}$ subdivision to RRTs. Radial RRT divides $\mathcal{C}_{space}$ into conical regions with a common origin, the root, and builds an RRT in each of them. These trees are then connected to trees in neighboring regions and cycles are pruned. While Radial RRT has almost linear scalability, it may fail to successfully explore $\mathcal{C}_{space}$ and find valid paths, lacking probabilistic completeness (as shown in Section IV). This is due to the fact that an obstacle can completely prevent a region from expanding its tree, leaving a portion of the space, that may contain a valid path to the goal, unexplored.

CHAPTER III
BLIND RRT

In this section, we describe the design, motivation and advantages of Blind RRT compared to the standard RRT. Although used in this work to improve Radial RRT, we present Blind RRT as a probabilistically complete strategy for motion planning, capable of solving problems independently of parallel computation. The motivation behind Blind RRT is the incapability of expansion for Radial RRT when an obstacle completely blocks progress in a region. Therefore, we propose to ignore obstacles, or blindly expand through them. Blind RRT takes advantage of the rapid expansion rate of RRTs, i.e., growing towards unexplored areas of all of $\mathcal{C}_{space}$.

Algorithm

The Blind RRT strategy, shown in Algorithm 2, starts by iteratively expanding a tree τ rooted at a configuration q_{rand}, similar to RRT. We alter the standard RRT `Expand` subroutine to continue growing through obstacles recording a set of configurations Q_{new} that occur during an expansion step. These witnesses are added to τ in the `Update` function. If valid edges exist between successive nodes in Q_{new}, these edges are added as well. Note that at this point, a Blind RRT has the same nodes as an RRT in an obstacle free environment and the RRT edges that are valid for the given environment. After performing N Blind RRT iterations (N_{br}) of expansion, Blind RRT deletes all invalid nodes in τ and performs a connection phase for the various connected components (CCs) of the tree. Following this, all CCs other than the CC containing the root are deleted. τ is returned.

Blind Tree Expansion. We describe two alternatives when performing blind expansion. Note that other RRT expansion algorithms could be modified and used appropriately. The

Algorithm 2 Blind RRT

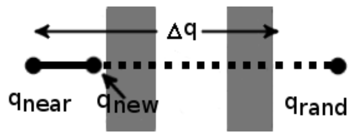
Input: A root configuration q_{root}, the initial number of nodes N_{br}, a maximum expanding distance Δq

Output: A tree τ containing N_{br} nodes rooted at q_{root}

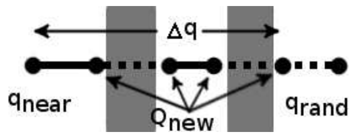
```
1: $\tau \leftarrow \{q_{root}\}$
2: for $n = 1 \ldots N_{br}$ do
3:     $q_{rand} \leftarrow$ RandomNode()
4:     $q_{near} \leftarrow$ NearestNeighbor($\tau, q_{rand}$)
5:     $Q_{new} \leftarrow$ Expand($q_{near}, q_{rand}, \Delta q$)
6:     $\tau$.Update($q_{near}, Q_{new}$)
7: end for
8: $\tau$.DeleteInvalidNodes()
9: ConnectCCs($\tau$)
10: $\tau$.DeleteInvalidCCs()
11: return $\tau$
```

first performs validity checking for the entire line from q_{near} to q_{new} (either at a distance Δq from q_{near} towards q_{rand} or q_{rand} itself, whichever is closer), collecting nodes that are valid along the boundary of $\mathcal{C}_{obst}$ (Figure III.1(b)). The second stops at the first validity change, records the valid node, and directly jumps to q_{new} and validates it (Figure III.1(c)). The latter skips part of the collision detection, while the former keeps track of more valid nodes along the expansion. The first expansion method retains all witnesses along an expansion step, which collects more configurations in narrow passages, while the second expansion method more rapidly jumps to Δq to avoid extraneous cost. Contrast these to regular expansion (Figure III.1(a)) which stops at the first obstacle, a distance of Δq towards q_{rand} or q_{rand} itself, whichever is closest. It is important to note that nodes contained in $\mathcal{C}_{obst}$ may be added to the tree if they are found Δq away from q_{near}, but only edges between valid configurations are added to the tree.

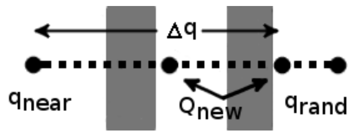
Connected Component Connection. At the end of the first step, any obstacles found along the expansion may have caused parts of τ to become disconnected from the root, yielding multiple CCs in τ. However, we would like to only have one CC in τ to find a path from the root to any node within τ. For this, we attempt to connect pairs of CCs, CC_a and CC_b, using RRT-Connect where $\tau_a = CC_a$ and $\tau_b = CC_b$ (refer to [12] for a detailed



(a) RRT Expand



(b) Blind RRT Expand 1



(c) Blind RRT Expand 2

Fig. III.1.: RRT Expand expands greedily up to Δq, q_{rand}, or an obstacle is hit (a). Blind RRT Expand always expands up to Δq distance or q_{rand} while also retaining either all free witnesses (b) or only the first free witness (c) to return a set of expansion nodes Q_{new}.

description of RRT-Connect). In the connection step, we first choose CC_a as a random source CC, and then choose a target CC, CC_b, by one of the following criteria:

- *Random CC.* This method is a naive and simplistic approach to avoid any nearest neighbor queries. Although we save time getting the pair of CCs, it can be ineffective in connecting CCs.
- *The CC whose centroid is closest to the centroid of CC_a.* A nearest neighbor query is performed between the centroid of CC_a and the centroids of the other CCs. It significantly reduces the nearest neighbor computation, as the number of CCs is much smaller than the number of nodes in the tree. This is used as an approximation scheme in selecting the closest CC.
- *The CC with the closest node to the centroid of CC_a.* We chose the CC which minimizes the distance between its closest node to the centroid of CC_a and the centroid of CC_a. This method is a slightly more costly approximation scheme in determining the closest CC to CC_a.
- *The CC whose centroid is closest to a node in CC_a.* Here, we calculate the centroid of all other CCs, find the nearest node in CC_a to each CC, and select the CC which minimizes this distance. It is similar to the previous approach in that it is again a cheap approximation compared with an overall closest node pair query.
- *The CC containing the closest node to a node in CC_a.* We choose the CC which minimizes the distance between the closest node pair between CC_a and the candidate CC. This is computationally expensive but accurate form of determining the closest CC to CC_a. As such, it can increase the possibility of a successful connection.

Component connection iteratively selects CCs to connect to until either one CC is achieved or a maximum number of failures is reached. If after the CC connection phase the algorithm was unable to connect all CCs to the root, we proceed to delete any CCs that are disconnected from the root.

Probabilistic Completeness

Probabilistic completeness is a desirable property of randomized planners which refers to their ability to find a solution path, assuming one exists, as the number of samples tends to infinity. In this section, we describe and prove the probabilistic completeness of Blind RRT. In the following proof, we assume that the $\mathcal{C}_{space}$ is ϵ-good [22] for some $\epsilon > 0$.

Theorem III.0.1 *Blind RRT is probabilistically complete.*

Proof Given any two configurations q_s and q_g in the same connected component of $\mathcal{C}_{free}$, a path exists between q_s and q_g. If no obstacles are present in the environment, i.e., $\mathcal{C}_{free} \equiv \mathcal{C}_{space}$, then an RRT rooted at q_s will reach within ϵ of q_g after n_0 fixed step expansions of distance Δq. (i.e., a path exists in the tree between q_s and q_g.) n_0 Blind RRT expansions are also sufficient to reach within ϵ of q_g. This is due to the fact that Blind RRT explores $\mathcal{C}_{space}$ identically to an RRT grown in the absence of obstacles because Blind RRT expansions ignore $\mathcal{C}_{obst}$.

After Blind RRT removes invalid nodes of the tree, q_g exists in some component of the tree CC_g. If $CC_s \equiv CC_g$, where CC_s is the component of the tree containing q_s, then a path exists in the tree between q_s and q_g. If $CC_s \not\equiv CC_g$, then Blind RRT uses RRT-Connect to merge CC_g with CC_s. It follows from the probabilistic completeness of RRT-Connect [12] that Blind RRT will connect CC_g to CC_s to yield a valid path between q_s and q_g. $\blacksquare$

CHAPTER IV
AN IMPROVED RADIAL RRT USING BLIND RRT

In this section, we introduce an improved Radial RRT framework for distributed RRTs which uses Blind RRT as a subroutine, shown in Algorithm 3.

Algorithm

Radial Blind RRT starts by radially subdividing $\mathcal{C}_{space}$ as in Radial RRT [9]. It makes use of a region graph, which is an abstraction of the different subdivisions of the space we are handling. To construct the region graph, the algorithm randomly samples N_r points Q_{N_r} on a d dimensional hypersphere of radius r centered at q_{root}, where d is the dimension of $\mathcal{C}_{space}$, r is a bound on the growth of the region, and q_{root} is the root configuration of the RRT. These samples become the vertices of the region graph, each representing a region. Then, a k-closest connection routine determines the adjacency of the regions defining the edges of the region graph.

Radial Blind RRT, shown in Algorithm 3, constructs a Blind RRT in parallel in each region. Each region construct a tree of N_{br}/p nodes whose growth is bounded to the region, where N_{br} is input as the number of nodes for the tree and p is the number of processing elements. Most likely, the result will be that each region will contain several CCs that need to be connected back to the root component of the tree. This takes place in a global region connection phase.

The region connection phase is described in Algorithm 4 which attempts to connect CCs from neighboring regions. The neighboring regions found from the region graph allow for reducing the global communication between processing elements, thus improving scalability of the approach. Prior to the region connection phase, a minimum spanning tree of the region graph

Algorithm 3 Radial Blind RRT

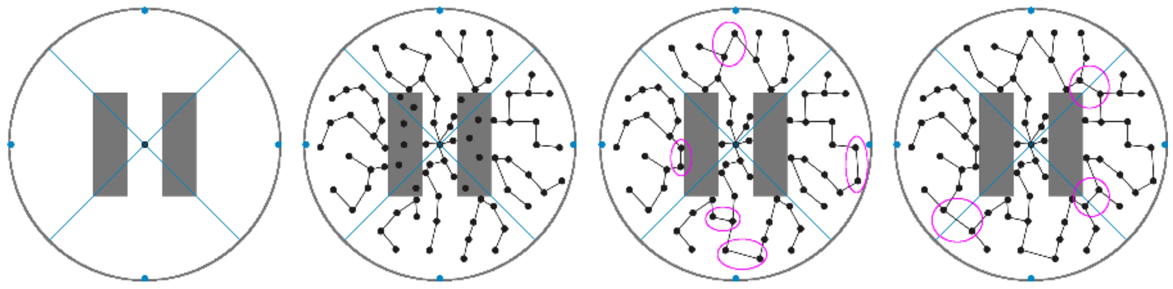
```
Input: A root configuration $q_{root}$, the number of nodes $N_{br}$, a maximum expansion distance
    $\Delta q$, the number of processors $p$, the number of regions $N_r$, a region radius $r$, the number
    of adjacent regions $k$
Output: A tree $\tau$ containing $N_{br}$ nodes rooted at $q_{root}$
1: $G_r(V, E) \leftarrow$ ConstructRegionGraph$(N_r, r, k)$
2: for all $v_i \in V$ par do
3:     $\tau \leftarrow \tau \cup$ BlindRRT$(q_{root}, N_{br}/p, \Delta q, v_i)$
4: end for
5: $\tau_{mst} \leftarrow$ MinimumSpanningTree$(G(V, E))$
6: for all $(v_i, v_j) \in \tau_{mst}$ par do
7:     ConnectRegions$(v_i, v_j)$
8: end for
9: return $\tau$
```

is computed so that no cycles are produced in the tree when connecting regions. Additionally, the minimum spanning tree provides information as to which neighbors are closest, and thus there is a higher probability of successful connection. To reduce the communication overhead in the region connection phase, we import all necessary information from the target region R_t, instead of updating the CC information every time a connection is performed. At the beginning, we know that none of the CCs in the source region R_s are connected to the CCs in R_t, so we initialize two sets: U the unconnected CCs and C the already merged CCs. Initially, the first contains all R_s CCs and the second is the empty set. P is a queue containing all the CCs in the source region, R_s. The goal is to merge U with P without creating cycles. First, we dequeue a CC, CC_{local} from P, and iterate through the CCs in C, attempting connections and stopping if one is found. Then, we iterate through the CCs in U, attempting connections to all of them; if a connection is made, we update the sets C and U, by adding CC_{local} to C and removing it from U. We perform this operation until P is empty. Note that connections between CCs from the same region are never attempted, but multiple CCs may connect to the same remote CC progressively merging the CCs into one. This procedure not only performs region connection with reduced communication overhead, but also indirectly connects local CCs through the remote CCs. After this global CC connection step, we may or may not have connected all CCs of the overall tree back to the root component CC_{root}. Even with all the measures taken to connect all disconnected components back to the root, there is no guarantee that only one CC will exist globally. Therefore, we remove all remaining CCs.

Figure IV.1 shows an example of the different steps of the parallel algorithm on a simple 2-D environment with $p = 4$ processors. Figure IV.1(a) shows the example environment with regions decomposed. Regions are represented by points (blue) on the outer sphere. Figure IV.1(b) shows a Blind RRT expanded for $N_{br}/p = 20$ expansions. Notice how Blind RRT ignores and expands through $\mathcal{C}_{obst}$ covering all of $\mathcal{C}_{space}$. Figure IV.1(c) shows the tree after local CC connection is performed. New edges are emphasized by magenta ellipses. Figure IV.1(d) shows the tree after global region connection. Again new edges are emphasized with magenta ellipses.

Algorithm 4 Connect Regions

```
Input: Two regions $R_s$ and $R_t$
1: Pending $CCs$ Queue $P \leftarrow R_s.\texttt{GetCCs}()$
2: Connected $CCs$ $C \leftarrow \emptyset$
3: Unconnected $CCs$ $U \leftarrow R_t.\texttt{GetCCs}()$
4: while $\neg P.\texttt{IsEmpty}()$ do
5:     $CC_{local} \leftarrow P.\texttt{Dequeue}()$
6:     for all $CC_{remote} \in C$ do
7:         if $\texttt{RRT} - \texttt{Connect}(CC_{local}, CC_{remote})$ then
8:             break
9:         end if
10:    end for
11:    for all $CC_{remote} \in U$ do
12:        if $\texttt{RRT} - \texttt{Connect}(CC_{local}, CC_{remote})$ then
13:            $C = C \cup CC_{remote}$
14:            $U = U \setminus CC_{remote}$
15:        end if
16:    end for
17: end while
```



(a) Region Decomposition (b) Blind RRT Expansion (c) Local CC Connection (d) Region Connection

Fig. IV.1.: (a) An example environment with four regions, represented by their points (blue) on the outer circle. (b) Blind RRT concurrently expanding in the four regions ignoring obstacles as it goes. (c) Blind RRT concurrently and locally removes invalid nodes of the tree and connects CCs within each region (new edges emphasized in magenta). (d) Blind RRT connects CCs between regions yielding a final tree.

Probabilistic Completeness

In this section, we show two things: the probabilistic incompleteness of Radial RRT and the probabilistic completeness of Radial Blind RRT.

Observation IV.0.1 *Radial RRT is probabilistically incomplete because an obstacle can entirely block exploration of a region in such a way that connections between adjacent regions will not be able to cover $\mathcal{C}_{space}$. This is shown in Figure IV.2.*

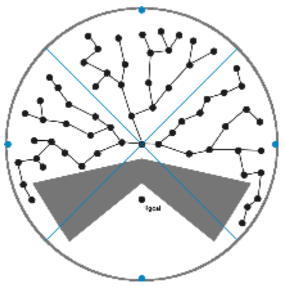


Fig. IV.2.: Example of Radial RRT not being able to solve an example query.

Theorem IV.0.2 *Radial Blind RRT is probabilistically complete.*

Proof Without loss of generality assume $\mathcal{C}_{free}$ is a single connected component. Collectively the Blind RRTs built in each region will be able to expand and cover all of $\mathcal{C}_{space}$ in the initial expansion phase, for the reasons stated in the proof of Theorem III.0.1. After the local connection phase, Radial Blind RRT recombines adjacent regions with RRT-Connect. By the probabilistic completeness of RRT-Connect [12], all regions will be merged and all components of the tree will be merged into one. Thus, Blind RRT is probabilistically complete. ■

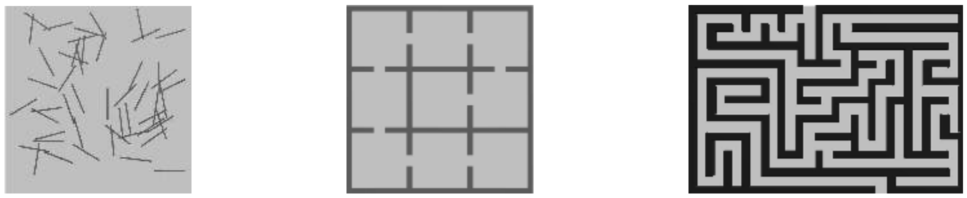
CHAPTER V
EXPERIMENTAL SETUP AND RESULTS

In this section, we analyze Radial Blind RRT under two different perspectives. We compare its effectiveness to that of Radial RRT and sequential RRT. Also, we present the scalability of the algorithm. Section V compares the methods in some environments showing the efficiency of Blind RRT covering the map, and Section V presents the performance of Blind RRT with different processor counts. Recall, the goal of this algorithm is to have a scalable RRT useful for motion planning. Standard parallel RRT methods do not scale well, whereas Radial RRT does. However, Radial RRT is unable to cover the planning space as well as RRT. Thus, the goal of this work is to show that Radial Blind RRT allows both scalability, like Radial RRT, and good coverage, almost as good (sometimes better) than RRT.

Experimental Setup

Experiments were conducted on a Linux computer center at Texas A&M University. The cluster has a total of 300 nodes, 172 of which are made of two quad core Intel Xeon and AMD Opteron processors running at 2.5GHz with 16 to 32GB per node. The 300 nodes have 8TB of memory and a peak performance of 24 Tflops. Each node of the cluster is made of 8 processor cores, thus, for this machine we present results for processor counts in multiples of 8. RRT, Radial RRT, and Radial Blind RRT were implemented in a C++ motion planning library which uses the Standard Template Adaptive Parallel Library (STAPL), a C++ parallel library a [23, 24]. Our code was compiled with gcc-4.5.2.

All the methods use Euclidean distance as distance metric, straight-line local planner, brute force neighborhood finder and collision detection tests as validity tests. Four different en-



(a) 2D Clutter (b) 2D Grid (c) 2D Maze

Fig. V.1.: (a), (b), and (c) are $2DOF$ problems. All RRTs begin expansion from the origin of the environment.

vironments were used: 2D Clutter (Figure V.1a), 2D Grid (Figure V.1b), and 2D Maze (Figure V.1c).

We analyze the effect of varying the CC selection policy and the number of regions in Radial Blind RRT; we also compare against a sequential RRT and Radial RRT. We approximate the map coverage of each method by taking a uniformly sampled set of nodes in an environment and determining the percentage of nodes visible to the tree.

Map Coverage

In this section, we compare the methods ability to map space by analyzing the coverage of the generated trees. We approximate coverage with a sample size of 250 uniformly sampled nodes. Since Radial Blind RRT deletes nodes at two points of its execution, it is not effective to use a desired final number of nodes. Instead, we fixed the parameter N_{br} to be 500. Another parameter that plays an important role is the number of CC connection attempts in the local phase. Given that for each environment the number of CCs will vary, we decided to set the number of CC connection attempts to be 5 times the number of CCs after the initial expansion phase. This number was chosen according to initial testing results which demonstrated that a high number of CC connection attempts only increases the number of nodes but does not connect the tree significantly better, making the method rather slow. To have a fair comparison between methods, we ran Radial Blind RRT first and recorded the average number of nodes per test case, and we took the maximum number of nodes to be the N for both RRT and Radial RRT. This N was different for each environment. Radial Blind RRT and Radial RRT were tested with $N_r = [1, 2, 4, 8]$. Radial Blind RRT was also tested for each of the CC selection criteria except for Random. Coverage results are normalized to RRT.

As shown in Figures V.2 and V.3, Radial Blind RRT results in a better map coverage compared to Radial RRT except for one case in the 2D Maze. We omitted a plot of the

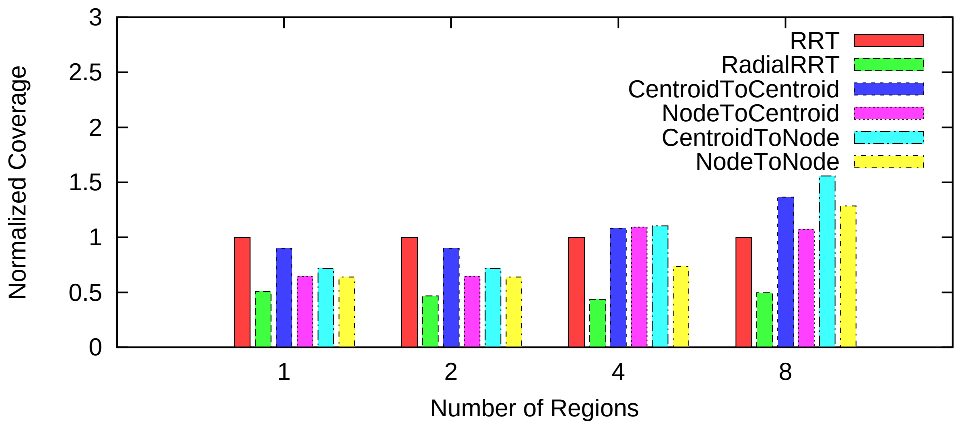


Fig. V.2.: Comparing coverage with RRT, Radial RRT, and Blind RRT in the 2D Grid environment.

coverage in 2D Clutter because all RRTs were able to reach 100% coverage. Differentiating between CC selection policies is difficult, we observed that Centroid to Centroid had the most reliable results and was the cheapest method to perform. Even though there are cases where Radial Blind RRT outperforms RRT, the results are usually in between those of Radial RRT and RRT. We observe that Radial Blind RRT performs well in mapping the space, but recall the goal of this thesis is to show a scalable RRT method for useful motion planning. In the future, we will further analyze the cases where the coverage is better than RRT to optimize them and implement its sequential version.

Scalability Experiment

We evaluated the Blind RRT algorithm on the LINUX cluster varying the processor count from 1 to 16. This experiment is to substantiate our claim for scalability. We carried out this experiment in the 2D Clutter, 2D Grid and 2D Maze environments. The initial input sample size was fixed at 1600 for each environment tested. Each experiment was run five times and the average maximum time for the 5 runs was computed. Figure V.4 shows the running time in the three environments.

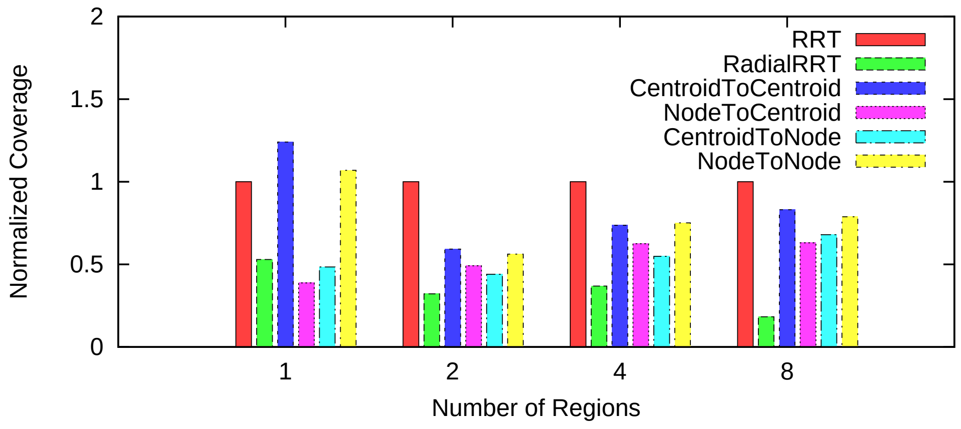


Fig. V.3.: Comparing coverage with RRT, Radial RRT, and Blind RRT in the 2D Maze environment.

Except for the outlier spike at processor count 2 for the 2D Maze environment, we observe that the running time decreases with an increase in the number of processors. Initial investigation to understand the cause of the spike show that the resulting tree size at processor count 2 is more than doubled of the tree size at processor count 1 for the 2D Maze environment, which could explain the reason for the unexpected increase in time. However, we are investigating further the actual cause of the increase in both the tree size and time.

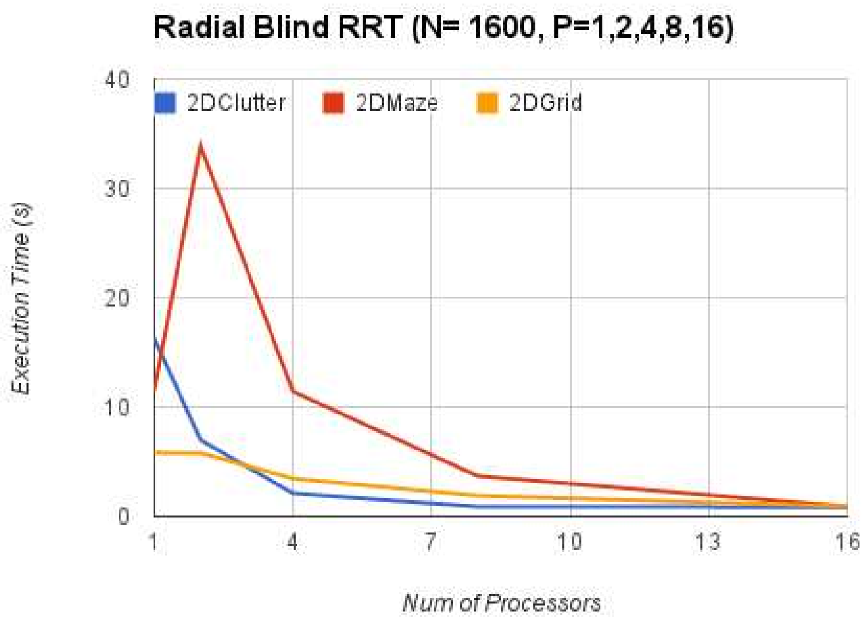


Fig. V.4.: Running Time with increase in processors(p=1,2,4,8,16).

CHAPTER VI
CONCLUSIONS

We show a scalable Radial Blind RRT that subdivides $\mathcal{C}_{space}$, builds an RRT in each section by ignoring obstacles, then connects disjoint CCs. After the local computation, it attempts connections across CCs in different regions. As a post-processing step, it removes any parts of the graph that could not be connected back to the root. In the experiments, we show that Radial Blind RRT scales almost linearly while effectively covering the space as opposed to Radial RRT. In future work, we will analyze the CC selection policy to optimize it and implement a sequential version of Blind RRT.

REFERENCES

[1] O. B. Bayazit, J.-M. Lien, and N. M. Amato, "Roadmap-based flocking for complex environments," in *Proc. Pacific Graphics*, pp. 104–113, Oct 2002.

[2] X. Sheng, "Motion planning for computer animation and virtual reality applications," in *Computer Animation '95., Proceedings.*, pp. 56–66, Apr.

[3] O. B. Bayazit, G. Song, and N. M. Amato, "Ligand binding with OBPRM and haptic user input: Enhancing automatic motion planning with virtual touch," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 954–959, 2001. This work was also presented as a poster at *RECOMB 2001*.

[4] H. Chang and T. Y. Li, "Assembly maintainability study with motion planning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 1012–1019, 1995.

[5] J. H. Reif, "Complexity of the mover's problem and generalizations," in *Proc. IEEE Symp. Foundations of Computer Science (FOCS)*, (San Juan, Puerto Rico), pp. 421–427, October 1979.

[6] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, vol. 12, pp. 566–580, August 1996.

[7] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, pp. 378–400, May 2001.

[8] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, pp. 560–570, October 1979.

[9] S. A. Jacobs, N. Stradford, C. Rodriguez, S. Thomas, and N. M. Amato, "A scalable distributed rrt for motion planning.," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, To appear May 2013.

[10] C. A. Rodriguez, J. Denny, S. A. Jacobs, S. Thomas, and N. M. Amato, "Blind rrt: A probabilistically complete, distributed rrt," Tech. Rep. TR13-004, Parasol Lab, Dept. of Computer Science, Texas A&M University, Apr 2013.

[11] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations.* Cambridge, MA: MIT Press, June 2005.

[12] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An Efficient Approach to Single-Query Path Planning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 995–1001, 2000.

[13] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, "Obprm: an obstacle-based prm for 3d workspaces," in *Proceedings of the third workshop on the algorithmic foundations of robotics on Robotics : the algorithmic perspective: the algorithmic perspective*, WAFR '98, (Natick, MA, USA), pp. 155–168, A. K. Peters, Ltd., 1998.

[14] V. Boor, M. H. Overmars, and A. F. van der Stappen, "The Gaussian sampling strategy for probabilistic roadmap planners," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 2, pp. 1018–1023, May 1999.

[15] D. Hsu, T. Jiang, J. Reif, and Z. Sun, "Bridge test for sampling narrow passages with probabilistic roadmap planners," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 4420–4426, 2003.

[16] D. J. Challou, M. Gini, and V. Kumar, "Parallel search algorithms for robot motion planning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 2, pp. 46–51, 1993.

[17] T. Lozano-Pérez and P. O'Donnell, "Parallel robot motion planning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 1000–1007, 1991.

[18] N. M. Amato and L. K. Dale, "Probabilistic roadmap methods are embarrassingly parallel," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 688–694, 1999.

[19] S. A. Jacobs, K. Manavi, J. Burgos, J. Denny, S. Thomas, and N. M. Amato, "A scalable method for parallelizing sampling-based motion planning algorithms," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2012.

[20] S. Carpin and E. Pagello, "On parallel rrts for multi-robot systems," in *Proc. Italian Assoc. AI*, pp. 834–841, 2002.

[21] D. Devaurs, T. Simeon, and J. Cortes, "Parallelizing rrt on distributed-memory architectures," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2011.

[22] D. Hsu, J.-C. Latombe, and H. Kurniawati, "On the probabilistic foundations of probabilistic roadmap planning," *Int. J. Robot. Res.*, vol. 25, pp. 627–643, July 2006.

[23] A. Buss, Harshvardhan, I. Papadopoulos, O. Pearce, T. Smith, G. Tanase, N. Thomas, X. Xu, M. Bianco, N. M. Amato, and L. Rauchwerger, "STAPL: Standard template adaptive parallel library," in *Proc. Annual Haifa Experimental Systems Conference (SYSTOR)*, (New York, NY, USA), pp. 1–10, ACM, 2010.

[24] G. Tanase, A. Buss, A. Fidel, Harshvardhan, I. Papadopoulos, O. Pearce, T. Smith, N. Thomas, X. Xu, N. Mourad, J. Vu, M. Bianco, N. M. Amato, and L. Rauchwerger, "The STAPL Parallel Container Framework," in *Proc. ACM SIGPLAN Symp. Prin. Prac. Par. Prog. (PPoPP)*, (San Antonio, Texas, USA), pp. 235–246, 2011.