Blind RRT: A Probabilistically Complete Distributed RRT

Cesar Rodriguez, Jory Denny, Sam Ade Jacobs, Shawna Thomas, and Nancy M. Amato

***Abstract*— Rapidly-Exploring Random Trees (RRTs) have been successful at finding feasible solutions for many types of 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 computation locality. However, if an obstacle completely blocks RRT growth in a region, the planning space is not covered and is thus not probabilistically complete. 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. 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 effectiveness in motion planning. This method is 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.**

I. INTRODUCTION

Motion planning has been an active research area in the field of robotics. It also has many other applications including computer graphics [2], virtual reality [22], computational biology [3], and computer-aided design [7]. 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 $DOF \geq 5$ [20], sampling-based motion planning has emerged as a promising technique for solving such problems [14], [17].

Within sampling-based motion planning, we find two main approaches: graph-based approaches, e.g., Probabilistic Roadmaps (PRM) [14], and tree-based approaches, e.g., Rapidly-Exploring Random Trees (RRT) [17]. RRTs iteratively grow one or more trees from a given configuration towards random configurations in the configuration space ($\mathcal{C}_{space}$) [19]. In each iteration, a robot configuration q_{rand} is chosen as an expansion direction, and the nearest node q_{near} in the tree 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 of obtaining a solution can be 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 [13], 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 in adjacent regions using connected component connection techniques. Finally, cycles are removed so the resulting graph is a tree. This method scales well but is not probabilistically complete.

In this work, we present an RRT variant, Blind RRT, that ignores obstacles during the initial tree construction, retaining all vertices (invalid and valid) and all valid edges. Blind RRT expands regardless of encountering obstacles. After initial construction, a post processing step is required where invalid nodes are removed and unconnected pieces of the tree are merged back to the root. We apply this method in parallel, providing a probabilistically complete alternative to Radial RRT.

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 to Radial RRT to provide both scalability and probabilistic completeness 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.

II. PRELIMINARIES AND RELATED WORK

In this section, we review definitions and present related motion planning techniques relevant to this work.

A. 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 [8]. 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

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.

C. Rodriguez, J. Denny, S. A. Jacobs, S. Thomas, and N. M. Amato are with the Department of Computer Science and Engineering, Texas A&M University, College Station, TX, 77843, USA {cesar094,jdenny,sjacobs,sthomas,amato}@cse.tamu.edu.

called the configuration space ($\mathcal{C}_{space}$) [19] and consists of 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 [20]. However, the feasibility of a configuration can be determined quite efficiently using collision detection (CD) tests.

B. 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 [14], [17]. Probabilistic Roadmaps (PRMs) [14] and Rapidly-Exploring Random Trees (RRTs) [17] are two common approaches to sample-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 $\mathcal{C}_{space}$, as 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 $q_{new} \in \mathcal{C}_{free}$. 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 [15] 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 Δq
Output: Tree τ

1: $\tau \leftarrow q_{root}$
2: **while** $\neg done$ **do**
3: $q_{rand} \leftarrow$ `RandomCfg`()
4: $q_{near} \leftarrow$ `NearestNeighbor`(τ, 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** τ

C. Parallel Motion Planning

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 [6], [18]. PRM was first parallelized in [1]. A scalable Parallel PRM was presented in [12] 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 [5]. Three different distributed RRT algorithms are presented in [10]. 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 [13] 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.

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.

A. Algorithm

The Blind RRT strategy, shown in Algorithm 2, starts by iteratively expanding a tree τ rooted at a configuration q_{root}, 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, the Blind RRT has the same nodes as an RRT in an obstacle free environment and the RRT edges that are valid considering obstacles. After performing N_{br} Blind RRT expansion iterations, Blind RRT deletes all invalid nodes in τ and performs a connection phase for the various connected components (CCs). Following this, all CCs other than the CC containing the root are deleted. τ is returned.

Note that one benefit of the standard RRT algorithm is early termination if coarse coverage is sufficient to solve the query. This can be achieved here by interleaving tree construction and evaluation and setting N_{br} appropriately.

Blind Tree Expansion. We describe two alternatives when performing blind expansion. Note that other RRT expansion algorithms could be modified and used appropriately. The first performs validity checking for the entire line from q_{near}

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

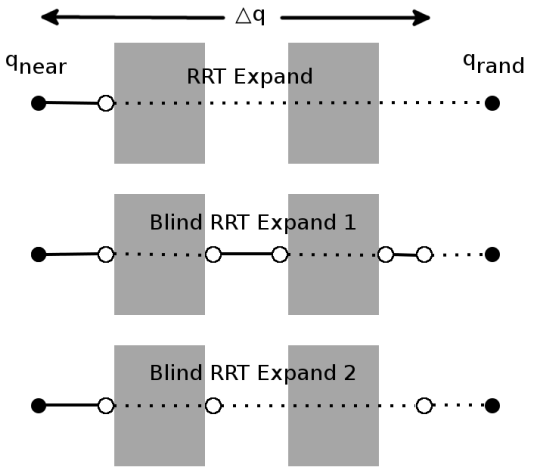


Fig. 1: RRT Expand expands greedily up to Δq, q_{rand}, or an obstacle is hit (a). Blind RRT Expand always expands up to Δq distance or q_{rand} while also retaining either all free witnesses (b) or only the first free witness (c) to return a set of expansion nodes Q_{new}.

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 1(b)). It adds all the nodes collected as well as q_{new} (which itself may or may not be valid). The second stops at the first validity change, records the valid node, and directly jumps to q_{new} and adds it (Figure 1(c)). The latter skips part of the collision detection, while the former keeps track of more valid nodes. 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 join pairs of CCs, CC_a and CC_b, using RRT-Connect [15] where $\tau_a = CC_a$ and $\tau_b = CC_b$. In the connection step, we first choose CC_a as a random CC, and then choose a target CC, CC_b, by the CC whose centroid is closest to the centroid of CC_a. A nearest neighbor query is performed from the centroid of CC_a to the centroids of the other CCs. It significantly reduces the nearest neighbor computation, as the number of CCs is much less than the number of nodes in the tree. This is used as an approximation scheme in selecting the closest CC. We have explored a number of other ways to select CC_b and found that this mechanism has the most consistent results and is the cheapest method to perform [21]. Component connection iteratively selects CCs to connect to until either one CC is achieved or a maximum number of failures is reached. After the CC connection phase we retain only the component containing the root.

B. Probabilistic Completeness

Probabilistic completeness is a desirable property of randomized planners which describes 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. We assume that the $\mathcal{C}_{space}$ is ϵ-good [11] for some $\epsilon > 0$.

Theorem 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 [15] that Blind RRT will connect CC_g to CC_s to yield a valid path between q_s and q_g. ■

IV. An Improved Radial RRT using Blind RRT

In this section, we introduce an improved Radial RRT framework for parallelizing RRTs which uses Blind RRT as a subroutine.

A. Algorithm

Radial Blind RRT starts by radially subdividing $\mathcal{C}_{space}$ as in Radial RRT [13]. It makes use of a region graph, which is an abstraction of the different subdivisions of

$\mathcal{C}_{space}$. To subdivide $\mathcal{C}_{space}$ and 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. Note that this applies to any dimension of $\mathcal{C}_{space}$. These samples that define the subdivision become the vertices of the region graph. A k-closest connection routine determines the adjacency of the regions defining the edges of the region graph. The number of neighbors a region has, and thus the communication later required, can be tuned by k.

Radial Blind RRT, shown in Algorithm 3, constructs a Blind RRT in parallel in each region. Each region constructs 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, each region will contain several CCs that need to be connected back to the root component. This takes place in a global region connection phase.

Algorithm 3 Radial Blind RRT

Input: A root configuration q_{root}, the number of nodes N_{br}, a maximum expansion distance Δq, the number of processors p, the number of regions N_r, a region radius r, the number of adjacent regions k

Output: A tree τ 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: $\quad \tau \leftarrow \tau \cup$ `BlindRRT`$(q_{root}, N_{br}/p, \Delta q, v_i)$
4: **end for**
5: $\tau_{mst} \leftarrow$ `MinimumSpanningTree`$(G_r(V, E))$
6: **for all** $(v_i, v_j) \in \tau_{mst}$ **par do**
7: $\quad$ `ConnectRegions`(v_i, v_j)
8: **end for**
9: **return** τ

The region connection phase, described in Algorithm 4, 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 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_t 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 not explicitly attempted in this phase. They have already been attempted in the BlindRRT call for each region (see Algorithm 3, line 3). However, 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 may also indirectly connect 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. Therefore, we remove all remaining CCs.

Algorithm 4 Connect Regions

Input: Two regions R_s and R_t

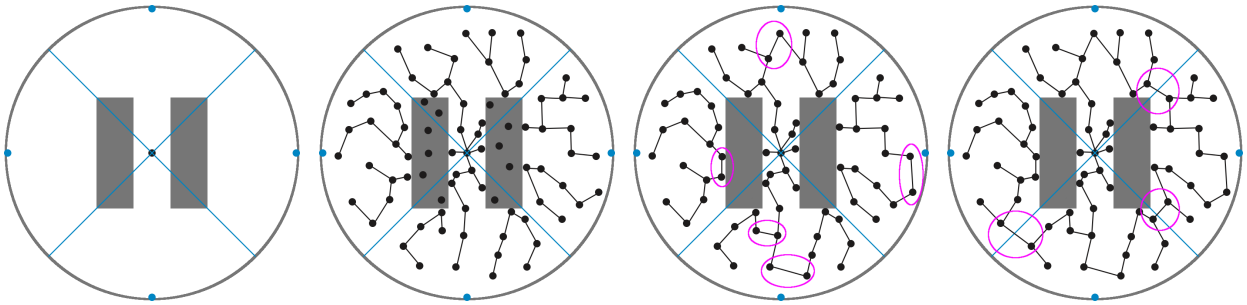
1: Pending CCs Queue $P \leftarrow R_s$.`GetCCs`()
2: Connected CCs $C \leftarrow \emptyset$
3: Unconnected CCs $U \leftarrow R_t$.`GetCCs`()
4: **while** $\neg P$.`IsEmpty`() **do**
5: $\quad CC_{local} \leftarrow P$.`Dequeue`()
6: $\quad$ **for all** $CC_{remote} \in C$ **do**
7: $\quad\quad$ **if**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

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

B. Probabilistic Completeness

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

Observation 1: Radial RRT is not probabilistically complete because an obstacle can entirely block exploration of a



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

Fig. 2: (a) An example environment with four regions, represented by their points (blue) on the outer circle. (b) Radial Blind RRT concurrently expanding in the four regions ignoring obstacles as it goes. (c) Radial Blind RRT concurrently and locally removes invalid nodes of the tree and connects CCs within each region (new edges emphasized in magenta). (d) Radial Blind RRT connects CCs between regions yielding a final tree.

region, in such a way that connections between adjacent regions will not be able to cover $\mathcal{C}_{space}$, see Figure 3.

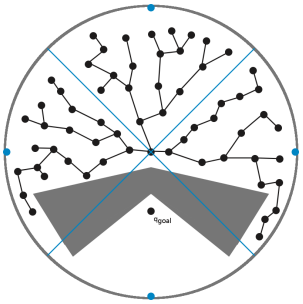


Fig. 3: Example of Radial RRT not being able to solve an example query.

Theorem 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 1. After the local connection phase, Radial Blind RRT recombines adjacent regions with RRT-Connect. By the probabilistic completeness of RRT-Connect [15], all regions will be merged and all components of the tree will be merged into one. Thus, Radial Blind RRT is probabilistically complete. ■

C. Algorithm Analysis

In this section, we present complexity analysis of Radial Blind RRT as a conceptual proof of our claim for scalability. Recall that the original RRT algorithm as presented in [16] requires $O(N^2)$ time (in the worst case) to construct a tree with N configurations. This analysis assumes a brute force strategy for nearest neighbor queries, as will our analysis. We note to the reader that more efficient mechanisms for nearest neighbor queries exist in the literature, e.g., KD-trees, but for simplicity of analysis we assume worst case computation.

The Radial Blind RRT given in Algorithm 3 can be broken down into four phases: region graph construction, Blind RRT construction, minimum spanning tree (MST) computation, and region connection. The overall time complexity of the algorithm can be described in terms of these four phases as:

$$T = T_{rg} + T_{BRRT} + T_{MST} + T_c$$

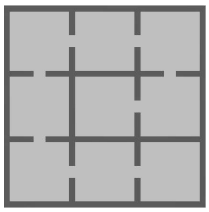
where the total time T is the sum of the cost T_{rg} of region graph construction for a given environment subdivided into n_r regions with each region having d neighbors, the cost T_{BRRT} of constructing n_r sequential Blind RRTs, the cost T_{MST} of computing an MST of the region graph, and the cost T_c of connecting neighboring CCs between adjacent regions given from the MST. In our analysis, p refers to the number of parallel processing elements. Please note that our formulation assumes a uniform cost of constructing subtrees in each region; this assumption may fail in a situation where the regions are non-uniform.

In the first phase, we construct the region graph of n_r vertices and dn_r edges. The dominant factor in constructing the region graph is the d-nearest neighbor search, with $O(n_r^2 \log d)$ complexity assuming a brute force search. Each processor p will generate $\frac{n_r}{p}$ regions. So in parallel constructing the dn_r edges will take $O(\frac{n_r^2 logd}{p})$ time.

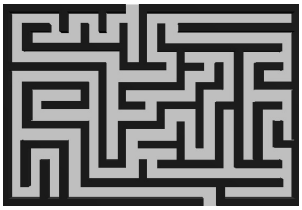
The second phase of the algorithm constructs a Blind RRT in each region of size $N_{br} = \frac{N}{n_r}$, where N is the expected number of nodes for tree construction. Since the complexity of sequential Blind RRT is equivalent to the complexity of RRT, the total work for this phase is $O((\frac{N}{n_r})^2)$. Assuming uniform distribution of work across the processing elements, the time complexity is $O((\frac{N}{n_r})^2/p)$.

Most inter-processor communication occurs when connecting regional subtrees in phase four. However, this communication is managed by reducing the region graph to a MST in phase three, which will require a time complexity of $O(\frac{dn_r \log n_r}{p})$ [9]. Then, phase four will require $O(cn_r)$ instantiations of RRT-Connect, each requiring $O(N_{rrtc}^2)$ work, where c is the maximum number of CCs within a region and N_{rrtc} is the maximum number of nodes allotted per RRT-Connect tree. Upon parallelization, phase four requires $O(\frac{cn_r N_{rrtc}^2}{p})$ time.

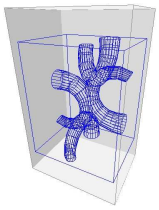
(a) 2-D Clutter



(b) 2-D Grid



(c) 2-D Maze



(d) 3-D Maze

Fig. 4: (a), (b), and (c) are $2DOF$ problems, and (d) is a $6DOF$ problem. All RRTs begin expansion from the origin of the environment.

We assume that $N >> n_r$ and $N_{rrtc} >> n_r$, so the final time complexity for Radial Blind RRT can be reduced to:

$$T = O((\frac{N/n_r}{p})^2) + O(\frac{cn_rN_{rrtc}^2}{p})$$

implying that the time spent per phase can vary based upon the success in covering the space with fewer CCs, i.e., lower connection time.

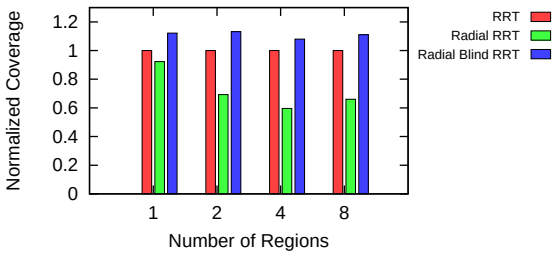
V. Experimental Setup and Results

In this section, we analyze Radial Blind RRT under two different perspectives. We compare its effectiveness to that of sequential RRT and Radial RRT. Also, we present the scalability of the algorithm against Radial RRT. Section V-B compares the methods in a few environments showing the efficiency of Radial Blind RRT exploration, and Section V-C presents the performance of Radial 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, comparable to RRT.

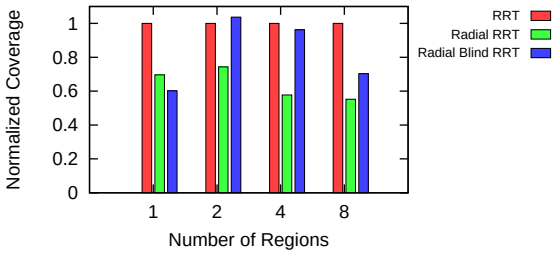
A. Experimental Setup

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. All methods were implemented in a C++ motion planning library which uses the Standard Template Adaptive Parallel Library (STAPL), a C++ parallel library [4], [23]. Our code was compiled with gcc-4.5.2.

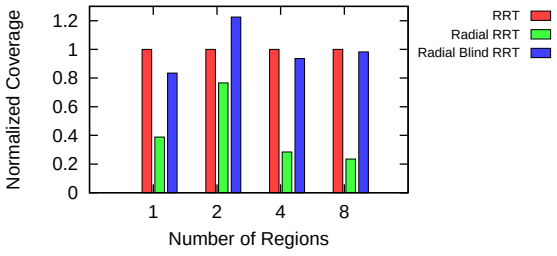
All the methods use Euclidean distance as the distance metric, straight-line local planning, brute force neighborhood finding, and collision detection tests as validity tests. Four different environments were used: 2-D Clutter (Figure 4(a)), 2-D Grid (Figure 4(b)), 2-D Maze (Figure 4(c)), and 3-D Maze (Figure 4(d)).



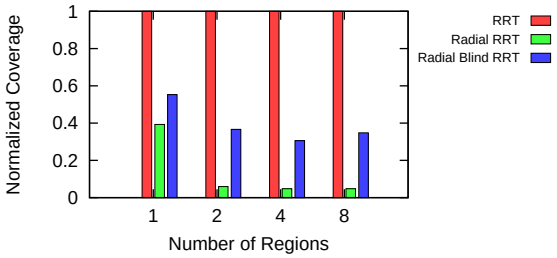
(a) 2-D Clutter



(b) 2-D Grid



(c) 2-D Maze



(d) 3-D Maze

Fig. 5: Comparing coverage after performing RRT, Radial RRT, and Radial Blind RRT. All results are normalized to RRT.

B. Map Coverage

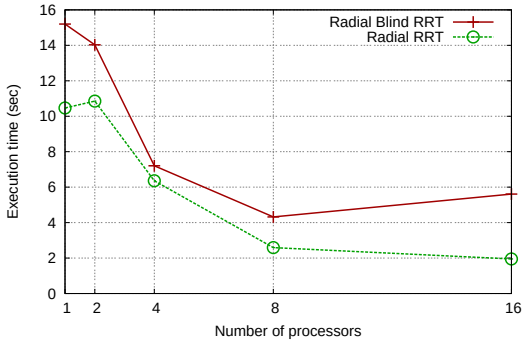
In this section, we compare each method's ability to map space by analyzing the coverage of the generated trees. We approximate coverage with a sample size of 250 uniformly sampled nodes. Since Radial Blind RRT deletes nodes at two points of its execution, it is not effective to use a desired final number 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 set the number of CC connection attempts to be five 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, for each random seed, we ran Radial Blind RRT first and recorded the number of nodes, N_i. Then, we took the number of nodes to be the N_i for both RRT and Radial RRT. Radial Blind RRT and Radial RRT were tested with $N_r = [1, 2, 4, 8]$. Coverage results are averaged over 10 random seeds and normalized to RRT. Results are shown in Figure 5.

Radial Blind RRT results in better map coverage compared with Radial RRT, except in one case (2D-Grid with one region) in which Radial Blind RRT was comparable to Radial RRT. Moreover, in most of the 2D environments Radial Blind RRT has higher or comparable coverage compared to RRT. In higher dimensional cases (3D-Maze), we believe radial decomposition hampers exploration for a fixed number of nodes. However, we note that Radial Blind RRT still performs better than Radial RRT in these cases. We will look at improving these results for higher dimensional problems in the future. From these results, we can see that Radial Blind RRT shows usefulness in planning over Radial RRT alone, and in some cases, e.g., 2D homogeneous environments, Radial Blind RRT actually outperforms RRT.

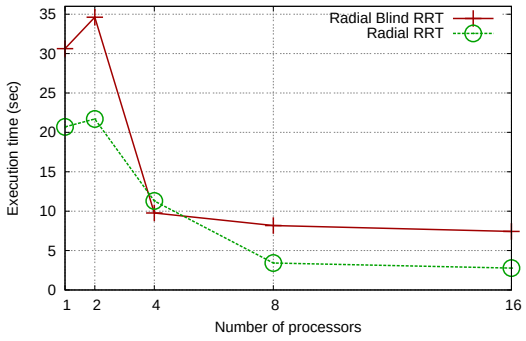
C. Parallel Performance

We evaluated Radial Blind RRT on the Linux cluster varying the processor count from 1 to 16. We compare Radial Blind RRT to Radial RRT to see differences in performance as the number of regions increases. In these experiments, the number of cores is equal to the number of regions. It is important to note that Radial RRT's and Radial Blind RRT's trees differ as the region count differs. We carried out the experiments in the 2-D Clutter, 2-D Grid, and 3-D Maze environments. The initial input sample size was fixed at 1600. Each experiment was run five times and the average runtime of the longest running processor was computed. Results are shown in Figure 6.

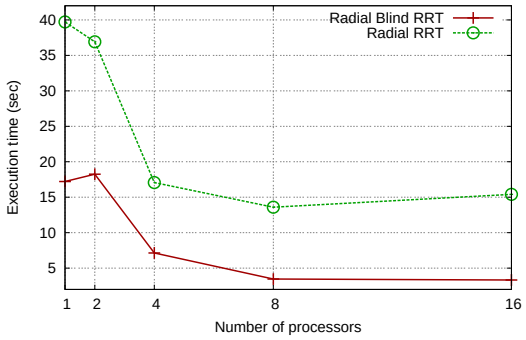
We observe that the relative performance of each algorithm depends on the environment. When Radial RRT requires more time, many failed attempts occur as regions restrict the expandability of the tree. In these cases, more computation time is spent attempting expansions as the tree attempts to grow to a specific size. When Radial Blind RRT requires more time, more work is spent in attempting to connect disconnected components. The tree size for Radial Blind RRT is larger, so we expect that Radial Blind RRT requires more work from an initial sample set. Considering runtime, we can see that even though Radial Blind RRT does more work, as it explores space better, running times are still comparable. Additionally, as the number of regions and processors increases, the running times decrease. We would like to explore this effect at larger core counts in the future and perform scaling experiments with a fixed region count to see how our implementation scales on distributed platforms.



(a) 2-D Clutter



(b) 2-D Grid



(c) 3-D Maze

Fig. 6: Execution times of Radial RRT and Radial Blind RRT in (a) 2-D Clutter, (b) 2-D Grid, and (c) 3-D Maze.

VI. Conclusions

We show a scalable Radial Blind RRT that subdivides $\mathcal{C}_{space}$, builds an RRT in each region by ignoring obstacles, and 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 well while effectively covering the space as opposed to Radial RRT which does not cover the space well. In future work, we will analyze the CC selection policy to optimize it and implement a sequential version.

References

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

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

[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)*, pages 954–959, 2001. This work was also presented as a poster at *RECOMB 2001*.

[4] 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)*, pages 1–10, New York, NY, USA, 2010. ACM.

[5] S. Carpin and E. Pagello. On parallel rrts for multi-robot systems. In *Proc. Italian Assoc. AI*, pages 834–841, 2002.

[6] D. J. Challou, M. Gini, and V. Kumar. Parallel search algorithms for robot motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 2, pages 46–51, 1993.

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

[8] 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*. MIT Press, Cambridge, MA, June 2005.

[9] S. Chung and A. Condon. Parallel implementation of boruvka's minimum spanning tree algorithm. *Parallel Processing Symposium, International*, 0:302, 1996.

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

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

[12] 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.

[13] 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)*, 2013.

[14] 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.*, 12(4):566–580, August 1996.

[15] 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)*, pages 995–1001, 2000.

[16] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 473–479, 1999.

[17] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *Int. J. Robot. Res.*, 20(5):378–400, May 2001.

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

[19] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979.

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

[21] C. Rodriguez, J. Denny, S. Jacobs, S. Thomas, and N. M. Amato. Blind RRT: A probabilistically complete, distributed RRT. Technical Report TR13-004, Parasol Lab, Dept. of Computer Science, Texas A&M University, Apr 2013.

[22] X. Sheng. Motion planning for computer animation and virtual reality applications. In *Computer Animation '95., Proceedings.*, pages 56–66, Apr.

[23] 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)*, pages 235–246, San Antonio, Texas, USA, 2011.