Rapidly-Exploring Random Trees: from an automated planning perspective

Paula Moraes

June 19, 2018

IME-USP: Institute of Mathematics and Statistics - University of Sao Paulo

Revision Motion Planning

Motion Planning

Motion Planning is an important area in robotics that addresses the problem of finding a sequence of control inputs so as to drive the robot from its initial state (or configuration) to one of the goal states while obeying the rules of the environment [4]

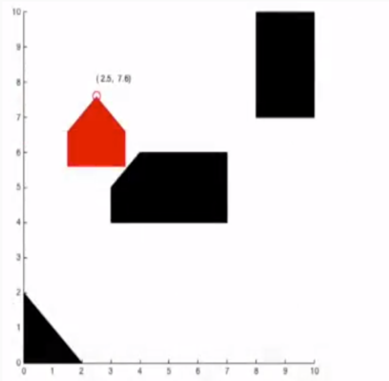
This problem is known to be at least PSPACE-hard...

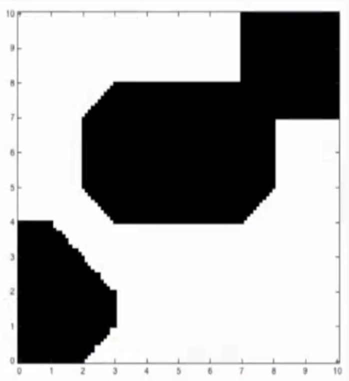
Besides robotics other areas rely on motion planning algorithms:

- computer graphics
- computational biology
- virtual prototyping

Configuration Space - C_{space}

C_{space} is the space of all possible configurations (or positions) that a system can attain given its rigid body degrees of freedom (DOF[1]) and the presence of obstacles in the environment.





[1]the number of independent parameters that define its configuration

The need of sampling-based methods

Since real world applications often requires planning in high-dimensional spaces, an explicit representation of the obstacles in C_{space} to construct a solution is unfeasible.

Introduced in the mid-90s, **sampling-based algorithms** avoided representations of the world by relying on a collision checking module building a graph with only the feasible trajectories.

- *Probabilistic RoadMaps* - constructs a graph of sets of collision-free trajectories to then answer multiple queries by computing a shortest path that connects the initial with a final state through the roadmap [4]
- *Rapidly-Exploring Random Trees* - incrementally grows a tree from an initial configuration towards unexplored portions of the space until the goal region is reached

Rapidly-Exploring Random Trees

Overview

RRT is an efficent data structure and sampling scheme to quickly search high-dimensional spaces [5], widely used in continuous path planning problems.

The exploration is biased toward unexplored regions by sampling random points and incrementally growing the search tree towards them.

Caracteristics:

- suboptimal
- probabilistic completeness
- aimed at single-query applications
- heavily relies on a collision checking module given it doesn't explicit construct obstacles in the state space

Step-by-step basic RRT algorithm

```
BUILD_RRT($q_{\text{init}}$)
1:  $tree \leftarrow q_{\text{init}}$;
2:  while $\neg end\_criteria$ do
3:      $q_{\text{rand}} \leftarrow$ RANDOM_STATE();
4:      EXTEND($q_{\text{rand}}$, $tree$);
5:  return $tree$;
```

```
EXTEND($q_{\text{rand}}$, $tree$)
1:  $q_{\text{near}} \leftarrow$ NEAREST_NEIGHBOR($q_{\text{rand}}$, $tree$);
2:  ($q_{\text{new}}$, $collided$) $\leftarrow$ NEW_STATE($q_{\text{rand}}$, $q_{\text{near}}$, FIXED_STEP_SIZE,
    SEARCH_SPACE);
3:  if $\neg collided$ then
4:      add_vertex($q_{\text{new}}$, $tree$);
5:      add_edge($q_{\text{new}}$, $q_{\text{near}}$,$tree$);
6:      if $q_{\text{new}} = q_{\text{rand}}$ then
7:          return $Reached$;
8:      else
9:          return $Advanced$;
10: return $Trapped$;
```

Sampling random state

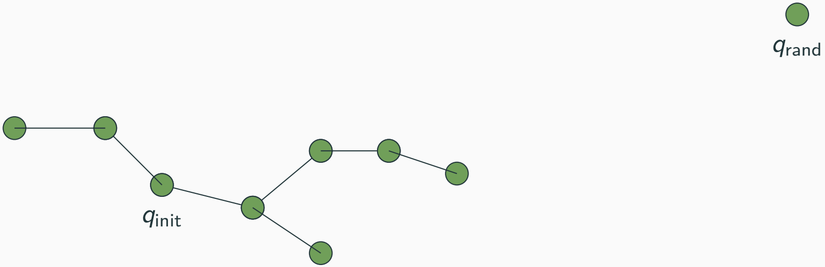


Figure 1: Tree rooted at q_{init} and sampled state q_{rand}

Sampling: $U(C_{space}) \rightarrow q_{rand}$

Step-by-step basic RRT algorithm

```
BUILD_RRT($q_{init}$)
1: $tree \leftarrow q_{init}$;
2: while $\neg end\_criteria$ do
3:     $q_{rand} \leftarrow$ RANDOM_STATE();
4:     EXTEND($q_{rand}$, $tree$);
5: return $tree$;
```

```
EXTEND($q_{rand}$, $tree$)
1: $q_{near} \leftarrow$ NEAREST_NEIGHBOR($q_{rand}$, $tree$);
2: ($q_{new}$, $collided$) $\leftarrow$ NEW_STATE($q_{rand}$, $q_{near}$, FIXED_STEP_SIZE,
   SEARCH_SPACE);
3: if $\neg collided$ then
4:     add_vertex($q_{new}$, $tree$);
5:     add_edge($q_{new}$, $q_{near}$,$tree$);
6:     if $q_{new} = q_{rand}$ then
7:         return $Reached$;
8:     else
9:         return $Advanced$;
10: return $Trapped$;
```

Finding nearest neighbor

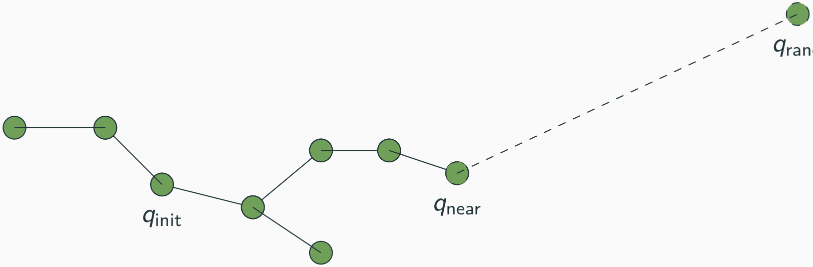


Figure 2: q_{near} is the closest state to q_{rand} that already belongs to the tree

Step-by-step basic RRT algorithm

```
BUILD_RRT($q_{\text{init}}$)
1: $tree \leftarrow q_{\text{init}}$;
2: while $\neg end\_criteria$ do
3:     $q_{\text{rand}} \leftarrow$ RANDOM_STATE();
4:     EXTEND($q_{\text{rand}}$, $tree$);
5: return $tree$;
```

```
EXTEND($q_{\text{rand}}$, $tree$)
1: $q_{\text{near}} \leftarrow$ NEAREST_NEIGHBOR($q_{\text{rand}}$, $tree$);
2: ($q_{\text{new}}$, $collided$) $\leftarrow$ NEW_STATE($q_{\text{rand}}$, $q_{\text{near}}$, FIXED_STEP_SIZE,
   SEARCH_SPACE);
3: if $\neg collided$ then
4:     add_vertex($q_{\text{new}}$, $tree$);
5:     add_edge($q_{\text{new}}$, $q_{\text{near}}$,$tree$);
6:     if $q_{\text{new}} = q_{\text{rand}}$ then
7:         return $Reached$;
8:     else
9:         return $Advanced$;
10: return $Trapped$;
```

Extending tree

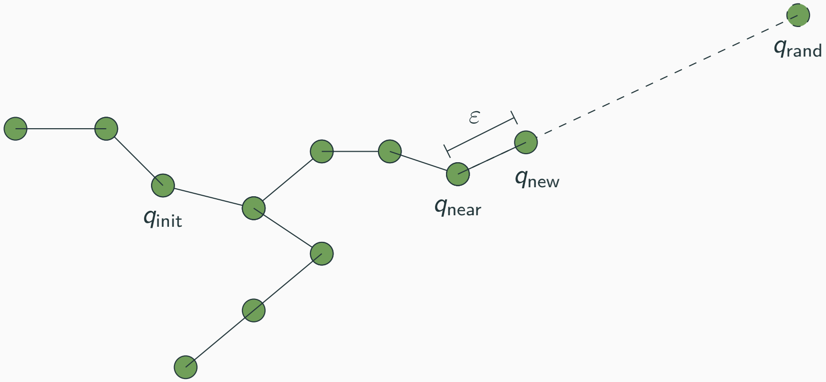


Figure 3: A new state q_{new} will be generated by a local search (NEW_STATE function) considering a fixed step size of ε and its feasibility

Step-by-step basic RRT algorithm

```
BUILD_RRT($q_{\text{init}}$)
1: $tree \leftarrow q_{\text{init}}$;
2: while $\neg end\_criteria$ do
3:     $q_{\text{rand}} \leftarrow$ RANDOM_STATE();
4:     EXTEND($q_{\text{rand}}$, $tree$);
5: return $tree$;
```

```
EXTEND($q_{\text{rand}}$, $tree$)
1: $q_{\text{near}} \leftarrow$ NEAREST_NEIGHBOR($q_{\text{rand}}$, $tree$);
2: ($q_{\text{new}}$, $collided$) $\leftarrow$ NEW_STATE($q_{\text{rand}}$, $q_{\text{near}}$, FIXED_STEP_SIZE,
   SEARCH_SPACE);
3: if  then$\neg collided$
4:     add_vertex($q_{\text{new}}$, $tree$);
5:     add_edge($q_{\text{new}}$, $q_{\text{near}}$,$tree$);
6:     if $q_{\text{new}} = q_{\text{rand}}$ then
7:         return $Reached$;
8:     else
9:         return $Advanced$;
10: return $Trapped$;
```

Tree after one extend iteration

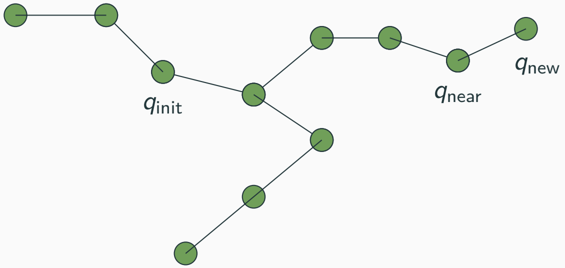


Figure 4: In this case the EXTEND function will return *Advanced* and q_{rand} will be discarded

Voronoi bias

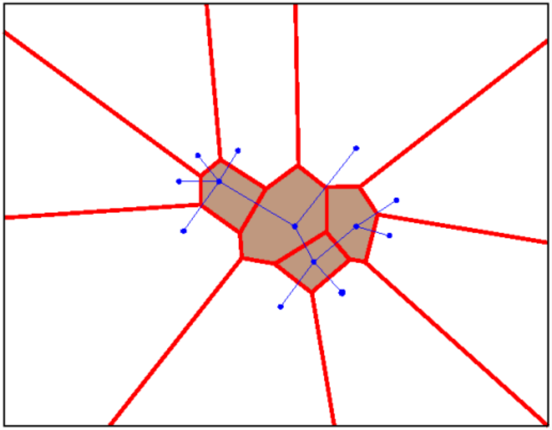


Figure 5: Frontier nodes have a Voronoi region bounded by the space limits. On the contrary, the Voronoi region of nonfrontier nodes is bounded by the Voronoi region of other nodes [3]. RRTs naturally balance between exploration and exploitation.

RRT core modules

As stated earlier RRTs are widely used in robotics, but they can be adapted to other areas if the following functions are modified.

Core modules:

- **Sampling**
  strong relation on how the search space will be explored (i.e. bias toward unexplored regions or a goal state)
- **Estimating the nearest neighbor**
  proximity evaluation of the tree nodes and a sampled state (usually measured by Euclidean distance)
- **Extending the tree**
  selection of a new node to be added that doesn't violate any constraints

Paper - Adapting a RRT for AP

Motivation

Motion planning is an area closely related to automated planning [1]

RRTs may be able to overcome some of the shortcomings that forward search planners have while keeping most of their desired properties.

Employ RRTs focused on satisficing planning problems.

Advantages of RRT for Automated Planning:

- good balance between exploitation and exploration during search
- bounded local searches (minimizing exploration of h plateaus)
- relatively small memory footprint
- broad range of techniques during local searches

Objective

Adapt RRTs to propositional planning domains, with the following formulation: $P = (S, A, I, G)$

Where:

- **S** is a set of atomic propositons;
- **A** is the set of grounded actions derived from the operators of the domain;
- **I** is the initial state;
- **G** is the set of goal propositions

Contributions from previous work

RRTs in discrete search spaces[6]

Adapting RRTs to grid worlds and similar classical search problems

Table 1: RRT modifications for discrete space

RRT module	**Employed technique**
Sampling	randomly sample a state that it's not already in the tree
Nearest Neighbor	"ad-hoc" heuristic estimation of the cost of reaching the sampled state from a given node of the tree
Extend	local planner limited by the number of nodes expanded

Contributions from previous work

RRT-Plan[2]

Directly related to Automated Planning, was proposed as a stochastic planning algorithm inspired by RRTs for deterministic planning tasks.

Table 2: RRT modifications for classical planning

RRT module	Employed technique
Sampling	sample from a subset of propositions from the goal
Nearest Neighbor	caches the cost of achieving every goal proposition from a state
Extend	local planner limited by the number of nodes expanded

Random Planning Tree (RPT)

SAMPLING

Uniform sampling of the search space using state invariants as constraints. Sampling is perform by choosing propositions from each invariant group such that is not in mutex with any other selected proposition for q_{rand}.

For every sampled state a reachability of goals is performed, if some goal proposition is unreachable, the state is discarded.

“exactly-1" invariant group - set of propositions that only one can be true in a state at the same time.

h^2 **heuristic** - approximate the cost of a set of atoms by the cost of the most costly atom *pair* in the set. A pair with infinite cost is a binary mutex, implying that they can’t belong to the same state.

Random Planning Tree (RPT)

NEAREST NEIGHBOR

Finding the closest node of the tree to a sampled state q_{rand} requires a distance estimation. In Automated Planning, this measurement is evaluated using heuristics derived from reachability analysis, like h^{add} and h^{ff}.

Since each extend phase of RRT requires a distance evaluation, estimating the cost to every sample state can make the problem intractable.

RPT caches the *best support* action (first action to achieve a proposition) for every proposition in the state.

– esperar resposta do email para detalhar mais o funcionamento dessa parte do hff –

Random Planning Tree (RPT)

EXTEND

Tree is built from the initial state and node contains a state, a link to its parent, a plan that leads from its parent to the state and the cached best supporters for every proposition.

There is a probability p of advancing towards the goal and a probability $1 - p$ of advancing towards a sampled state.

Local search limited by an ε number of nodes can be performed at most two times per iteration:

- if with a probability p, tries to extend toward a sampled state, returning q_{new} as the closest node of this local search

- another one to find a promising node $q_{new_{goal}}$ towards the goal

After every local search, a new node is added to the tree.

References

References I

V. Alcázar, M. Veloso, and D. Borrajo.
Adapting a rapidly-exploring random tree for automated planning.
In *Fourth Annual Symposium on Combinatorial Search*, 2011.

D. Burfoot, J. Pineau, and G. Dudek.
Rrt-plan: A randomized algorithm for strips planning.
In *ICAPS*, pages 362–365, 2006.

L. Jaillet, J. Cortés, and T. Siméon.
Sampling-based path planning on configuration-space costmaps.
IEEE Transactions on Robotics, 26(4):635–646, 2010.

References II

S. Karaman and E. Frazzoli.
Sampling-based algorithms for optimal motion planning.
The international journal of robotics research, 30(7):846–894, 2011.

S. M. LaValle and J. J. Kuffner Jr.
Rapidly-exploring random trees: Progress and prospects.
2000.

S. Morgan and M. S. Branicky.
Sampling-based planning for discrete spaces.
In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 2, pages 1938–1945. IEEE, 2004.