Motion Planning Lecture 6

Tree-based and Asymptotically-Optimal Planning

Wolfgang Hönig (TU Berlin) and Andreas Orthey (Realtime Robotics)

May 29, 2024

Recap

Foundations

2 Weeks (problem formulation, terminology, collision checking)

Search-based

2 Weeks (A* and variants; state-lattice-based planning)

Sampling-based

5 Weeks (RRT, PRM, OMPL, Sampling Theory)

Optimization-based

2 Weeks (SCP, TrajOpt)

Current and Advanced Topics

3 Weeks (Comparative Analysis, Hybrid- and Multi-Robot approaches)

Today

- Tree-based motion planning (RRT)
- Introduction asymptotically optimal planning
- Optimal tree-based planning (RRT*, BIT*)

Tree-based motion planning

Rapidly-exploring random tree (RRT)

- Invented independently by Steve M. LaValle (1998) and David Hsu (1997)
- One of the most efficient algorithms for motion planning
- Growing a tree through random extensions

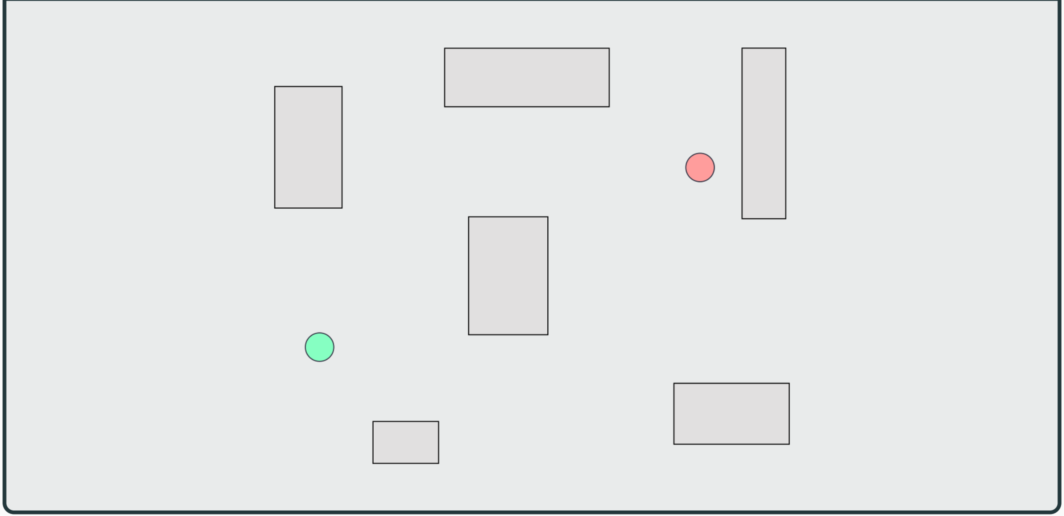
D Hsu, et al., "Path planning in expansive configuration spaces" (1999)
SM LaValle, "Rapidly-exploring random trees: A new tool for path planning", (1998)
JJ Kuffner, SM LaValle, "RRT-connect: An efficient approach to single-query path planning", (2000)

Pseudocode RRT

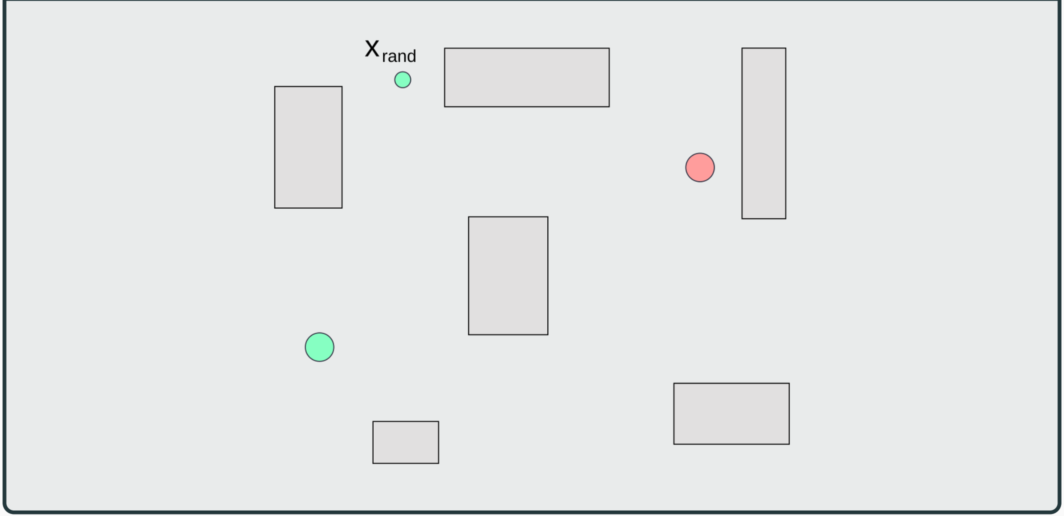
```
def RRT(xstart, xgoal, mu):
    V.AddNode(xstart)
    while not finished:
        xrand = SampleRandom()
        xnear = NearestNeighbor(xrand)
        xnew = Steer(xnear, xrand, mu)
        if xnear == xnew:
            continue
        V.AddNode(xnew)
        V.AddEdge(xnear, xnew)
        if Distance(xnew, xgoal) < Epsilon:
            return Path(xnew)
```

Rapidly-exploring random tree

Rapidly-exploring random tree (RRT)

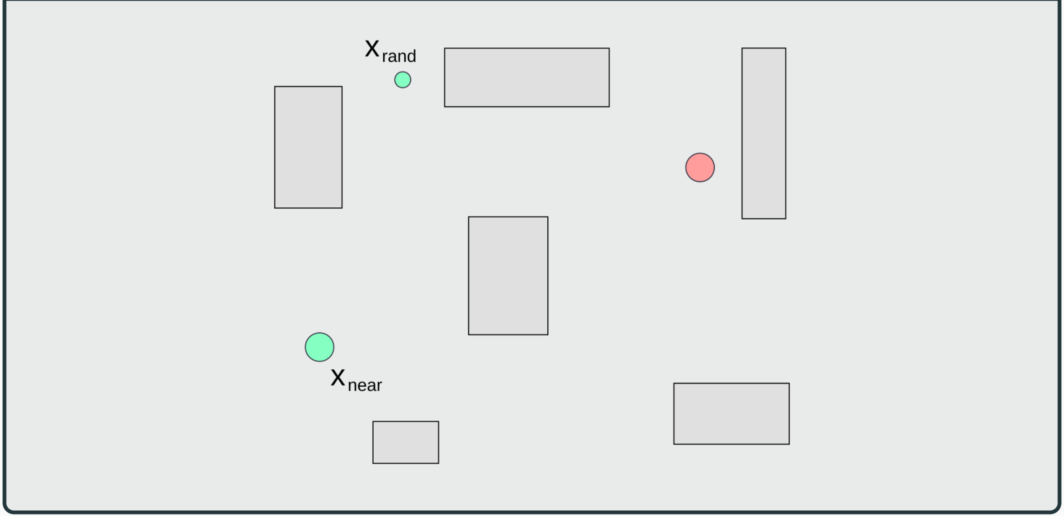


Rapidly-exploring random tree (RRT)

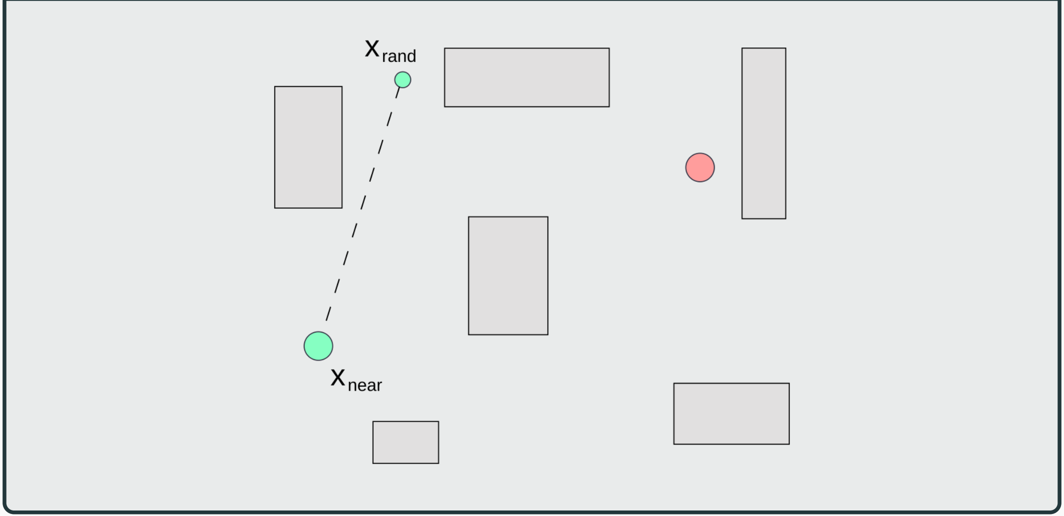


Rapidly-exploring random tree

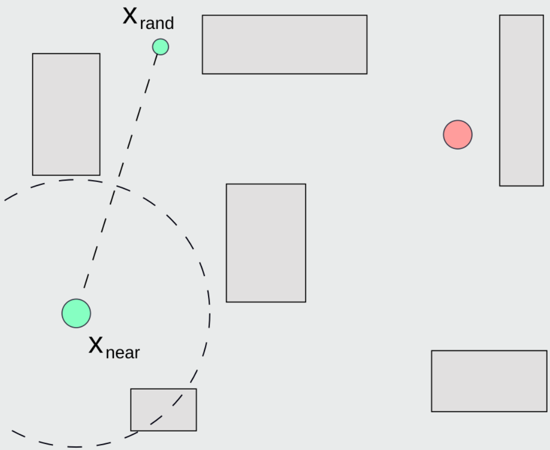
Rapidly-exploring random tree (RRT)



Rapidly-exploring random tree (RRT)

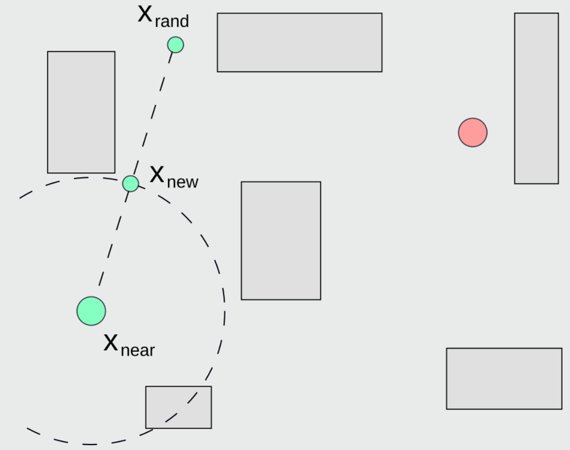


Rapidly-exploring random tree (RRT)

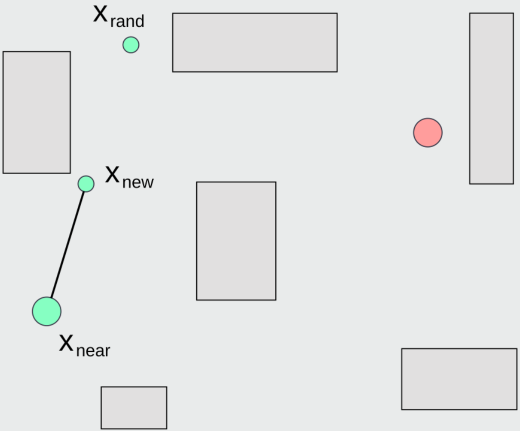


Rapidly-exploring random tree

Rapidly-exploring random tree (RRT)

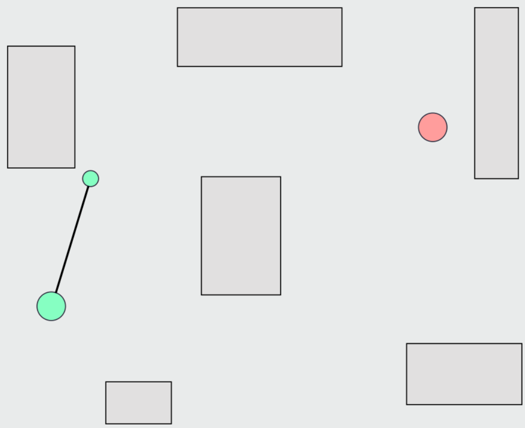


Rapidly-exploring random tree (RRT)

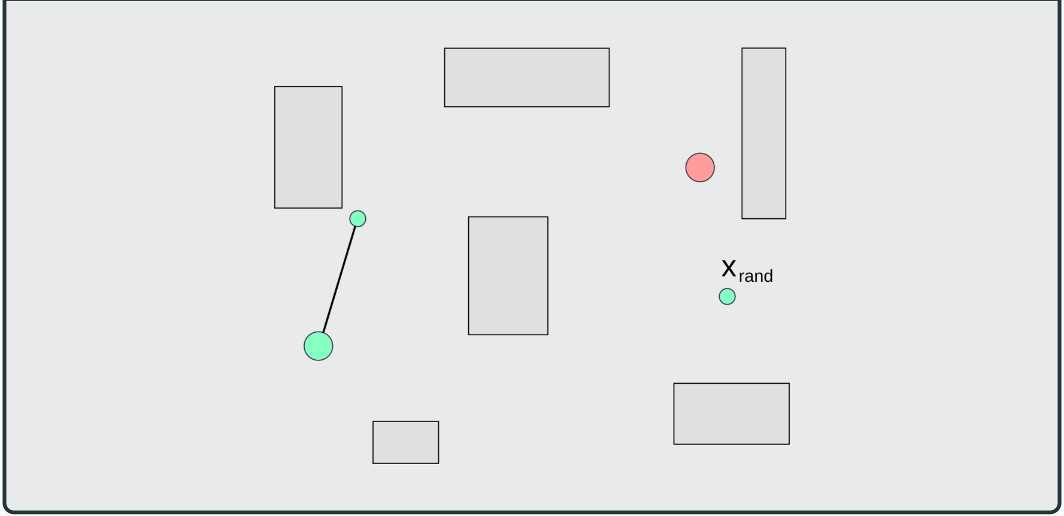


Rapidly-exploring random tree

Rapidly-exploring random tree (RRT)

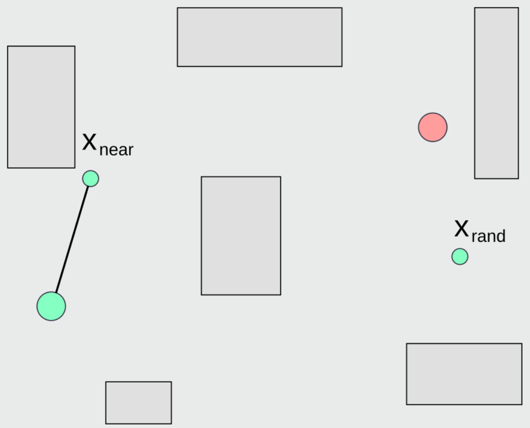


Rapidly-exploring random tree (RRT)



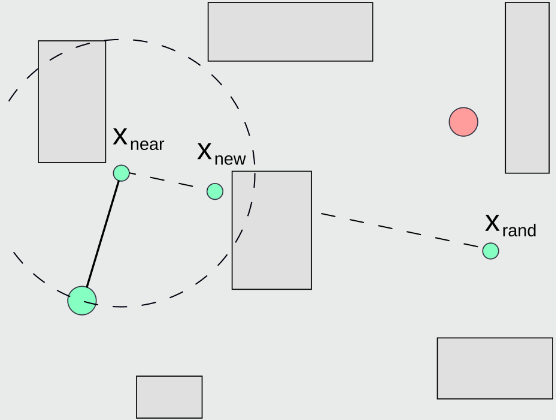
Rapidly-exploring random tree

Rapidly-exploring random tree (RRT)

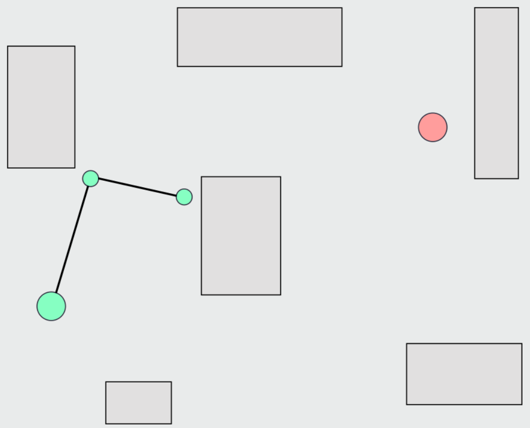


Rapidly-exploring random tree

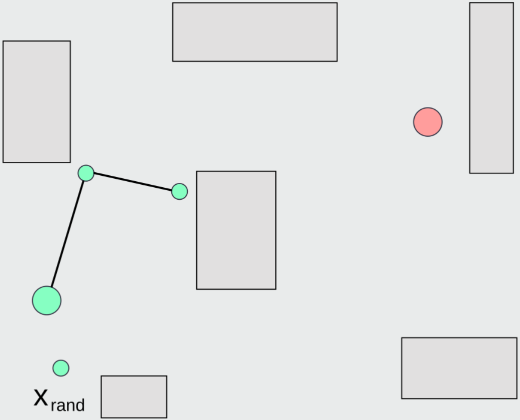
Rapidly-exploring random tree (RRT)



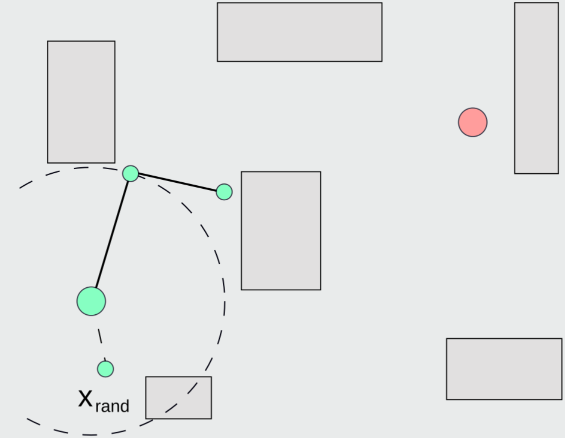
Rapidly-exploring random tree (RRT)



Rapidly-exploring random tree (RRT)

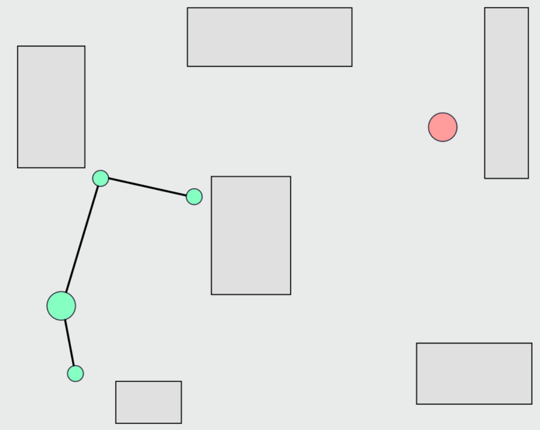


Rapidly-exploring random tree (RRT)



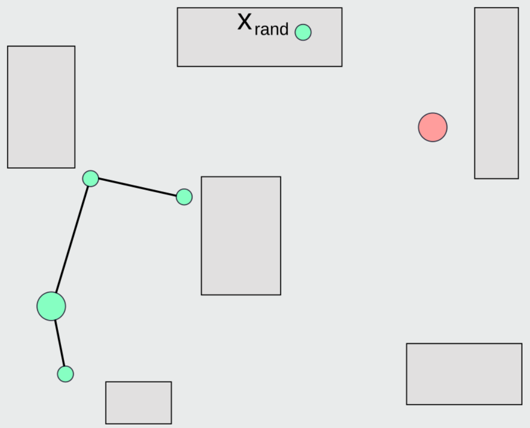
Rapidly-exploring random tree

Rapidly-exploring random tree (RRT)



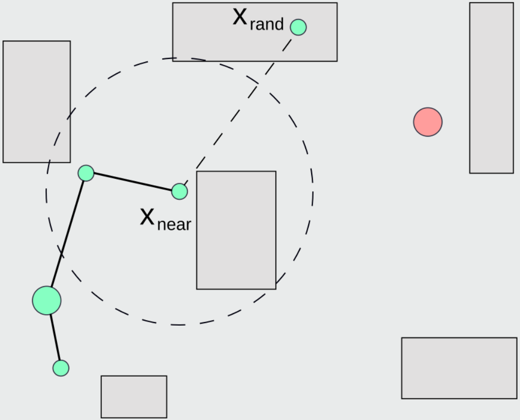
Rapidly-exploring random tree

Rapidly-exploring random tree (RRT)

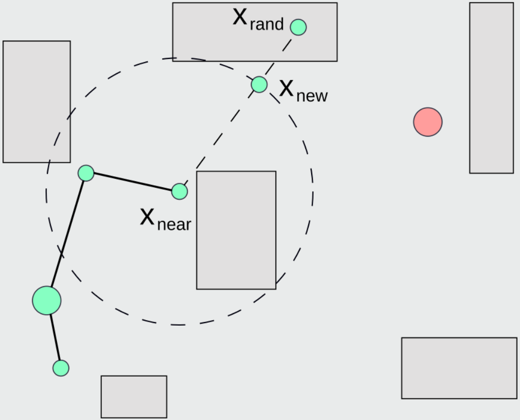


Rapidly-exploring random tree

Rapidly-exploring random tree (RRT)

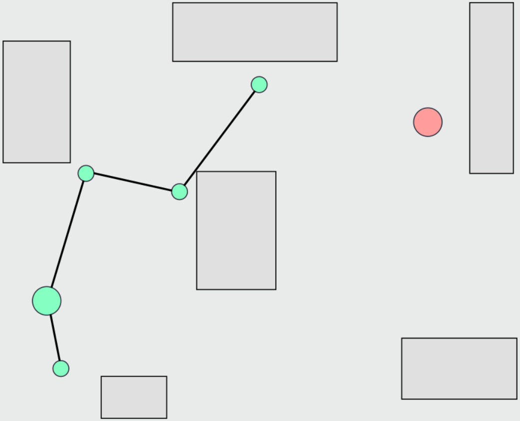


Rapidly-exploring random tree (RRT)



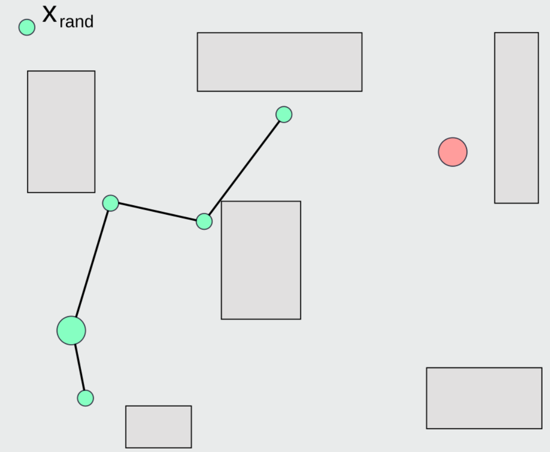
Rapidly-exploring random tree

Rapidly-exploring random tree (RRT)



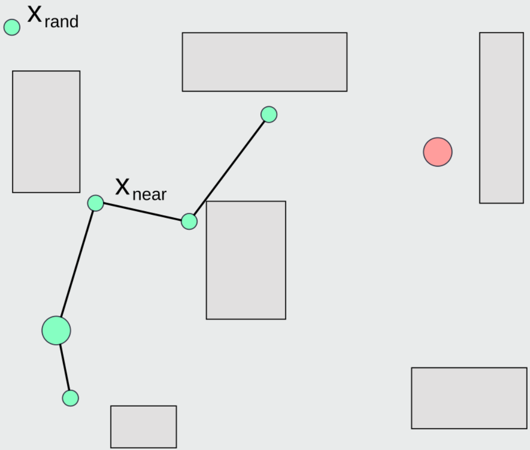
Rapidly-exploring random tree

Rapidly-exploring random tree (RRT)

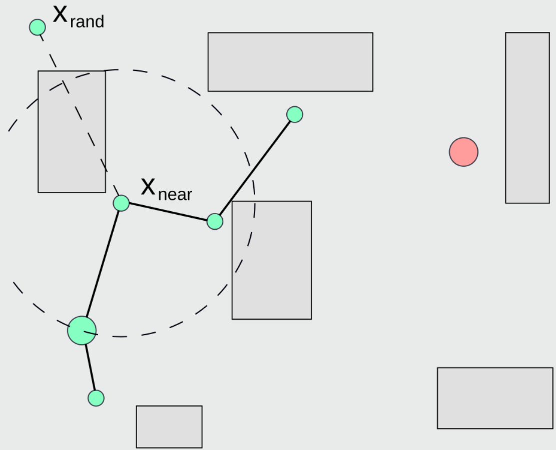


Rapidly-exploring random tree

Rapidly-exploring random tree (RRT)

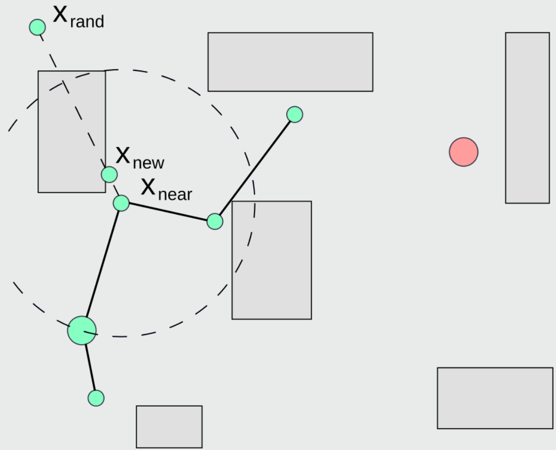


Rapidly-exploring random tree (RRT)



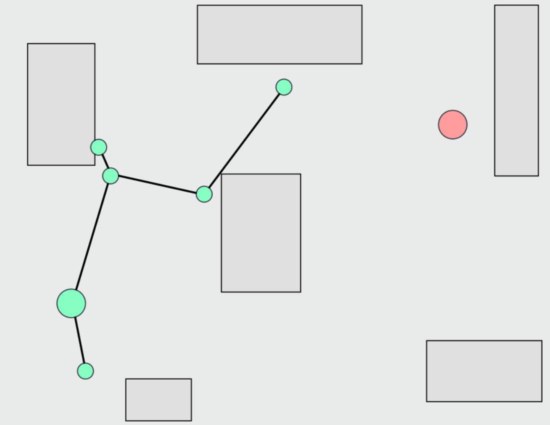
Rapidly-exploring random tree

Rapidly-exploring random tree (RRT)

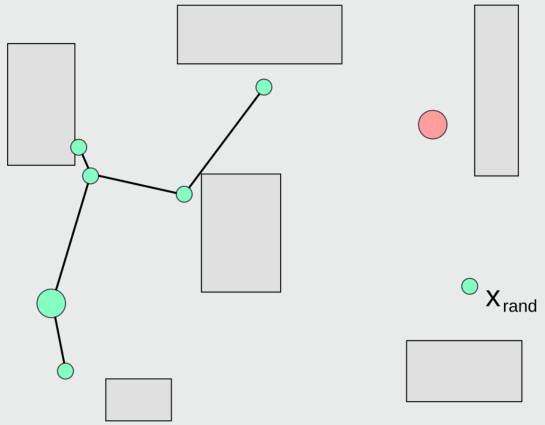


Rapidly-exploring random tree

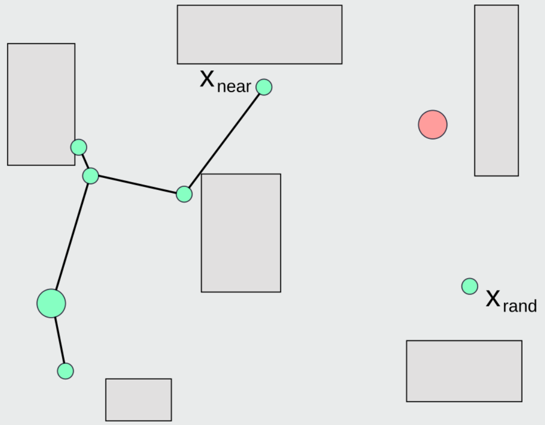
Rapidly-exploring random tree (RRT)



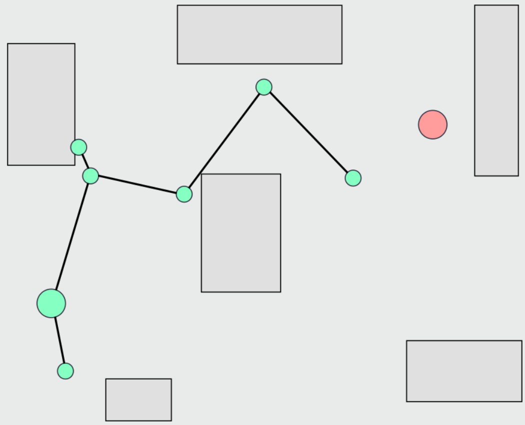
Rapidly-exploring random tree (RRT)



Rapidly-exploring random tree (RRT)

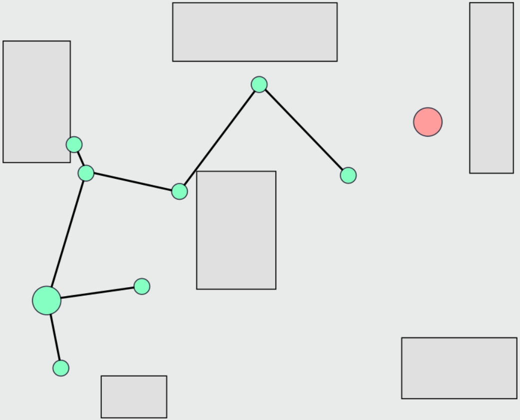


Rapidly-exploring random tree (RRT)



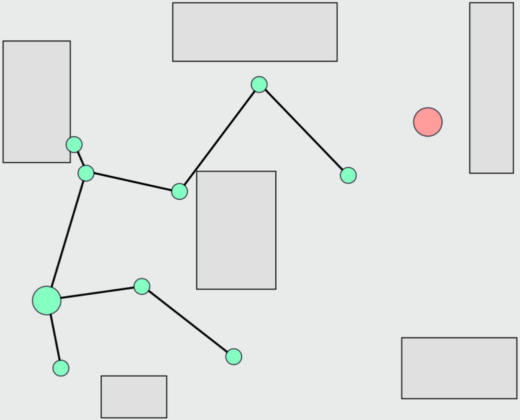
Rapidly-exploring random tree

Rapidly-exploring random tree (RRT)

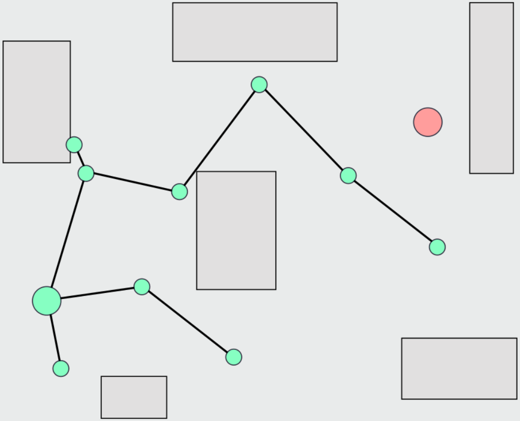


Rapidly-exploring random tree

Rapidly-exploring random tree (RRT)

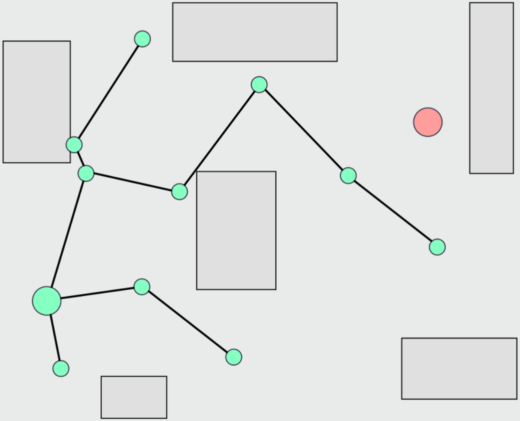


Rapidly-exploring random tree (RRT)

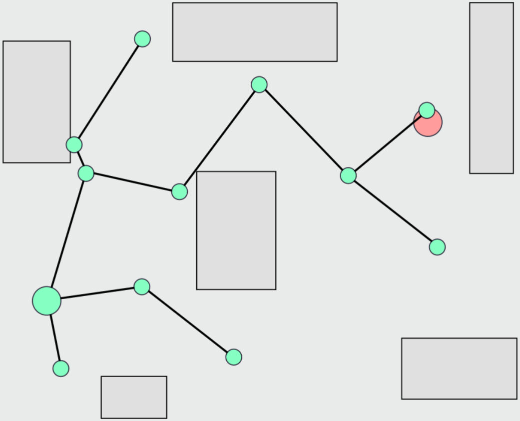


Rapidly-exploring random tree

Rapidly-exploring random tree (RRT)

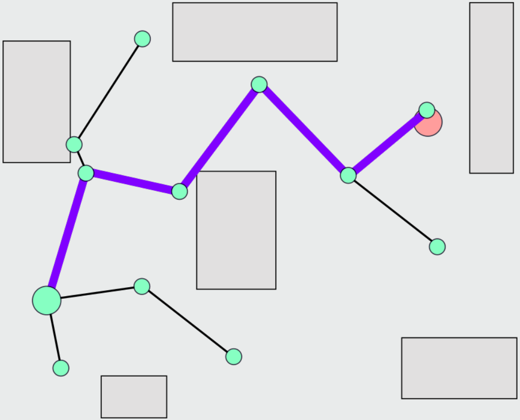


Rapidly-exploring random tree (RRT)



Rapidly-exploring random tree

Rapidly-exploring random tree (RRT)



Rapidly-exploring random tree (RRT)

Efficiency

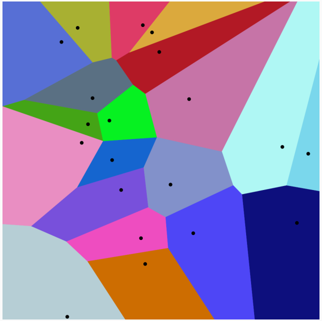
RRT is one of the most efficient planners because it has an implicit Voronoi bias

Rapidly-exploring random tree (RRT)

Voronoi region

Let $q_1, \ldots, q_K$ be a set of configurations on the state space $\mathcal{Q}$. The Voronoi region is defined as

$$R_k = \{q \in \mathcal{Q} \mid d(q, R_k) \leq d(q, R_j), \text{ for all } j \neq k\}$$



Rapidly-exploring random tree (RRT)

Voronoi bias

Probability of being selected is proportional to Voronoi region of a node in the tree.

Exploration/Exploitation trade-off.

45 iterations　　390 iterations

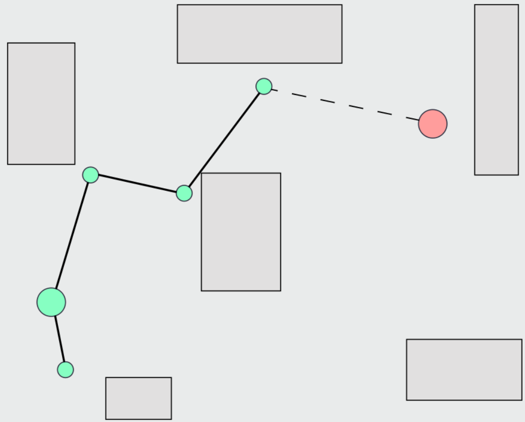
Rapidly-exploring random tree (RRT)

Improvements

- Extend tree towards goal
- Sample goal region (with probability μ)
- Bidirectional tree

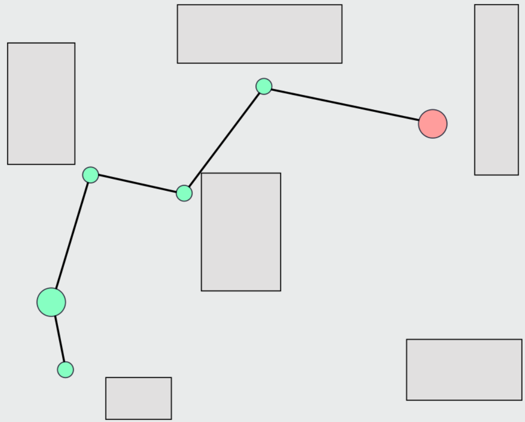
Improvement A - Extend towards goal

Extend towards goal



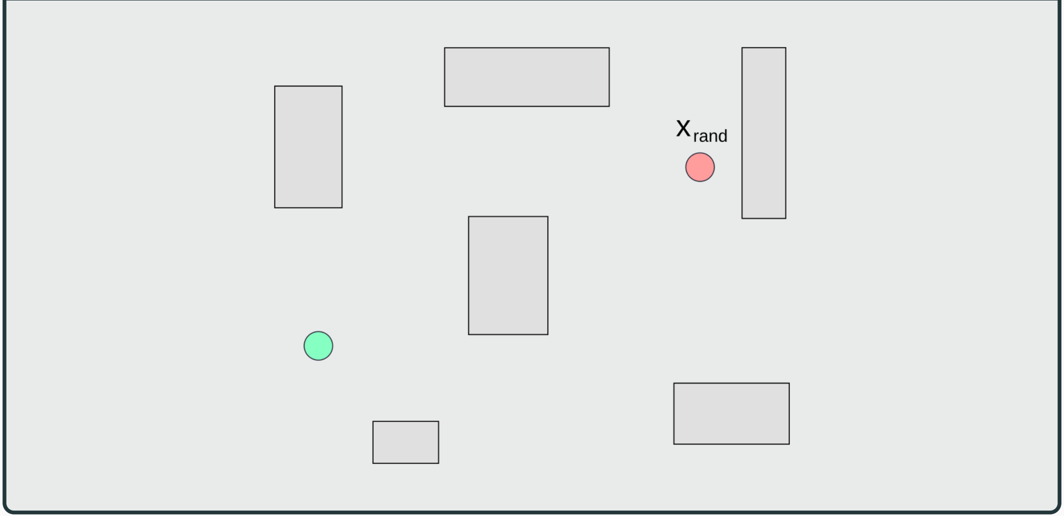
Improvement A - Extend towards goal

Extend towards goal



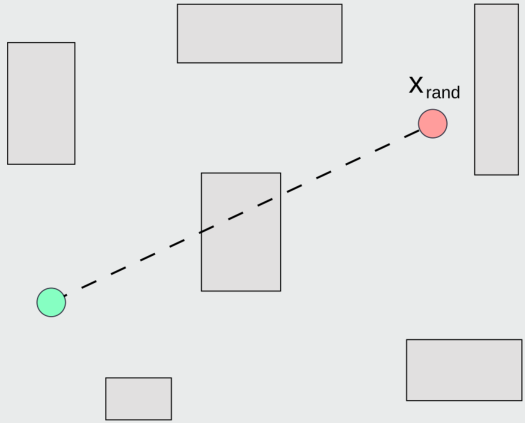
Improvement B - Sample goal region

Goal-bias



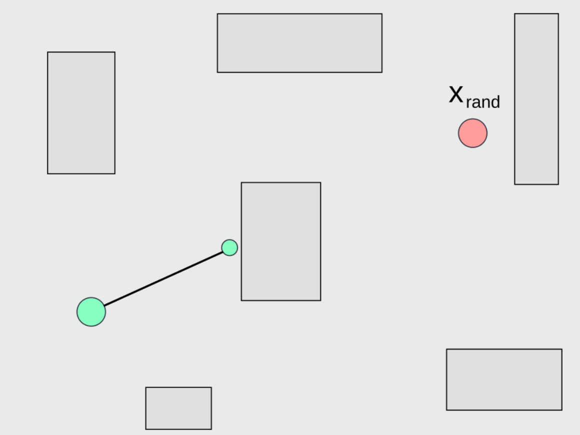
Improvement B - Sample goal region

Goal-bias



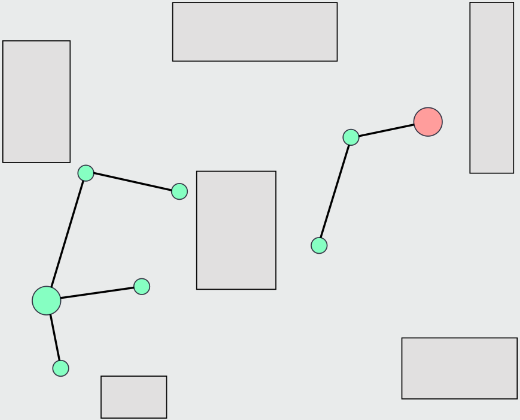
Improvement B - Sample goal region

Goal-bias



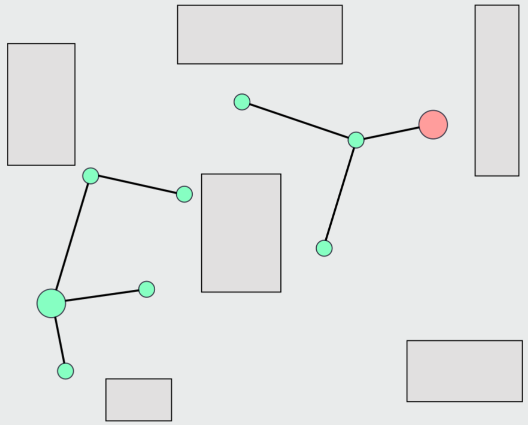
Improvement C - Bidirectional Tree

Bidirectional Rapidly-exploring random tree (Bi-RRT)



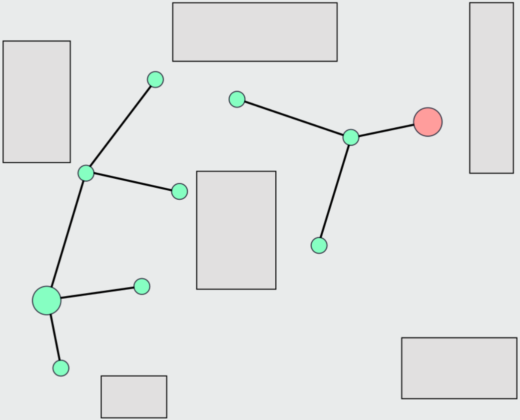
Improvement C - Bidirectional Tree

Bidirectional Rapidly-exploring random tree (Bi-RRT)



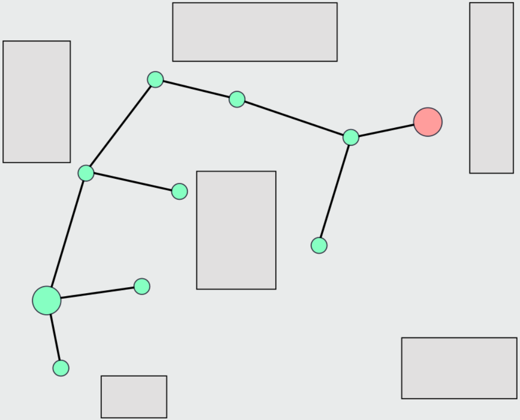
Improvement C - Bidirectional Tree

Bidirectional Rapidly-exploring random tree (Bi-RRT)



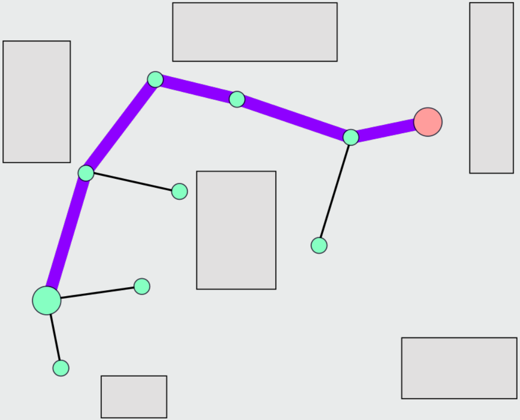
Improvement C - Bidirectional Tree

Bidirectional Rapidly-exploring random tree (Bi-RRT)



Improvement C - Bidirectional Tree

Bidirectional Rapidly-exploring random tree (Bi-RRT)



Rapidly-exploring random tree (RRT)

Further Improvements

- Path shortening after solution is found
- Multi-tree extension
- Targeted sampling

Introduction to Asymptotic Optimality Planning

Optimality

Question

What is optimality?

Optimality

Question

What is optimality?

Optimality (High-level)

The property of a planner to return a motion which surpasses all other motions in quality.

Optimality

Question

What is optimality?

Optimality (Mid-level)

From all possible paths, return the one which minimizes an objective function.

Optimality

Question

What is optimality?

Optimality (Low-level)

Given a motion planning problem $\mathcal{Q}, q_I, q_G$, find a solution path $p*$, which minimizes an objective cost function c, i.e. $c(p*) \leq c(p)$ for all p which solve the problem.

Optimality

Question

Why do we need optimality?

Aesthetics


R.O.B.O.T. Comics
A
B
JORGE CHAM 2009
WWW.WILLOWGARAGE.COM
"HIS PATH-PLANNING MAY BE SUB-OPTIMAL, BUT IT'S GOT FLAIR."

Efficiency

Safety

Coverage

Usefulness of Optimality

- Aesthetics: Should look good from an observer perspective
- Efficiency: Should find time optimal paths
- Safety: Should keep distance to prevent collisions
- Coverage: Should reach every point of the workspace

Optimality

Optimality principles also help us to search efficiently [1].

- A* heuristic: Prioritization search of best-cost paths VS. brute force search
- Pruning using necessary conditions

Introduction to Asymptotic Optimality Planning

Cost framework

Cost function types

Objective (or cost) function c. Graph $G = (V, E)$ and paths $P = (e_1, \ldots, e_N)$.

- Cost for a configuration $c : V \to \mathbb{R}_{\geq 0}$
- Cost of an edge $c : E \to \mathbb{R}_{\geq 0}$
- Cost of a path $c : P \to \mathbb{R}_{\geq 0}$

Cost functions examples

Shortest length

- Configuration cost: Zero
- Edge cost: Length of segment, metric distance
- Path cost: Sum of edge costs

Cost functions examples

Maximum clearance

- Configuration cost: Distance from robot to environment
- Edge cost: Maximum over all configurations on edge
- Path cost: Maximum over all edges on path

Cost functions examples

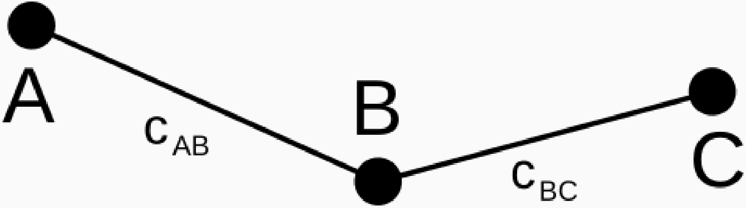
Lowest energy

- Configuration cost: Zero
- Edge cost: Energy spent going from A to B
- Path cost: Sum of edge energies

Costs

Additive costs

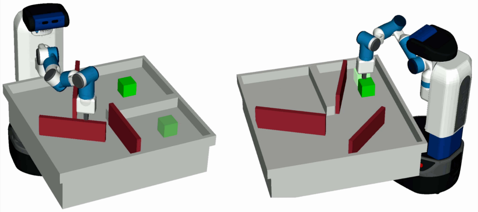
- Note: Most planners like RRT*, BIT* require additive cost!
- Additive cost: cost(A,B,C) = cost(A,B) + cost(B,C)



Costs

Non-additive cost example

Number of objects manipulated by a robot manipulator

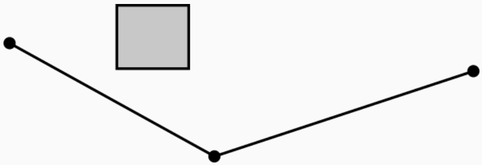


Bayraktar et al., "Solving Rearrangement Puzzles using Path Defragmentation in Factored State Spaces", Robotics and Automation Letters (RA-L), 2023

Costs

Non-additive cost example

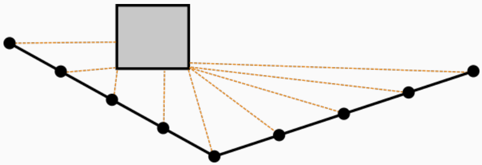
Average clearance cost.



Costs

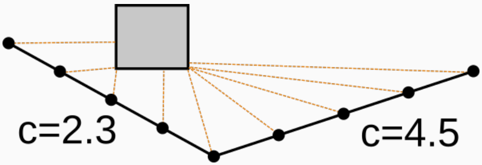
Non-additive cost example

Average clearance cost.



Non-additive cost example

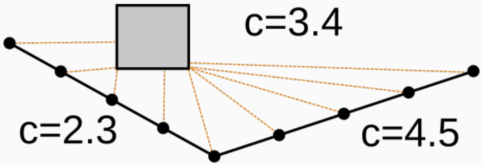
Average clearance cost.



Costs

Non-additive cost example

Average clearance cost.



Cost framework

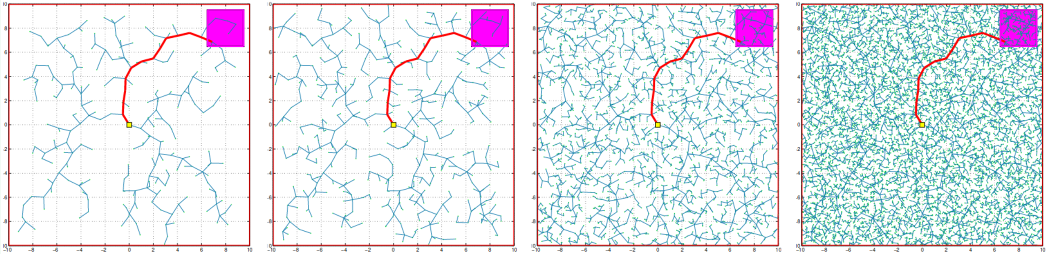
Cost framework recap

- What: Best possible motion
- Why: Aesthetics, Efficiency, Safety, Coverag, Optimality for efficient search
- How: Cost framework, additive costs

Optimal tree-based motion planning

RRT and Optimality (1)

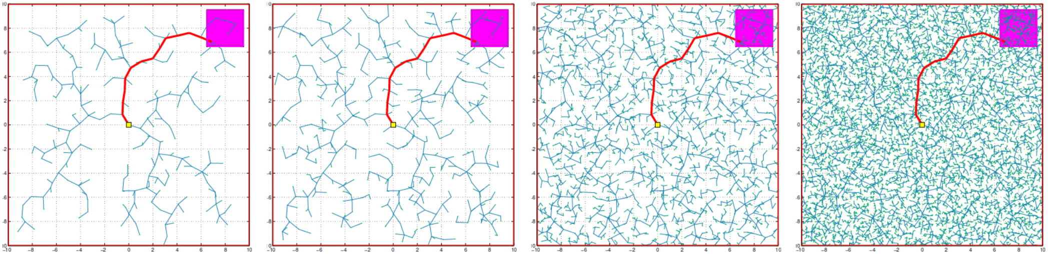
What if we keep running RRT?



Source: [2]

RRT and Optimality (1)

What if we keep running RRT?

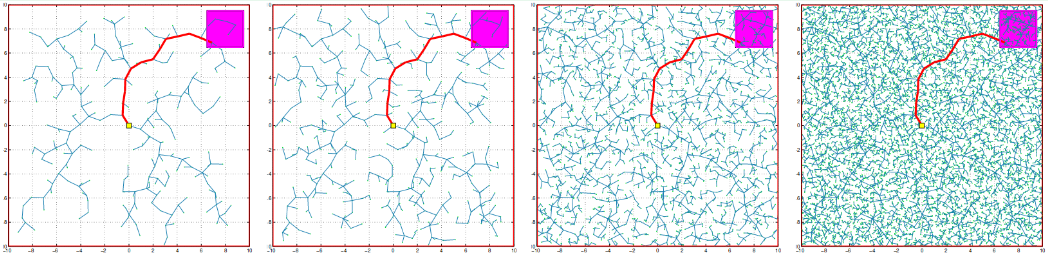


Source: [2]

What causes this?

RRT and Optimality (1)

What if we keep running RRT?



Source: [2]

What causes this?

Edges are only added, never changed (rewired).

RRT and Optimality (2)

Sampling-based algorithms for optimal motion planning

S Karaman, E Frazzoli - The international journal of robotics …, 2011 - journals.sagepub.com

During the last decade, sampling-based path planning algorithms, such as probabilistic roadmaps (PRM) and rapidly exploring random trees (RRT), have been shown to work well …

☆ Save 99 Cite Cited by 3551 Related articles

RRT and Optimality (2)

Sampling-based algorithms for optimal motion planning
S Karaman, E Frazzoli - The international journal of robotics …, 2011 - journals.sagepub.com
During the last decade, sampling-based path planning algorithms, such as probabilistic roadmaps (PRM) and rapidly exploring random trees (RRT), have been shown to work well …
☆ Save 99 Cite Cited by 3551 Related articles

RRT is Suboptimal [2, Theorem 33]

The cost of the best solution returned by RRT converges to a suboptimal value, with probability one:

$$\mathbb{P}\left(\left\{\lim_{n\to\infty} Y_n^{RRT} > c^*\right\}\right) = 1.$$

RRT*: RRT with Rewiring

Algorithm 2 RRT* $(x_{\texttt{init}} := s, x_{\texttt{goal}} := t, n, r, \eta)$

$V = \{x_{\text{init}}\}$
for $j = 1$ to n **do**
    $x_{\text{rand}} \leftarrow$ SAMPLE-FREE( )
    $x_{\text{near}} \leftarrow \text{NEAREST}(x_{\text{rand}}, V)$
    $x_{\text{new}} \leftarrow \text{STEER}(x_{\text{near}}, x_{\text{rand}}, \eta)$
    if COLLISION-FREE$(x_{\text{near}}, x_{\text{new}})$ **then**
        $X_{\text{near}} = \text{NEAR}(x_{\text{new}}, V, \min\{r(|V|), \eta\})$
        $V = V \cup \{x_{\text{new}}\}$
        $x_{\text{min}} = x_{\text{near}}$
        $c_{\text{min}} = \text{COST}(x_{\text{near}}) + \|x_{\text{new}} - x_{\text{near}}\|$
        for $x_{\text{near}} \in X_{\text{near}}$ **do**
            if COLLISION-FREE$(x_{\text{near}}, x_{\text{new}})$ **then**
                if $\text{COST}(x_{\text{near}}) + \|x_{\text{new}} - x_{\text{near}}\| < c_{\text{min}}$ **then**
                    $x_{\text{min}} = x_{\text{near}}$
                    $c_{\text{min}} = \text{COST}(x_{\text{near}}) + \|x_{\text{new}} - x_{\text{near}}\|$
        $E = E \cup \{(x_{\text{min}}, x_{\text{new}})\}$
        for $x_{\text{near}} \in X_{\text{near}}$ **do**
            if COLLISION-FREE$(x_{\text{new}}, x_{\text{near}})$ **then**
                if $\text{COST}(x_{\text{new}}) + \|x_{\text{near}} - x_{\text{new}}\| < \text{COST}(x_{\text{near}})$ **then**
                    $x_{\text{parent}} = \text{PARENT}(x_{\text{near}})$
                    $E = E \cup \{(x_{\text{new}}, x_{\text{near}})\} \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$
return $G = (V, E)$

- Pseudo code from [3]

RRT*: RRT with Rewiring

Algorithm 2 RRT* $(x_{\texttt{init}} := s, x_{\texttt{goal}} := t, n, r, \eta)$

$V = \{x_{\text{init}}\}$
for $j = 1$ to n **do**
  $x_{\text{rand}} \leftarrow$ SAMPLE-FREE( )
  $x_{\text{near}} \leftarrow$ NEAREST(x_{rand}, V)
  $x_{\text{new}} \leftarrow$ STEER$(x_{\text{near}}, x_{\text{rand}}, \eta)$
  if COLLISION-FREE$(x_{\text{near}}, x_{\text{new}})$ **then**
    $X_{\text{near}} = \text{NEAR}(x_{\text{new}}, V, \min\{r(|V|), \eta\})$
    $V = V \cup \{x_{\text{new}}\}$
    $x_{\text{min}} = x_{\text{near}}$
    $c_{\text{min}} = \text{COST}(x_{\text{near}}) + \|x_{\text{new}} - x_{\text{near}}\|$
    for $x_{\text{near}} \in X_{\text{near}}$ **do**
      if COLLISION-FREE$(x_{\text{near}}, x_{\text{new}})$ **then**
        if $\text{COST}(x_{\text{near}}) + \|x_{\text{new}} - x_{\text{near}}\| < c_{\text{min}}$ **then**
          $x_{\text{min}} = x_{\text{near}}$
          $c_{\text{min}} = \text{COST}(x_{\text{near}}) + \|x_{\text{new}} - x_{\text{near}}\|$
    $E = E \cup \{(x_{\text{min}}, x_{\text{new}})\}$
    for $x_{\text{near}} \in X_{\text{near}}$ **do**
      if COLLISION-FREE$(x_{\text{new}}, x_{\text{near}})$ **then**
        if $\text{COST}(x_{\text{new}}) + \|x_{\text{near}} - x_{\text{new}}\| < \text{COST}(x_{\text{near}})$ **then**
          $x_{\text{parent}} = \text{PARENT}(x_{\text{near}})$
          $E = E \cup \{(x_{\text{new}}, x_{\text{near}})\} \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$
return $G = (V, E)$

- Pseudo code from [3]
- Parent of $\mathbf{q}_{new}$: May use other parent than $\mathbf{q}_{near}$ with lowest cost (within neighborhood of $\mathbf{q}_{new}$)

RRT*: RRT with Rewiring

Algorithm 2 $\mathtt{RRT}^*$ $(x_{\mathtt{init}} := s, x_{\mathtt{goal}} := t, n, r, \eta)$

$V = \{x_{\text{init}}\}$
for $j = 1$ to n **do**
  $x_{\text{rand}} \leftarrow \text{SAMPLE-FREE}(\,)$
  $x_{\text{near}} \leftarrow \text{NEAREST}(x_{\text{rand}}, V)$
  $x_{\text{new}} \leftarrow \text{STEER}(x_{\text{near}}, x_{\text{rand}}, \eta)$
  if COLLISION-FREE$(x_{\text{near}}, x_{\text{new}})$ **then**
    $X_{\text{near}} = \text{NEAR}(x_{\text{new}}, V, \min\{r(|V|), \eta\})$
    $V = V \cup \{x_{\text{new}}\}$
    $x_{\text{min}} = x_{\text{near}}$
    $c_{\text{min}} = \text{COST}(x_{\text{near}}) + \|x_{\text{new}} - x_{\text{near}}\|$
    for $x_{\text{near}} \in X_{\text{near}}$ **do**
      if COLLISION-FREE$(x_{\text{near}}, x_{\text{new}})$ **then**
        if $\text{COST}(x_{\text{near}}) + \|x_{\text{new}} - x_{\text{near}}\| < c_{\text{min}}$ **then**
          $x_{\text{min}} = x_{\text{near}}$
          $c_{\text{min}} = \text{COST}(x_{\text{near}}) + \|x_{\text{new}} - x_{\text{near}}\|$
    $E = E \cup \{(x_{\text{min}}, x_{\text{new}})\}$
    for $x_{\text{near}} \in X_{\text{near}}$ **do**
      if COLLISION-FREE$(x_{\text{new}}, x_{\text{near}})$ **then**
        if $\text{COST}(x_{\text{new}}) + \|x_{\text{near}} - x_{\text{new}}\| < \text{COST}(x_{\text{near}})$ **then**
          $x_{\text{parent}} = \text{PARENT}(x_{\text{near}})$
          $E = E \cup \{(x_{\text{new}}, x_{\text{near}})\} \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$
return $G = (V, E)$

- Pseudo code from [3]
- Parent of $\mathbf{q}_{new}$: May use other parent than $\mathbf{q}_{near}$ with lowest cost (within neighborhood of $\mathbf{q}_{new}$)
- Rewire edges: Use $\mathbf{q}_{new}$ as a new parent, for neighboring configurations, if it reduces costs

RRT*: RRT with Rewiring

Algorithm 2 RRT* $(x_{\mathtt{init}} := s, x_{\mathtt{goal}} := t, n, r, \eta)$

$V = \{x_{\text{init}}\}$
for $j = 1$ to n **do**
  $x_{\text{rand}} \leftarrow \text{SAMPLE-FREE}(\,)$
  $x_{\text{near}} \leftarrow \text{NEAREST}(x_{\text{rand}}, V)$
  $x_{\text{new}} \leftarrow \text{STEER}(x_{\text{near}}, x_{\text{rand}}, \eta)$
  if COLLISION-FREE$(x_{\text{near}}, x_{\text{new}})$ **then**
    $X_{\text{near}} = \text{NEAR}(x_{\text{new}}, V, \min\{r(|V|), \eta\})$
    $V = V \cup \{x_{\text{new}}\}$
    $x_{\text{min}} = x_{\text{near}}$
    $c_{\text{min}} = \text{COST}(x_{\text{near}}) + \|x_{\text{new}} - x_{\text{near}}\|$
    for $x_{\text{near}} \in X_{\text{near}}$ **do**
      if COLLISION-FREE$(x_{\text{near}}, x_{\text{new}})$ **then**
        if $\text{COST}(x_{\text{near}}) + \|x_{\text{new}} - x_{\text{near}}\| < c_{\text{min}}$ **then**
          $x_{\text{min}} = x_{\text{near}}$
          $c_{\text{min}} = \text{COST}(x_{\text{near}}) + \|x_{\text{new}} - x_{\text{near}}\|$
    $E = E \cup \{(x_{\text{min}}, x_{\text{new}})\}$
    for $x_{\text{near}} \in X_{\text{near}}$ **do**
      if COLLISION-FREE$(x_{\text{new}}, x_{\text{near}})$ **then**
        if $\text{COST}(x_{\text{new}}) + \|x_{\text{near}} - x_{\text{new}}\| < \text{COST}(x_{\text{near}})$ **then**
          $x_{\text{parent}} = \text{PARENT}(x_{\text{near}})$
          $E = E \cup \{(x_{\text{new}}, x_{\text{near}})\} \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$
return $G = (V, E)$

- Pseudo code from [3]
- Parent of $\mathbf{q}_{new}$: May use other parent than $\mathbf{q}_{near}$ with lowest cost (within neighborhood of $\mathbf{q}_{new}$)
- Rewire edges: Use $\mathbf{q}_{new}$ as a new parent, for neighboring configurations, if it reduces costs
- Neighborhood radius depends on tree size:

$$r(|\mathcal{V}|) = \gamma \left(\frac{\log |\mathcal{V}|}{|\mathcal{V}|} \right)^{\frac{1}{d+1}}$$

Rewiring

Rewiring

If we add a new configuration x, we execute two rewiring operations:

- Rewire x to best parent
- Rewire all children nodes

Pseudocode tree rewiring

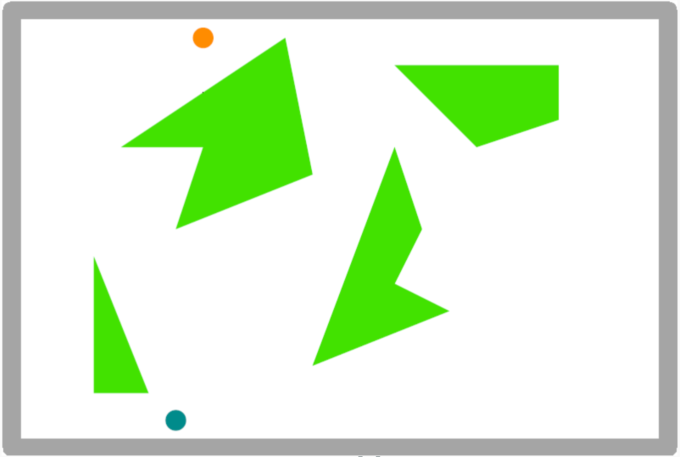
```
def Rewire(x):
  N = Neighbours(x)
  for x_n in N:
    Rewire(x_n, x)
  for x_n in N:
    Rewire(x, x_n)

def Rewire(x, y):
  p = Steer(x, y)
  if ConstraintFree(p):
    if cost(x)+cost(p) < cost(y):
      y.parent = x
```

Pseudocode RRT

```
def RRT(xstart, xgoal, mu):
    V.AddNode(xstart)
    while not finished:
        xrand = SampleRandom()
        xnear = NearestNeighbor(xrand)
        xnew = Steer(xnear, xrand, mu)
        if xnear == xnew:
            continue
        V.AddNode(xnew)
        V.AddEdge(xnear, xnew)
        Rewire(xnew) ##Rewiring operation to make it AO
        if Distance(xnew, xgoal) < Epsilon:
            return Path(xnew)
```

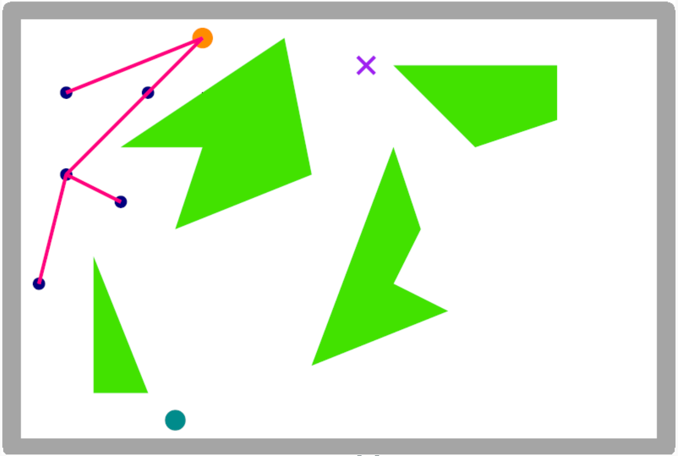
RRT* Example (1)



Source: [4]

Motion planning problem (orange = $\mathbf{q}_{start}$)

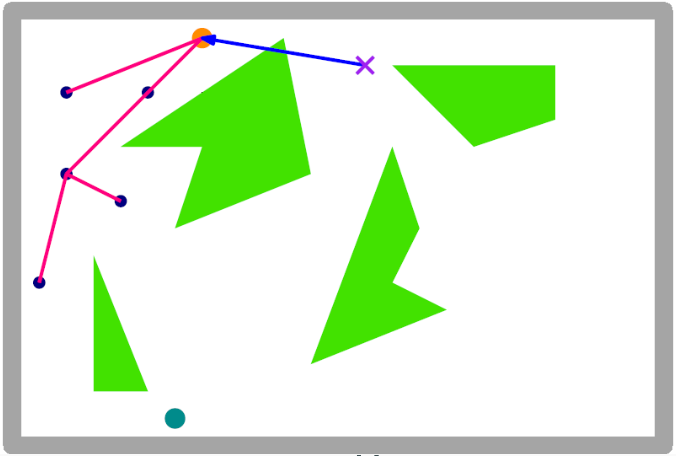
RRT* Example (2)



Source: [4]

Intermediate tree and new sample $\mathbf{q}_{rand}$ (purple $\times$)

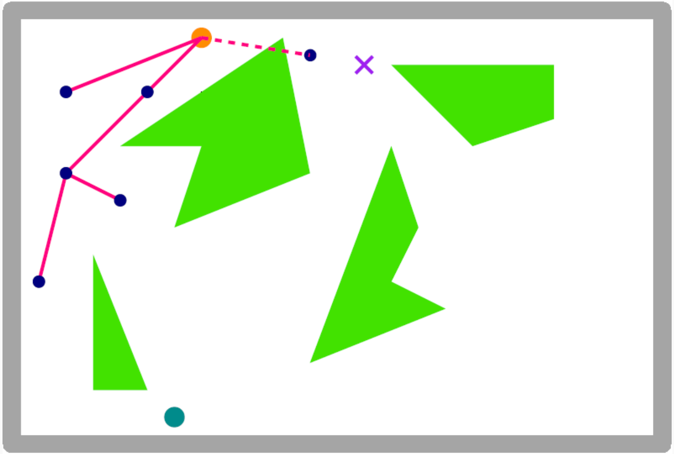
RRT* Example (3)



Source: [4]

Nearest $\mathbf{q}_{near}$ in existing tree is found (here: $\mathbf{q}_{start}$)

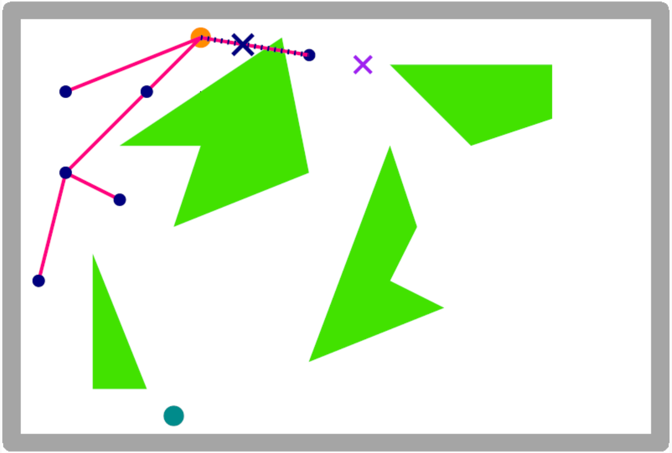
RRT* Example (4)



Source: [4]

Steer computes $\mathbf{q}_{new}$ on the line from $\mathbf{q}_{start}$ to $\mathbf{q}_{rand}$

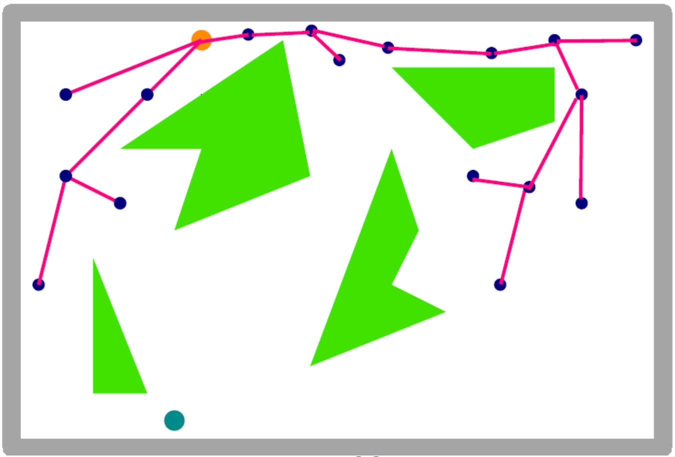
RRT* Example (5)



Source: [4]

New edge is rejected (not collision-free)

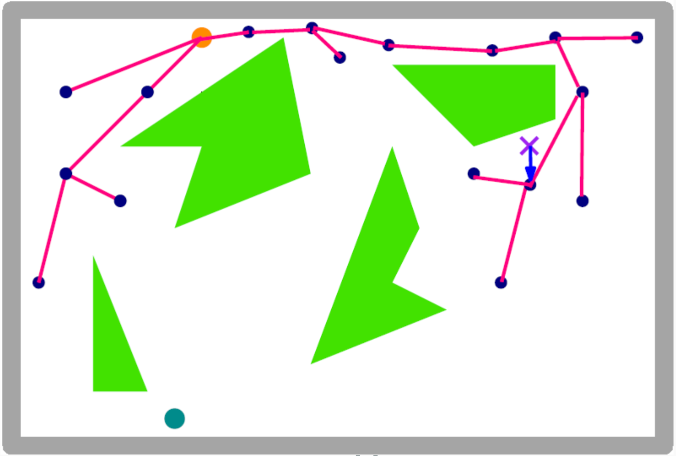
RRT* Example (6)



Source: [4]

So far behavior is exactly the same as RRT; Fast-forward we have a larger tree

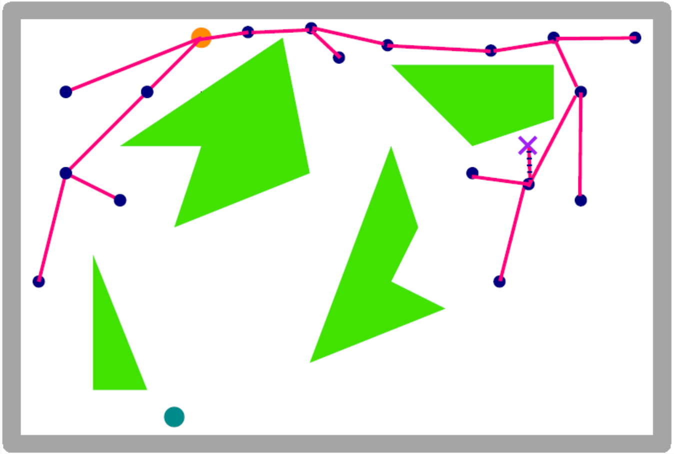
RRT* Example (7)



Source: [4]

New sample $\mathbf{q}_{rand}$ and closest node in the tree $\mathbf{q}_{near}$

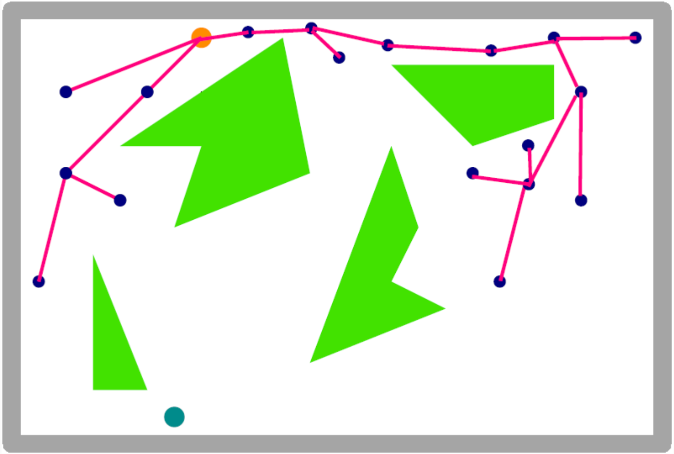
RRT* Example (8)



Source: [4]

Resulting edge $(\mathbf{q}_{new}, \mathbf{q}_{near})$ is collision-free

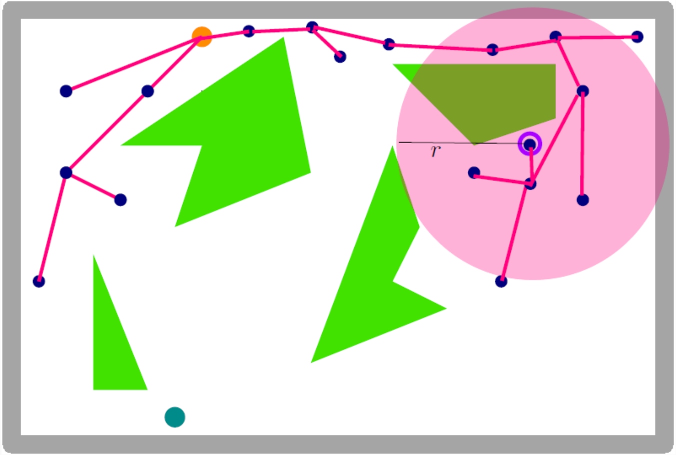
RRT* Example (9)



Source: [4]

This edge would be added in RRT

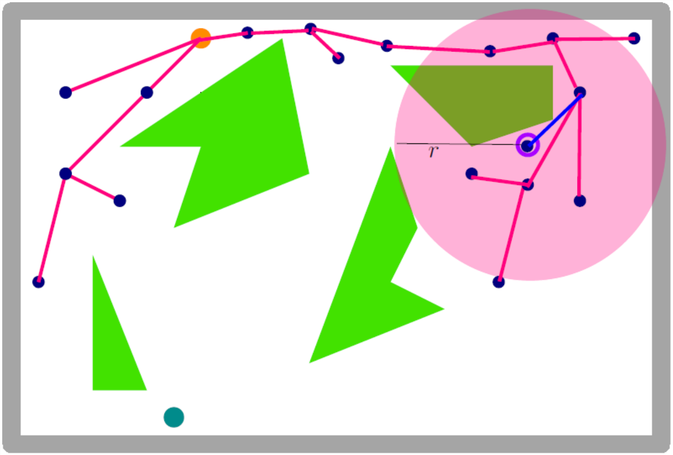
RRT* Example (10)



Source: [4]

RRT*: Consider all configuration of the tree in the neighborhood of $\mathbf{q}_{new}$

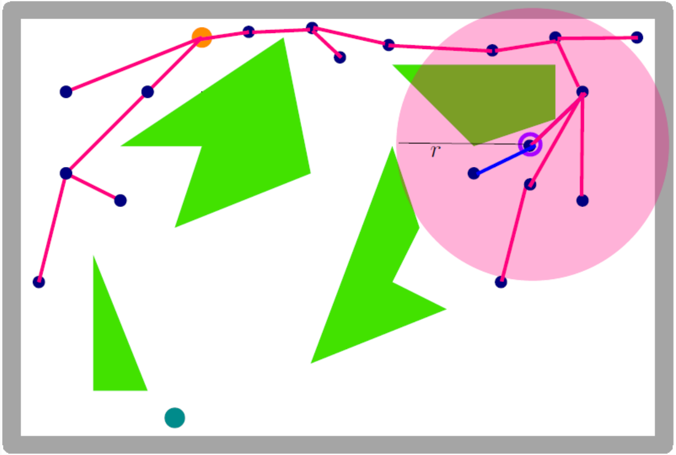
RRT* Example (11)



Source: [4]

RRT*: Use a lower-cost parent for $\mathbf{q}_{new}$ (other than $\mathbf{q}_{near}$)

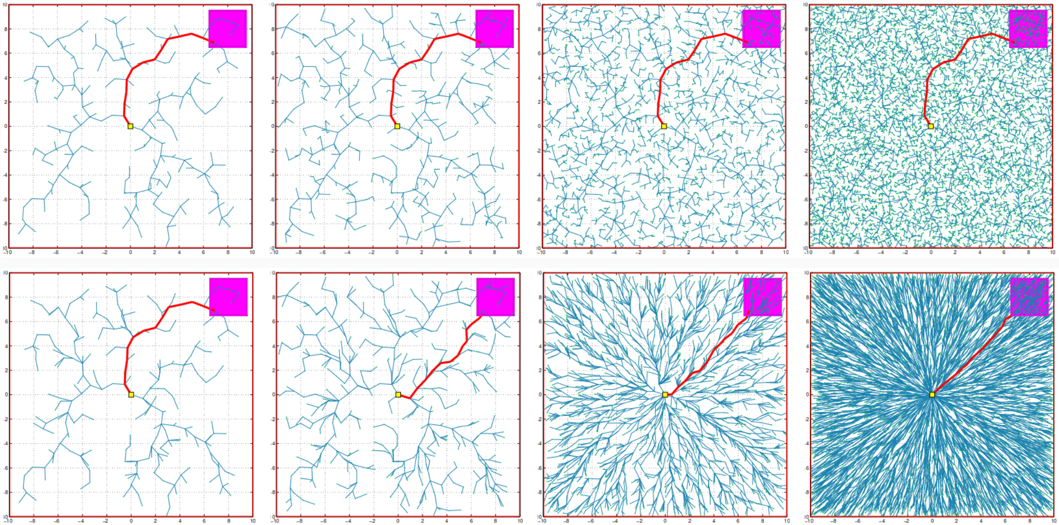
RRT* Example (12)



Source: [4]

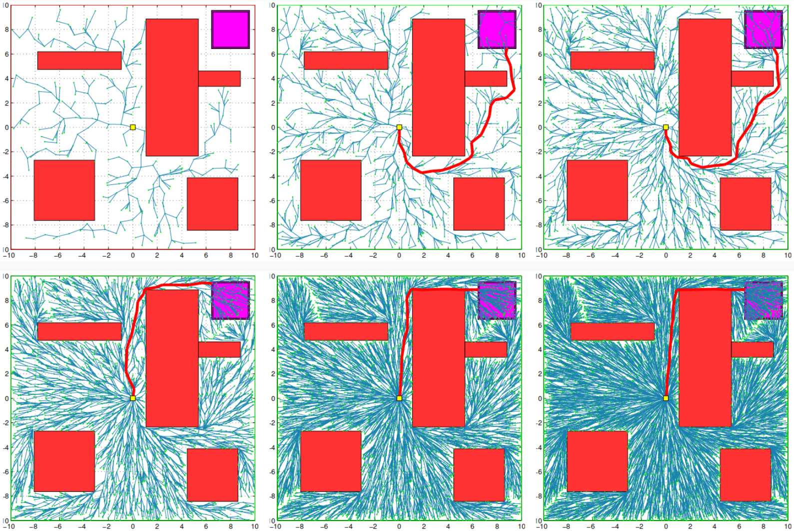
RRT*: Rewire the neighbors to use $\mathbf{q}_{new}$ as a parent to reduce the cost

RRT vs. RRT* (1)



Source: [2]

RRT vs. RRT* (2)



Source: [2]

RRT* Properties

RRT* is asymptotically optimal

The probability that the solution cost of RRT* is not more than $(1+\epsilon)c^*$ is 1, as the number of iterations go to infinity:

$$\lim_{n\to\infty} \mathbb{P}(\{c_n - c^* > \epsilon\}) = 0,\ \forall \epsilon > 0.$$

RRT* Properties

RRT* is asymptotically optimal

The probability that the solution cost of RRT* is not more than $(1 + \epsilon)c^*$ is 1, as the number of iterations go to infinity:

$$\lim_{n \to \infty} \mathbb{P}(\{c_n - c^* > \epsilon\}) = 0, \; \forall \epsilon > 0.$$

However, the convergence rate is unknown!

RRT* vs RRT

- Why is RRT probabilistically complete?
- Why is RRT not asymptotically optimal?
- Why is RRT* asymptotically optimal?

Optimal tree-based motion planning

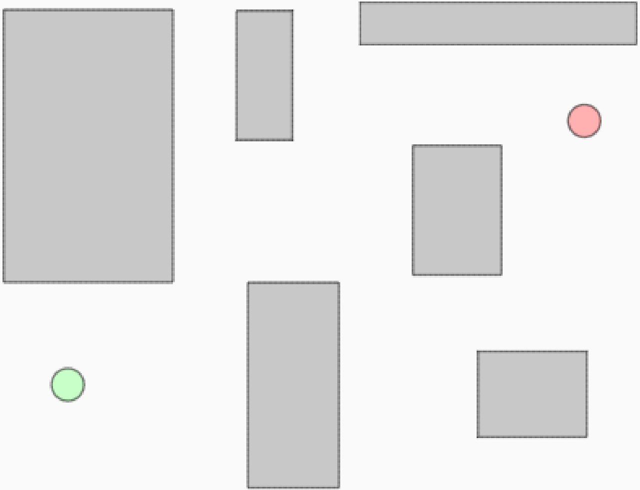
Probabilistic completeness proof RRT

Proof Sketch

Probabilistic Completeness RRT

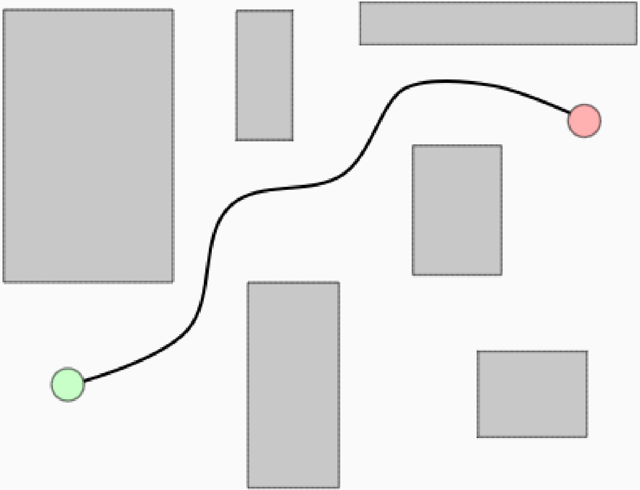
- A planner is probabilistic complete if it finds a solution if one exists.
- Main proof for RRT is based on induction.
- Requires number of samples going to infinity.

Proof sketch



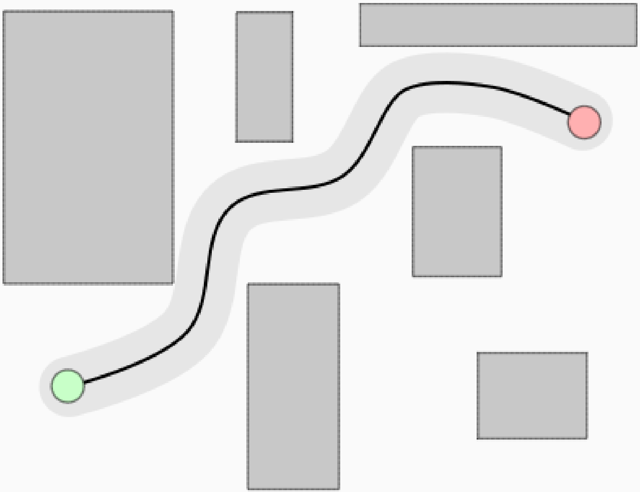
Petr Svestka, "On Probabilistic Completeness and Expected Complexity of Probabilistic Path Planning", 1998 [**svestka˙1998**]

Proof sketch



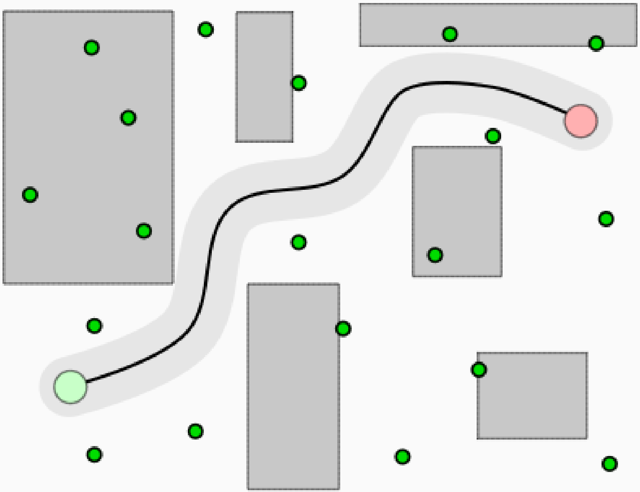
Assumption A: There exists a feasible path.

Proof sketch



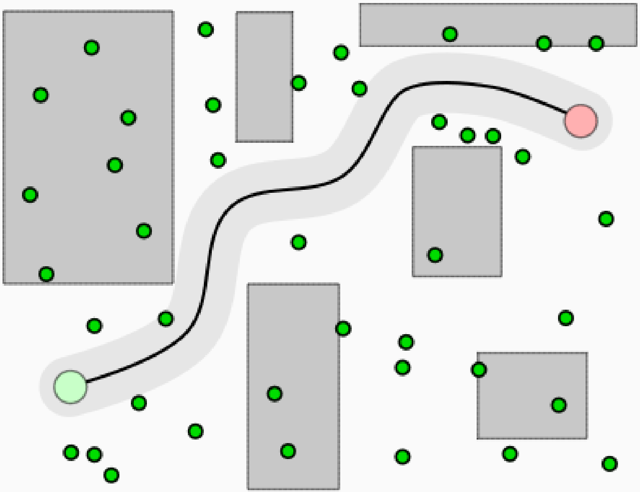
Assumption B: Feasible path has ϵ clearance.

Proof sketch



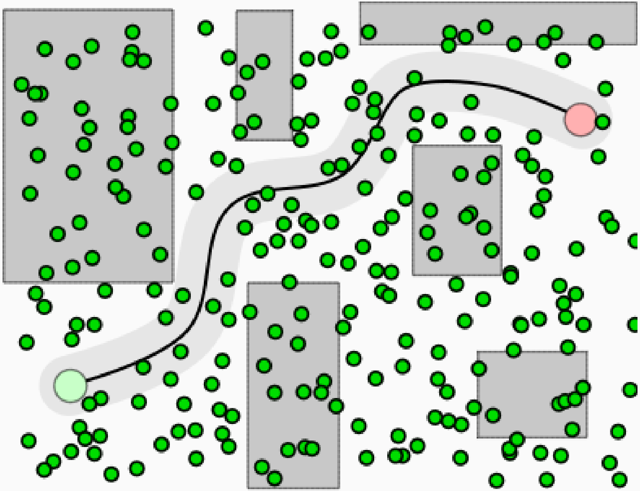
Assumption C: Sampling is dense.

Proof sketch



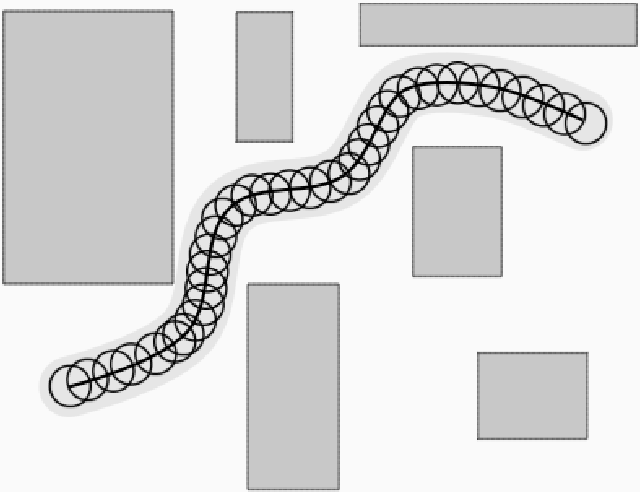
Assumption C: Sampling is dense.

Proof sketch



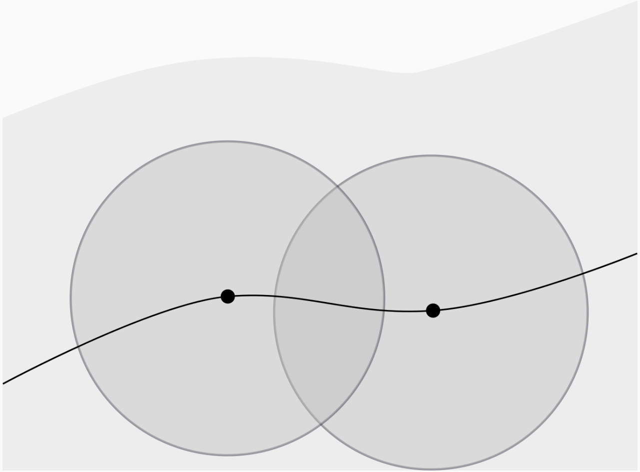
Assumption C: Sampling is dense.

Proof sketch



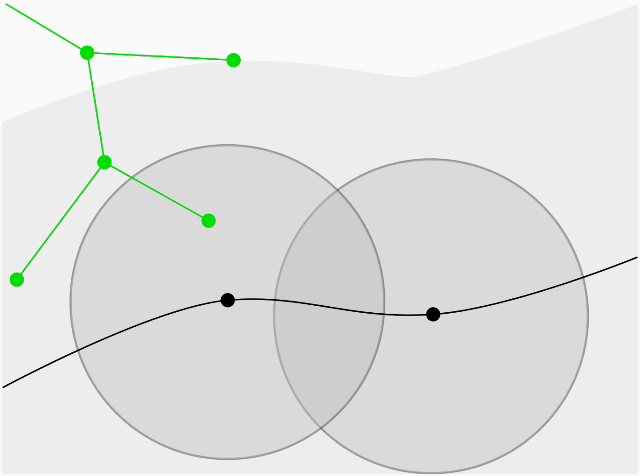
Step 1: Cover feasible path with δ-spaced discs.

Proof sketch



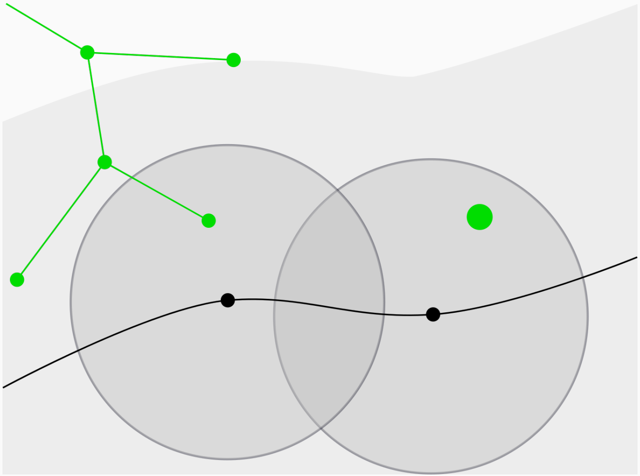
Step 2: Induction step (Base case is trivial)

Proof sketch



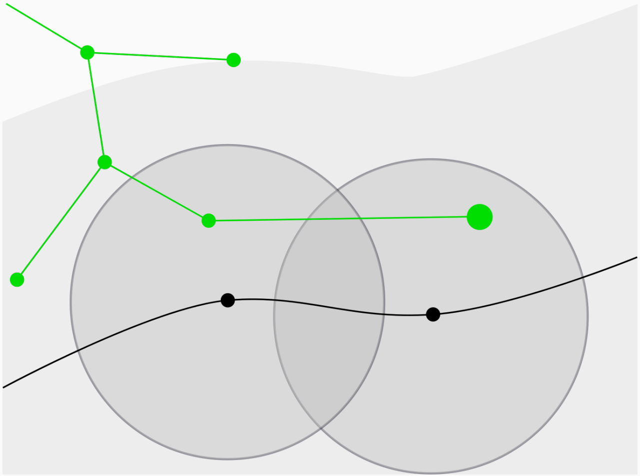
Step 2a: Assume we reached the n-th ball (Induction Assumption).
Need to prove that we reach (n+1)-th ball.

Proof sketch



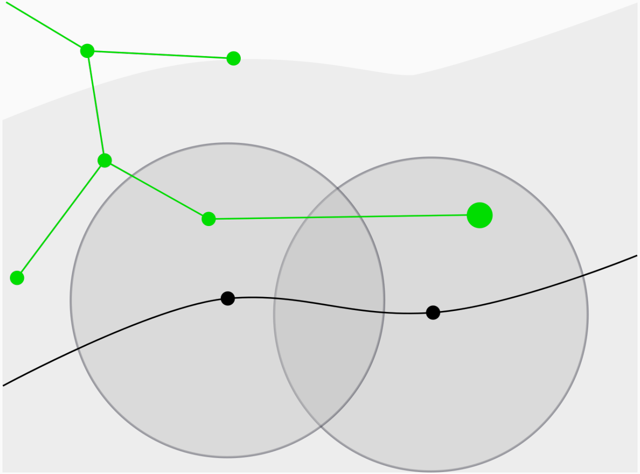
Step 2b: Sample in (n+1)-th ball

Proof sketch



Step 2c: There exists a valid connection in free space

Proof sketch



This shows that you can construct a δ-similar path

Proof sketch

Summary

- Assumption A: There is a feasible path
- Assumption B: It has ϵ clearance
- Assumption C: Sampling is dense

Proof sketch

- Put δ-spaced balls onto feasible path (depending on ϵ)
- Execute induction proof
  - Proof that the first ball is reached (trivial)
  - Proof that you reach ball B_{k+1} from B_k (main part)

Proof sketch

Question

What if we replace "feasible path" with "optimal path". Does the proof still hold?

Proof sketch

Note

- There is no guarantee that you make a connection from B_k to B_{k+1} (there might be a different nearest neighbor)

This is why this is not an optimality proof!

Proof sketch

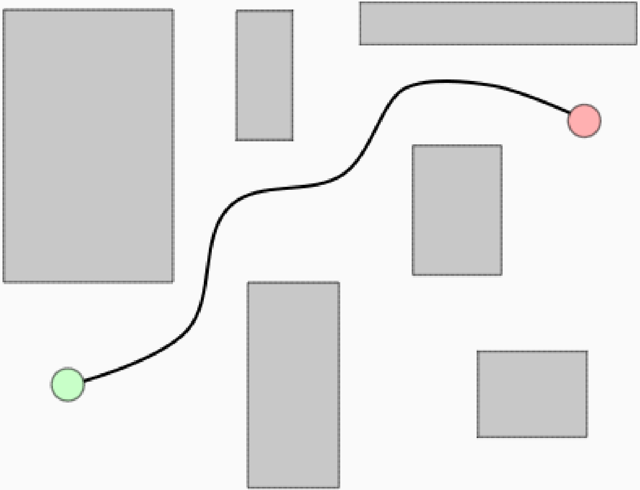
Question

How do we fix this proof for optimality?

Optimal tree-based motion planning

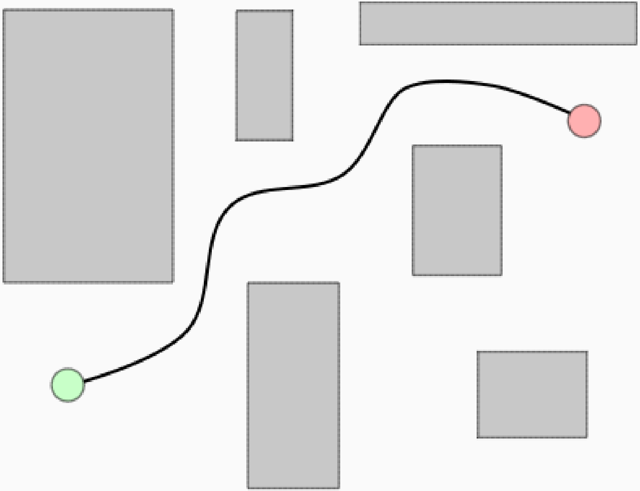
Asymptotically optimal proof RRT*

Proof sketch



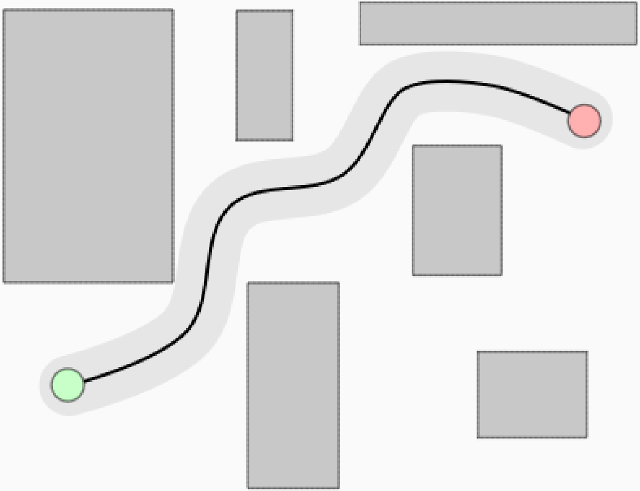
S Karaman and E Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning", 2011 [2]

Proof sketch



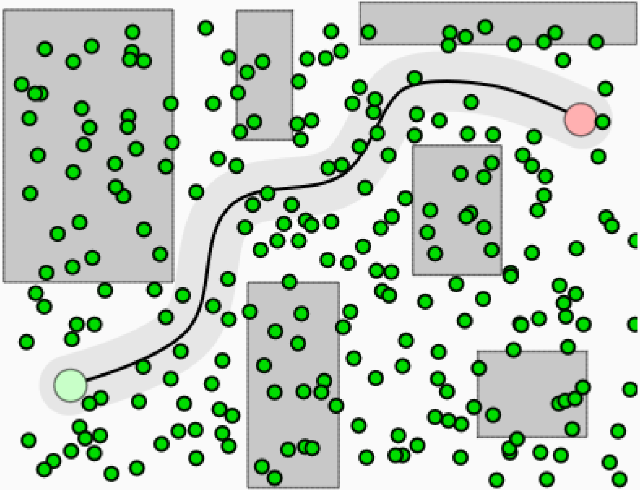
Assumption A: There exists an *optimal* path.

Proof sketch



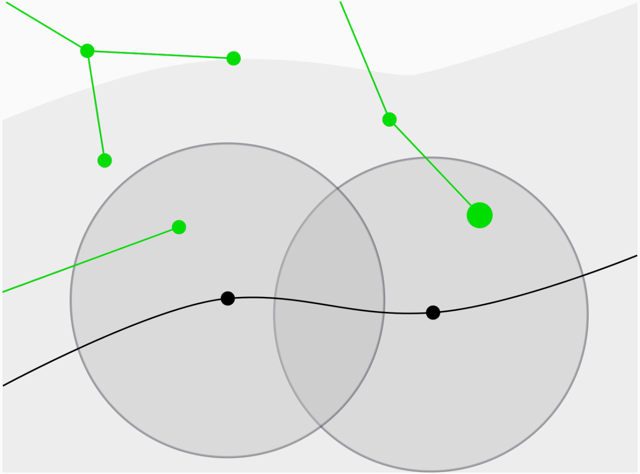
Assumption B: Optimal path has ϵ clearance.

Proof sketch



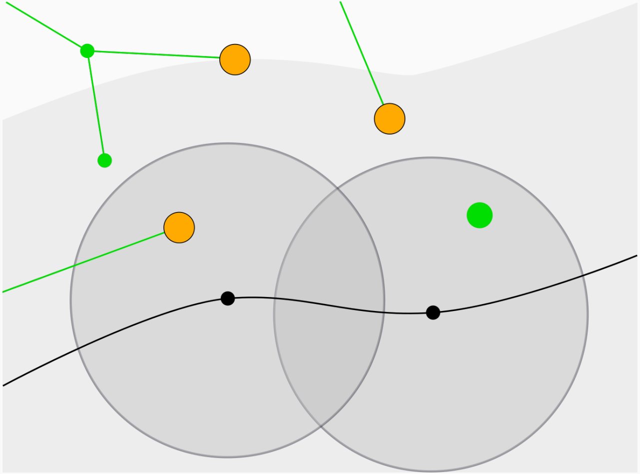
Assumption C: Sampling is dense.

Proof sketch



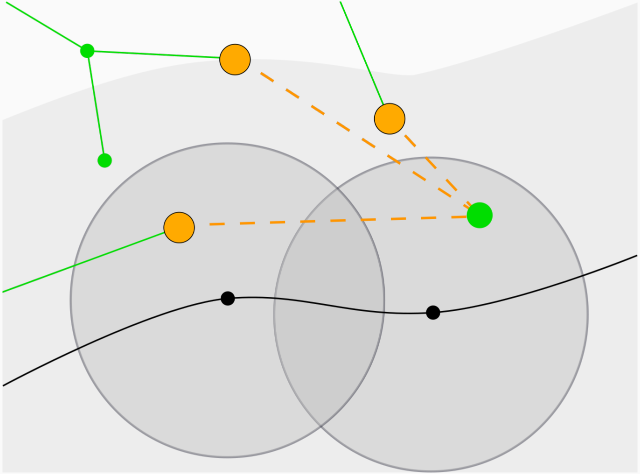
RRT might find wrong wiring.

Proof sketch



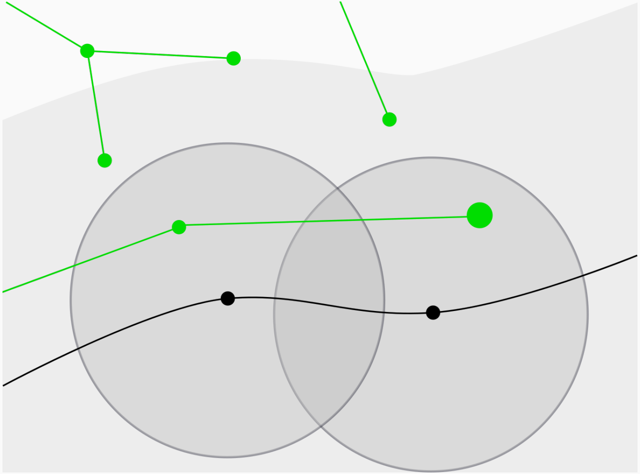
RRT* considers neighbors.

Proof sketch



RRT* computes cost to come.

Proof sketch



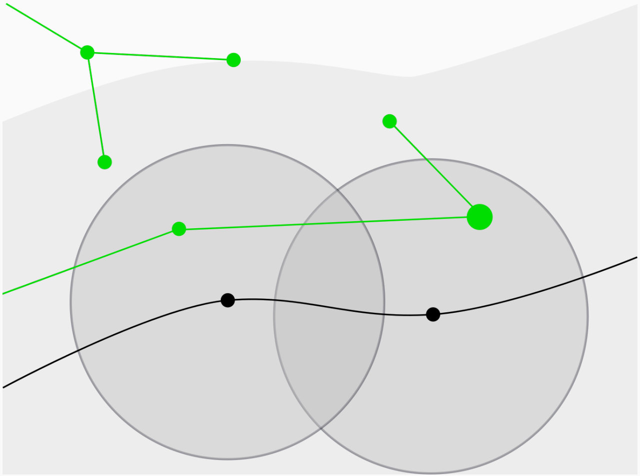
RRT* rewires accordingly.

Proof sketch

Proof idea

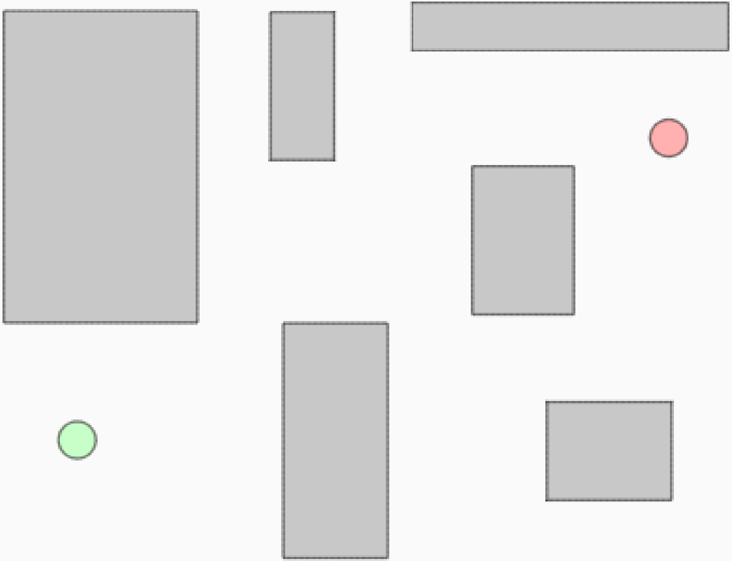
Use rewiring operation to show that we reach B_{k+1} always from B_k.

Proof sketch



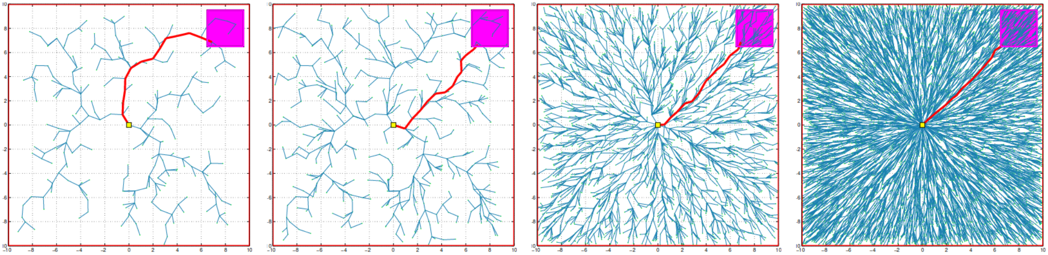
Question: Do we need the second rewiring step?

Visualization



Informed optimal planning

RRT* example



Problems with RRT*

Two problems with RRT*

- Does not prioritize paths as A* does
- Once path is found, it still samples region which cannot improve solution

Informed sampling

- Informed sampling restricts sampling to region which can improve solution
- Based upon concept of Omniscient set

Informed sampling

Reminder (see Lecture 3)

- Optimal cost-to-come $g(x)$ (minimal cost from start to x)
- Optimal cost-to-go $h(x)$ (minimal cost from x to goal)
- Optimal f-value $f(x) = g(x) + h(x)$ (minimal cost, constrained to go through x)

Definition omniscient set

Let c be the cost of a our current solution. Definition omniscient set:

$$X = \{x \in \mathcal{Q} \mid f(x) < c\}$$

Question

What does the omniscient set represent?

Informed sampling

Definition informed set

Let c be the cost of a our current solution. Definition admissible informed set:

$$\hat{X} = \{x \in \mathcal{Q} \mid \hat{f}(x) < c\}$$

whereby $\hat{f} = g(x) + \hat{h}(x)$ with $\hat{h}(x)$ being an admissible heuristic.

Informed sampling

Definition L2-informed set

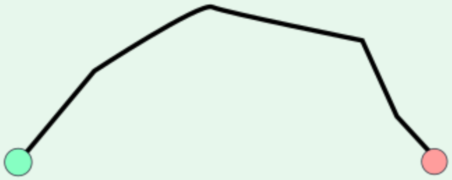
Let c be the cost of a our current solution. Definition admissible informed set:

$$\hat{X} = \{x \in \mathcal{Q} \mid d(xstart, x) + d(x, xgoal) < c\}$$

For the L2-metric, this is called a **prolate hyperspheroid**

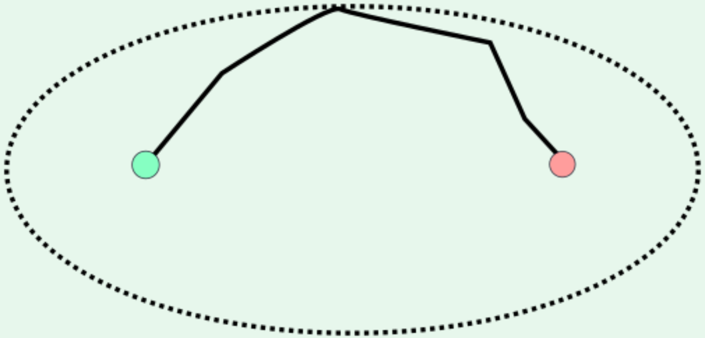
Informed sampling

Informed Set



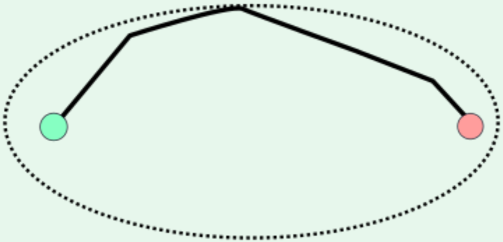
Informed sampling

Informed Set



Informed sampling

Informed Set



Informed sampling

Informed Set

Informed sampling

- Informed RRT* uses Informed Sets to sample more efficiently
- BIT* uses a growing informed set to be more efficient in the beginning

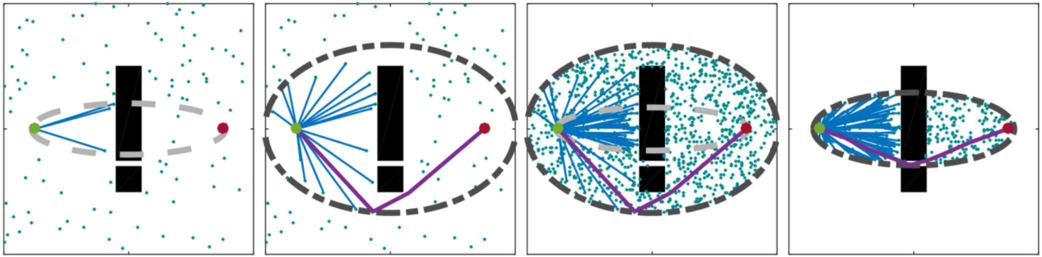
---

JD Gammell et al., "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic", (2014)

JD Gammell et al. "Batch Informed Trees (BIT*): Informed asymptotically optimal anytime search", (2020)

Batch Informed Trees (BIT*)

BIT* example



Drawbacks

Drawbacks of BIT*

- Only works for shortest path cost
- Only works in euclidean spaces

Conclusion

- Asymptotic optimal planning
- Tree-based (RRT, RRT*)

Next time

- Tree-based motion planning for kindynamic systems
- AO-RRT: Asymptotic optimality using cost extension
- SST*: Asymptotic optimality using forward propagation

References i

[1] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. 1984.

[2] Sertac Karaman and Emilio Frazzoli. "Sampling-Based Algorithms for Optimal Motion Planning". In: *International Journal of Robotics Research (IJRR)* 30.7 (2011), pp. 846–894. DOI: 10.1177/0278364911406761.

[3] Kiril Solovey, Lucas Janson, Edward Schmerling, Emilio Frazzoli, and Marco Pavone. "Revisiting the Asymptotic Optimality of RRT*". In: *IEEE International Conference on Robotics and Automation (ICRA)*. May 2020, pp. 2189–2195. DOI: 10.1109/ICRA40945.2020.9196553.

[4] Dan Halperin. *Algorithmic Robotics and Motion Planning*. 2020. URL: http://acg.cs.tau.ac.il/courses/algorithmic-robotics/fall-2019-2020/algorithmic-robotics-and-motion-planning.

[5] Steven M. LaValle. *Planning algorithms*. Cambridge University Press, 2006. ISBN: 978-0-521-86205-9. URL: http://planning.cs.uiuc.edu.