Rapidly-exploring random trees (RRTs)

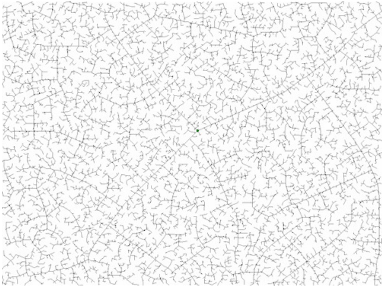
Oren Salzman

Search Base Planning Laboratory, RI

Today's lecture

Rapidly-exploring Random Trees (RRTs)—sampling-based single-query motion planning

- RRT—algorithmic description, implementation details, theoretical properties
- RRT-connect
- High-quality motion planning—RRT*, LBT-RRT



Animation by Javed Hossain, adapted from https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree

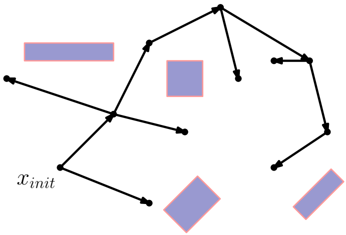
Rapidly-exploring Random Tree—RRT [LaValle, Kuffner01]

- RRTs have been the proven to be an effective, conceptually simple algorithm for single-query planning in high-dimensional C-spaces
- Variants of the basic algorithm have been used for
  - Robotic applications—mobile robotics, manipulation, Mars rovers, humanoid etc.
  - Biological application—drug design
  - Manufacturing and virtual prototyping (assembly analysis)
  - ...
- Variants include
  - High-quality planning
  - Planning for non-holonomic systems
  - Planning on low-dimensional manifolds
  - Parallel RRTs
  - ...

RRT—algorithmic description

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal region $\mathcal{X}_{\text{goal}}$; no. of iterations n; steering param η
Output: Tree $\mathcal{T}$ rooted at x_{start}

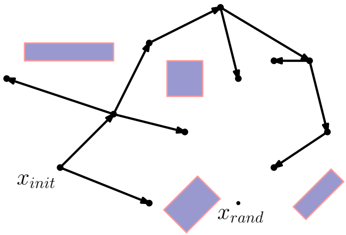
```
T.init(x_start)
for i = 1 to n do
    x_rand ← sample_random_state(X)
    x_near ← nearest_neighbor(T, x_rand)
    x_new ← extend(x_rand, x_near, η)
    if collision_free(x_near, x_new) then
        T.add_vertex(x_new)
        T.add_edge(x_near, x_new)
return T
```



RRT—algorithmic description

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal region $\mathcal{X}_{\text{goal}}$; no. of iterations n; steering param η
Output: Tree $\mathcal{T}$ rooted at x_{start}

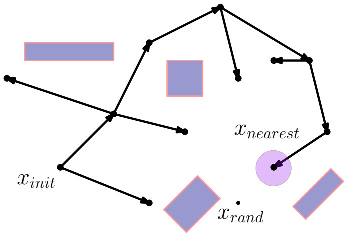
$\mathcal{T}$.`init`(x_{start})
for $i = 1$ to n **do**
  $x_{\text{rand}} \leftarrow$ `sample_random_state`($\mathcal{X}$)
  $x_{\text{near}} \leftarrow$ `nearest_neighbor`($\mathcal{T}, x_{\text{rand}}$)
  $x_{\text{new}} \leftarrow$ `extend`($x_{\text{rand}}, x_{\text{near}}, \eta$)
  if `collision_free`($x_{\text{near}}, x_{\text{new}}$) **then**
    $\mathcal{T}$.`add_vertex`(x_{new})
    $\mathcal{T}$.`add_edge`($x_{\text{near}}, x_{\text{new}}$)
return $\mathcal{T}$



RRT—algorithmic description

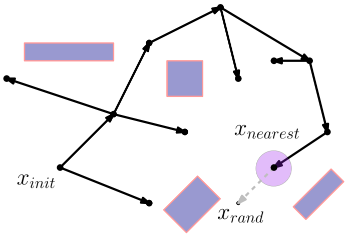
Input: The C-space $\mathcal{X}$; start configuration x_{start} goal region $\mathcal{X}_{\text{goal}}$; no. of iterations n; steering param η
Output: Tree $\mathcal{T}$ rooted at x_{start}

```
T.init(x_start)
for i = 1 to n do
    x_rand ← sample_random_state(X)
    x_near ← nearest_neighbor(T, x_rand)
    x_new ← extend(x_rand, x_near, η)
    if collision_free(x_near, x_new) then
        T.add_vertex(x_new)
        T.add_edge(x_near, x_new)
return T
```



Input: The C-space $\mathcal{X}$; start configuration x_{start} goal region $\mathcal{X}_{\text{goal}}$; no. of iterations n; steering param η
Output: Tree $\mathcal{T}$ rooted at x_{start}

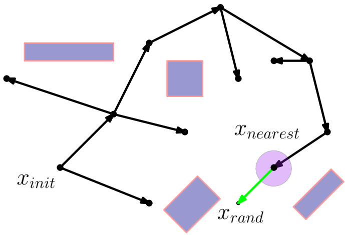
1: $\mathcal{T}$.init(x_{start})
2: **for** $i = 1$ to n **do**
3: $x_{\text{rand}} \leftarrow$ sample_random_state($\mathcal{X}$)
4: $x_{\text{near}} \leftarrow$ nearest_neighbor($\mathcal{T}, x_{\text{rand}}$)
5: $x_{\text{new}} \leftarrow$ extend($x_{\text{rand}}, x_{\text{near}}, \eta$)
6: **if** collision_free($x_{\text{near}}, x_{\text{new}}$) **then**
7: $\mathcal{T}$.add_vertex(x_{new})
8: $\mathcal{T}$.add_edge($x_{\text{near}}, x_{\text{new}}$)
9: **return** $\mathcal{T}$



RRT—algorithmic description

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal region $\mathcal{X}_{\text{goal}}$; no. of iterations n; steering param η
Output: Tree $\mathcal{T}$ rooted at x_{start}

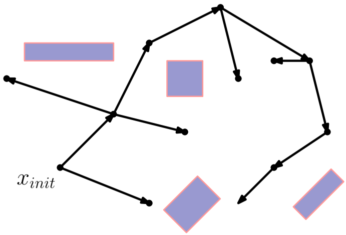
$\mathcal{T}$.`init`(x_{start})
for $i = 1$ to n **do**
    $x_{\text{rand}} \leftarrow$ `sample_random_state`($\mathcal{X}$)
    $x_{\text{near}} \leftarrow$ `nearest_neighbor`($\mathcal{T}, x_{\text{rand}}$)
    $x_{\text{new}} \leftarrow$ `extend`($x_{\text{rand}}, x_{\text{near}}, \eta$)
    if `collision_free`($x_{\text{near}}, x_{\text{new}}$) **then**
        $\mathcal{T}$.`add_vertex`(x_{new})
        $\mathcal{T}$.`add_edge`($x_{\text{near}}, x_{\text{new}}$)
return $\mathcal{T}$



RRT—algorithmic description

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal region $\mathcal{X}_{\text{goal}}$; no. of iterations n; steering param η
Output: Tree $\mathcal{T}$ rooted at x_{start}

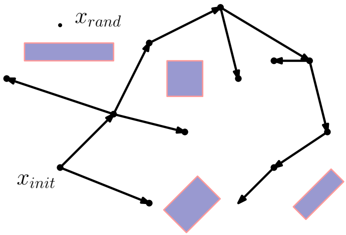
```
T.init(x_start)
for i = 1 to n do
    x_rand ← sample_random_state(X)
    x_near ← nearest_neighbor(T, x_rand)
    x_new ← extend(x_rand, x_near, η)
    if collision_free(x_near, x_new) then
        T.add_vertex(x_new)
        T.add_edge(x_near, x_new)
return T
```



RRT—algorithmic description

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal region $\mathcal{X}_{\text{goal}}$; no. of iterations n; steering param η
Output: Tree $\mathcal{T}$ rooted at x_{start}

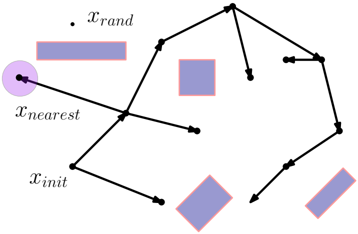
```
1: T.init(x_start)
2: for i = 1 to n do
3:     x_rand ← sample_random_state(X)
4:     x_near ← nearest_neighbor(T, x_rand)
5:     x_new ← extend(x_rand, x_near, η)
6:     if collision_free(x_near, x_new) then
7:         T.add_vertex(x_new)
8:         T.add_edge(x_near, x_new)
9: return T
```



RRT—algorithmic description

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal region $\mathcal{X}_{\text{goal}}$; no. of iterations n; steering param η
Output: Tree $\mathcal{T}$ rooted at x_{start}

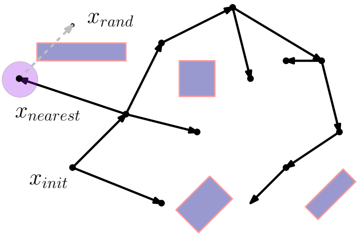
$\mathcal{T}$.`init`(x_{start})
for $i = 1$ to n **do**
  $x_{\text{rand}} \leftarrow$ `sample_random_state`($\mathcal{X}$)
  $x_{\text{near}} \leftarrow$ `nearest_neighbor`($\mathcal{T}, x_{\text{rand}}$)
  $x_{\text{new}} \leftarrow$ `extend`($x_{\text{rand}}, x_{\text{near}}, \eta$)
  if `collision_free`($x_{\text{near}}, x_{\text{new}}$) **then**
    $\mathcal{T}$.`add_vertex`(x_{new})
    $\mathcal{T}$.`add_edge`($x_{\text{near}}, x_{\text{new}}$)
return $\mathcal{T}$



RRT—algorithmic description

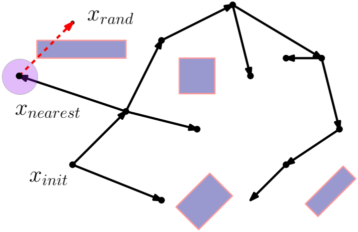
Input: The C-space $\mathcal{X}$; start configuration x_{start} goal region $\mathcal{X}_{\text{goal}}$; no. of iterations n; steering param η
Output: Tree $\mathcal{T}$ rooted at x_{start}

1: $\mathcal{T}$.init(x_{start})
2: **for** $i = 1$ to n **do**
3: $\quad x_{\text{rand}} \leftarrow$ sample_random_state($\mathcal{X}$)
4: $\quad x_{\text{near}} \leftarrow$ nearest_neighbor($\mathcal{T}, x_{\text{rand}}$)
5: $\quad x_{\text{new}} \leftarrow$ extend($x_{\text{rand}}, x_{\text{near}}, \eta$)
6: $\quad$ **if** collision_free($x_{\text{near}}, x_{\text{new}}$) **then**
7: $\quad\quad \mathcal{T}$.add_vertex(x_{new})
8: $\quad\quad \mathcal{T}$.add_edge($x_{\text{near}}, x_{\text{new}}$)
9: **return** $\mathcal{T}$



Input: The C-space $\mathcal{X}$; start configuration x_{start} goal region $\mathcal{X}_{\text{goal}}$; no. of iterations n; steering param η
Output: Tree $\mathcal{T}$ rooted at x_{start}

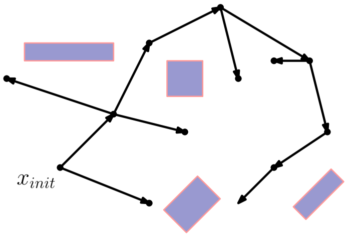
```
T.init(x_start)
for i = 1 to n do
    x_rand ← sample_random_state(X)
    x_near ← nearest_neighbor(T, x_rand)
    x_new ← extend(x_rand, x_near, η)
    if collision_free(x_near, x_new) then
        T.add_vertex(x_new)
        T.add_edge(x_near, x_new)
return T
```



RRT—algorithmic description

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal region $\mathcal{X}_{\text{goal}}$; no. of iterations n; steering param η
Output: Tree $\mathcal{T}$ rooted at x_{start}

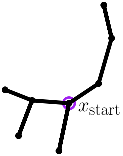
```
T.init(x_start)
for i = 1 to n do
    x_rand ← sample_random_state(X)
    x_near ← nearest_neighbor(T, x_rand)
    x_new ← extend(x_rand, x_near, η)
    if collision_free(x_near, x_new) then
        T.add_vertex(x_new)
        T.add_edge(x_near, x_new)
return T
```



RRT—algorithmic description(extend)

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal region $\mathcal{X}_{\text{goal}}$; no. of iterations n; steering param η
Output: Tree $\mathcal{T}$ rooted at x_{start}

$\mathcal{T}$.init(x_{start})
for $i = 1$ to n **do**
  $x_{\text{rand}} \leftarrow$ sample_random_state($\mathcal{X}$)
  $x_{\text{near}} \leftarrow$ nearest_neighbor($\mathcal{T}, x_{\text{rand}}$)
  $x_{\text{new}} \leftarrow$ extend($x_{\text{rand}}, x_{\text{near}}, \eta$)
  if collision_free($x_{\text{near}}, x_{\text{new}}$) **then**
    $\mathcal{T}$.add_vertex(x_{new})
    $\mathcal{T}$.add_edge($x_{\text{near}}, x_{\text{new}}$)
return $\mathcal{T}$

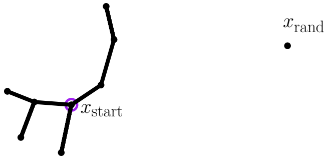


Figures adapted from https://www.cs.cmu.edu/~motionplanning/lecture/Chap7-Prob-Planning_howie.pdf

RRT—algorithmic description(extend)

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal region $\mathcal{X}_{\text{goal}}$; no. of iterations n; steering param η
Output: Tree $\mathcal{T}$ rooted at x_{start}

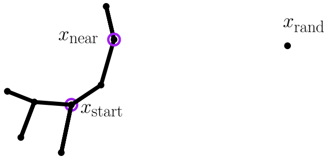
```
T.init(x_start)
for i = 1 to n do
    x_rand ← sample_random_state(X)
    x_near ← nearest_neighbor(T, x_rand)
    x_new ← extend(x_rand, x_near, η)
    if collision_free(x_near, x_new) then
        T.add_vertex(x_new)
        T.add_edge(x_near, x_new)
return T
```



Figures adapted from https://www.cs.cmu.edu/~motionplanning/lecture/Chap7-Prob-Planning_howie.pdf

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal region $\mathcal{X}_{\text{goal}}$; no. of iterations n; steering param η
Output: Tree $\mathcal{T}$ rooted at x_{start}

```
1: T.init(x_start)
2: for i = 1 to n do
3:     x_rand ← sample_random_state(X)
4:     x_near ← nearest_neighbor(T, x_rand)
5:     x_new ← extend(x_rand, x_near, η)
6:     if collision_free(x_near, x_new) then
7:         T.add_vertex(x_new)
8:         T.add_edge(x_near, x_new)
9: return T
```

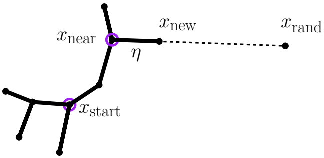


Figures adapted from https://www.cs.cmu.edu/~motionplanning/lecture/Chap7-Prob-Planning_howie.pdf

RRT—algorithmic description(`extend`)

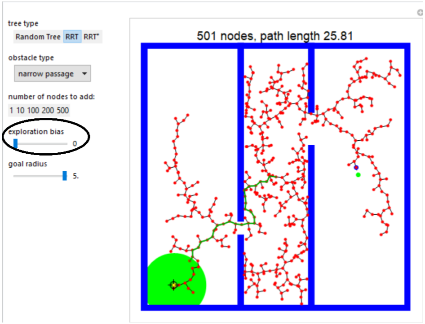
Input: The C-space $\mathcal{X}$; start configuration x_{start} goal region $\mathcal{X}_{\text{goal}}$; no. of iterations n; steering param η
Output: Tree $\mathcal{T}$ rooted at x_{start}

$\mathcal{T}$.init(x_{start})
for $i = 1$ to n **do**
    $x_{\text{rand}} \leftarrow$ sample_random_state($\mathcal{X}$)
    $x_{\text{near}} \leftarrow$ nearest_neighbor($\mathcal{T}, x_{\text{rand}}$)
    $x_{\text{new}} \leftarrow$ extend($x_{\text{rand}}, x_{\text{near}}, \eta$)
    if collision_free($x_{\text{near}}, x_{\text{new}}$) **then**
        $\mathcal{T}$.add_vertex(x_{new})
        $\mathcal{T}$.add_edge($x_{\text{near}}, x_{\text{new}}$)
return $\mathcal{T}$



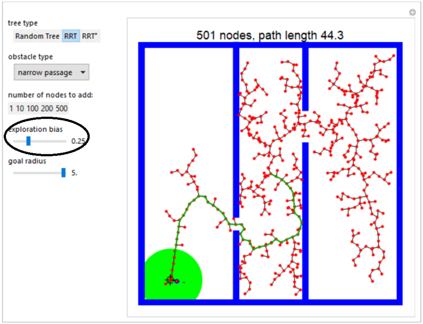
Figures adapted from https://www.cs.cmu.edu/~motionplanning/lecture/Chap7-Prob-Planning_howie.pdf

- Goal biasing
  - Sample x_{rand} uniformly from $\mathcal{X}$ with prob. $1 - p_{\text{bias}}$ and uniformly from $\mathcal{X}_{\text{goal}}$ with prob. p_{bias}
  - Rule of thumb—use $p_{\text{bias}} = 0.05$
- Step size η—what if it is too big? too small?
- Metric—typical C-spaces are non-Eucl. How can we compute NN?



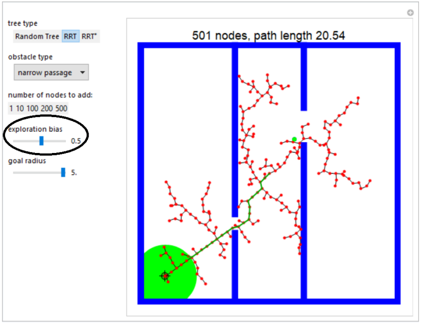
Figures constructed using http://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/

- Goal biasing
  - Sample x_{rand} uniformly from $\mathcal{X}$ with prob. $1 - p_{\text{bias}}$ and uniformly from $\mathcal{X}_{\text{goal}}$ with prob. p_{bias}
  - Rule of thumb—use $p_{\text{bias}} = 0.05$
- Step size η—what if it is too big? too small?
- Metric—typical C-spaces are non-Eucl. How can we compute NN?



Figures constructed using http://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/

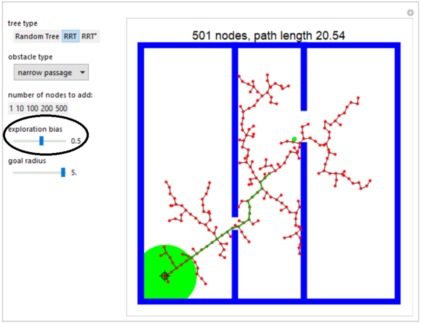
- Goal biasing
  - Sample x_{rand} uniformly from $\mathcal{X}$ with prob. $1 - p_{\text{bias}}$ and uniformly from $\mathcal{X}_{\text{goal}}$ with prob. p_{bias}
  - Rule of thumb—use $p_{\text{bias}} = 0.05$
- Step size η—what if it is too big? too small?
- Metric—typical C-spaces are non-Eucl. How can we compute NN?



Figures constructed using http://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/

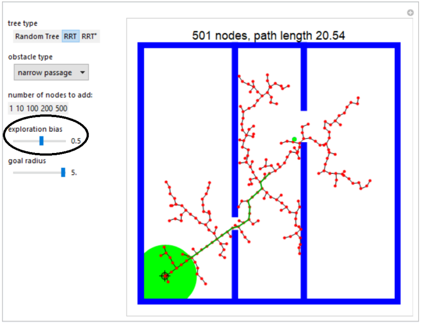
RRT—implementation details

- Goal biasing
  - Sample x_{rand} uniformly from $\mathcal{X}$ with prob. $1 - p_{\text{bias}}$ and uniformly from $\mathcal{X}_{\text{goal}}$ with prob. p_{bias}
  - Rule of thumb—use $p_{\text{bias}} = 0.05$
- Step size η—what if it is too big? too small?
- Metric—typical C-spaces are non-Eucl. How can we compute NN?



Figures constructed using http://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/

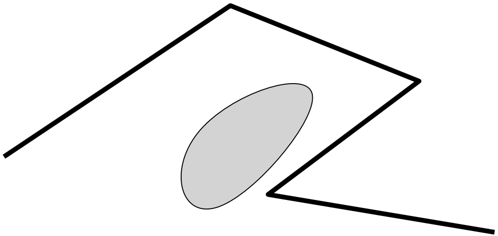
- Goal biasing
  - Sample x_{rand} uniformly from $\mathcal{X}$ with prob. $1 - p_{\text{bias}}$ and uniformly from $\mathcal{X}_{\text{goal}}$ with prob. p_{bias}
  - Rule of thumb—use $p_{\text{bias}} = 0.05$
- Step size η—what if it is too big? too small?
- Metric—typical C-spaces are non-Eucl. How can we compute NN?



Figures constructed using http://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/

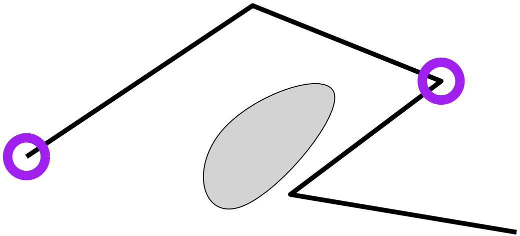
RRT—implementation details (cont.)

- Smoothing / post-processing—Paths produced by RRT are long and have unnecessary turns. What can we do?



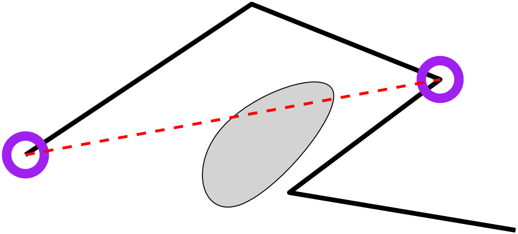
RRT—implementation details (cont.)

- Smoothing / post-processing—Paths produced by RRT are long and have unnecessary turns. What can we do?

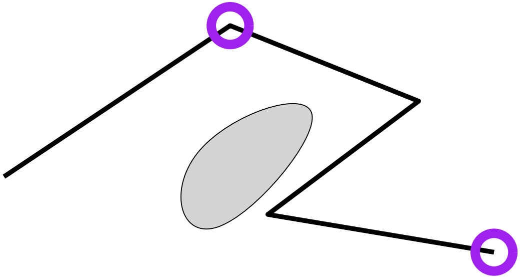


RRT—implementation details (cont.)

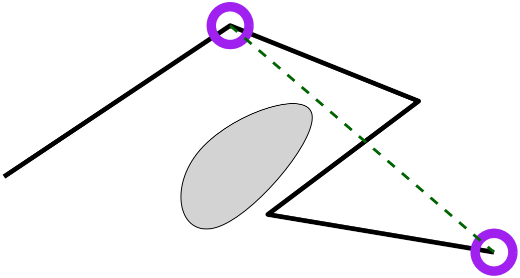
- Smoothing / post-processing—Paths produced by RRT are long and have unnecessary turns. What can we do?



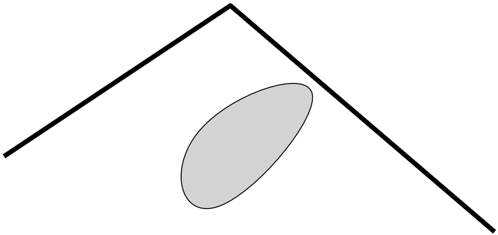
- Smoothing / post-processing—Paths produced by RRT are long and have unnecessary turns. What can we do?



- Smoothing / post-processing—Paths produced by RRT are long and have unnecessary turns. What can we do?



- Smoothing / post-processing—Paths produced by RRT are long and have unnecessary turns. What can we do?



- Smoothing / post-processing—Paths produced by RRT are long and have unnecessary turns. What can we do?

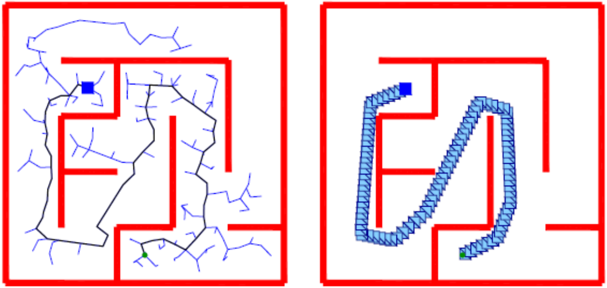


Figure adapted from [Kuffner, LaValle00]

- Smoothing / post-processing—Paths produced by RRT are long and have unnecessary turns. What can we do?

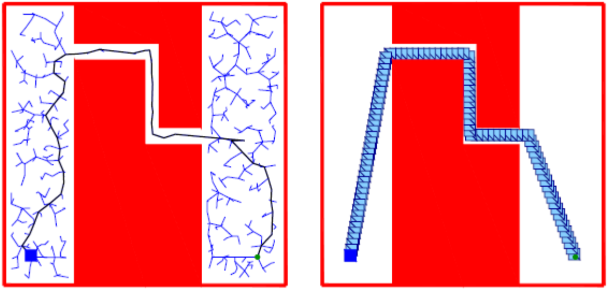


Figure adapted from [Kuffner, LaValle00]

- Smoothing / post-processing—Paths produced by RRT are long and have unnecessary turns. What can we do?

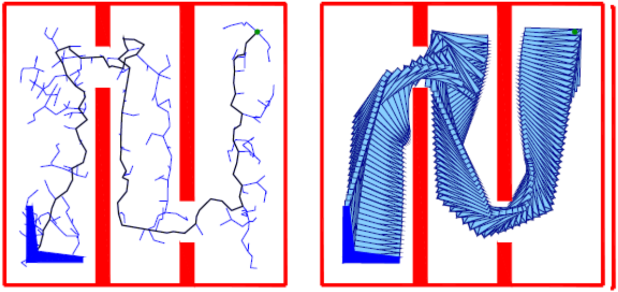


Figure adapted from [Kuffner, LaValle00]

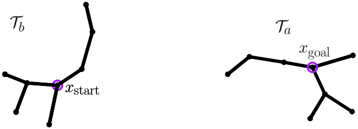
- It is often beneficial to search both from the start and from the goal
- This has been used in graph-search (e.g., bidirectional Dijkstra)

Animation adapted from https://meyavuz.wordpress.com/2017/05/14/dijkstra-vs-bi-directional-dijkstra-comparison-on-sample-us-road-network/

RRT-connect—algorithmic description

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal configuration x_{goal}; no. of iterations n; steering param η
Output: Path connecting x_{start} to x_{goal}

$\mathcal{T}_a.\texttt{init}(x_{\text{start}})$, $\mathcal{T}_b.\texttt{init}(x_{\text{goal}})$
for $i = 1$ to n **do**
  $x_{\text{rand}} \leftarrow \texttt{sample\_random\_state}(\mathcal{X})$
  $x_{\text{near}} \leftarrow \texttt{nearest\_neighbor}(\mathcal{T}_a, x_{\text{rand}})$
  $x_{\text{new}} \leftarrow \texttt{extend}(x_{\text{rand}}, x_{\text{near}}, \eta)$
  if $\texttt{collision\_free}(x_{\text{near}}, x_{\text{new}})$ **then**
    $\mathcal{T}_a.\texttt{add\_vertex}(x_{\text{new}})$
    $\mathcal{T}_a.\texttt{add\_edge}(x_{\text{near}}, x_{\text{new}})$
    if $\texttt{connect}(\mathcal{T}_b, x_{\text{new}})$ **then**
      return $\texttt{path}(\mathcal{T}_a, \mathcal{T}_b)$
  $\texttt{swap}(\mathcal{T}_a, \mathcal{T}_b)$
return failure

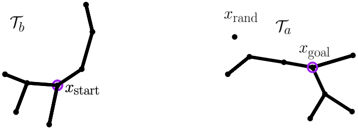


Figures adapted from https://www.cs.cmu.edu/~motionplanning/lecture/Chap7-Prob-Planning_howie.pdf

RRT-connect—algorithmic description

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal configuration x_{goal}; no. of iterations n; steering param η
Output: Path connecting x_{start} to x_{goal}

$\mathcal{T}_a$.init(x_{start}), $\mathcal{T}_b$.init(x_{goal})
for $i = 1$ to n **do**
  $x_{\text{rand}} \leftarrow$ sample_random_state($\mathcal{X}$)
  $x_{\text{near}} \leftarrow$ nearest_neighbor($\mathcal{T}_a, x_{\text{rand}}$)
  $x_{\text{new}} \leftarrow$ extend($x_{\text{rand}}, x_{\text{near}}, \eta$)
  if collision_free($x_{\text{near}}, x_{\text{new}}$) **then**
    $\mathcal{T}_a$.add_vertex(x_{new})
    $\mathcal{T}_a$.add_edge($x_{\text{near}}, x_{\text{new}}$)
    if connect ($\mathcal{T}_b, x_{\text{new}}$) **then**
      return path($\mathcal{T}_a, \mathcal{T}_b$)
  swap($\mathcal{T}_a, \mathcal{T}_b$)
return failure

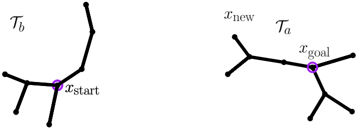


Figures adapted from https://www.cs.cmu.edu/~motionplanning/lecture/Chap7-Prob-Planning_howie.pdf

RRT-connect—algorithmic description

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal configuration x_{goal}; no. of iterations n; steering param η
Output: Path connecting x_{start} to x_{goal}

```
Ta.init(x_start),    Tb.init(x_goal)
for i = 1 to n do
    x_rand ← sample_random_state(X)
    x_near ← nearest_neighbor(Ta, x_rand)
    x_new ← extend(x_rand, x_near, η)
    if collision_free(x_near, x_new) then
        Ta.add_vertex(x_new)
        Ta.add_edge(x_near, x_new)
        if connect (Tb, x_new) then
            return path(Ta, Tb)
    swap(Ta, Tb)
return failure
```

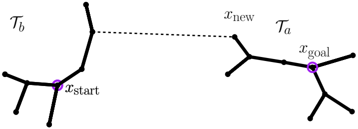


Figures adapted from https://www.cs.cmu.edu/~motionplanning/lecture/Chap7-Prob-Planning_howie.pdf

RRT-connect—algorithmic description

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal configuration x_{goal}; no. of iterations n; steering param η
Output: Path connecting x_{start} to x_{goal}

$\mathcal{T}_a$.init(x_{start}), $\mathcal{T}_b$.init(x_{goal})
for $i = 1$ to n **do**
  $x_{\text{rand}} \leftarrow$ sample_random_state($\mathcal{X}$)
  $x_{\text{near}} \leftarrow$ nearest_neighbor($\mathcal{T}_a, x_{\text{rand}}$)
  $x_{\text{new}} \leftarrow$ extend($x_{\text{rand}}, x_{\text{near}}, \eta$)
  if collision_free($x_{\text{near}}, x_{\text{new}}$) **then**
    $\mathcal{T}_a$.add_vertex(x_{new})
    $\mathcal{T}_a$.add_edge($x_{\text{near}}, x_{\text{new}}$)
    if connect ($\mathcal{T}_b, x_{\text{new}}$) **then**
      return path($\mathcal{T}_a, \mathcal{T}_b$)
  swap($\mathcal{T}_a, \mathcal{T}_b$)
return failure

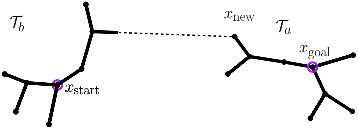


Figures adapted from https://www.cs.cmu.edu/~motionplanning/lecture/Chap7-Prob-Planning_howie.pdf

RRT-connect—algorithmic description

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal configuration x_{goal}; no. of iterations n; steering param η
Output: Path connecting x_{start} to x_{goal}

```
T_a.init(x_start),    T_b.init(x_goal)
for i = 1 to n do
    x_rand ← sample_random_state(X)
    x_near ← nearest_neighbor(T_a, x_rand)
    x_new ← extend(x_rand, x_near, η)
    if collision_free(x_near, x_new) then
        T_a.add_vertex(x_new)
        T_a.add_edge(x_near, x_new)
        if connect (T_b, x_new) then
            return path(T_a, T_b)
    swap(T_a, T_b)
return failure
```

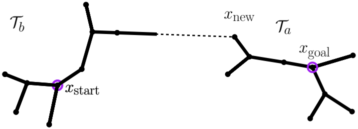


Figures adapted from https://www.cs.cmu.edu/~motionplanning/lecture/Chap7-Prob-Planning_howie.pdf

RRT-connect—algorithmic description

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal configuration x_{goal}; no. of iterations n; steering param η
Output: Path connecting x_{start} to x_{goal}

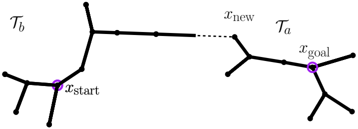
$\mathcal{T}_a$.init(x_{start}), $\mathcal{T}_b$.init(x_{goal})
for $i = 1$ to n **do**
  $x_{\text{rand}} \leftarrow$ sample_random_state($\mathcal{X}$)
  $x_{\text{near}} \leftarrow$ nearest_neighbor($\mathcal{T}_a, x_{\text{rand}}$)
  $x_{\text{new}} \leftarrow$ extend($x_{\text{rand}}, x_{\text{near}}, \eta$)
  if collision_free($x_{\text{near}}, x_{\text{new}}$) **then**
    $\mathcal{T}_a$.add_vertex(x_{new})
    $\mathcal{T}_a$.add_edge($x_{\text{near}}, x_{\text{new}}$)
    if connect ($\mathcal{T}_b, x_{\text{new}}$) **then**
      return path($\mathcal{T}_a, \mathcal{T}_b$)
  swap($\mathcal{T}_a, \mathcal{T}_b$)
return failure



Figures adapted from https://www.cs.cmu.edu/~motionplanning/lecture/Chap7-Prob-Planning_howie.pdf

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal configuration x_{goal}; no. of iterations n; steering param η
Output: Path connecting x_{start} to x_{goal}

$\mathcal{T}_a$.init(x_{start}), $\mathcal{T}_b$.init(x_{goal})
for $i = 1$ to n **do**
    $x_{\text{rand}} \leftarrow$ sample_random_state($\mathcal{X}$)
    $x_{\text{near}} \leftarrow$ nearest_neighbor($\mathcal{T}_a, x_{\text{rand}}$)
    $x_{\text{new}} \leftarrow$ extend($x_{\text{rand}}, x_{\text{near}}, \eta$)
    if collision_free($x_{\text{near}}, x_{\text{new}}$) **then**
        $\mathcal{T}_a$.add_vertex(x_{new})
        $\mathcal{T}_a$.add_edge($x_{\text{near}}, x_{\text{new}}$)
        if connect ($\mathcal{T}_b, x_{\text{new}}$) **then**
            return path($\mathcal{T}_a, \mathcal{T}_b$)
    swap($\mathcal{T}_a, \mathcal{T}_b$)
return failure

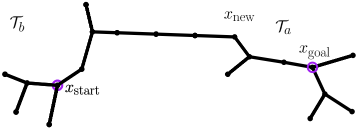


Figures adapted from https://www.cs.cmu.edu/~motionplanning/lecture/Chap7-Prob-Planning_howie.pdf

RRT-connect—algorithmic description

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal configuration x_{goal}; no. of iterations n; steering param η
Output: Path connecting x_{start} to x_{goal}

$\mathcal{T}_a$.init(x_{start}), $\mathcal{T}_b$.init(x_{goal})
for $i = 1$ to n **do**
  $x_{\text{rand}} \leftarrow$ sample_random_state($\mathcal{X}$)
  $x_{\text{near}} \leftarrow$ nearest_neighbor($\mathcal{T}_a, x_{\text{rand}}$)
  $x_{\text{new}} \leftarrow$ extend($x_{\text{rand}}, x_{\text{near}}, \eta$)
  if collision_free($x_{\text{near}}, x_{\text{new}}$) **then**
    $\mathcal{T}_a$.add_vertex(x_{new})
    $\mathcal{T}_a$.add_edge($x_{\text{near}}, x_{\text{new}}$)
    if connect ($\mathcal{T}_b, x_{\text{new}}$) **then**
      return path($\mathcal{T}_a, \mathcal{T}_b$)
  swap($\mathcal{T}_a, \mathcal{T}_b$)
return failure

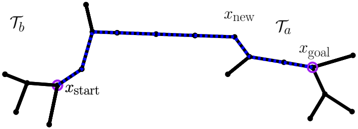


Figures adapted from https://www.cs.cmu.edu/~motionplanning/lecture/Chap7-Prob-Planning_howie.pdf

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal configuration x_{goal}; no. of iterations n; steering param η

Output: Path connecting x_{start} to x_{goal}

$\mathcal{T}_a$.init(x_{start}), $\mathcal{T}_b$.init(x_{goal})
for $i = 1$ to n **do**
  $x_{\mathrm{rand}} \leftarrow$ sample_random_state($\mathcal{X}$)
  $x_{\mathrm{near}} \leftarrow$ nearest_neighbor($\mathcal{T}_a, x_{\mathrm{rand}}$)
  $x_{\mathrm{new}} \leftarrow$ extend($x_{\mathrm{rand}}, x_{\mathrm{near}}, \eta$)
  if collision_free($x_{\mathrm{near}}, x_{\mathrm{new}}$) **then**
    $\mathcal{T}_a$.add_vertex(x_{new})
    $\mathcal{T}_a$.add_edge($x_{\mathrm{near}}, x_{\mathrm{new}}$)
    if connect ($\mathcal{T}_b, x_{\mathrm{new}}$) **then**
      return path($\mathcal{T}_a, \mathcal{T}_b$)
  swap($\mathcal{T}_a, \mathcal{T}_b$)
return failure



Figures adapted from https://www.cs.cmu.edu/~motionplanning/lecture/Chap7-Prob-Planning_howie.pdf

`RRT-connect`—food for thought

- Why do we swap the trees?
- How do we maintain the rapid exploration?
- What is the additional assumption taken and what are the implications?
  - Notice the exact connection made

RRT-connect—food for thought

- Why do we swap the trees?
- How do we maintain the rapid exploration?
- What is the additional assumption taken and what are the implications?
  - Notice the exact connection made

RRT—Theoretical properties

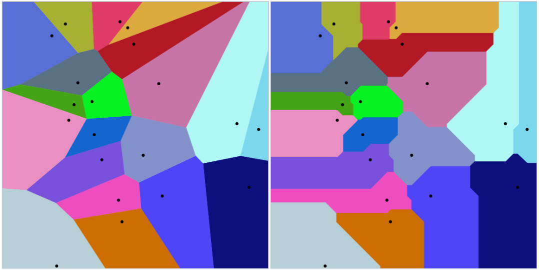
- Rapid exploration (Voronoi bias)
- Prob. completeness
- (low) Quality of solutions

Voronoi Diagrams

Definition

Let $P = \{p_1 \ldots p_n\}$ be a set of n points (sites) in the plane. The Voronoi Diagram of P is the subdivision of the plane into n cells, one for each site, with the property that a point q lies in the cell corresponding to a site p_i if and only if $\text{dist}(q, p_i) < \text{dist}(q, p_j)$ for each $p_j \in P$ with $j \neq i$

- Can be extended to any metric space and any types of sites



Euclidean distance Manhattan distance

Figure adapted from https://en.wikipedia.org/wiki/Voronoi_diagram

Voronoi diagrams and RRTs

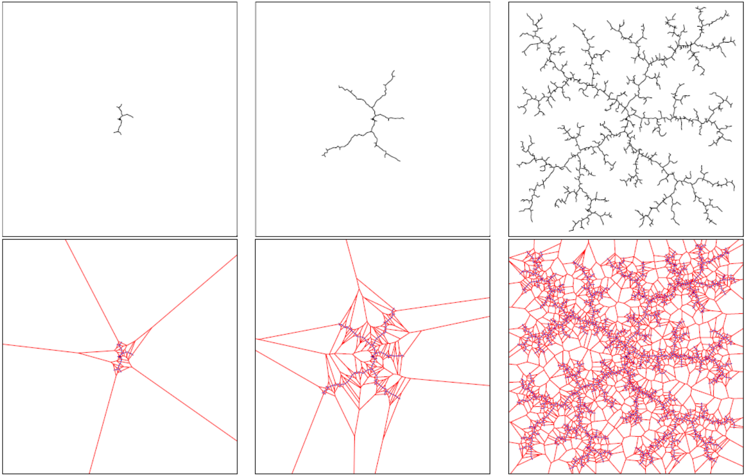


Figure adapted from [Kuffner, LaValle00]

Prob. completeness

- Let $\texttt{ALG}(n)$ be a sampling-based motion-planning algorithm that samples n configurations.
- Let $P_{\text{succ}}\left(x_{\text{start}}, \mathcal{X}, \mathcal{X}_{\text{goal}}\right)$ be the probability that $\texttt{ALG}(n)$ returns a collision-free path from x_{start} to $\mathcal{X}_{\text{goal}}$ in $\mathcal{X}$

Definition

An algorithm $\texttt{ALG}$ is said to be probabilistically complete if

$$\lim_{n \to \infty} P_{\text{succ}}\left(x_{\text{start}}, \mathcal{X}, \mathcal{X}_{\text{goal}}\right) = 1.$$

Prob. completeness

Thm [LaValle Kuffner01, Kleinbort et al.18]

The `RRT` algorithm is probabilistically complete.

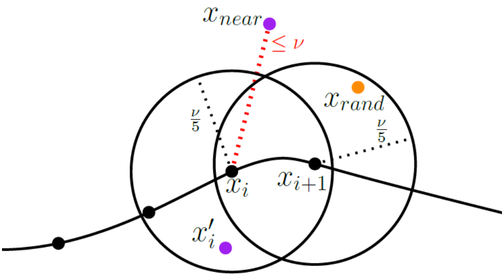


Figure adapted from [Kleinbort et al.18]

The quality of the solutions produced by RRT

- The probability for low-quality paths is bounded away from zero [Nechushtan, Raveh, halperin10]
- This hold regardless if post-processing us applied
- Empirically, for certain scenarios the solution path is over 140 times worse than optimal in 5.9% of independent runs.

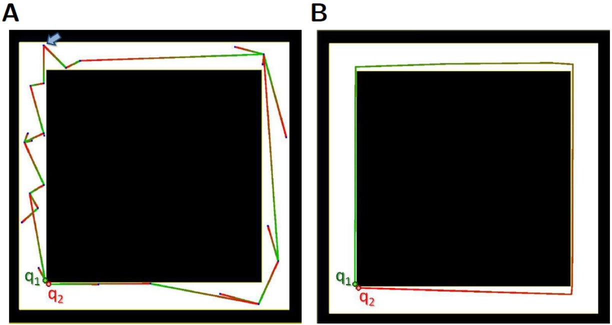


Figure adapted from [Nechushtan, Raveh, halperin10]

RRT * [Karaman Frazzoli11]

Input: The C-space $\mathcal{X}$; start configuration x_{start} goal configuration x_{goal}; no. of iterations n; steering param η

Output: Path connecting x_{start} to x_{goal}

$\mathcal{T}$.init (x_{start})
for $i = 1$ to n **do**
    $x_{\text{rand}} \leftarrow$ sample_random_state($\mathcal{X}$)
    $x_{\text{nearest}} \leftarrow$ nearest_neighbor($\mathcal{T}, x_{\text{rand}}$)
    $x_{\text{new}} \leftarrow$ extend($x_{\text{nearest}}, x_{\text{rand}}$)
    if collision_free($x_{\text{nearest}}, x_{\text{new}}$) **then**
        $\mathcal{T}$.add_vertex(x_{new})
        $\mathcal{T}$.add_edge($x_{\text{nearest}}, x_{\text{new}}$)
        $X_{\text{near}} \leftarrow$ nearest_neighbors($\mathcal{T}, x_{\text{new}}, r_i$)
        for all $(x_{\text{near}}, X_{\text{near}})$ **do**
            rewire_RRT*($x_{\text{near}}, x_{\text{new}}$)
        for all $(x_{\text{near}}, X_{\text{near}})$ **do**
            rewire_RRT*($x_{\text{new}}, x_{\text{near}}$)

RRT *—rewire operations

RRT* locally rewires nodes in $\mathcal{T}$ using a connection radius of

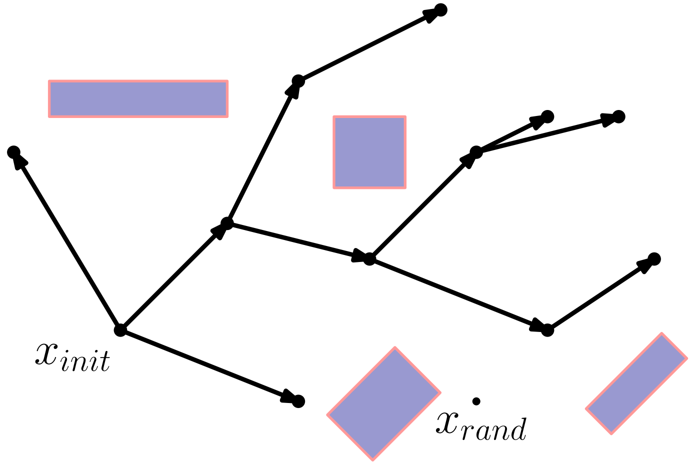
$$r(i) \approx \gamma(d) \cdot \left(\frac{\log i}{i}\right)^{1/d}$$

Input: A new potential parent $x_{\text{potential\_parent}}$ to childe node x_{child}
Output: Updated tree $\mathcal{T}$

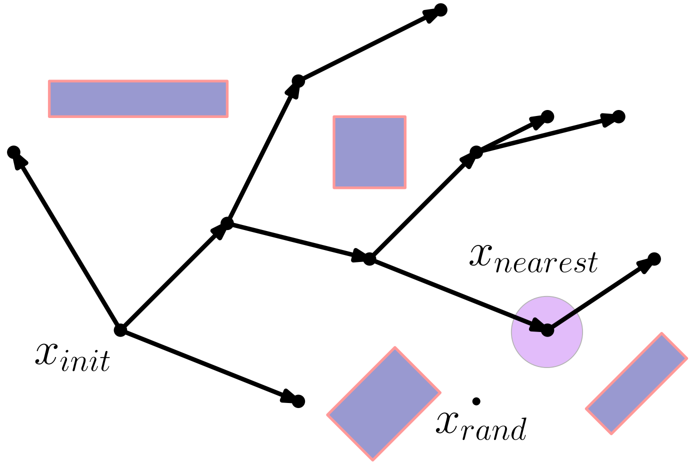
---

if ($\texttt{collision\_free}(x_{\text{potential\_parent}}, x_{\text{child}})$) **then**
  $c \leftarrow \texttt{cost}(x_{\text{potential\_parent}}, x_{\text{child}})$
  if ($\texttt{cost}_{\mathcal{T}}(x_{\text{potential\_parent}}) + c < \texttt{cost}_{\mathcal{T}}(x_{\text{child}})$) **then**
    $\mathcal{T}.\texttt{parent}(x_{\text{child}}) \leftarrow x_{\text{potential\_parent}}$

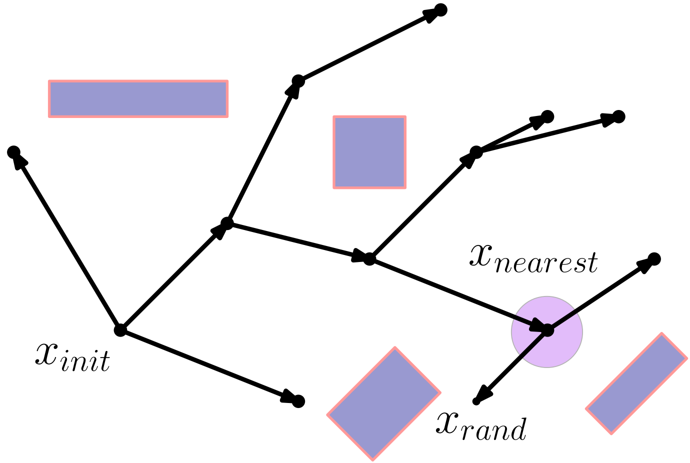
RRT*—`rewire` operations



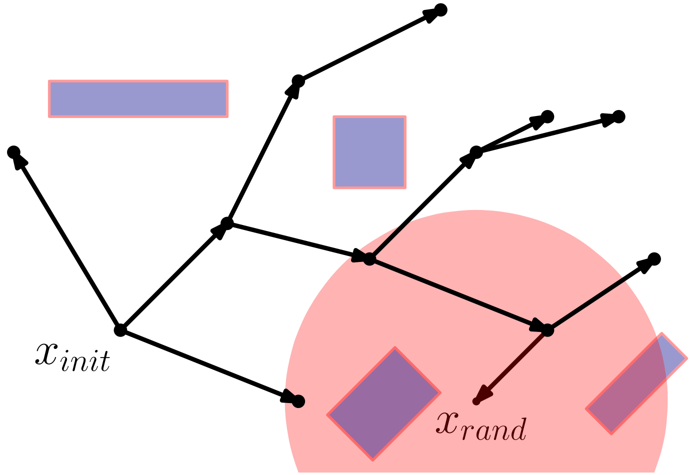
RRT *—rewire operations



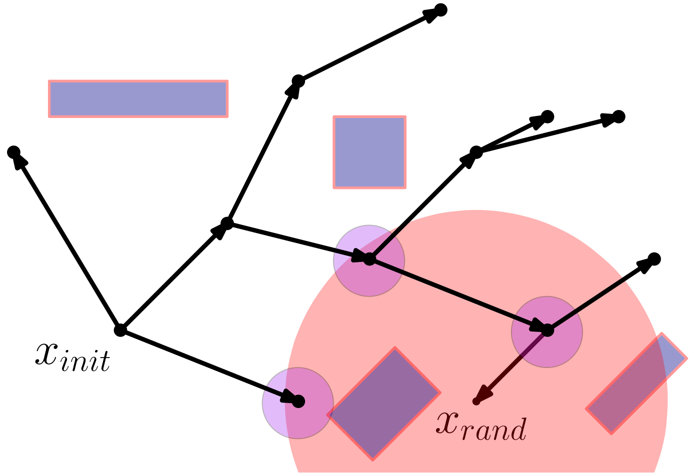
RRT*—`rewire` operations



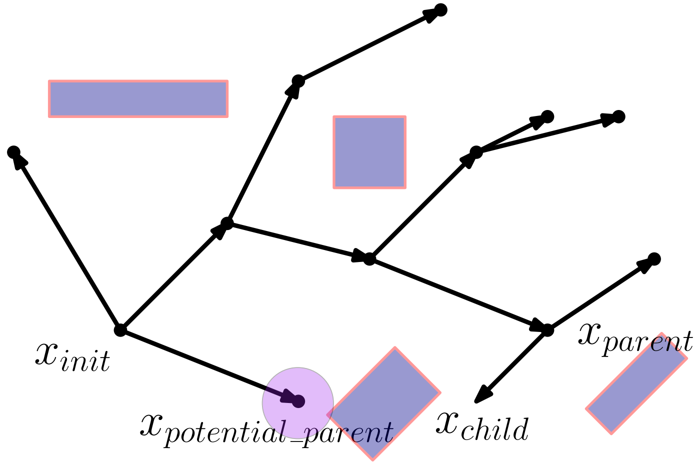
RRT*—rewire operations



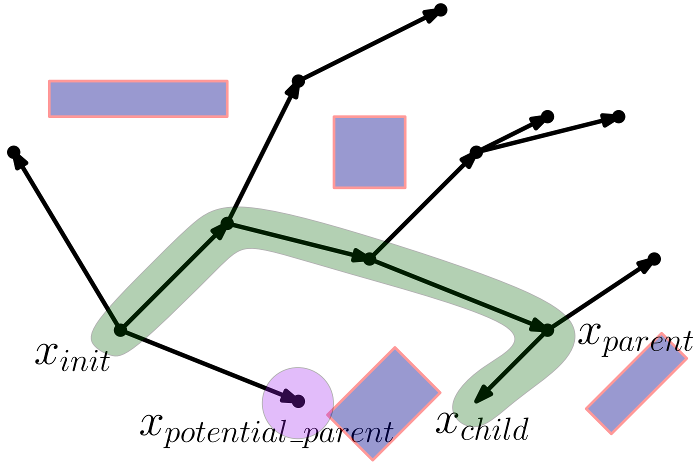
RRT*—rewire operations

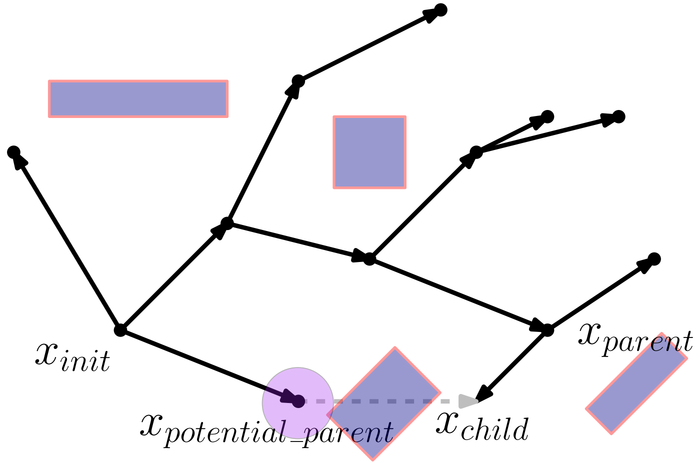


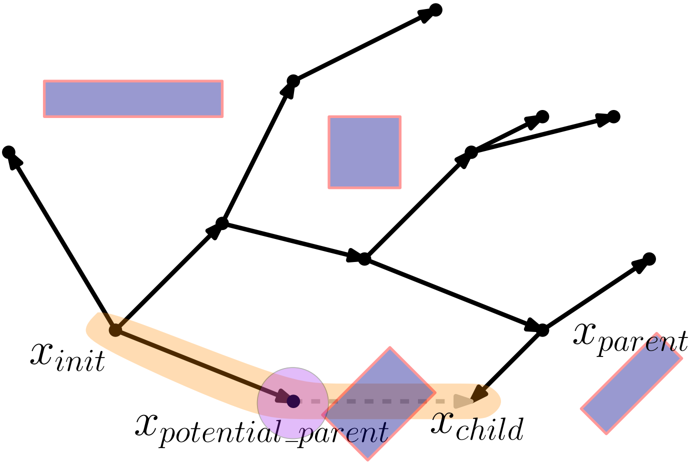
RRT*—rewire operations

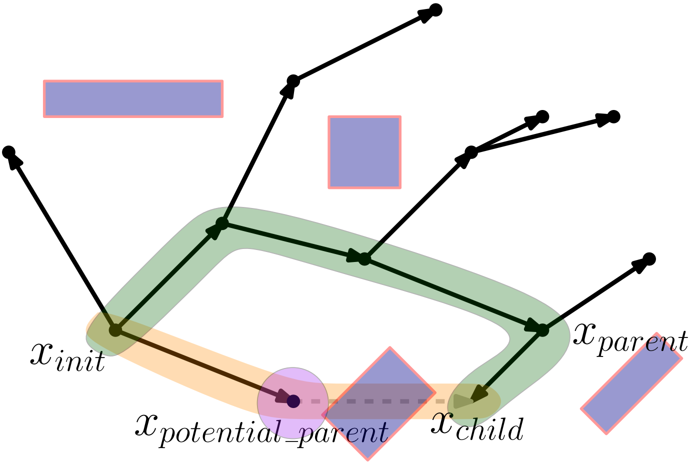


RRT*—`rewire` operations

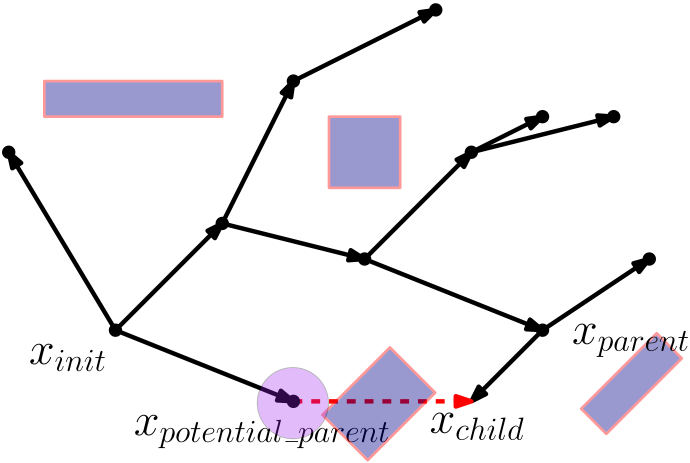


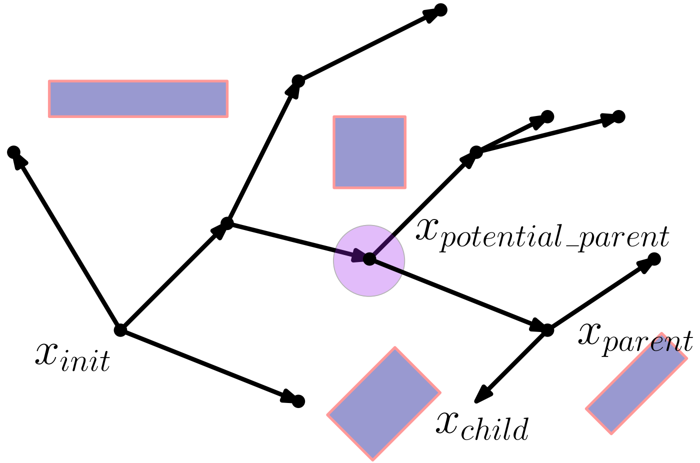

x_{init}
x_{parent}
$x_{potential\_parent}$
x_{child}

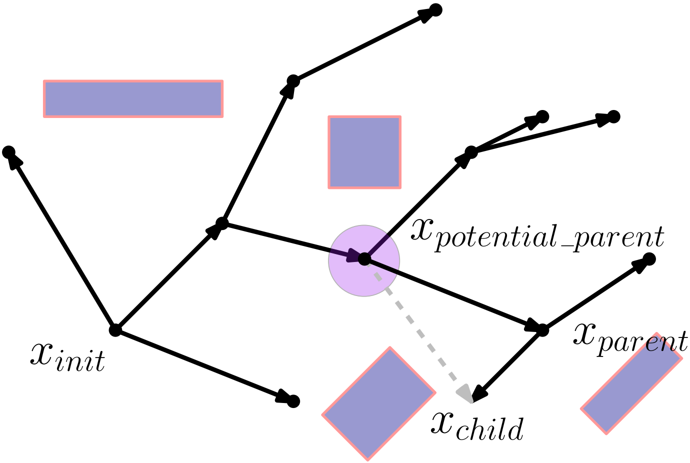

x_{init}
$x_{potential\_parent}$
x_{child}
x_{parent}


x_{init}
x_{parent}
$x_{potential\_parent}$
x_{child}

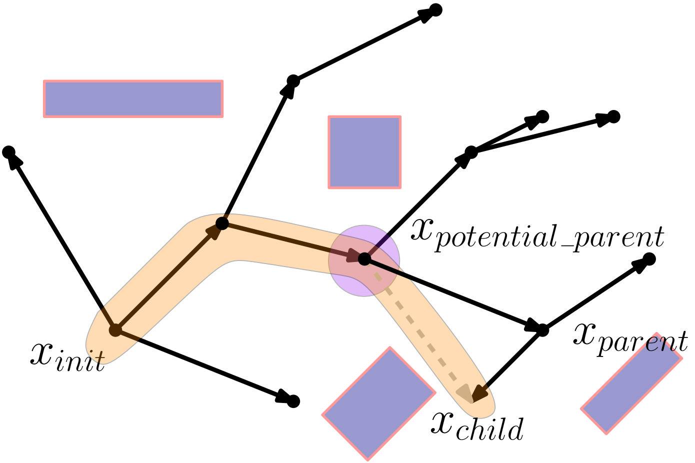
RRT *—rewire operations



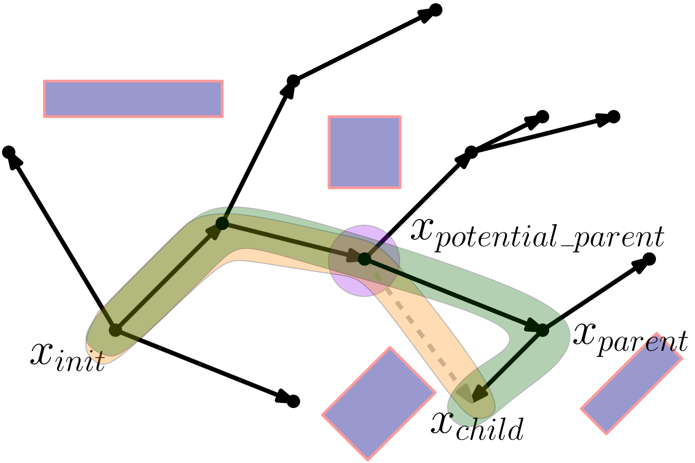

$x_{potential\_parent}$
x_{init}
x_{parent}
x_{child}


$x_{potential\_parent}$
x_{parent}
x_{init}
x_{child}

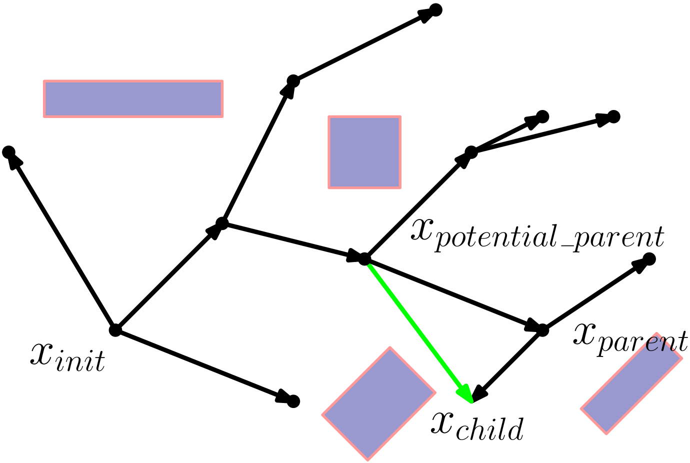
RRT*—`rewire` operations



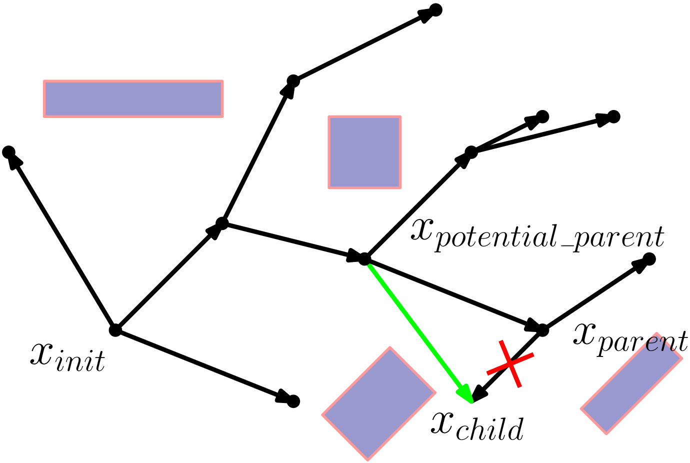
RRT *—rewire operations



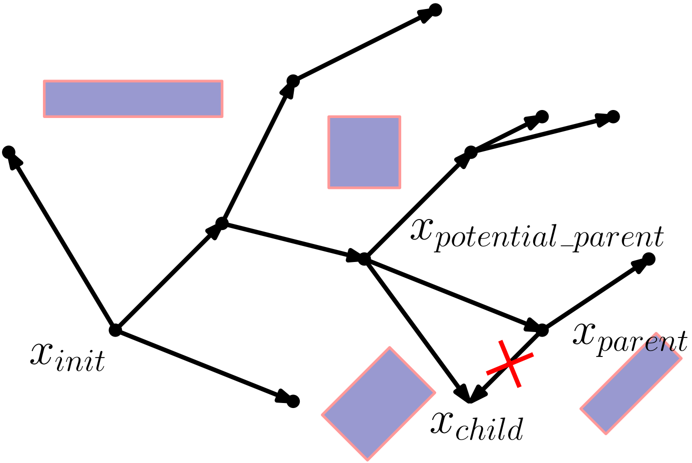
RRT *—`rewire` operations



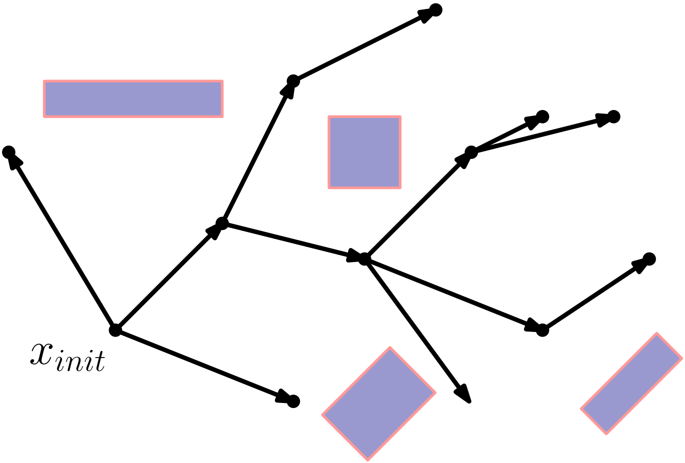
RRT *—rewire operations

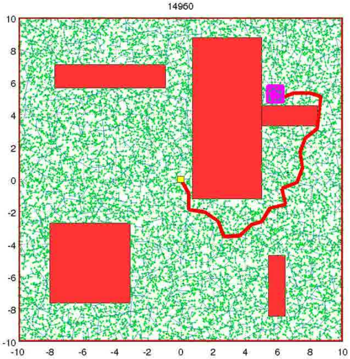


RRT *—rewire operations

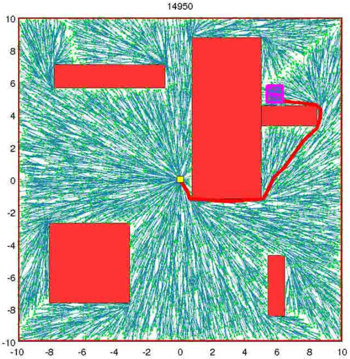


RRT*—`rewire` operations





RRT



RRT*

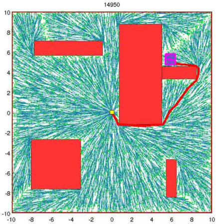
Videos created by Sertac Karaman, adapted from `http://y2u.be/FAFw8DoKvik` and `http://y2u.be/YKiQTJpPFkA`

RRT *—computational complexity

- The connection radius ensures that at the i^{th} iteration, we consider $O(\log i)$ nodes in $\mathcal{T}$ (in expectation) [Karaman, Frazzoli11]
- Nearest-neighbors computation takes $\Omega(n \log n)$ time [Karaman, Frazzoli11, Kleinbort, S., Halperin16]

Thm [Karaman, Frazzoli11, Kleinbort, S., Halperin16]

The complexity of the RRT* algorithm run with n samples is $\Omega(n \log n)$.



Definitions

Definition

The δ-interior (of $\mathcal{X}_{\text{free}}$), denoted by $\text{int}_\delta(\mathcal{X}_{\text{free}})$, is the set of all points $x \in \mathcal{X}_{\text{free}}$ that are within distance δ from $\mathcal{X}_{\text{obs}}$.

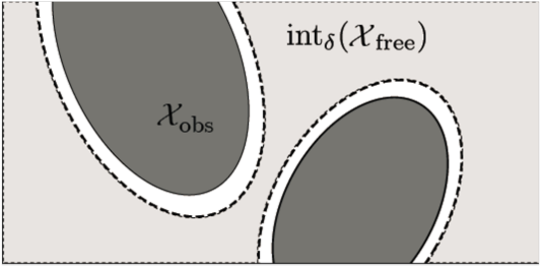


Figure adapted from [Karaman Frazzoli11]

Definitions (for quality of paths produced by RRT *)

Definition

A collision-free path σ is said to have strong δ-clearance if $\forall \tau \in [0,1]\ \sigma(\tau) \in \mathrm{int}_\delta(\mathcal{X}_{\mathrm{free}})$

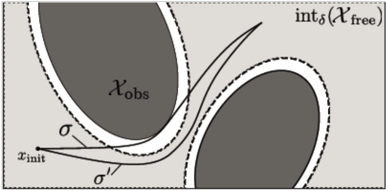
Definition

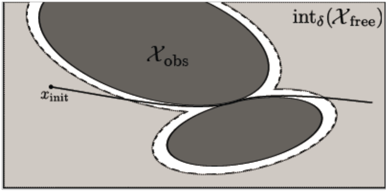
Two collision-free paths are said to be in the same homotopy class if there exists a continuous deformation between the paths that is in $\mathcal{X}_{\mathrm{free}}$

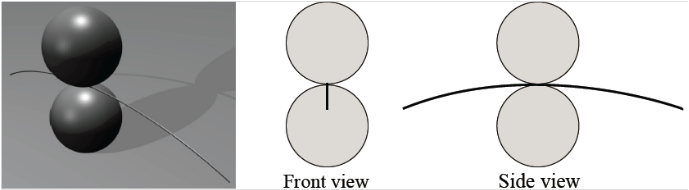
Definition

A collision-free path σ is said to have weak δ-clearance if there exists a path in its homotopy class with strong δ-clearance

Path type - strong or weak δ-clearance







Figures adapted from [Karaman Frazzoli11]

Definitions (for quality of paths produced by $\mathtt{RRT}^*$)

Definition

A feasible path $\sigma^* \in \mathcal{X}_{\text{free}}$ is said to be a robustly optimal solution if:

- It is optimal (i.e. $c(\sigma^*) = \min\{c(\sigma),\ \sigma \text{ is feasible}\}$ for a cost c
- It has weak δ-clearance
- For any sequence of collision free paths $\{\sigma_n\}$ s.t. $\lim_{n\to\infty}\sigma_n = \sigma^*$, $\lim_{n\to\infty} c(\sigma_n) = c(\sigma^*)$

Definitions (for quality of paths produced by RRT *)

Definition

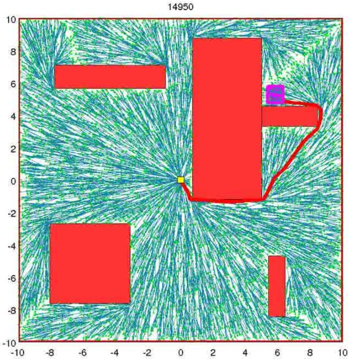
An algorithm ALG is asymptotically optimal if, for any path planning problem $(\mathcal{X}_{\mathrm{free}}, x_{init}, \mathcal{X}_{goal})$ and cost function $c : \Sigma \rightarrow \mathbb{R}_{\geq 0}$ that admit a robustly optimal solution with finite cost c^*

$$\Pr\left(lim\ sup_{n \rightarrow \infty} Y_n^{ALG} = c^* \right) = 1.$$

Where Y_n^{ALG} is the random variable corresponding to the cost of the minimum-cost solution included in the graph returned by ALG at the end of iteration n

Thm [Karaman, Frazzoli11]

The RRT* algorithm is asymptotically optimal.



Setting: Single-query motion planning
Common approach: Sampling-based (RRTs)
Optimize: Path-length

Scenario taken from OMPL

- RRT [LaValle Kuffner01] — Fast, not optimal
- RRG, RRT* [Karaman Frazzoli 11] — Slower, asymptotically optimal

Setting: Single-query motion planning
Common approach: Sampling-based (RRTs)
Optimize: Path-length

Scenario taken from OMPL

- RRT [LaValle Kuffner01] — Fast, not optimal
- RRG, RRT* [Karaman Frazzoli 11] — Slower, asymptotically optimal

Scenario taken from OMPL

Setting: Single-query motion planning
Common approach: Sampling-based (RRTs)
Optimize: Path-length

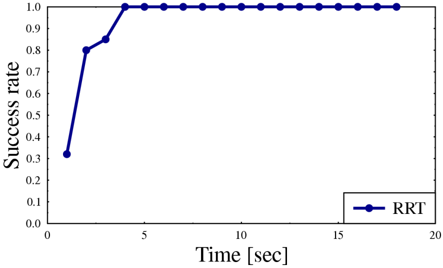
- RRT [LaValle Kuffner01] — Fast, not optimal
- RRG, RRT* [Karaman Frazzoli 11] — Slower, asymptotically optimal

RRTs: quality vs. computational complexity

Setting: Single-query motion planning
Common approach: Sampling-based (RRTs)
Optimize: Path-length

Scenario taken from OMPL

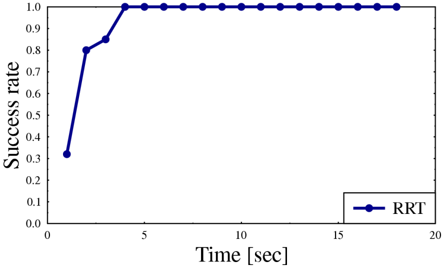
- RRT [LaValle Kuffner01] — Fast, not optimal
- RRG, RRT* [Karaman Frazzoli 11] — Slower, asymptotically optimal

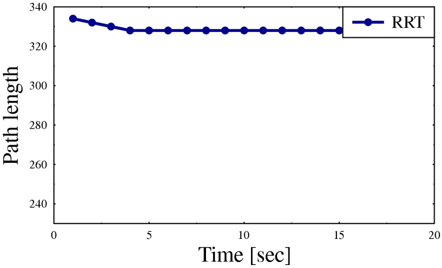


Setting: Single-query motion planning
Common approach: Sampling-based (RRTs)
Optimize: Path-length

Scenario taken from OMPL

- RRT [LaValle Kuffner01] — Fast, not optimal
- RRG, RRT* [Karaman Frazzoli 11] — Slower, asymptotically optimal

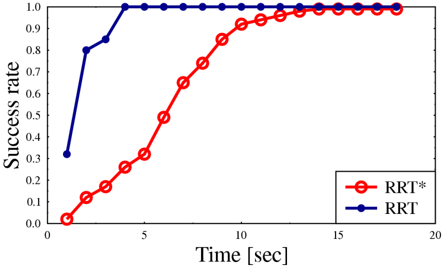


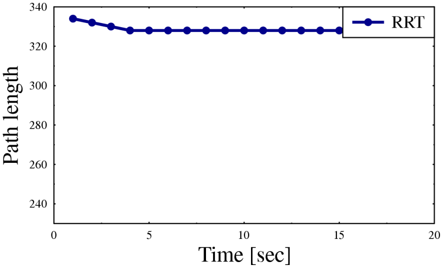


Scenario taken from OMPL

Setting: Single-query motion planning
Common approach: Sampling-based (RRTs)
Optimize: Path-length

- RRT [LaValle Kuffner01] — Fast, not optimal
- RRG, RRT* [Karaman Frazzoli 11] — Slower, asymptotically optimal



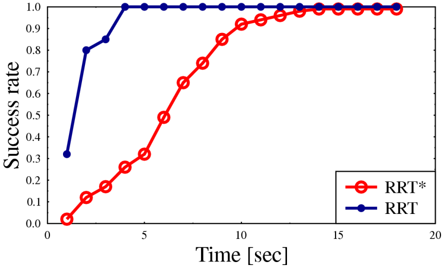


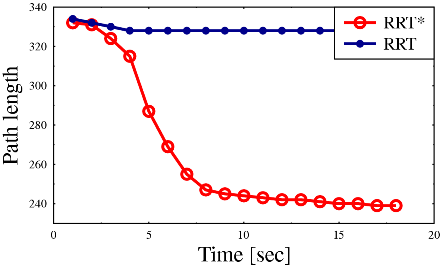
RRTs: quality vs. computational complexity

Setting: Single-query motion planning
Common approach: Sampling-based (RRTs)
Optimize: Path-length

Scenario taken from OMPL

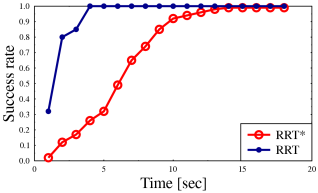
- RRT [LaValle Kuffner01] — Fast, not optimal
- RRG, RRT* [Karaman Frazzoli 11] — Slower, asymptotically optimal

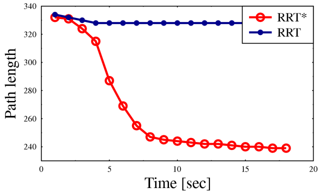




Related work - high-quality RRTs (partial list)

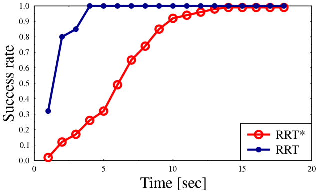
- Improving the quality of RRT
  - Post-processing existing paths [GO07]
  - Path hybridization [REH11]
  - Changing the sampling strategy [LTA03, US03, SWT09]
  - Changing connection scheme to a new milestone [US03, SLN00]
- Improving the convergence rate of RRT*
  - Lazy computation [FKSFTW11]
  - Adding heuristics [INMAH12]
  - Changing the sampling strategy [GSB14]
- Relaxing optimality of RRT* [LLB13]

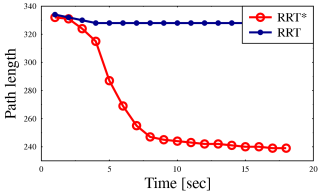




Related work - high-quality RRTs (partial list)

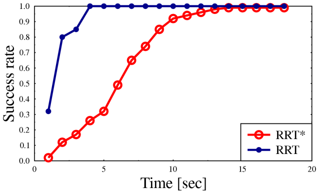
- Improving the quality of RRT
  - Post-processing existing paths [GO07]
  - Path hybridization [REH11]
  - Changing the sampling strategy [LTA03, US03, SWT09]
  - Changing connection scheme to a new milestone [US03, SLN00]
- Improving the convergence rate of RRT*
  - Lazy computation [FKSFTW11]
  - Adding heuristics [INMAH12]
  - Changing the sampling strategy [GSB14]
- Relaxing optimality of RRT* [LLB13]

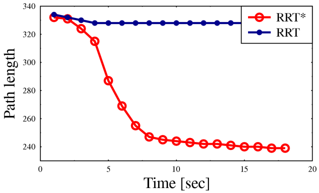




Related work - high-quality RRTs (partial list)

- Improving the quality of RRT
  - Post-processing existing paths [GO07]
  - Path hybridization [REH11]
  - Changing the sampling strategy [LTA03, US03, SWT09]
  - Changing connection scheme to a new milestone [US03, SLN00]
- Improving the convergence rate of RRT*
  - Lazy computation [FKSFTW11]
  - Adding heuristics [INMAH12]
  - Changing the sampling strategy [GSB14]
- Relaxing optimality of RRT* [LLB13]





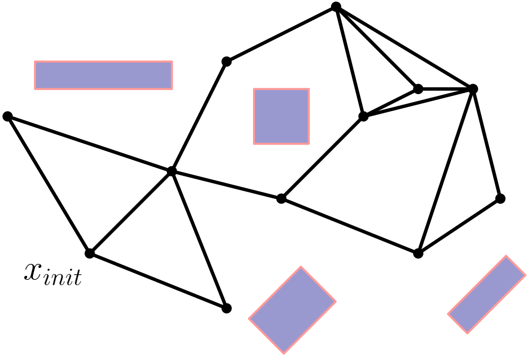
Lower Bound Tree-RRT (LBT−RRT) [S. Halperin16]

- Lower Bound Tree-RRT (LBT−RRT) is an asymptotically near-optimal planner
- LBT−RRT continuously interpolates between the fast RRT and the asymptotically optimal RRT*

Approximation factor ($1+\varepsilon$)	**Behavior**
No approximation ($\varepsilon = 0$)	like RRT* (asymptotically optimal)
Unbounded approximation ($\varepsilon = \infty$)	like RRT (fast)
In between ($0 < \varepsilon < \infty$)	higher-quality paths than RRT faster than RRG, RRT*

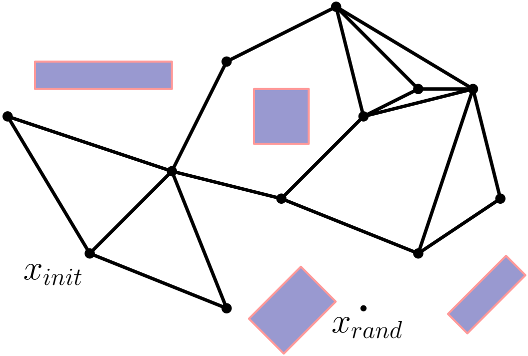
Algorithmic background - RRG

- Explores the configuration-space by constructing a graph
- Uses connection radius $r(n) \approx \gamma_{\mathrm{RRG}}(d) \left(\frac{\log n}{n}\right)^{1/d}$
  n - number of nodes, d - dimension of configuration space

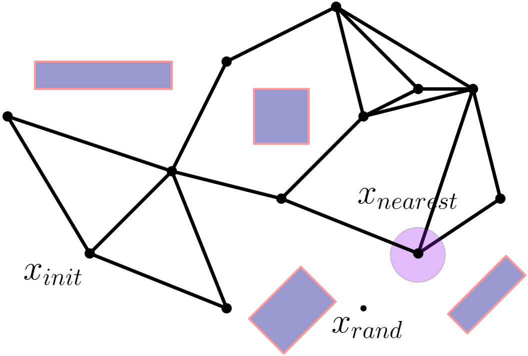


Algorithmic background - RRG

- Explores the configuration-space by constructing a graph
- Uses connection radius $r(n) \approx \gamma_{\mathrm{RRG}}(d) \left(\frac{\log n}{n} \right)^{1/d}$
  n - number of nodes, d - dimension of configuration space

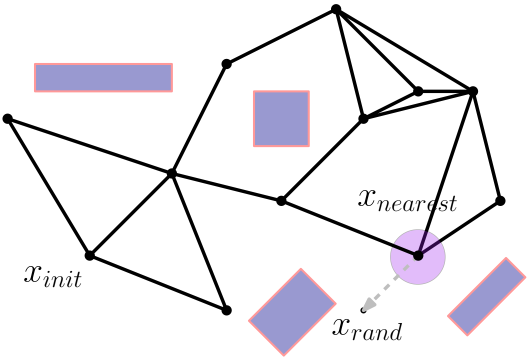


- Explores the configuration-space by constructing a graph
- Uses connection radius $r(n) \approx \gamma_{\text{RRG}}(d)\left(\frac{\log n}{n}\right)^{1/d}$
  n - number of nodes, d - dimension of configuration space



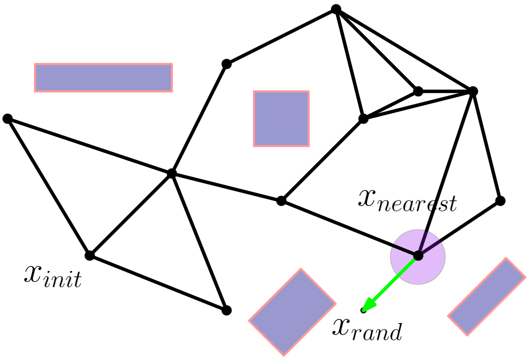
- Explores the configuration-space by constructing a graph
- Uses connection radius $r(n) \approx \gamma_{\text{RRG}}(d) \left(\frac{\log n}{n}\right)^{1/d}$

  n - number of nodes, d - dimension of configuration space



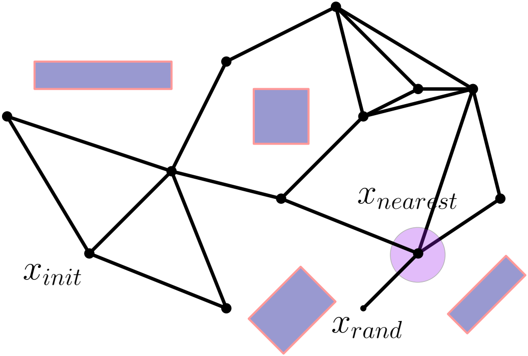
Algorithmic background - RRG

- Explores the configuration-space by constructing a graph
- Uses connection radius $r(n) \approx \gamma_{\mathrm{RRG}}(d)\left(\frac{\log n}{n}\right)^{1/d}$
  n - number of nodes, d - dimension of configuration space



Algorithmic background - RRG

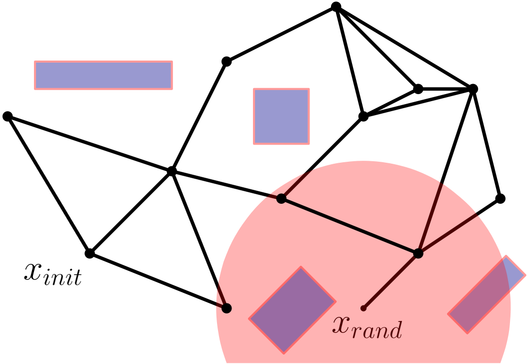
- Explores the configuration-space by constructing a graph
- Uses connection radius $r(n) \approx \gamma_{\text{RRG}}(d) \left(\frac{\log n}{n}\right)^{1/d}$
  n - number of nodes, d - dimension of configuration space



Algorithmic background - RRG

- Explores the configuration-space by constructing a graph
- Uses connection radius $r(n) \approx \gamma_{\text{RRG}}(d) \left(\frac{\log n}{n}\right)^{1/d}$

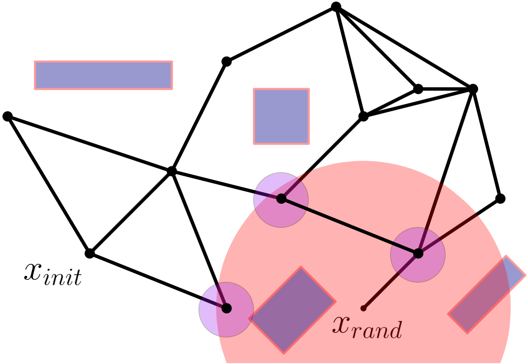
  n - number of nodes, d - dimension of configuration space



Algorithmic background - RRG

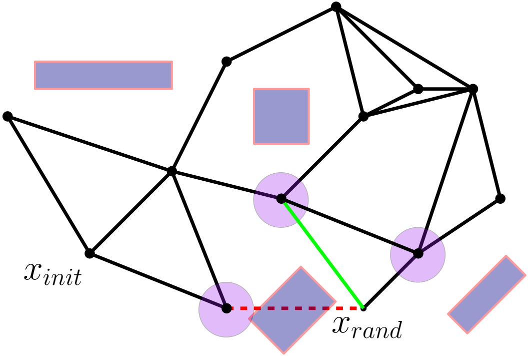
- Explores the configuration-space by constructing a graph
- Uses connection radius $r(n) \approx \gamma_{\mathrm{RRG}}(d)\left(\frac{\log n}{n}\right)^{1/d}$

  n - number of nodes, d - dimension of configuration space



Algorithmic background - RRG

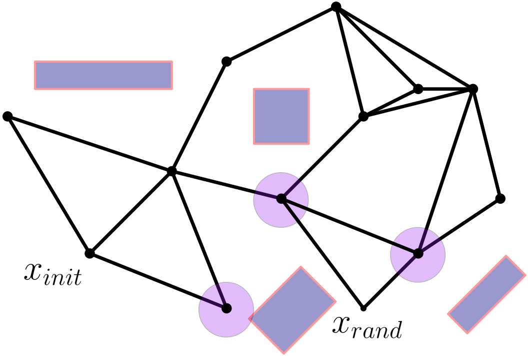
- Explores the configuration-space by constructing a graph
- Uses connection radius $r(n) \approx \gamma_{\mathrm{RRG}}(d) \left(\frac{\log n}{n}\right)^{1/d}$
  n - number of nodes, d - dimension of configuration space



Algorithmic background - RRG

- Explores the configuration-space by constructing a graph
- Uses connection radius $r(n) \approx \gamma_{\text{RRG}}(d) \left(\frac{\log n}{n}\right)^{1/d}$

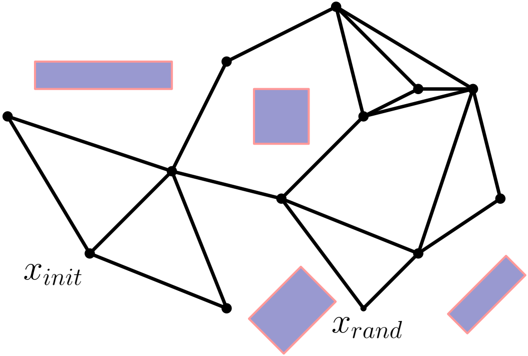
  n - number of nodes, d - dimension of configuration space



Algorithmic background - RRG

- Explores the configuration-space by constructing a graph
- Uses connection radius $r(n) \approx \gamma_{\mathrm{RRG}}(d) \left(\frac{\log n}{n}\right)^{1/d}$

  n - number of nodes, d - dimension of configuration space



Lower Bound Tree-RRT (LBT−RRT) - motivation

Problem:

- Rewiring may call the expensive local planner $O(\log n)$ times per sample
- This is one of the most time-consuming parts of RRT*

Solution:

- LBT−RRT maintains two roadmaps: $\mathcal{G}_{lb}$ and $\mathcal{T}_{apx}$ (over the same set of vertices as the RRG roadmap)
- The two roadmaps, which are faster to maintain than the RRT* tree and the RRG roadmap, guarantee asymptotic near-optimality

Lower Bound Tree-RRT (LBT-RRT) - motivation

Problem:

- Rewiring may call the expensive local planner $O(\log n)$ times per sample
- This is one of the most time-consuming parts of RRT*

Solution:

- LBT-RRT maintains two roadmaps: $\mathcal{G}_{lb}$ and $\mathcal{T}_{apx}$ (over the same set of vertices as the RRG roadmap)
- The two roadmaps, which are faster to maintain than the RRT* tree and the RRG roadmap, guarantee asymptotic near-optimality

Lower Bound Tree-RRT (LBT−RRT) - motivation

Problem:

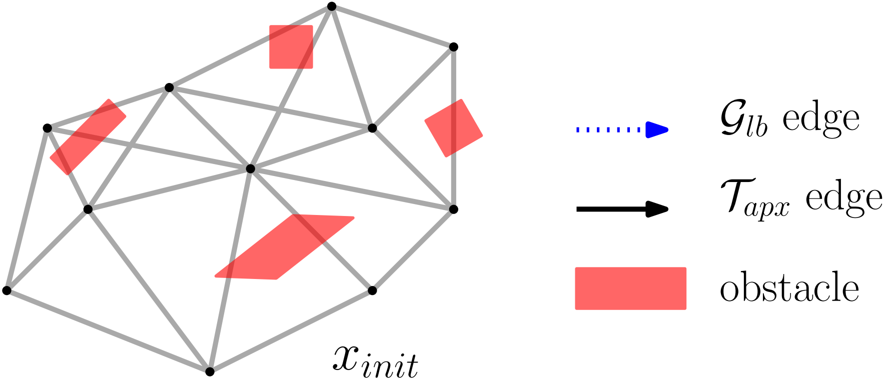
- Rewiring may call the expensive local planner $O(\log n)$ times per sample
- This is one of the most time-consuming parts of RRT*

Solution:

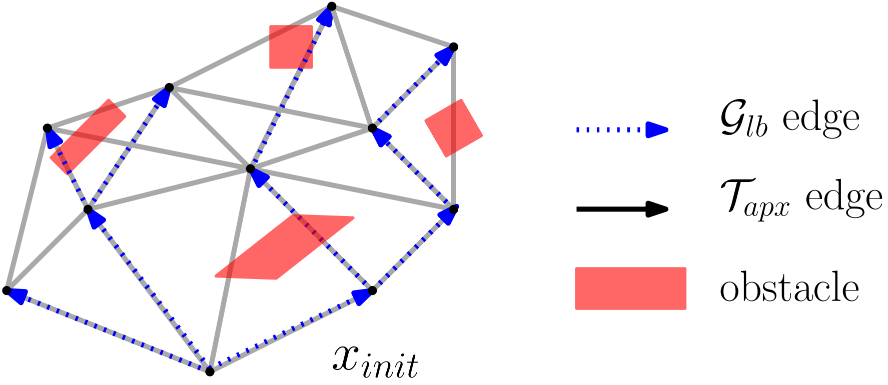
- LBT−RRT maintains two roadmaps: $\mathcal{G}_{lb}$ and $\mathcal{T}_{apx}$ (over the same set of vertices as the RRG roadmap)
- The two roadmaps, which are faster to maintain than the RRT* tree and the RRG roadmap, guarantee asymptotic near-optimality

Lower Bound Tree-RRT (LBT-RRT) - roadmaps

- $\mathcal{G}_{lb}$ is roadmap possibly containing non collision-free edges
- $\mathcal{T}_{apx}$ is a subgraph of the $\mathcal{G}_{lb}$ roadmap with collision-free edges only

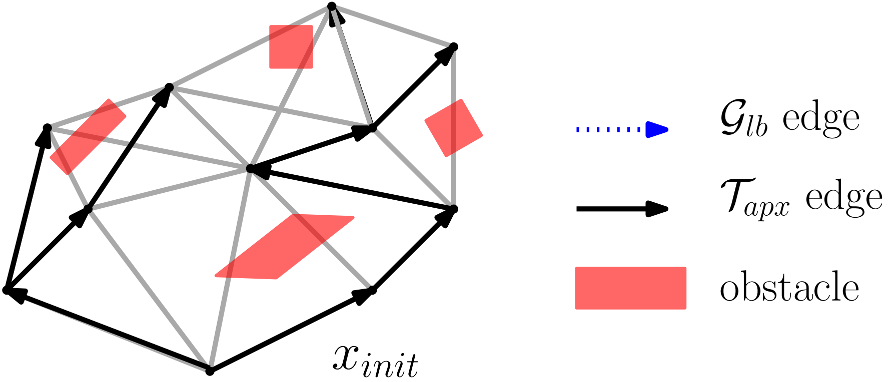


- $\mathcal{G}_{lb}$ is roadmap possibly containing non collision-free edges
- $\mathcal{T}_{apx}$ is a subgraph of the $\mathcal{G}_{lb}$ roadmap with collision-free edges only

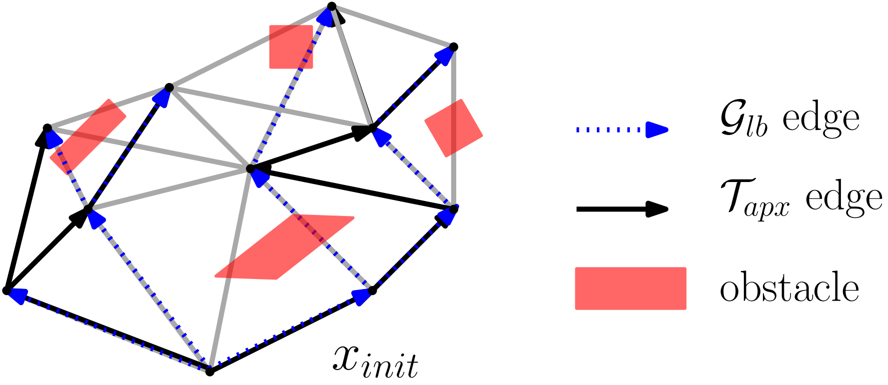


Lower Bound Tree-RRT (LBT−RRT) - roadmaps

- $\mathcal{G}_{lb}$ is roadmap possibly containing non collision-free edges
- $\mathcal{T}_{apx}$ is a subgraph of the $\mathcal{G}_{lb}$ roadmap with collision-free edges only



- $\mathcal{G}_{lb}$ is roadmap possibly containing non collision-free edges
- $\mathcal{T}_{apx}$ is a subgraph of the $\mathcal{G}_{lb}$ roadmap with collision-free edges only



Lower Bound Tree-RRT (LBT−RRT) - invariants

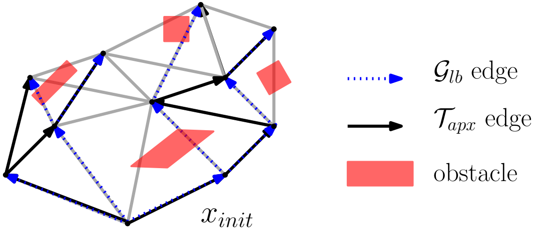
Given a parameter ε, the following invariants are maintained:

Bounded approximation invariant

For every node $x \in \mathcal{G}_{lb}, \mathcal{T}_{apx}$, $\mathtt{cost}_{\mathcal{T}_{apx}}(x) \leq (1+\varepsilon) \cdot \mathtt{cost}_{\mathcal{G}_{lb}}(x)$.

Lower bound invariant

For every node $x \in \mathcal{G}_{lb}, \mathcal{T}_{apx}$, $\mathtt{cost}_{\mathcal{G}_{lb}}(x) \leq \mathtt{cost}_{\mathcal{G}_{RRG}}(x)$.



Lower Bound Tree-RRT (LBT-RRT) - invariants

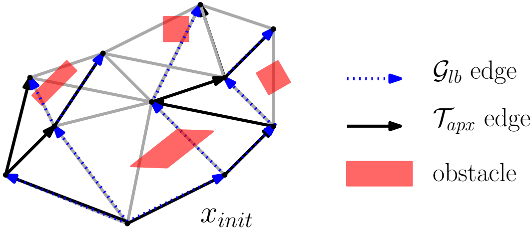
Given a parameter ε, the following invariants are maintained:

Bounded approximation invariant

For every node $x \in \mathcal{G}_{lb}, \mathcal{T}_{apx}$, $\texttt{cost}_{\mathcal{T}_{apx}}(x) \leq (1+\varepsilon) \cdot \texttt{cost}_{\mathcal{G}_{lb}}(x)$.

Lower bound invariant

For every node $x \in \mathcal{G}_{lb}, \mathcal{T}_{apx}$, $\texttt{cost}_{\mathcal{G}_{lb}}(x) \leq \texttt{cost}_{\mathcal{G}_{RRG}}(x)$.



Lower Bound Tree-RRT (LBT−RRT) - invariants

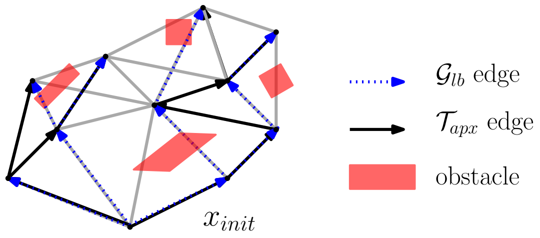
Given a parameter ε, the following invariants are maintained:

Bounded approximation invariant

For every node $x \in \mathcal{G}_{lb}, \mathcal{T}_{apx}$, $\mathtt{cost}_{\mathcal{T}_{apx}}(x) \leq (1+\varepsilon) \cdot \mathtt{cost}_{\mathcal{G}_{lb}}(x)$.

Lower bound invariant

For every node $x \in \mathcal{G}_{lb}, \mathcal{T}_{apx}$, $\mathtt{cost}_{\mathcal{G}_{lb}}(x) \leq \mathtt{cost}_{\mathcal{G}_{RRG}}(x)$.



Lower Bound Tree-RRT (LBT-RRT) - invariants

Given a parameter ε, the following invariants are maintained:

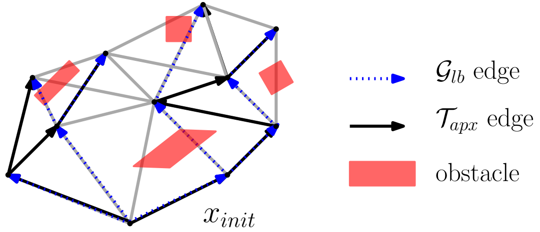
Bounded approximation invariant

For every node $x \in \mathcal{G}_{lb}, \mathcal{T}_{apx}$, $\texttt{cost}_{\mathcal{T}_{apx}}(x) \leq (1+\varepsilon) \cdot \texttt{cost}_{\mathcal{G}_{lb}}(x)$.

Lower bound invariant

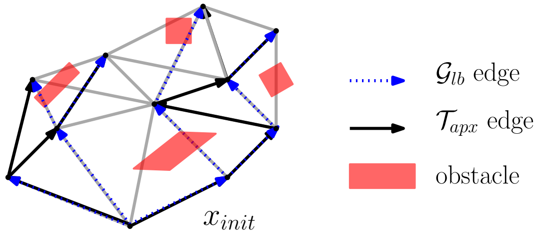
For every node $x \in \mathcal{G}_{lb}, \mathcal{T}_{apx}$, $\texttt{cost}_{\mathcal{G}_{lb}}(x) \leq \texttt{cost}_{\mathcal{G}_{RRG}}(x)$.

The combination of the two invariants ensure that LBT-RRT is asymptotically near-optimal with an approximation factor of $1+\varepsilon$



Maintaining the trees

LBT−RRT follows the same structure as RRT, RRG RRT* with respect to adding a new milestone

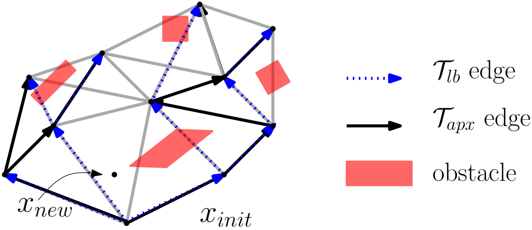


It differs with respect to the additional edges considered

In contrast to RRT* where rewiring is always performed, and RRG where the local planner is called for all all neighbors, LBT−RRT only calls the local planner for edges necessary to maintain the lower bound invariant

Maintaining the trees

LBT-RRT follows the same structure as RRT, RRG RRT* with respect to adding a new milestone

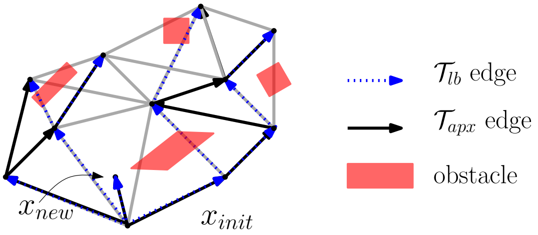


It differs with respect to the additional edges considered

In contrast to RRT* where rewiring is always performed, and RRG where the local planner is called for all all neighbors, LBT-RRT only calls the local planner for edges necessary to maintain the lower bound invariant

Maintaining the trees

LBT-RRT follows the same structure as RRT, RRG RRT* with respect to adding a new milestone

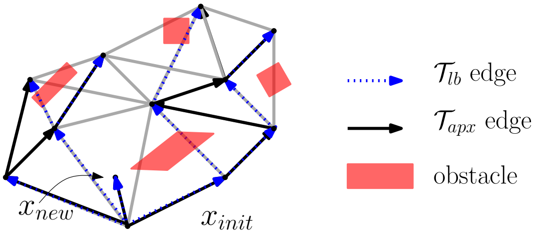


It differs with respect to the additional edges considered

In contrast to RRT* where rewiring is always performed, and RRG where the local planner is called for all all neighbors, LBT-RRT only calls the local planner for edges necessary to maintain the lower bound invariant

Maintaining the trees

LBT-RRT follows the same structure as RRT, RRG RRT* with respect to adding a new milestone

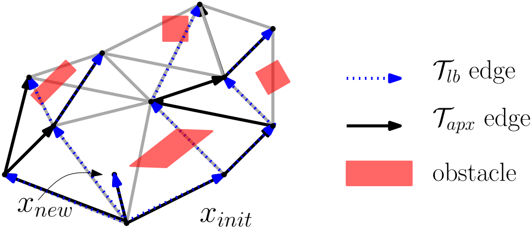


It differs with respect to the additional edges considered

In contrast to RRT* where rewiring is always performed, and RRG where the local planner is called for all all neighbors, LBT-RRT only calls the local planner for edges necessary to maintain the lower bound invariant

Maintaining the trees

LBT−RRT follows the same structure as RRT, RRG RRT* with respect to adding a new milestone



It differs with respect to the additional edges considered

In contrast to RRT* where rewiring is always performed, and RRG where the local planner is called for all all neighbors, LBT−RRT only calls the local planner for edges necessary to maintain the lower bound invariant

Dynamic single-sink shortest-path problem (SSSP)

Let $G = (V, E)$ be a graph that undergoes a series of edge insertions and edge deletions.

Maintaining the shortest-path from a given node to every other node in V is referred to as the Dynamic single-sink shortest-path problem

Efficient algorithms for Dynamic SSSP exist (e.g. [RR96, FSN00])

consider_edge(x_1, x_2)

Given two nodes x_1 and x_2 such that:

- $x_{new} \in \{x_1, x_2\}$
- $\|x_1, x_2\| \leq r(n)$

consider_edge adds (lazily) the edge (x_1, x_2) to $\mathcal{G}_{lb}$
$\Rightarrow$ $\mathtt{cost}_{\mathcal{G}_{lb}}$ possibly decreases

It then ensures that the bounded approximation invariant is maintained for all nodes by:

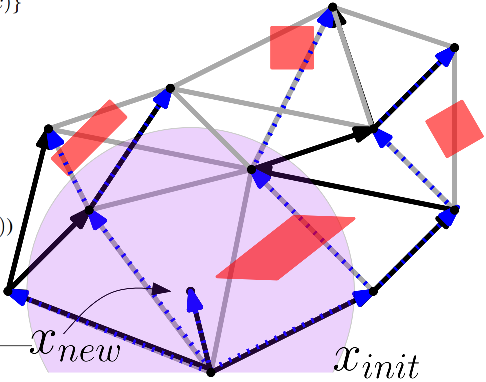
- Removing (in-collision) edges from $\mathcal{G}_{lb}$
  $\Rightarrow$ $\mathtt{cost}_{\mathcal{G}_{lb}}$ possibly increases
- Adding (collision-free) edges to $\mathcal{T}_{apx}$
  $\Rightarrow$ $\mathtt{cost}_{\mathcal{T}_{apx}}$ possibly decreases

Updates are performed using the procedures: insert_edge$_{\text{SSSP}}$, remove_edge$_{\text{SSSP}}$

`consider_edge`(x_1, x_2) - cont.

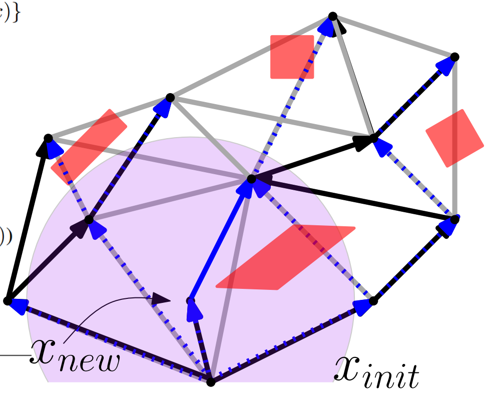
Algorithm 6 `consider_edge`(x_1, x_2)

$I \leftarrow \texttt{insert\_edge}_{\texttt{SSSP}}(\mathcal{G}_{lb}, (x_1, x_2))$
$Q \leftarrow \{x \in I \mid \text{cost}_{\mathcal{T}_{apx}}(x) > (1+\varepsilon) \cdot \text{cost}_{\mathcal{G}_{lb}}(x)\}$
while $Q \neq \emptyset$ **do**
  $x \leftarrow Q.\texttt{top}();$
  if $\text{cost}_{\mathcal{T}_{apx}}(x) > (1+\varepsilon) \cdot \text{cost}_{\mathcal{G}_{lb}}(x)$ **then**
    $x_{parent} \leftarrow \texttt{parent}_{\texttt{SSSP}}(\mathcal{G}_{lb}, x)$
    if $(\texttt{collision\_free}\ (x_{parent}, x))$ **then**
      $\mathcal{T}_{apx}.\texttt{parent}(x) \leftarrow x_{\text{parent}}$
      $Q.\texttt{pop}()$
    else
      $D \leftarrow \texttt{delete\_edge}_{\texttt{SSSP}}(\mathcal{G}_{lb}, (x_{\text{parent}}, x))$
      for all $y \in D \cap Q$ **do**
        $Q.\texttt{update\_cost}(y)$
  else
    $Q.\texttt{pop}()$



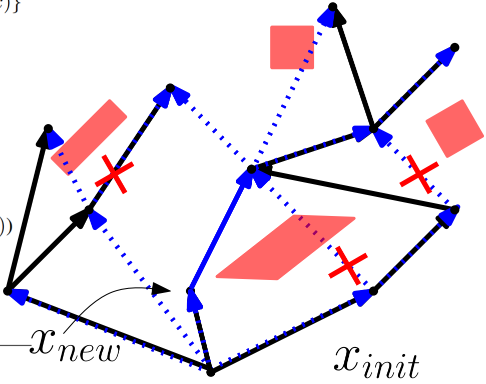
Algorithm 6 consider_edge(x_1, x_2)

```
I ← insert_edge_SSSP(G_lb, (x1, x2))
Q ← {x ∈ I | cost_T_apx(x) > (1 + ε) · cost_G_lb(x)}
while Q ≠ ∅ do
    x ← Q.top();
    if cost_T_apx(x) > (1 + ε) · cost_G_lb(x) then
        x_parent ← parent_SSSP(G_lb, x)
        if (collision_free (x_parent, x)) then
            T_apx.parent(x) ← x_parent
            Q.pop()
        else
            D ← delete_edge_SSSP(G_lb, (x_parent, x))
            for all y ∈ D ∩ Q do
                Q.update_cost(y)
    else
        Q.pop()
```



Algorithm 6 consider_edge(x_1, x_2)

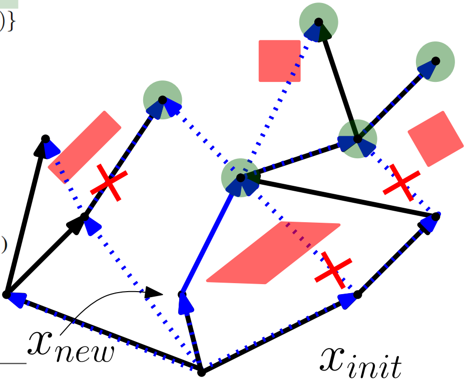
$I \leftarrow$ insert_edge$_{\text{SSSP}}(\mathcal{G}_{lb}, (x_1, x_2))$
$Q \leftarrow \{x \in I \mid \text{cost}_{\mathcal{T}_{apx}}(x) > (1+\varepsilon) \cdot \text{cost}_{\mathcal{G}_{lb}}(x)\}$
while $Q \neq \emptyset$ **do**
    $x \leftarrow Q$.top();
    if $\text{cost}_{\mathcal{T}_{apx}}(x) > (1+\varepsilon) \cdot \text{cost}_{\mathcal{G}_{lb}}(x)$ **then**
        $x_{parent} \leftarrow$ parent$_{\text{SSSP}}(\mathcal{G}_{lb}, x)$
        if (collision_free (x_{parent}, x)) **then**
            $\mathcal{T}_{apx}$.parent(x) $\leftarrow x_{\text{parent}}$
            Q.pop()
        else
            $D \leftarrow$ delete_edge$_{\text{SSSP}}(\mathcal{G}_{lb}, (x_{\text{parent}}, x))$
            for all $y \in D \cap Q$ **do**
                Q.update_cost(y)
    else
        Q.pop()



consider_edge(x_1, x_2) - cont.

Algorithm 6 consider_edge(x_1, x_2)

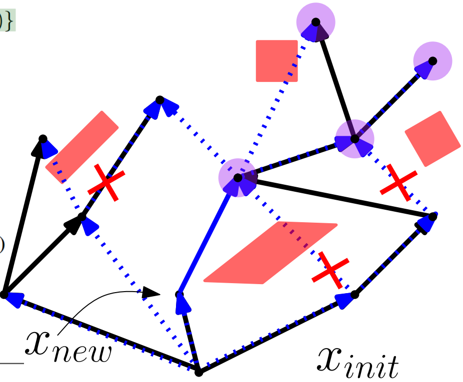
```
I ← insert_edge_SSSP(G_lb, (x1, x2))
Q ← {x ∈ I | cost_T_apx(x) > (1 + ε) · cost_G_lb(x)}
while Q ≠ ∅ do
    x ← Q.top();
    if cost_T_apx(x) > (1 + ε) · cost_G_lb(x) then
        x_parent ← parent_SSSP(G_lb, x)
        if (collision_free (x_parent, x)) then
            T_apx.parent(x) ← x_parent
            Q.pop()
        else
            D ← delete_edge_SSSP(G_lb, (x_parent, x))
            for all y ∈ D ∩ Q do
                Q.update_cost(y)
    else
        Q.pop()
```



consider_edge(x_1, x_2) - cont.

Algorithm 6 consider_edge(x_1, x_2)

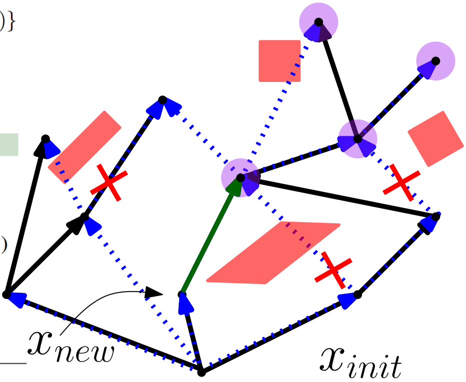
$I \leftarrow$insert_edge$_{\texttt{SSSP}}(\mathcal{G}_{lb}, (x_1, x_2))$
$Q \leftarrow \{x \in I \mid \text{cost}_{\mathcal{T}_{apx}}(x) > (1+\varepsilon) \cdot \text{cost}_{\mathcal{G}_{lb}}(x)\}$
while $Q \neq \emptyset$ **do**
  $x \leftarrow Q$.top();
  if $\text{cost}_{\mathcal{T}_{apx}}(x) > (1+\varepsilon) \cdot \text{cost}_{\mathcal{G}_{lb}}(x)$ **then**
    $x_{parent} \leftarrow$ parent$_{\texttt{SSSP}}(\mathcal{G}_{lb}, x)$
    if (collision_free (x_{parent}, x)) **then**
      $\mathcal{T}_{apx}$.parent$(x) \leftarrow x_{\text{parent}}$
      Q.pop()
    else
      $D \leftarrow$delete_edge$_{\texttt{SSSP}}(\mathcal{G}_{lb}, (x_{\text{parent}}, x))$
      for all $y \in D \cap Q$ **do**
        Q.update_cost(y)
  else
    Q.pop()



consider_edge(x_1, x_2) - cont.

Algorithm 6 consider_edge(x_1, x_2)

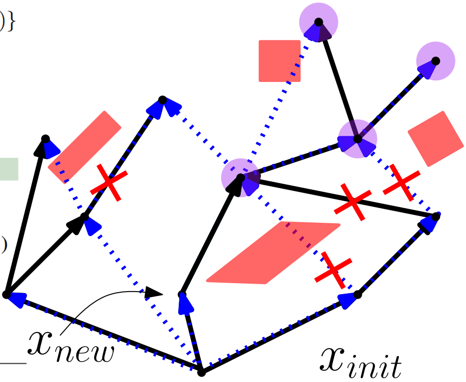
$I \leftarrow$ insert_edge$_{\texttt{SSSP}}(\mathcal{G}_{lb}, (x_1, x_2))$
$Q \leftarrow \{x \in I \mid \text{cost}_{\mathcal{T}_{apx}}(x) > (1+\varepsilon) \cdot \text{cost}_{\mathcal{G}_{lb}}(x)\}$
while $Q \neq \emptyset$ **do**
  $x \leftarrow Q$.top();
  if $\text{cost}_{\mathcal{T}_{apx}}(x) > (1+\varepsilon) \cdot \text{cost}_{\mathcal{G}_{lb}}(x)$ **then**
    $x_{parent} \leftarrow$ parent$_{\texttt{SSSP}}(\mathcal{G}_{lb}, x)$
    if (collision_free (x_{parent}, x)) **then**
      $\mathcal{T}_{apx}$.parent$(x) \leftarrow x_{\text{parent}}$
      Q.pop()
    else
      $D \leftarrow$ delete_edge$_{\texttt{SSSP}}(\mathcal{G}_{lb}, (x_{\text{parent}}, x))$
      for all $y \in D \cap Q$ **do**
        Q.update_cost(y)
  else
    Q.pop()



consider_edge(x_1, x_2) - cont.

Algorithm 6 consider_edge(x_1, x_2)

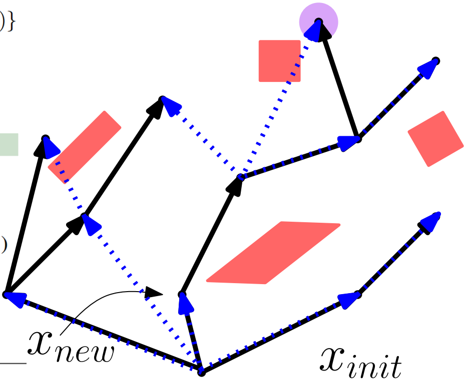
$I \leftarrow$ insert_edge$_{\text{SSSP}}(\mathcal{G}_{lb}, (x_1, x_2))$
$Q \leftarrow \{x \in I \mid \text{cost}_{\mathcal{T}_{apx}}(x) > (1+\varepsilon) \cdot \text{cost}_{\mathcal{G}_{lb}}(x)\}$
while $Q \neq \emptyset$ **do**
  $x \leftarrow Q$.top();
  if $\text{cost}_{\mathcal{T}_{apx}}(x) > (1+\varepsilon) \cdot \text{cost}_{\mathcal{G}_{lb}}(x)$ **then**
    $x_{parent} \leftarrow$ parent$_{\text{SSSP}}(\mathcal{G}_{lb}, x)$
    if (collision_free (x_{parent}, x)) **then**
      $\mathcal{T}_{apx}$.parent$(x) \leftarrow x_{\text{parent}}$
      Q.pop()
    else
      $D \leftarrow$ delete_edge$_{\text{SSSP}}(\mathcal{G}_{lb}, (x_{\text{parent}}, x))$
      for all $y \in D \cap Q$ **do**
        Q.update_cost(y)
  else
    Q.pop()



`consider_edge`(x_1, x_2) - cont.

Algorithm 6 `consider_edge`(x_1, x_2)

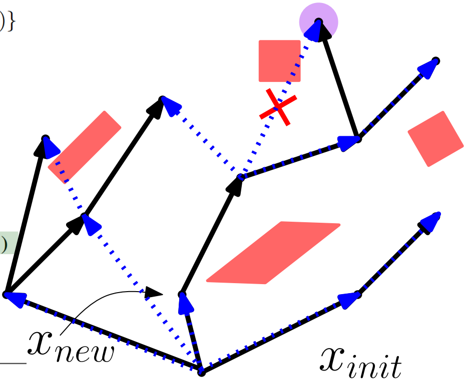
$I \leftarrow$ `insert_edge`$_{\texttt{SSSP}}(\mathcal{G}_{lb}, (x_1, x_2))$
$Q \leftarrow \{x \in I \mid \text{cost}_{\mathcal{T}_{apx}}(x) > (1+\varepsilon) \cdot \text{cost}_{\mathcal{G}_{lb}}(x)\}$
while $Q \neq \emptyset$ **do**
  $x \leftarrow Q$.`top`();
  if $\text{cost}_{\mathcal{T}_{apx}}(x) > (1+\varepsilon) \cdot \text{cost}_{\mathcal{G}_{lb}}(x)$ **then**
    $x_{parent} \leftarrow$ `parent`$_{\texttt{SSSP}}(\mathcal{G}_{lb}, x)$
    if (`collision_free` (x_{parent}, x)) **then**
      $\mathcal{T}_{apx}$.`parent`$(x) \leftarrow x_{\text{parent}}$
      Q.`pop`()
    else
      $D \leftarrow$ `delete_edge`$_{\texttt{SSSP}}(\mathcal{G}_{lb}, (x_{\text{parent}}, x))$
      for all $y \in D \cap Q$ **do**
        Q.`update_cost`(y)
  else
    Q.`pop`()



consider_edge(x_1, x_2) - cont.

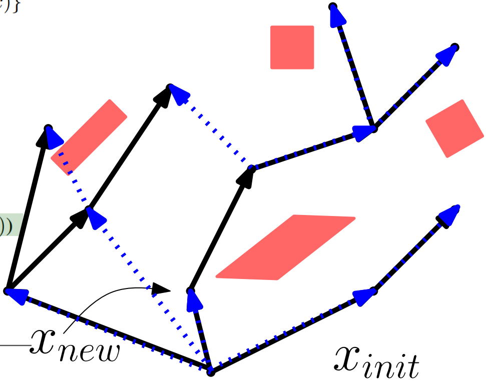
Algorithm 6 consider_edge(x_1, x_2)

$I \leftarrow$ insert_edge$_{\text{SSSP}}(\mathcal{G}_{lb}, (x_1, x_2))$
$Q \leftarrow \{x \in I \mid \text{cost}_{\mathcal{T}_{apx}}(x) > (1+\varepsilon) \cdot \text{cost}_{\mathcal{G}_{lb}}(x)\}$
while $Q \neq \emptyset$ **do**
  $x \leftarrow Q$.top();
  if $\text{cost}_{\mathcal{T}_{apx}}(x) > (1+\varepsilon) \cdot \text{cost}_{\mathcal{G}_{lb}}(x)$ **then**
    $x_{parent} \leftarrow$ parent$_{\text{SSSP}}(\mathcal{G}_{lb}, x)$
    if (collision_free (x_{parent}, x)) **then**
      $\mathcal{T}_{apx}$.parent$(x) \leftarrow x_{\text{parent}}$
      Q.pop()
    else
      $D \leftarrow$ delete_edge$_{\text{SSSP}}(\mathcal{G}_{lb}, (x_{\text{parent}}, x))$
      for all $y \in D \cap Q$ **do**
        Q.update_cost(y)
  else
    Q.pop()



Algorithm 6 $\texttt{consider\_edge}(x_1, x_2)$

```
I ← insert_edge_SSSP(G_lb, (x1, x2))
Q ← {x ∈ I | cost_T_apx(x) > (1 + ε) · cost_G_lb(x)}
while Q ≠ ∅ do
    x ← Q.top();
    if cost_T_apx(x) > (1 + ε) · cost_G_lb(x) then
        x_parent ← parent_SSSP(G_lb, x)
        if (collision_free (x_parent, x)) then
            T_apx.parent(x) ← x_parent
            Q.pop()
        else
            D ← delete_edge_SSSP(G_lb, (x_parent, x))
            for all y ∈ D ∩ Q do
                Q.update_cost(y)
    else
        Q.pop()
```



Analysis - lower bound invariant

Observations

- A node x is added to $\mathcal{G}_{lb}$ and to $\mathcal{T}_{apx}$ if and only if x is added to $\mathcal{G}_{RRG}$
- Both LBT-RRT and RRG consider the same set of $k_{RRG} \log(|V|)$ nearest neighbors of x_{new}
- Every edge added to the RRG roadmap is added to $\mathcal{G}_{lb}$
- Every edge of $\mathcal{T}_{apx}$ is collision free

Corollary

After every iteration of LBT-RRT the lower bound invariant is maintained

Analysis - lower bound invariant

Observations

- A node x is added to $\mathcal{G}_{lb}$ and to $\mathcal{T}_{apx}$ if and only if x is added to $\mathcal{G}_{RRG}$
- Both LBT-RRT and RRG consider the same set of $k_{RRG} \log(|V|)$ nearest neighbors of x_{new}
- Every edge added to the RRG roadmap is added to $\mathcal{G}_{lb}$
- Every edge of $\mathcal{T}_{apx}$ is collision free

Corollary

After every iteration of LBT-RRT the lower bound invariant is maintained

Analysis - lower bound invariant

Observations

- A node x is added to $\mathcal{G}_{lb}$ and to $\mathcal{T}_{apx}$ if and only if x is added to $\mathcal{G}_{RRG}$
- Both LBT-RRT and RRG consider the same set of $k_{RRG} \log(|V|)$ nearest neighbors of x_{new}
- Every edge added to the RRG roadmap is added to $\mathcal{G}_{lb}$
- Every edge of $\mathcal{T}_{apx}$ is collision free

Corollary

After every iteration of LBT-RRT the lower bound invariant is maintained

Analysis - lower bound invariant

Observations

- A node x is added to $\mathcal{G}_{lb}$ and to $\mathcal{T}_{apx}$ if and only if x is added to $\mathcal{G}_{RRG}$
- Both LBT-RRT and RRG consider the same set of $k_{RRG} \log(|V|)$ nearest neighbors of x_{new}
- Every edge added to the RRG roadmap is added to $\mathcal{G}_{lb}$
- Every edge of $\mathcal{T}_{apx}$ is collision free

Corollary

After every iteration of LBT-RRT the lower bound invariant is maintained

Analysis - lower bound invariant

Observations

- A node x is added to $\mathcal{G}_{lb}$ and to $\mathcal{T}_{apx}$ if and only if x is added to $\mathcal{G}_{RRG}$
- Both LBT-RRT and RRG consider the same set of $k_{RRG} \log(|V|)$ nearest neighbors of x_{new}
- Every edge added to the RRG roadmap is added to $\mathcal{G}_{lb}$
- Every edge of $\mathcal{T}_{apx}$ is collision free

Corollary

After every iteration of LBT-RRT the lower bound invariant is maintained

Analysis - bounded approximation invariant

Observations

- The only place where $\texttt{cost}_{\mathcal{G}_{lb}}$ is decreased is during a call to $\texttt{insert\_edge}_{\texttt{SSSP}}(\mathcal{G}_{lb}, (x_1, x_2))$
- A node x is removed from the queue Q only if the bounded approximation invariant holds for x

Lemma

If the bounded approximation invariant holds prior to a call to the procedure $\texttt{consider\_edge}_{\texttt{SSSP}}(x_1, x_2)$ then the procedure will terminate with the invariant maintained

Prove by induction on the calls to $\texttt{consider\_edge}_{\texttt{SSSP}}(x_1, x_2)$

Theorem

LBT-RRT is asymptotically near-optimal with an approximation factor of $(1 + \varepsilon)$

Analysis - bounded approximation invariant

Observations

- The only place where $\texttt{cost}_{\mathcal{G}_{lb}}$ is decreased is during a call to $\texttt{insert\_edge}_{\texttt{SSSP}}(\mathcal{G}_{lb}, (x_1, x_2))$
- A node x is removed from the queue Q only if the bounded approximation invariant holds for x

Lemma

If the bounded approximation invariant holds prior to a call to the procedure $\texttt{consider\_edge}_{\texttt{SSSP}}(x_1, x_2)$ then the procedure will terminate with the invariant maintained

Prove by induction on the calls to $\texttt{consider\_edge}_{\texttt{SSSP}}(x_1, x_2)$

Theorem

LBT-RRT is asymptotically near-optimal with an approximation factor of $(1 + \varepsilon)$

Analysis - bounded approximation invariant

Observations

- The only place where $\texttt{cost}_{\mathcal{G}_{lb}}$ is decreased is during a call to $\texttt{insert\_edge}_{\texttt{SSSP}}(\mathcal{G}_{lb}, (x_1, x_2))$
- A node x is removed from the queue Q only if the bounded approximation invariant holds for x

Lemma

If the bounded approximation invariant holds prior to a call to the procedure $\texttt{consider\_edge}_{\texttt{SSSP}}(x_1, x_2)$ then the procedure will terminate with the invariant maintained

Prove by induction on the calls to $\texttt{consider\_edge}_{\texttt{SSSP}}(x_1, x_2)$

Theorem

LBT-RRT is asymptotically near-optimal with an approximation factor of $(1 + \varepsilon)$

Analysis - bounded approximation invariant

Observations

- The only place where $\texttt{cost}_{\mathcal{G}_{lb}}$ is decreased is during a call to $\texttt{insert\_edge}_{\texttt{SSSP}}(\mathcal{G}_{lb}, (x_1, x_2))$
- A node x is removed from the queue Q only if the bounded approximation invariant holds for x

Lemma

If the bounded approximation invariant holds prior to a call to the procedure $\texttt{consider\_edge}_{\texttt{SSSP}}(x_1, x_2)$ then the procedure will terminate with the invariant maintained

Prove by induction on the calls to $\texttt{consider\_edge}_{\texttt{SSSP}}(x_1, x_2)$

Theorem

LBT-RRT is asymptotically near-optimal with an approximation factor of $(1 + \varepsilon)$

Simulations

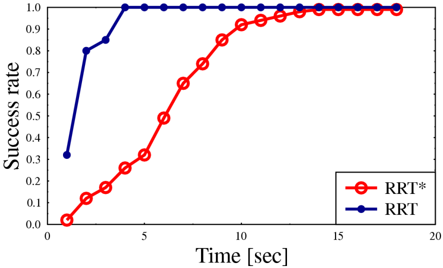
Implemented in OMPL (available in latest release)
Maze scenario - 3 degrees of freedom (DOFs)

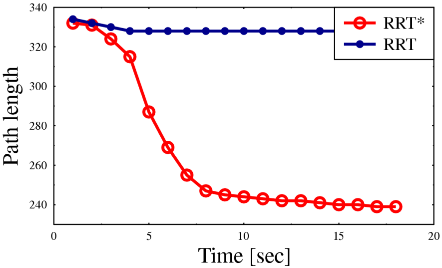
Scenario taken from OMPL

Simulations

Implemented in OMPL (available in latest release)
Maze scenario - 3 degrees of freedom (DOFs)

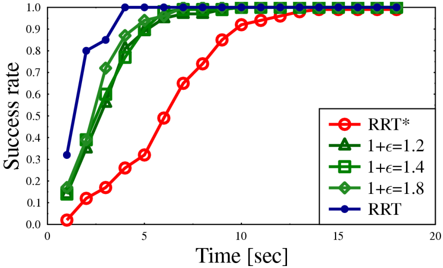
Scenario taken from OMPL

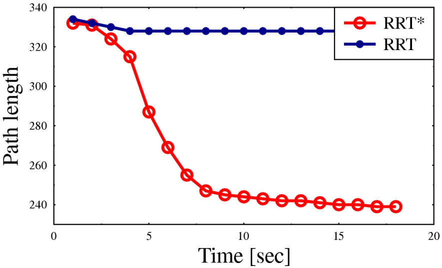




Implemented in OMPL (available in latest release)
Maze scenario - 3 degrees of freedom (DOFs)

Scenario taken from OMPL

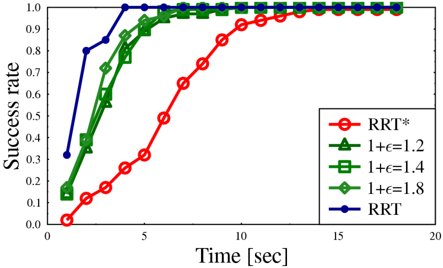


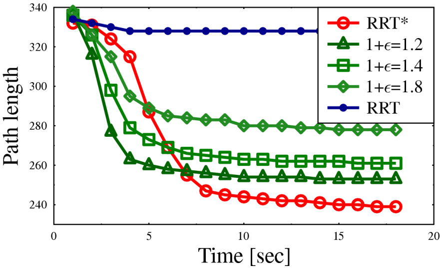


Simulations

Implemented in OMPL (available in latest release)
Maze scenario - 3 degrees of freedom (DOFs)

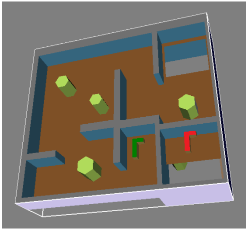
Scenario taken from OMPL





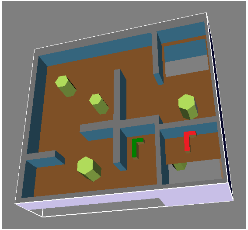
Simulations (cont.)

Implemented in OMPL (available in latest release)
Cubicles scenario (2 robots) - 12 DOFs

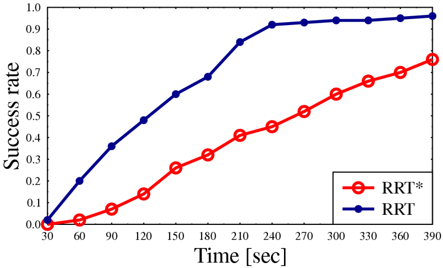


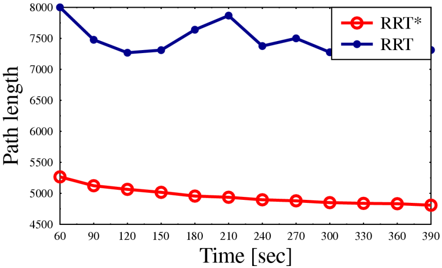
Scenario taken from OMPL

Implemented in OMPL (available in latest release)
Cubicles scenario (2 robots) - 12 DOFs

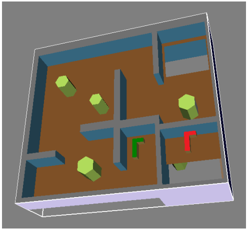


Scenario taken from OMPL

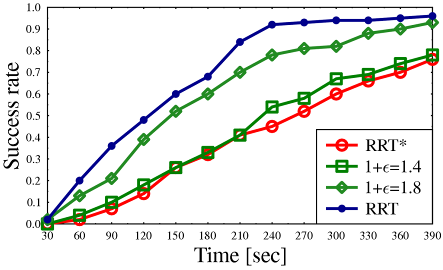


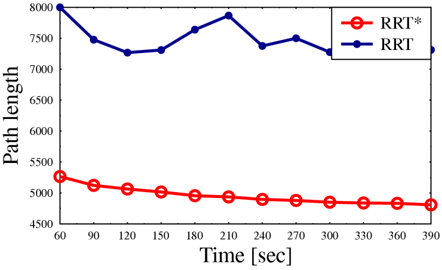


Implemented in OMPL (available in latest release)
Cubicles scenario (2 robots) - 12 DOFs



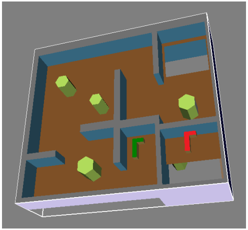
Scenario taken from OMPL



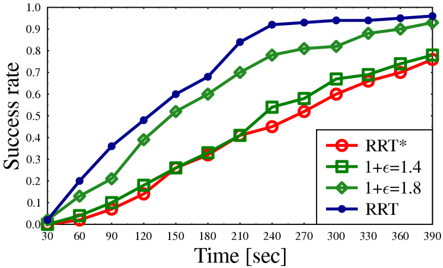


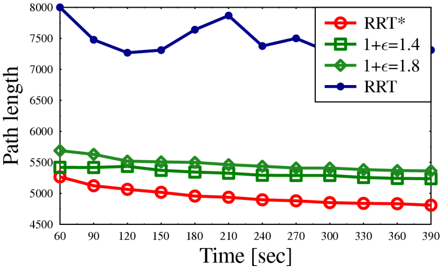
Simulations (cont.)

Implemented in OMPL (available in latest release)
Cubicles scenario (2 robots) - 12 DOFs



Scenario taken from OMPL





Summary (LBT-RRT)

- LBT-RRT continuously interpolates between the fast RRT and the asymptotically optimal RRT*
- The framework may be applied to most variants of RRT or RRT*
  - different sampling heuristics, parallel implementations, planning on implicitly-defined manifolds etc.
- The framework may be applied to different tree based planners such as FMT* [Janson Pavone13]
- Applications
  - Improve quality of RRT
  - Improve convergence rate of RRT*

Summary (LBT-RRT)

- LBT-RRT continuously interpolates between the fast RRT and the asymptotically optimal RRT*
- The framework may be applied to most variants of RRT or RRT*
  - different sampling heuristics, parallel implementations, planning on implicitly-defined manifolds etc.
- The framework may be applied to different tree based planners such as FMT* [Janson Pavone13]
- Applications
  - Improve quality of RRT
  - Improve convergence rate of RRT*

Summary (LBT-RRT)

- LBT-RRT continuously interpolates between the fast RRT and the asymptotically optimal RRT*
- The framework may be applied to most variants of RRT or RRT*
  - different sampling heuristics, parallel implementations, planning on implicitly-defined manifolds etc.
- The framework may be applied to different tree based planners such as FMT* [Janson Pavone13]
- Applications
  - Improve quality of RRT
  - Improve convergence rate of RRT*

Summary (LBT-RRT)

- LBT-RRT continuously interpolates between the fast RRT and the asymptotically optimal RRT*
- The framework may be applied to most variants of RRT or RRT*
  - different sampling heuristics, parallel implementations, planning on implicitly-defined manifolds etc.
- The framework may be applied to different tree based planners such as FMT* [Janson Pavone13]
- Applications
  - Improve quality of RRT
  - Improve convergence rate of RRT*

Thm [LaValle Kuffner01, Kleinbort et al.18]

The RRT algorithm is probabilistically complete.

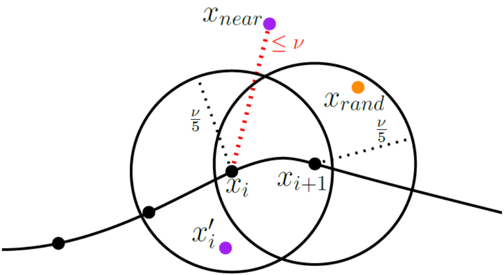


Figure adapted from [Kleinbort et al.18]

Assumptions & notation

- $\exists \pi \subset \mathcal{X}_{\text{free}}$ s.t. $\pi[0] = x_{\text{start}}$ and $\pi[1] = x_{\text{goal}}$
- $\mathcal{X} = [0, 1]^d$ (proof is much more complex for the general case)
- $B_r(x)$—ball of radius r centered at x
- δ—clearance of π
- L—length of π
- η—extend parameter
- $\nu := \min(\delta, \eta)$
- $m := \frac{5L}{\nu}$

Sequence of points—X

Define a sequence of $m+1$ points $X = x_0, \ldots x_m$ such that

- $x_i \in \pi$, $x_0 = \pi[0]$, $x_m = \pi[1]$
- The length of the sub-path between every two consecutive points is $\nu/5$
- Thus, $\forall i, ||x_{i+1} - x_i|| \leq \nu/5$
- Define $S = \{B_\delta(x_i) | 0 < i < m\}$

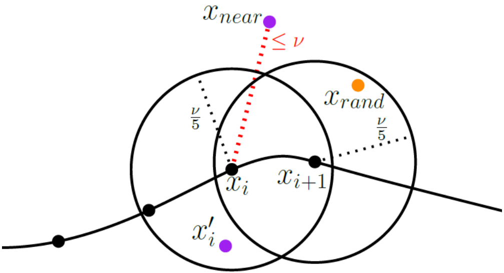


Figure adapted from [Kleinbort et al.18]

Reaching a specific ball

Lemma [Kleinbort et al.18]

Suppose that RRT has reached $B_{\nu/5}(x_i)$, that is, $\exists x_i' \in T \cap B_{\nu/5}(x_i)$. If $x_{\mathrm{rand}} \in B_{\nu/5}(x_{i+1})$, then $\overline{x_{\mathrm{rand}} x_{\mathrm{near}}} \in \mathcal{X}_{\mathrm{free}}$

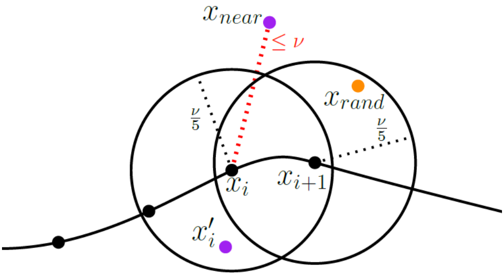


Figure adapted from [Kleinbort et al.18]

Reaching a specific ball—cont.

$$\begin{aligned}
||x_{\text{near}} - x_i|| &\leq ||x_{\text{near}} - x_{\text{rand}} + ||x_{\text{rand}} - x_i|| && \text{/* triangle inequality */} \\
&\leq \underbrace{||x_i' - x_{\text{rand}}||}_{(1)} + \underbrace{||x_{\text{rand}} - x_i||}_{(2)} && \text{/*}x_{\text{near}}\text{ is NN */} \\
&\leq \nu && \text{/* 2 + 3 = 5*/}
\end{aligned}$$

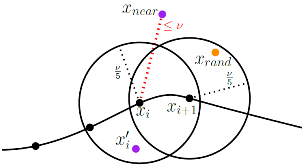


Figure adapted from [Kleinbort et al.18]

(1) $||x_i' - x_{\text{rand}}|| \leq ||x_i' - x_i|| + ||x_i - x_{i+1}|| + ||x_{i+1} - x_{\text{rand}}|| \leq 3 \cdot \frac{\nu}{5}$

(2) $||x_{\text{rand}} - x_i|| \leq ||x_{\text{rand}} - x_{i+1}|| + ||x_{i+1} - x_i|| \leq 2 \cdot \frac{\nu}{5}$

Reaching a specific ball—cont.

$$||x_{near} - x_i|| \leq ||x_{near} - x_{rand} + ||x_{rand} - x_i||$$ /* triangle inequality */

$$\leq \underbrace{||x_i' - x_{rand}||}_{(1)} + \underbrace{||x_{rand} - x_i||}_{(2)}$$ /* x_{near} is NN */

$$\leq \nu$$ /* 2 + 3 = 5*/

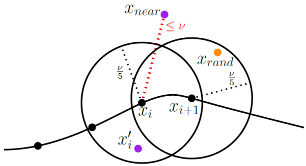


Figure adapted from [Kleinbort et al.18]

(1) $||x_i' - x_{rand}|| \leq ||x_i' - x_i|| + ||x_i - x_{i+1}|| + ||x_{i+1} - x_{rand}|| \leq 3 \cdot \frac{\nu}{5}$

(2) $||x_{rand} - x_i|| \leq ||x_{rand} - x_{i+1}|| + ||x_{i+1} - x_i|| \leq 2 \cdot \frac{\nu}{5}$

$$\begin{aligned}||x_{\text{near}} - x_i|| &\leq ||x_{\text{near}} - x_{\text{rand}} + ||x_{\text{rand}} - x_i|| && \text{/* triangle inequality */}\\ &\leq \underbrace{||x_i' - x_{\text{rand}}||}_{(1)} + \underbrace{||x_{\text{rand}} - x_i||}_{(2)} && \text{/*} x_{\text{near}} \text{ is NN */}\\ &\leq \nu && \text{/* 2 + 3 = 5*/}\end{aligned}$$

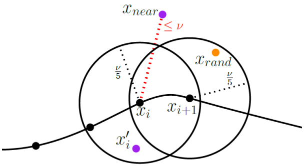


Figure adapted from [Kleinbort et al.18]

(1) $||x_i' - x_{\text{rand}}|| \leq ||x_i' - x_i|| + ||x_i - x_{i+1}|| + ||x_{i+1} - x_{\text{rand}}|| \leq 3 \cdot \frac{\nu}{5}$

(2) $||x_{\text{rand}} - x_i|| \leq ||x_{\text{rand}} - x_{i+1}|| + ||x_{i+1} - x_i|| \leq 2 \cdot \frac{\nu}{5}$

Reaching a specific ball—cont.

$$
\begin{aligned}
||x_{\text{near}} - x_i|| &\leq ||x_{\text{near}} - x_{\text{rand}} + ||x_{\text{rand}} - x_i|| && \text{/* triangle inequality */} \\
&\leq \underbrace{||x_i' - x_{\text{rand}}||}_{(1)} + \underbrace{||x_{\text{rand}} - x_i||}_{(2)} && \text{/*} x_{\text{near}} \text{ is NN */} \\
&\leq \nu && \text{/* 2 + 3 = 5*/}
\end{aligned}
$$

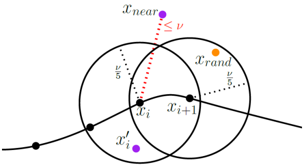


Figure adapted from [Kleinbort et al.18]

(1) $||x_i' - x_{\text{rand}}|| \leq ||x_i' - x_i|| + ||x_i - x_{i+1}|| + ||x_{i+1} - x_{\text{rand}}|| \leq 3 \cdot \frac{\nu}{5}$

(2) $||x_{\text{rand}} - x_i|| \leq ||x_{\text{rand}} - x_{i+1}|| + ||x_{i+1} - x_i|| \leq 2 \cdot \frac{\nu}{5}$

Reaching a specific ball—cont.

- We showed that $||x_{\mathrm{near}} - x_i|| \leq \nu$
- Thus $x_{\mathrm{near}}, x_{\mathrm{rand}} \in B_\nu(x_i)$
- As $\nu \leq \delta$, we have that $\overline{x_{\mathrm{rand}} x_{\mathrm{near}}} \in \mathcal{X}_{\mathrm{free}}$

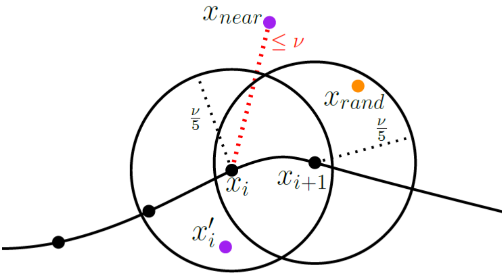


Figure adapted from [Kleinbort et al.18]

Prob. completeness proof

- Assume that $B_{\nu/5}(x_i)$ contains an `RRT` vertex
- The prob. p of sampling in $B_{\nu/5}(x_{i+1})$ is $|B_{\nu/5}|/|[0,1]^d| = |B_{\nu/5}|$
- If `RRT` successfully moves from one ball to the next m times, it will find a path

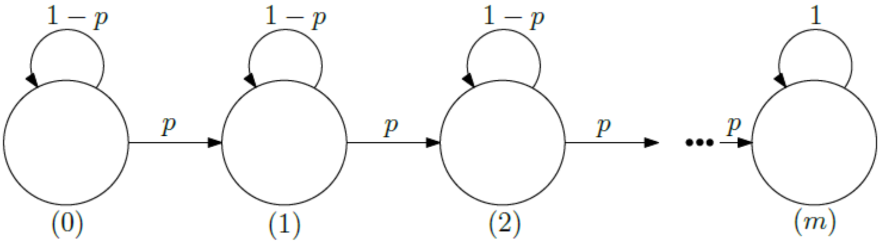


Figure adapted from [Kleinbort et al.18]

Prob. completeness proof (cont.)

- X_n—the number of successes after n samples
- We want to show that $\lim_{n\to\infty} \Pr[X_n < m] = 0$

$$
\begin{aligned}
\Pr[X_n < m] &= \sum_{i=0}^{m-1} \binom{n}{i} p^i (1-p)^{n-i} \\
&\leq \sum_{i=0}^{m-1} \binom{n}{m-1} p^i (1-p)^{n-i} && /^* m \ll n ^*/ \\
&\leq \binom{n}{m-1} \sum_{i=0}^{m-1} (1-p)^n && /^* p \leq 1/2 ^*/ \\
&\leq \binom{n}{m-1} \sum_{i=0}^{m-1} \left(e^{-p}\right)^n && /^* (1-p) \leq e^{-p} ^*/ \\
&= \binom{n}{m-1} m \cdot \left(e^{-p}\right)^n \\
&\leq \frac{m}{(m-1)!} n^m e^{-pn} && /^* \binom{a}{b} \leq a^b/b! ^*/
\end{aligned}
$$

Prob. completeness proof (cont.)

- X_n—the number of successes after n samples
- We want to show that $\lim_{n\to\infty} \Pr[X_n < m] = 0$

$$
\begin{aligned}
\Pr[X_n < m] &= \sum_{i=0}^{m-1} \binom{n}{i} p^i (1-p)^{n-i} \\
&\leq \sum_{i=0}^{m-1} \binom{n}{m-1} p^i (1-p)^{n-i} && /^* m \ll n ^*/ \\
&\leq \binom{n}{m-1} \sum_{i=0}^{m-1} (1-p)^n && /^* p \leq 1/2 ^*/ \\
&\leq \binom{n}{m-1} \sum_{i=0}^{m-1} \left(e^{-p}\right)^n && /^*(1-p) \leq e^{-p} {}^*/ \\
&= \binom{n}{m-1} m \cdot \left(e^{-p}\right)^n \\
&\leq \frac{m}{(m-1)!} n^m e^{-pn} && /^* \binom{a}{b} \leq a^b / b! ^*/
\end{aligned}
$$

Prob. completeness proof (cont.)

- X_n—the number of successes after n samples
- We want to show that $\lim_{n\to\infty} \Pr[X_n < m] = 0$

$$
\begin{aligned}
\Pr[X_n < m] &= \sum_{i=0}^{m-1} \binom{n}{i} p^i (1-p)^{n-i} \\
&\leq \sum_{i=0}^{m-1} \binom{n}{m-1} p^i (1-p)^{n-i} && /^{*} m \ll n^{*}/ \\
&\leq \binom{n}{m-1} \sum_{i=0}^{m-1} (1-p)^n && /^{*} p \leq 1/2^{*}/ \\
&\leq \binom{n}{m-1} \sum_{i=0}^{m-1} \left(e^{-p}\right)^n && /^{*}(1-p) \leq e^{-p*}/ \\
&= \binom{n}{m-1} m \cdot \left(e^{-p}\right)^n \\
&\leq \frac{m}{(m-1)!} n^m e^{-pn} && /^{*} \binom{a}{b} \leq a^b / b!^{*}/
\end{aligned}
$$

Prob. completeness proof (cont.)

- X_n—the number of successes after n samples
- We want to show that $\lim_{n\to\infty} \Pr[X_n < m] = 0$

$$\begin{aligned}
\Pr[X_n < m] &= \sum_{i=0}^{m-1} \binom{n}{i} p^i (1-p)^{n-i} \\
&\leq \sum_{i=0}^{m-1} \binom{n}{m-1} p^i (1-p)^{n-i} && /^* m \ll n ^*/ \\
&\leq \binom{n}{m-1} \sum_{i=0}^{m-1} (1-p)^n && /^* p \leq 1/2 ^*/ \\
&\leq \binom{n}{m-1} \sum_{i=0}^{m-1} \left(e^{-p}\right)^n && /^* (1-p) \leq e^{-p} ^*/ \\
&= \binom{n}{m-1} m \cdot \left(e^{-p}\right)^n \\
&\leq \frac{m}{(m-1)!} n^m e^{-pn} && /^* \binom{a}{b} \leq a^b/b! ^*/
\end{aligned}$$

Prob. completeness proof (cont.)

- X_n—the number of successes after n samples
- We want to show that $\lim_{n\to\infty} \Pr[X_n < m] = 0$

$$
\begin{aligned}
\Pr[X_n < m] &= \sum_{i=0}^{m-1} \binom{n}{i} p^i (1-p)^{n-i} \\
&\leq \sum_{i=0}^{m-1} \binom{n}{m-1} p^i (1-p)^{n-i} && /^* m \ll n ^*/ \\
&\leq \binom{n}{m-1} \sum_{i=0}^{m-1} (1-p)^n && /^* p \leq 1/2 ^*/ \\
&\leq \binom{n}{m-1} \sum_{i=0}^{m-1} \left(e^{-p}\right)^n && /^* (1-p) \leq e^{-p} {}^*/ \\
&= \binom{n}{m-1} m \cdot \left(e^{-p}\right)^n \\
&\leq \frac{m}{(m-1)!} n^m e^{-pn} && /^* \binom{a}{b} \leq a^b / b! ^*/
\end{aligned}
$$

Prob. completeness proof (cont.)

- X_n—the number of successes after n samples
- We want to show that $\lim_{n\to\infty} \Pr[X_n < m] = 0$

$$
\begin{aligned}
\Pr[X_n < m] &= \sum_{i=0}^{m-1} \binom{n}{i} p^i (1-p)^{n-i} \\
&\leq \sum_{i=0}^{m-1} \binom{n}{m-1} p^i (1-p)^{n-i} && /^* m \ll n ^*/ \\
&\leq \binom{n}{m-1} \sum_{i=0}^{m-1} (1-p)^n && /^* p \leq 1/2 ^*/ \\
&\leq \binom{n}{m-1} \sum_{i=0}^{m-1} \left(e^{-p}\right)^n && /^* (1-p) \leq e^{-p} {}^*/ \\
&= \binom{n}{m-1} m \cdot \left(e^{-p}\right)^n \\
&\leq \frac{m}{(m-1)!} n^m e^{-pn} && /^* \binom{a}{b} \leq a^b / b! ^*/
\end{aligned}
$$

Prob. completeness proof (cont.)

- X_n—the number of successes after n samples
- We want to show that $\lim_{n\to\infty} \Pr[X_n < m] = 0$

$$\begin{aligned}
\Pr[X_n < m] &= \sum_{i=0}^{m-1} \binom{n}{i} p^i (1-p)^{n-i} \\
&\leq \sum_{i=0}^{m-1} \binom{n}{m-1} p^i (1-p)^{n-i} && /^* m \ll n ^*/ \\
&\leq \binom{n}{m-1} \sum_{i=0}^{m-1} (1-p)^n && /^* p \leq 1/2 ^*/ \\
&\leq \binom{n}{m-1} \sum_{i=0}^{m-1} \left(e^{-p}\right)^n && /^*(1-p) \leq e^{-p} {}^*/ \\
&= \binom{n}{m-1} m \cdot \left(e^{-p}\right)^n \\
&\leq \frac{m}{(m-1)!} n^m e^{-pn} && /^* \binom{a}{b} \leq a^b / b! ^*/
\end{aligned}$$

Prob. completeness proof (cont.)

- X_n—the number of successes after n samples
- We want to show that $\lim_{n\to\infty} \Pr[X_n < m] = 0$

$$
\begin{aligned}
\Pr[X_n < m] &= \sum_{i=0}^{m-1} \binom{n}{i} p^i (1-p)^{n-i} \\
&\leq \sum_{i=0}^{m-1} \binom{n}{m-1} p^i (1-p)^{n-i} && /^* m \ll n ^*/ \\
&\leq \binom{n}{m-1} \sum_{i=0}^{m-1} (1-p)^n && /^* p \leq 1/2 ^*/ \\
&\leq \binom{n}{m-1} \sum_{i=0}^{m-1} \left(e^{-p}\right)^n && /^*(1-p) \leq e^{-p *}/ \\
&= \binom{n}{m-1} m \cdot \left(e^{-p}\right)^n \\
&\leq \frac{m}{(m-1)!} n^m e^{-pn} && /^* \binom{a}{b} \leq a^b / b!^*/
\end{aligned}
$$

Prob. completeness proof (cont.)

- X_n—the number of successes after n samples
- We want to show that $\lim_{n\to\infty} \Pr[X_n < m] = 0$
- We just showed that $\Pr[X_n < m] \leq \frac{m}{(m-1)!} n^m e^{-pn}$

As p, m are fixed

$$\lim_{n\to\infty} \Pr[X_n < m] \leq \lim_{n\to\infty} \frac{m}{(m-1)!} n^m e^{-pn} = 0$$

Prob. completeness proof (cont.)

- X_n—the number of successes after n samples
- We want to show that $\lim_{n\to\infty} \Pr[X_n < m] = 0$
- We just showed that $\Pr[X_n < m] \le \frac{m}{(m-1)!} n^m e^{-pn}$

As p, m are fixed

$$\lim_{n\to\infty} \Pr[X_n < m] \le \lim_{n\to\infty} \frac{m}{(m-1)!} n^m e^{-pn} = 0$$