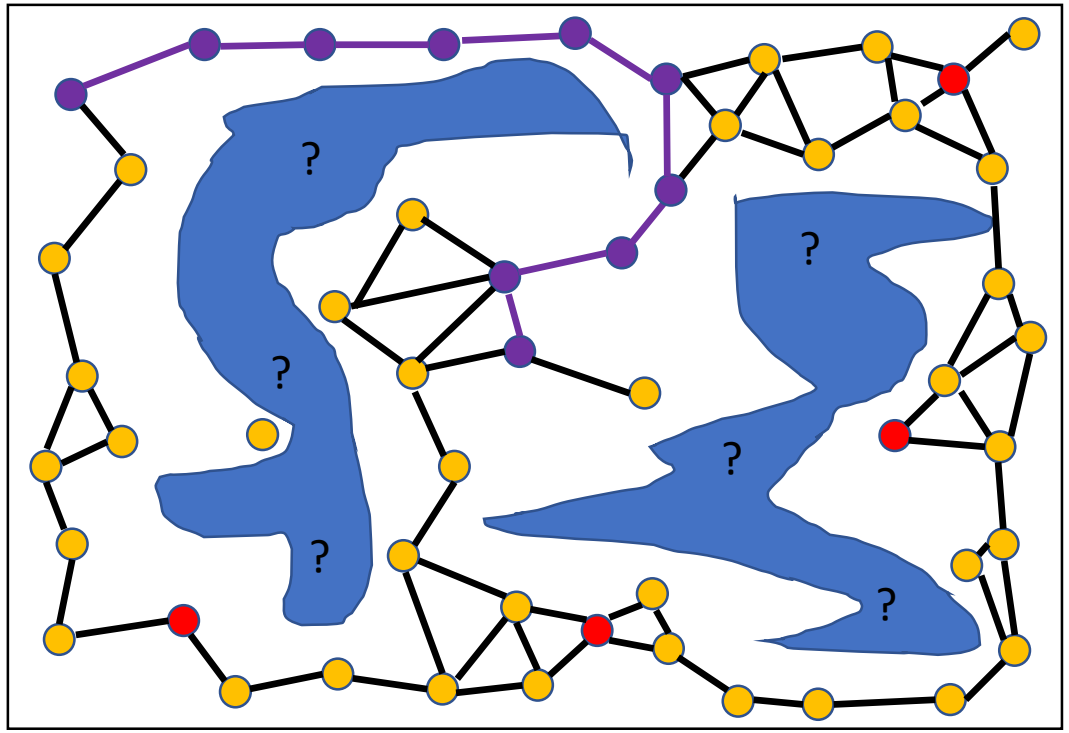
The single-query motion planners

Algorithms and Data Structures 2 – Motion Planning and its applications

University of Applied Sciences Stuttgart

Dr. Daniel Schneider

Motion Planning Algorithms



The **sPRM** is not the only motion planning algorithm.

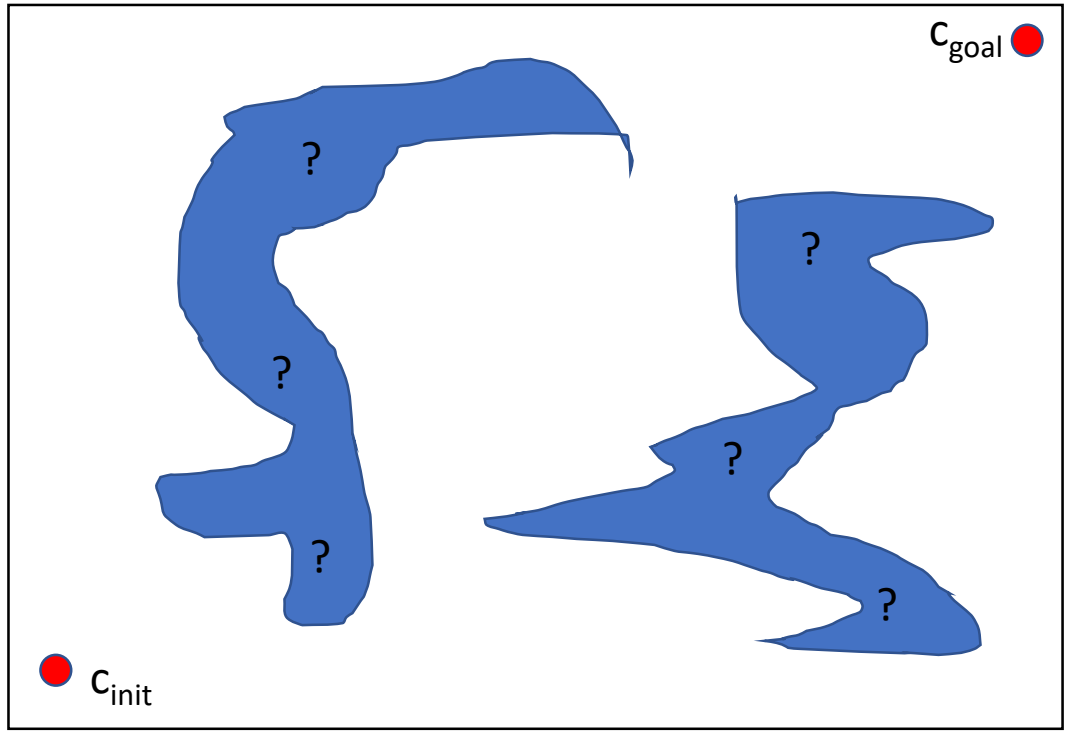
- The roadmap-based motion planner are used for multi-query motion planning.
- If you only need to compute a single path then there are better approaches.

→ These will be covered in this lection.

The Rapidly-Exploring Random Tree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

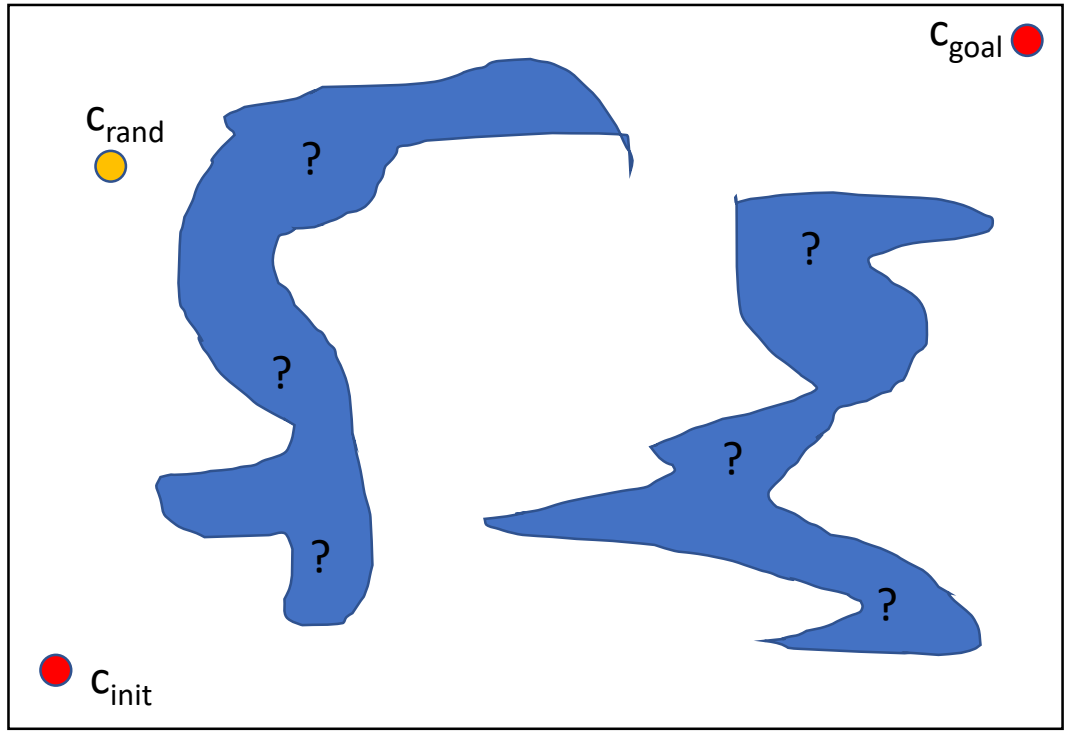
```
T.init(c_init)                                                    1
while (! TimeElapsed(T_max)) do                                   2
    c_rand ← RandomState()                                        3
    c_near ← NearestN(c_rand, T)                                  4
    c_new ← getValidEdge(c_near, c_rand, range_max)               5
    if (! Trapped) then                                           6
        T.addVertex(c_new)                                        7
        T.addEdge(c_near, c_new)                                  8
        if goalReached(c_new, c_goal) then                        9
            return Path(T)                                        10
return ApproxPath(T)                                              11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

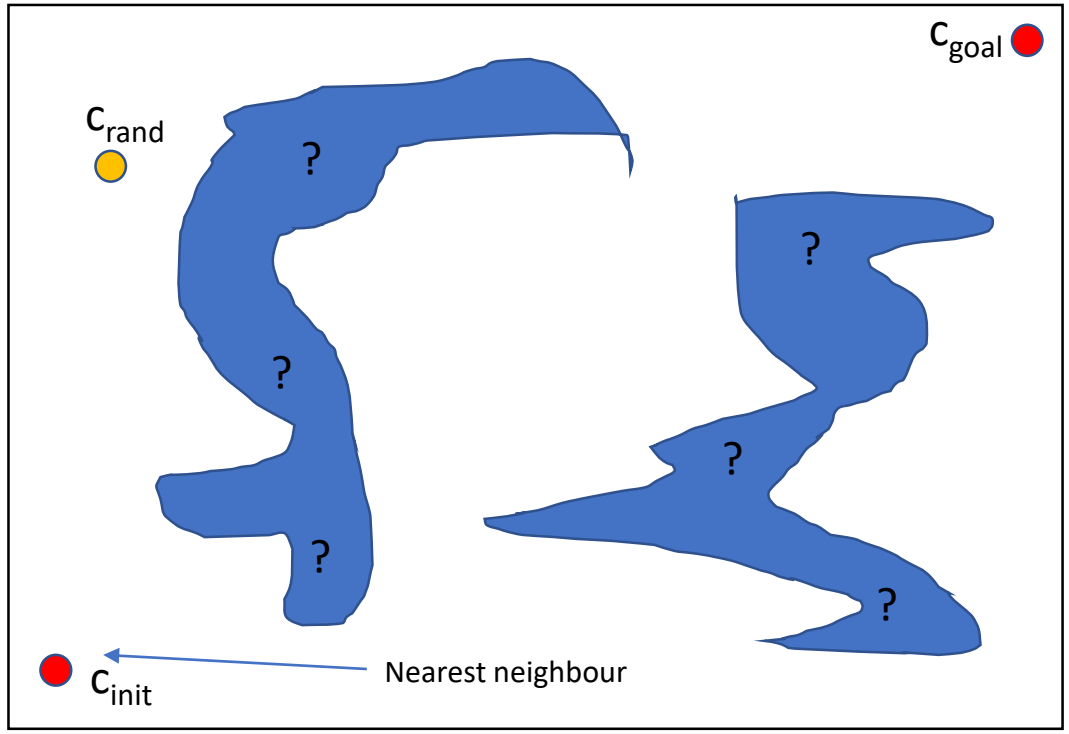
```
T.init(c_init)                                                    1
while (! TimeElapsed(T_max)) do                                   2
    c_rand ← RandomState()                                        3
    c_near ← NearestN(c_rand, T)                                  4
    c_new ← getValidEdge(c_near, c_rand, range_max)               5
    if (! Trapped) then                                           6
        T.addVertex(c_new)                                        7
        T.addEdge(c_near, c_new)                                  8
        if goalReached(c_new, c_goal) then                        9
            return Path(T)                                        10
return ApproxPath(T)                                              11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

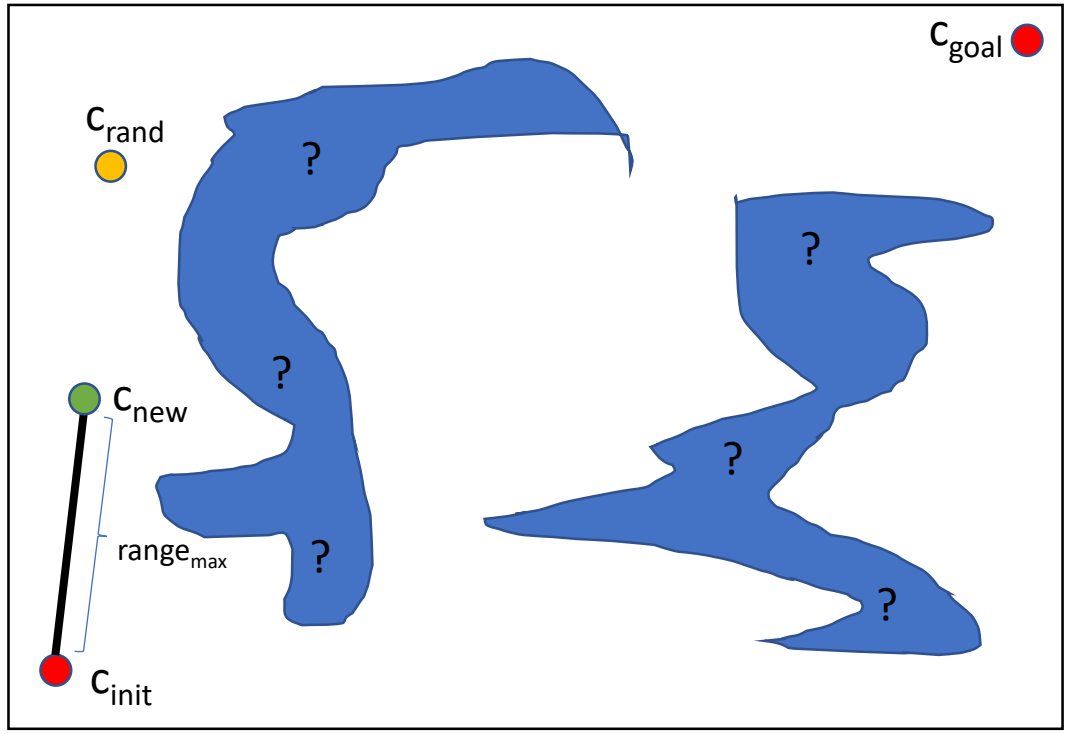
```
T.init(c_init)                                                    1
while (! TimeElapsed(T_max)) do                                   2
    c_rand ← RandomState()                                        3
    c_near ← NearestN(c_rand, T)                                  4
    c_new ← getValidEdge(c_near, c_rand, range_max)               5
    if (! Trapped) then                                           6
        T.addVertex(c_new)                                        7
        T.addEdge(c_near, c_new)                                  8
        if goalReached(c_new, c_goal) then                        9
            return Path(T)                                        10
return ApproxPath(T)                                              11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

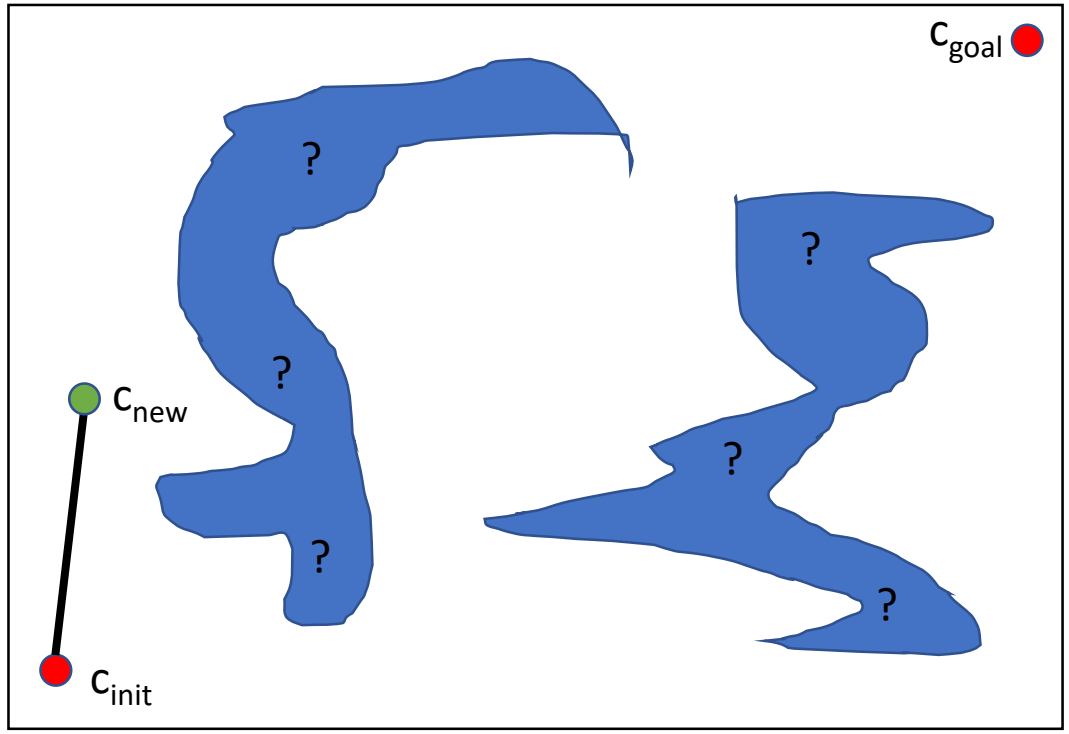
```
T.init(c_init)                                                          1
while (! TimeElapsed(T_max)) do                                         2
    c_rand ← RandomState()                                              3
    c_near ← NearestN(c_rand, T)                                        4
    c_new ← getValidEdge(c_near, c_rand, range_max)                     5
    if (! Trapped) then                                                 6
        T.addVertex(c_new)                                              7
        T.addEdge(c_near, c_new)                                        8
        if goalReached(c_new, c_goal) then                              9
            return Path(T)                                              10
return ApproxPath(T)                                                    11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

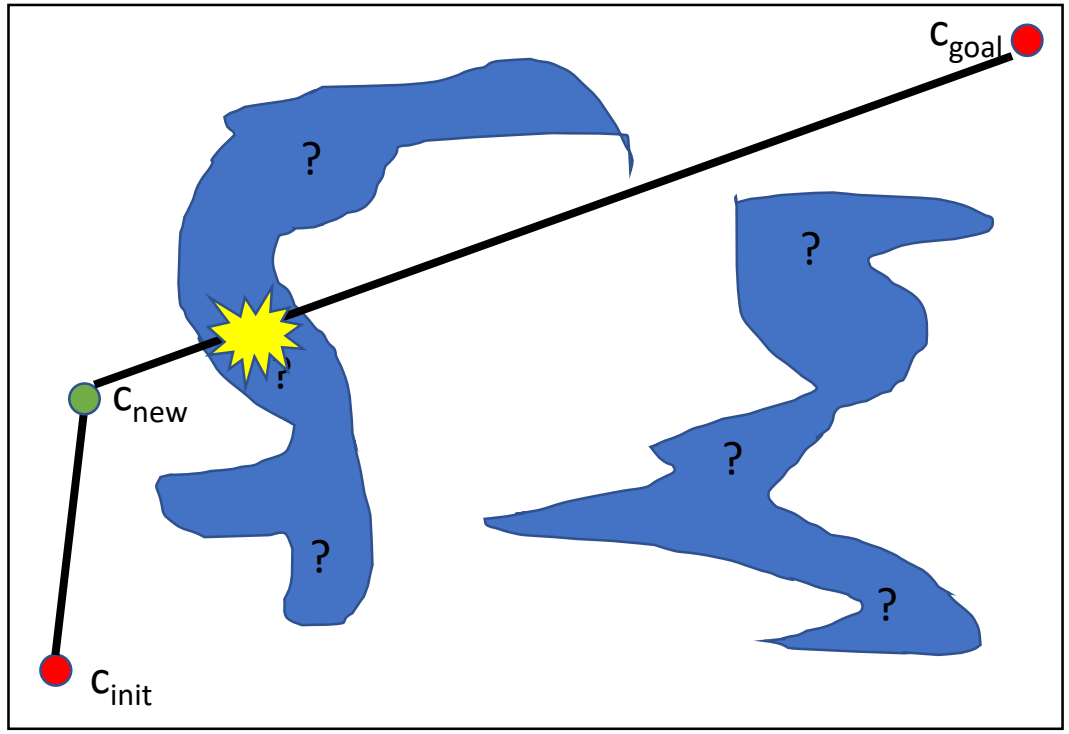
```
T.init(c_init)                                                          1
while (! TimeElapsed(T_max)) do                                         2
    c_rand ← RandomState()                                              3
    c_near ← NearestN(c_rand, T)                                        4
    c_new ← getValidEdge(c_near, c_rand, range_max)                     5
    if (! Trapped) then                                                 6
        T.addVertex(c_new)                                              7
        T.addEdge(c_near, c_new)                                        8
        if goalReached(c_new, c_goal) then                              9
            return Path(T)                                              10
return ApproxPath(T)                                                    11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

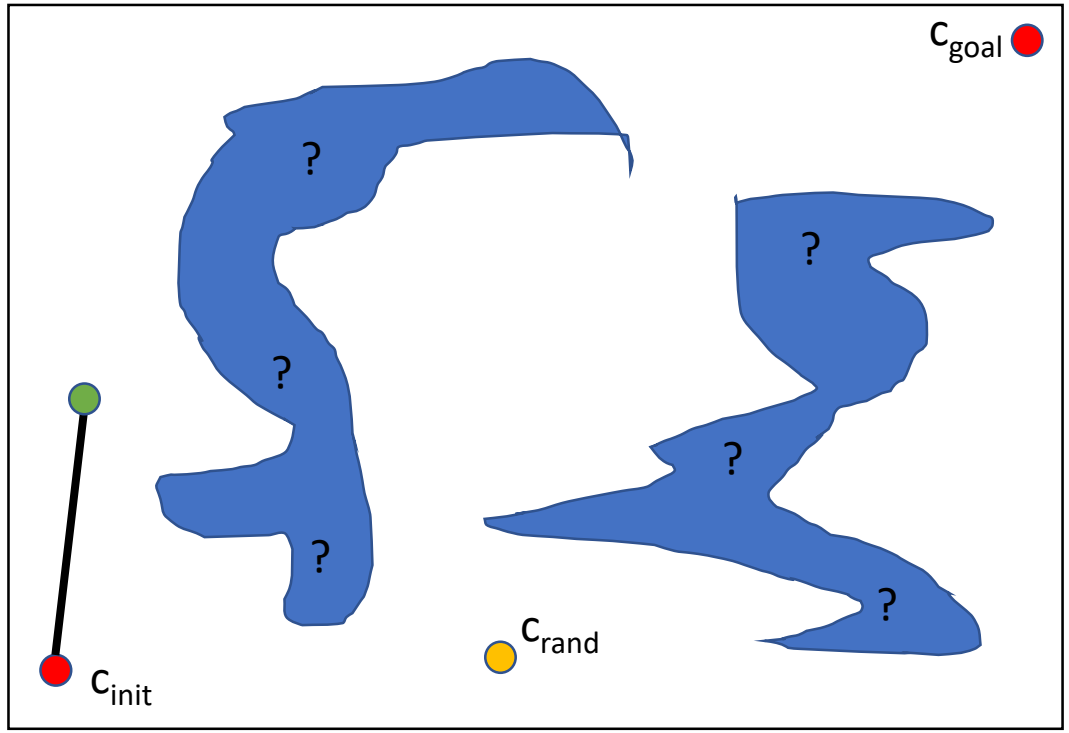
```
T.init(c_init)                                                    1
while (! TimeElapsed(T_max)) do                                   2
    c_rand ← RandomState()                                        3
    c_near ← NearestN(c_rand, T)                                  4
    c_new ← getValidEdge(c_near, c_rand, range_max)               5
    if (! Trapped) then                                           6
        T.addVertex(c_new)                                        7
        T.addEdge(c_near, c_new)                                  8
        if goalReached(c_new,c_goal) then                         9
            return Path(T)                                        10
return ApproxPath(T)                                              11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

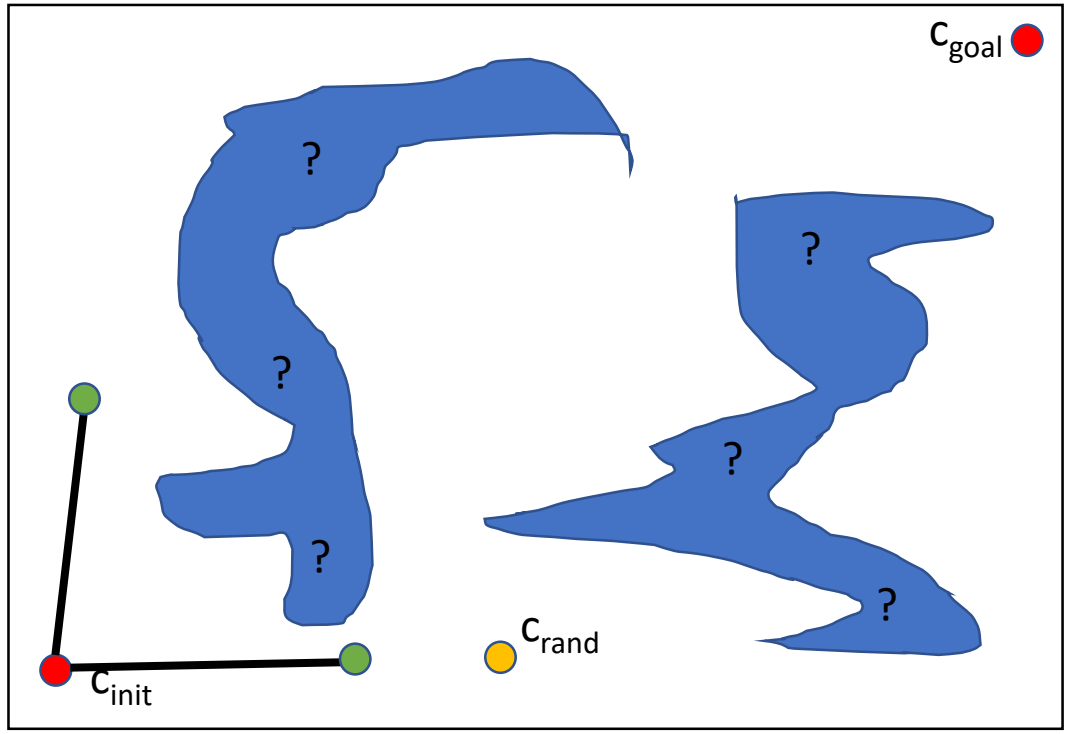
```
T.init(c_init)                                                    1
while (! TimeElapsed(T_max)) do                                   2
    c_rand ← RandomState()                                        3
    c_near ← NearestN(c_rand, T)                                  4
    c_new ← getValidEdge(c_near, c_rand, range_max)               5
    if (! Trapped) then                                           6
        T.addVertex(c_new)                                        7
        T.addEdge(c_near, c_new)                                  8
        if goalReached(c_new,c_goal) then                         9
            return Path(T)                                        10
return ApproxPath(T)                                              11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

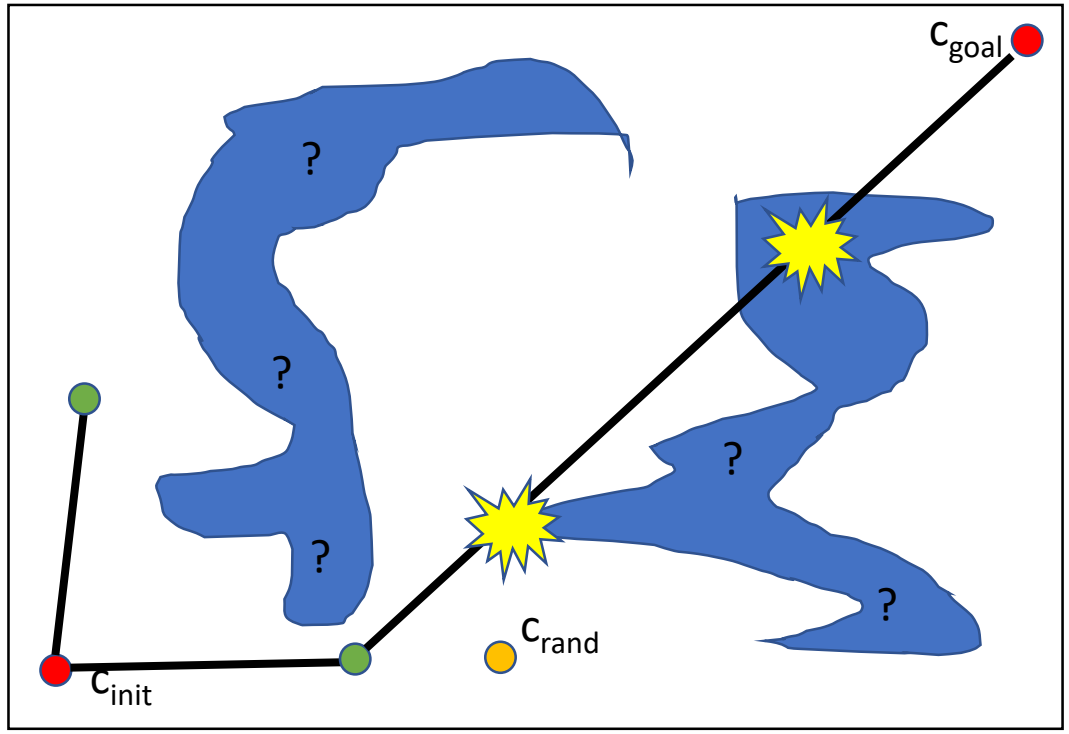
```
T.init(c_init)                                                    1
while (! TimeElapsed(T_max)) do                                   2
    c_rand ← RandomState()                                        3
    c_near ← NearestN(c_rand, T)                                  4
    c_new ← getValidEdge(c_near, c_rand, range_max)               5
    if (! Trapped) then                                           6
        T.addVertex(c_new)                                        7
        T.addEdge(c_near, c_new)                                  8
        if goalReached(c_new, c_goal) then                        9
            return Path(T)                                        10
return ApproxPath(T)                                              11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT (c_{init}, c_{goal}, $range_{max}$, $time_{max}$)

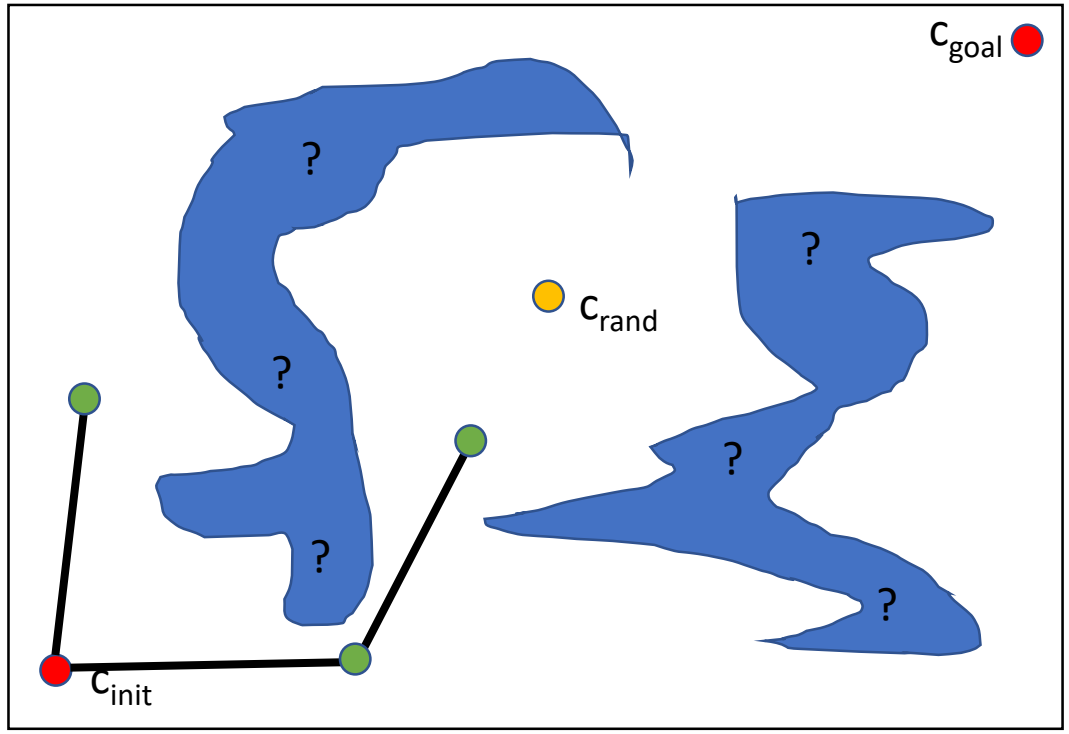
```
T.init(c_init)                                                    1
while (! TimeElapsed(T_max)) do                                   2
    c_rand ← RandomState()                                        3
    c_near ← NearestN(c_rand, T)                                  4
    c_new ← getValidEdge(c_near, c_rand, range_max)               5
    if (! Trapped) then                                           6
        T.addVertex(c_new)                                        7
        T.addEdge(c_near, c_new)                                  8
        if goalReached(c_new,c_goal) then                         9
            return Path(T)                                        10
return ApproxPath(T)                                              11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

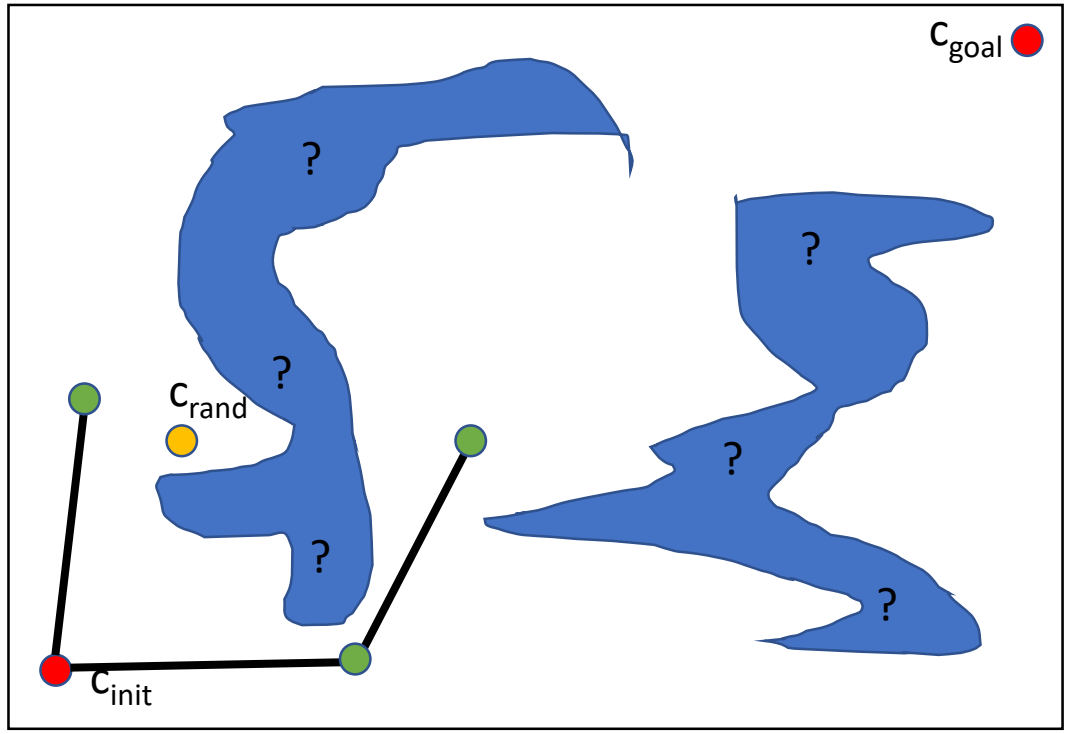
```
T.init(c_init)                                                          1
while (! TimeElapsed(T_max)) do                                         2
    c_rand ← RandomState()                                              3
    c_near ← NearestN(c_rand, T)                                        4
    c_new ← getValidEdge(c_near, c_rand, range_max)                     5
    if (! Trapped) then                                                 6
        T.addVertex(c_new)                                              7
        T.addEdge(c_near, c_new)                                        8
        if goalReached(c_new, c_goal) then                              9
            return Path(T)                                              10
return ApproxPath(T)                                                    11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

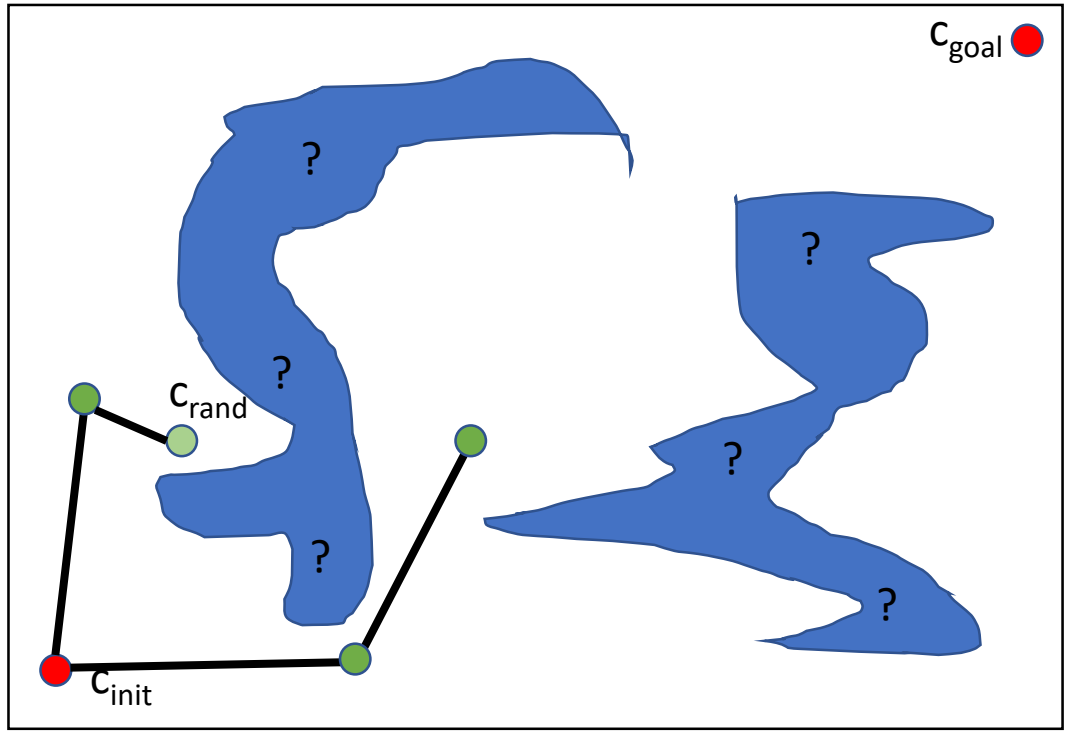
```
T.init(c_init)                                                    1
while (! TimeElapsed(T_max)) do                                   2
    c_rand ← RandomState()                                        3
    c_near ← NearestN(c_rand, T)                                  4
    c_new ← getValidEdge(c_near, c_rand, range_max)               5
    if (! Trapped) then                                           6
        T.addVertex(c_new)                                        7
        T.addEdge(c_near, c_new)                                  8
        if goalReached(c_new, c_goal) then                        9
            return Path(T)                                        10
return ApproxPath(T)                                              11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

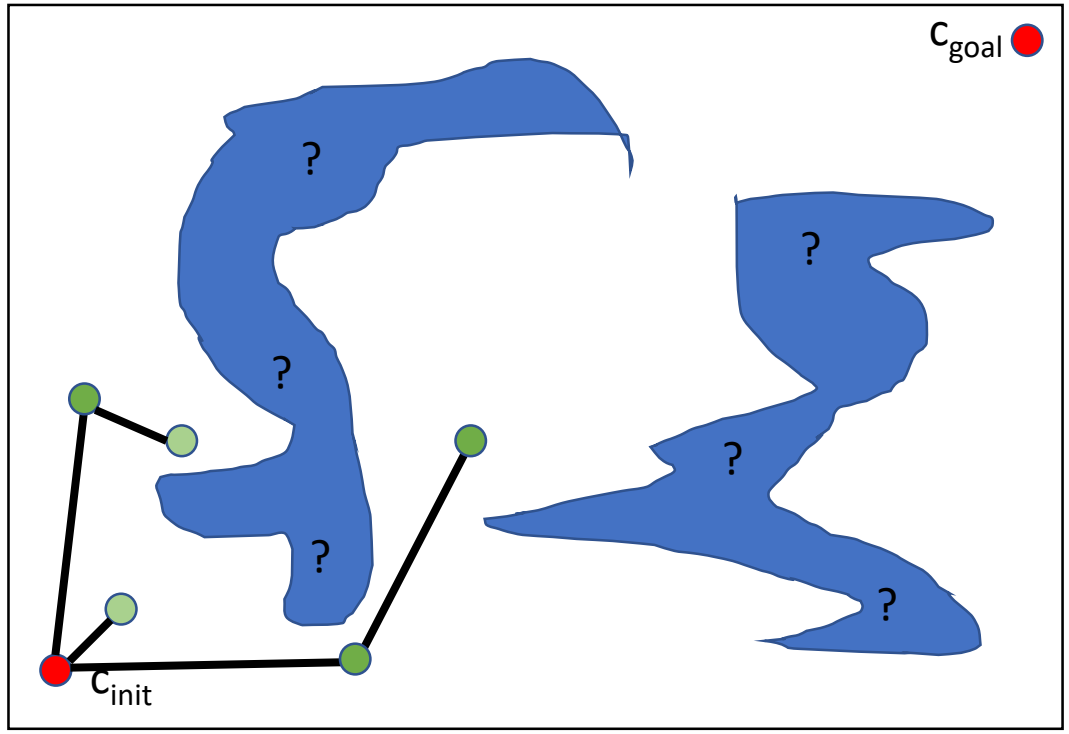
```
T.init(c_init)                                                          1
while (! TimeElapsed(T_max)) do                                         2
    c_rand ← RandomState()                                              3
    c_near ← NearestN(c_rand, T)                                        4
    c_new ← getValidEdge(c_near, c_rand, range_max)                     5
    if (! Trapped) then                                                 6
        T.addVertex(c_new)                                              7
        T.addEdge(c_near, c_new)                                        8
        if goalReached(c_new, c_goal) then                              9
            return Path(T)                                              10
return ApproxPath(T)                                                    11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

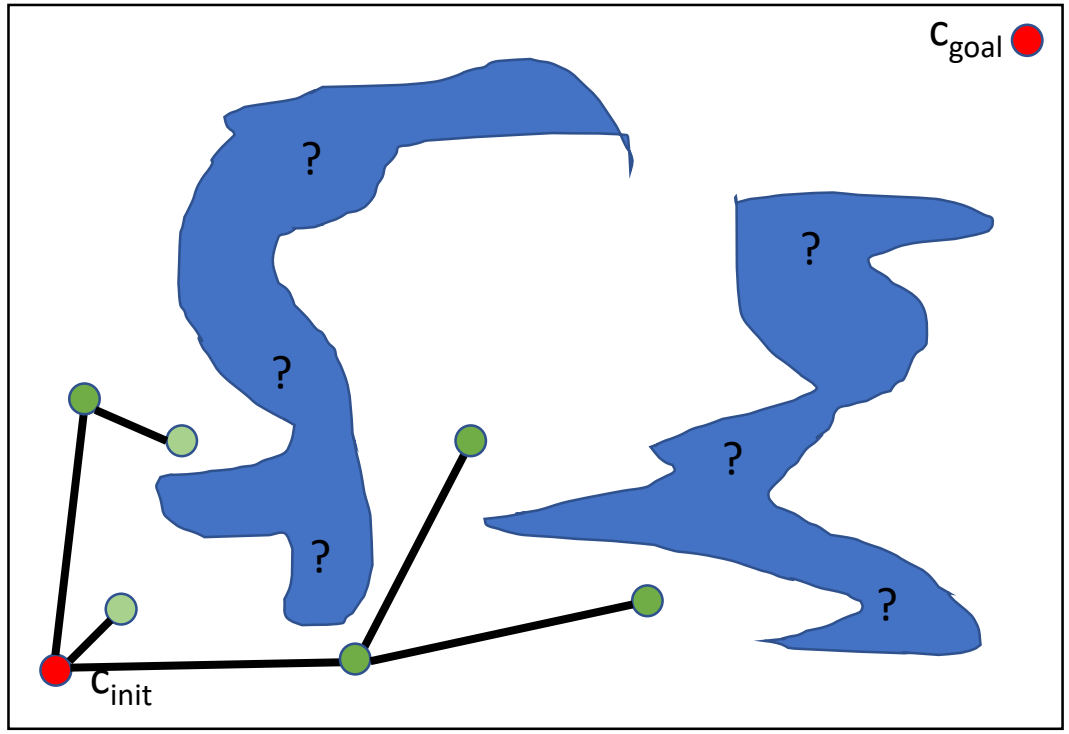
```
T.init(c_init)                                                    1
while (! TimeElapsed(T_max)) do                                   2
    c_rand ← RandomState()                                        3
    c_near ← NearestN(c_rand, T)                                  4
    c_new ← getValidEdge(c_near, c_rand, range_max)               5
    if (! Trapped) then                                           6
        T.addVertex(c_new)                                        7
        T.addEdge(c_near, c_new)                                  8
        if goalReached(c_new, c_goal) then                        9
            return Path(T)                                        10
return ApproxPath(T)                                              11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

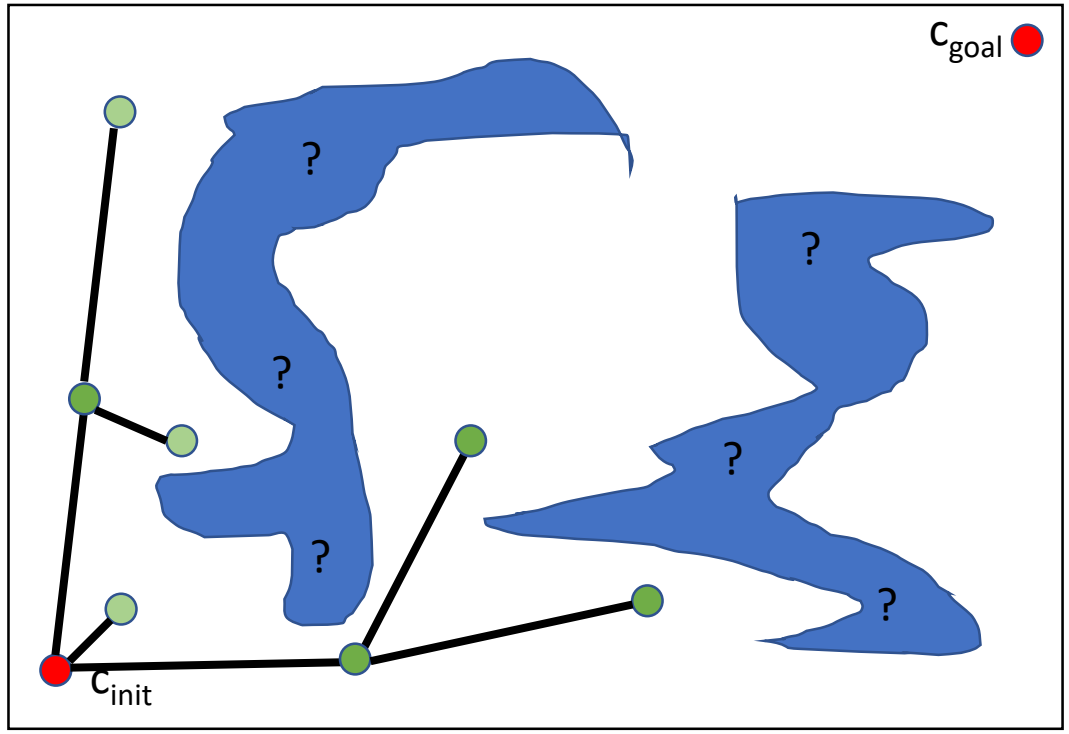
```
T.init(c_init)                                                    1
while (! TimeElapsed(T_max)) do                                   2
    c_rand ← RandomState()                                        3
    c_near ← NearestN(c_rand, T)                                  4
    c_new ← getValidEdge(c_near, c_rand, range_max)               5
    if (! Trapped) then                                           6
        T.addVertex(c_new)                                        7
        T.addEdge(c_near, c_new)                                  8
        if goalReached(c_new, c_goal) then                        9
            return Path(T)                                        10
return ApproxPath(T)                                              11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

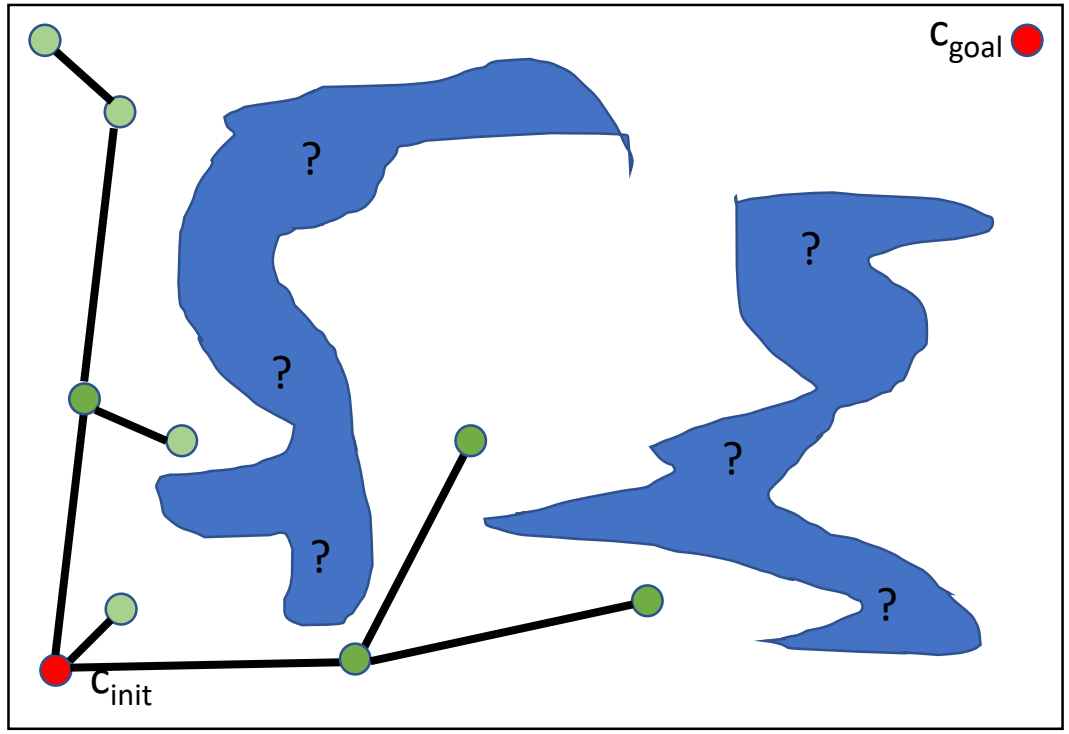
```
T.init(c_init)                                                    1
while (! TimeElapsed(T_max)) do                                   2
    c_rand ← RandomState()                                        3
    c_near ← NearestN(c_rand, T)                                  4
    c_new ← getValidEdge(c_near, c_rand, range_max)               5
    if (! Trapped) then                                           6
        T.addVertex(c_new)                                        7
        T.addEdge(c_near, c_new)                                  8
        if goalReached(c_new, c_goal) then                        9
            return Path(T)                                        10
return ApproxPath(T)                                              11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

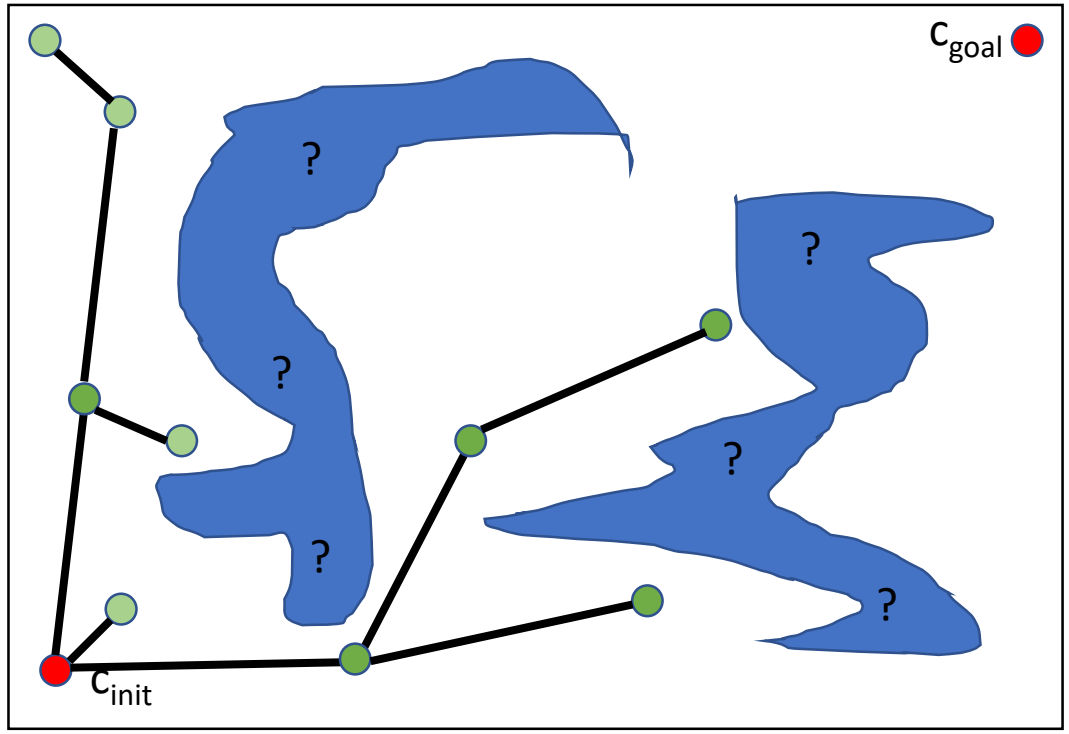
```
T.init(c_init)                                                    1
while (! TimeElapsed(T_max)) do                                   2
    c_rand ← RandomState()                                        3
    c_near ← NearestN(c_rand, T)                                  4
    c_new ← getValidEdge(c_near, c_rand, range_max)               5
    if (! Trapped) then                                           6
        T.addVertex(c_new)                                        7
        T.addEdge(c_near, c_new)                                  8
        if goalReached(c_new, c_goal) then                        9
            return Path(T)                                        10
return ApproxPath(T)                                              11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

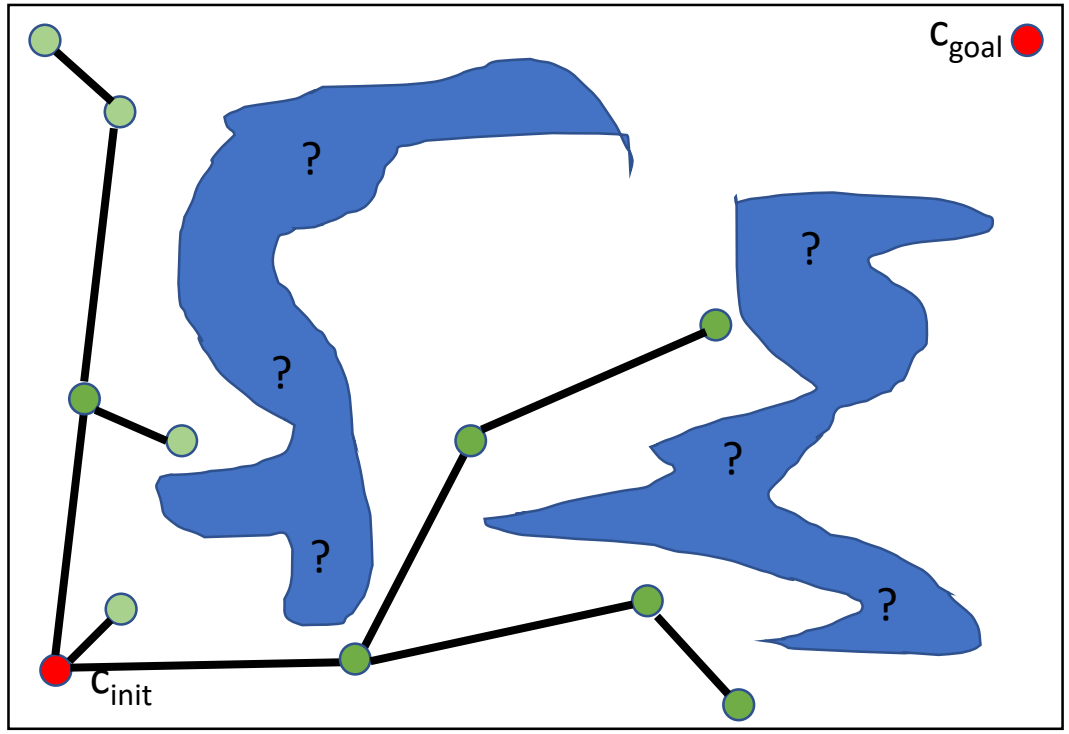
```
T.init(c_init)                                                    1
while (! TimeElapsed(T_max)) do                                   2
    c_rand ← RandomState()                                        3
    c_near ← NearestN(c_rand, T)                                  4
    c_new ← getValidEdge(c_near, c_rand, range_max)               5
    if (! Trapped) then                                           6
        T.addVertex(c_new)                                        7
        T.addEdge(c_near, c_new)                                  8
        if goalReached(c_new, c_goal) then                        9
            return Path(T)                                        10
return ApproxPath(T)                                              11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

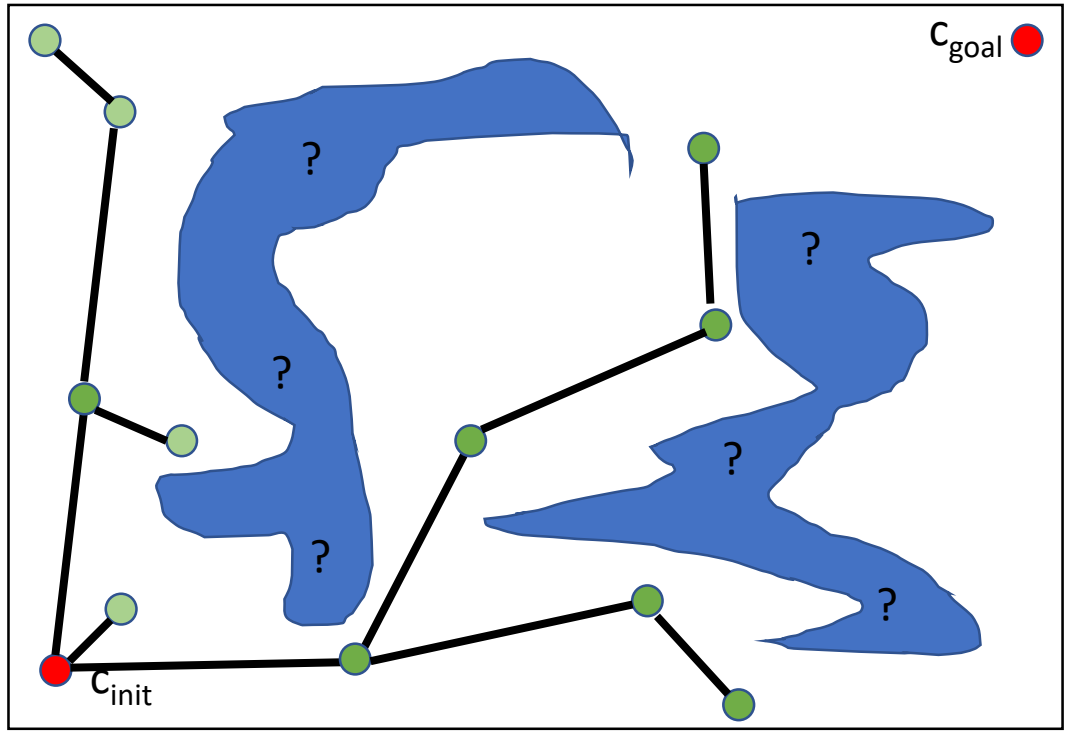
```
T.init(c_init)                                                          1
while (! TimeElapsed(T_max)) do                                         2
    c_rand ← RandomState()                                              3
    c_near ← NearestN(c_rand, T)                                        4
    c_new ← getValidEdge(c_near, c_rand, range_max)                     5
    if (! Trapped) then                                                 6
        T.addVertex(c_new)                                              7
        T.addEdge(c_near, c_new)                                        8
        if goalReached(c_new, c_goal) then                              9
            return Path(T)                                              10
return ApproxPath(T)                                                    11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

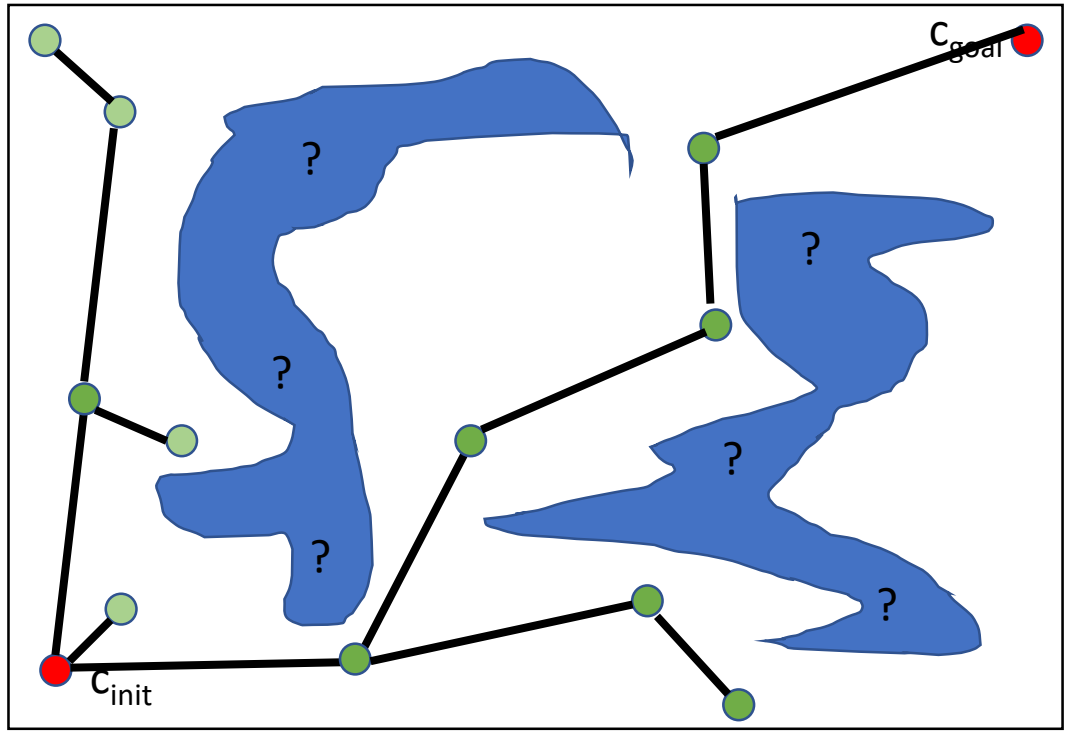
```
T.init(c_init)                                                    1
while (! TimeElapsed(T_max)) do                                   2
    c_rand ← RandomState()                                        3
    c_near ← NearestN(c_rand, T)                                  4
    c_new ← getValidEdge(c_near, c_rand, range_max)               5
    if (! Trapped) then                                           6
        T.addVertex(c_new)                                        7
        T.addEdge(c_near, c_new)                                  8
        if goalReached(c_new, c_goal) then                        9
            return Path(T)                                        10
return ApproxPath(T)                                              11
```



The **R**apidly-Exploring **R**andom **T**ree

Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

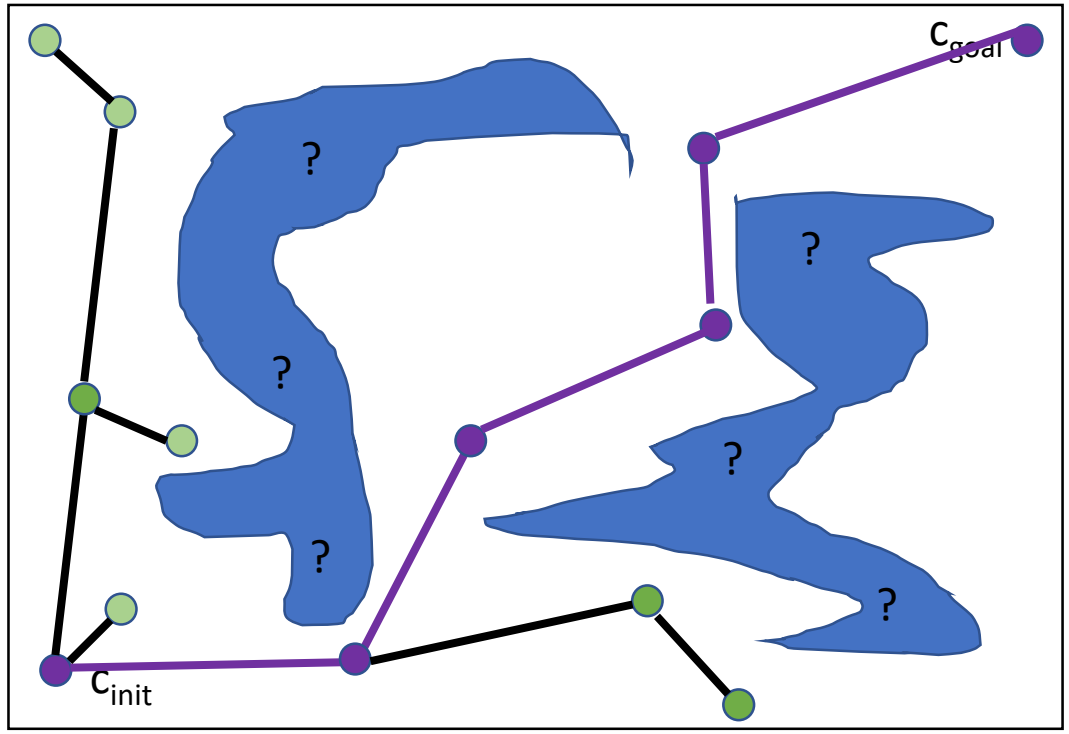
```
T.init(c_init)                                                    1
while (! TimeElapsed(T_max)) do                                   2
    c_rand ← RandomState()                                        3
    c_near ← NearestN(c_rand, T)                                  4
    c_new ← getValidEdge(c_near, c_rand, range_max)               5
    if (! Trapped) then                                           6
        T.addVertex(c_new)                                        7
        T.addEdge(c_near, c_new)                                  8
        if goalReached(c_new,c_goal) then                         9
            return Path(T)                                        10
return ApproxPath(T)                                              11
```



The **R**apidly-Exploring **R**andom **T**ree

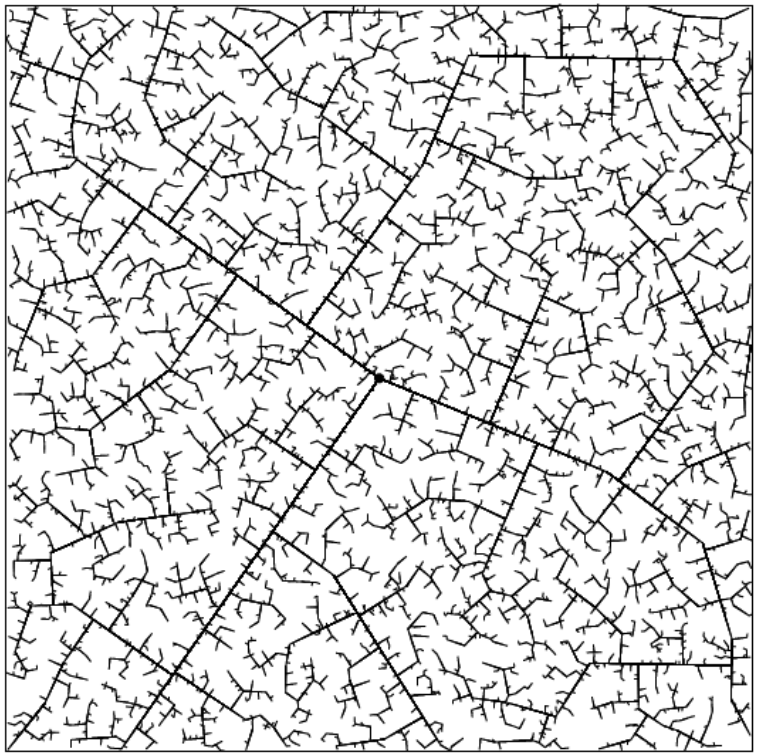
Algorithm 1: RRT $(c_{init}, c_{goal}, range_{max}, time_{max})$

```
T.init(c_init)                                                    1
while (! TimeElapsed(T_max)) do                                   2
    c_rand ← RandomState()                                        3
    c_near ← NearestN(c_rand, T)                                  4
    c_new ← getValidEdge(c_near, c_rand, range_max)               5
    if (! Trapped) then                                           6
        T.addVertex(c_new)                                        7
        T.addEdge(c_near, c_new)                                  8
        if goalReached(c_new,c_goal) then                         9
            return Path(T)                                        10
return ApproxPath(T)                                              11
```



On a Large scale

45 iterations



2345 iterations

This is why it is called rapidly exploring:
→it covers the c-space fast

Sources:
Motion Planning: The Essentials – LaValle - http://msl.cs.illinois.edu/~lavalle/papers/Lav11b.pdf

The **R**apidly-Exploring **R**andom **T**ree - **C**onnect

Algorithm 2: RRTC $(c_{init}, c_{goal}, range_{max})$

```
T1.init(c_init), T2.init(c_goal)                                   1
while (! TimeElapsed(T_max)) do                                    2
    T_current, T_other ← Swap(T1, T2)                              3
    c_rand ← RandomState()                                         4
    c_near ← NearestN(c_rand, T_current)
    c_new ← getExtend(c_near, range_max)
    if (! Trapped) then
        T_current.addVertex(c_new)                                 8
        T_current.addEdge(c_near, c_new)                           9
        c_near' ← NearestN(c_new, T_other)                         10
        c_new' ← getExtend(c_near', range_max)                     11
        if (! Trapped) then                                        12
            T_other.addVertex(c_new')                              13
            T_other.addEdge(c_near', c_new')                       14
            if (Reached) then                                      15
                return Path(T1, T2)                                16
return ApproxPath(T1, T2)                                          17
```

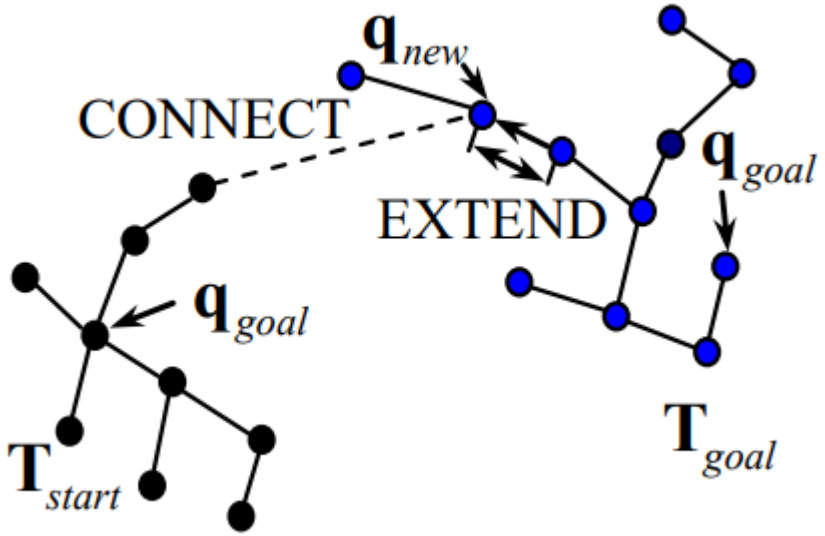
Assume getExtend = getValidEdge

$c_{new} \leftarrow \text{getValidEdge}(c_{near}, c_{rand}, range_{max})$

Exercise:

- Analyse the algorithm shown on the left.
- Write down an example and try to understand the difference.

Solution

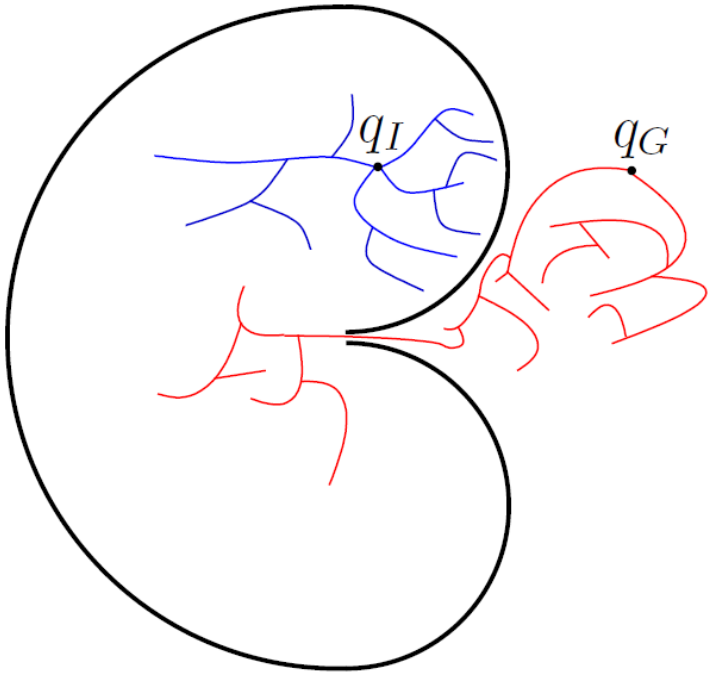


Sources:
Bidirectional RRT Algorithm for Collision Avoidance Motion Planning of FFSR – Huazhong Li, Yongsheng Liang

- The algorithm builds up two trees.
- In each iteration the algorithm tries to connect both trees and as soon as the trees are connected the path is found.
- Difference *extend* and *validEdge:* The extend function add also configurations on the edge of the tree.

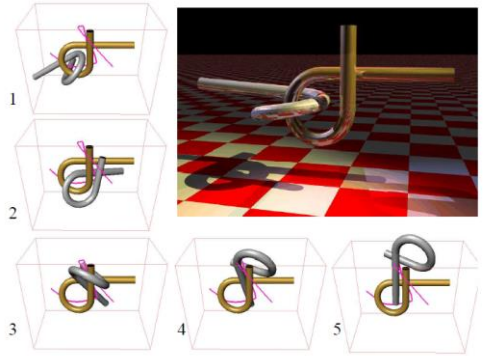
Why two trees?

The bugtrap problem



Sources:
Planning Algorithms– LaValle - http://planning.cs.uiuc.edu/

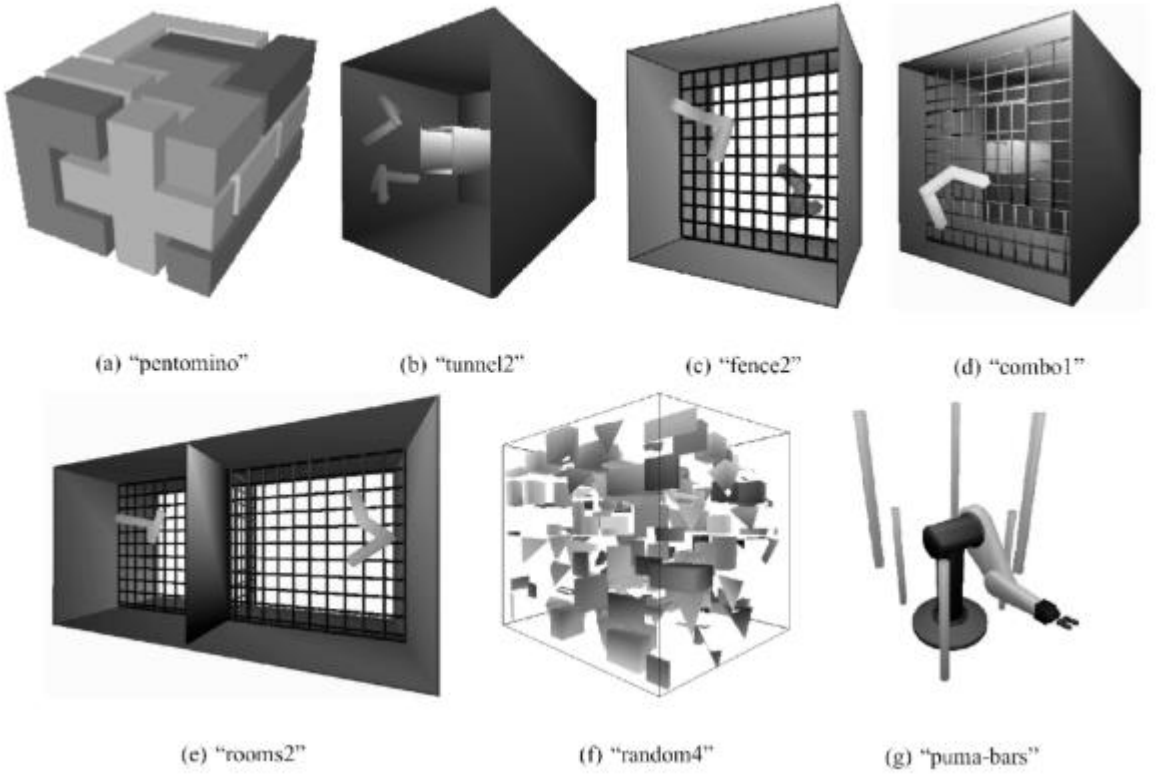
- There are multiple scenarios that one tree can be stuck. Two trees can help to solve this issues.
- The RRT-Connect was the first algorithm to solve the famous alpha puzzle.



Sources:
Planning Algorithms– LaValle - http://planning.cs.uiuc.edu/

More than two?

- Yes, it makes sense to go for more than two trees.
- One can even build up a “roadmap of Trees”. (Combination of sPRM and RRT)
- These algorithms are good for parallel programming on the CPU.
- With this one can solve algorithms in even more dimensions.



Sources:
Sampling-Based Roadmap of Trees for Parallel Motion Planning -Lydia E. Kavraki et. al.
- https://www.clear.rice.edu/comp450/papers/plaku2005srt-parallel-motion-planning.pdf

Expansive Space Tree

Algorithm 4: EST $(c_{init}, c_{goal}, r_1, r_2)$

```
T.init(c_init)                                              1
while (! TimeElapsed(T_max)) do                             2
    c_way ← chooseWaypoint(T)                               3
    c_new ← expandWaypoint(c_way, r_1)                      4
    if (! Trapped) then                                     5
        T.addVertex(c_new)                                  6
        T.addEdge(c_way, c_new)                             7
        T.updateWeights(r_2)                                8
        if goalReached(c_new,c_goal) then                   9
            return Path(T)                                  10
return ApproxPath(T)                                        11
```

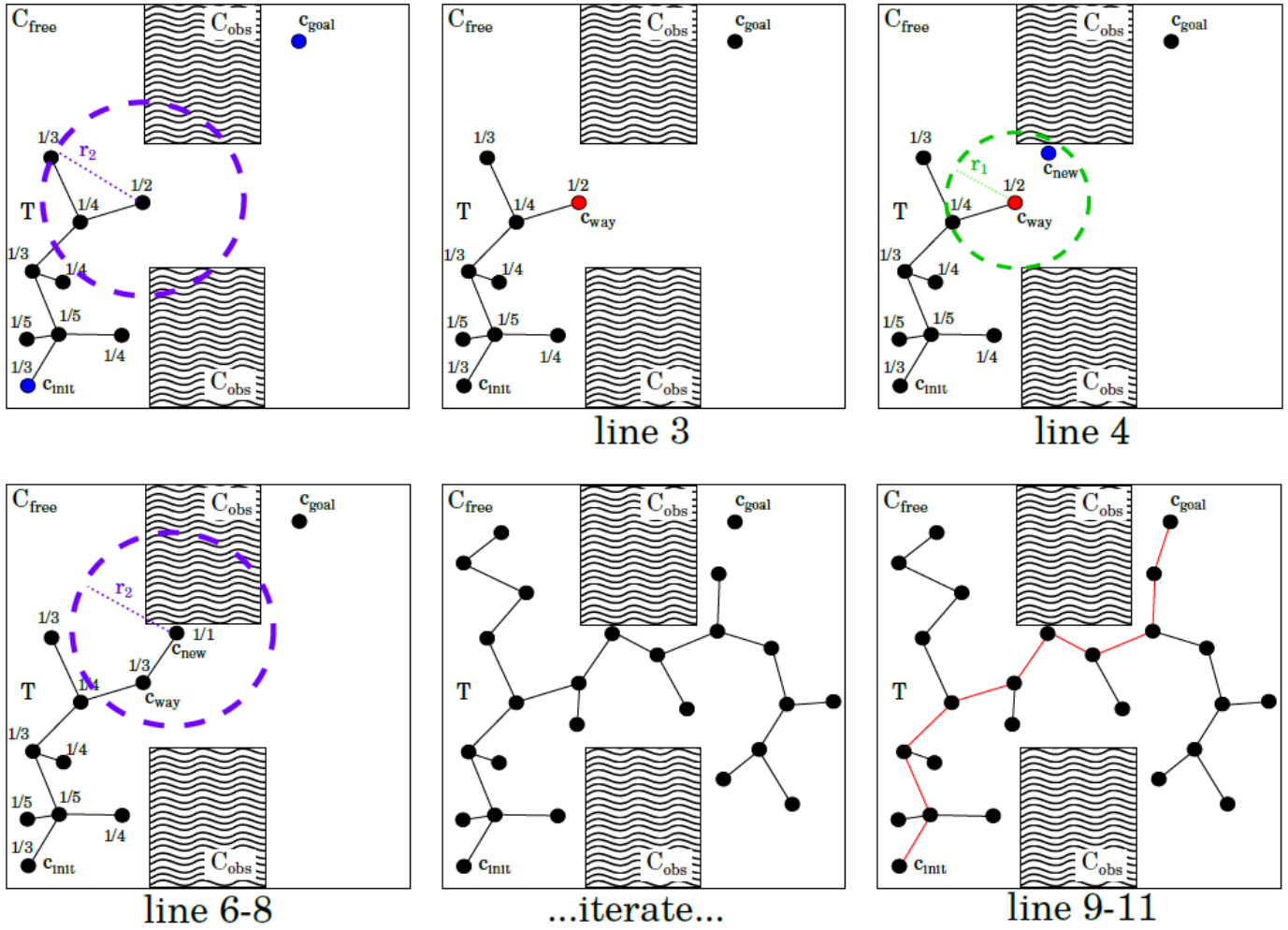


Fig. 11: An example of an Expansive Space Tree approach.