5.5 Rapidly Exploring Dense Trees (RDTs) 3/1/18

- Incremental and resolution complete
- Requires no tuning parameters?

  No, but no explicit setting of resolution parameters.

Idea: grow a tree incrementally that is equally close to every point in $\mathcal{C}_{free}$, and gradually increases resolution.

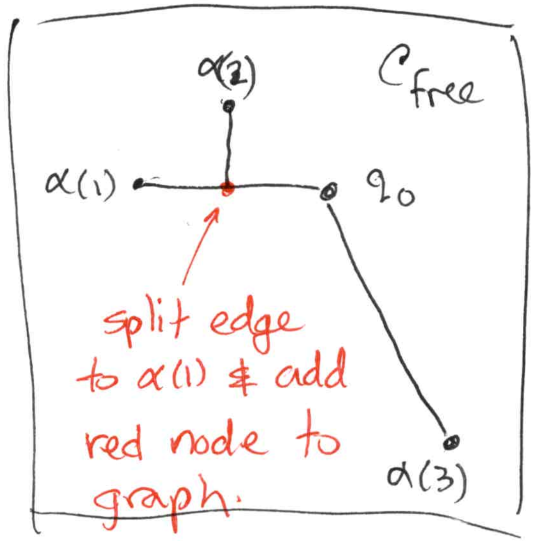
Let S denote, the swath of a Graph.

$$S(\mathcal{G}) = \bigcup_{i=1}^{n} e([0,1])$$ ← union of all edges

Let α be a sequence of points that is dense in $\mathcal{C}$.

Grow tree as follows: SIMPLE RDT alg.

```
𝒢.init(q_0)
for i = 1 to k do:
    𝒢.addvertex(α(i))
    q_n ← nearest(S(𝒢), α(i))
    𝒢.addedge(q_n, α(i))
```



Show Figure 5.19 in LaValle, Page 230.

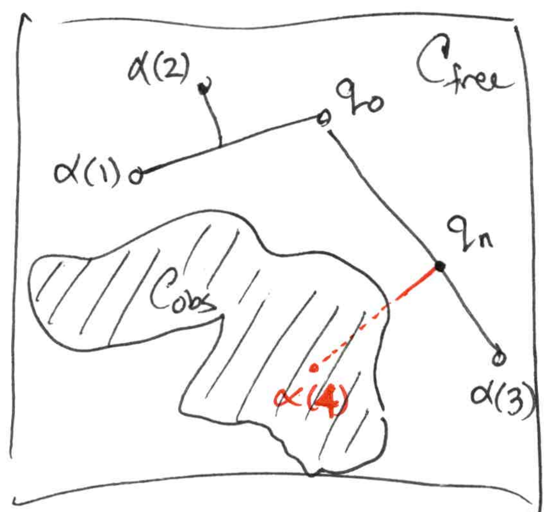
3/1/18

2

What about obstacles?

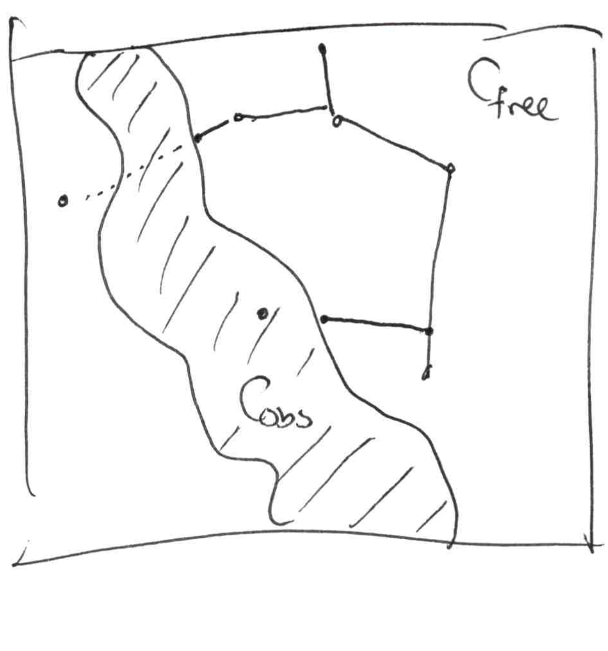
If $\alpha(i) \in C_{obs}$ OR if $\alpha(i)$ cannot be reached from S due to collision

Then just use the reachable portion of path to $\alpha(i)$.



Does this strategy retain dense coverage of C_{free}?

LaValle says "yes," but this cannot be true of disconnected components of C_{free}.



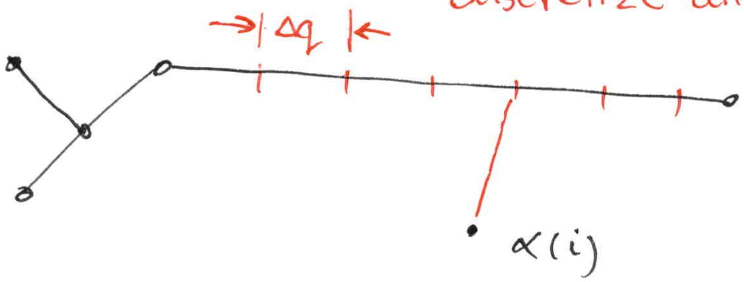
Efficiently finding nearest points

Exact – no parameter needed

but can be difficult in curved C-spaces.

Approx – parameter needed, Δq, & increase # of points

discretize all edges



Approx method is easier to implement.

Just repeatedly compute distance metric.

Use a kd-tree to speed search

(don't compute all distances)

Let k = # of points and n = # dimensions of C_{free}.

construction is $O(nk \lg(k))$

Closest point is $O(\lg(k))$

3/1/18

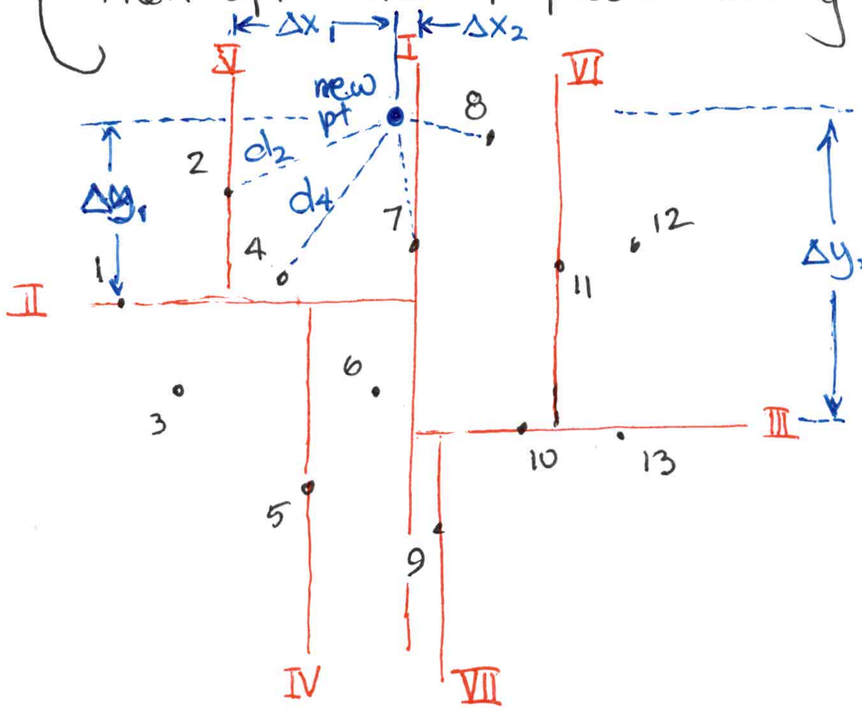
4

Given a set of points

Split them along x at median

Then split each half along medians in y-direction

Then split the 4 pieces along medians in 3rd direction



Finding closest pt.

Descend tree to leaf

Compute dist to pt in leaf (d_4)

Ascend to leaf's parent & check distance (d_2) (if $d_4 > \Delta x_1$)

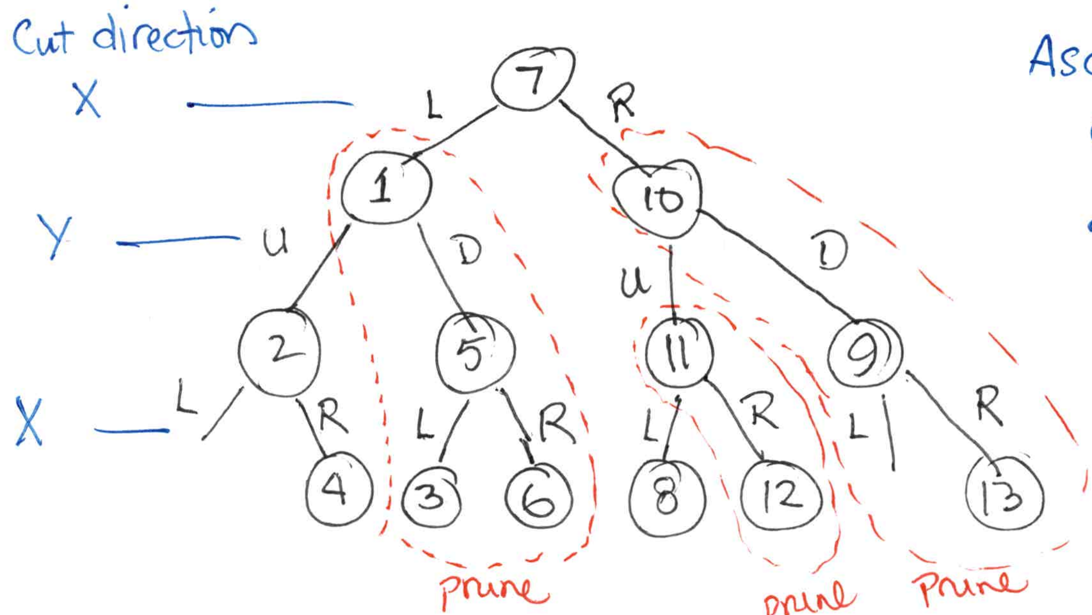
Ascend to parent & check ~~dist~~ $d_2 < \Delta y_1$

∴ prune 1

Ascend to parent $d_2 > \Delta x_2$, so compute d_7

Descend to 10.

$d_2 < \Delta y$, so prune 10 D.



Solution: closest pt is pt 8

Descend to 8

$\boxed{\text{solution}}$

Descend to 11

$d_2 < \Delta x_3$, so prune 11 R

Balanced tree allows faster searches due to smaller # of levels.

$\#\text{ levels} = \lceil \log_2(k) \rceil$

Finding nearest neighbor is $O(\log_2 k)$.

Using trees for planning

Single tree

grow from q_I

parameter

Every 100 or so samples, insert q_G into α which forces RDT to try to connect to q_G

Balanced Bi-directional search

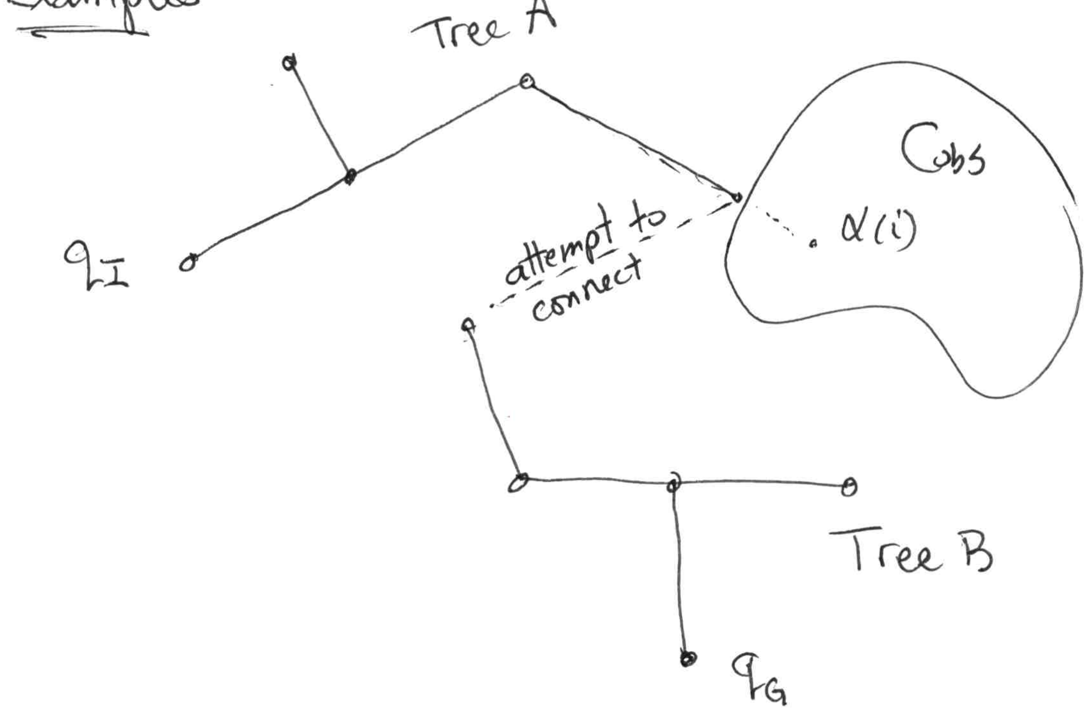
Grow 2 trees; from q_I & q_G

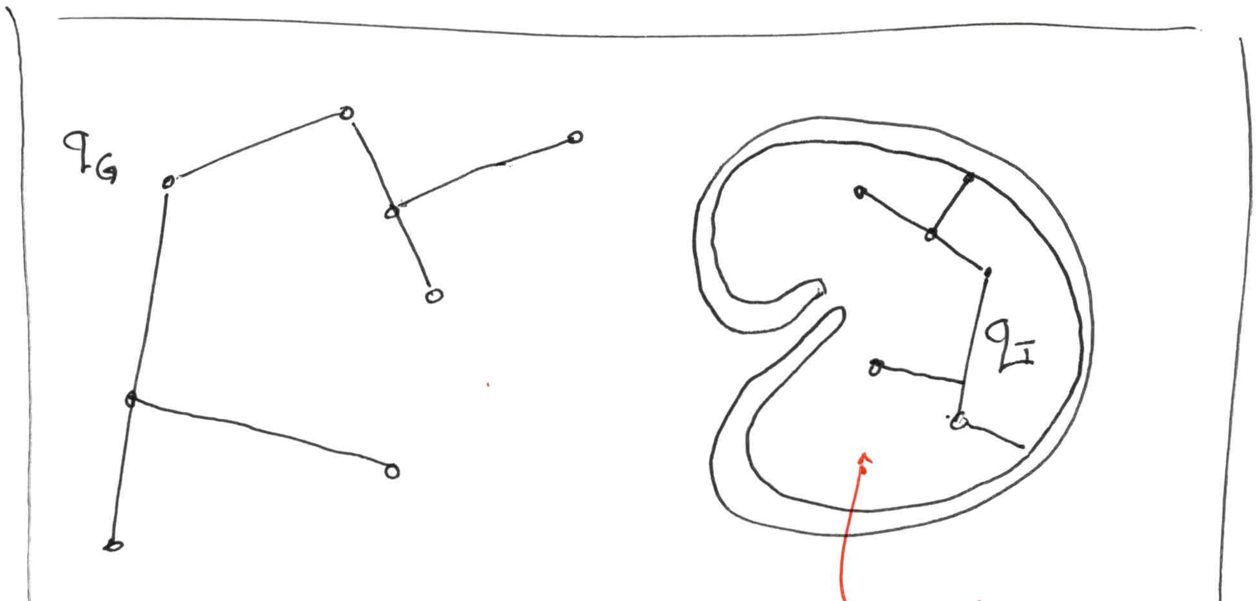
Add points to tree alternatively to keep trees the same size ← otherwise operations on one tree will dominate. Is that bad?

parameter

Every so often, after attaching a vertex to one tree, try to attach it to the other.

Examples





Tree in smaller space could get much denser. Maybe that's ok.

3/1/18

⑦

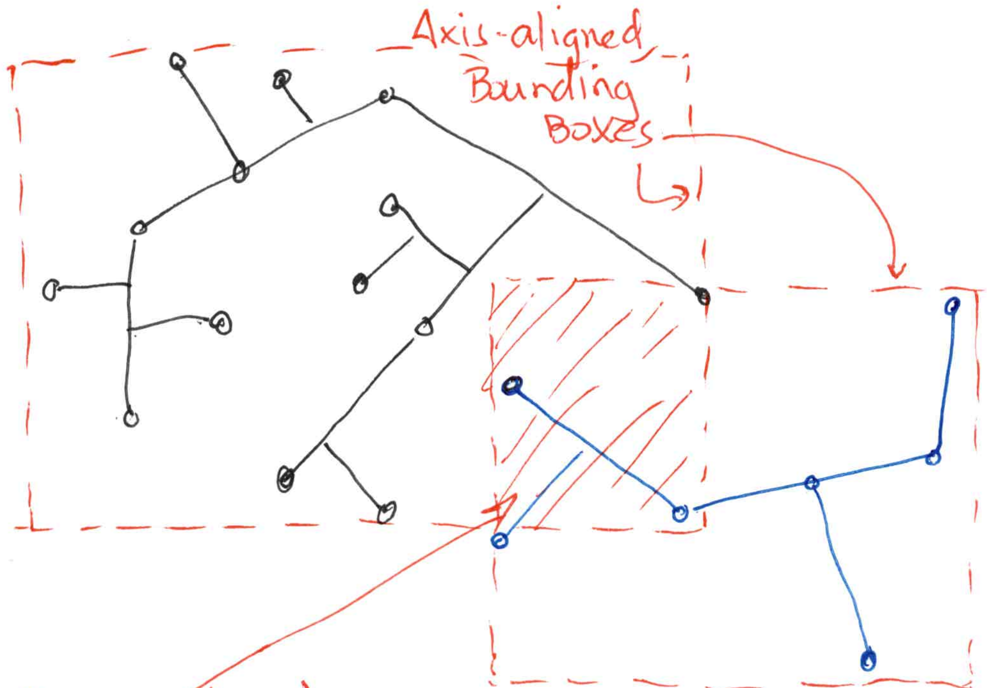
Many trees

Tree connection attempts could be too time consuming.

Possibly choose a pair of trees at random.

which pair of points? Leaves?

Maybe maintain min & max coords for each tree and choose from the overlap.



Choose pts at random from overlap of AABBs?

3/1/18

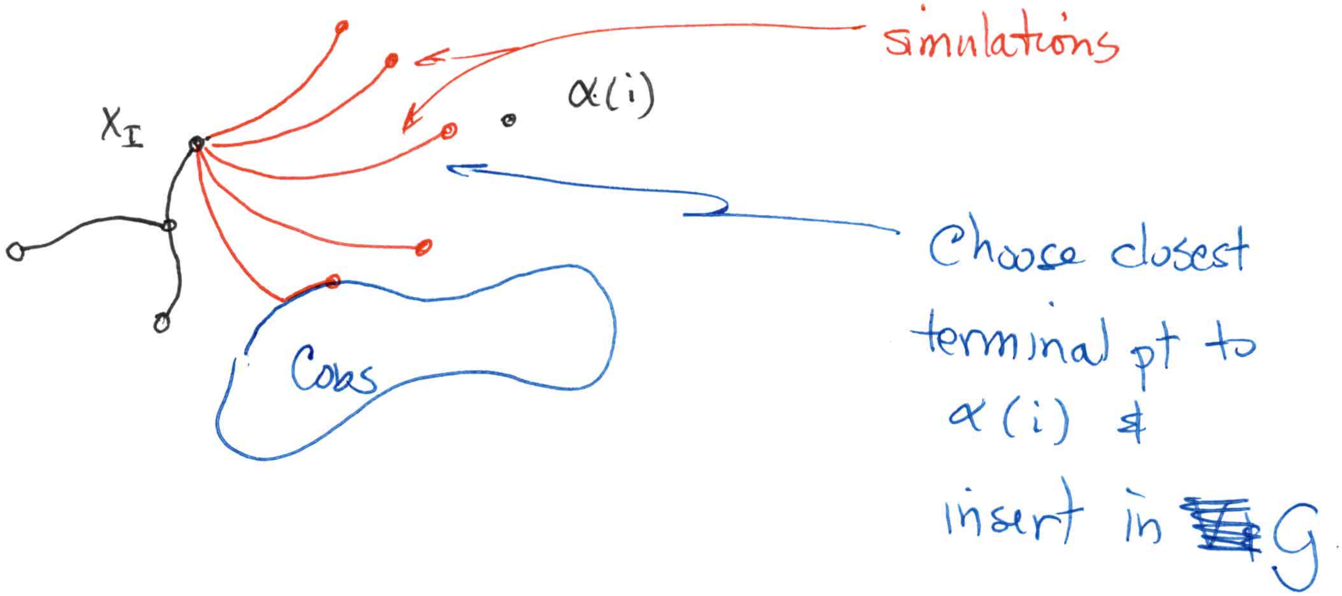
8

Modify RDT for complex dynamics

So far we've assumed easy soln of 2pt b.v. problems

Not easy for complex dynamic systems

Approach: Choose $\alpha(i)$ as before, but try to connect by sampling input space.



$\boxed{\text{Show Matlab example w/ bicycle model}}$

LaValle 5.6: Roadmap Methods for multiple queries

Suppose you will need ~~many~~ solutions for many (q_I, q_G) pairs?

Then its worth spending time to build a good approx to C_{free}

(Caveat: thar could be a large # of components of C_{free})

Goal: build $\mathcal{G}$ of C_{free} such that:

1. LPM can easily connect any point in C_{free} to $\mathcal{G}$.

2. $\mathcal{G}$ is small, so searching (solving each query) is fast.

Terminology:

Probabilistic roadmap methods (PRMs)

Sampling-based roadmap methods

2 phases

- Preprocessing - build G
- Query phase - connect q_I & q_G, extract path

Preprocessing

0. Select N pts from a dense sequence
1. G.init; $i=0$
2. While $i<N$
3. if $\alpha(i) \in C_{free}$ then
4. G.add_vertex$(\alpha(i))$; $i=i+1$.
5. for each $q \in$ nbhd$(\alpha(i), G)$
6. if $[\neg$ G.same_component$(\alpha(i), q) \wedge$ connect$(\alpha(i), q)]$ then G.add_edge$(\alpha(i), q)$

If N too big, long preprocessing time

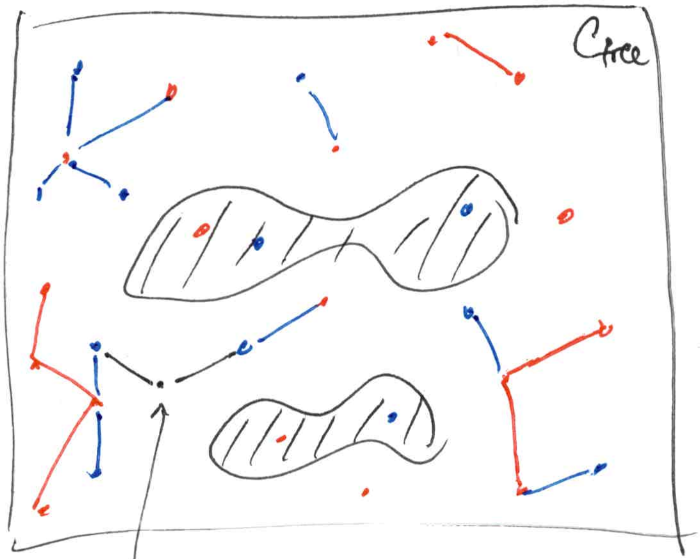
If N too small, queries often fail

If α's poorly chosen, too many pts needed for good roadmap.

Example

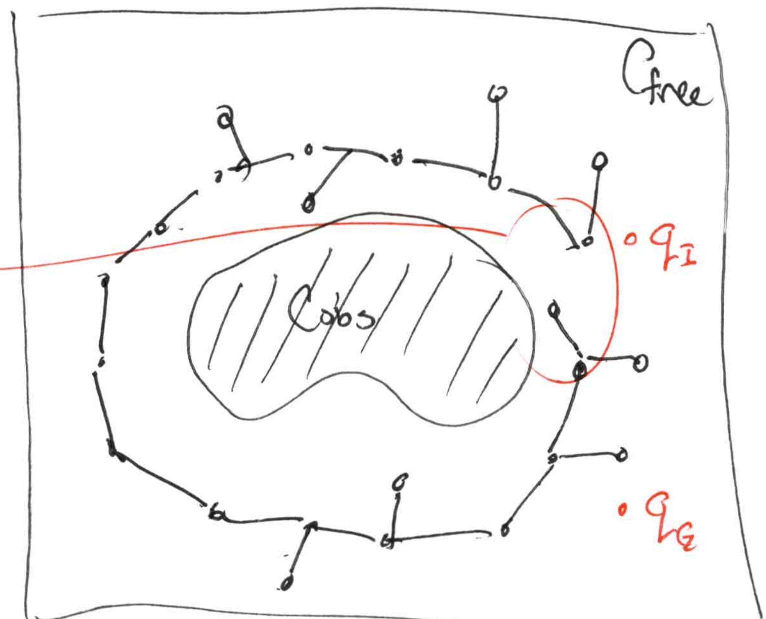
For this example, we know $\mathcal{G}$ should have 1 component with two "holes."

(How do we count holes in a graph?)



this pt connects two components of $\mathcal{G}$.

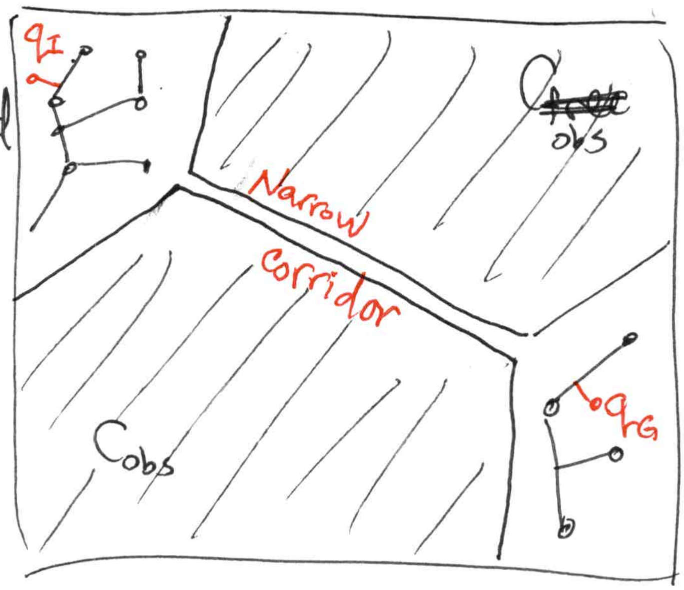
The given alg will not close loop, so path could be very long.



Possibly replace $\neg \mathcal{G}.same\_comp(\alpha(i), q)$ with $\mathcal{G}.vertex\_degree(q < D)$

If LPM fails to connect q_I or q_G, then N was too small.

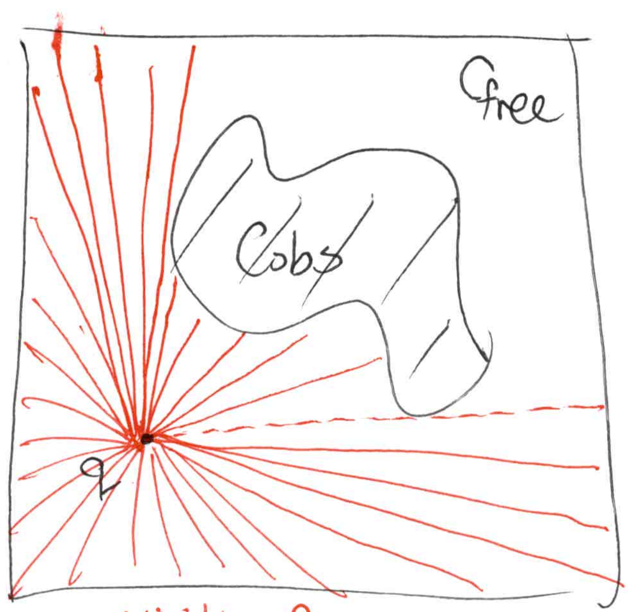
If q_I and q_G are connected but no solution is found, then path $\neg\exists$ or G ~~not have same~~ ~~compo~~ does not reflect structure of C_{free} well.



Visibility Roadmap

Cover C_{free} well with small # of pts

(For complex dynamics, reachability region is analogous)



Visible from q.

Let q be a "guard" if it cannot "see" any other guard

i.e. easily connected by LPM

Let q be a "connector" if it can see at least 2 guards in two diff components of $\mathcal{G}$

Vis Roadmap Alg

0. $q_I, q_G \rightarrow \mathcal{G}$ If can't connect, they are guards.
1. while $i < N$
2. place sample $\alpha(i)$
3. if $\alpha(i)$ is guard, $\mathcal{G}$.insert($\alpha(i)$)
4. if $\alpha(i)$ is connector, $\mathcal{G}$.insert($\alpha(i)$) & connect edges.
5. otherwise discard $\alpha(i)$.

How to chose N?

Construct incrementally?

Stop when 1,000 pts in a row have been discarded?