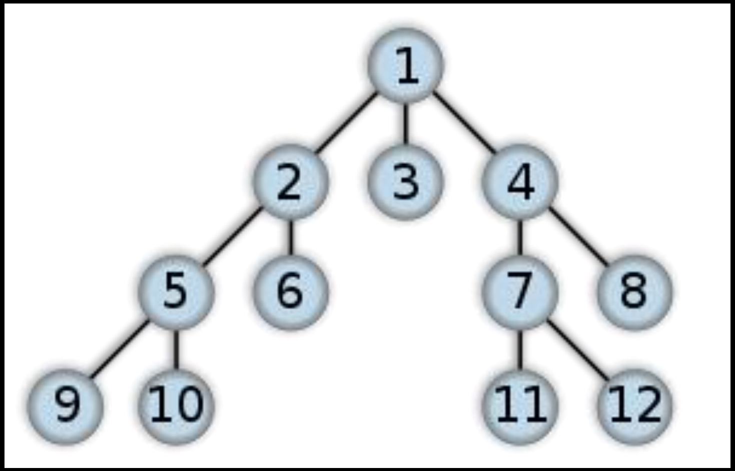
Sampling-based Planning 1

A lot of Material from Howie Choset, Nancy Amato, Sujay Bhattacharjee, G.D. Hager, S. LaValle, J. Kuffner, D. Hsu

Previously…

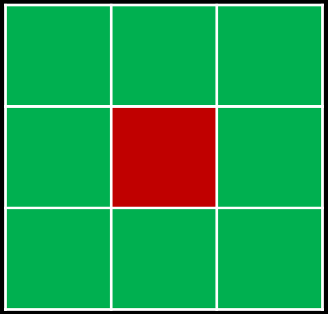
- We learned about discrete motion planning



- Discrete planning is best suited for
  - Low-dimensional motion planning problems
  - Problems where the control set can be easily discretized
- What if we need to plan in **high-dimensional** spaces?

Discrete Planning: The problem

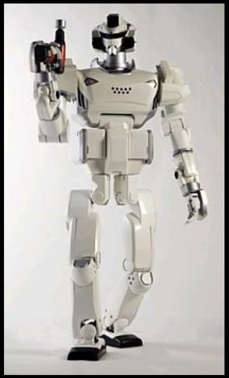
- Discrete search run-time and memory requirements are very sensitive to branching factor (number of successors)
- Number of successors depend on dimension
- For a 3-dimensional 8-connected space, how many successors?
- For an n-dimensional 8-connected space, how many successors?


8-connected

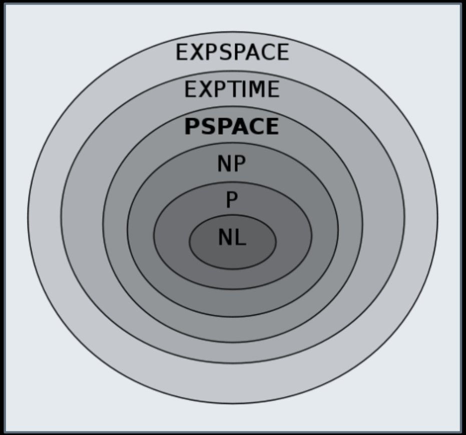
The problem

- Need a path planning method that isn't so sensitive to dimensionality
- But:
  - Path planning is PSPACE-hard

    [Reif 79, Hopcroft et al. 84, 86]
  - Complexity is exponential in dimension of the C-space [Canny 86]
- What if we weaken *completeness* and *optimality* requirements?



Real robots can have 20+ DOF!



Weakening requirements

<u>Ideal</u>		<u>Practical in High Dimensions</u>
Complete	$\longrightarrow$	Probabilistically Complete
Optimal	$\longrightarrow$	Feasible

*More recent methods show *asymptotic* optimality

- Probabilistic completeness: A path planner is probabilistically complete if, <u>given a solvable problem</u>, the probability that the planner solves the problem goes to 1 as time goes to infinity.

- Feasibility: Path obeys all constraints (usually obstacles).
- A feasible path can be optimized *locally* after it is found

Sampling-based planning

- Main idea: Instead of systematically-discretizing the C-space, take samples in the C-space and use them to construct a path

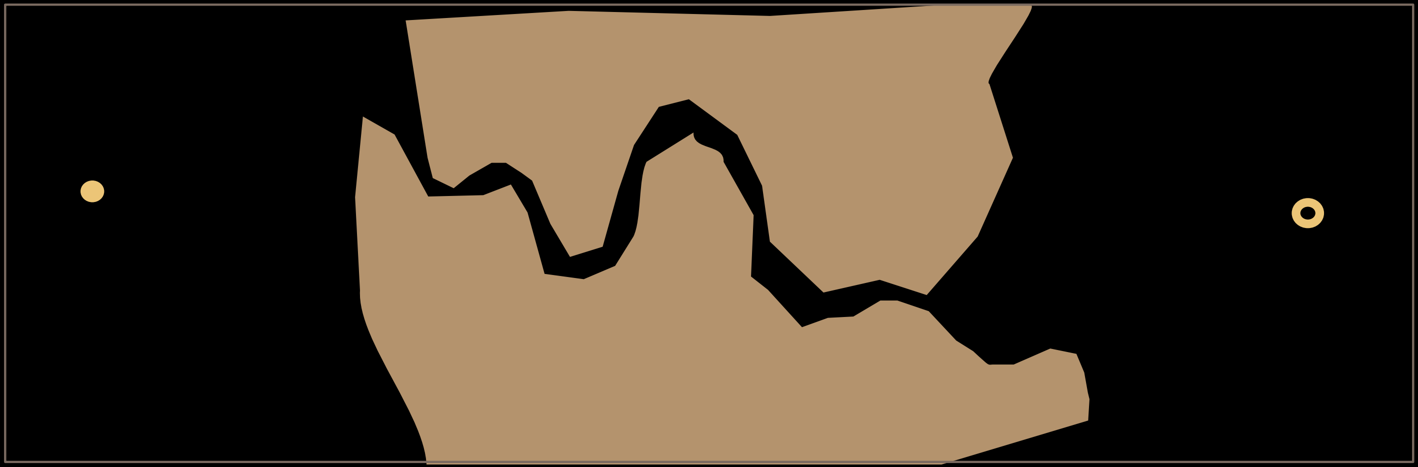
Sampling-based planning

Advantages

- Don’t need to discretize C-space
- Don’t need to explicitly represent C-space
- Easy to sample high-dimensional spaces

Disadvantages

- Probability of sampling an area depends on the area’s size
    - Hard to sample *narrow passages*
- No strict completeness/optimality

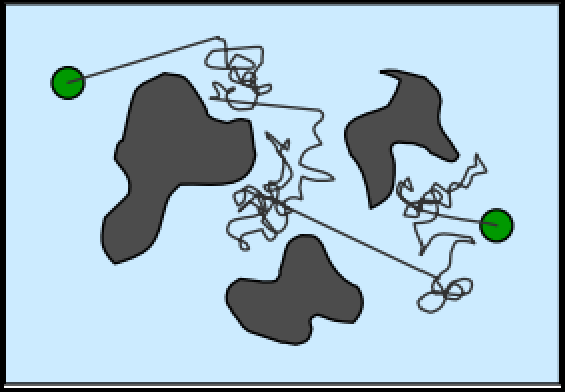


Outline

- RPP
- PRM
- Expansive Spaces

Randomized Path Planner (RPP)

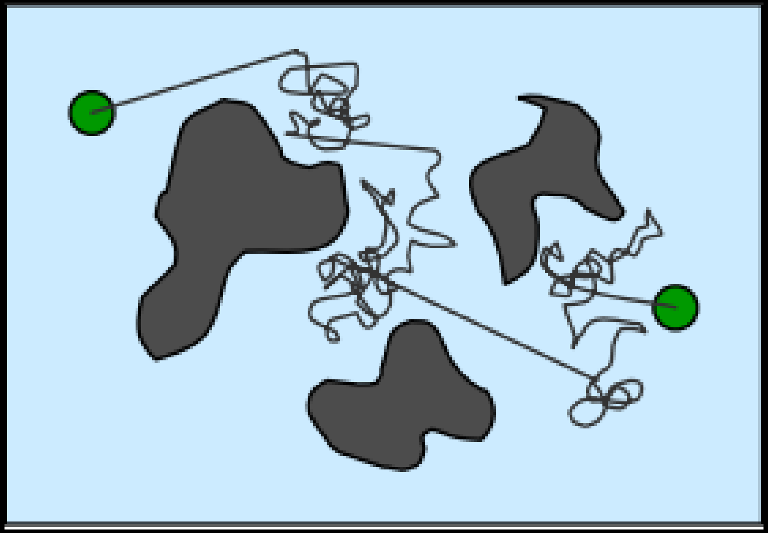
- Developed by Barraquand and Latombe in 1991 at Stanford



- Main idea: Follow a potential function, occasionally introduce random motion
  - Potential field biases search toward goal
  - Random motion avoids getting stuck in local minima

RPP

- Advantage: Doesn’t get stuck in local minima
- Disadvantage: Parameters needed to
  - define potential field
  - decide when to apply random motion
  - how much random motion to apply



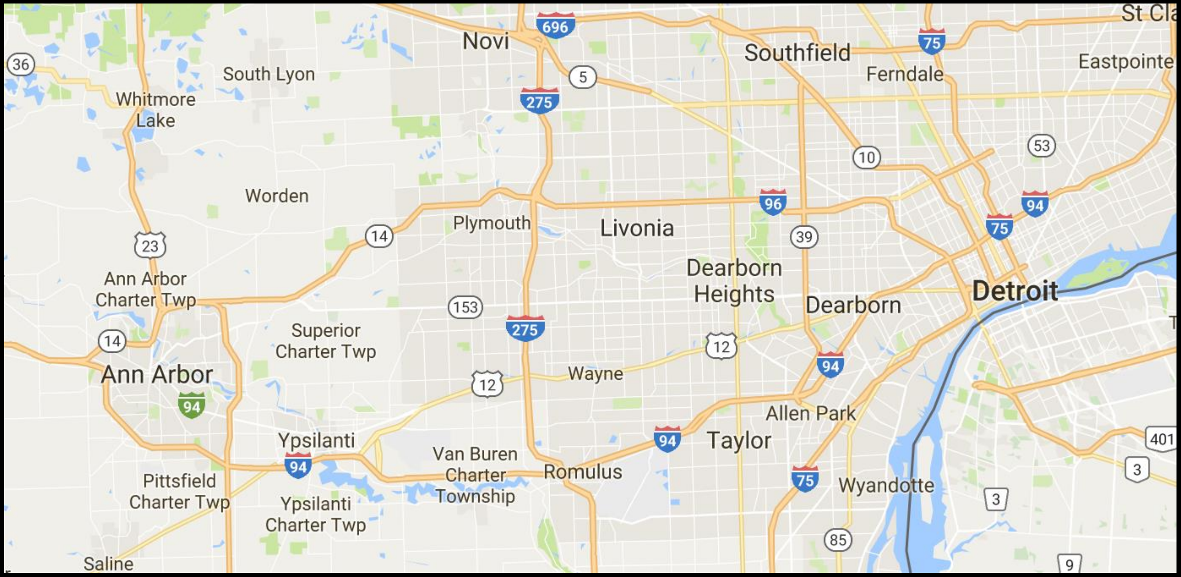
Probabilistic Roadmap (PRM)

- Perhaps the most famous motion planning paper:

  Kavraki, Lydia E., Petr Svestka, J-C. Latombe, and Mark H. Overmars. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces." IEEE Transactions on Robotics and Automation 12, no. 4, 1996.

- Main idea: Build a roadmap of the space from sampled points, search the roadmap to find a path

- Roadmap should capture the *connectivity* of the free space

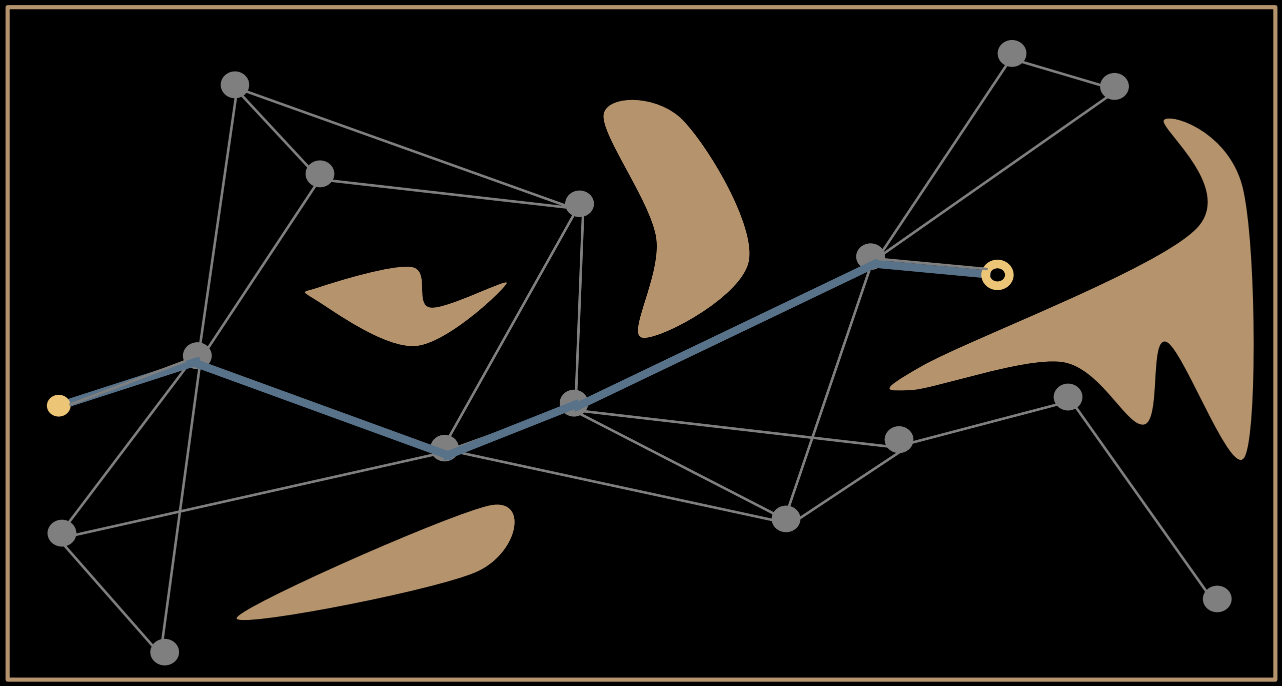


Probabilistic Roadmap (PRM)

- Building a PRM: 2 phase process
- "Learning" Phase
  - Construction Step
  - Expansion Step
- Query Phase
  - Answer a given path planning query

- PRMs are known as *multi-query algorithms*, because roadmap can be re-used if environment and robot haven't changed between queries.

PRM Example



“Learning” Phase

- Construction step: Build the roadmap by sampling random free configurations and connect them using a fast *local planner*
- Store these configurations as nodes in a graph
    - Note: In PRM literature, nodes are sometimes called “milestones”
- Edges of the graph are the paths between nodes found by the local planner

Construction Step

```
Start with an empty graph G = (V,E)
For i = 1 to MaxIterations
    Generate random configuration q ⟵
    If q is collision-free
        Add q to V
        Select k nearest nodes in V ⟵
        Attempt connection between each of these nodes and q using local planner
        If a connection is successful, add it as an edge in E
```

Construction Step

```
Start with an empty graph G = (V,E)
For i = 1 to MaxIterations
    Generate random configuration q
    If q is collision-free
        Add q to V
        Select k nearest nodes in V
        Attempt connection between each of these nodes and q using local planner
        If a connection is successful, add it as an edge in E
```

Sampling Collision-free Configurations

- Easiest and most common: uniform random sampling in C-space
  - Draw random value in allowable range for each DOF, combine into a vector
  - Place robot at the configuration and check collision
  - Repeat above until you get a collision-free configuration
  - AKA “Rejection Sampling”
- MANY ways to do this, many papers published, we will discuss more methods later

Construction Step

```
Start with an empty graph G = (V,E)
For i = 1 to MaxIterations
    Generate random configuration q ←
    If q is collision-free
        Add q to V
        Select k nearest nodes in V ←
        Attempt connection between each of these nodes and q using local planner
        If a connection is successful, add it as an edge in E
```

Finding Nearest Neighbors (NN)

- Need to decide a distance metric $D(q_1,q_2)$ to define “nearest”
- D should reflect likelihood of success of local planner connection (roughly)
  - If $D(q_1,q_2)$ is small, success should be likely
  - If $D(q_1,q_2)$ is large, success should be less likely
- By default, use Euclidian distance:

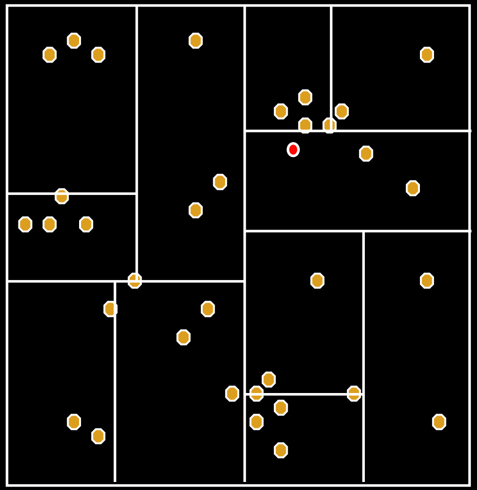
$$D(q_1,q_2) = ||q_1 - q_2||$$

- Can weigh different dimensions of C-space differently
  - Often used to weigh translation vs. rotation

Finding Nearest Neighbors (NN)

- Two popular ways to do NN in PRM
  - Find k nearest neighbors (even if they are distant)
  - Find all nearest neighbors within a certain distance

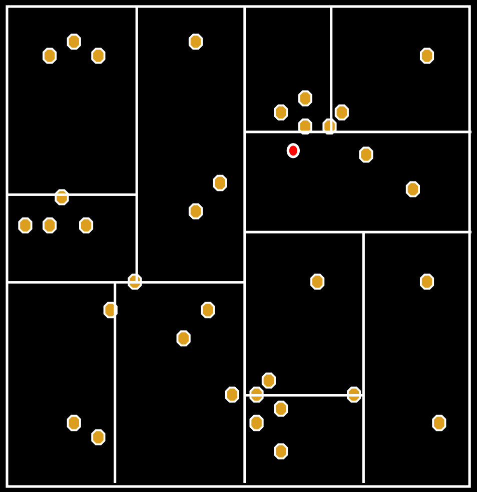
- Naïve NN computation can be slow with 1000s of nodes, so use *kd-tree* to store nodes and do NN queries
  - A kd-tree is a data-structure that recursively divides the space into bins that contain points (like Oct-tree and Quad-tree)
  - NN then searches through bins (not individual points) to find nearest point



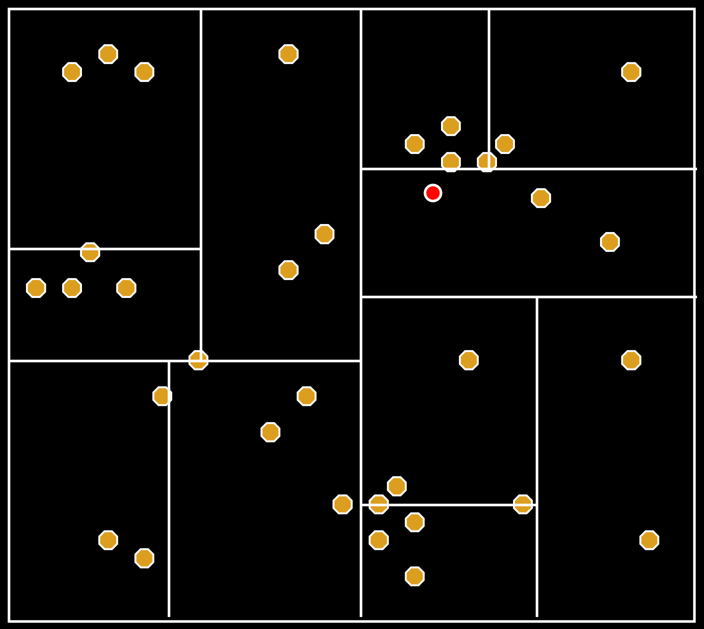
kd-trees (2D example)

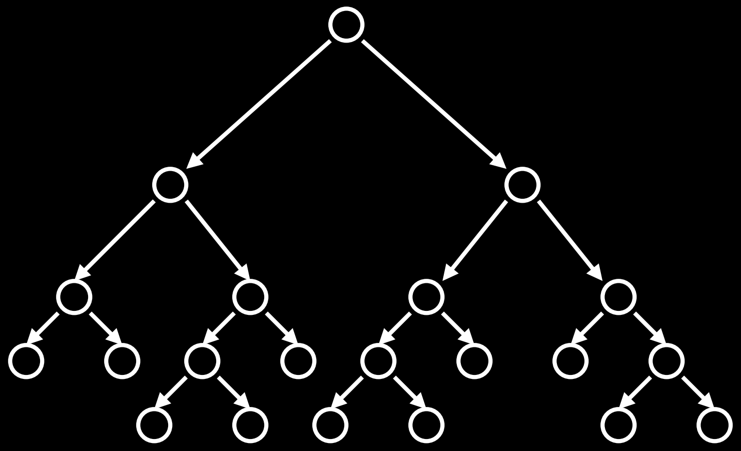
- Data structure with one-dimensional splits in the data
- Algorithm
  - Choose x or y coordinate (alternate between them)
  - Choose the median of the coordinate for the points in this cell
    - this defines a horizontal or vertical line
    - can use other variants instead of median to select split line
  - Recurse on both sides until there is only one point left, which is stored as a leaf

- We get a binary tree
  - Size O(n)
  - Construction time O(nlogn)
  - Depth O(logn)



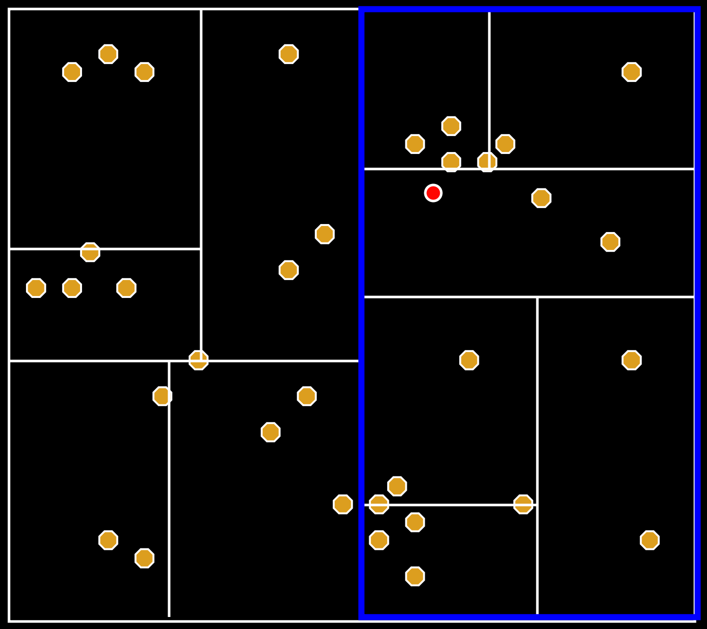
Nearest Neighbor with KD Trees

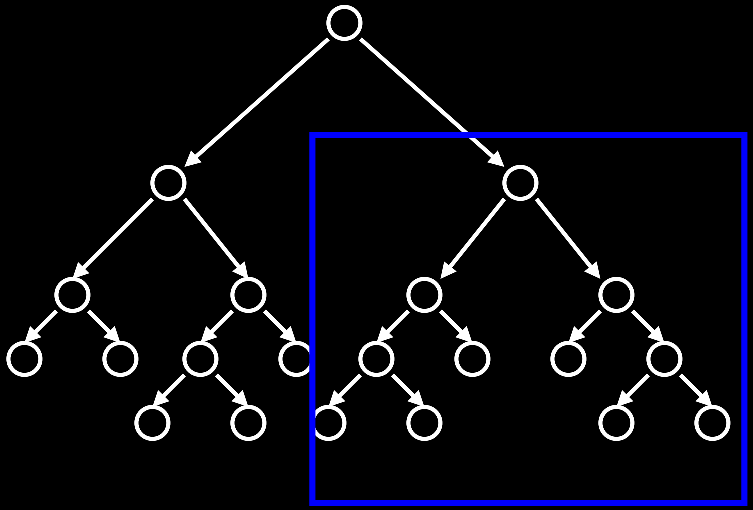




We traverse the tree looking for the nearest neighbor of the query point

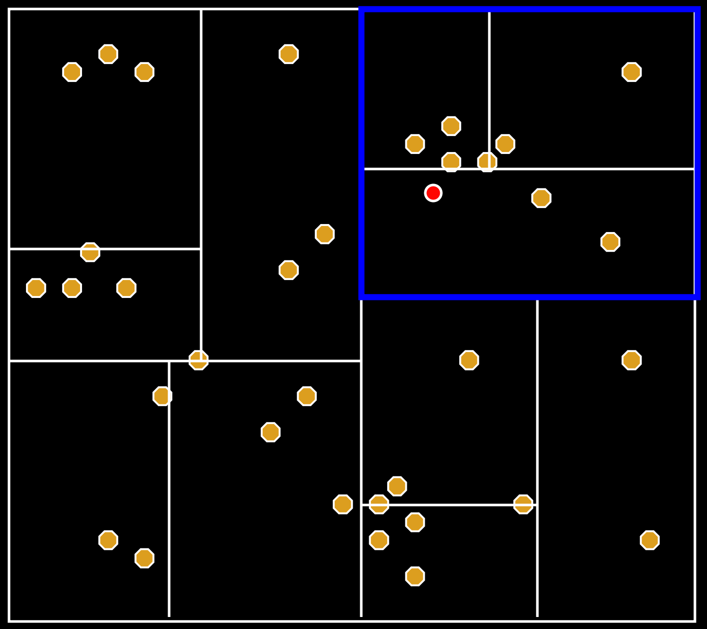
Nearest Neighbor with KD Trees

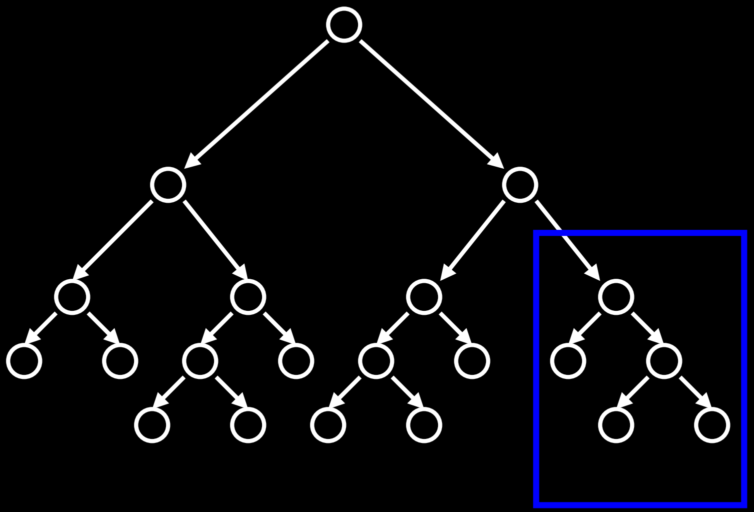




Examine nearby points first: Explore the branch of the tree that is closest to the query point first.

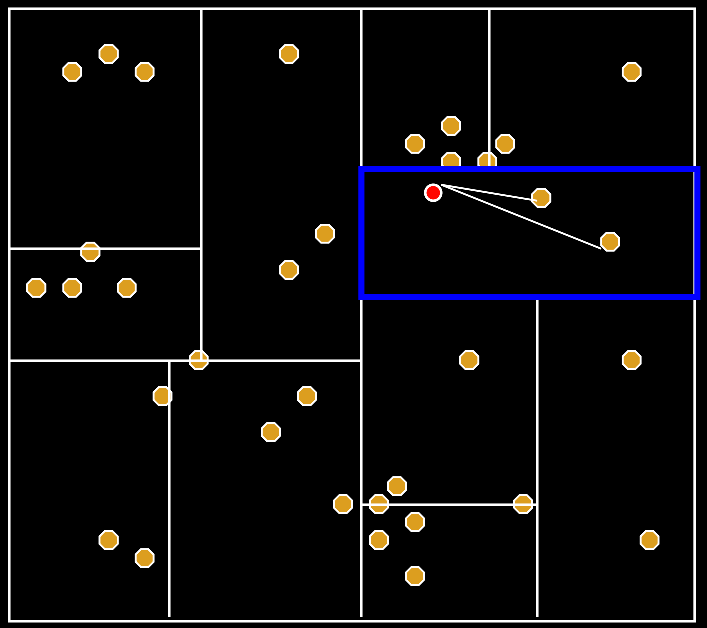
Nearest Neighbor with KD Trees

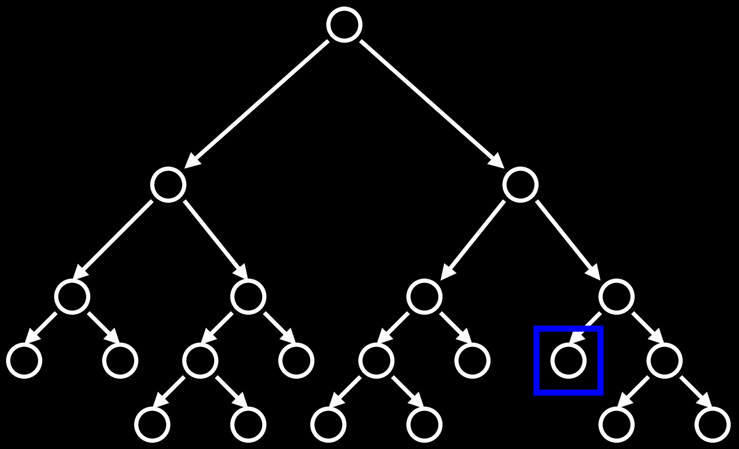




Examine nearby points first: Explore the branch of the tree that is closest to the query point first.

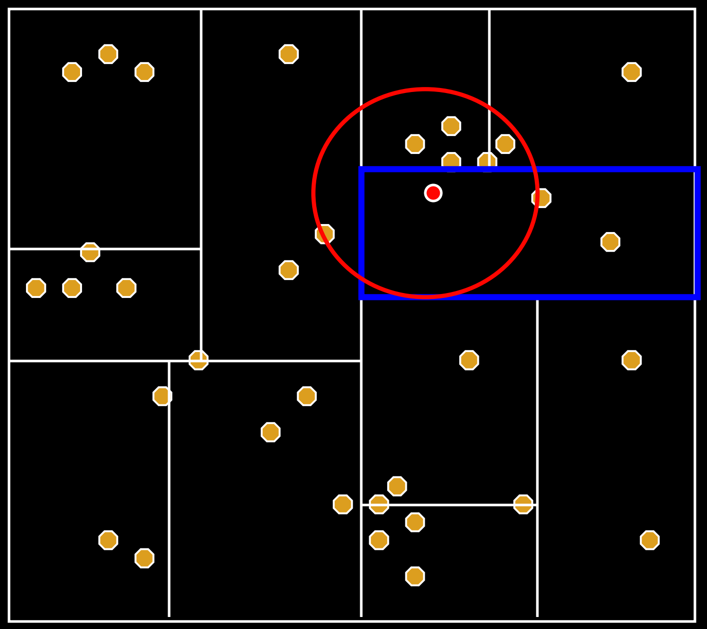
Nearest Neighbor with KD Trees

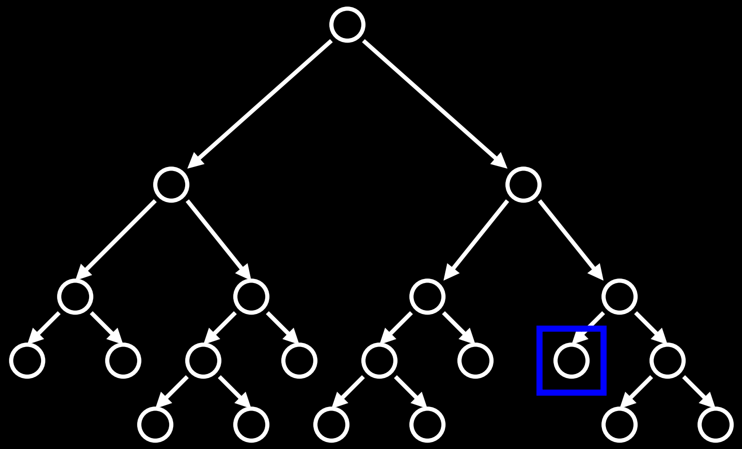




When we reach a leaf node: compute the distance to each point in the node.

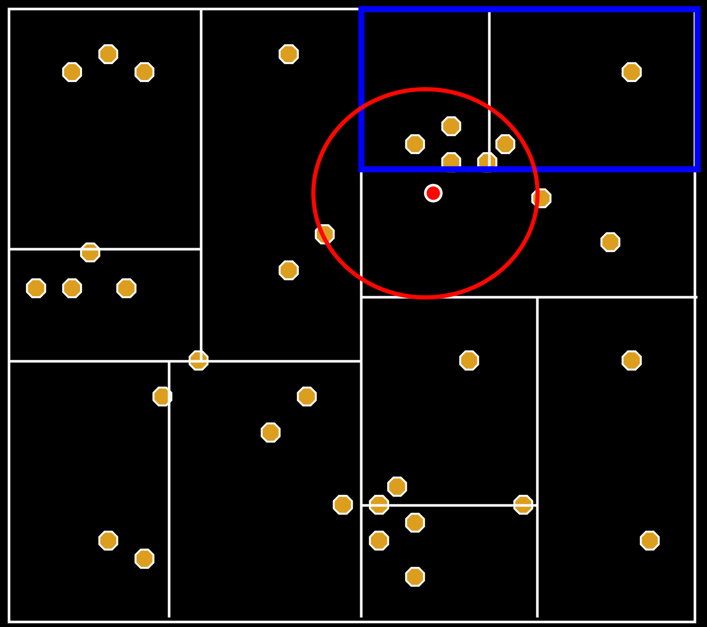
Nearest Neighbor with KD Trees

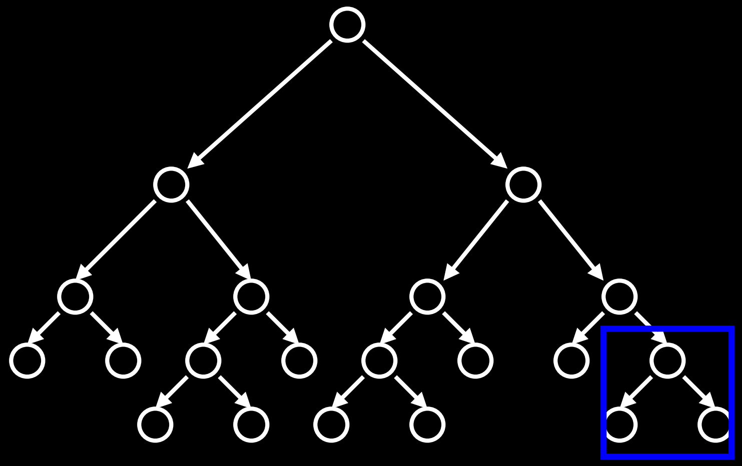




When we reach a leaf node: compute the distance to each point in the node.

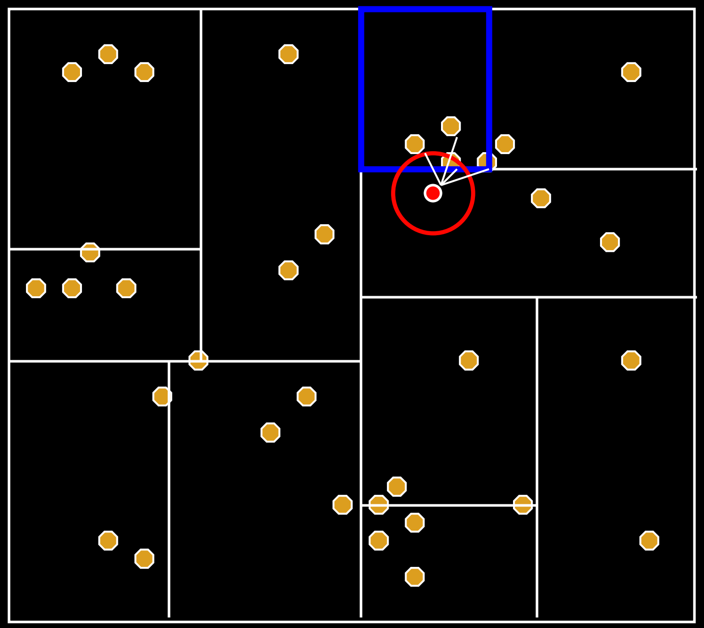
Nearest Neighbor with KD Trees

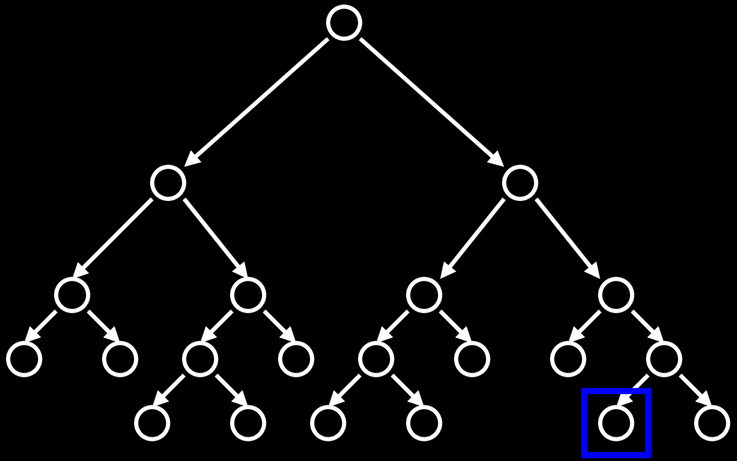




Then we can backtrack and try the other branch at each node visited.

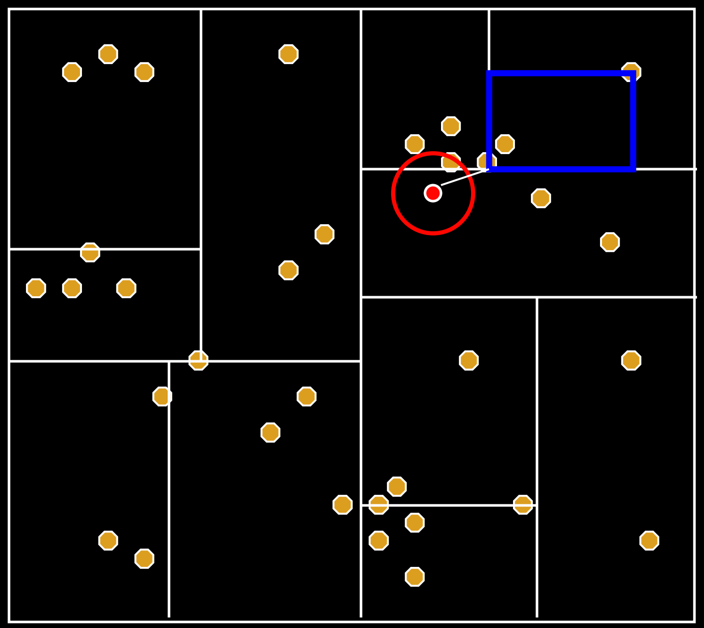
Nearest Neighbor with KD Trees

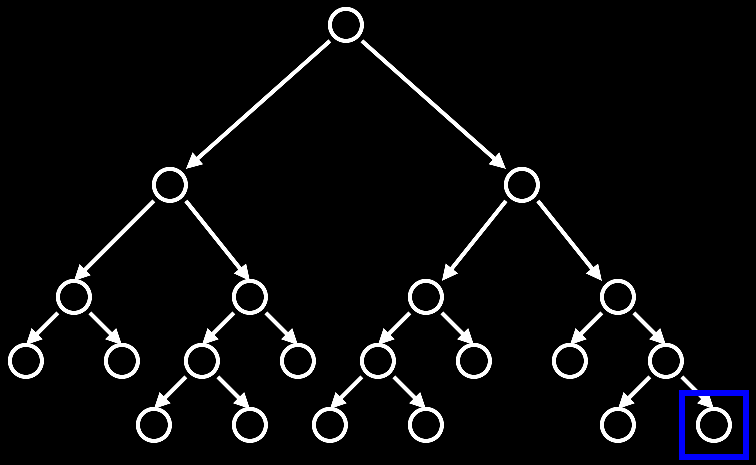




Each time a new closest node is found, we can update the distance bounds.

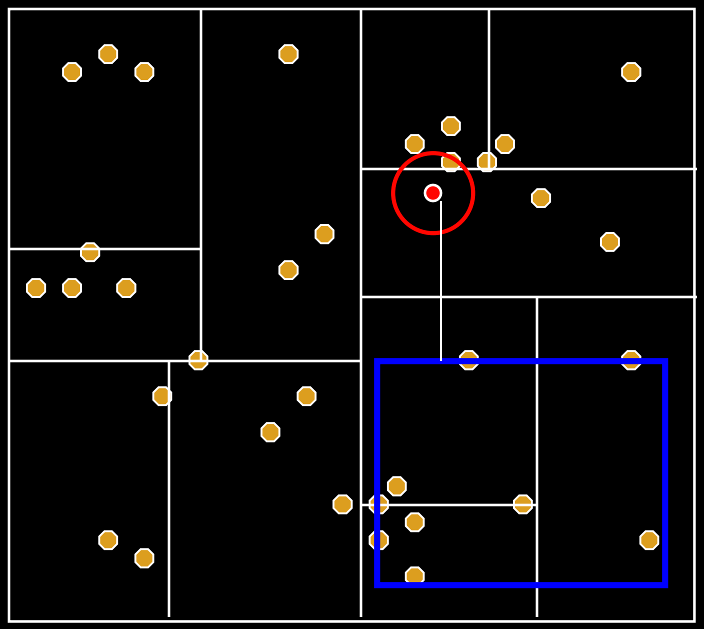
Nearest Neighbor with KD Trees

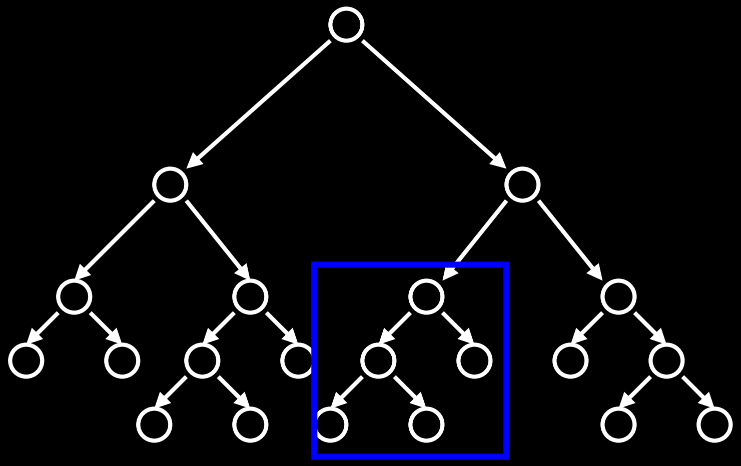




Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could NOT include the nearest neighbor.

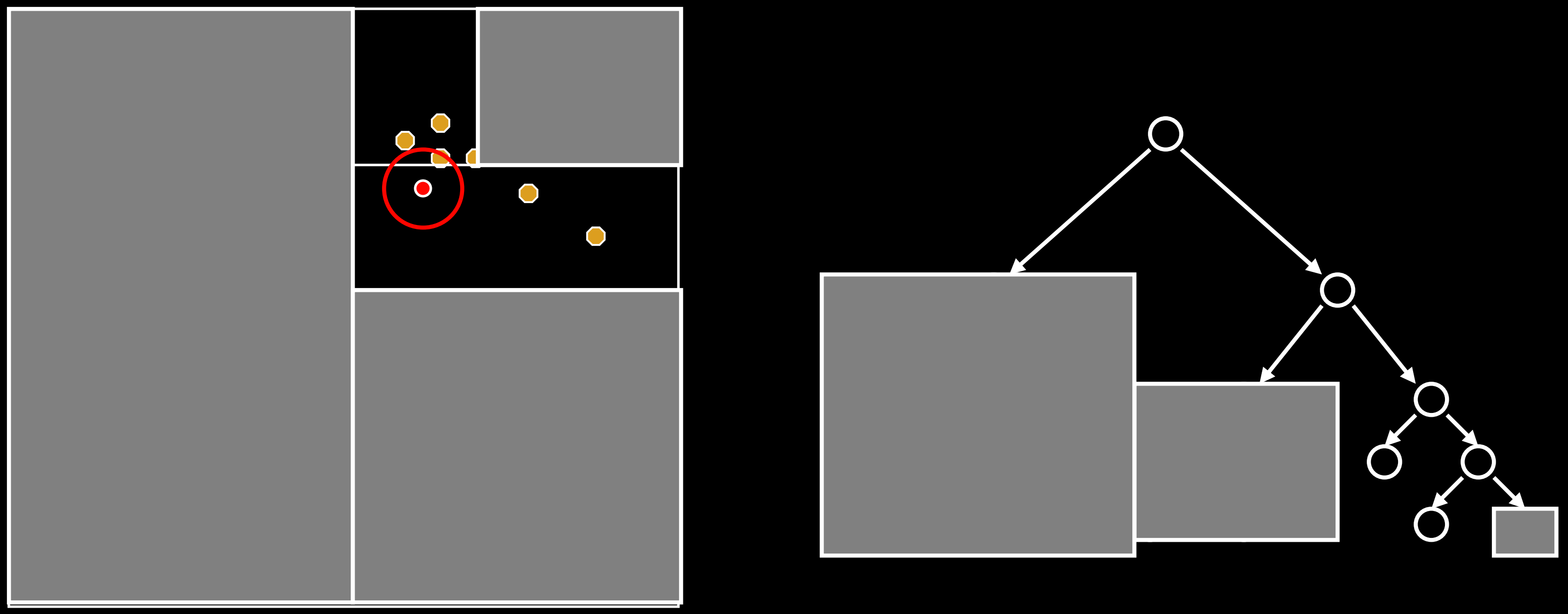
Nearest Neighbor with KD Trees





Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could NOT include the nearest neighbor.

Nearest Neighbor with KD Trees



Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could NOT include the nearest neighbor.

Using kd-trees

- kd-tree code is easy to find, don’t implement it yourself
- kd-tree is much faster than naïve nearest-neighbor for large numbers of nodes
- BUT, cost of constructing a kd-tree is significant, so only regenerate tree once in a while (not for every new node!)
- Maintain two structures:
  - List of new nodes (clear after 1000 nodes added)
  - kd-tree (rebuilt after every 1000 new nodes)
- For each nearest-neighbor call, query kd-tree **and** list of new nodes to find nearest-neighbors

Construction Step

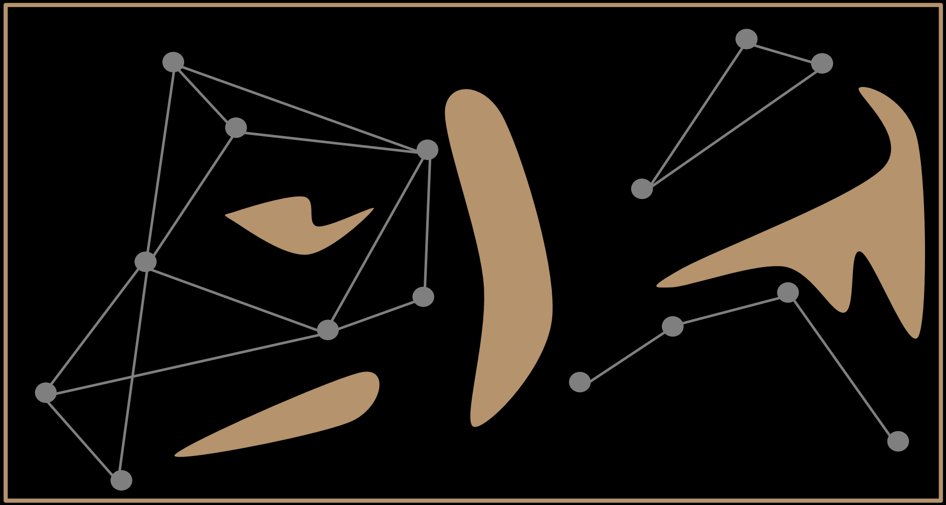
```
Start with an empty graph G = (V,E)
For i = 1 to MaxIterations
    Generate random configuration q ←
    If q is collision-free
        Add q to V
        Select k nearest nodes in V ←
        Attempt connection between each of these nodes and q using local planner
        If a connection is successful, add it as an edge in E
```

Local Planner

- In general, local planner can be anything that attempts to find a path between points, even another PRM!
- BUT, local planner needs to be fast b/c it's called many times by the algorithm
- Easiest and most common: Connect the two configurations with a straight line in C-space, check that the line is collision-free
  - Advantages:
    - Fast
    - Don't need to store local paths

Expansion step

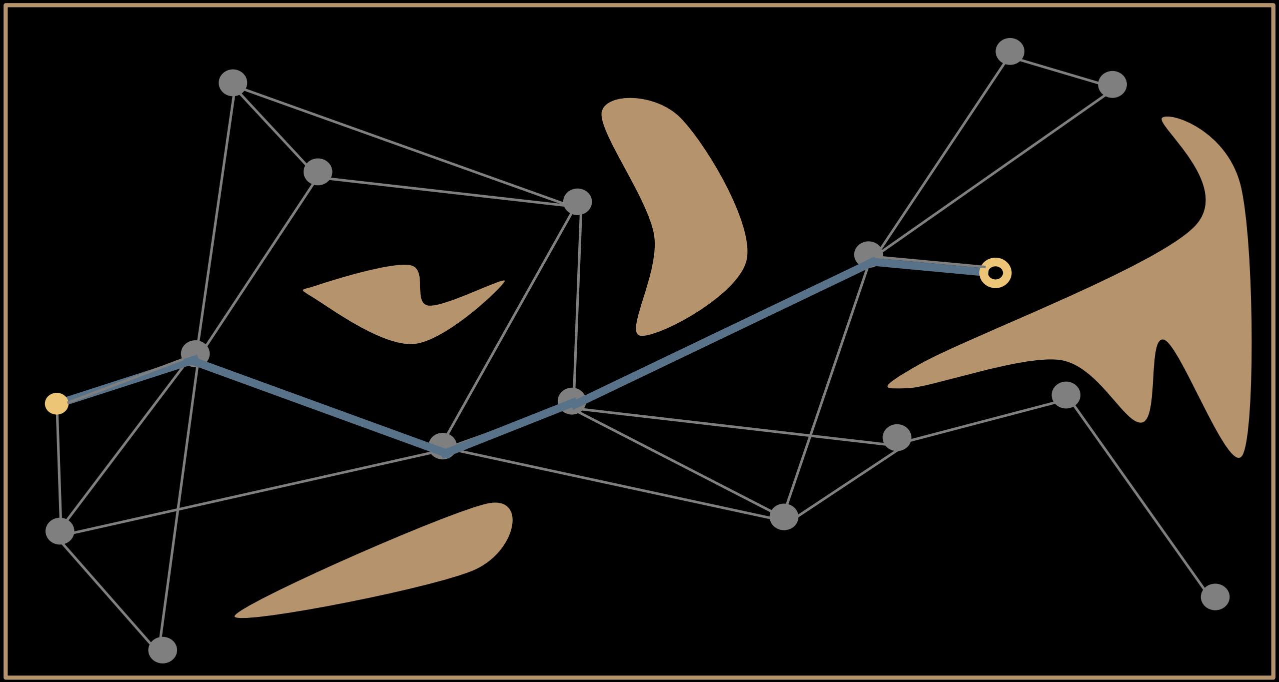
- Problem: Can have disconnected components that should be connected
    - I.e. you haven't captured the true connectivity of the space



- Expansion step uses heuristics to sample more nodes in an effort to connect disconnected components
    - Unclear how to do this the "right" way, very environment-dependent
    - See Kavraki et al. 1996 for one way to do this

Query Phase

- Given a start q_s and goal q_g
  1. Connect them to the roadmap using local planner
     - May need to try more than k nearest neighbors before connection is made
  2. Search G to find shortest path between q_s and q_g using A*/Dijkstra's/etc.

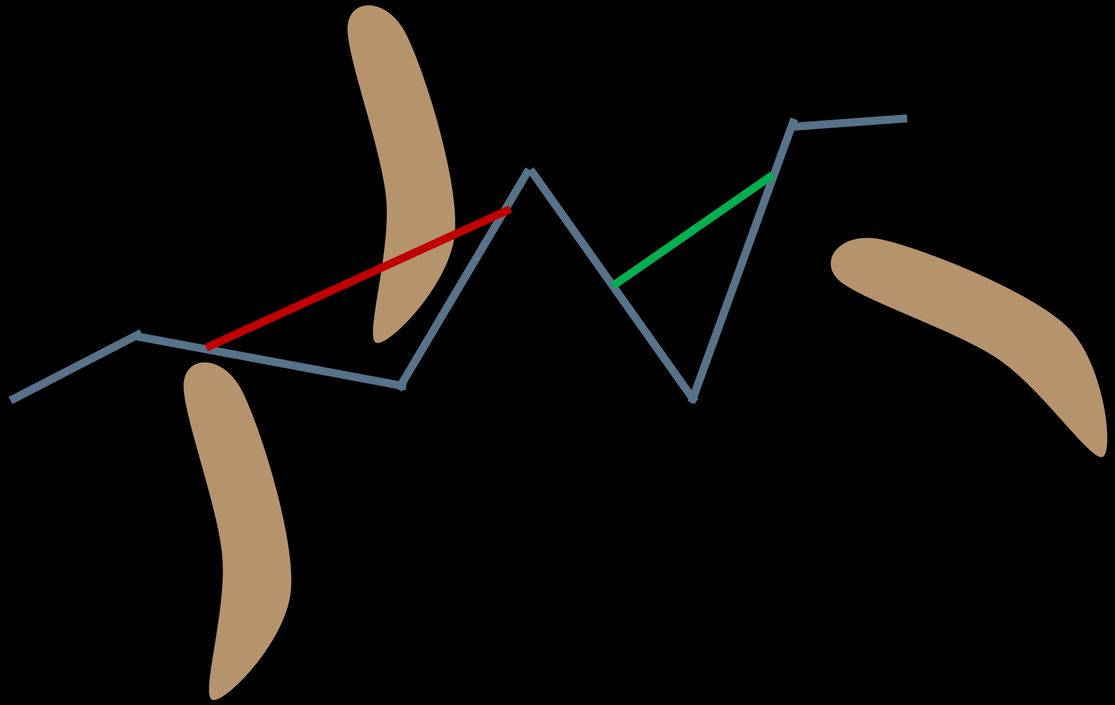


Path Shortening / Smoothing

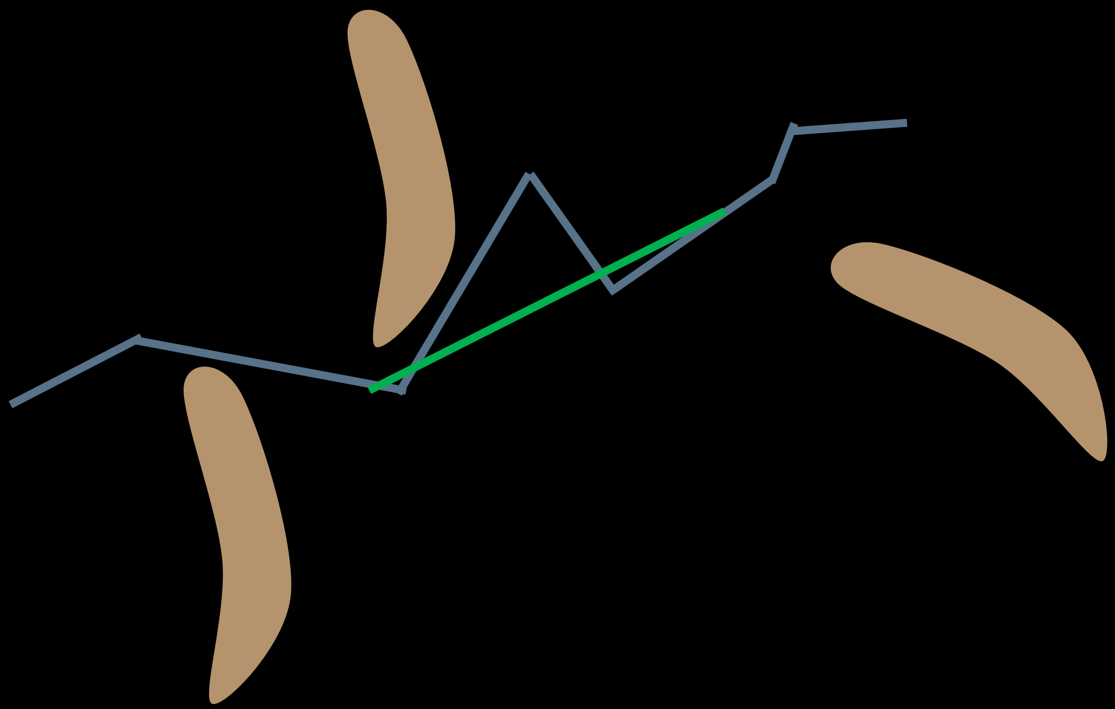
- **Don't even think of using a path generated by a sampling-based planner without smoothing it first!!!**
- "Shortcut" Smoothing

```
For i = 0 to MaxIterations
    Pick two points, $q_1$ and $q_2$, on the path randomly
    Attempt to connect ($q_1$, $q_2$) with a line segment
    If successful, replace path between $q_1$ and $q_2$ with the line segment
```

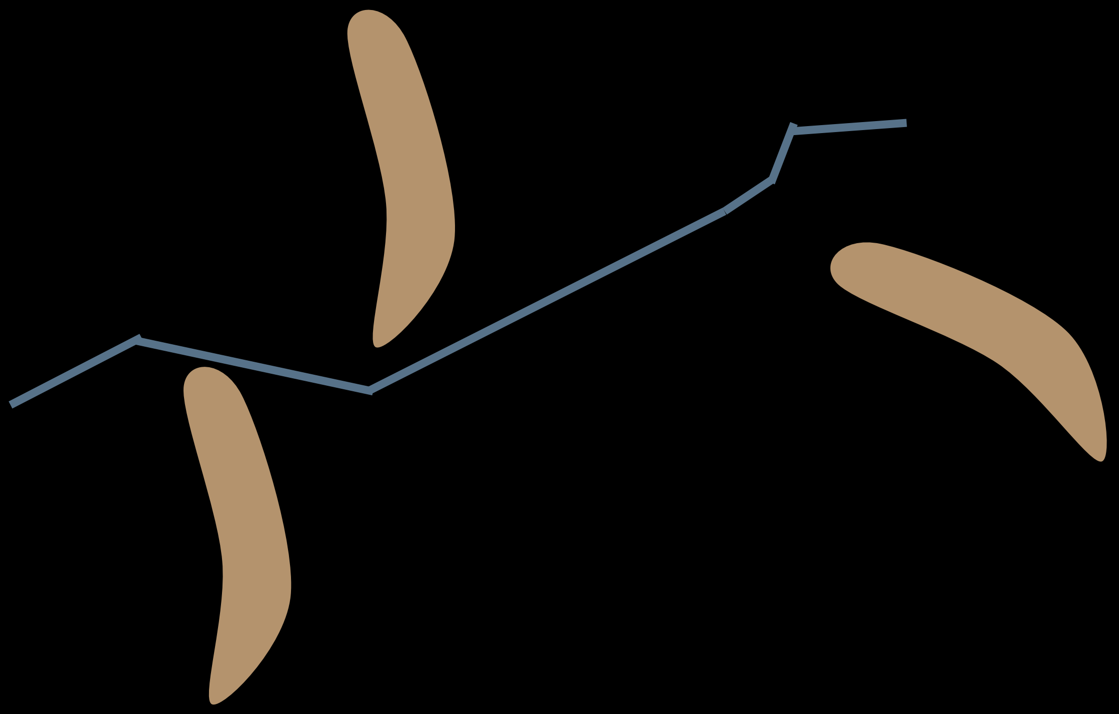
Shortcut Smoothing



Shortcut Smoothing

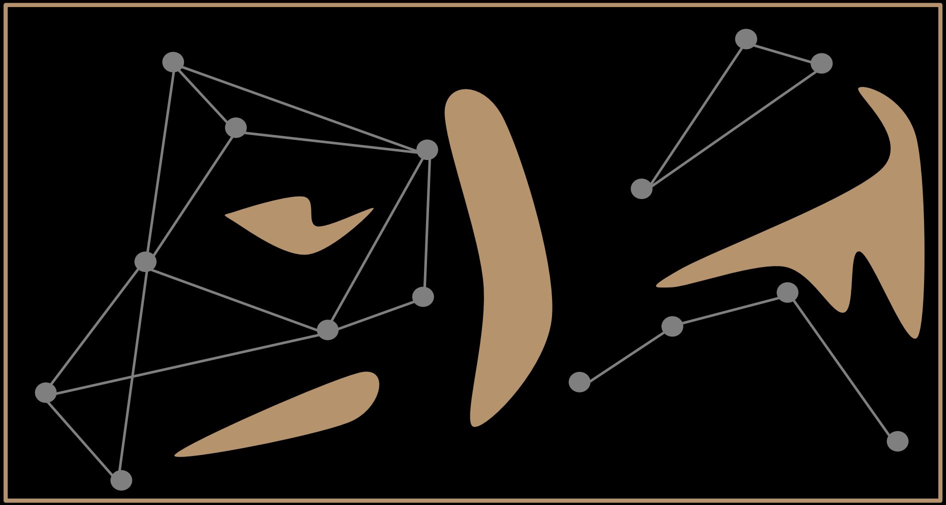


Shortcut Smoothing



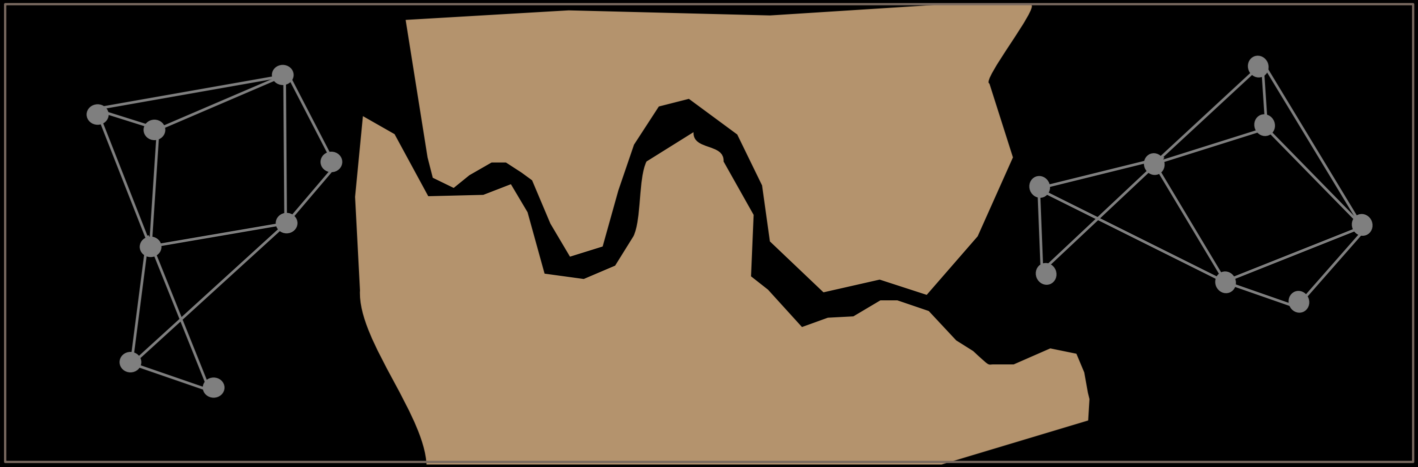
PRM Failure Modes

1. Can’t connect q_s and q_g to any nodes in the graph
    - Come up with an example in the graph below
2. Can’t find a path in the graph but a path is possible (more common)
    - Come up with in example in the graph below



Why do failures happen?

- Roadmap doesn't capture connectivity of space
  - Can run the learning phase longer
  - Can change sampling strategy to focus on narrow passages



- Local planner is too simple
  - Can use more sophisticated local planner

What happens in the limit?

- What if we ran the construction step of the PRM for infinite time…
  - What would the graph look like?
  - Would it capture the connectivity of the free space?
  - Would any start and goal be able to connect to the graph?
  - Is the PRM algorithm probabilistically complete?

Break

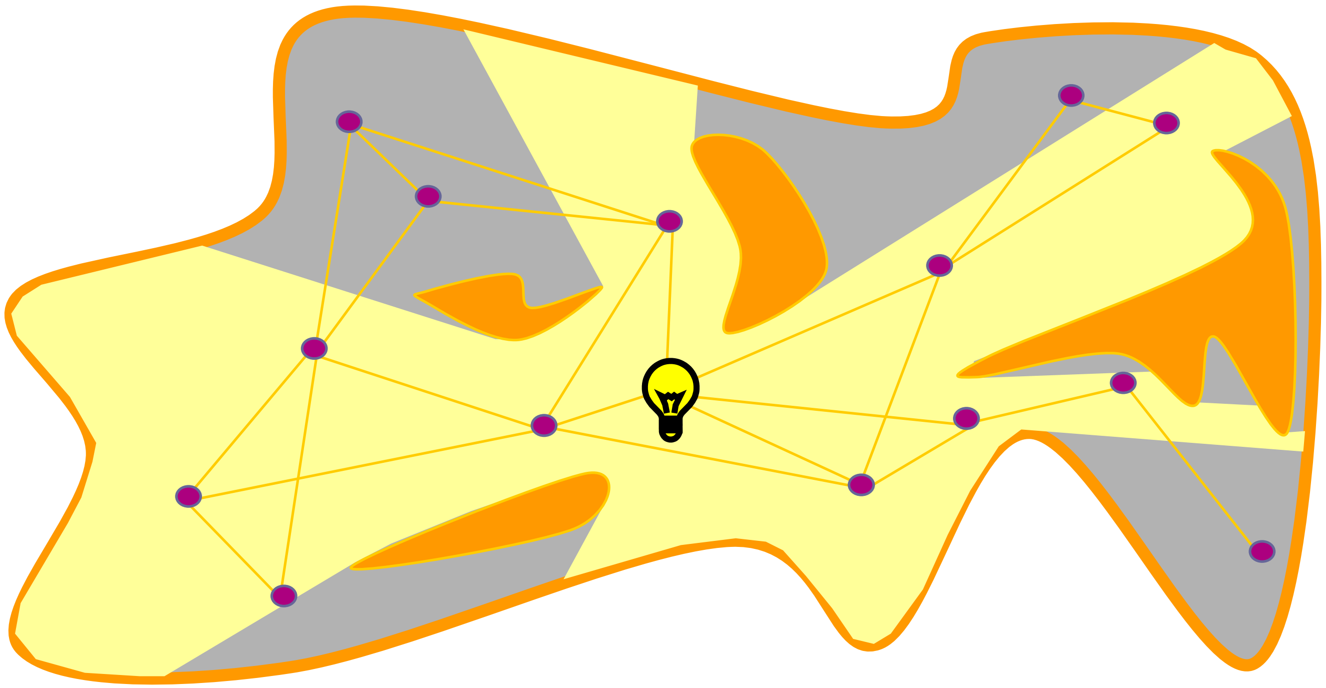
Expansive Spaces

Analysis of Probabilistic Roadmaps

Slides from David Hsu

Issues of probabilistic roadmaps

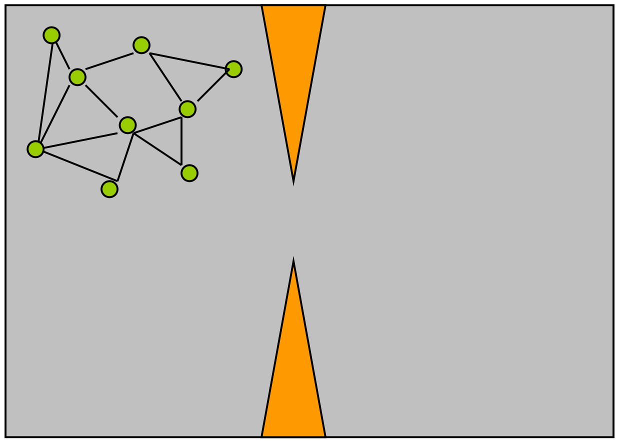
- Coverage
- Connectivity



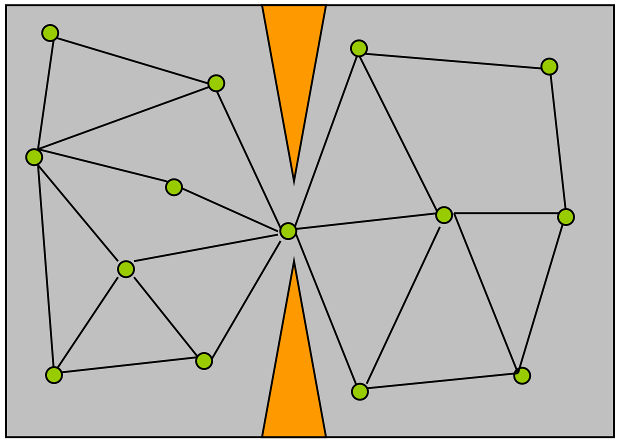
Is the coverage adequate?

- Milestones should be distributed so that almost **any** point of the configuration space can be connected by a straight line segment to one milestone.

Bad



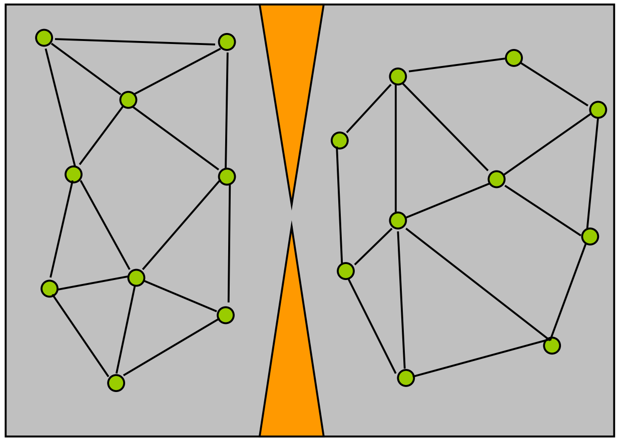
Good



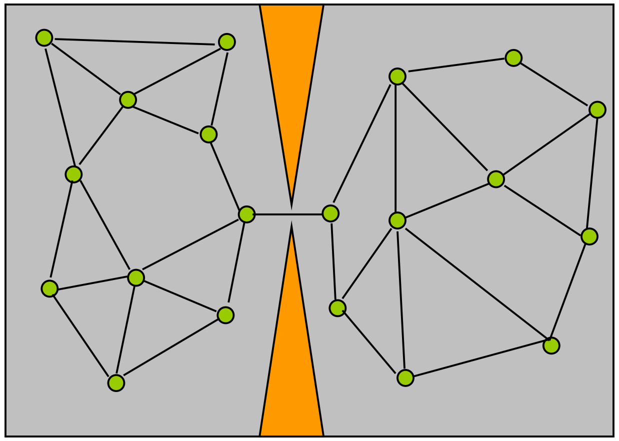
Connectivity

- There should be a one-to-one correspondence between the connected components of the roadmap and those of F (F is C_{free}).

Bad



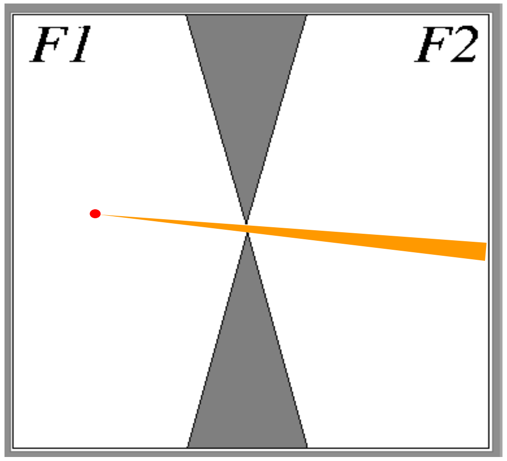
Good



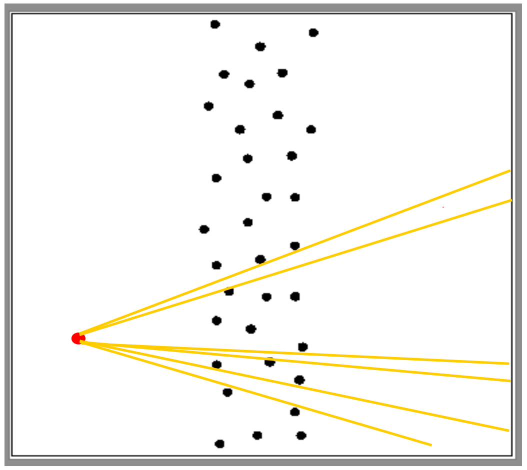
Narrow passages

- Connectivity is difficult to capture when there are narrow passages.
- Narrow passages are difficult to define.

difficult



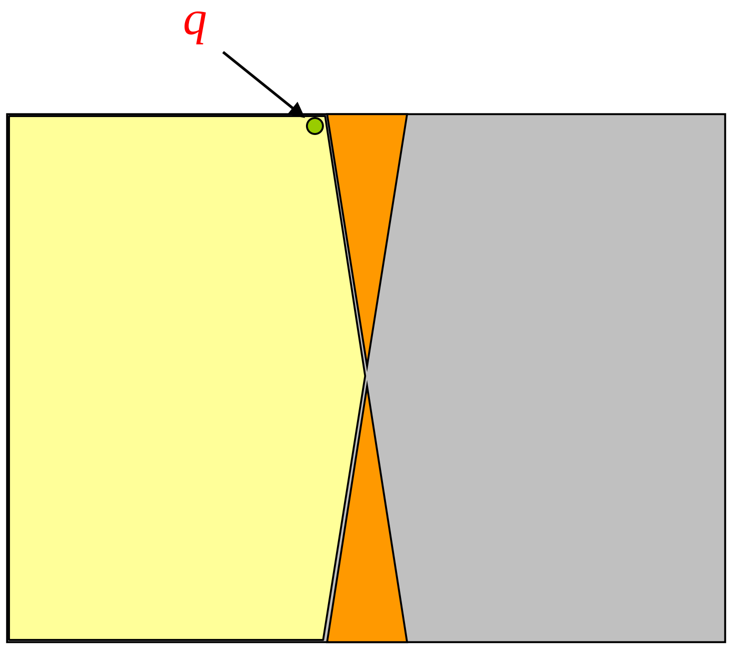
easy



- How to characterize coverage & connectivity?
  Expansiveness

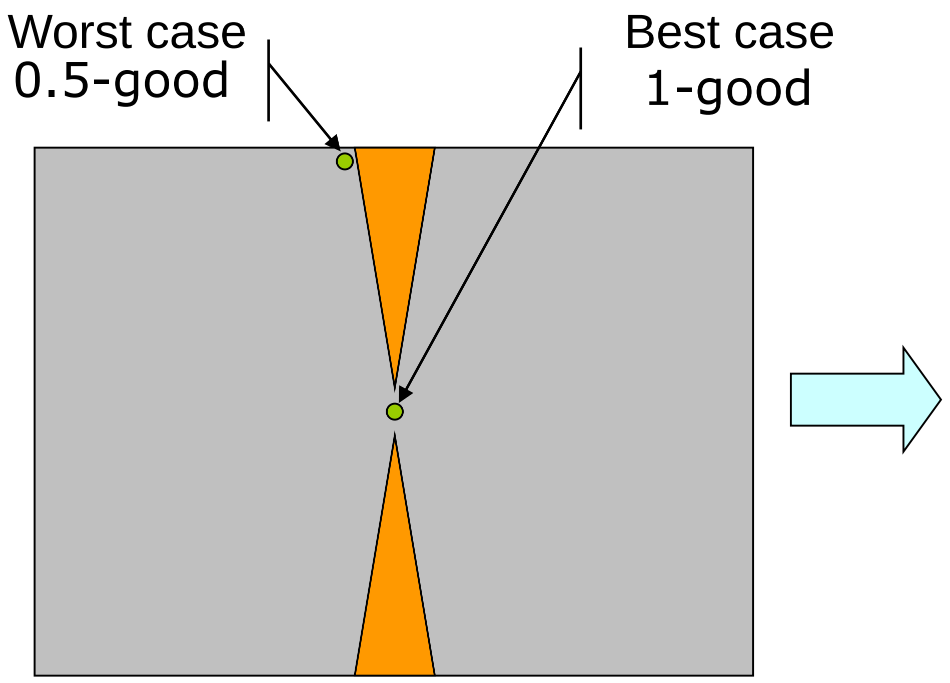
Definition: visibility set

- Visibility set of q
  - All configurations in F that can be connected to q by a straight-line path in F
  - All configurations “seen” by q



Definition: ϵ-good

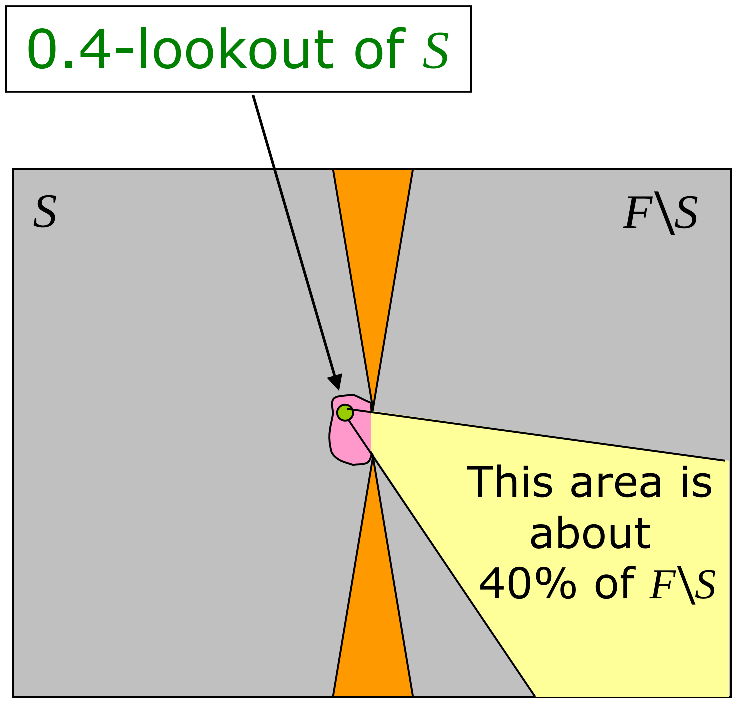
- Every free configuration sees at least ϵ fraction of the free space, ϵ in (0,1].

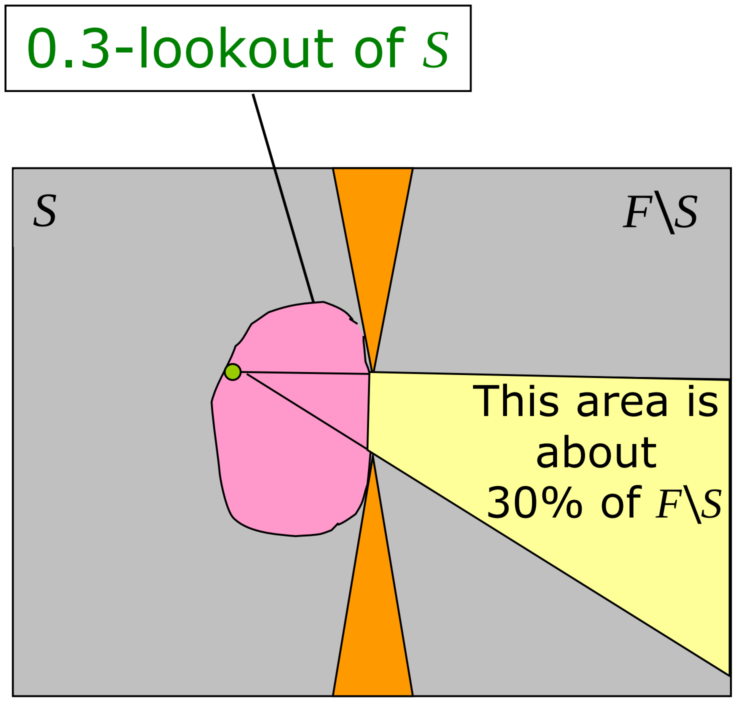


F is 0.5-good

Definition: lookout of a subset S

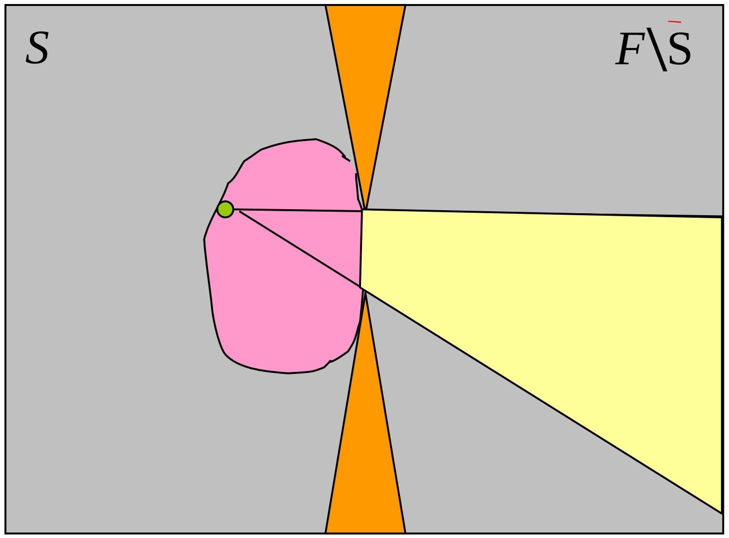
- Subset of points in S that can see at least β fraction of $F\backslash S$, β is in (0,1].





Definition: (ε, α, β)-expansive

- The free space *F* is $(\varepsilon,\alpha,\beta)$-expansive if
  - Free space *F* is ε-good
  - For each subset *S* of *F*, its β-lookout is at least α fraction of *S*. ε,α,β are in (0,1]



F is ε-good → ε=0.5

β-lookout → β=0.4

$$\frac{\text{Volume}(\beta\text{-lookout})}{\text{Volume}(S)} \rightarrow \alpha=0.2$$

F is (ε, α, β)-expansive, where ε=0.5, α=0.2, β=0.4.

Why expansiveness?

- ε, α, and β measure the expansiveness of a free space.
- Bigger ε, α, and β $\rightarrow$ easier to construct a roadmap with good connectivity and coverage.

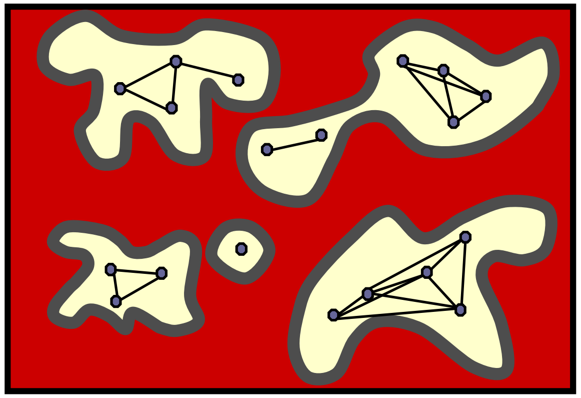
Theorem 1(Connectivity)

- Probability of achieving good connectivity increases exponentially with the number of milestones (in an expansive space).
- If ε, α, β decreases, then need to increase the number of milestones (to maintain good connectivity)

D. Hsu, J. C. Latombe and R. Motwani, "Path planning in expansive configuration spaces," *International Journal of Computational Geometry & Applications* Vol. 9, Nos. 4 & 5 (1999), pg. 495-512

Uniform sampling

- All-pairs path planning



- **Theorem 1** : A roadmap of $\frac{16\ln(1/\gamma)}{\varepsilon\alpha} + \frac{6}{\beta}$ uniformly-sampled milestones has the correct connectivity with probability at least $1-\gamma$.

Theorem 2 (Coverage)

- Probability of achieving good coverage, increases exponentially with the number of milestones (in an expansive space).

Completeness

- Complete algorithms are slow.
  - A **complete** algorithm finds a path if one exists and reports no otherwise.
  - Example: Visibility graph
- Heuristic algorithms are unreliable.
  - Example: potential field
- **Probabilistic completeness**
  - Intuition: If there is a solution path, the algorithm will find it with high probability.

Probabilistic completeness

In an expansive space, the probability that a PRM planner fails to find a path when one exists goes to 0 exponentially in the number of milestones (~ running time).

[Kavraki, Latombe, Motwani, Raghavan,95]
[Hsu, Latombe, Motwani, 97]

Summary

- Main result
  - If a C-space is expansive, then a roadmap can be constructed efficiently with good connectivity and coverage.
- Limitations in practice
  - It does not tell you when to stop growing the roadmap.
  - A planner stops when either a path is found or max steps are reached.
  - $\varepsilon, \alpha, \beta$ are nearly impossible to compute in high-dimensional spaces

Homework

- HW2 is posted!
- Make sure you’ve read Chapter 5