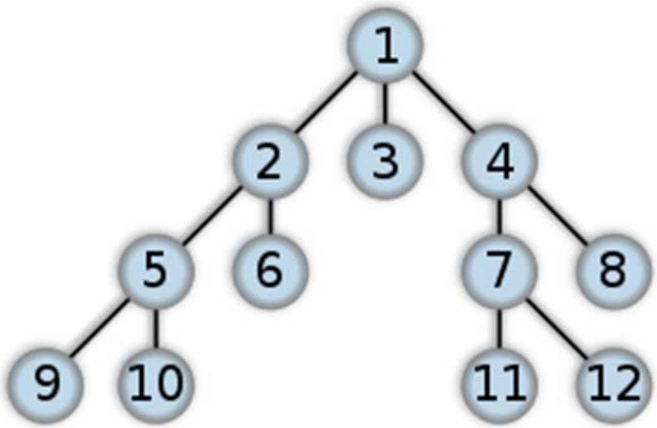
Sampling-based Planning 1

Jane Li
Assistant Professor
Mechanical Engineering & Robotics Engineering
http://users.wpi.edu/~zli11

Recap

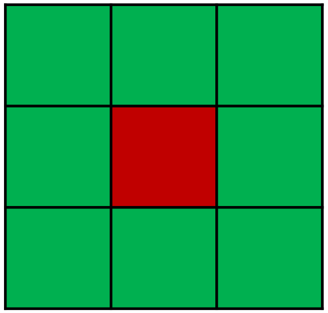
- **Discrete planning** is best suited for
  - **Low-dimensional** motion planning problems
  - Problems where the **control set** can be **easily discretized**



- What if we need to plan in **high-dimensional** spaces?

Discrete Planning – Limitation

- 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?

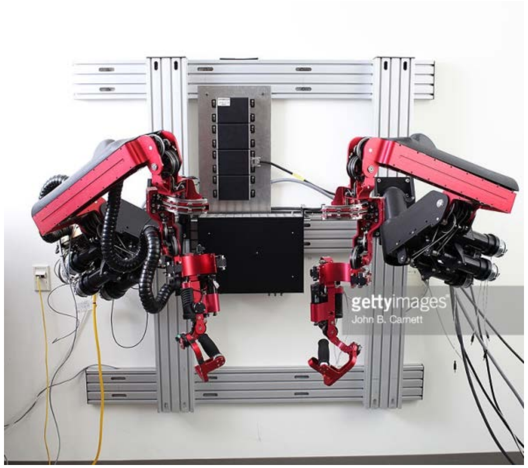


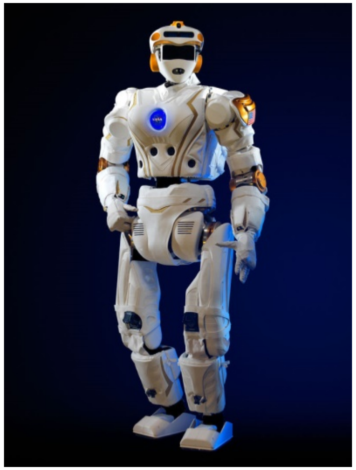
8-connected

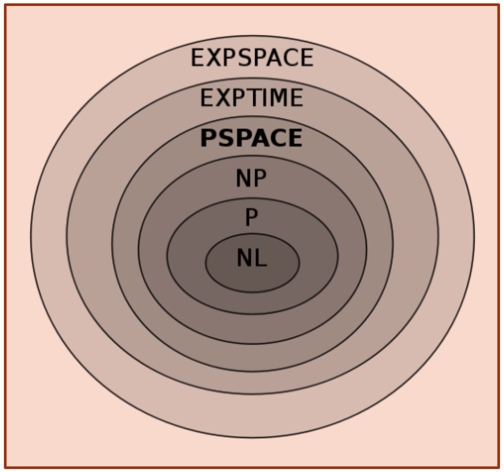
Motivation

- Need a path planning method not so sensitive to dimensionality
- Challenges:
  - 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!

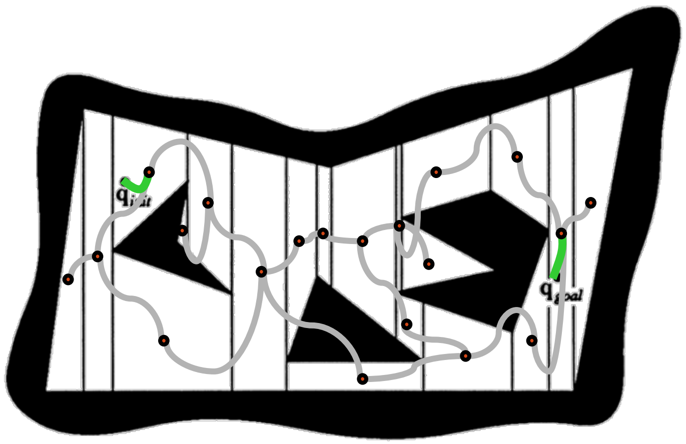
Weakening Requirements

- Probabilistic completeness
  - Given a solvable problem, 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

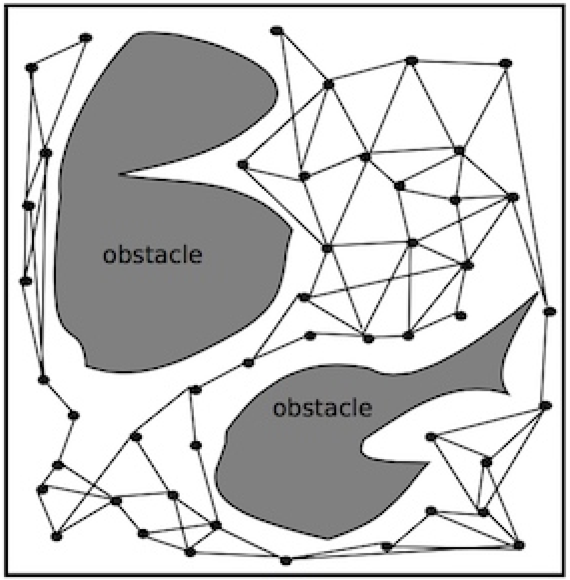
Ideal		**Practical in High Dimensions**
Complete	→	Probabilistically Complete
Optimal	→	Feasible
		*Recent methods show asymptotic optimality

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



Discrete planning



Sampling-based planning

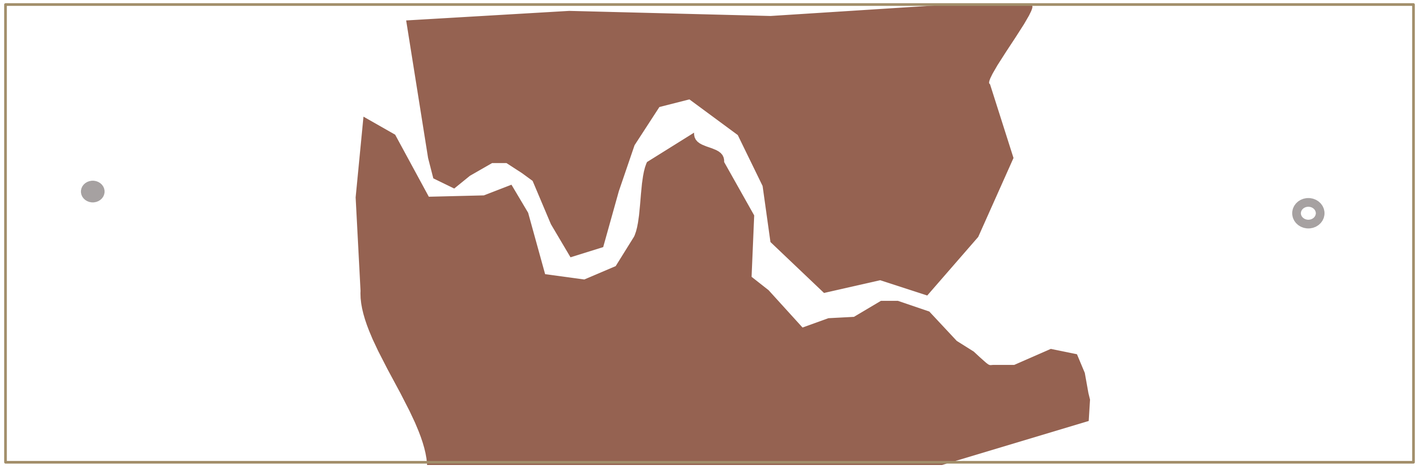
Comparison

Advantages

- Don't need to **discretize** C-space
- Don't need to **explicitly represent** C-space
- Not sensitive to C-space dimension

Disadvantages

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

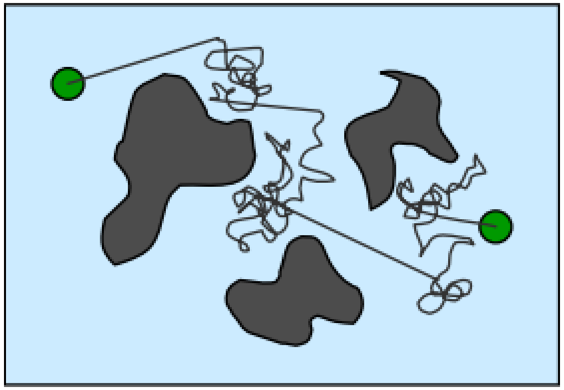


Outline

- Randomized Path Planner (RPP)
- Probabilistic Roadmap (PRM)
  - Construct and Search in PRM
  - Performance
    - Coverage, connectivity and completeness

Randomized Path Planner (RPP)

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



Barraquand and Latombe in 1991 at Stanford

Randomized Path Planner (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)

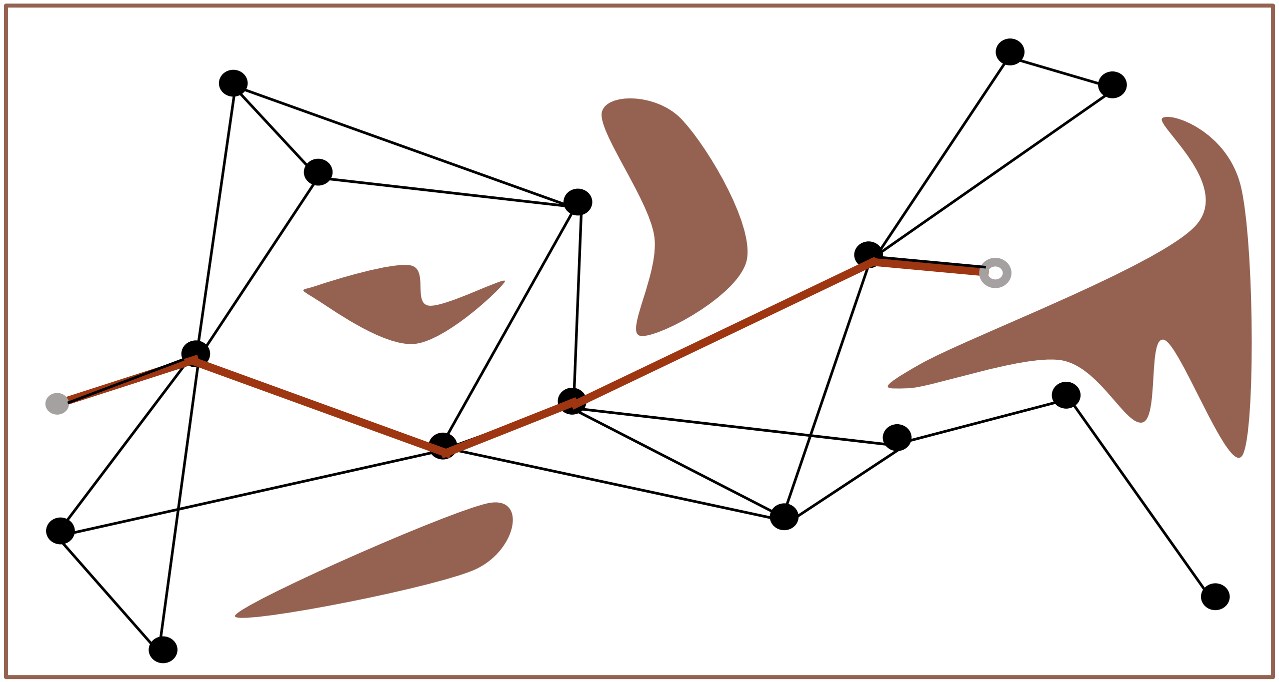
- Main idea:
  - **Build** a roadmap of the space from sampled points
  - **Search** the roadmap to find a path
- Roadmap should capture the **connectivity** of the free space

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

Probabilistic Roadmap (PRM)

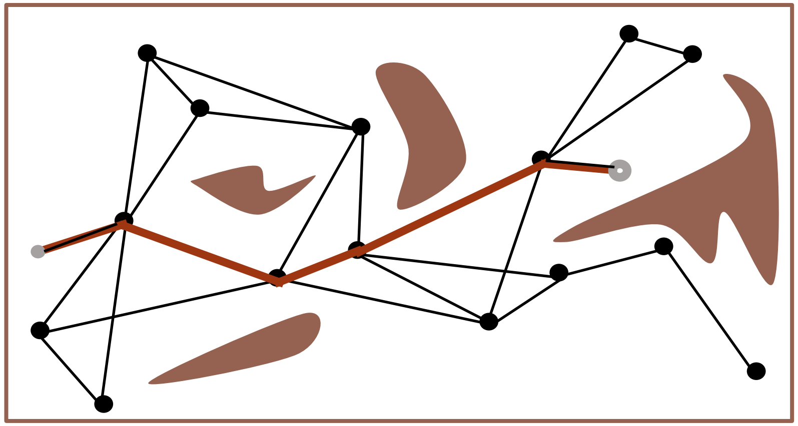
- PRM – Two steps
  - "Learning" Phase
    - Construction Step
    - Expansion Step
  - Query Phase
    - Answer a given path planning query
- PRMs are known as **multi-query** algorithms,
  - Roadmap can be **re-used** if environment and robot/envrionment remain **unchanged** between queries.

Example



"Learning" Phase

- Construction step:
  - Build the roadmap by sampling (random) free configurations
  - Connect them using a fast *local planner* – collision checking
  - Store these configurations as nodes in a **graph**
    - 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

Sampling Collision-free Configurations

- Uniform random sampling in C-space
  - Easiest and most common
  - AKA "(Acceptance)-Rejection Sampling"
- Steps
  - 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
- MANY other ways to sample …

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)
    - $D(q_1,q_2)$ is **small** → success should be **likely**
    - $D(q_1,q_2)$ is **large** → success should be **less likely**
- By default, use Euclidian distance:

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

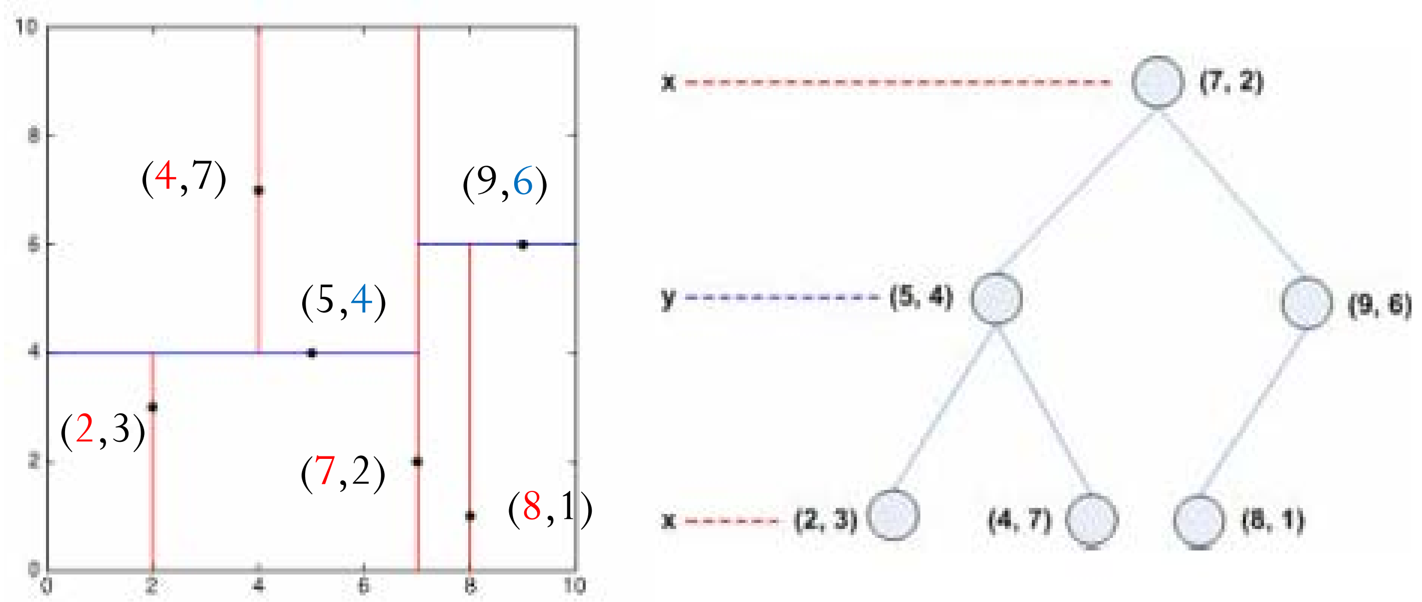
- Can weigh different dimensions of C-space differently
  - Often used to weigh translation vs. rotation

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 thousands of nodes
    - use *kd-tree* to store nodes and do NN queries

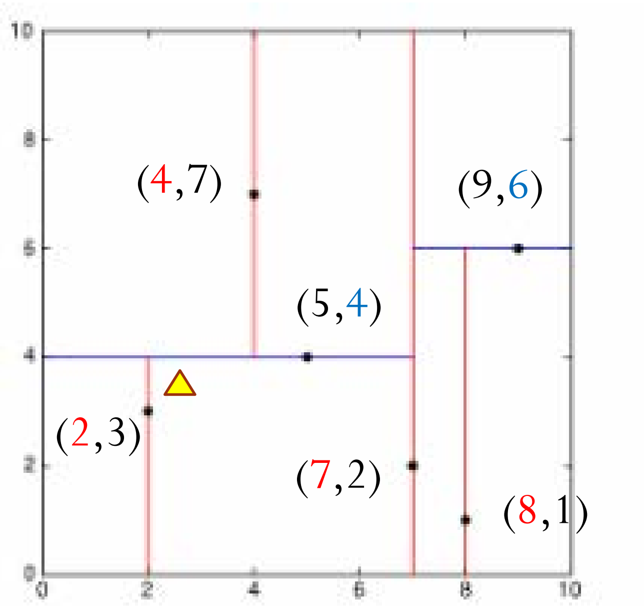
KD-trees

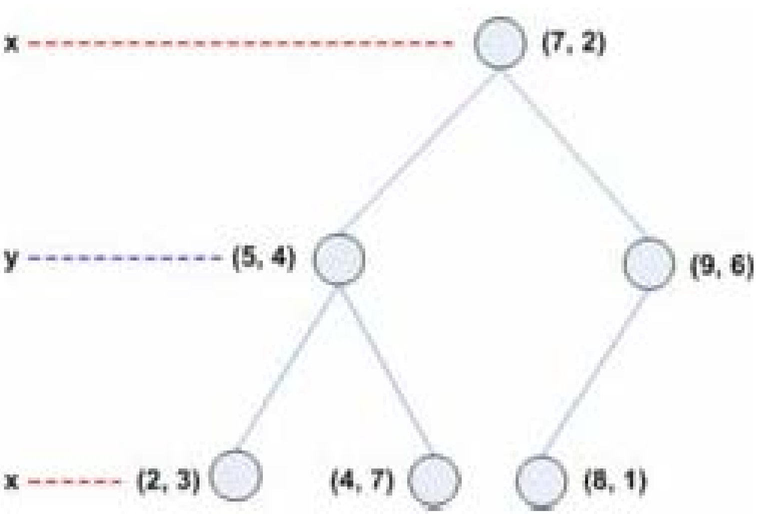
- A kd-tree
  - a data-structure that recursively divides the space into bins that contain points (like Oct-tree and Quad-tree)
  - NN searches through bins (not individual points) to find nearest point



Search in KD-Tree for Nearest Neighbor

- Goal – Find the closest point to the query point, in a 2D tree
  - Check the distance from the node point to query point
  - Recursively search if a subtree contains a closer point





KD-tree

- Performance
  - Much **faster** to use kd-tree for **large** numbers of nodes
  - **Cost** of constructing a kd-tree is **significant**
    - Only regenerate tree once in a while (not for every new node!)
- Implementation
  - kd-tree code is easy to find online

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 <u>local planner</u>

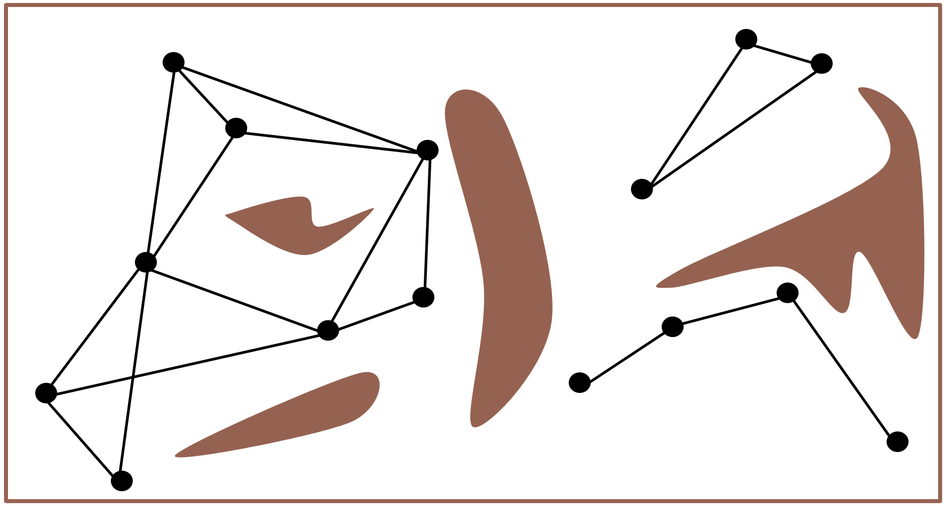
If a connection is successful, add it as an edge in E

Local Planner

- In general, local planner can be **anything** that attempts to find a path between points,
  - Even another PRM!
- Local planner needs to be **fast**
  - 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 line is collision-free
  - Advantages:
    - Fast
    - Don't need to store local paths

Expansion Step

- Problem – 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**

Possible ways to measure the connection difficulty?

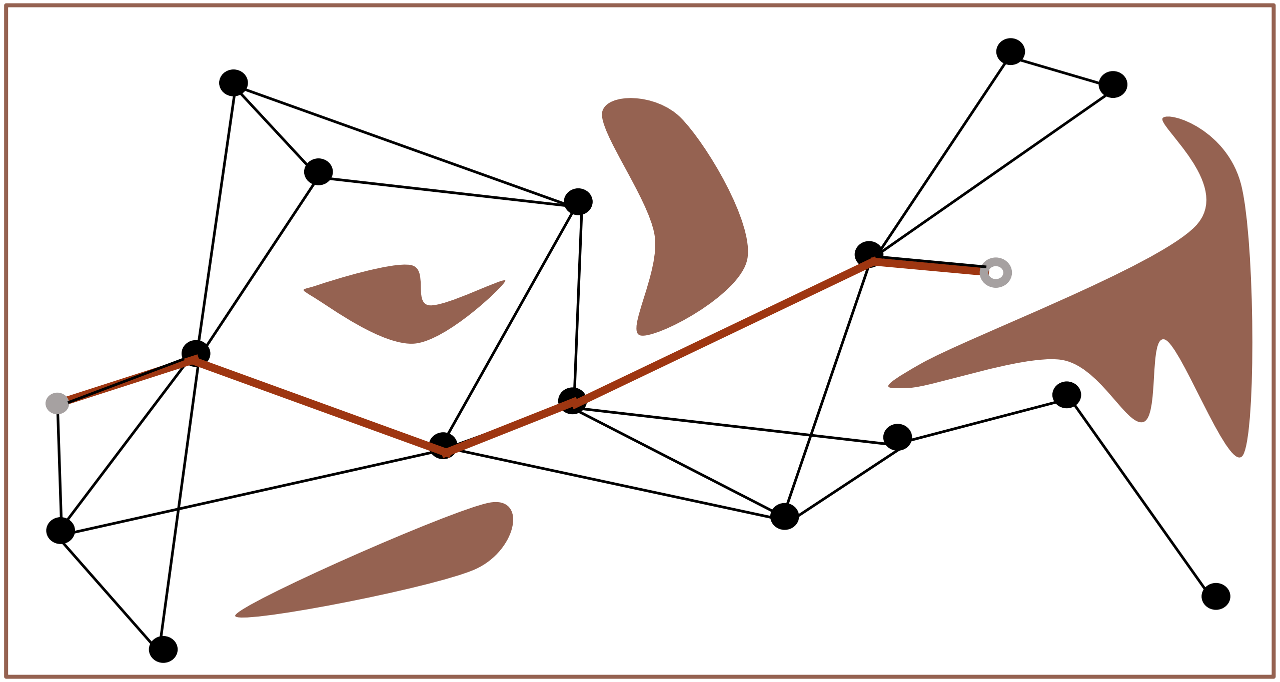
Possible Heuristics

- # of Nodes nearby
  - For a node **c**, count the # of nodes **N** within a predefined distance
  - Uniform random sampling,
  - N is small → obstacle region may occupy large portion of **c**'s neighborhood
  - Heuristics = 1/**N**
- Distance to nearest reacheable neighbor
  - For a node **c**, find the distance **d** to the nearest connected component that doesn't contains this node
  - **d** is small → **c** lies in the region where two components fail to connect
  - Heuristics = 1/d
- Others
  - Behavior of local planner?
  - Always fail to connect → difficult region

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.

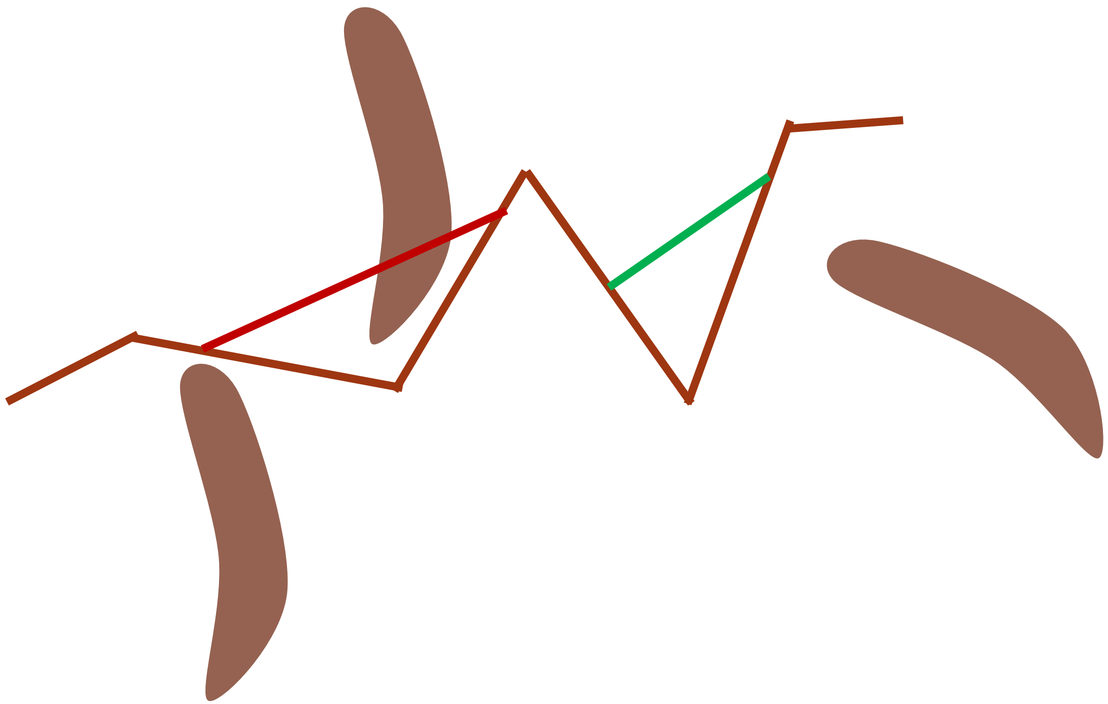


Path Shortening / Smoothing

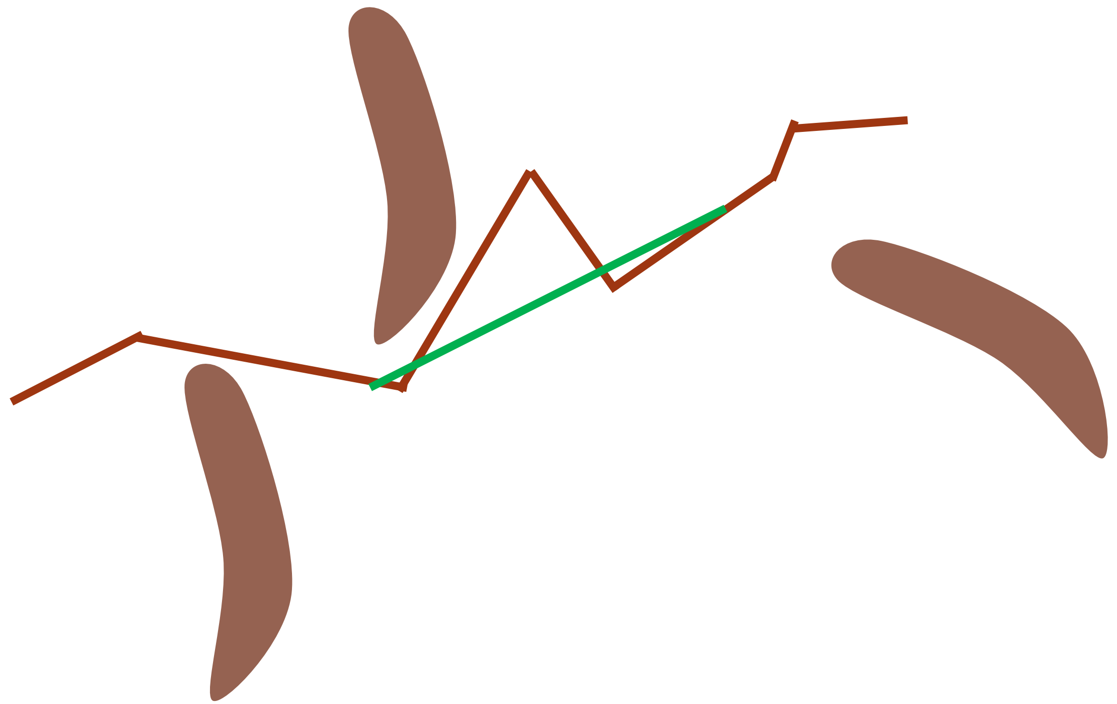
- **Never** use a path generated by a sampling-based planner without smoothing it!!!
- "Shortcut" Smoothing

> - For i = 0 to MaxIterations
> - Pick two points, q1 and q2, on the path randomly
> - Attempt to connect (q1, q2) with a line segment
> - If successful, replace path between q1 and q2 with the line segment

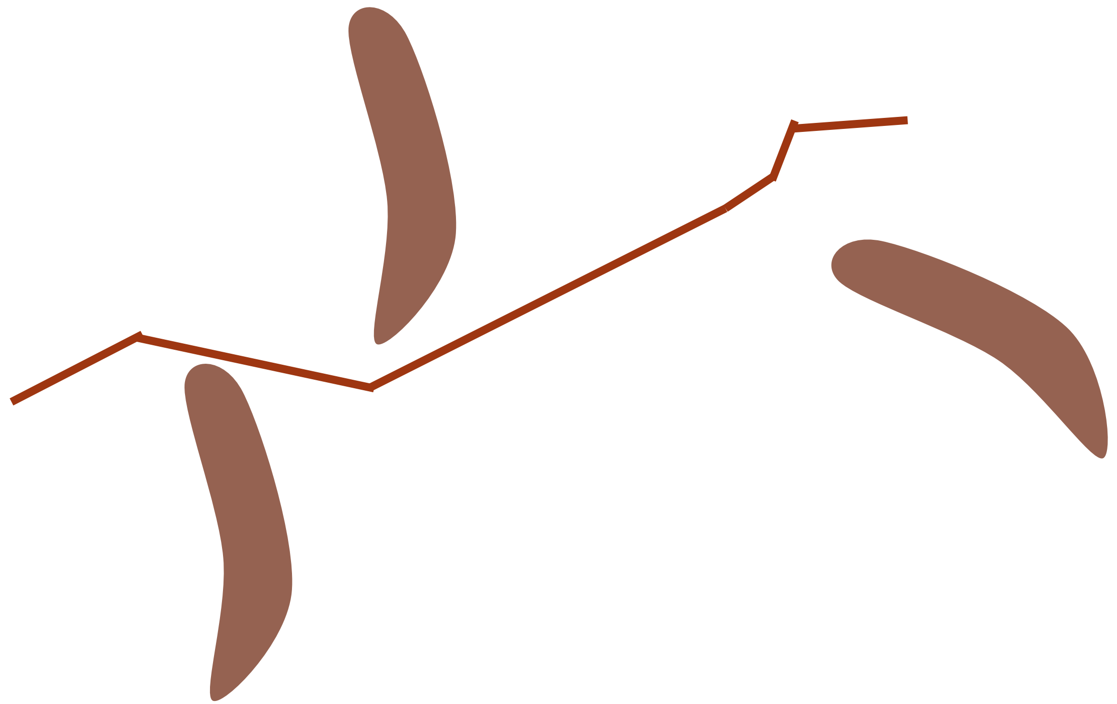
Shortcut Smoothing



Shortcut Smoothing

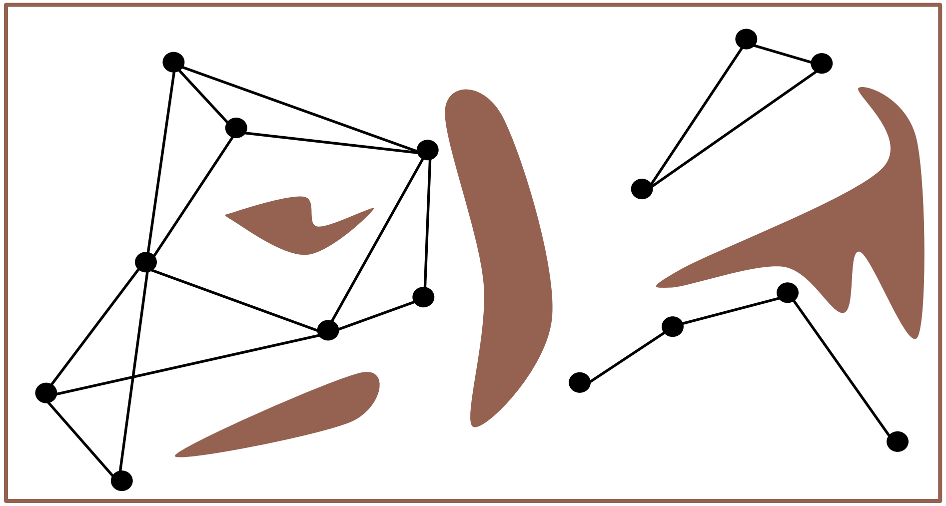


Shortcut Smoothing



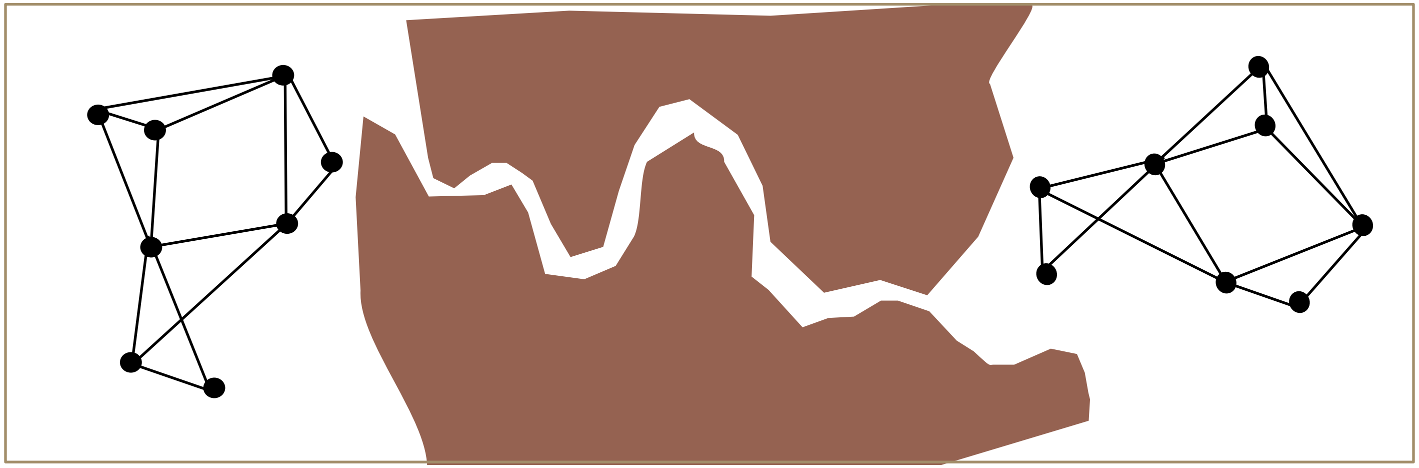
PRM Failure Modes

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
   - Come up with in example in the graph below



Why do failures happen?

- Local planner is too simple
  - Can use more sophisticated local planner
- Roadmap doesn't capture connectivity of space
  - Can run the learning phase longer
  - Can change **sampling strategy** to focus on narrow passages

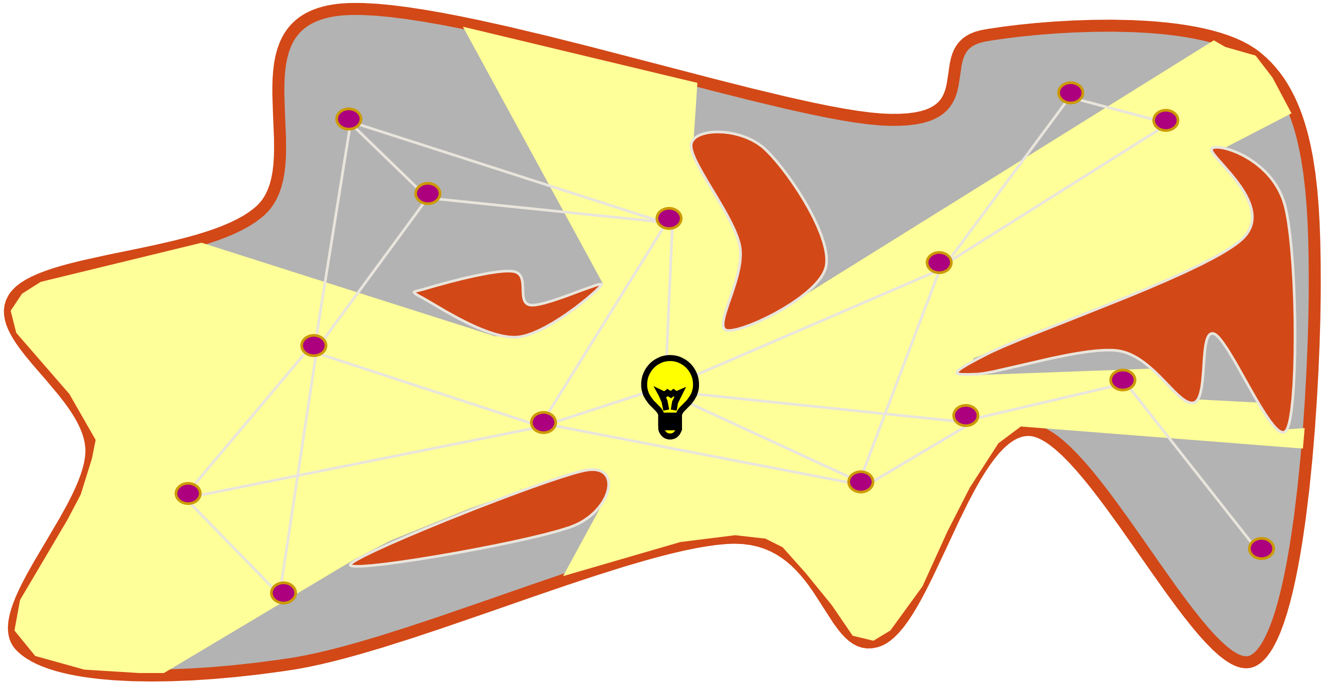


What happens in the limit?

- 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?

Issues of Probabilistic Roadmaps

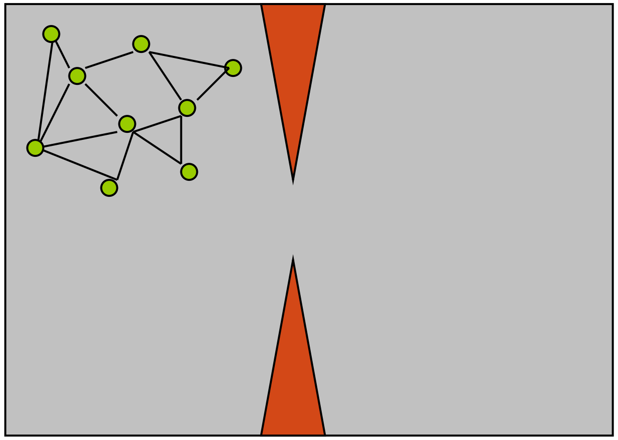
- Coverage
- Connectivity



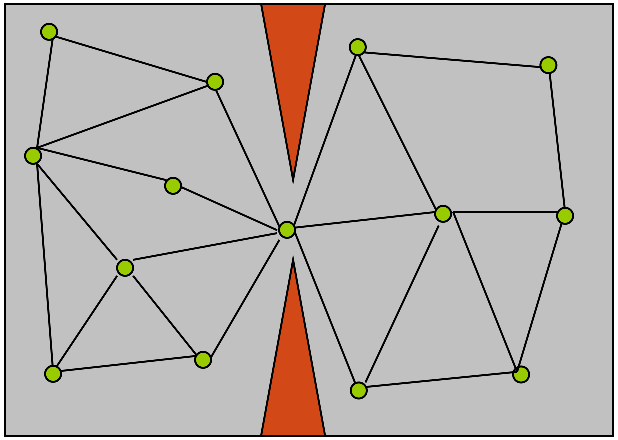
Is the Coverage Adequate?

- 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

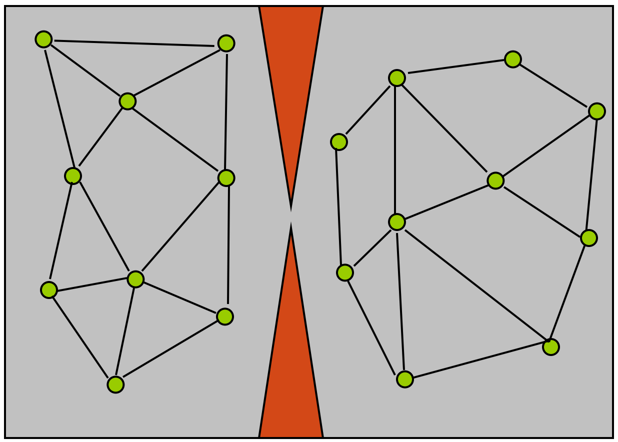


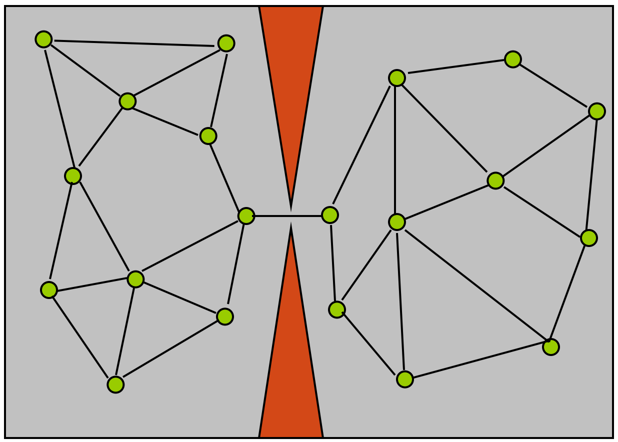
Good



Connectivity

- There should be a **one-to-one correspondence** between the connected components of the roadmap and those of the field *F*.

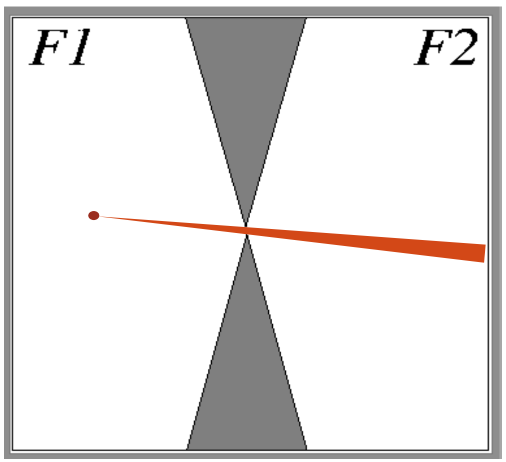




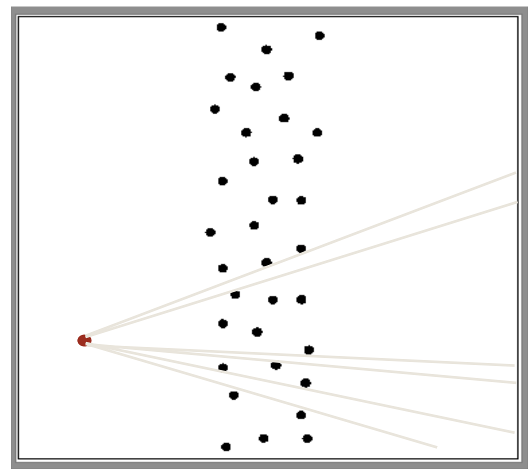
Narrow Passages

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

difficult



easy

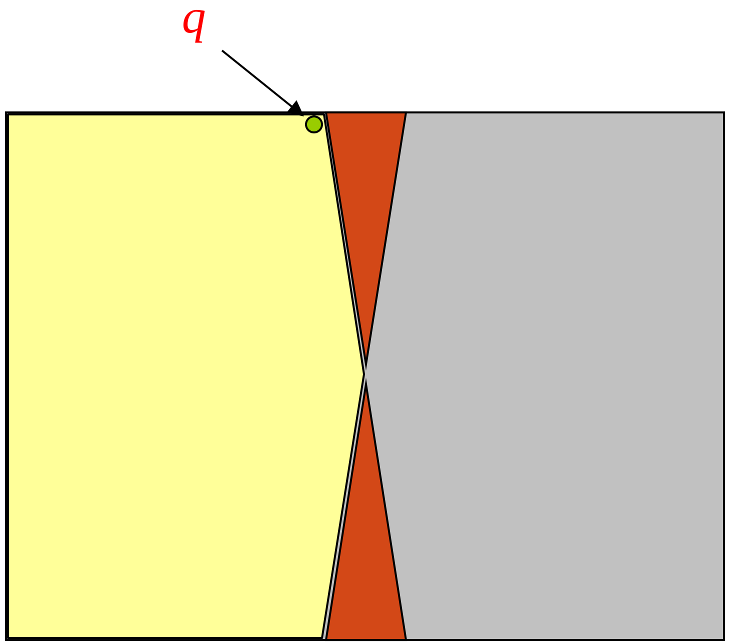


- How to characterize coverage & connectivity?

Expansiveness

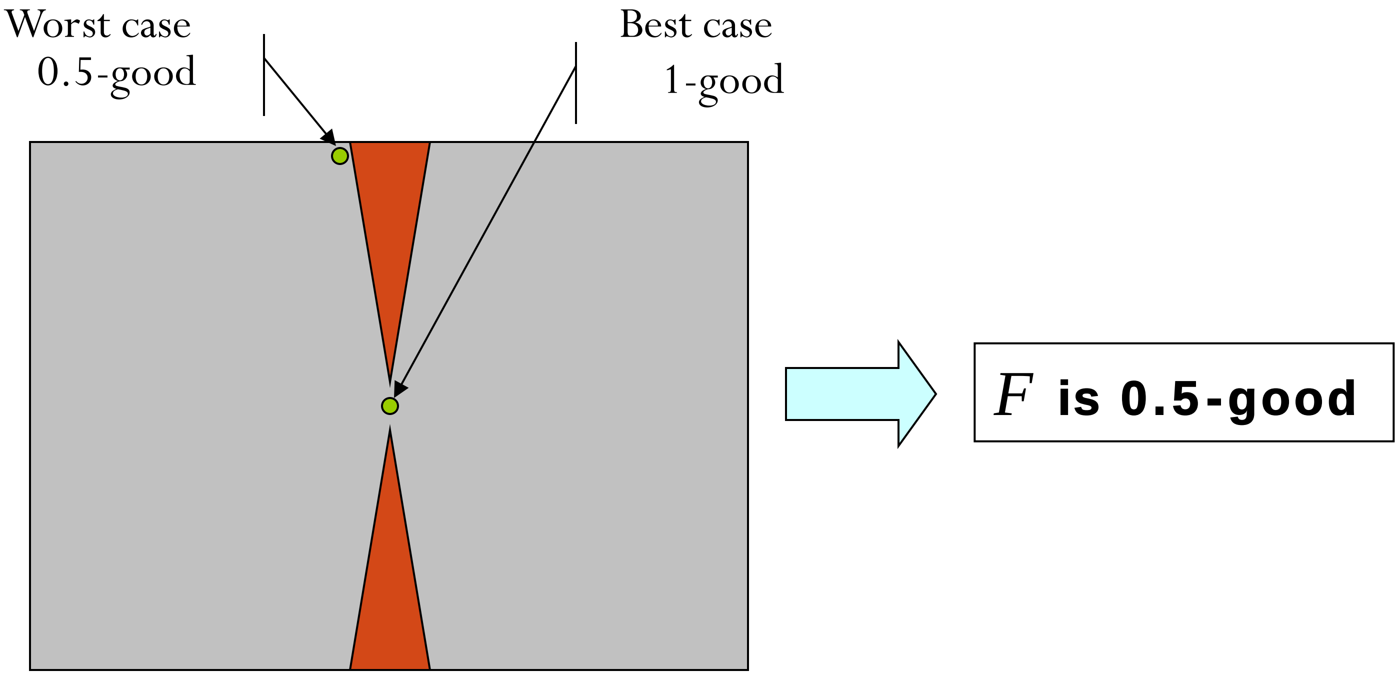
Definition: Visibility Set

- 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



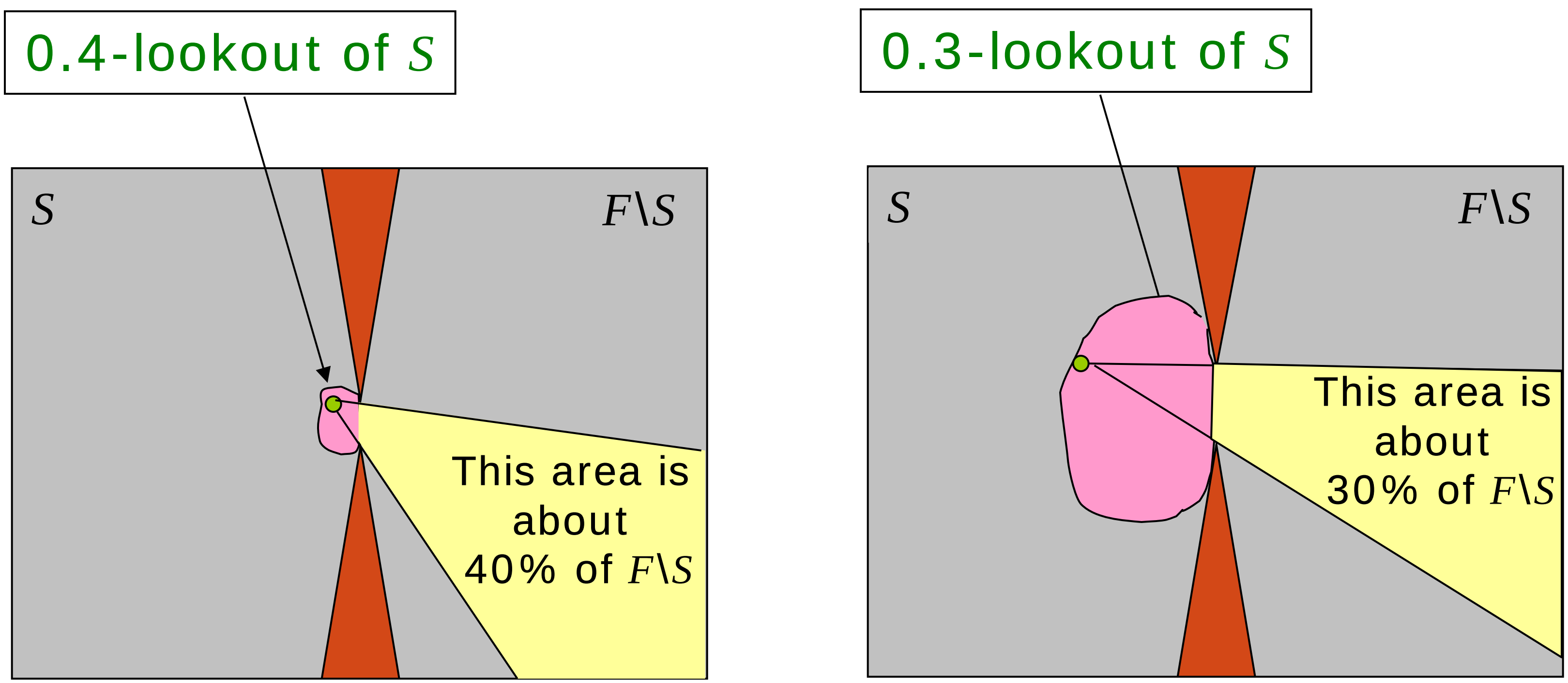
Definition: Є-good

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



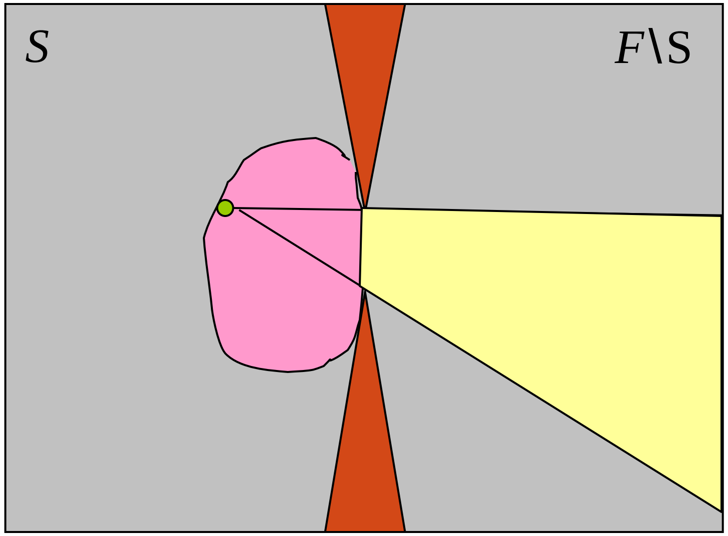
Definition: Lookout of a Subset S

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



Definition: $(\varepsilon, \alpha, \beta)$ - 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. $\varepsilon, \alpha, \beta$ are in (0,1]



F is ε-good → $\varepsilon=0.5$

β-lookout → $\beta=0.4$

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

F is $(\varepsilon, \alpha, \beta)$-expansive, where $\varepsilon=0.5$, $\alpha=0.2$, $\beta=0.4$.

Why Expansiveness?

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

Why Expansiveness?

- Connectivity
  - Probability of achieving good connectivity increases exponentially with the number of milestones (in an expansive space).
  - If $\varepsilon, \alpha, \beta$ decreases, then need to increase the number of milestones (to maintain good connectivity)
- 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.
- Heuristic algorithms are unreliable.
  - Example: potential field
- **Probabilistic completeness**
  - Intuition – If there is a solution path, the algorithm will find it with high probability.
  - 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).

Readings

- Read "How to Read a Paper" guide (link on website)

Lazy Collision Checking

- Eliminate the majority of collision checks using a lazy strategy