Sampling-based Planning 01

Jane Li

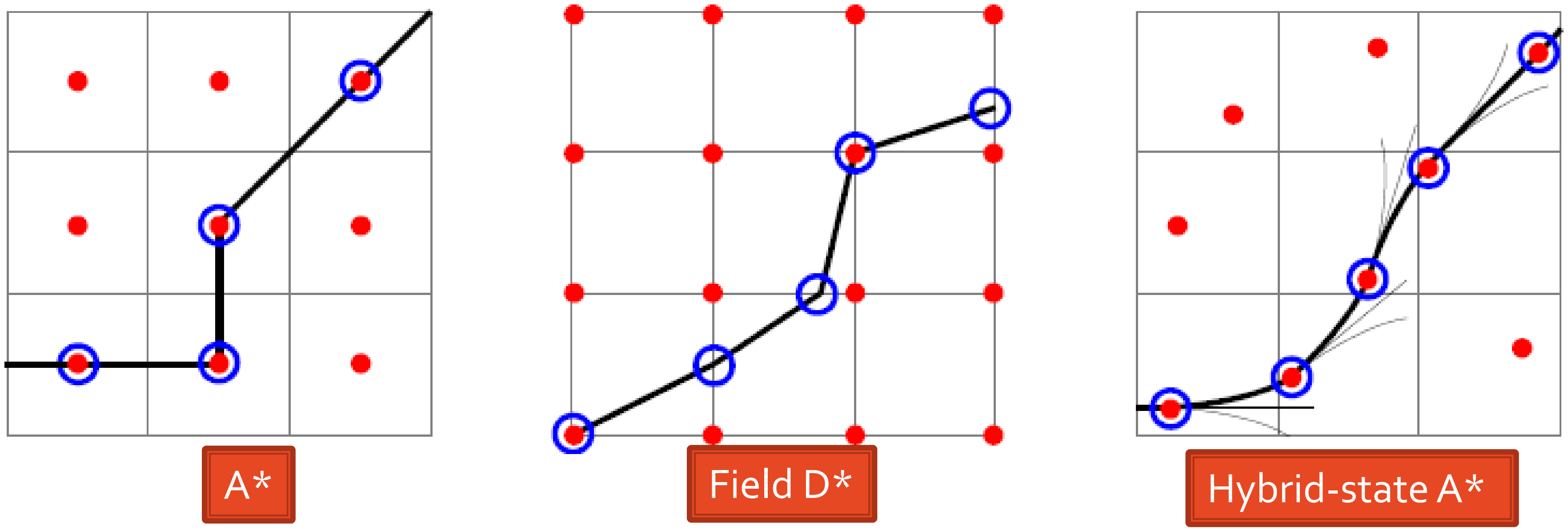
Assistant Professor
Mechanical Engineering Department, Robotic Engineering Program
Worcester Polytechnic Institute

Quiz (10 pts)

- Compare A*, Field D* and Hybrid state A* search by
  - (3 pts) How they associate cost to a cell?
  - (3 pts) How they connect path between states?
- (4 pts) How does Voronoi field differ from potential field?

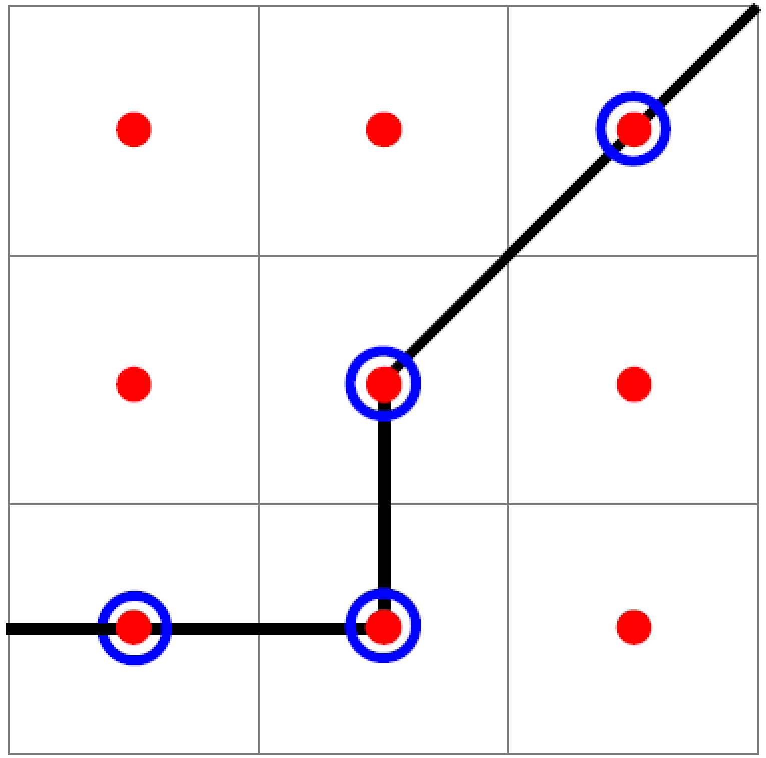
Hybrid-State A* Search

- Compared to A*
  - Search space (x, y, θ)
  - Associates with each grid cell a continuous 3D state of the vehicle



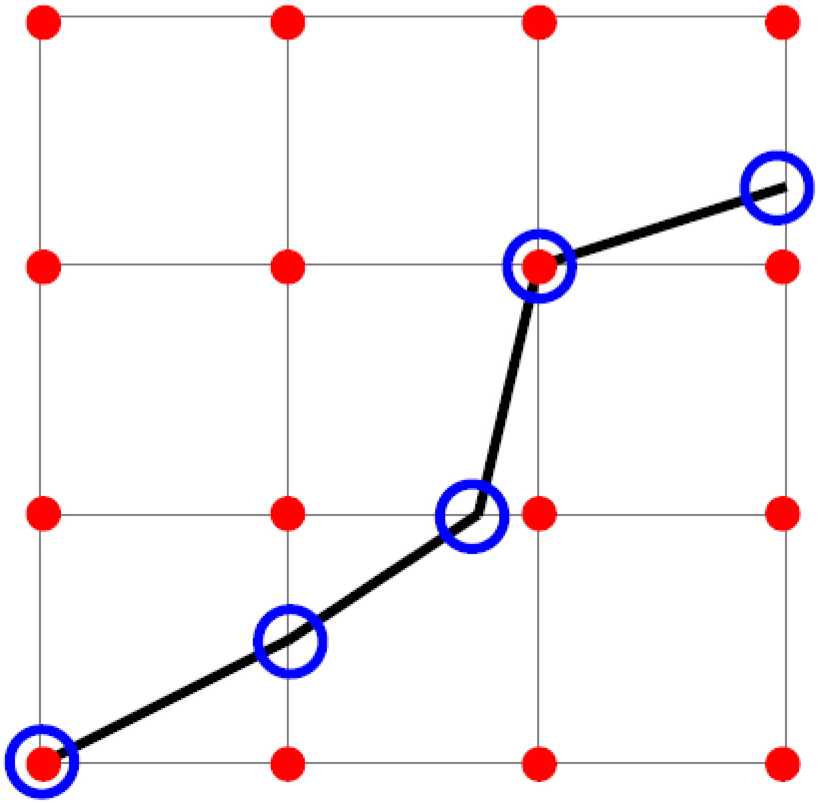
A*

- Features
  - Associates costs with cell centers
  - Only visit states at cell centers



Field D*

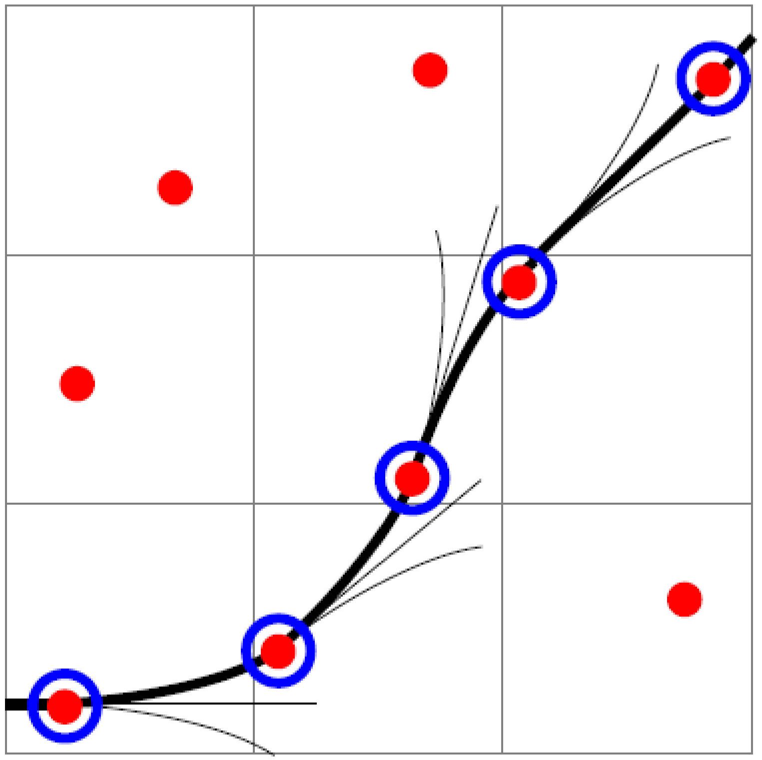
- Features
  - Associate costs with cell corners
  - Allow arbitrary linear paths from cell to cell



  - [Ferguson and Stentz 2005]

Hybrid-State A* Search

- Features
  - Associate a continuous state with each cell
  - Score of the cell = the cost of its associated continuous state



Voronoi field

- Rescale the field based on the workspace geometry

$$\rho_V(x,y) = \left(\frac{\alpha}{\alpha + d_{\mathcal{O}}(x,y)}\right)\left(\frac{d_{\mathcal{V}}(x,y)}{d_{\mathcal{O}}(x,y) + d_{\mathcal{V}}(x,y)}\right)\frac{(d_{\mathcal{O}} - d_{\mathcal{O}}^{max})^2}{(d_{\mathcal{O}}^{max})^2}, \qquad d_{\mathcal{O}} \le d_{\mathcal{O}}^{max}$$

- Otherwise, $\rho_V(x,y) = 0$

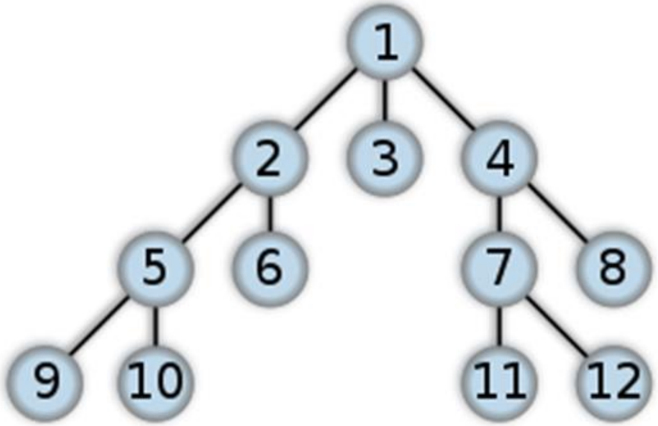
Voronoi field

- Rescale the field based on the workspace geometry

Sampling-based Planning

Recap

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

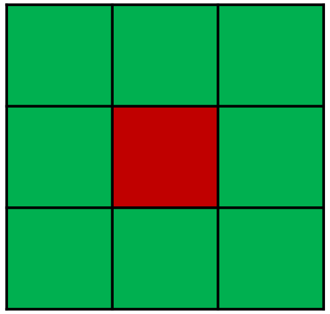


- What if we need to plan in **high-dimensional** spaces?

Discrete Planning – Limitation

- Discrete search
  - Run-time and memory requirements are **sensitive to branching factor** (number of successors)
  - Number of successors depend on **dimension**

How many successors
- For a 3-dimensional 8-connected space?
- For an n-dimensional 8-connected space?



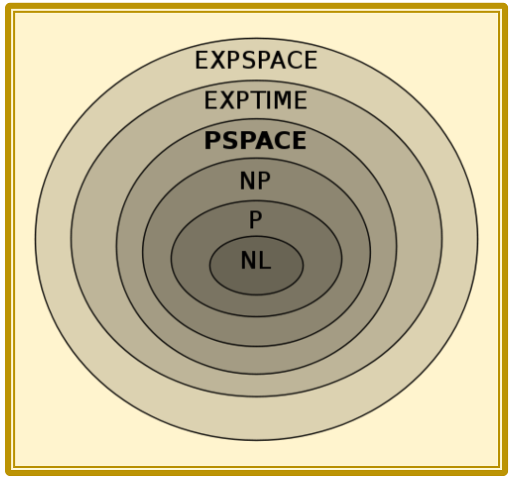
8-connected

Motivation

What if we weaken completeness and optimality requirements?

- 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]



Real robots can have 20+ DOF!

Weakening Requirements

Ideal
Complete
Optimal

Practical
Probabilistically Complete
Feasible

- Probabilistic completeness
  - Given a solvable problem, the probability that the planner solves the problem goes to 1 as time goes to infinity

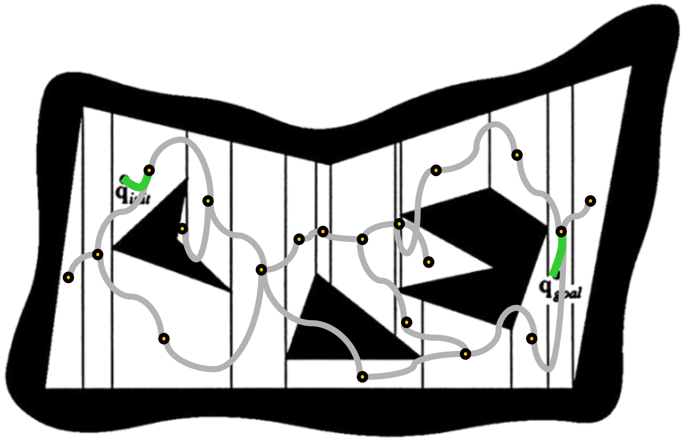
Weakening Requirements

Ideal		Practical
Complete	→	Probabilistically Complete
Optimal		Feasible

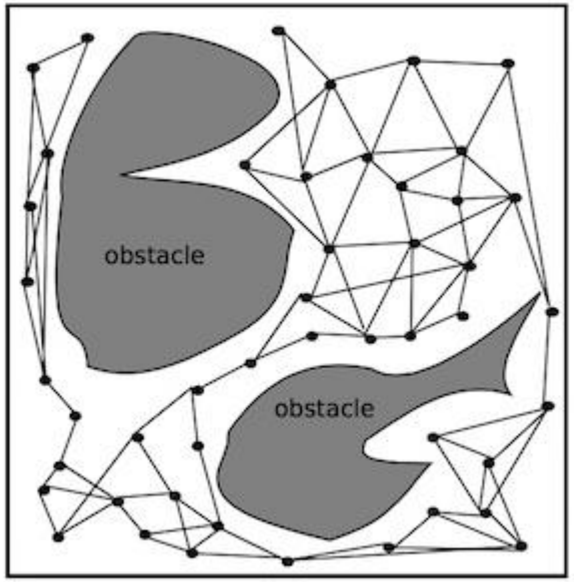
- Feasibility
  - Path obeys all constraints (usually obstacles)
  - A feasible path can be optimized *locally* after it is found

Sampling-based Planning

- Main idea
  - Take **samples** in the C-space and use them to construct a path



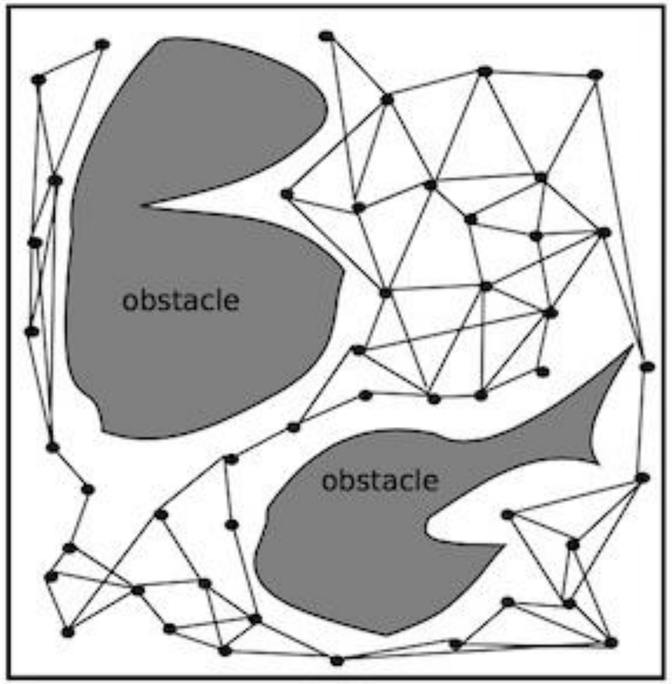
Discrete planning



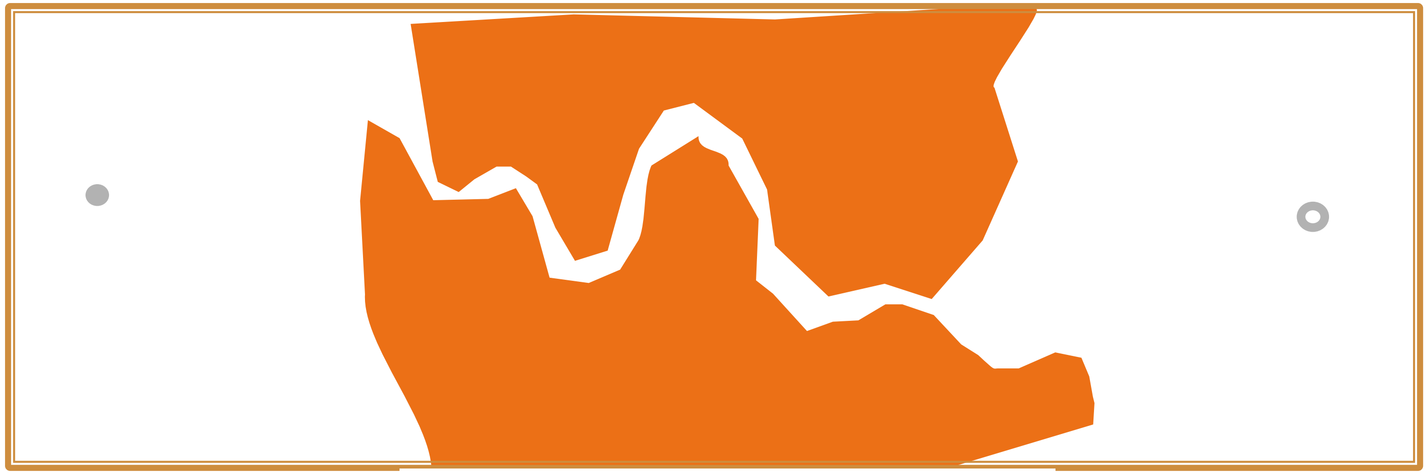
Sampling-based planning

Comparison

- Advantages?
  - No need to discretize C-space
  - No need to explicitly represent C-space
  - Not sensitive to C-space dimension



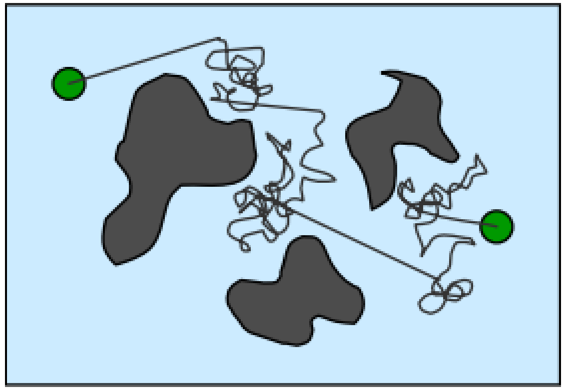
Disadvantages?



- Narrow passages?
  - Probability of sampling an area depends on the area’s size
- No strict completeness/optimality

Randomized Path Planner (RPP)

- Main idea
  - Follow a potential function
  - Occasionally introduce random motion



Barraquand and Latombe in 1991 at Stanford

Pros and Cons

- Advantage
  - Doesn't get stuck in local minima
- Disadvantage? – Many parameters to set
  - Define potential field
  - Decide when to apply random motion
  - How much random motion to apply

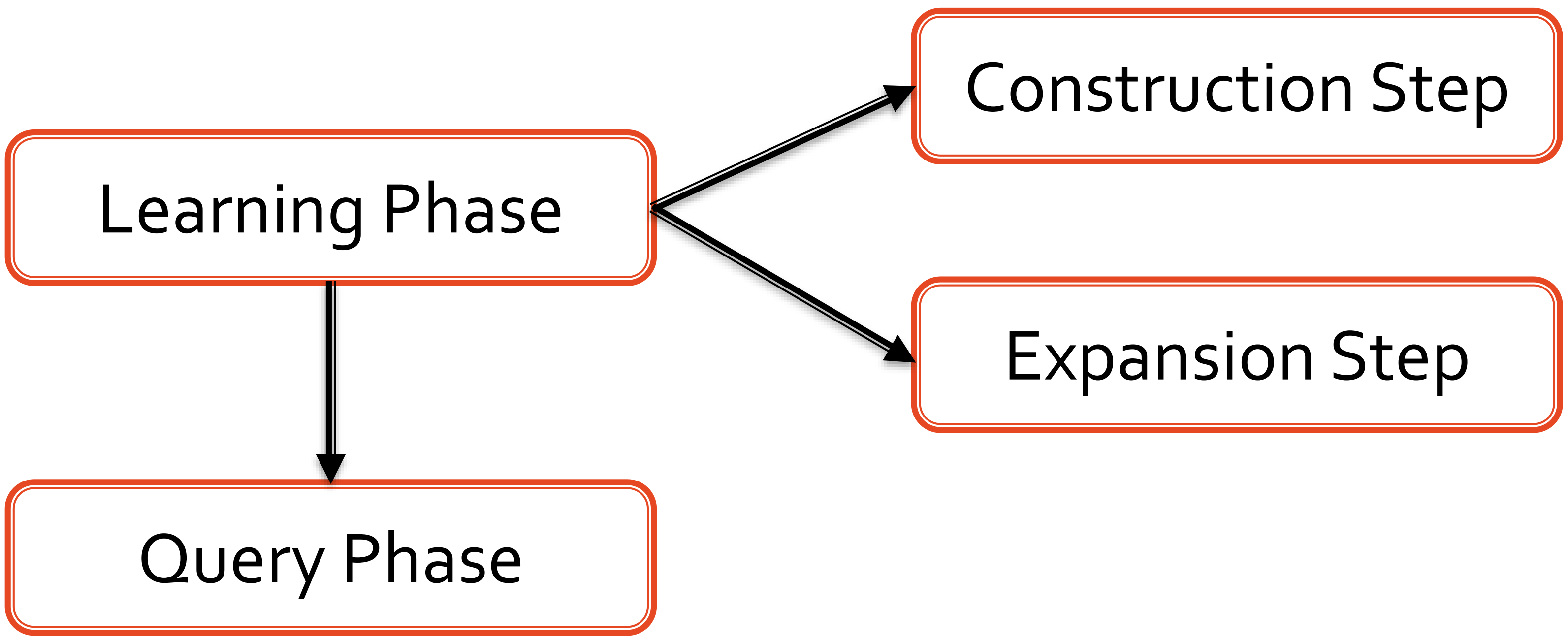
Probabilistic Roadmap (PRM)

- 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

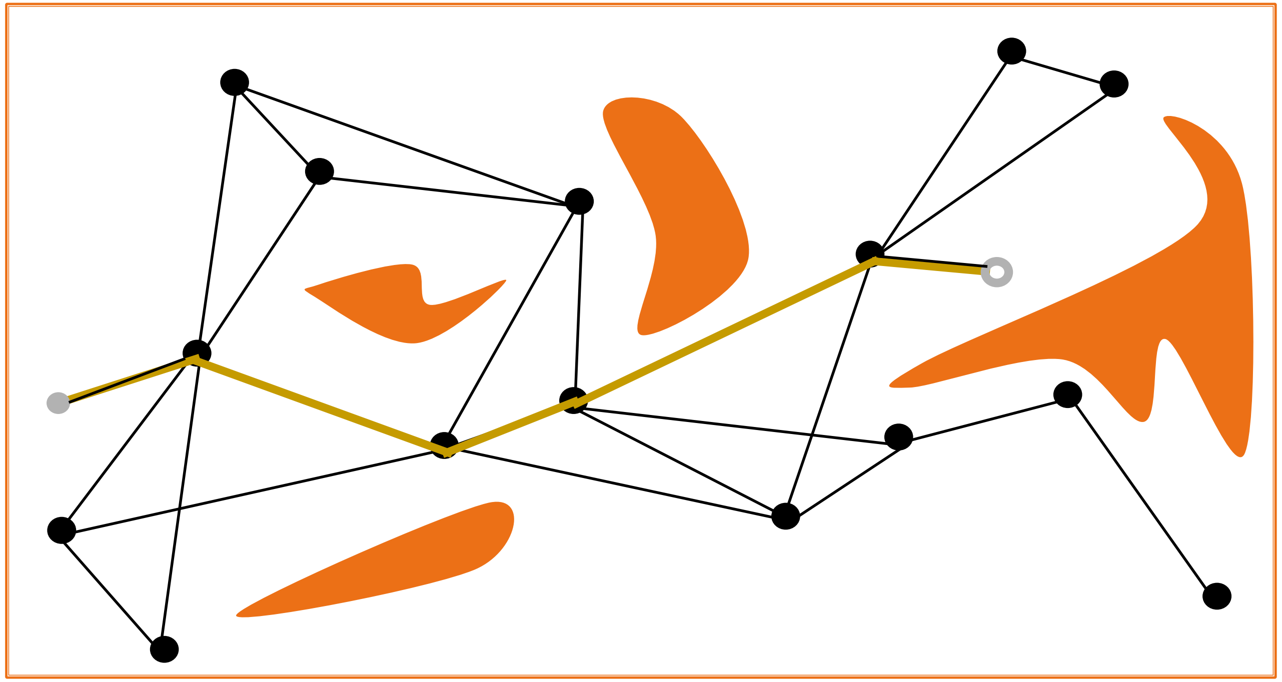
Kavraki, Lydia E., Petr Svestka, J-C. Latombe, and Mark H. Overmars. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces." Robotics and Automation, IEEE Transactions on 12, no. 4, 1996.

Two-phase solution



- Environment remains unchanged
- Reuse roadmap for multi-query

Example

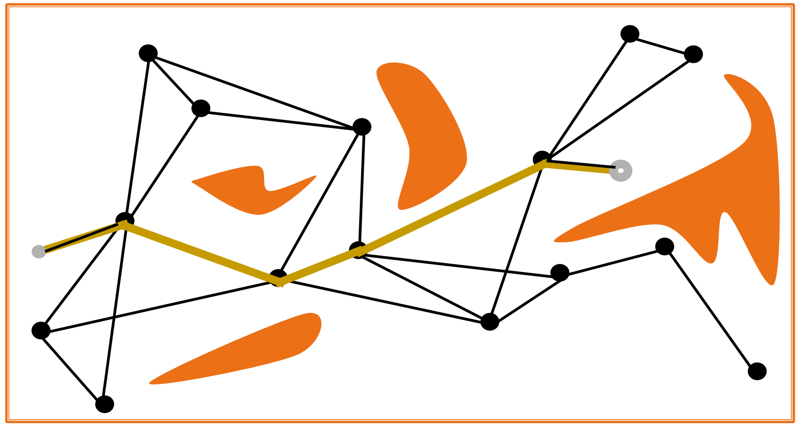


Learning Phase

- 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**

Learning Phase

- Notes
  - Graph **nodes** are sometimes called "**milestones**"
  - Graph **Edges** are the paths between nodes found by the local planner
    - Doesn't have to be linear segments



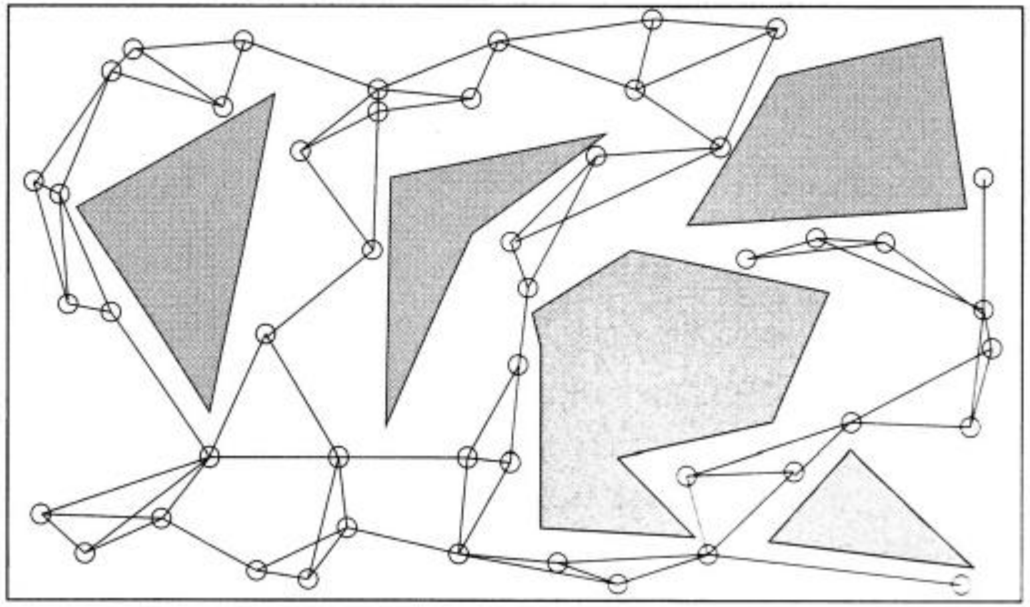
Map Construction

Algorithm 6 Roadmap Construction Algorithm

Input:
n : number of nodes to put in the roadmap
k : number of closest neighbors to examine for each configuration

Output:
A roadmap $G = (V, E)$

$V \leftarrow \emptyset$
$E \leftarrow \emptyset$
while $|V| < n$ **do**
repeat
$q \leftarrow$ a random configuration in $\mathcal{Q}$
until q is collision-free
$V \leftarrow V \cup \{q\}$
end while
for all $q \in V$ **do**
$N_q \leftarrow$ the k closest neighbors of q chosen from V according to *dist*
for all $q' \in N_q$ **do**
if $(q, q') \notin E$ **and** $\Delta(q, q') \neq \texttt{NIL}$ **then**
$E \leftarrow E \cup \{(q, q')\}$
end if
end for
end for



Map Construction – Sampling

Algorithm 6 Roadmap Construction Algorithm

Input:

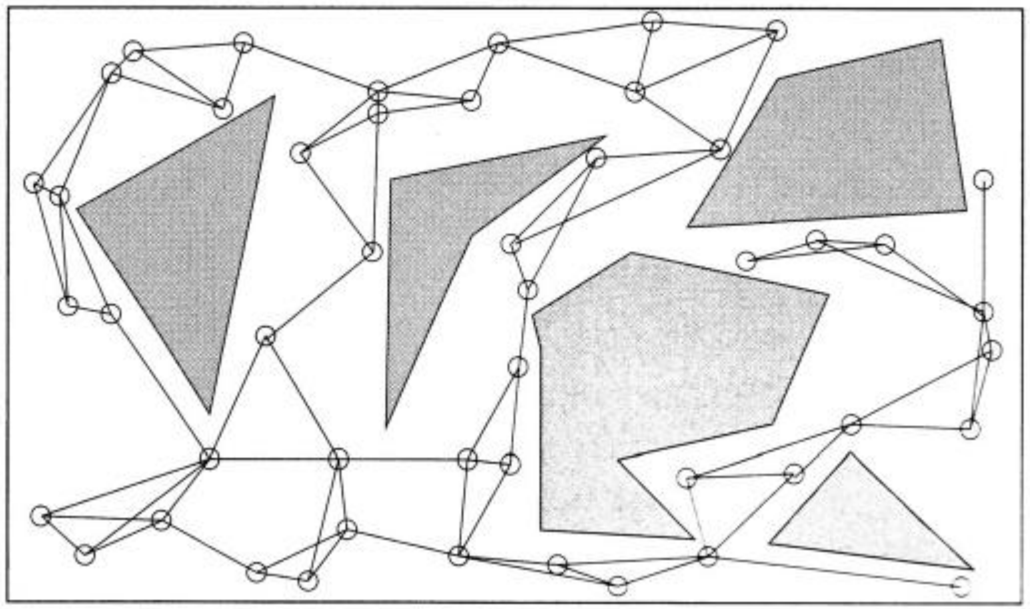
n : number of nodes to put in the roadmap

k : number of closest neighbors to examine for each configuration

Output:

A roadmap $G = (V, E)$

$V \leftarrow \emptyset$
$E \leftarrow \emptyset$
while $|V| < n$ **do**
    repeat
        $\boxed{q \leftarrow \text{a random configuration in } Q}$
    until q is collision-free
    $V \leftarrow V \cup \{q\}$
end while
for all $q \in V$ **do**
    $N_q \leftarrow$ the k closest neighbors of q chosen from V according to *dist*
    for all $q' \in N_q$ **do**
        if $(q, q') \notin E$ **and** $\Delta(q, q') \neq \texttt{NIL}$ **then**
            $E \leftarrow E \cup \{(q, q')\}$
        end if
    end for
end for



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, and 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 …

Map Construction – Nearest Neighbor

Algorithm 6 Roadmap Construction Algorithm

Input:

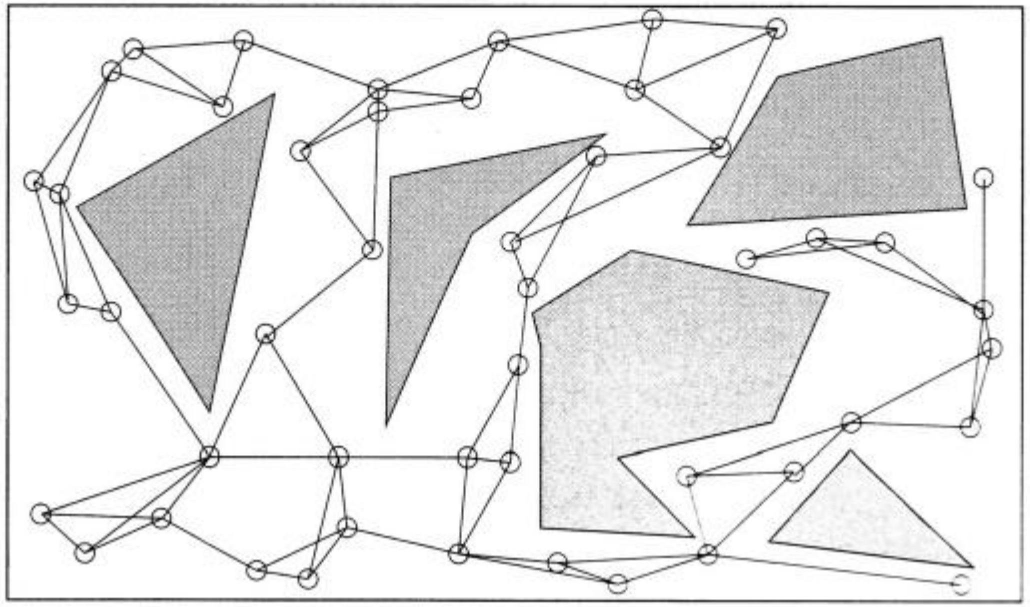
n : number of nodes to put in the roadmap

k : number of closest neighbors to examine for each configuration

Output:

A roadmap $G = (V, E)$

$V \leftarrow \emptyset$
$E \leftarrow \emptyset$
while $|V| < n$ **do**
    repeat
        $q \leftarrow$ a random configuration in $\mathcal{Q}$
    until q is collision-free
    $V \leftarrow V \cup \{q\}$
end while
for all $q \in V$ **do**
    $N_q \leftarrow$ the k closest neighbors of q chosen from V according to *dist*
    for all $q' \in N_q$ **do**
        if $(q, q') \notin E$ **and** $\Delta(q, q') \neq$ `NIL` **then**
            $E \leftarrow E \cup \{(q, q')\}$
        end if
    end for
end for



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

- Distance metrics?

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

- Can we weigh different dimensions of C-space differently?

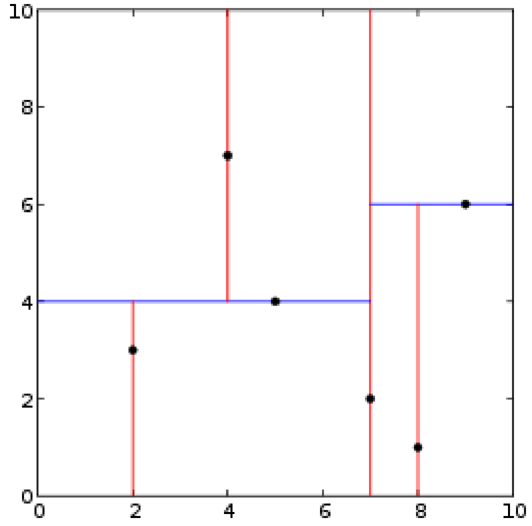
Finding Nearest Neighbors (NN)

- 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

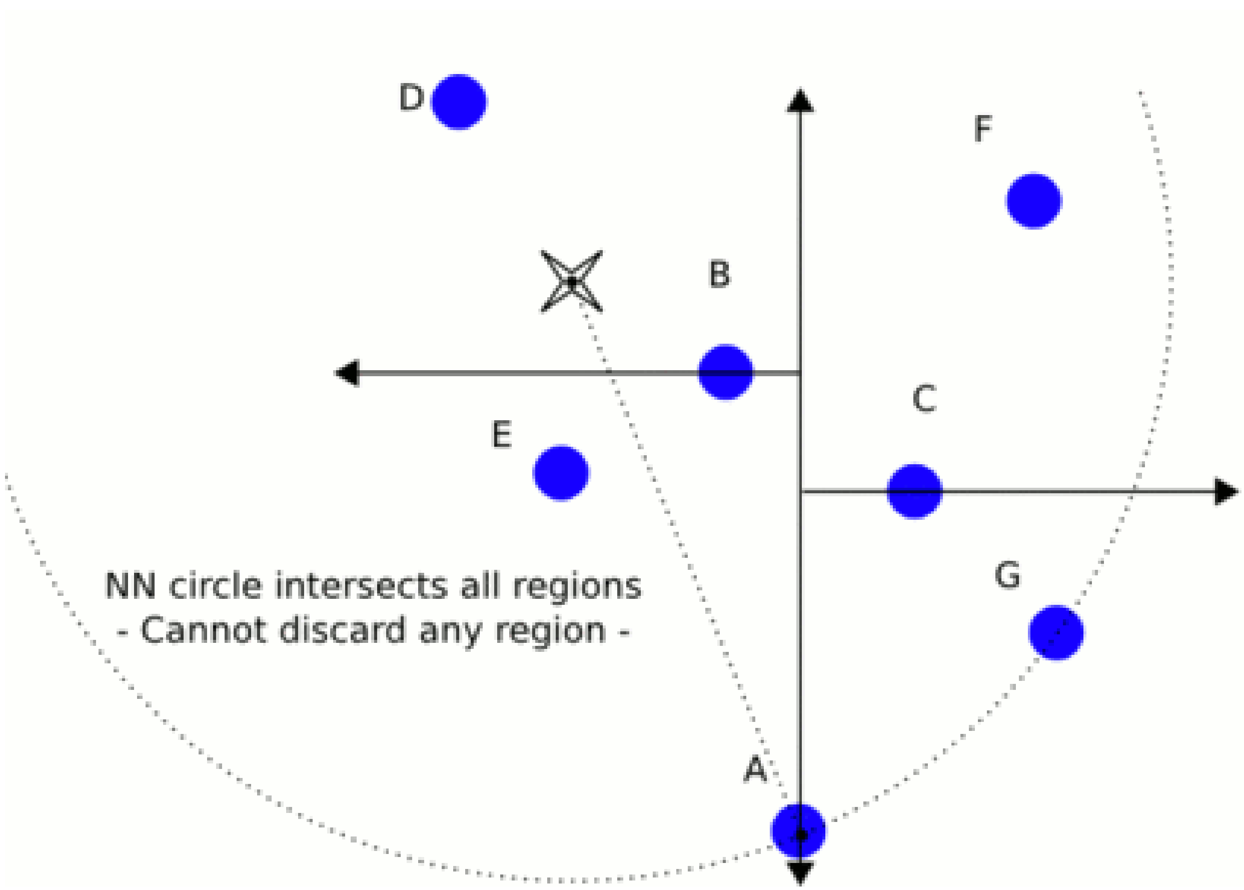
- Computational complexity?
  - Naive NN computation can be slow with thousands of nodes
  - use *kd-tree* to store nodes and do NN queries

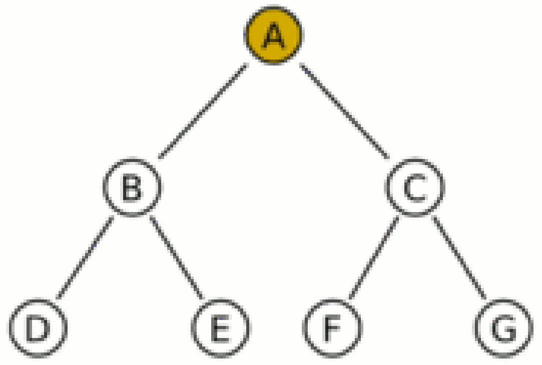
Kd-tree (k-dimensional tree)

- Organize points in a space with k dimensions
  - Each level of a k-d tree splits all children along a specific dimension
  - Each level down in the tree divides on the next dimension
  - Partition the tree to place the median point at the root



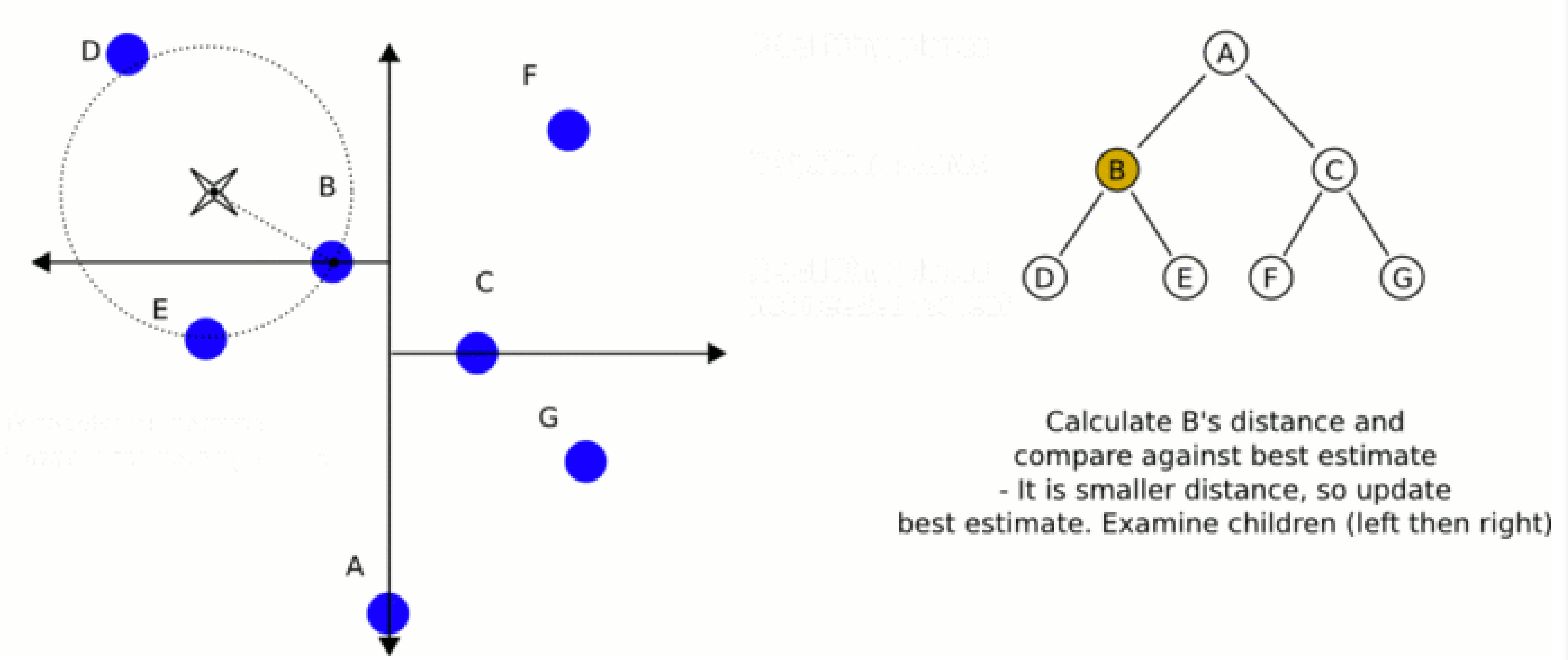
Kd-tree





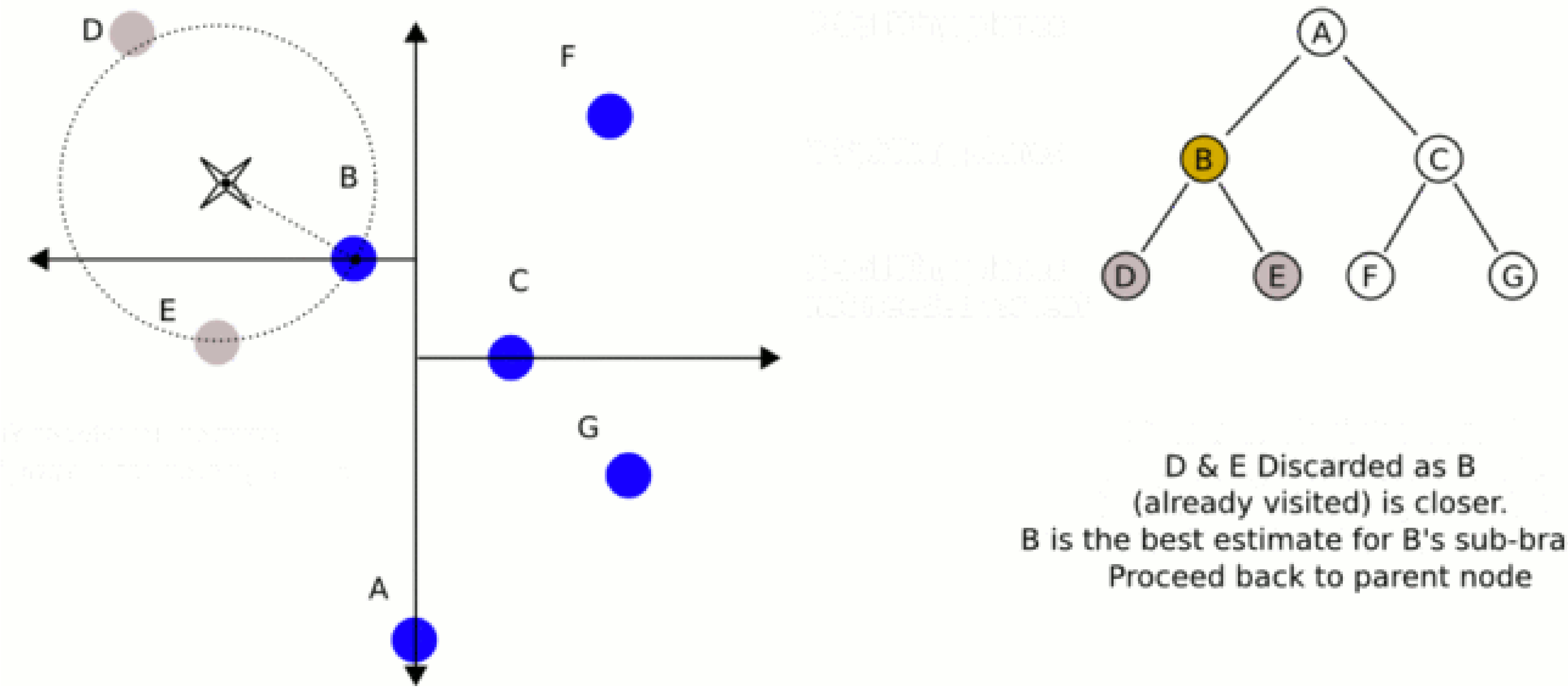
Start at A, then proceed in depth-first search (maintain a stack of parent-nodes if using a singly-linked tree). Set best estimate to A's distance Then examine left child node

Kd-tree



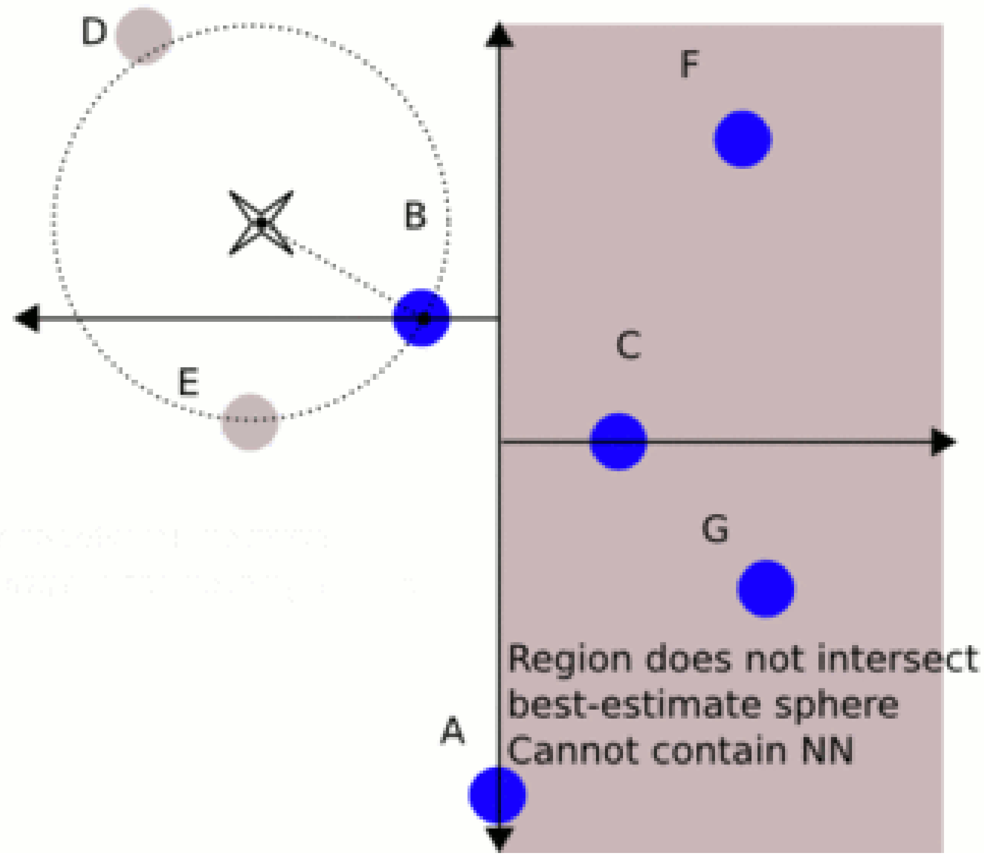
Calculate B's distance and
compare against best estimate
- It is smaller distance, so update
best estimate. Examine children (left then right)

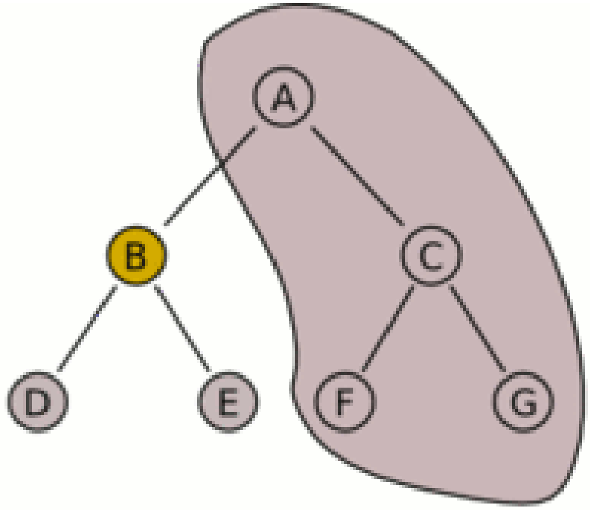
Kd-tree



D & E Discarded as B
(already visited) is closer.
B is the best estimate for B's sub-branch
Proceed back to parent node

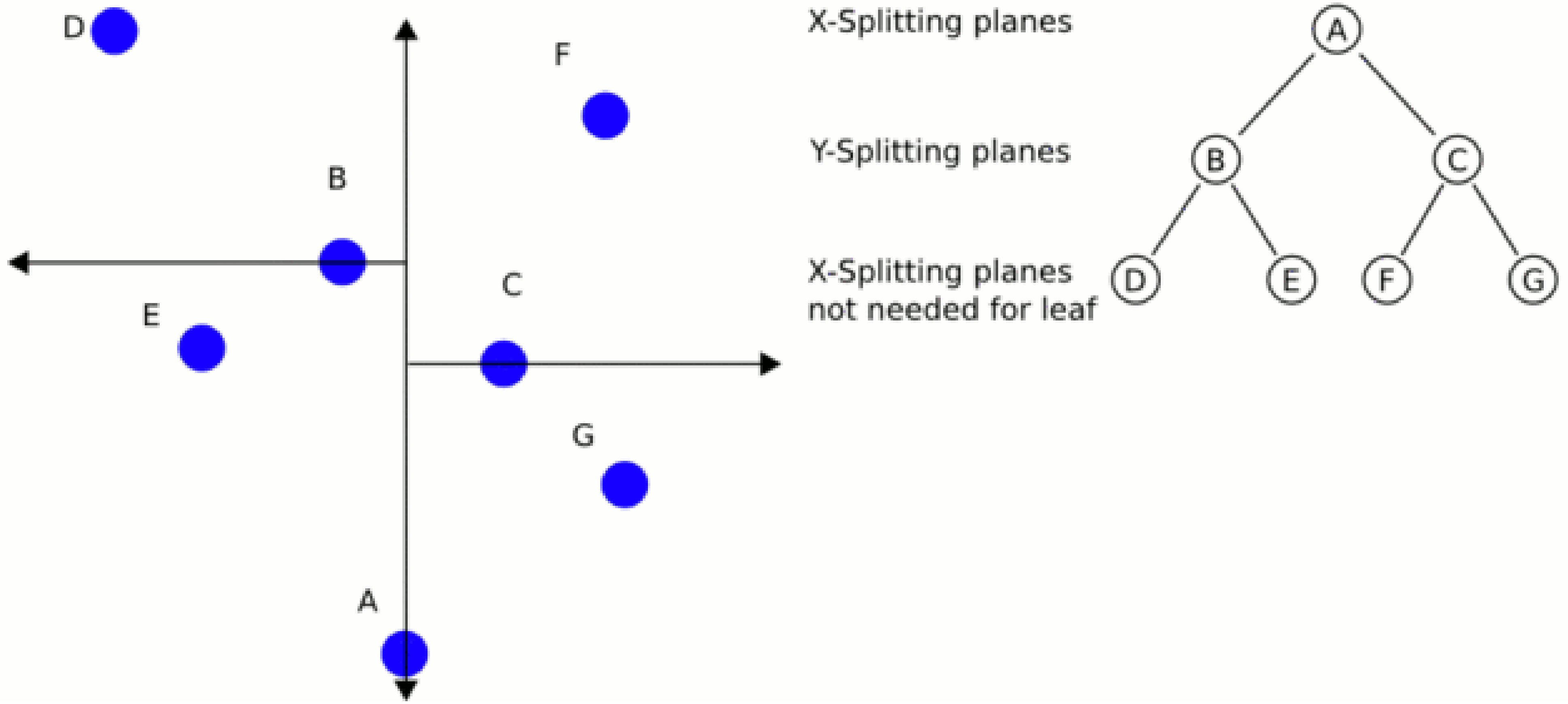
Kd-tree





A's children have all been searched,
B is the best estimate for entire tree

Kd-tree


D
F
B
C
E
G
A
X-Splitting planes
Y-Splitting planes
X-Splitting planes
not needed for leaf
A
B
C
D
E
F
G

Performance

- Speed
  - Much **faster** to use kd-tree for **large** numbers of nodes

- Cost
  - Cost of constructing a kd-tree is **significant**
  - Only regenerate tree once in a while (not for every new node!)

Map Construction – Local planner

Algorithm 6 Roadmap Construction Algorithm

Input:

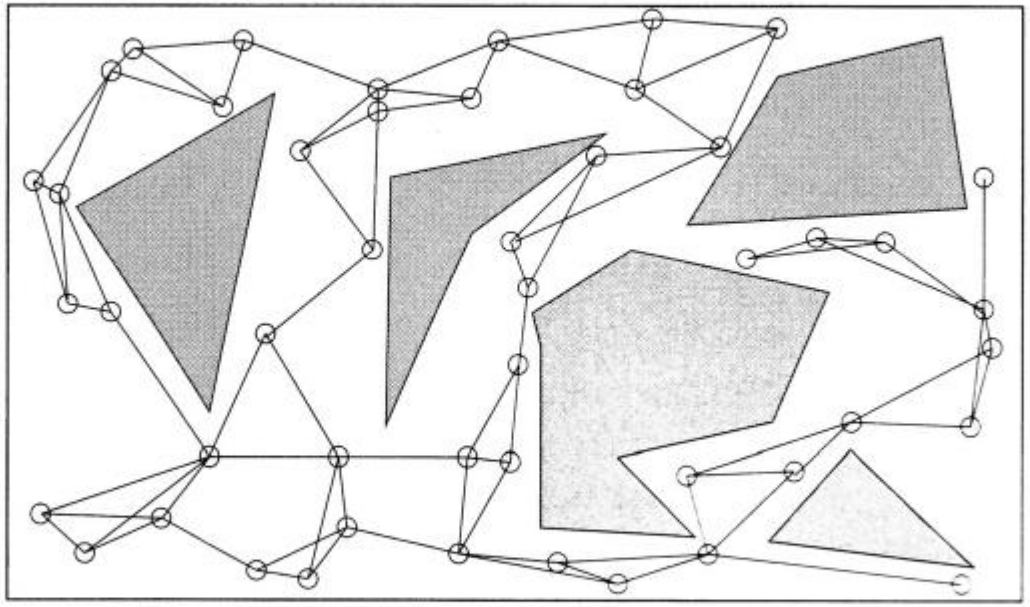
n : number of nodes to put in the roadmap

k : number of closest neighbors to examine for each configuration

Output:

A roadmap $G = (V, E)$

$V \leftarrow \emptyset$
$E \leftarrow \emptyset$
while $|V| < n$ **do**
  repeat
    $q \leftarrow$ a random configuration in $\mathcal{Q}$
  until q is collision-free
  $V \leftarrow V \cup \{q\}$
end while
for all $q \in V$ **do**
  $N_q \leftarrow$ the k closest neighbors of q chosen from V according to *dist*
  for all $q' \in N_q$ **do**
    if $\boxed{(q, q') \notin E \textbf{ and } \Delta(q, q') \neq \texttt{NIL}}$ **then**
      $E \leftarrow E \cup \{(q, q')\}$
    end if
  end for
end for



Local Planner

- In general, local planner can be **anything** that attempts to find a path between points

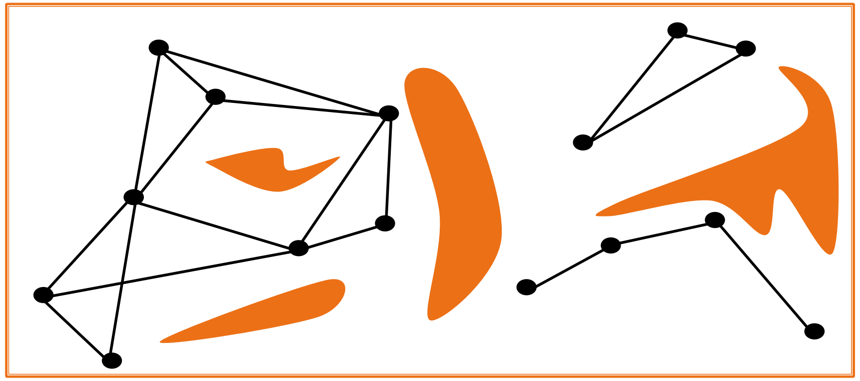
- Local planners you can think of?
  - Local discrete search
  - Potential field
  - Motion primitives
  - Another PRM!

Local planner

- Local planner needs to be …
  - **Fast** – It’s called many times by the algorithm
- Easiest and most common implementation
  - Connect the two configurations with **a straight line** in C-space,
  - Check that line is collision-free
    - Fast
    - Don’t need to store local paths

Expansion Step

- Disconnected components that should be connected
  - i.e., you haven't captured the true connectivity of the space
- Sample more nodes to connect disconnected components
  - Use **heuristics** to measure the connection difficulty
  - What heuristics?



Possible Heuristics

- # of Nodes nearby
  - For a node **c**, count the # of nodes **N** within a predefined distance
  - N is small → obstacle region may occupy large portion of **c**'s neighborhood
  - Use Heuristics = 1/**N** to guide random sampling

Possible Heuristics

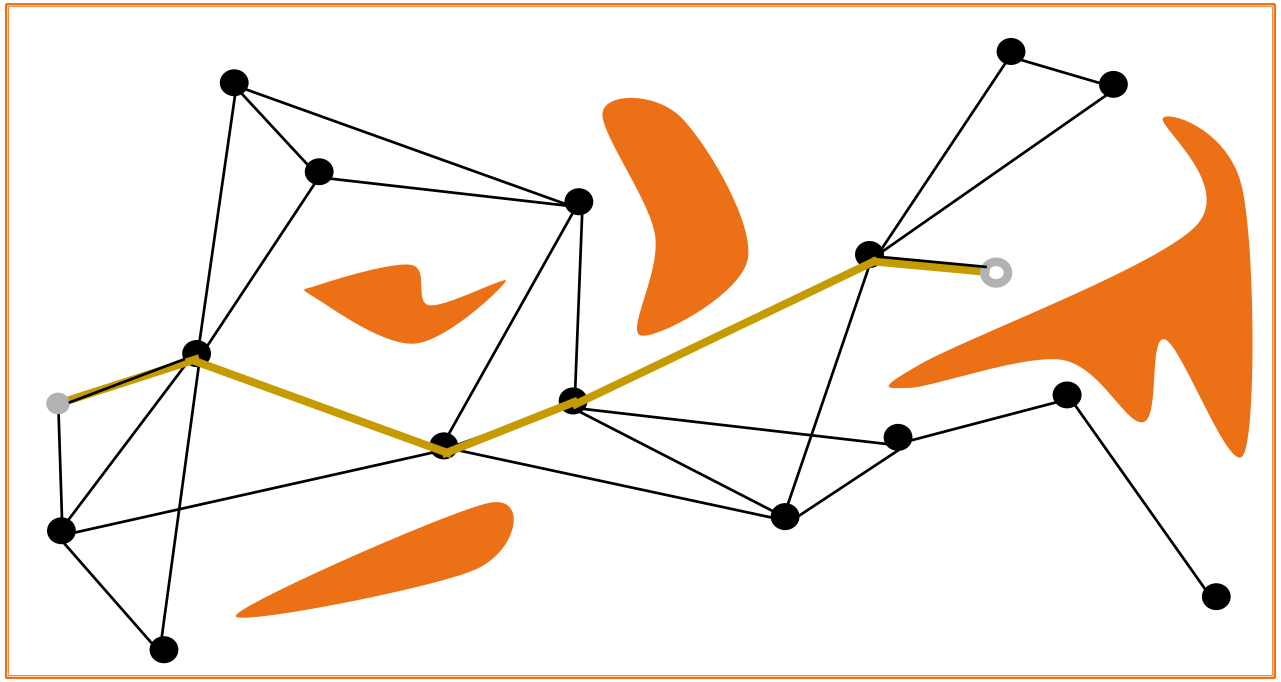
- Distance to nearest reacheable neighbor
  - For a node **c**, find the distance **d** to the nearest connected component that doesn't contains this node
  - **d** is small → **c** lies in the region where two components fail to connect
  - Heuristics = 1/d

Possible Heuristics

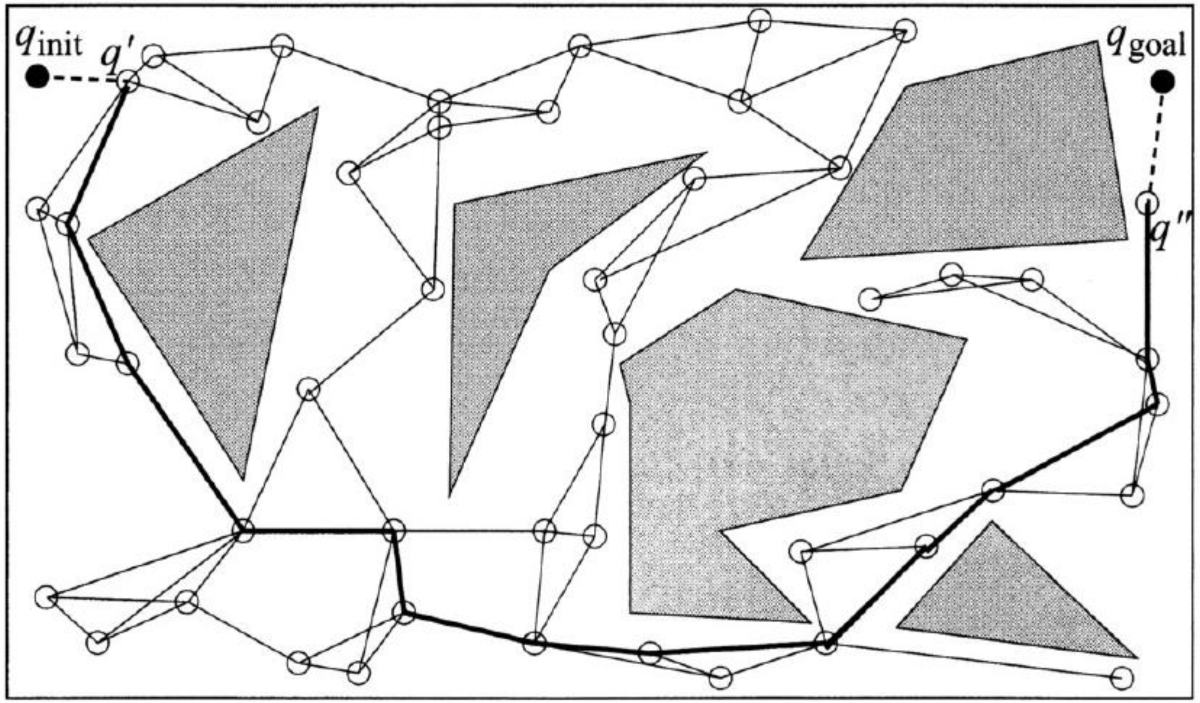
- Others?
- Behavior of local planner?
  - Always fail to connect → difficult region

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.



Path Query



How to search the shortest path in a graph?

Algorithm 7 Solve Query Algorithm

Input:

q_{init}: the initial configuration

q_{goal}: the goal configuration

k: the number of closest neighbors to examine for each configuration

$G = (V, E)$: the roadmap computed by algorithm 6

Output:

A path from q_{init} to q_{goal} or failure

$N_{q_{\text{init}}} \leftarrow$ the k closest neighbors of q_{init} from V according to *dist*
$N_{q_{\text{goal}}} \leftarrow$ the k closest neighbors of q_{goal} from V according to *dist*
$V \leftarrow \{q_{\text{init}}\} \cup \{q_{\text{goal}}\} \cup V$
set q' to be the closest neighbor of q_{init} in $N_{q_{\text{init}}}$
repeat
  if $\Delta(q_{\text{init}}, q') \neq$ `NIL` **then**
    $E \leftarrow (q_{\text{init}}, q') \cup E$
  else
    set q' to be the next closest neighbor of q_{init} in $N_{q_{\text{init}}}$
  end if
until a connection was succesful or the set $N_{q_{\text{init}}}$ is empty
set q' to be the closest neighbor of q_{goal} in $N_{q_{\text{goal}}}$
repeat
  if $\Delta(q_{\text{goal}}, q') \neq$ `NIL` **then**
    $E \leftarrow (q_{\text{goal}}, q') \cup E$
  else
    set q' to be the next closest neighbor of q_{goal} in $N_{q_{\text{goal}}}$
  end if
until a connection was succesful or the set $N_{q_{\text{goal}}}$ is empty
$P \leftarrow$ shortest path$(q_{\text{init}}, q_{\text{goal}}, G)$
if P is not empty **then**
  return P
else
  return `failure`
end if

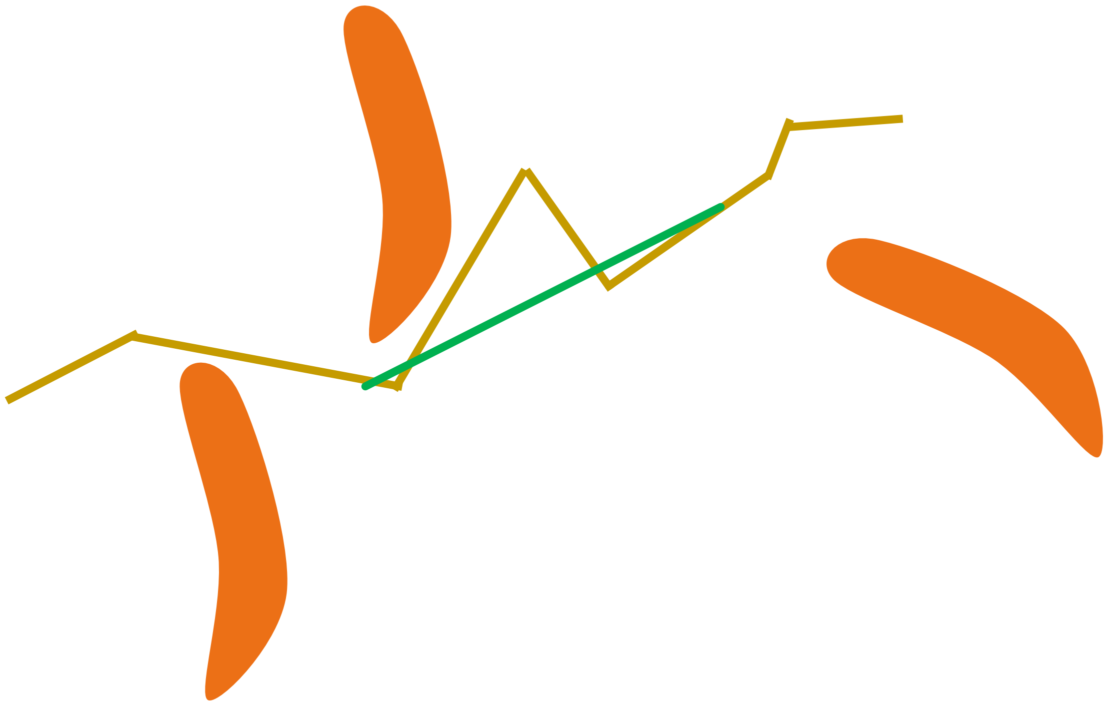
Path Shortening/Smoothing

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

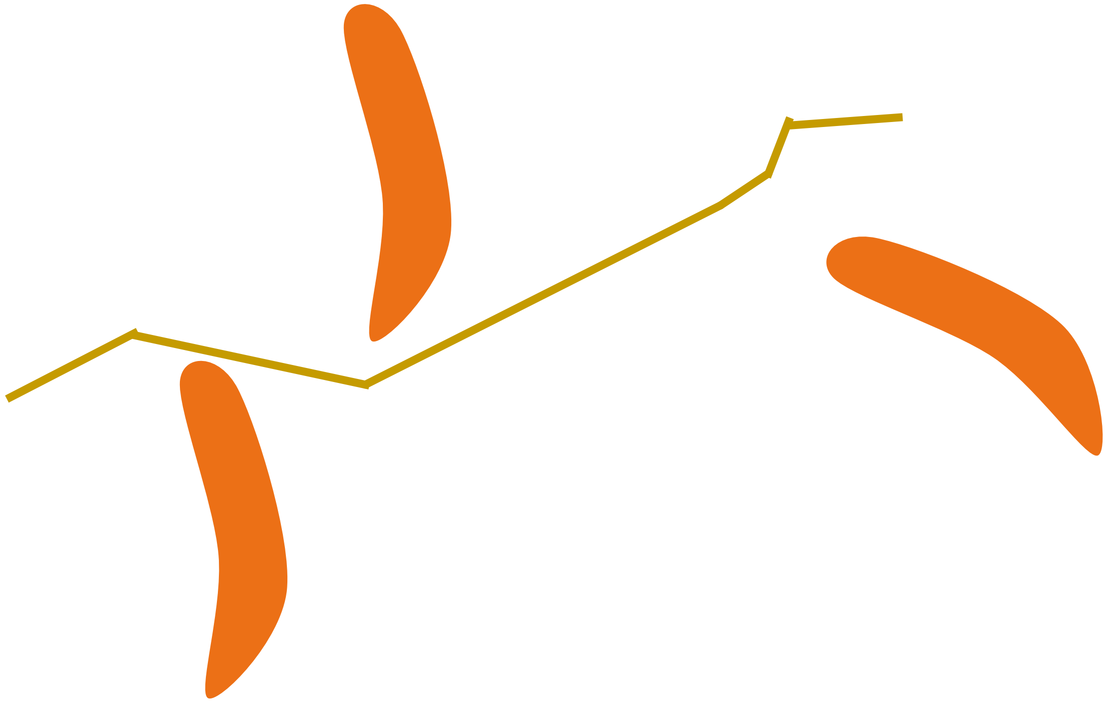
```
For i = 0 to MaxIterations
        Pick two points, q1 and q2, on the path randomly
        Attempt to connect (q1, q2) with a line segment
        If successful, replace path between q1 and q2 with the line segment
```

Shortcut Smoothing

Shortcut Smoothing

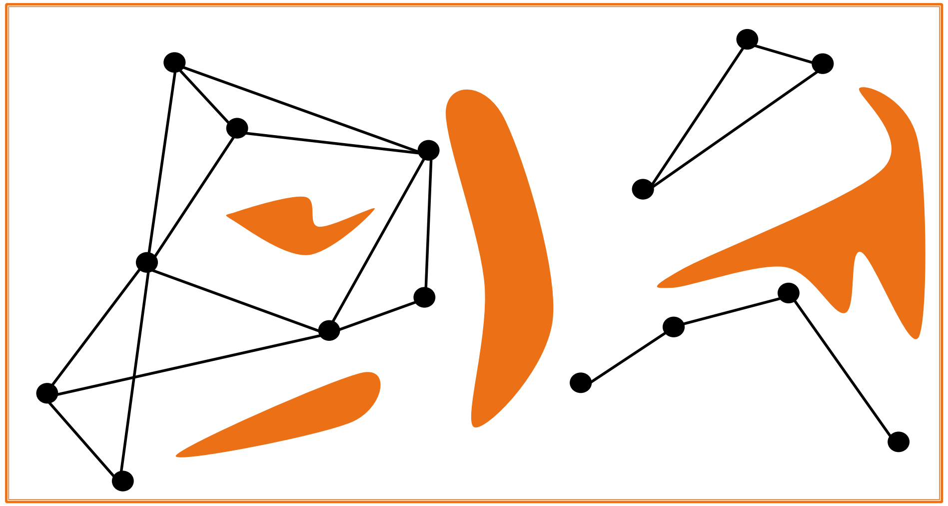


Shortcut Smoothing



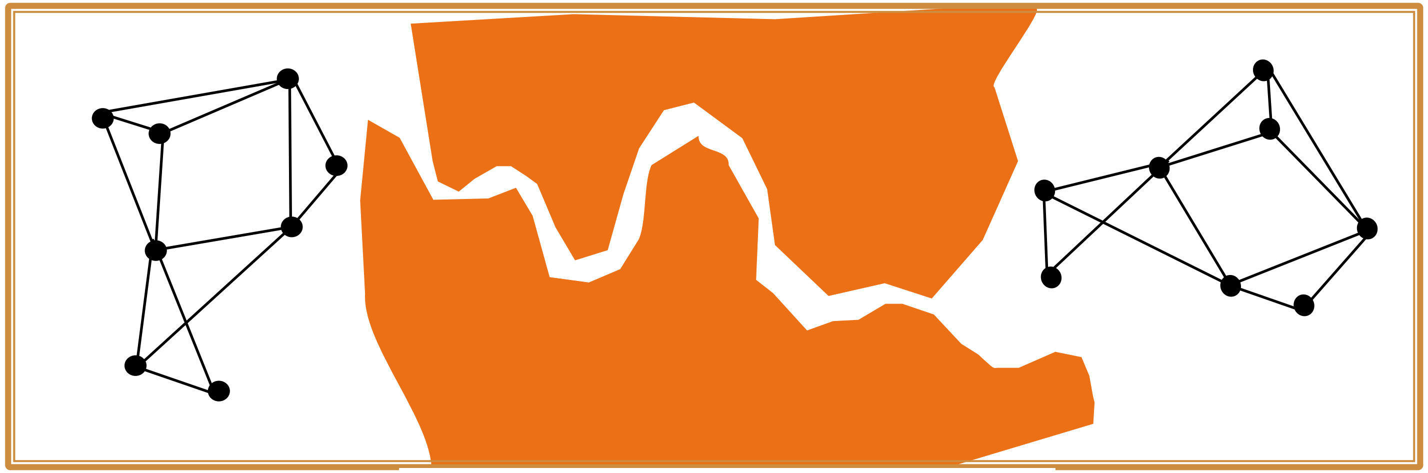
PRM Failure Modes

- Failure cases
  - Can’t connect start and goal to any node in the graph
  - Can’t find a path in the graph but a path is possible



How to fix?

- 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



Student talk

End