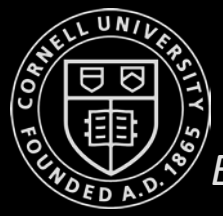
ECE 4960

Prof. Kirstin Hagelskjær Petersen
kirstin@cornell.edu

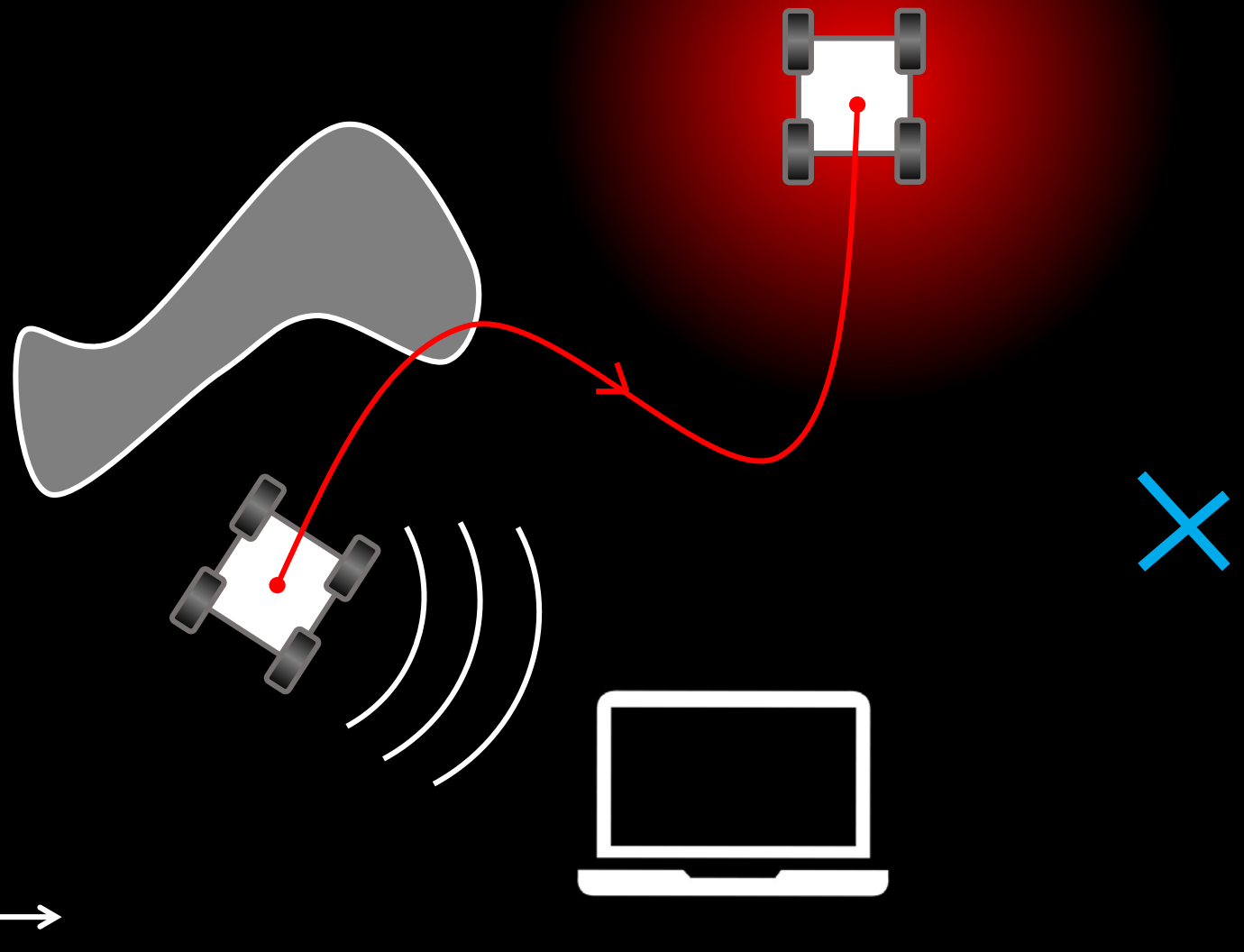
Fast Robots

Slides adapted from Vivek Thangavelu



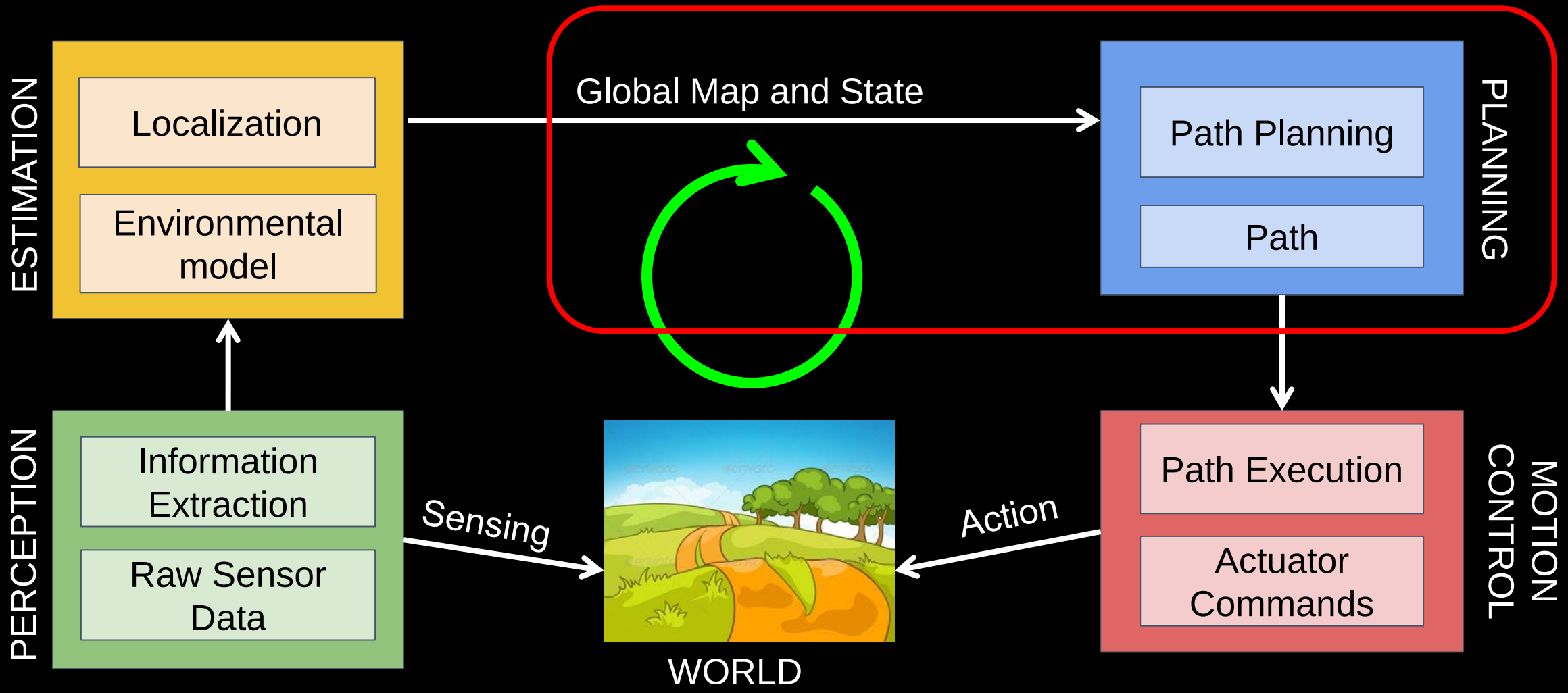
Outline of the next module on Navigation

- Local planners
- Global localization and planning
  - Configuration space
  - Map representations
    - Continuous
    - Discrete
    - Topological
  - Graph representations
  - Graph Search Algorithms
    - Breadth First Search
    - Depth First Search
    - Dijkstras
    - A*



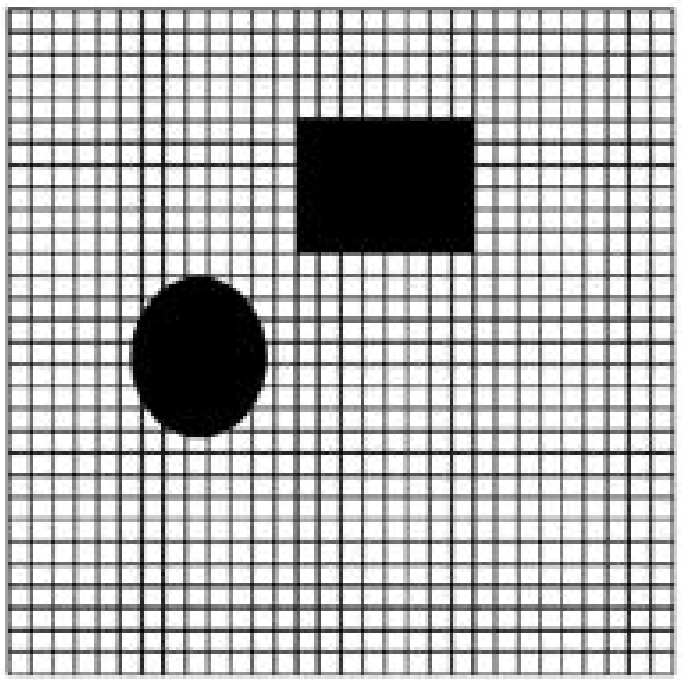
Outline of the next module on Navigation and Path Planning

- Navigation breaks down to: Localization, Map Building, Path Planning



Global Motion Planning with Maps

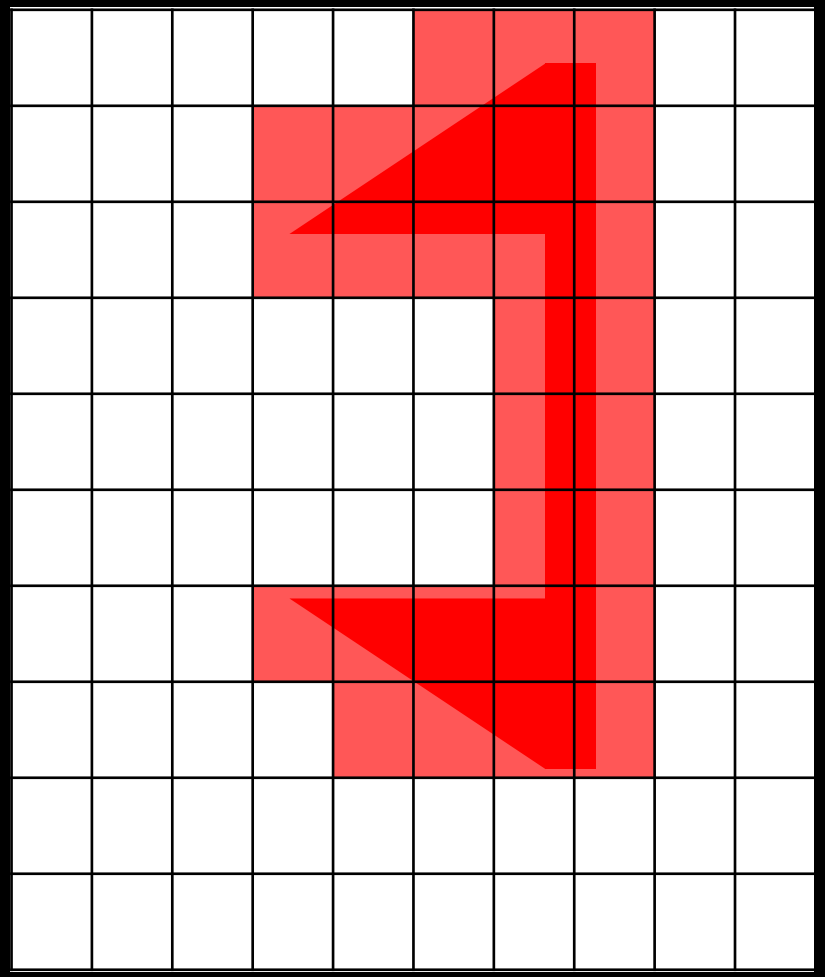
Occupancy Grid Map (discr. coord)

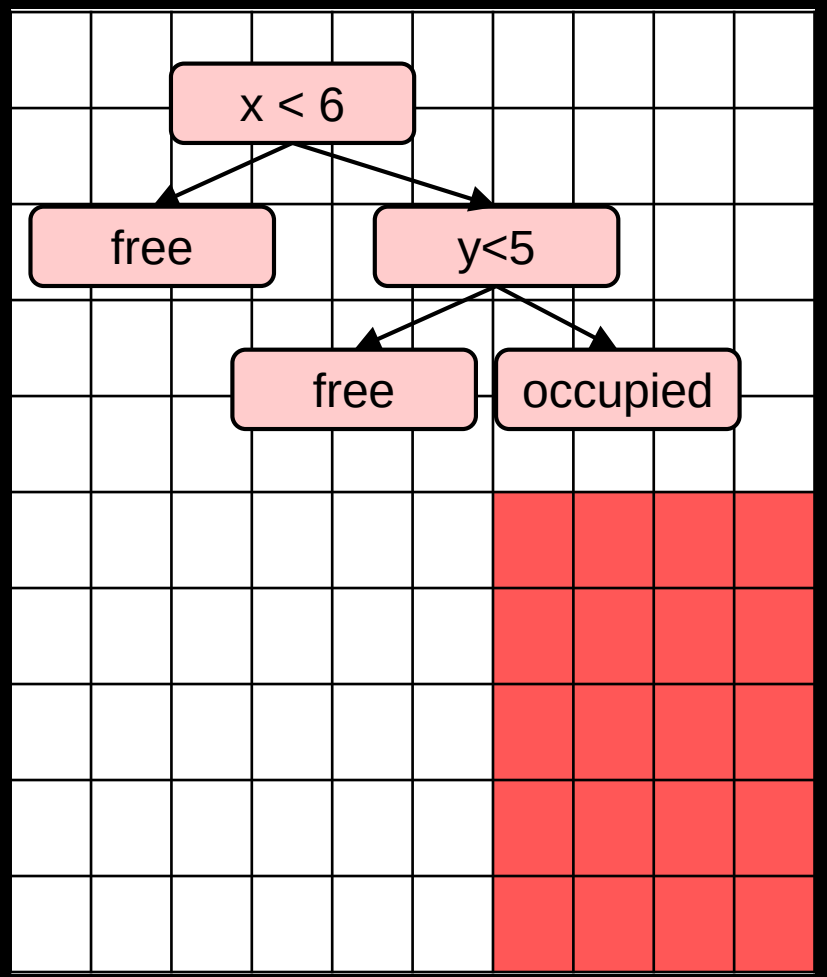


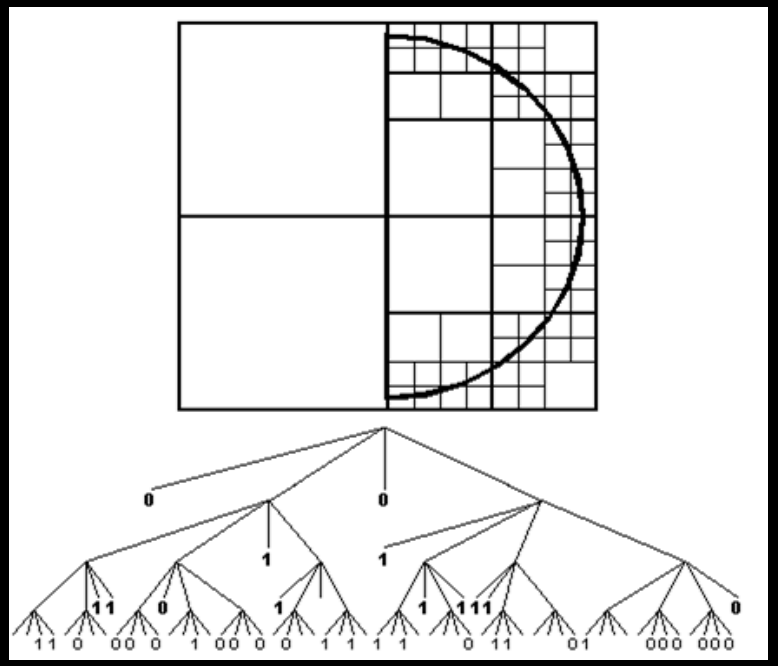
Global Motion Planning with Maps

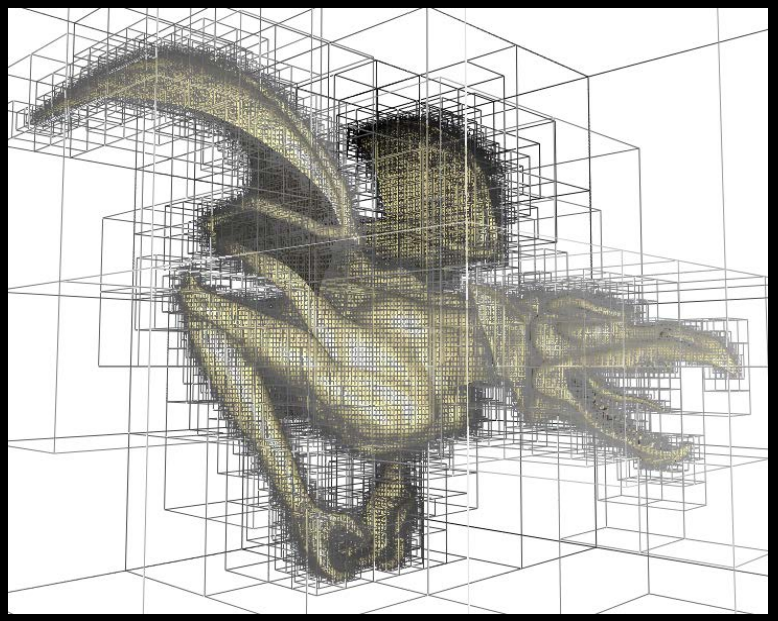
Occupancy grid map

- 2D array: Each cell represents a square of real world (cms)
- Wasteful for big open spaces



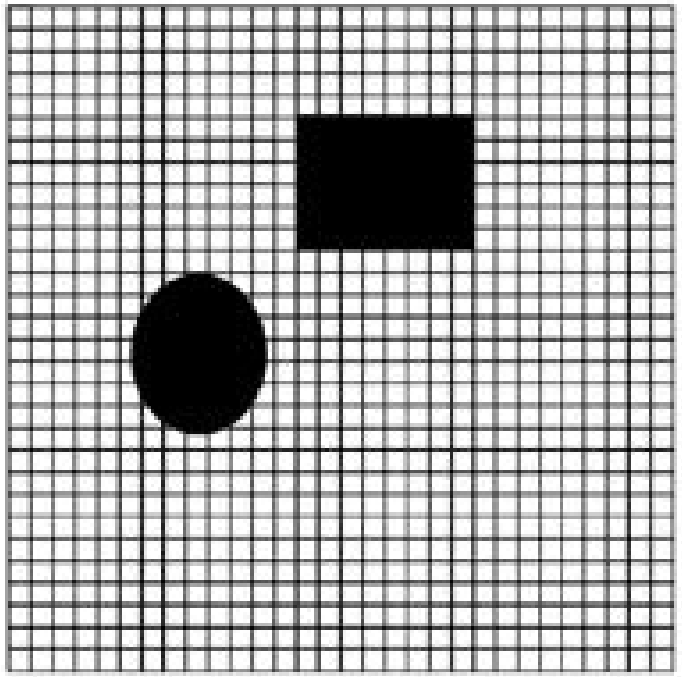




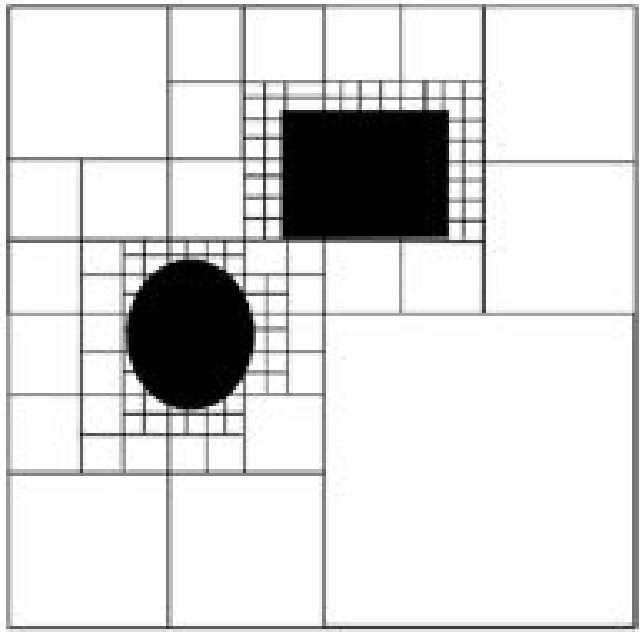


Global Motion Planning with Maps

Occupancy Grid Map (discr. coord)



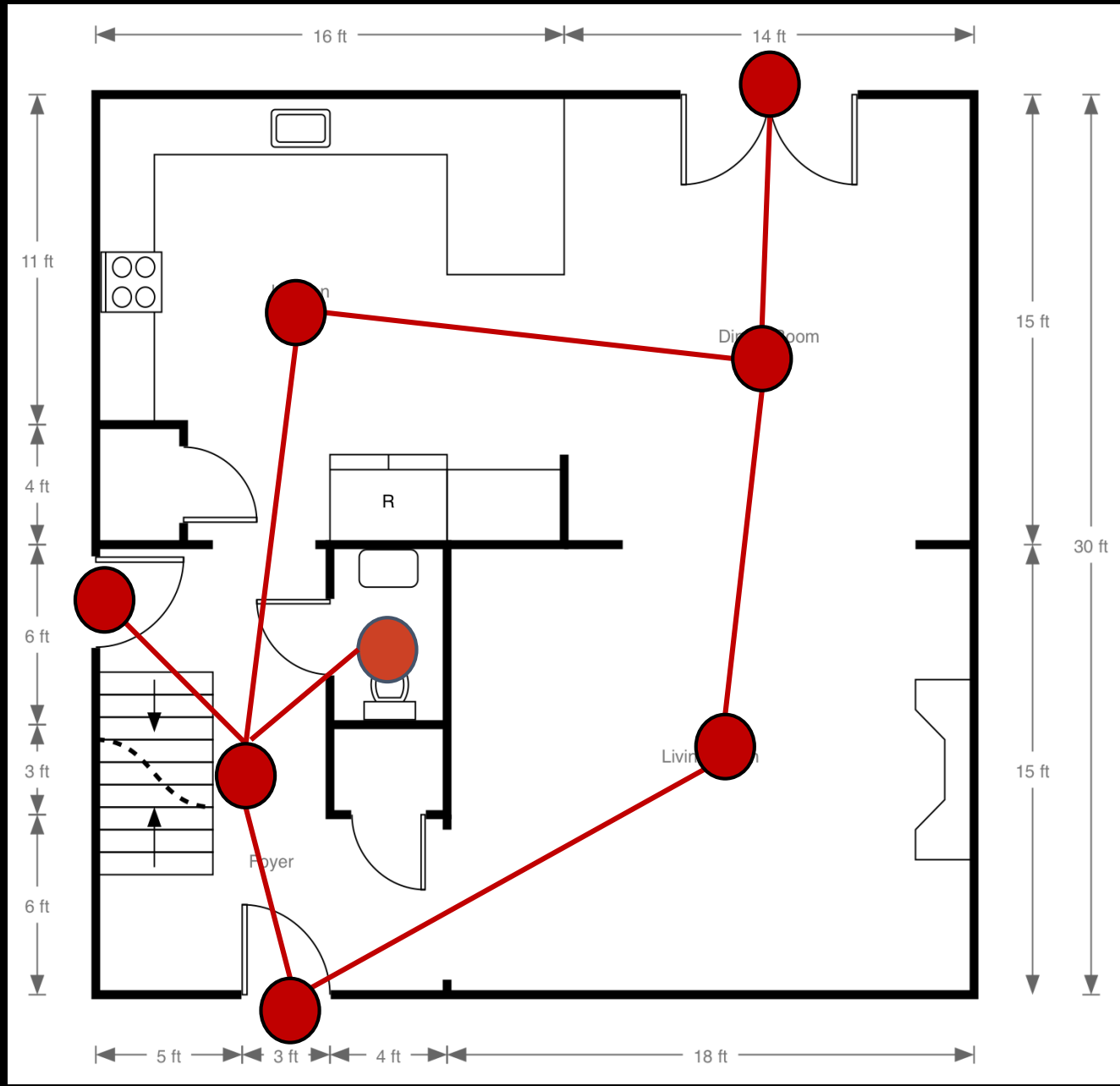
K-d Tree Map (Quadtree)



Global Motion Planning with Maps

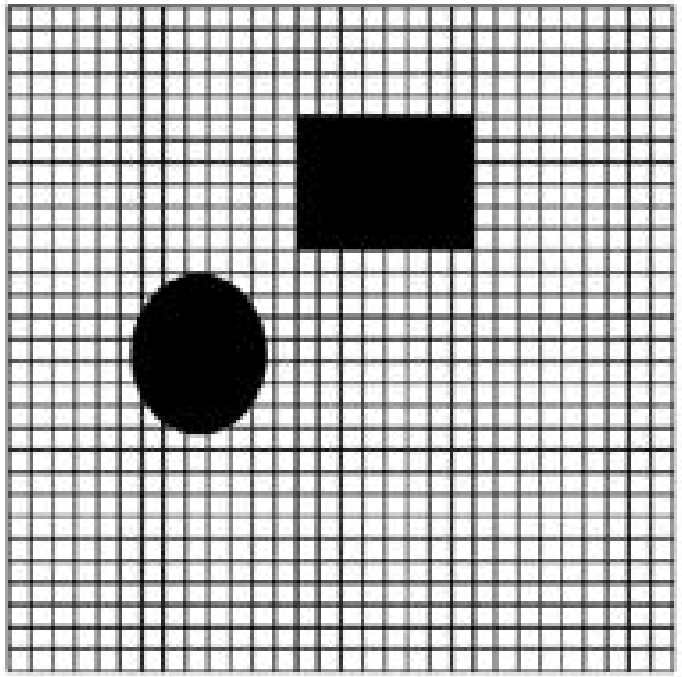
Topological maps

- Standard graph theory algorithms
  - Good abstract representation
  - Tradeoff in # of nodes
    - complexity vs. accuracy
  - Limited information

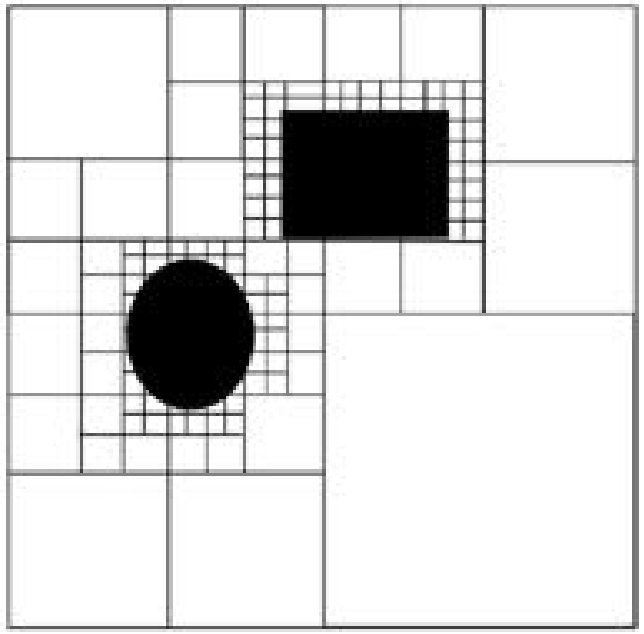


Global Motion Planning with Maps

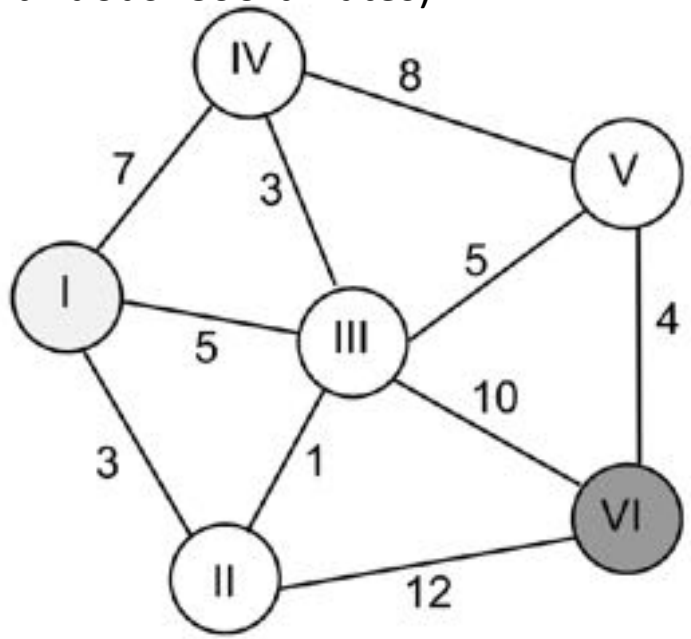
Occupancy Grid Map (discr. coord)



K-d Tree Map (Quadtree)



Topological Map
(Continuous Coordinates)



Overview of Algorithms for Path Planning

Optimal Control

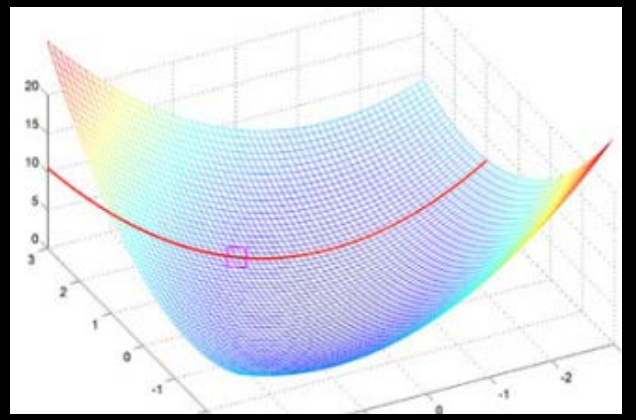
- Solves truly optimal solutions
- Intractable for moderately complex and nonconvex problems

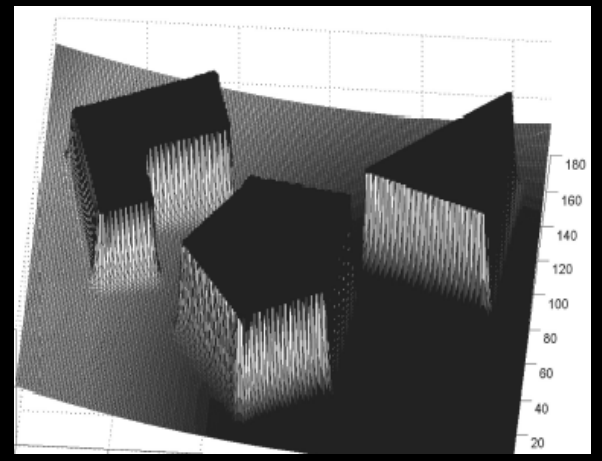
Potential Fields

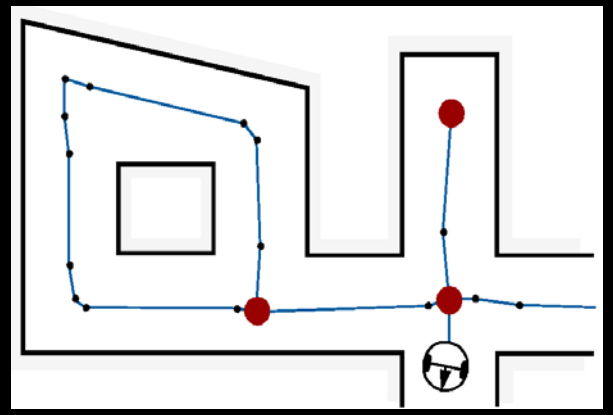
- Impose a math function over the work/C space
- Simple

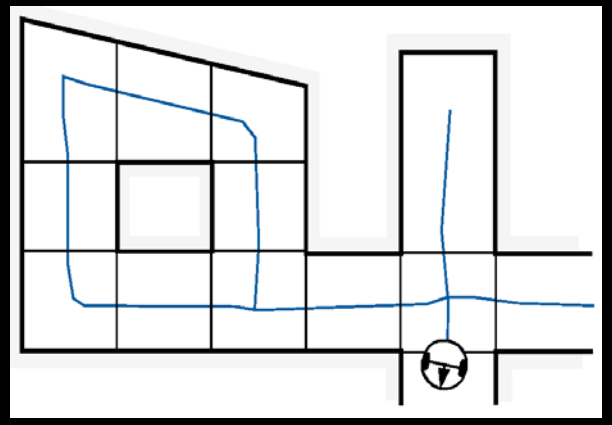
Graph Search

- Identify a set of edges between nodes within the free space



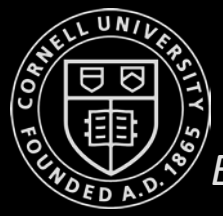






Prof. Kirstin Hagelskjær Petersen
kirstin@cornell.edu

Planning using Potential Fields

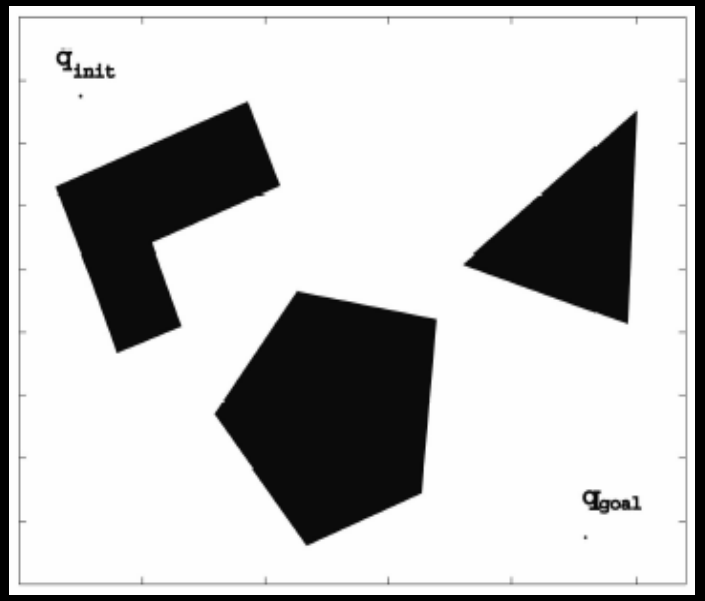


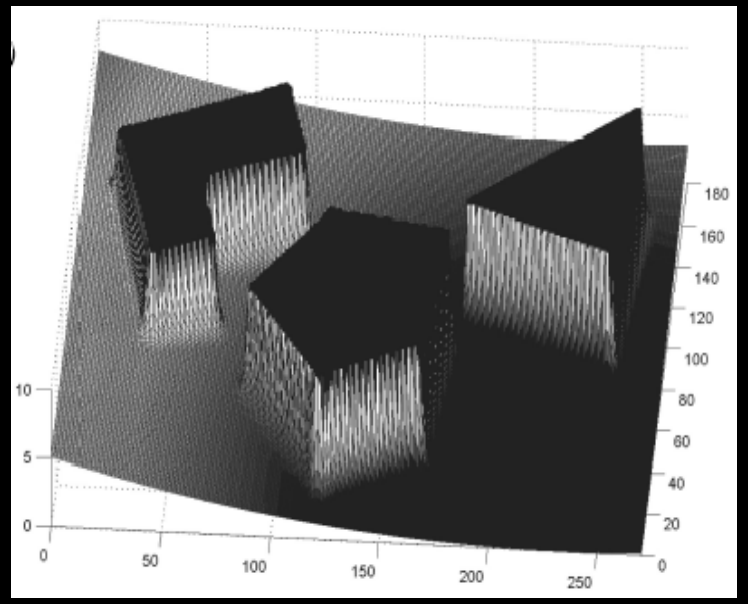
Planning using Potential Fields

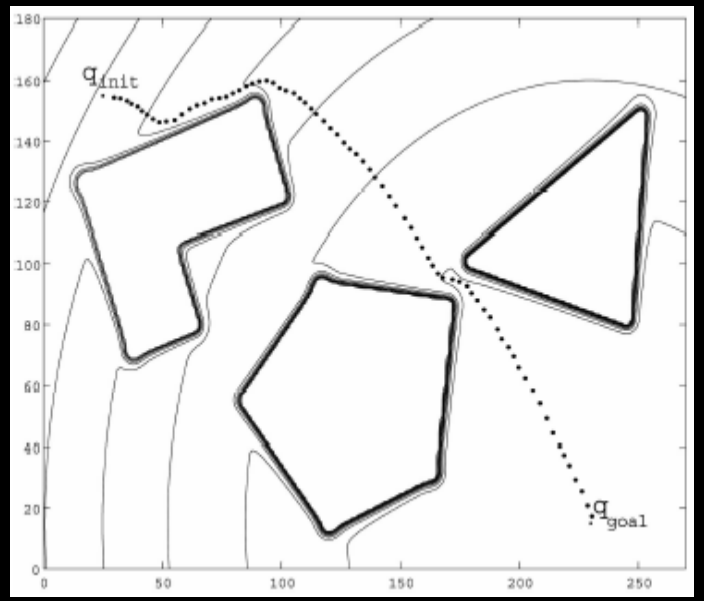
- Robot is treated as a point under the influence of a (continuous) artificial potential field
- Robot movement becomes similar to a ball rolling down a hill

Khatib, Stanford

Khatib, 1986

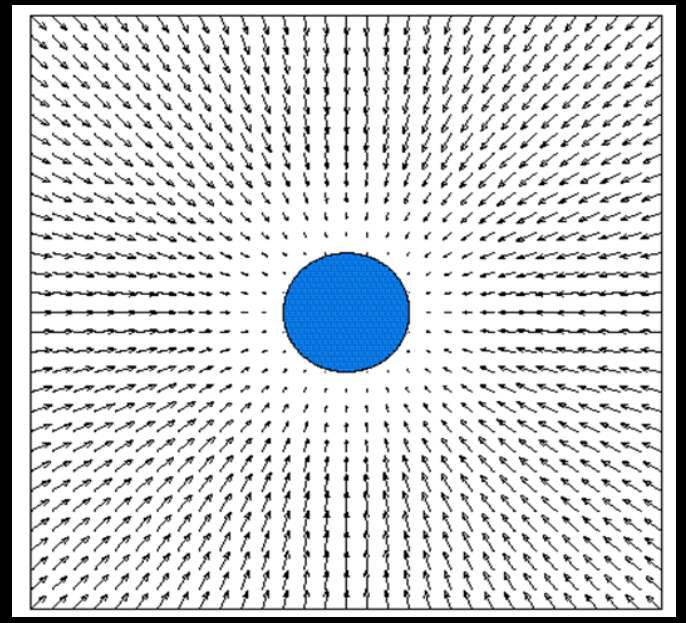


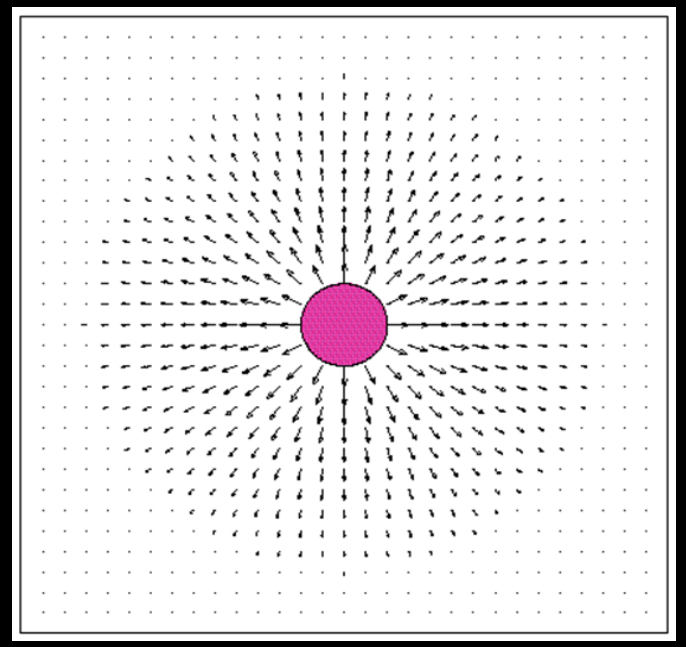




Planning using Potential Fields

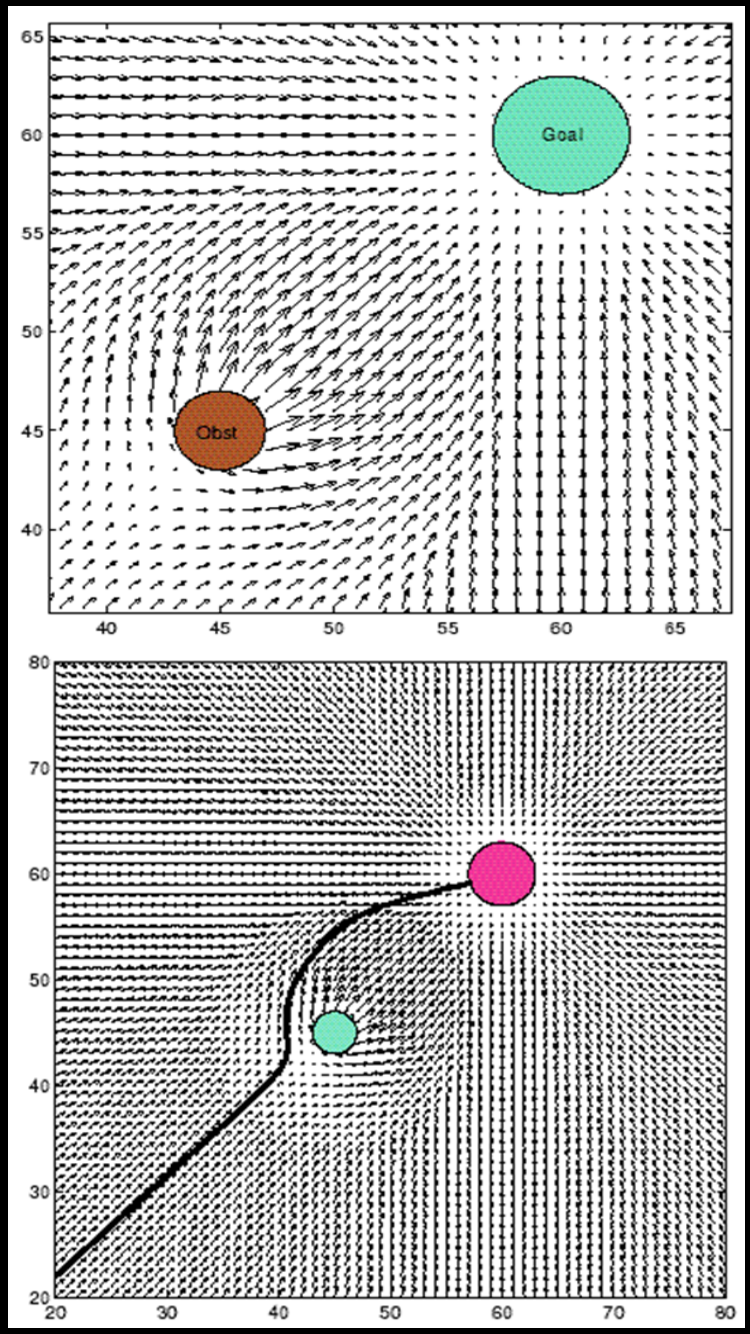
- The goal creates an attractive force
  - Modeled as a spring
  - Hooke's law: F = -kX
  - "Parabolic attractor"
  - $U_{att}(q) = k_{att}(q - q_{goal})^2$
  - $F_{att}(q) = -\nabla U_{att} = k_{att}(q - q_{goal})$
- Obstacles are repulsive forces
  - Modeled as charged particles
  - Coulomb's law: F= k $q_1 q_2$ / r^2
  - $U_{rep}(q) = \begin{cases} 0.5k_{rep}\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)^2 & ,if\ \rho(q) \leq \rho_0 \\ 0 & ,if\ \rho(q) \geq \rho_0 \end{cases}$
  - $F_{rep}(q) = \begin{cases} k_{rep}\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)\frac{1}{\rho(q)^2}\frac{q - q_{obst}}{\rho(q)} & ,if\ \rho(q) \leq \rho_0 \\ 0 & ,if\ \rho(q) \geq \rho_0 \end{cases}$





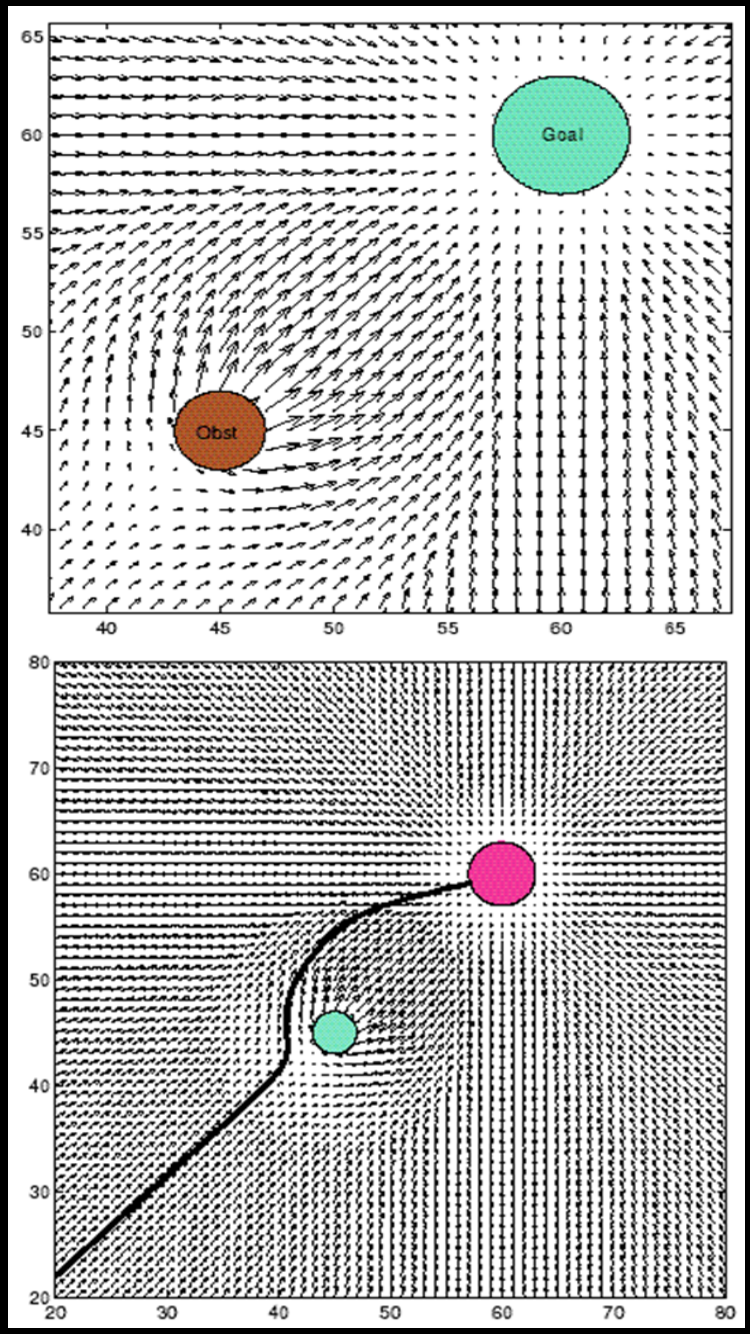
Planning using Potential Fields

- Goal generates attractive force
  - Modeled as a spring
  - Hooke's law: F = -kX
- Obstacle are repulsive forces
  - Modeled as charged particles
  - Coulomb's law: F= k $q_1 q_2$ / r^2
- Model navigation as the sum of forces on the robot
  - The overall potential field
    - $U(q) = U_{goal}(q) + \sum U_{obstacles}(q)$
  - Robot motion is proportional to induced force
    - $\mathrm{F(q)} = -\nabla U(q)$
  - e.g. 2 DOF robot will experience
    - $F(q) = -\nabla U(q) = \left(\frac{\partial U}{\partial x}, \frac{\partial U}{\partial y}\right)$



Planning using Potential Fields

- Goal generates attractive force
  - Modeled as a spring
  - Hooke’s law: $F = -kX$
- Obstacle are repulsive forces
  - Modeled as charged particles
  - Coulomb's law: $F= k\, q_1 q_2 / r^2$
- Model navigation as the sum of forces on the robot
- Pitfalls / local minima
  - U-shaped obstacles
  - Long walls
  - Solutions
    - Incorporate high-level planner
    - Incorporate procedural planner
    - Adapt the field to have gradual repulsion
    - Adding stochasticity



Overview of Algorithms for Path Planning

Optimal Control

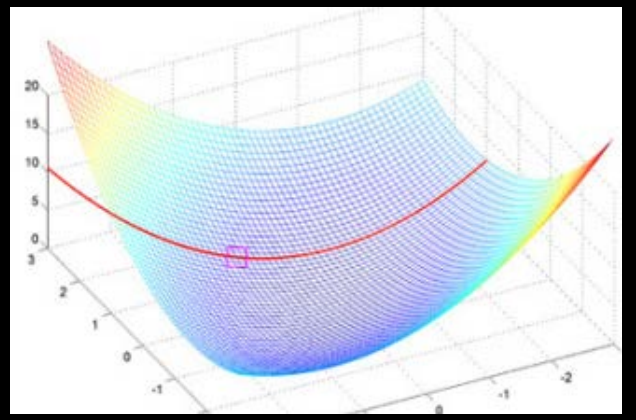
- Solves truly optimal solutions
- Intractable for moderately complex and nonconvex problems

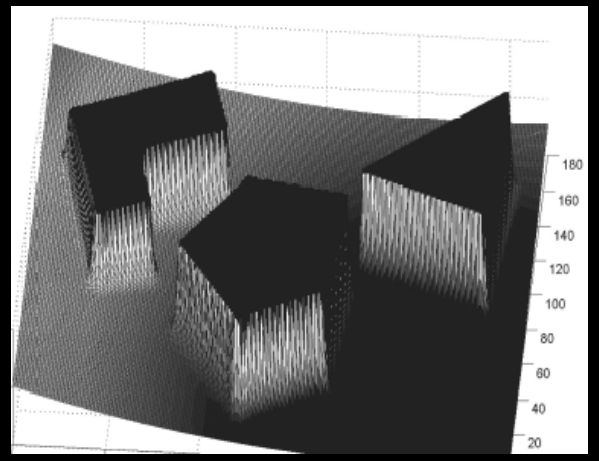
Potential Fields

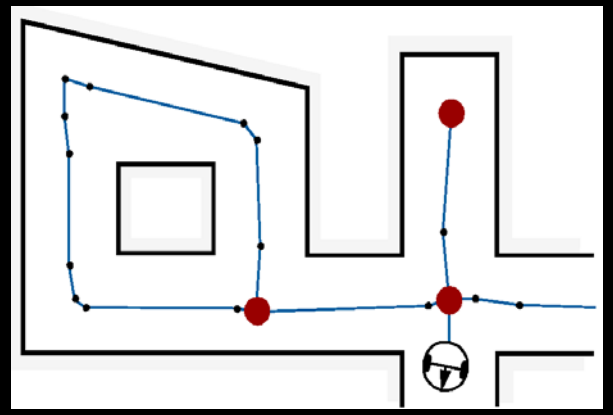
- Impose a math function over the work/C space
- Simple

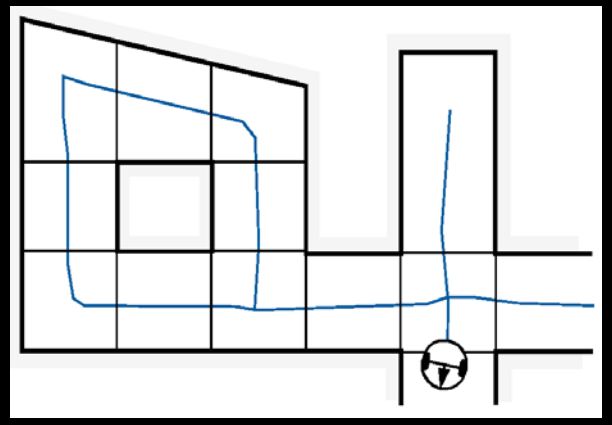
Graph Search

- Identify a set of edges between nodes within the free space



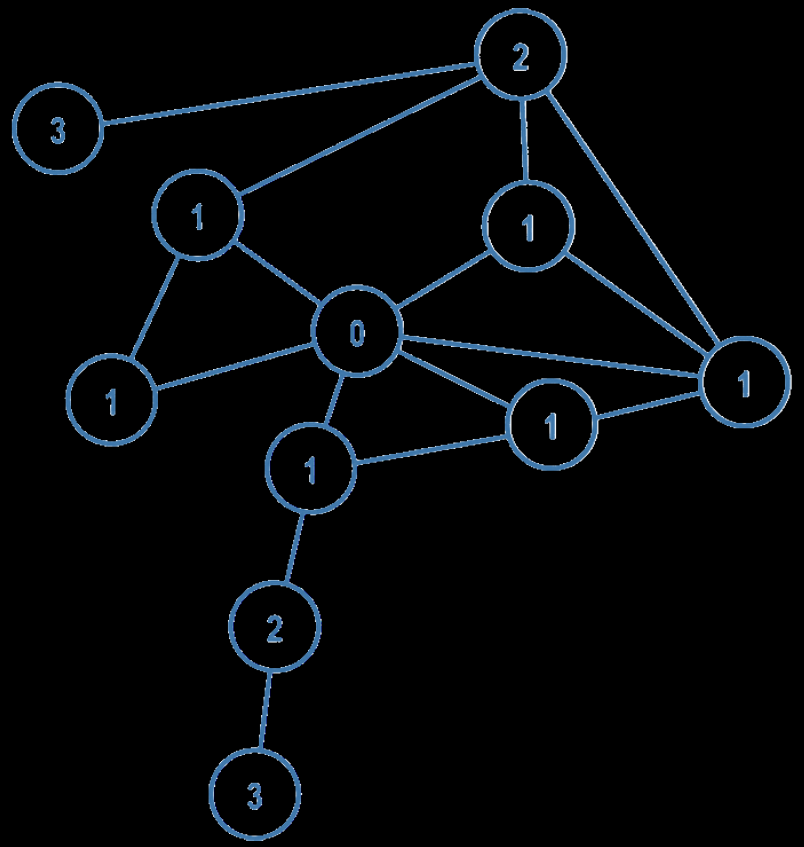






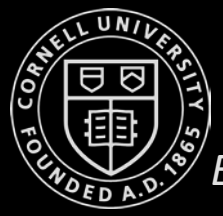
Graph Construction

- Transform continuous/discrete/topological map to a discrete graph
- Why graphs?
  - Model the path planning problem as a search problem
  - Graph theory has lots of tools
  - Real-time capable algorithms
  - Can accommodate for evolving maps



Prof. Kirstin Hagelskjær Petersen
kirstin@cornell.edu

Graph Construction

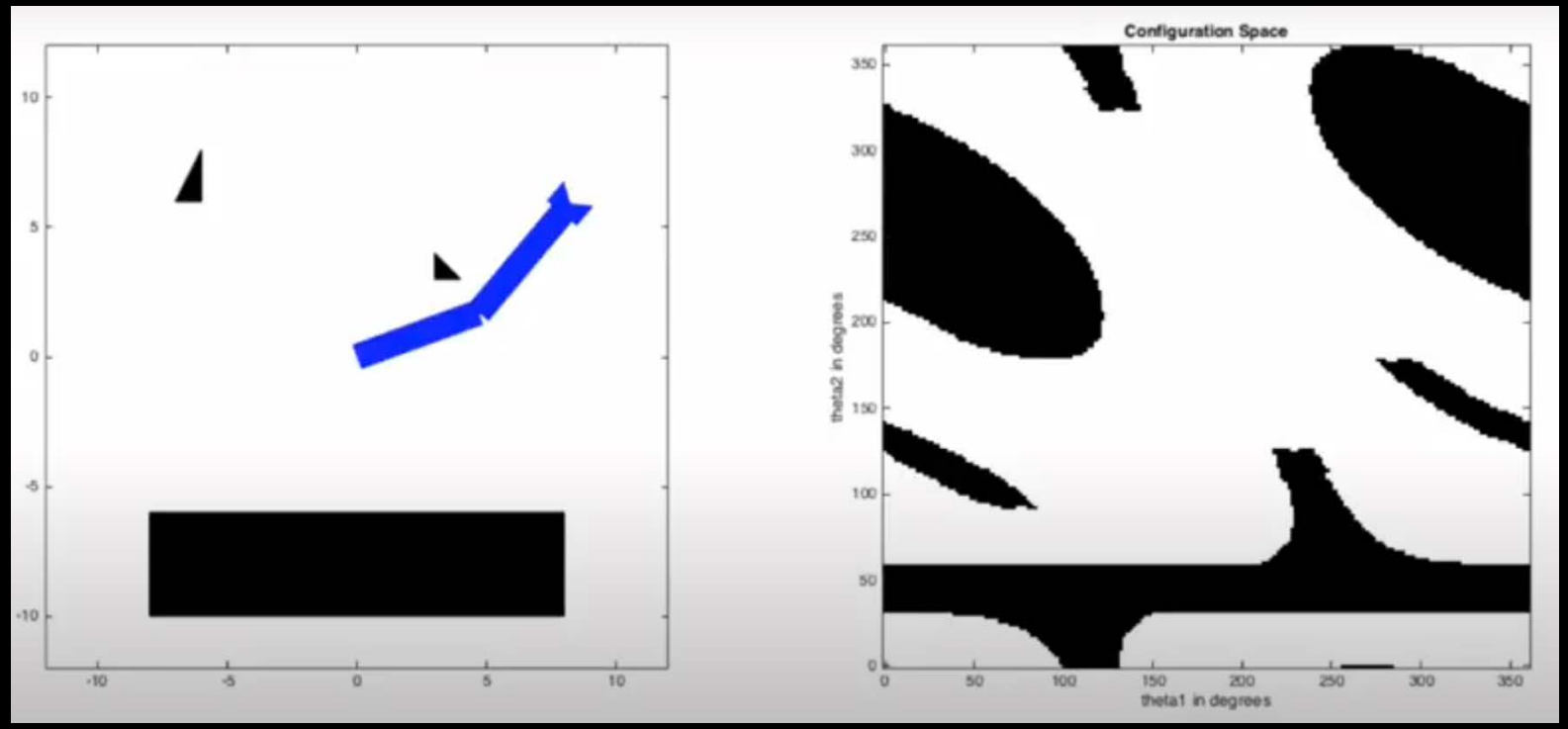


Modelling path planning as a graph search problem

Real system → Configuration Space → Map Representation → Graph Construction → Graph Search

Workspace

Configuration space



Map Representation and Graph Construction Methods

- Grid Worlds
  - Exact Cell Decomposition
  - Fixed Cell Decomposition
  - Approximate Cell Decomposition

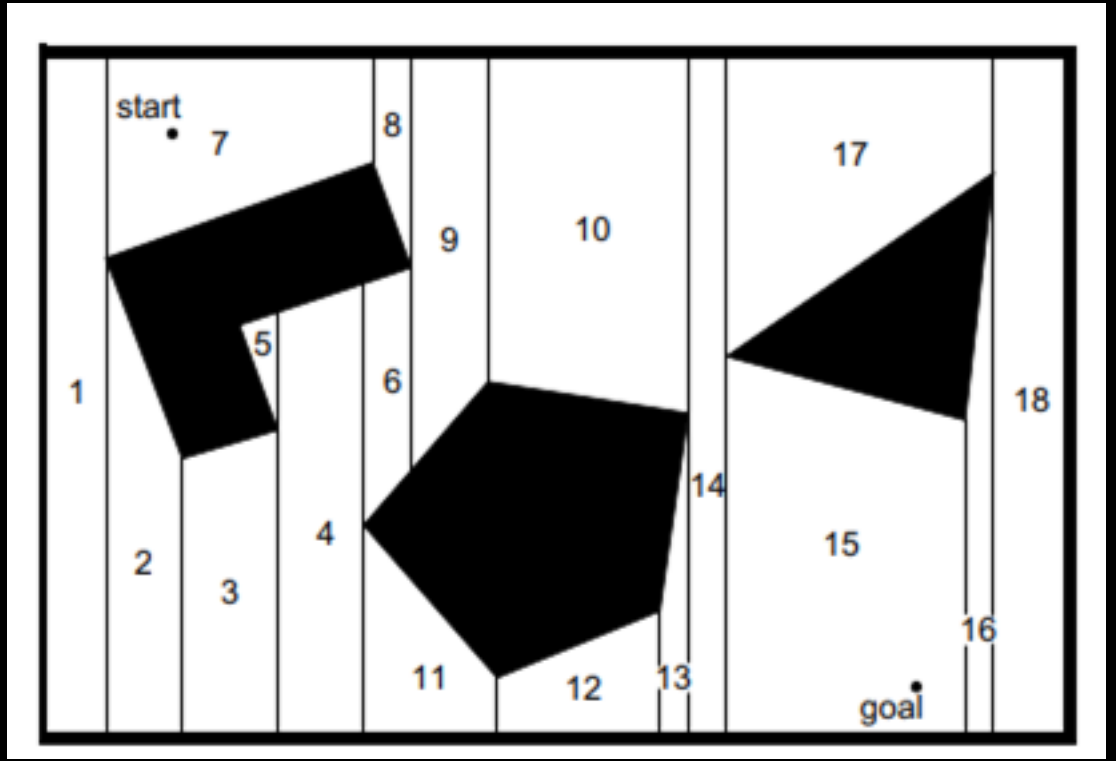
- Visibility graphs

- Probabilistic Roadmaps
- RRT

1. Divide space into simple, connected regions, or “cells”
2. Determine adjacency of open cells
3. Construct a connectivity graph
4. Find cells with initial and goal configuration
5. Search for a path in the connectivity graph to join them
6. From the sequence of cells, compute a path within each cell
   - e.g. passing through the midpoints of cell boundaries or by sequence of wall following movements

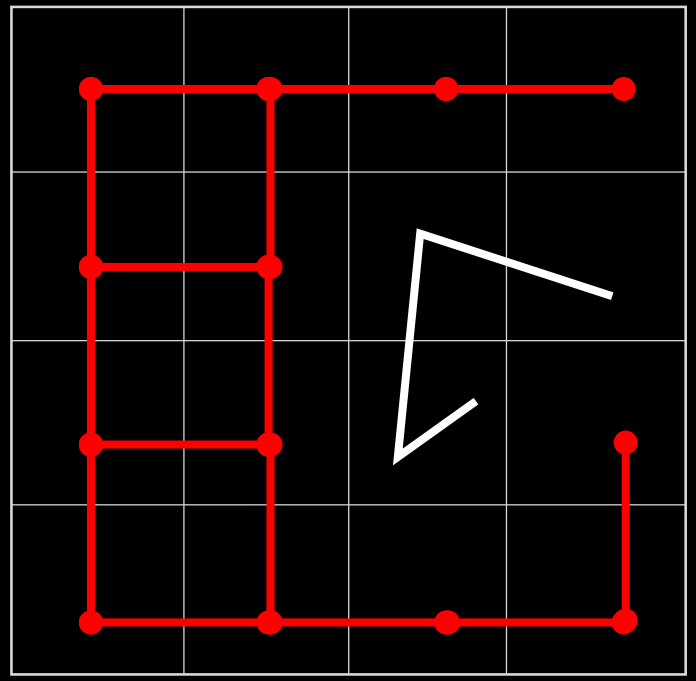
Exact Cell Decomposition

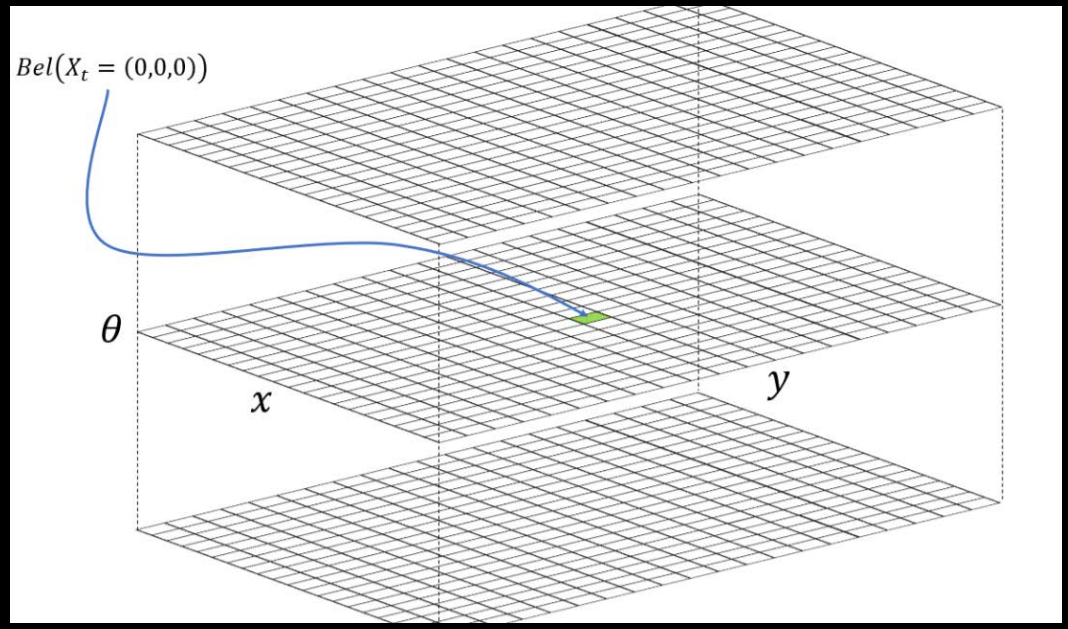
- Decomposition
  - Break down the map into cells based on geometric criticality
  - The map representation tessellates the space into areas of free space
- Abstraction
  - The position of the robot with the each cell of free space does not matter
  - What matters is the robot's ability to traverse from each free cell to adjacent free cell



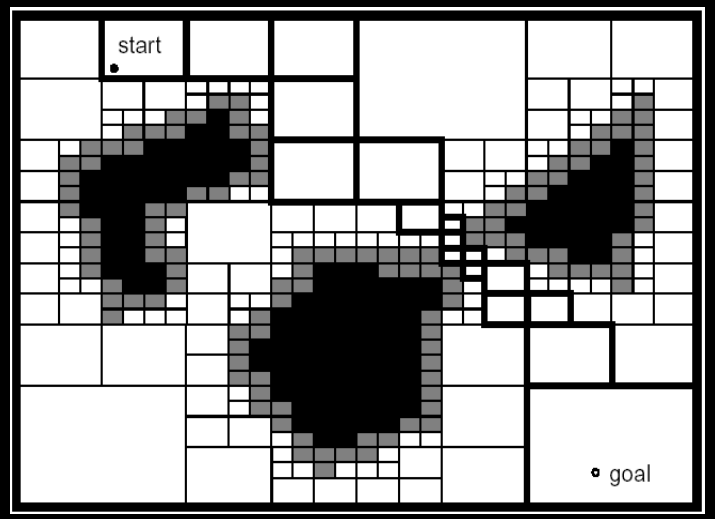
Approximate Cell Decomposition

(Lab 9)



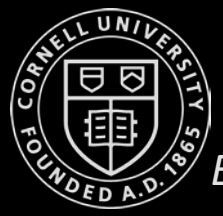


Adaptive Cell Decomposition



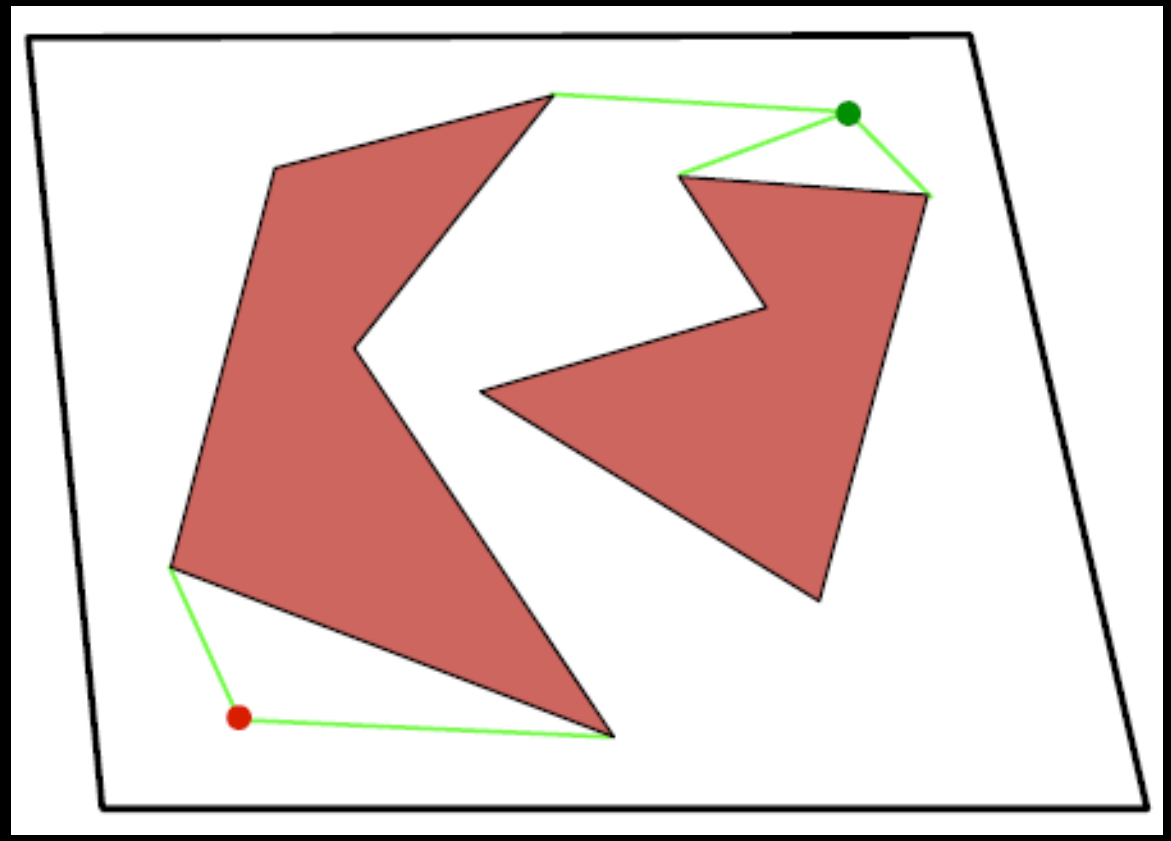
Map Representation and Graph Construction Methods

- Grid World
  - Exact Cell Decomposition
  - Fixed Cell Decomposition
  - Approximate Cell Decomposition
- Visibility graphs
- Probabilistic Roadmaps
- RRT



Visibility Graphs

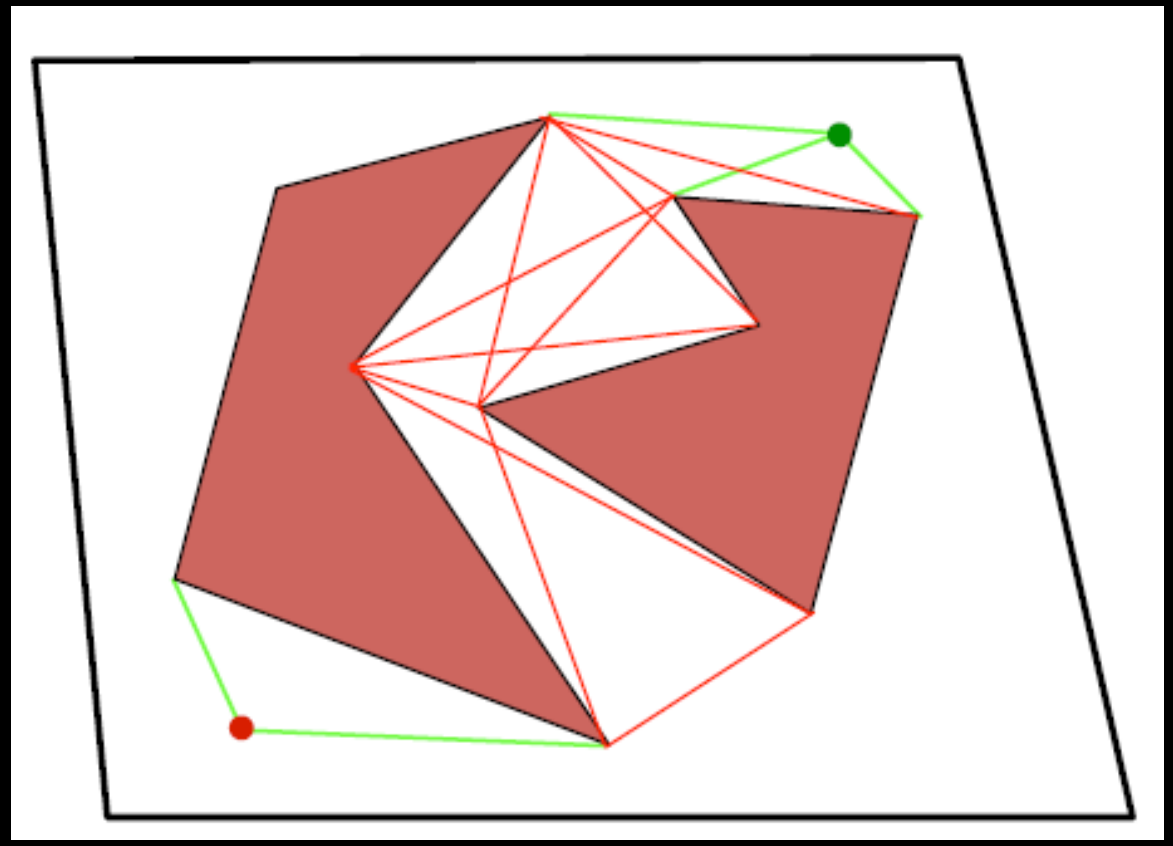
- Connect initial and goal locations with all visible vertices



Ioannis Rekleitis,
South Carolina

Visibility Graphs

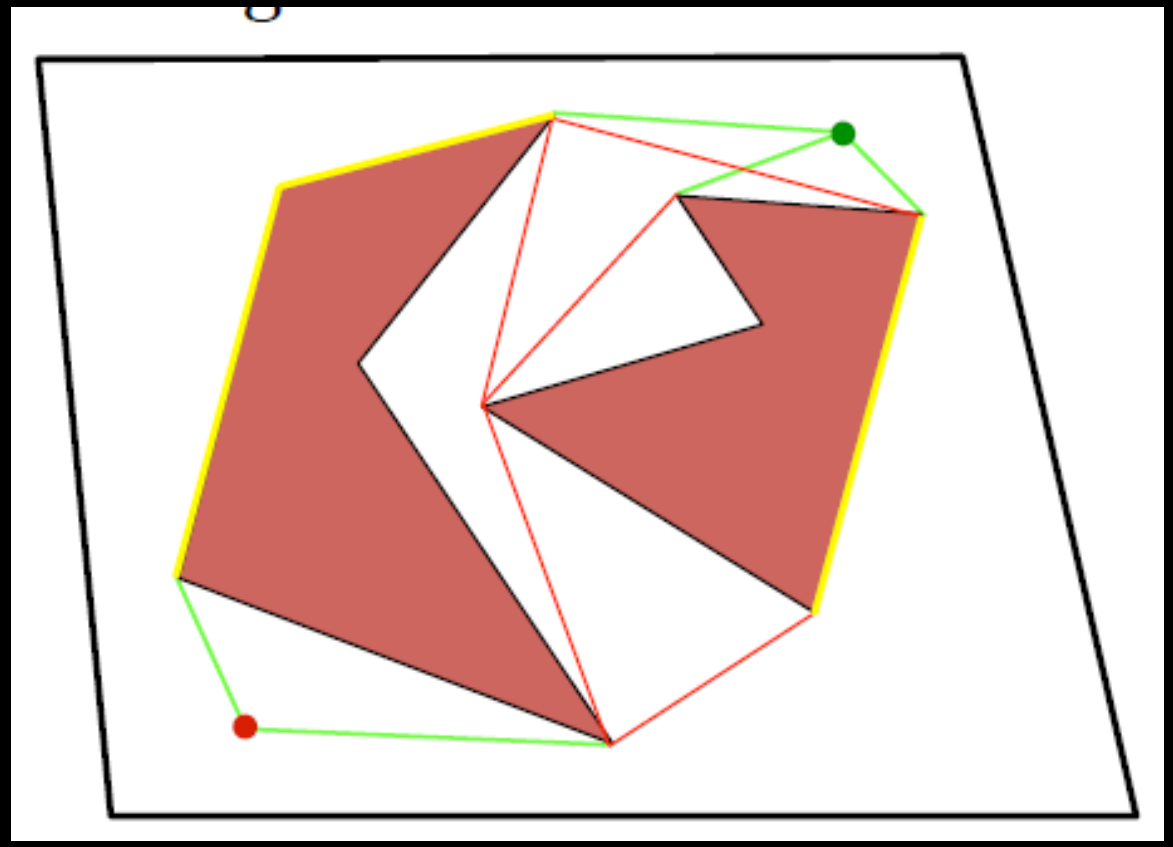
- Connect initial and goal locations with all visible vertices
- Connect each obstacle vertex to every visible obstacle vertex



Ioannis Rekleitis,
South Carolina

Visibility Graphs

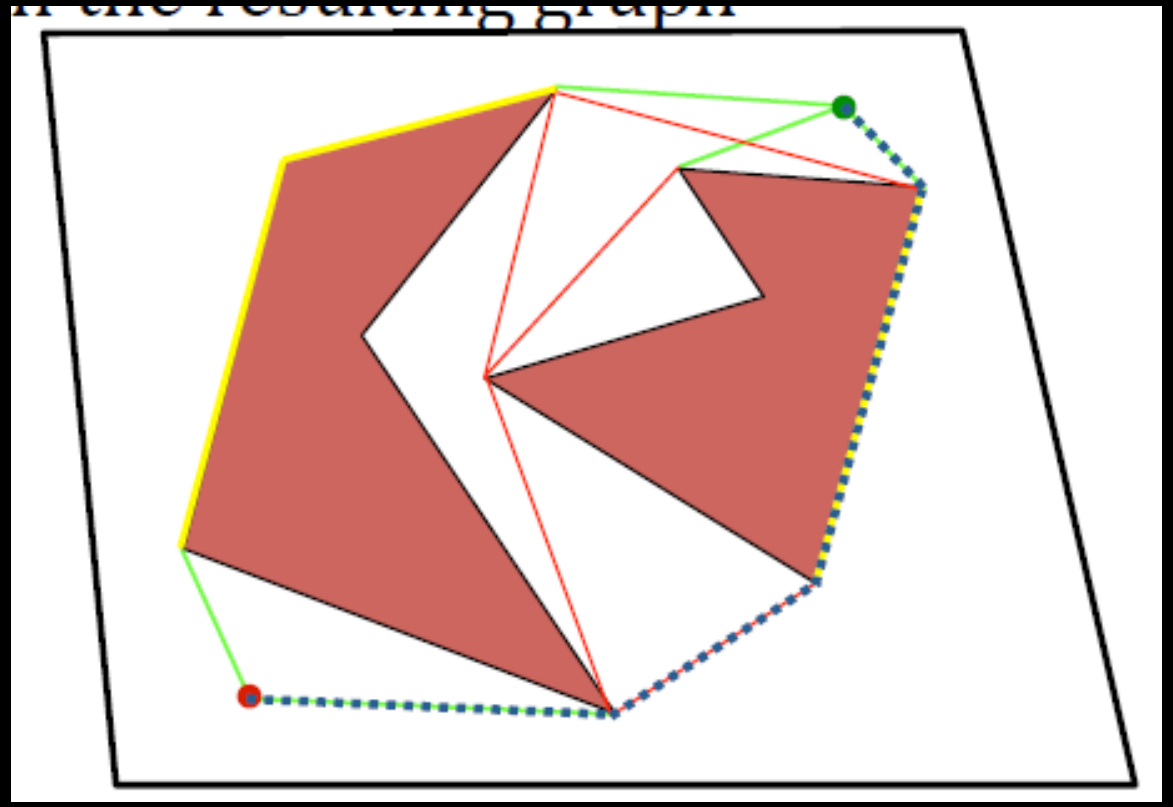
- Connect initial and goal locations with all visible vertices
- Connect each obstacle vertex to every visible obstacle vertex
- Remove edges that intersect the interior of an obstacle



Ioannis Rekleitis,
South Carolina

Visibility Graphs

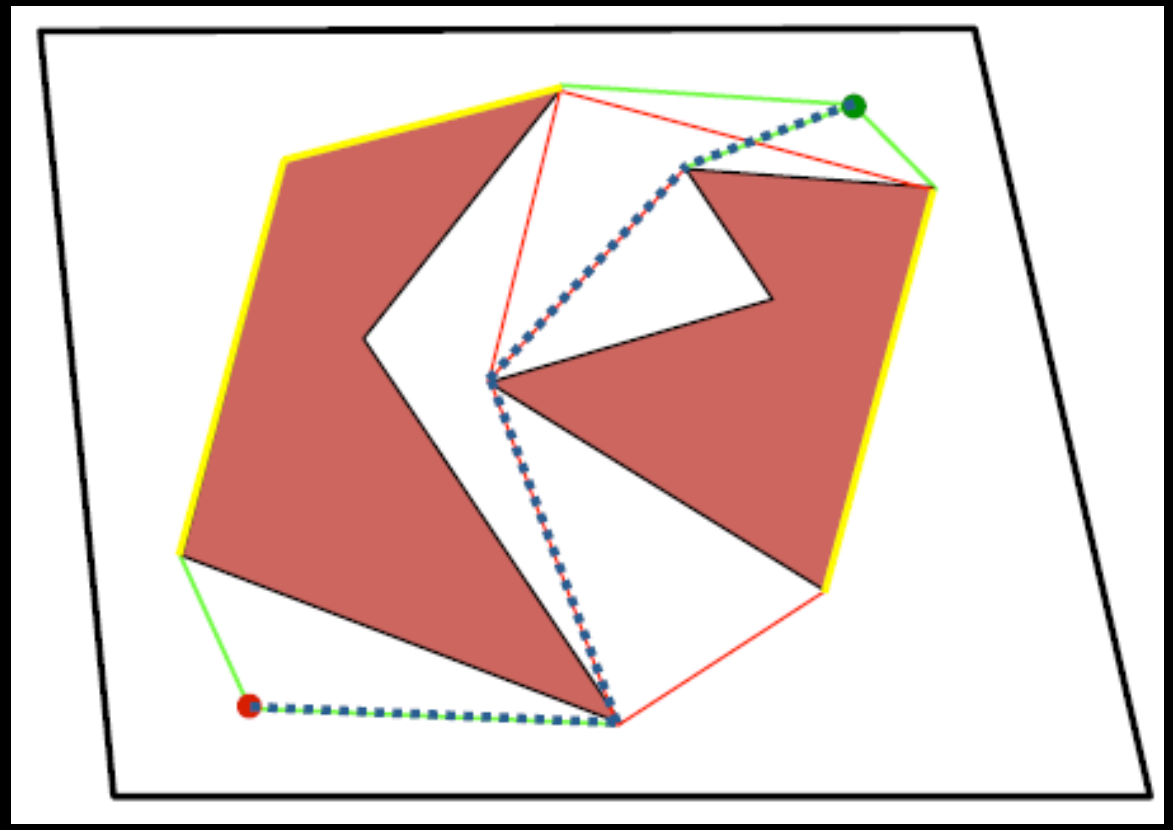
- Connect initial and goal locations with all visible vertices
- Connect each obstacle vertex to every visible obstacle vertex
- Remove edges that intersect the interior of an obstacle
- Plan on the resulting graph



Ioannis Rekleitis,
South Carolina

Visibility Graphs

- Connect initial and goal locations with all visible vertices
- Connect each obstacle vertex to every visible obstacle vertex
- Remove edges that intersect the interior of an obstacle
- Plan on the resulting graph



Ioannis Rekleitis,
South Carolina

Map Representation and Graph Construction Methods

- Graph Construction
  - Exact Cell Decomposition
  - Fixed Cell Decomposition
  - Approximate Cell Decomposition

- Visibility graphs

- Probabilistic Roadmaps
- RRT

Configuration Space Visualization | Dror Atariah & Günter Rote, Freie Universität Berlin, March 2012

Sampling-Based Planners

- Explicit geometry-based planners are impractical in high dimensional spaces
- Sampling-based planners
    - Often efficient in high dimensional spaces
    - Rather than computing the C-Space explicitly, we sample it
    - Simply need to know if a robot configuration is in collision
        - Collision detection is a separate module which can be tailored to the application
        - As collision detection improves, so do these algithms

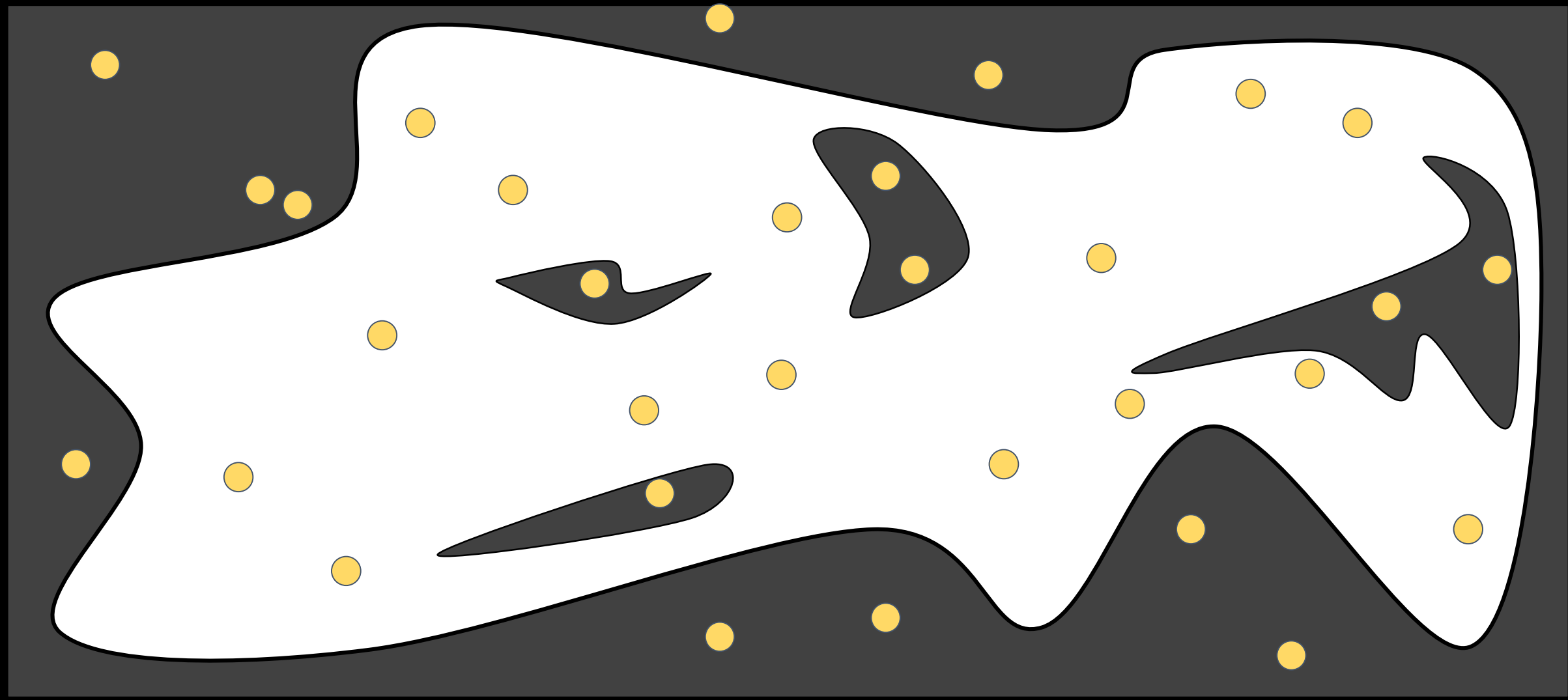
Prof. Kirstin Hagelskjær Petersen
kirstin@cornell.edu

Probabilistic Roadmaps

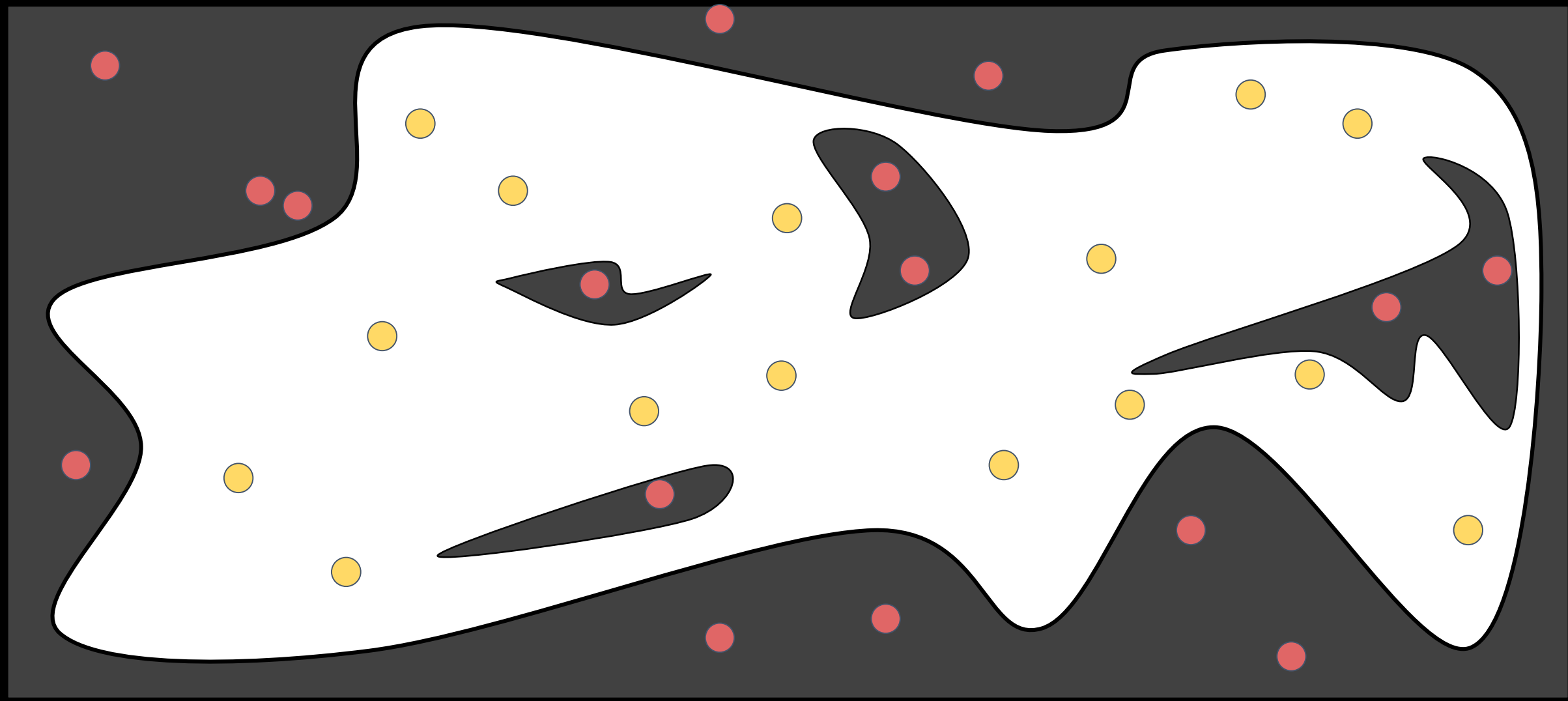
Lydia Kavraki, 1996
Rice Univeristy

Probabilistic Roadmap

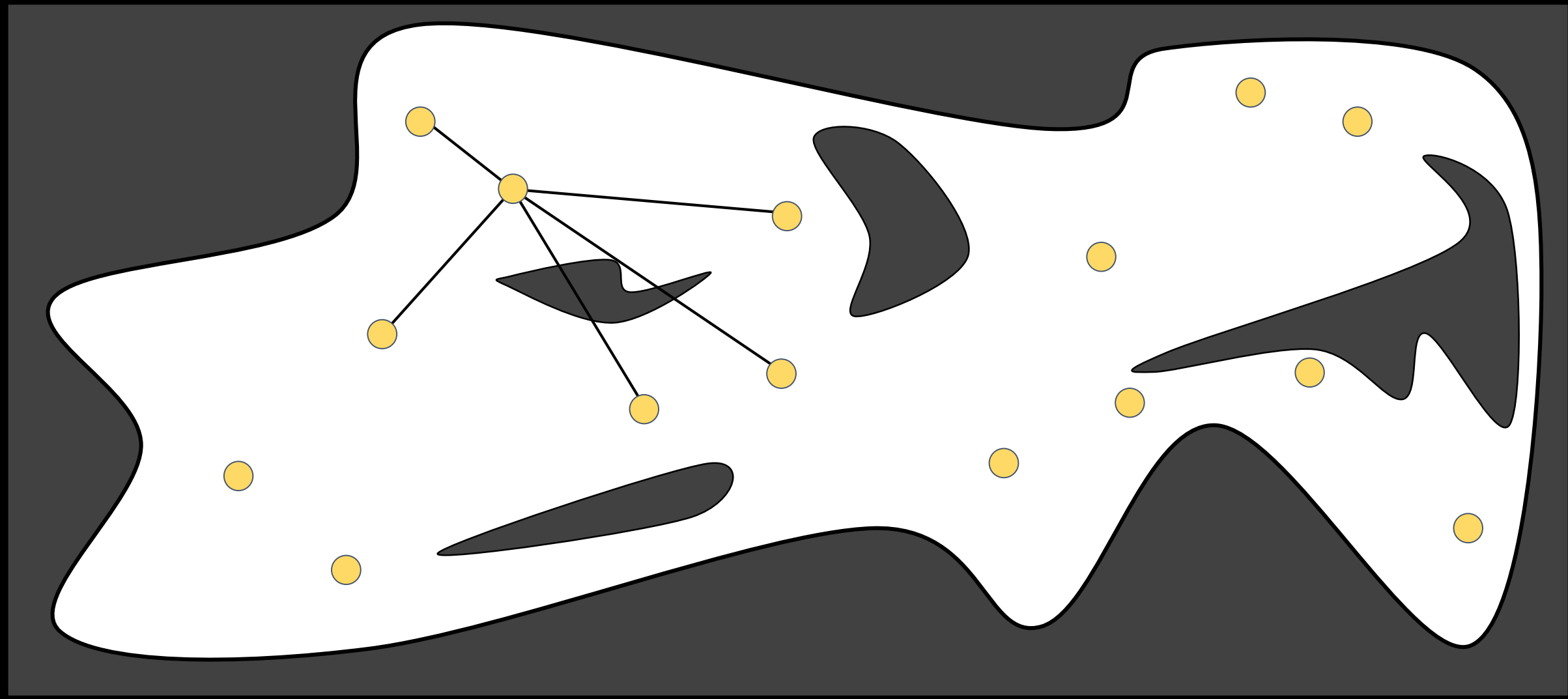
Probabilistic Roadmap



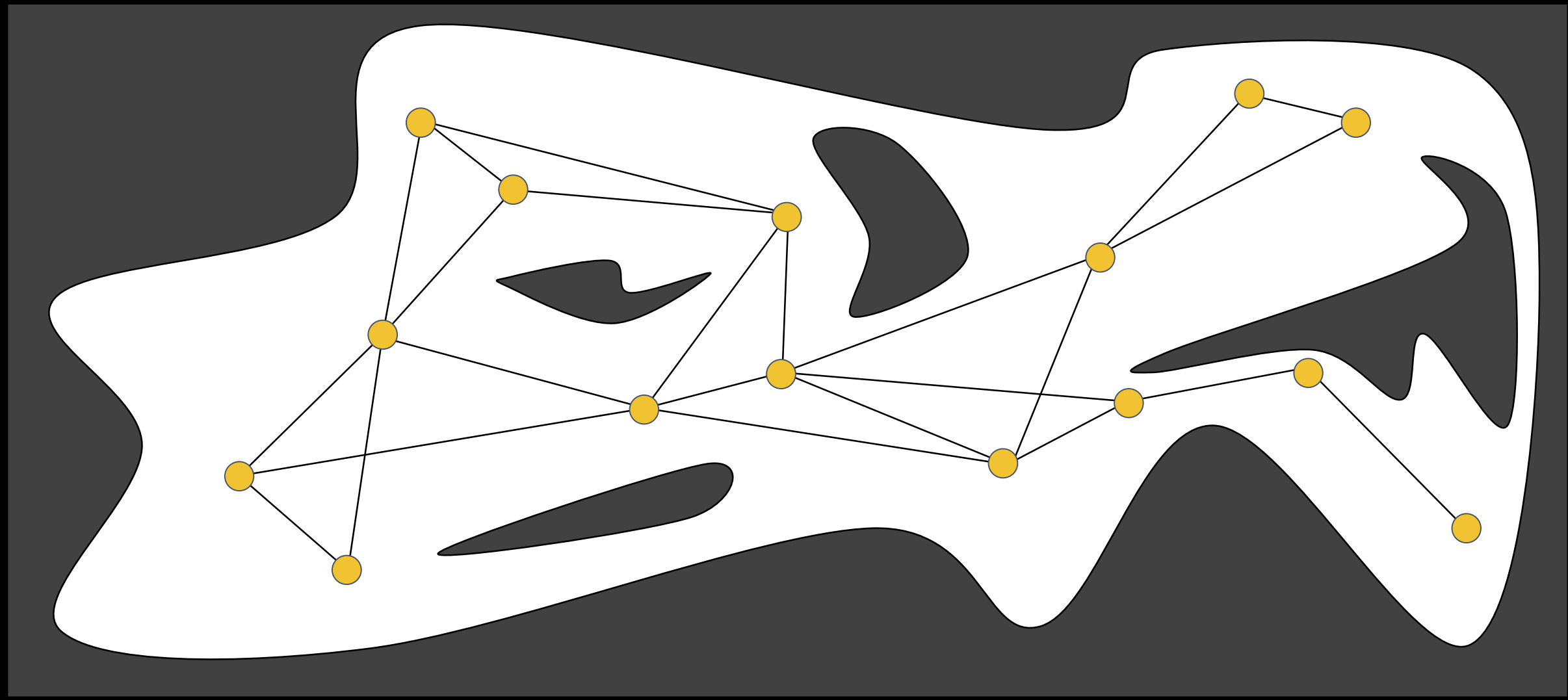
Probabilistic Roadmap



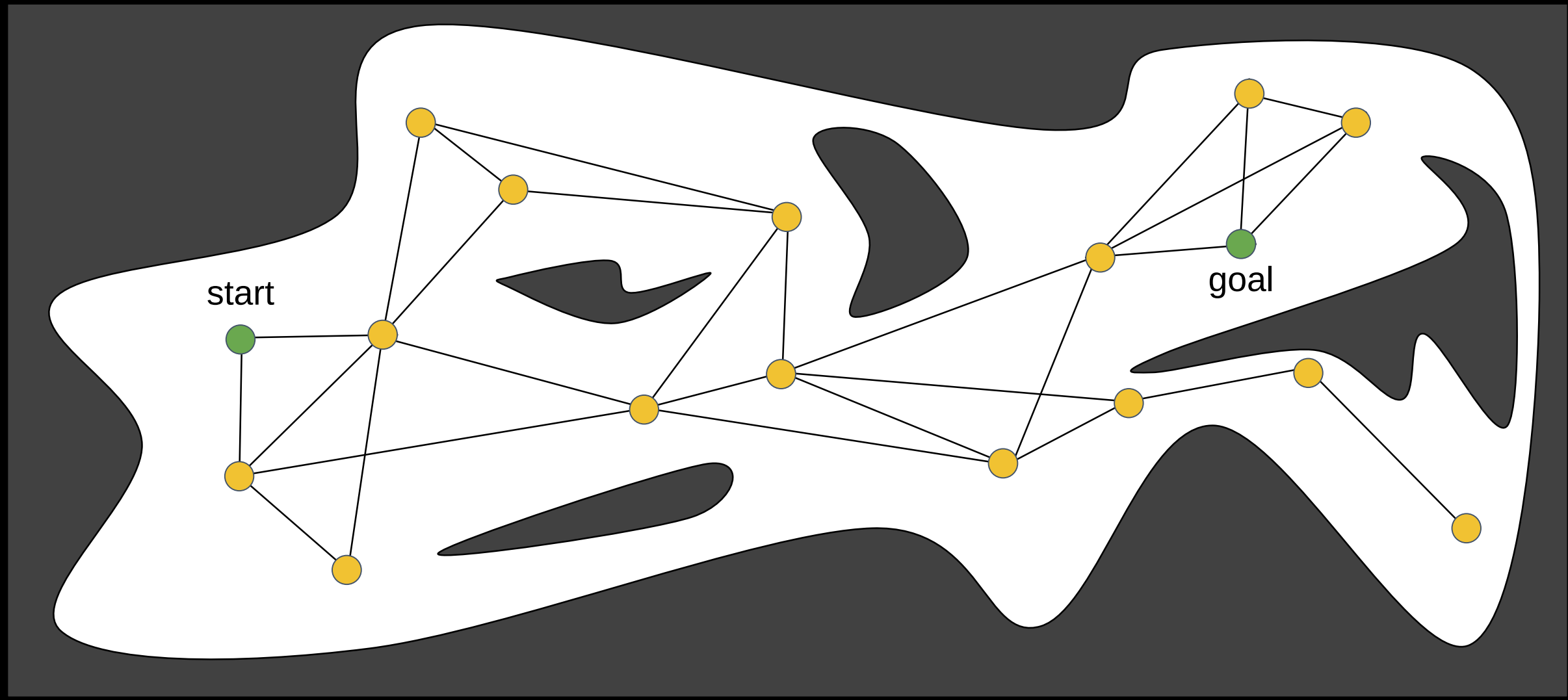
Probabilistic Roadmap



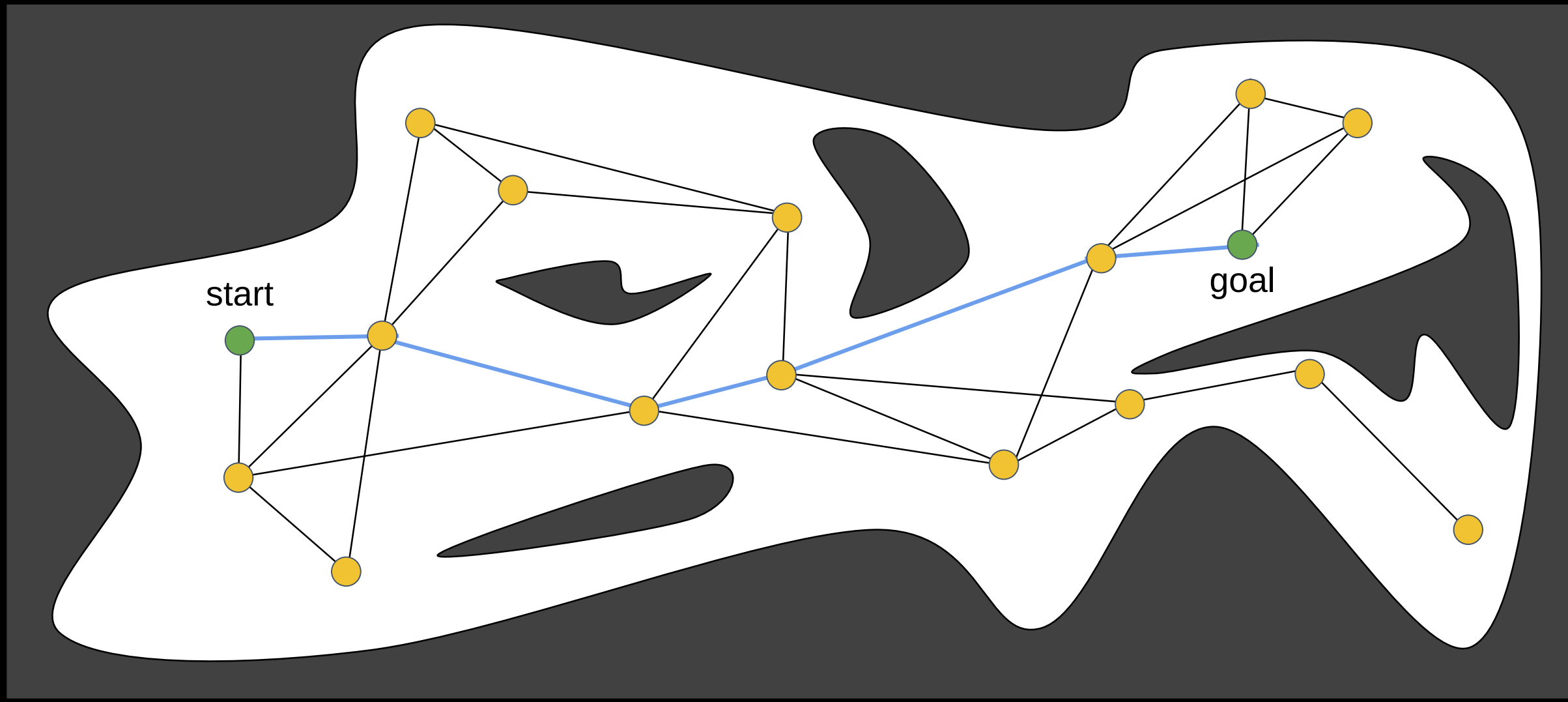
Probabilistic Roadmap



Probabilistic Roadmap



Probabilistic Roadmap



Probabilistic Roadmap – Constructing the graph

- Initially empty Graph G
- A configuration q is randomly chosen
- If q ∈ Q_{free} then add to G
  - <need collision detection>
- Repeat until N vertices chosen
- For each q, select k closest neighbors
- Local planner, Δ, connects q to neighbor q'
- If connect successful (i.e. collision free local path), add edge (q, q')

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 $G = (V, E)$

```
1:  V ← ∅
2:  E ← ∅
3:  while |V| < n do
4:     repeat
5:        q ← a random configuration in Q
6:     until q is collision-free
7:     V ← V ∪ {q}
8:  end while
9:  for all q ∈ V do
10:    N_q ← the k closest neighbors of q chosen from V according to dist
11:    for all q' ∈ N_q do
12:       if (q, q') ∉ E and Δ(q, q') ≠ NIL then
13:          E ← E ∪ {(q, q')}
14:       end if
15:    end for
16: end for
```

Probabilistic Roadmap – Constructing the graph

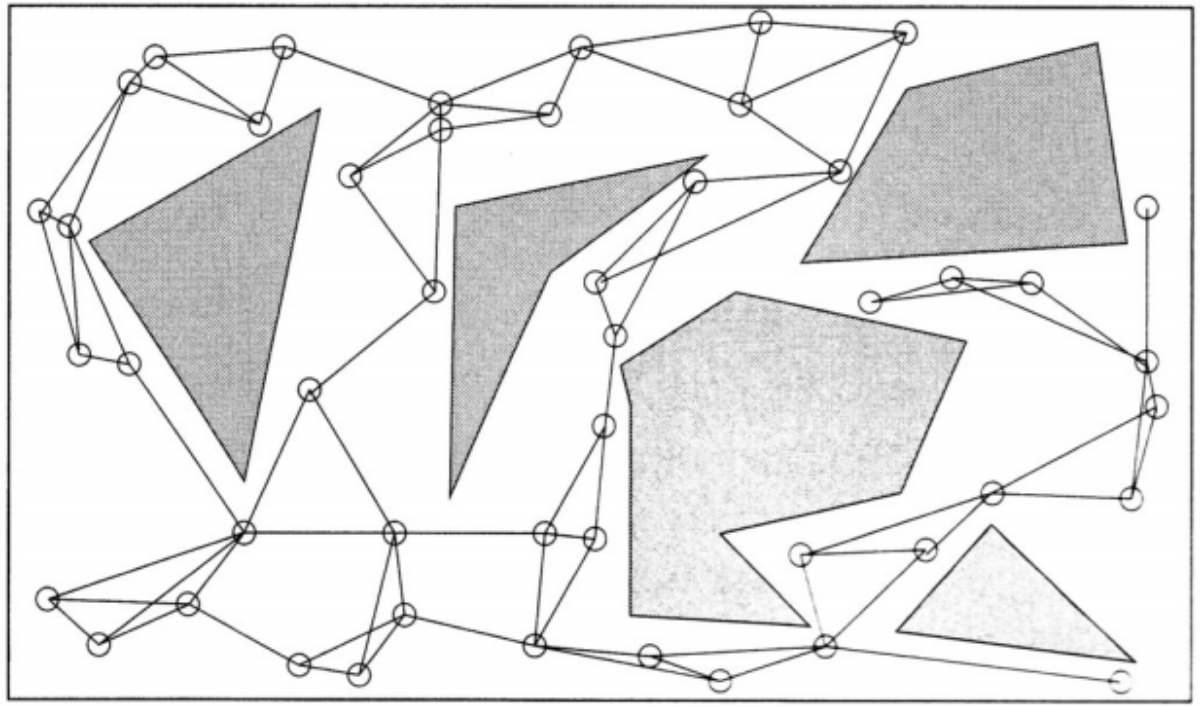


Figure 7.3 An example of a roadmap for a point robot in a two-dimensional Euclidean space. The gray areas are obstacles. The empty circles correspond to the nodes of the roadmap. The straight lines between circles correspond to edges. The number of k closest neighbors for the construction of the roadmap is three. The degree of a node can be greater than three since it may be included in the closest neighbor list of many nodes.

Probabilistic Roadmap – Finding the Path

- Connect q_{init} and q_{goal} to the roadmap
- Find k nearest neighbors of q_{init} and q_{goal} in roadmap, plan local path Δ
- Repeat until graphs are connected

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

1: $N_{q_{init}} \leftarrow$ the k closest neighbors of q_{init} from V according to *dist*
2: $N_{q_{goal}} \leftarrow$ the k closest neighbors of q_{goal} from V according to *dist*
3: $V \leftarrow \{q_{init}\} \cup \{q_{goal}\} \cup V$
4: set q' to be the closest neighbor of q_{init} in $N_{q_{init}}$
5: **repeat**
6: **if** $\Delta(q_{init}, q') \neq$ `NIL` **then**
7: $E \leftarrow (q_{init}, q') \cup E$
8: **else**
9: set q' to be the next closest neighbor of q_{init} in $N_{q_{init}}$
10: **end if**
11: **until** a connection was succesful or the set $N_{q_{init}}$ is empty
12: set q' to be the closest neighbor of q_{goal} in $N_{q_{goal}}$
13: **repeat**
14: **if** $\Delta(q_{goal}, q') \neq$ `NIL` **then**
15: $E \leftarrow (q_{goal}, q') \cup E$
16: **else**
17: set q' to be the next closest neighbor of q_{goal} in $N_{q_{goal}}$
18: **end if**
19: **until** a connection was succesful or the set $N_{q_{goal}}$ is empty
20: $P \leftarrow$ shortest path(q_{init}, q_{goal}, G)
21: **if** P is not empty **then**
22: **return** P
23: **else**
24: **return** `failure`
25: **end if**

Probabilistic Roadmap – Finding the Path

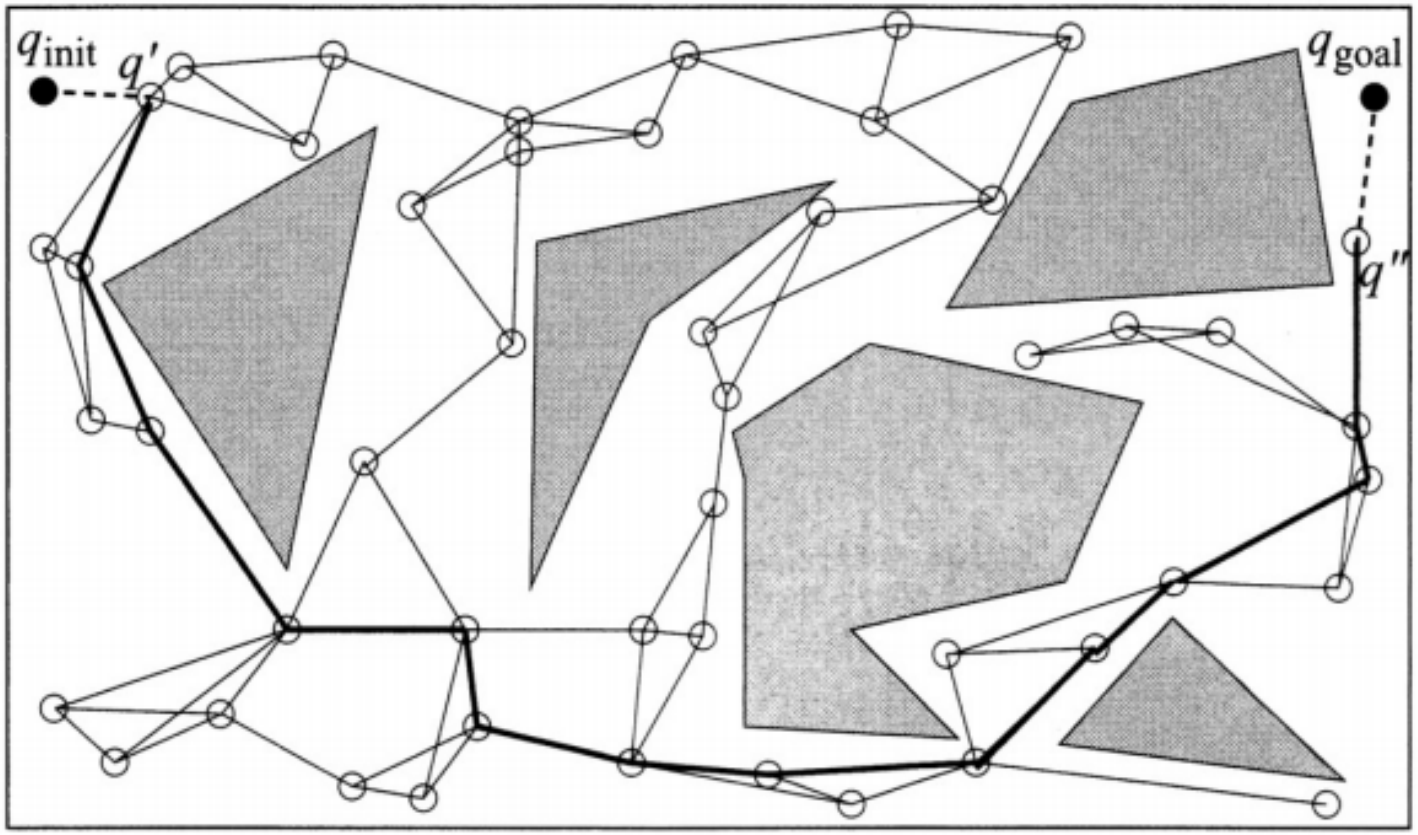
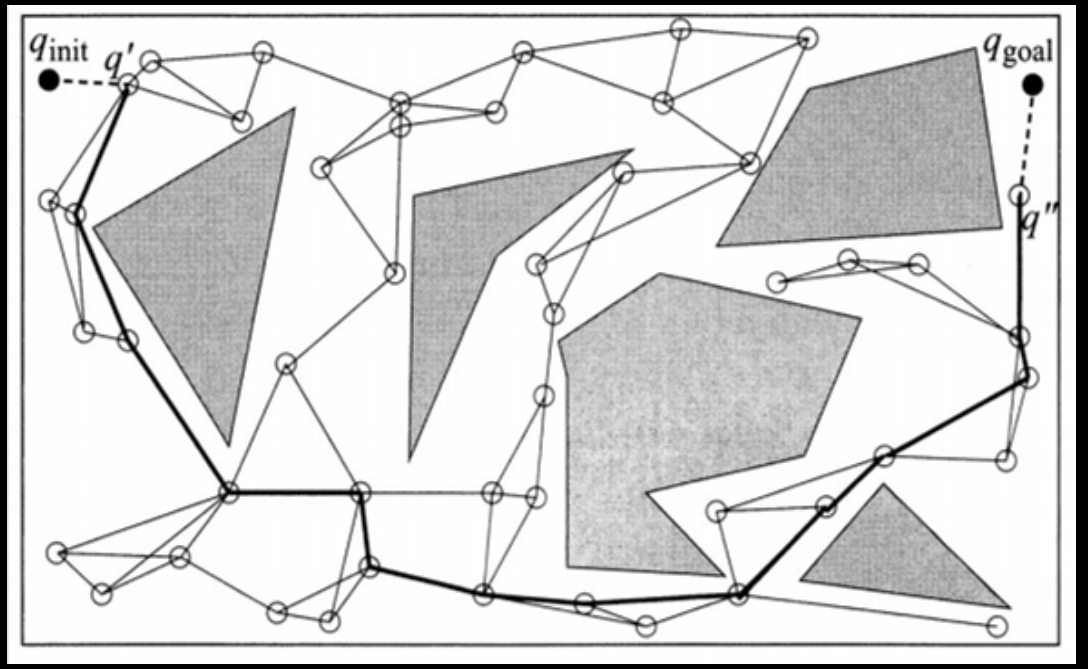


Figure 7.4 An example of how to solve a query with the roadmap from figure 7.3. The configurations q_{init} and q_{goal} are first connected to the roadmap through q' and q''. Then a graph-search algorithm returns the shortest path denoted by the thick black lines.

Probabilistic Roadmap – Considerations

- Single-Query / Multi-Query
- How are nodes placed?
  - Uniform sampling strategies
  - Non-uniform sampling strategies
- How are local neighbors found?
- How is collision detection performed?
  - Dominates time consumption in PRMs



Probabilistic Roadmap – Ultra Fast

- "Robot Motion Planning on a Chip", Murray et al. RSS 2016
- Company: Real Time Robotics
  - PRM on an FPGA
  - Collision detection circuits on each edge in logic gates for massive parallel operation
  - 6DOF planning in <1ms

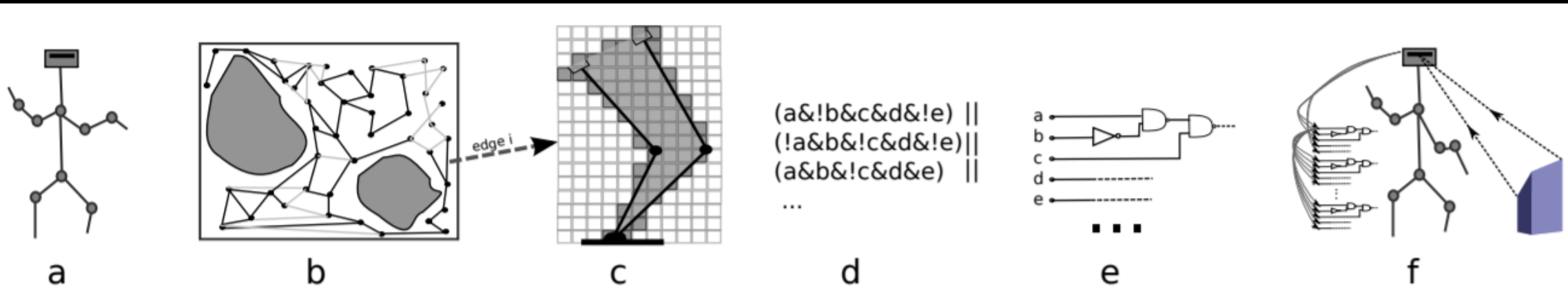
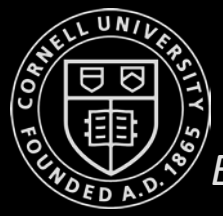


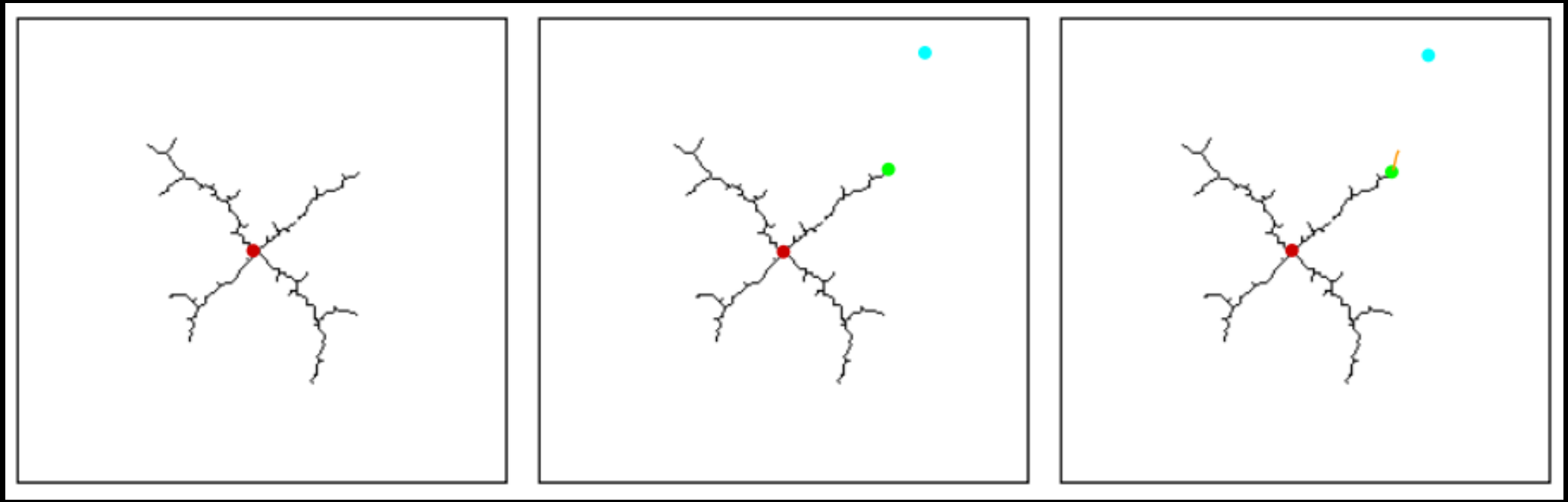
Fig. 3: Our process for producing robot-specific motion planning circuitry. Given a robot description (a), we construct a PRM (b), most likely subsampled for coverage from a much larger PRM. We discretize the robot's reachable space into depth pixels and, for each edge i on the PRM, precompute all the depth pixels that collide with the corresponding swept volume (c). We use these values to construct a logical expression that, given the coordinates of a depth pixel encoded in binary, returns `true` if that depth pixel collides with edge i (d); this logical expression is optimized and used to build a collision detection circuit (CDC) (e). For each edge in the PRM there is one such circuit. When the robot wishes to construct a motion plan, it perceives its environment, determines which depth pixels correspond to obstacles, and transmits their binary representations to every CDC (f). All CDCs perform collision detection *simultaneously, in parallel* for each depth pixel, storing a bit which indicates

Prof. Kirstin Hagelskjær Petersen
kirstin@cornell.edu

Rapidly Exploring Random Trees (RRT)



Rapidly Exploring Random Trees (RRT)



1. Maintain a tree of rooted at the starting point
2. Choose a point at random from free space
3. Find the closest configuration already in the tree
4. Extend the tree in the direction of the new configuration

Rapidly Exploring Random Trees (RRT)

```
1. Algorithm BuildRRT
2.   Input: Initial configuration q_init, number of vertices K,
     incremental distance Δq)
3.   Output: RRT graph G
4.   G.init(q_init)
5.   for k = 1 to K
6.     qrand ← RAND_CONF()
7.     qnear ← NEAREST_VERTEX(qrand, G)
8.     qnew ← NEW_CONF(qnear, qrand, Δq)
9.     G.add_vertex(qnew)
10.    G.add_edge(qnear, qnew)
11.  return G
```

Rapidly Exploring Random Trees (RRT) – Uniform/biased sampling

Aaron Becker, UH, Wolfram Player example

S. LaValle, UIUC / Oculus

Rapidly Exploring Random Trees (RRT) - Considerations

- **Sensitive to step-size (Δq)**
  - Small: many nodes, closely spaced, slowing down nearest neighbor computation
  - Large: Increased risk of suboptimal plans / not finding a solution
- **How are samples chosen?**
  - Uniform sampling may need too many samples to find the goal
  - Biased sampling towards goal can ease this problem
- **How are local paths generated?**
- **How are closest neighbors found?**

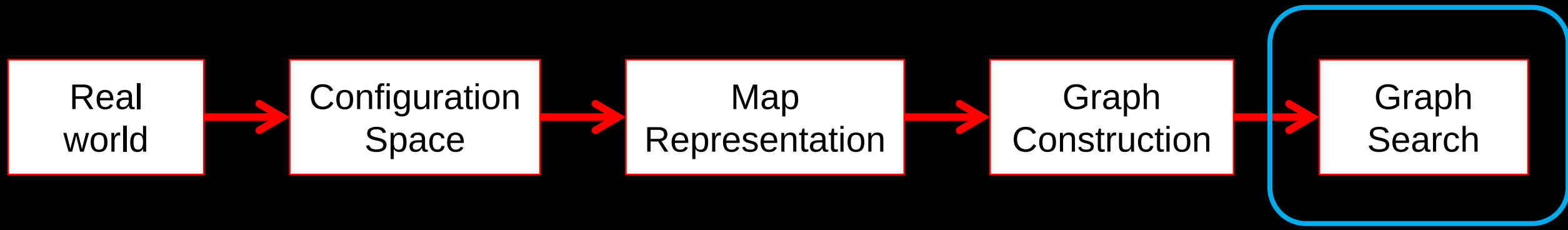
Rapidly Exploring Random Trees (RRT) - Variations

- RRT Connect
  - Two trees rooted at start and goal locations
- RRT*
  - Converges towards an optimal solution
  - Aaron Becker, UH, Wolfram Player example

- A*-RRT

- Informed RRT*, Real-Time RRT*, Theta*-RRT, etc.

Modelling path planning as a graph search problem



https://pythonrobotics.readthedocs.io/en/latest/modules/path_planning.html#basic-rrt

- Breadth first
- Depth first
- Dijstra
- A*

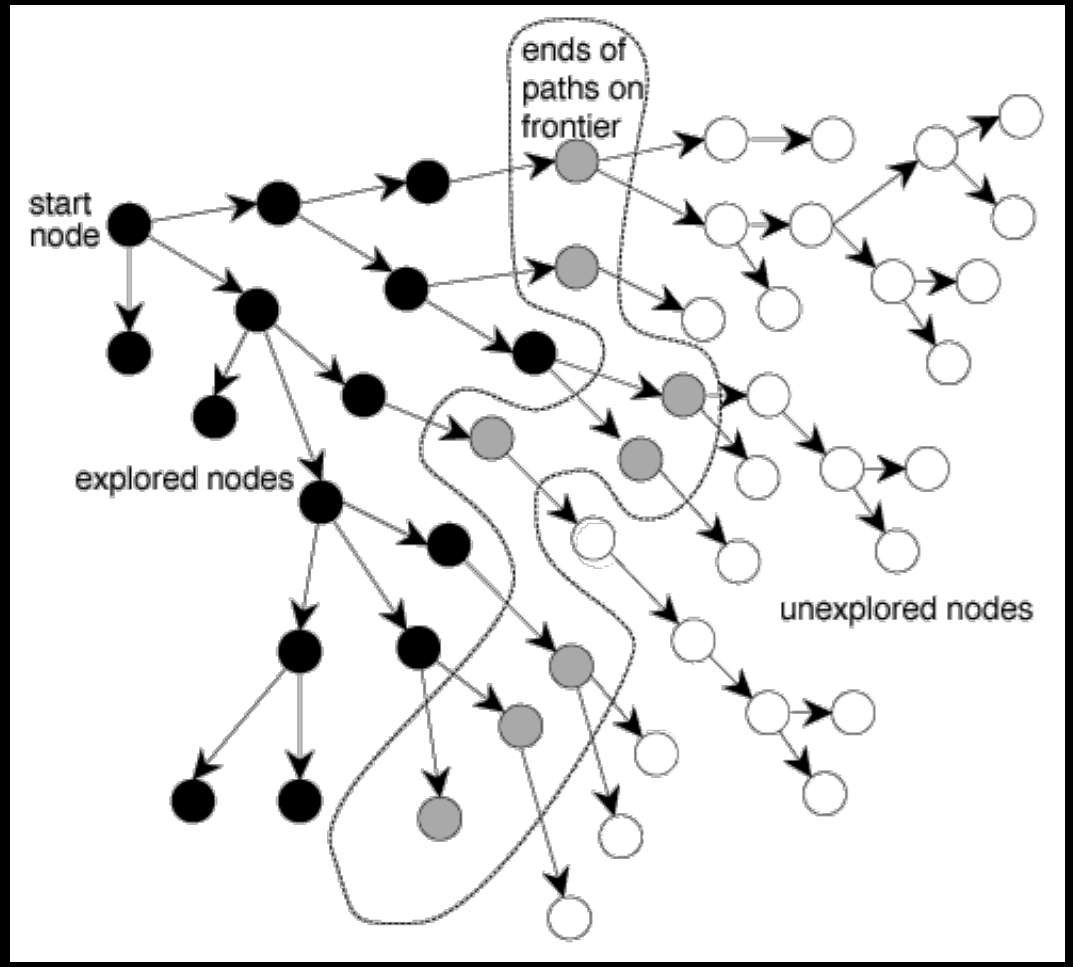
Graph Search Methods

Lecture Outline

- Graph Search Algorithms:
  - Djikstra's
  - A*

Graph Search Overview

- Solves least cost problem between two states on a (directed) graph
- Algorithms
  - Breadth-first
  - Depth-first
  - Dijkstra
  - A* and variants
  - D* and variants



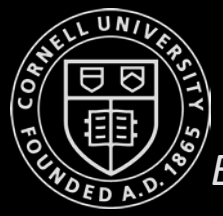
Graph Search Terms

f(n) = total cost of a node n

g(n) = the cost of the path from the start node to node n

h(n) = an estimate of the cost from node n to the goal node

c(n, n') = edge traversal cost/movement cost in going from node n to node n'



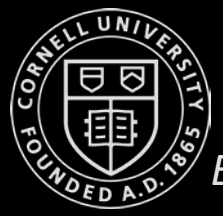
Informed vs Uninformed Search Methods

Informed Search

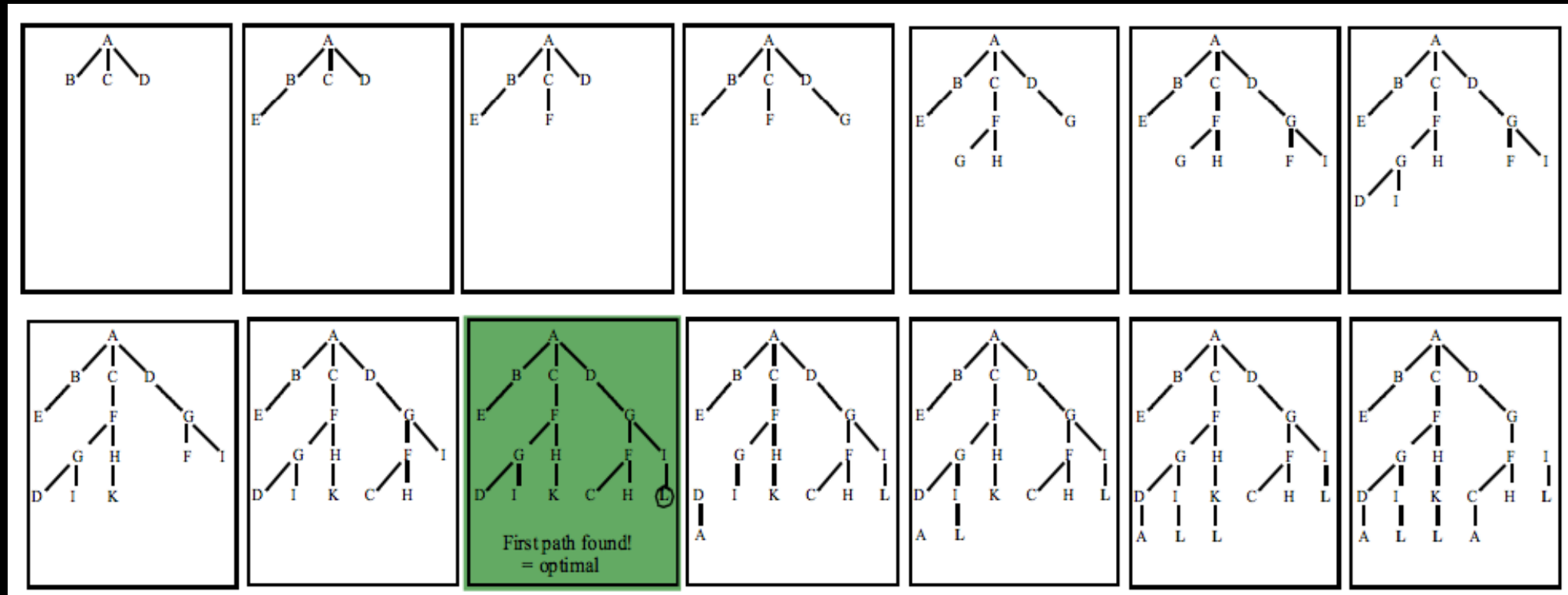
- Informed Search is a search technique that has additional information about the distance from the current state to the goal
- Finds the solution more quickly
- Not always optimal
- Ex: A*, D*, Heuristic depth-first and Heuristic breadth-first search

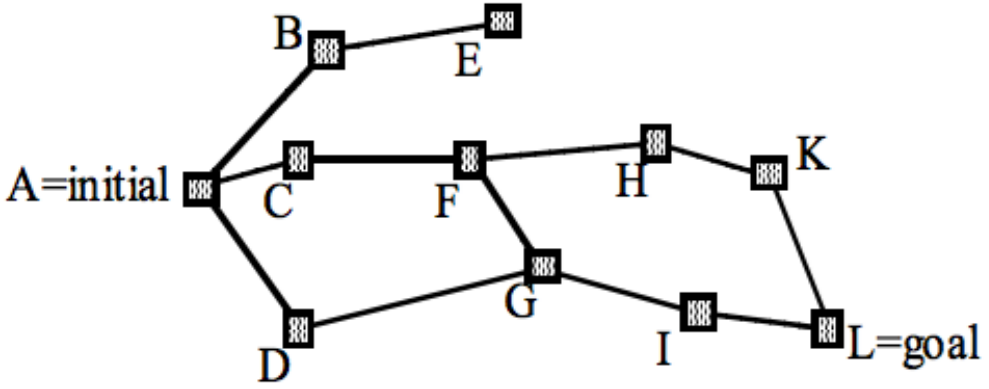
Uninformed Search

- An uninformed search is a searching technique that has no additional information about the distance from the current state to the goal
- Longer run times .
- Optimal
- Ex: Breadth-first, Depth-first, Dijkstra



Breadth First Search





Breadth First Search

- Corresponds to a “wavefront expansion” on a 2D grid
- Use a FIFO data structure (queue)
- First-found solution optimal if all edges have equal cost
- BFS Interactive Animation
- BFS Pseudocode

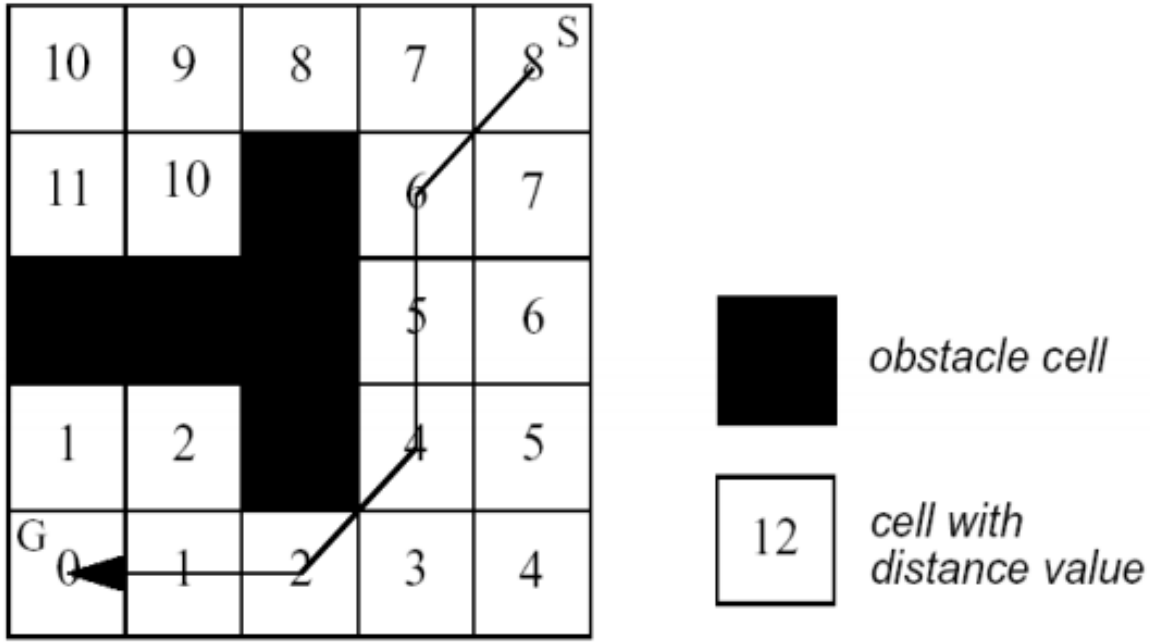
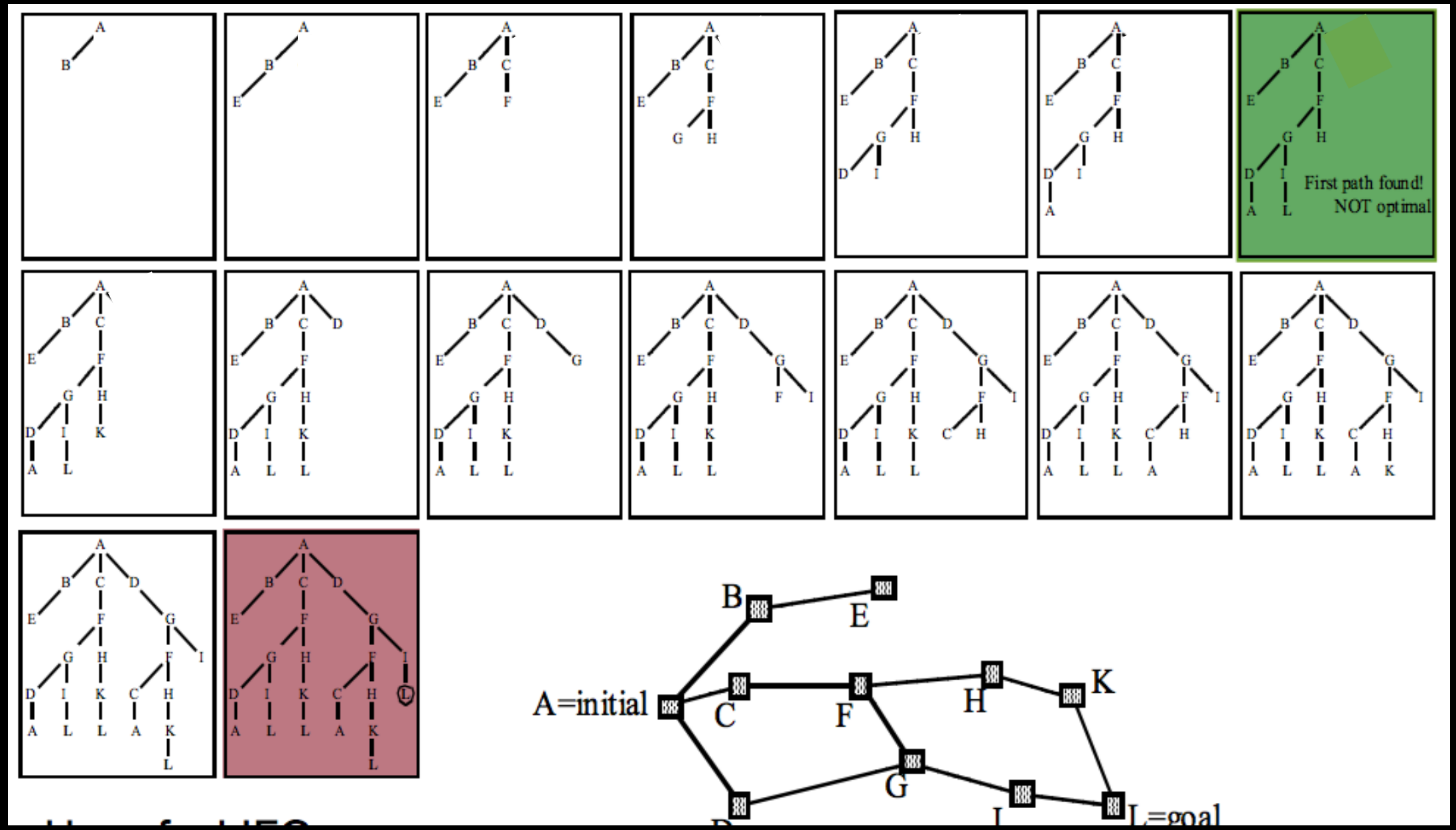


Fig. 1: NF1: put in each cell its L^1-distance from the goal position (used also in local path planning)

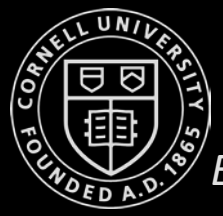
Depth First Search



Depth First Search

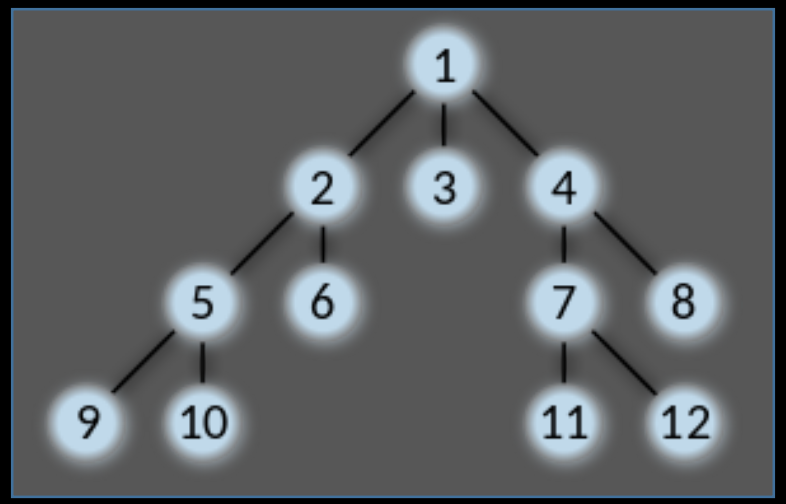
- Uses LIFO queue
- Memory efficient (can delete full sub-trees)

  DFS Pseudo Code



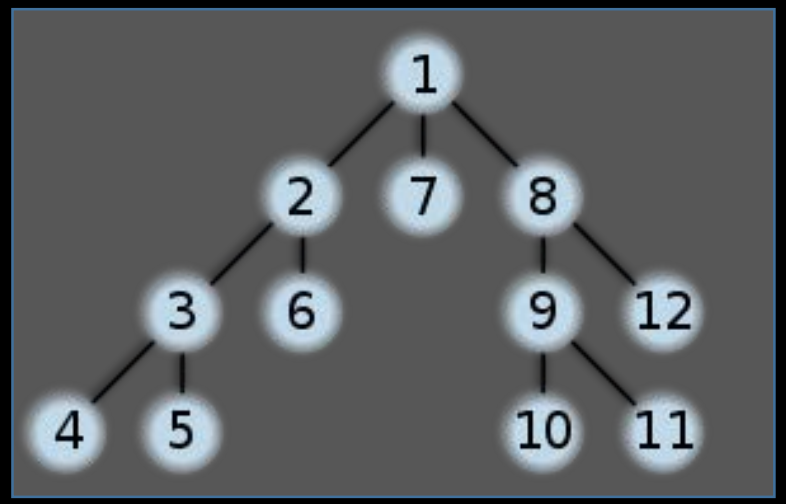
BFS

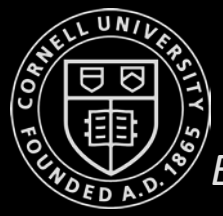
- Optimal and Complete
- Large Memory requirements
- Cannot be used for large search problems
- Implementation uses FIFO data structures



DFS

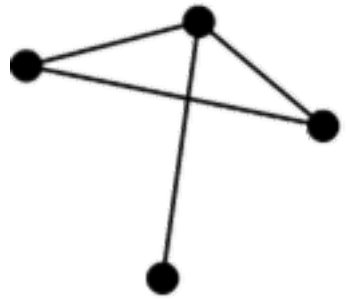
- Not optimal and not complete
- Memory efficient
- Can be used for large search problems
- Implementation uses LIFO data structures



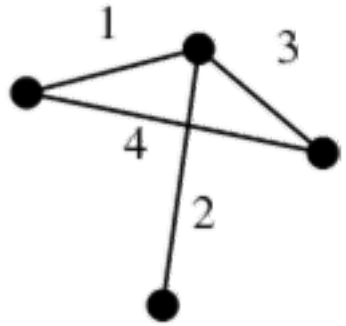

CORNELL UNIVERSITY
FOUNDED A.D. 1865

Edge Weighted Graphs

- An edge-weighted graph is a graph in which each edge is given a numerical weight
- What about search algorithms on (edge) weighted graphs?



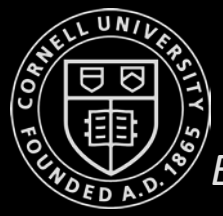
unlabeled graph



edge-labeled graph

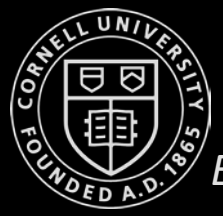
Dijkstra's Algorithm

- Works on a weighted graph
- Dijkstra search on **g(n)-sorted HEAP variation** of breadth first search



Heuristic Functions for Informed Search

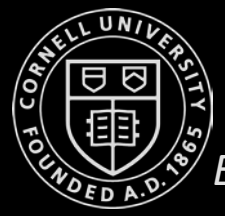
- A heuristic technique is any approach to problem solving that employs a practical method, not guaranteed to be optimal, perfect, or rational, but instead sufficient for reaching an immediate goal
- A heuristic function for path planning is **admissible** if it never overestimates the cost of reaching the goal i.e. the cost it estimates to reach the goal is not higher than the lowest possible cost from the current point in the path



Greedy best-first Search

- Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly.

$$f(n) = h(n)$$



A* Search

- Utilizes a cost function for node n

$$f(n) = g(n) + \epsilon . h(n)$$

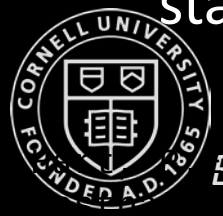
f(n) = total cost of a node

g(n) = the exact cost of the path from the start node to n

h(n) = a heuristic function that estimates the cost of the cheapest path from n to the goal

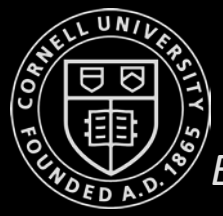
ϵ = co-efficient used to scale the heuristic function

- **Possible heuristic functions:** Hamming Distance, Manhattan Distance (L1 norm), Euclidean Distance L2 Norm
- If the heuristic function is **admissible**, A* is guaranteed to return a least-cost path from start to goal.



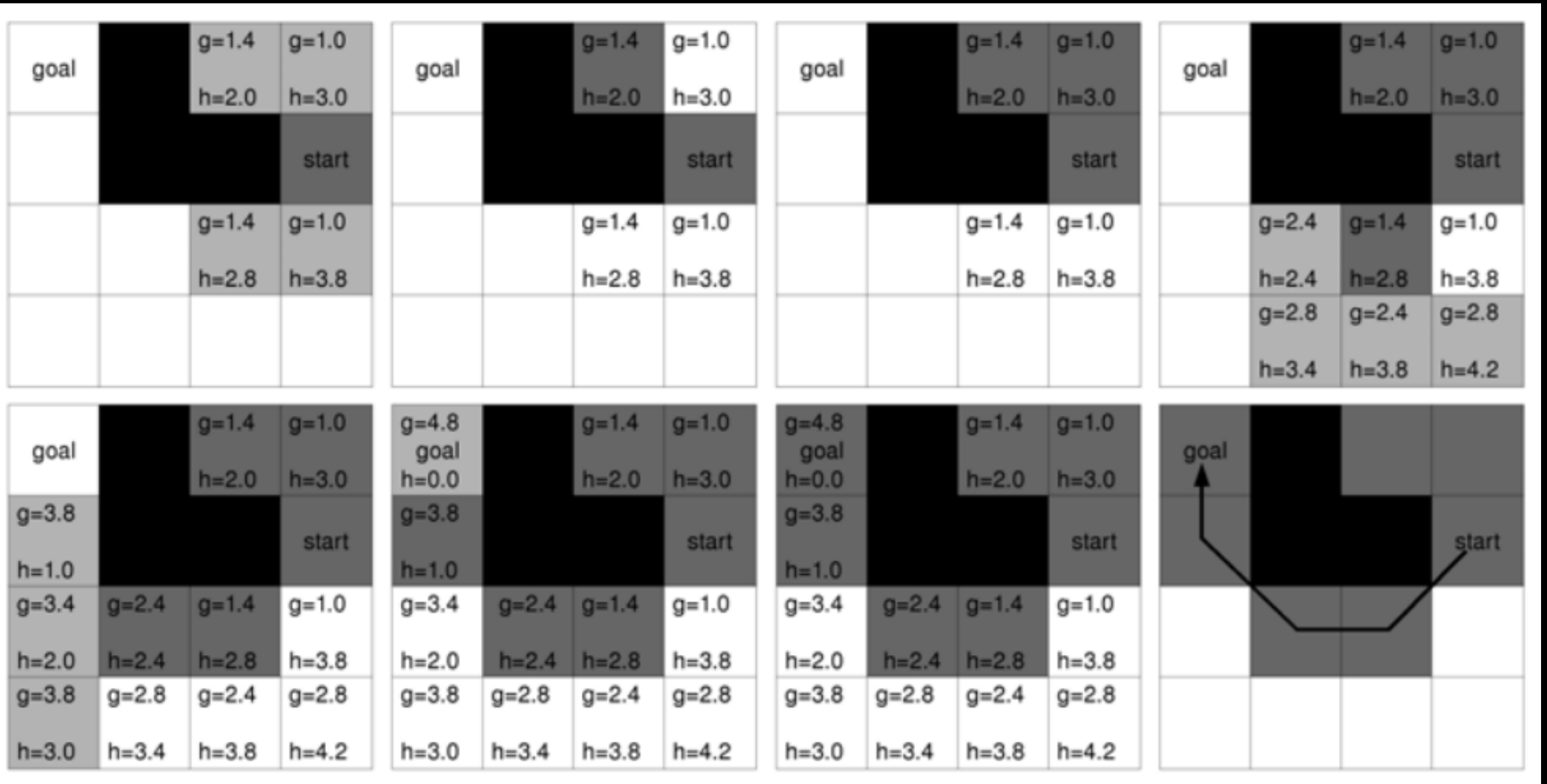
A* Search

- ϵ is a coefficient used to scale the heuristic function.
- $\epsilon = 0$ would lead to results similar to an uninformed Dijkstra's algorithm
- Very large values of ϵ would lead to results similar to the greedy best-first search algorithm
- A* Pseudo Code
- A* Interactive Comparisons

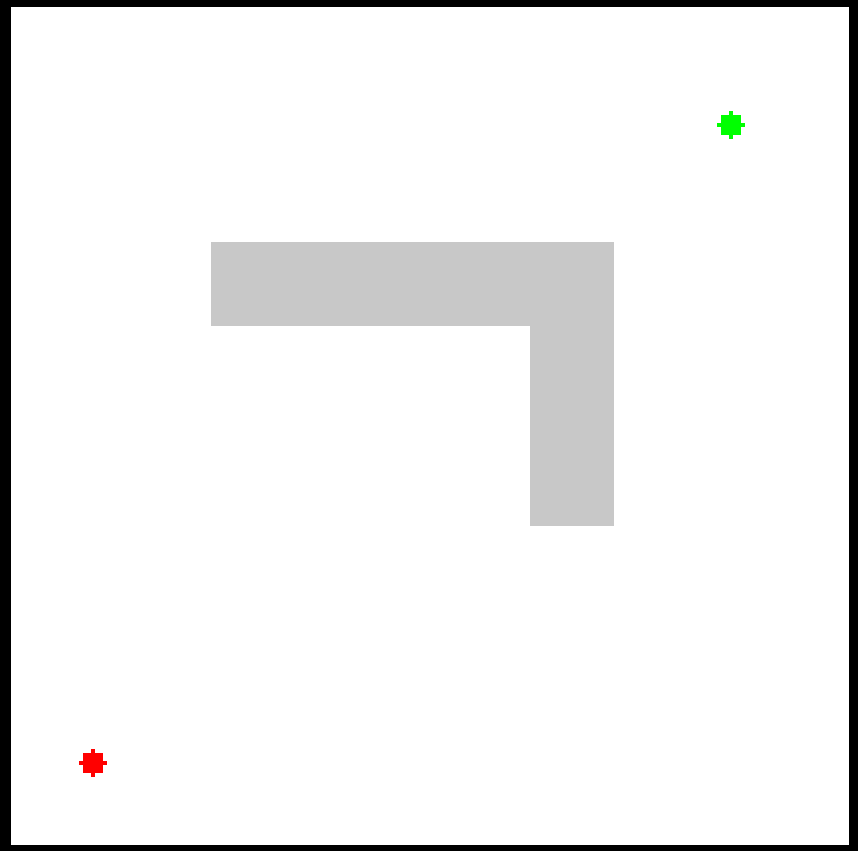


A* Search

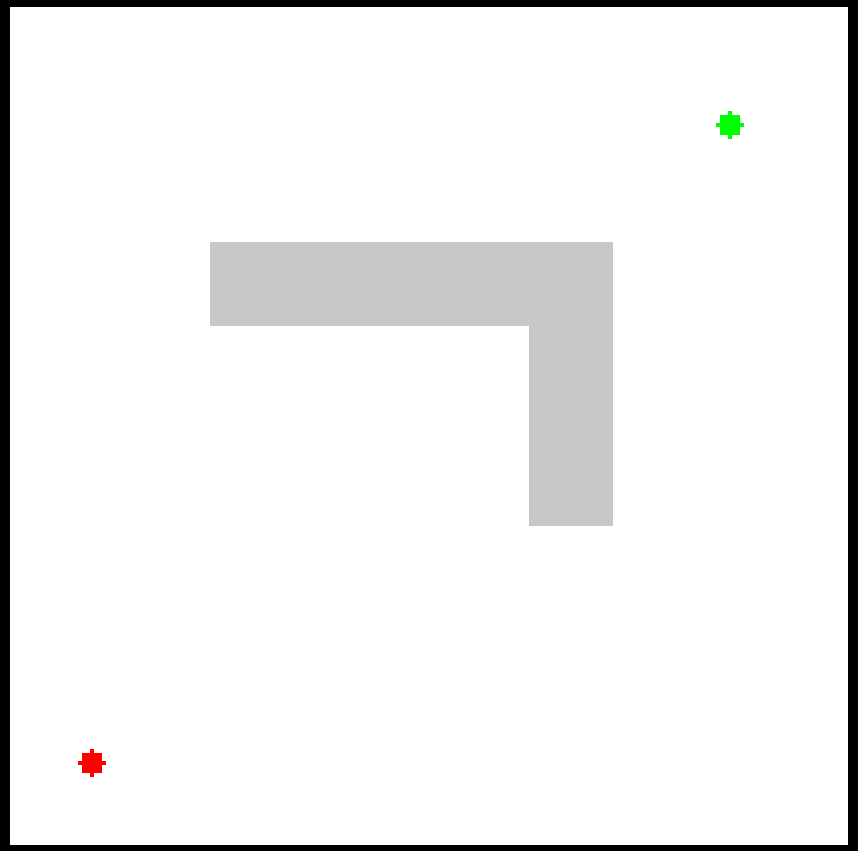
$$f(n) = g(n) + \epsilon . h(n)$$



Example where Dijkstra's is faster than A*



Dijkstra's



A*