Discrete Motion Planning

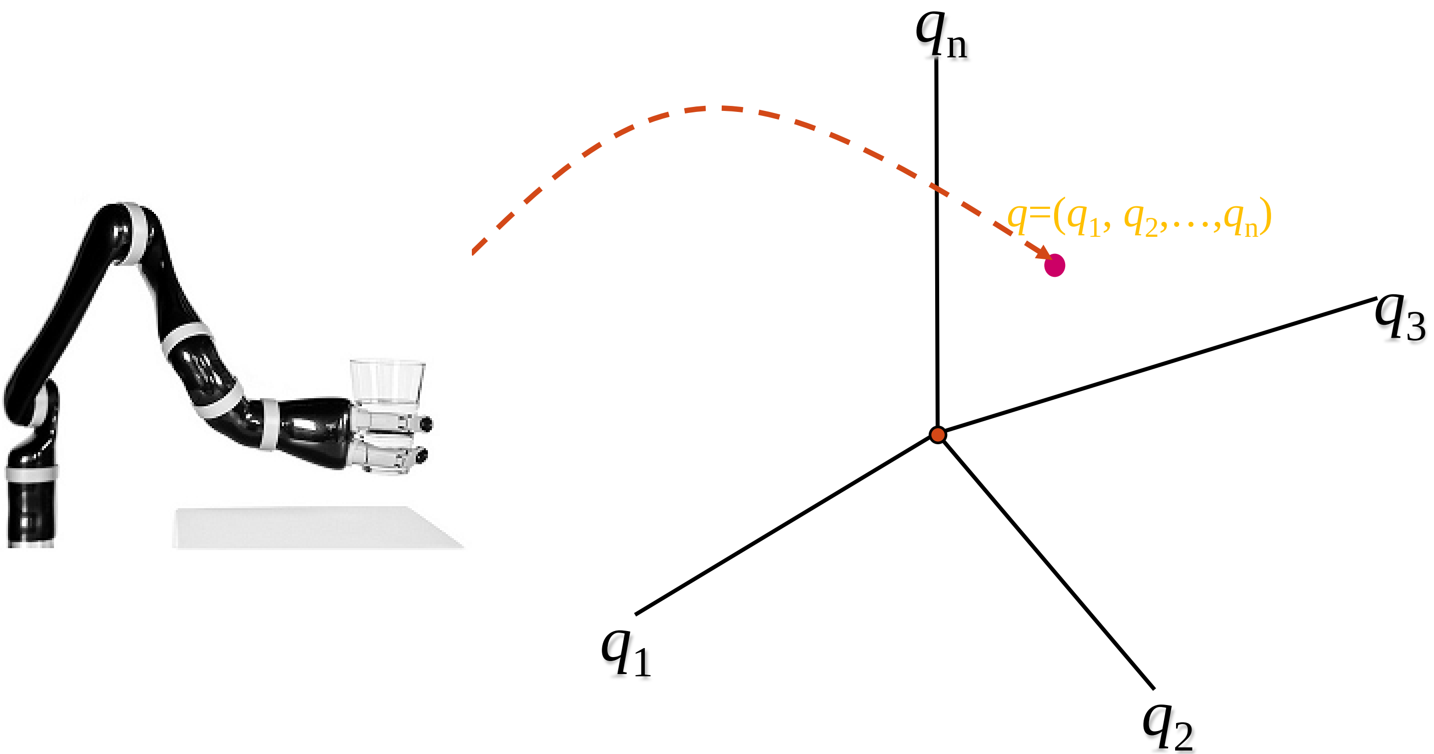
Jane Li
Assistant Professor
Mechanical Engineering & Robotics Engineering
http://users.wpi.edu/~zli11

Announcement

- Homework 1 is out
  - Due Date - **Feb 1**
  - Updated with Questions from the Textbook
- Meeting
  - Office hour VS Appointment?
  - A project description, with
    - Problem setup
    - Your proposed method for solving this problem, or
    - If you don't have any idea, I will assign some readings to inspire you

Dimension of Configuration Space

- Dimension of a Configuration Space
  - The **minimum** number of DOF needed to specify the configuration of the object completely.



Configuration Space for Articulated Objects

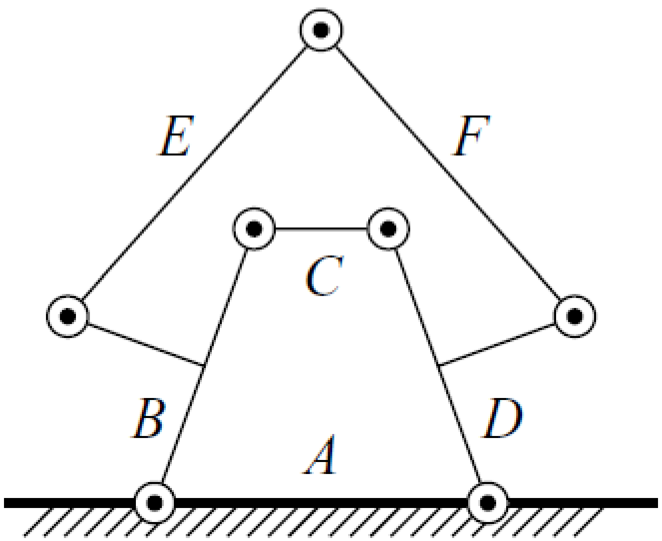
- For articulated robots (arms, humanoids, etc.), the DOF are **usually** the joints of the robot
- Exceptions? – **Parallel mechanism**
  - Closed chain mechanism with ***k*** links
  - Stationary ground link – 1
  - Movable links – ***k -1***
  - Degrees of freedom != number of joints
  - Difference between redundant robot and parallel robot

How to Compute number of DOFs?

$$M = N(k-1) - \sum_{i=1}^{n}(N - f_i) = N(k - n - 1) + \sum_{i=1}^{n} f_i$$

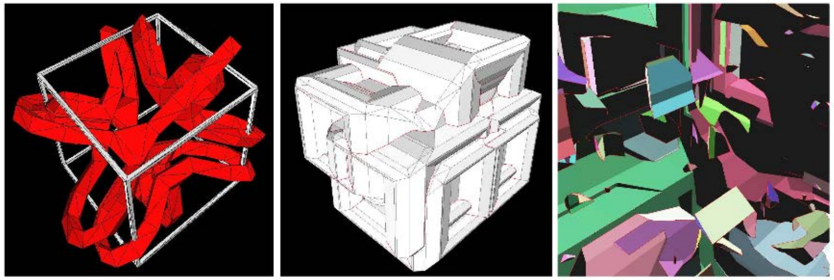
- A parallel mechanism with ***k*** links and ***n*** joints
- Each movable link has ***N*** DOFs
- Joint ***i*** has $\boldsymbol{f_i}$ degrees of freedoms

- For this example,
  - k = 6 links, n = 7 joints
  - N = 3 – planar rigid body
  - $\boldsymbol{f_i} = 1$
  - M = 3*(6 – 7 - 1) + 7 = **1**



Complexity of Minkowski Sum

- Can Minkowski Sums be computed in higher dimensions efficiently? – **Hard**



- Computational Complexity?
  - Two polytopes with ***m*** and ***n*** vertices, in ***d***-dimensional space
  - **Convex** for worst case $\Theta(mn^{\lfloor \frac{d}{2} \rfloor} + nm^{\lfloor \frac{d}{2} \rfloor})$
  - **Non-convex** $O(m^d n^d)$

Recap

- Last time
  - Configuration space – to represent the configuration of complex robot as a single point
- This class
  - **How to search for a path in C-space for a path?**

Outline

- Formulating the problem
- Search algorithms?
  - Breadth-first search
  - Depth-first search
  - Dijkstra's algorithm
  - Best-first Search
  - A* search
  - A* variants

Discrete Search

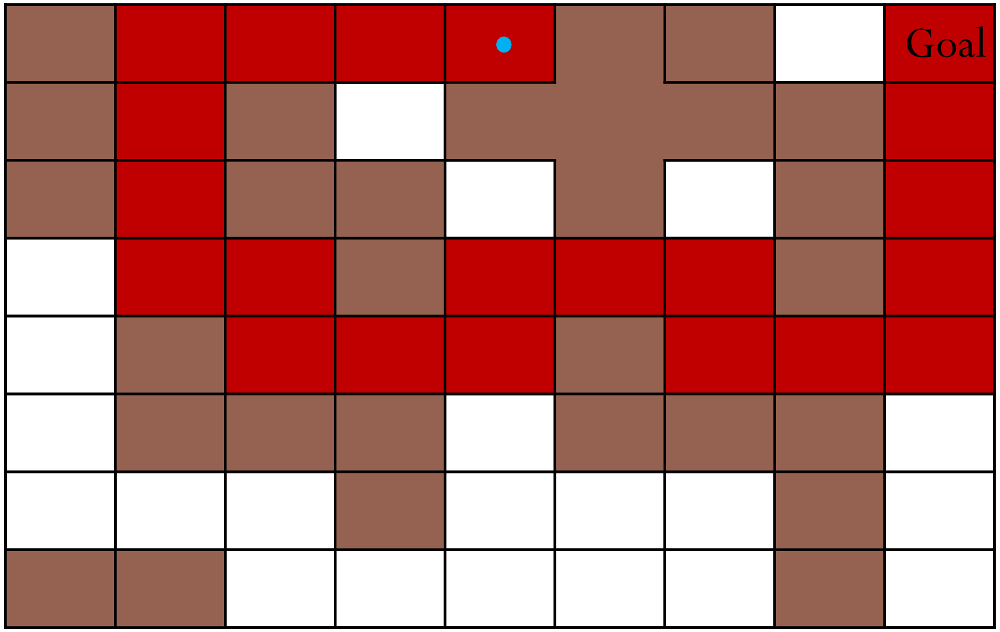
- Discrete search
  - find a ***finite*** *sequence* of discrete actions that a start state to a goal state
- Real world problems are usually continuous
  - Discretization
- CAUTION
  - Discrete search is usually very sensitive to dimensionality of state space

Problem Formulation

- What you need
  - State Space – The whole world to search in
  - Action – What action to take at a state
  - Successor – Given my current state, where to search next?
  - Action cost – The cost of performing action ***a*** at the state ***s***
  - Goal Test – The condition for termination

A classic example

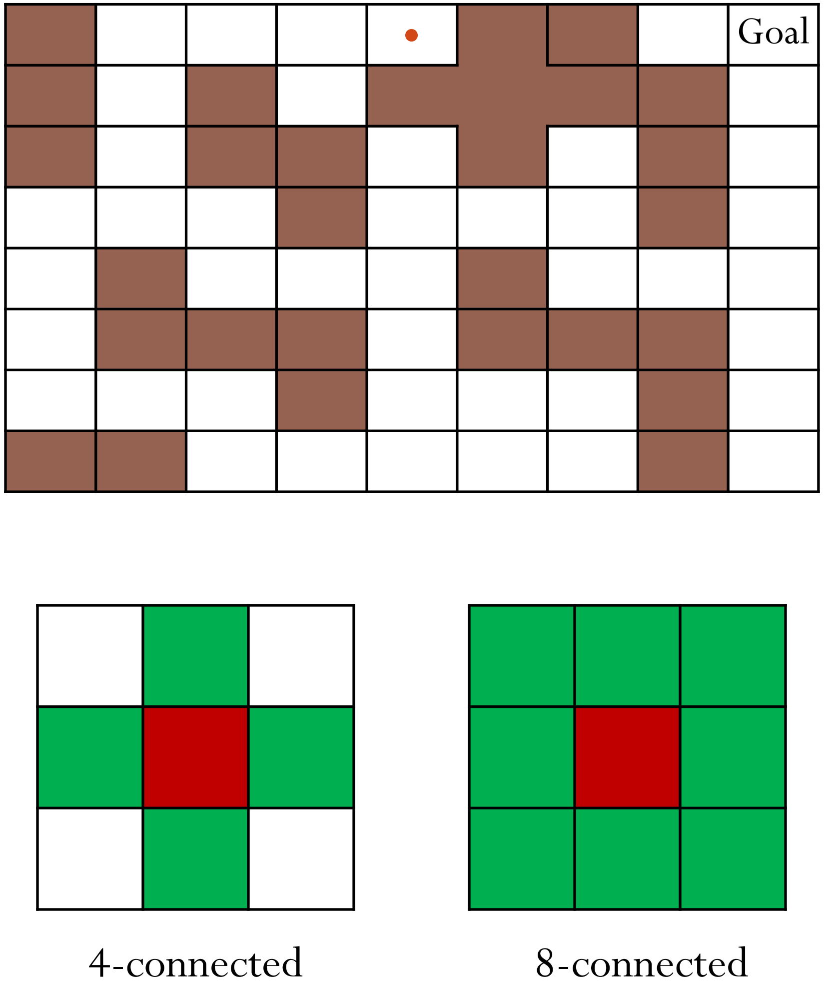
- Point robot in a maze:



- Find a sequence of free cells that goes from start to goal

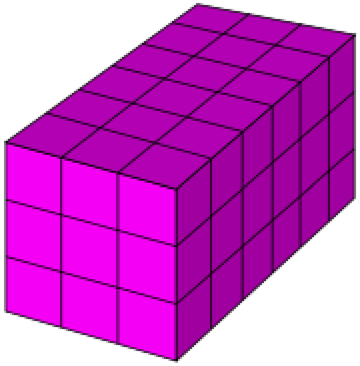
Point Robot Example

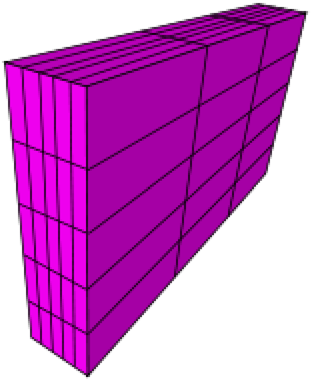
1. State Space
   - The space of cells, usually in x,y coordinates
2. Successor Function
   - A cell's successors are its neighbors
   - 4 connected vs. 8 connected
3. Actions
   - Move to a neighboring cell
4. Action Cost
   - Distance between cells traversed
   - Are costs the same for 4 vs 8 connected?
5. Goal Test
   - Check if at goal cell
   - Multiple cells can be marked as goals

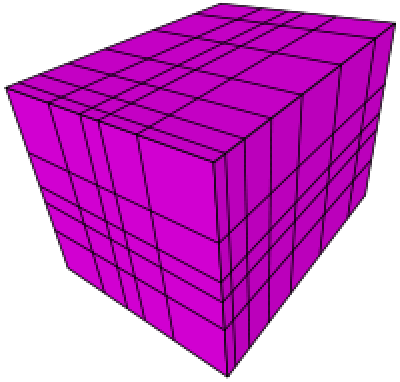


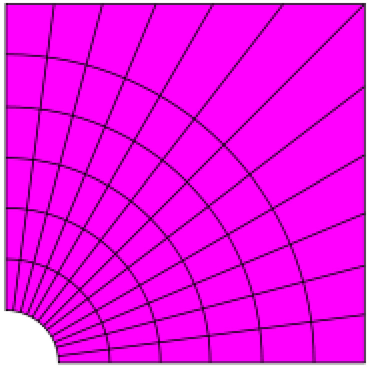
State space

- For motion planning, state space is usually a **grid**
- There are many kinds of grids!


Cartesian Grid


Regular Grid

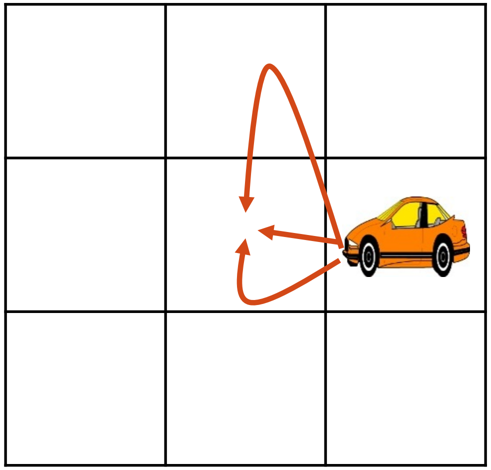

Rectilinear Grid


Curvilinear Grid

- Note that
  - The choice of grid (i.e. state space) is crucial to performance and accuracy
  - The world is really continuous; these are all approximations

Actions

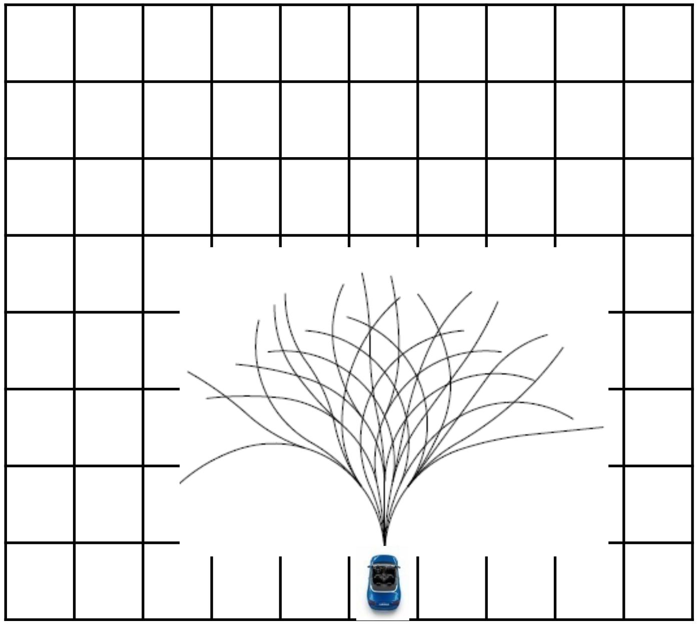
- Actions in motion planning are also often **continuous**
- There are **many** ways to move between neighboring cells
- Usually pick a discrete action set ***a priori***
- What are the **tradeoffs** in picking action sets? - A major issue in in **non-holonomic motion** planning



from Knepper and Mason, ICRA 2009

Successors

- These are largely determined by the action set
- Successors may not be known a priori
  - You have to try each action in your action set to see which cell you end in



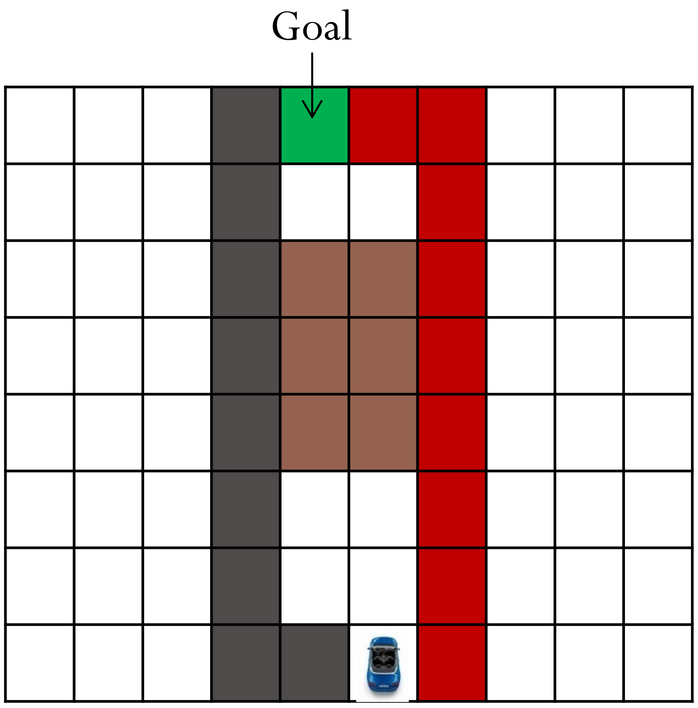
Action Cost

- Depends on what you're trying to optimize
  - Minimum Path Length: Cost is distance traversed when executing action
  - What is action cost for path smoothness?
- Sometimes we consider more than one criterion
  - Linear combination of cost functions (most common):

$$\text{Cost} = a_1C_1 + a_2C_2 + a_3C_3 \ldots.$$

Goal Test

- Goals are most commonly specific cells you want to get to
- But they can be more abstract, too!
- Example Goals:
  - A state where X is visible
  - A state where the robot is contacting X
  - Topological goals



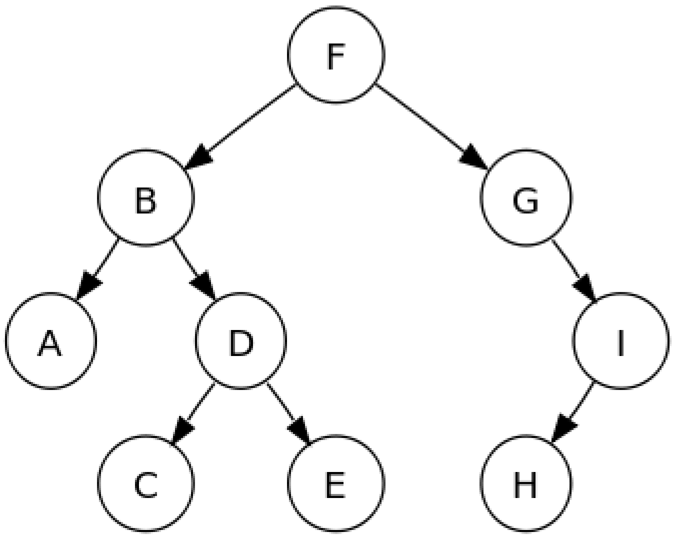
A topological goal could require the robot to go **right** around the obstacle (need whole path to evaluate if goal reached)

Tree Search Algorithms

```
function Tree-Search(problem, strategy)
    Root of search tree <- Initial state of the problem
    While 1
        If no nodes to expand
                return failure
        Choose a node n to expand according to strategy
        If n is a goal state
                return solution path  //back-track from goal to
                                      //start in the tree to get path
        Else
                NewNodes <- expand n
                Add NewNodes as children of n in the search tree
```

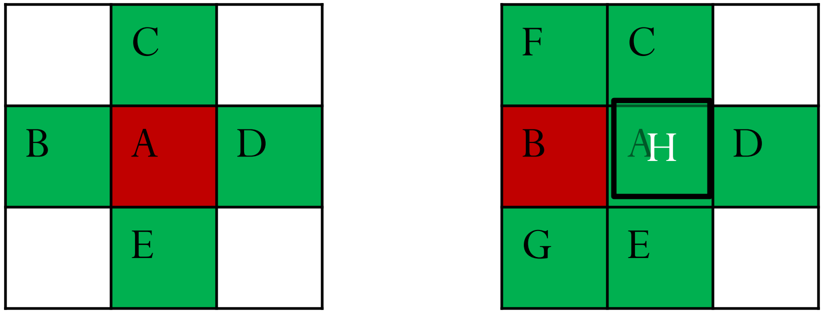
Tree Search Algorithms

- All you can choose is the **strategy**, i.e. which node to expand next
- Strategy choice affects
  - **Completeness** – Does the algorithm find a solution if one exists?
  - **Optimality** – Does it find the least-cost path?
  - **Run Time**
  - **Memory usage**
- Run time and memory usage are affected by
  - **Branching Factor** – how many successors a node has
  - **Solution Depth** – How many levels down the solution is
  - **Space Depth** – Maximum depth of the space



Tree Search Algorithms

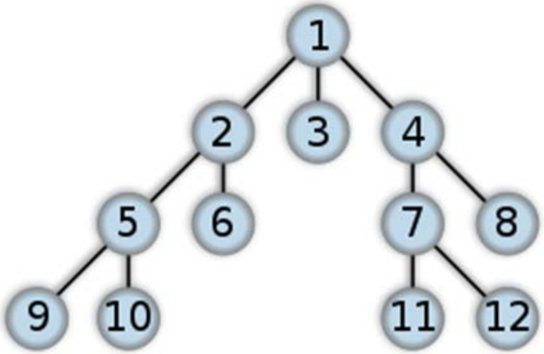
- Need to **avoid re-expanding** the same state



- Solution: An *open list* to track which nodes are unexpanded
  - E.g., a queue (First-in-first-out)

Breadth-first Search (BFS)

- Main idea
  - Build search tree in layers
- Open list is a **queue**,
  - Insert new nodes at the **back**
- Result:
  - "Oldest" nodes are expanded first
- BFS finds the **shortest** path to the goal

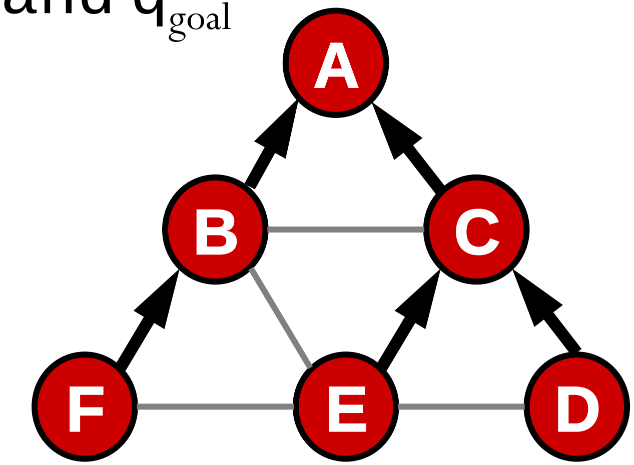


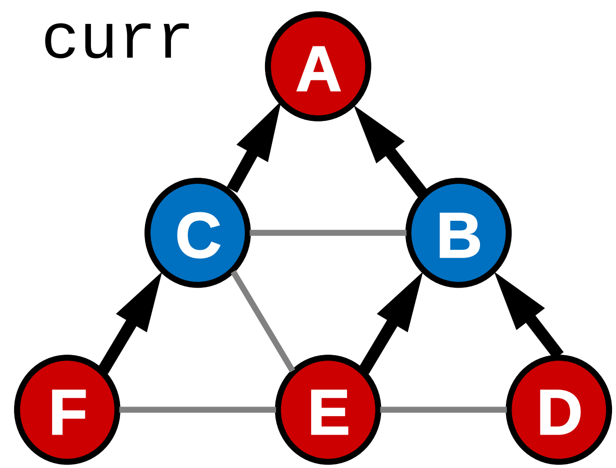
Breadth-first search

Stack

Input: q_{init}, q_{goal}, visibility graph G
Output: a path between q_{init} and q_{goal}

```
Q = new queue;
Q.enqueue(q_init);
mark q_init as visited;
while Q is not empty
   curr = Q.dequeue();
   if curr == q_goal then
      return curr;
   for each w adjacent to curr
      if w is not visited
         w.parent = curr;
          Q.enqueue(w)
          mark w as visited
```



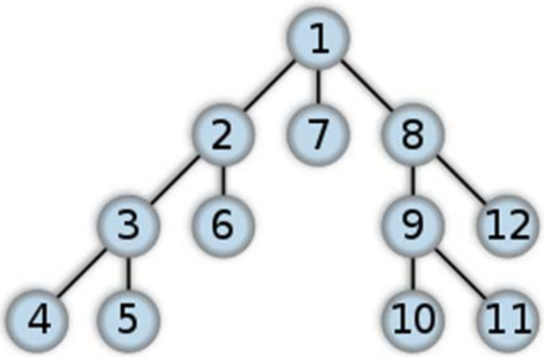


Search Order?

Queue

Depth-first Search (DFS)

- Main idea
  - Go as deep as possible as fast as possible
- Open list is a **stack**,
  - Insert new nodes at the **front**
- Result
  - "Newest" nodes are expanded first
- DFS does **NOT** necessarily find the **shortest** path to the goal

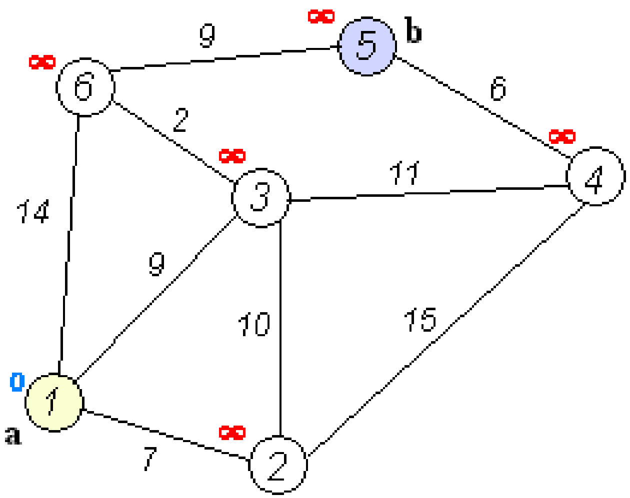


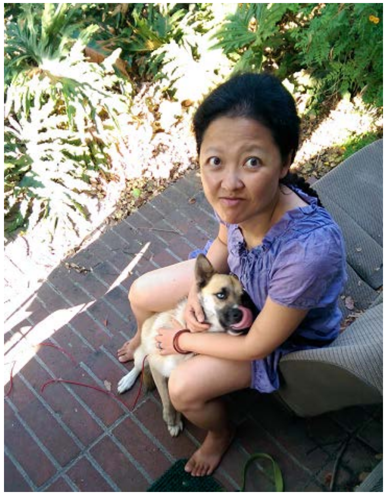
Efficiency

- BFS v.s. DFS - which is better?
  - Depending on the data and what you are looking for, either DFS or BFS could be advantageous.
- When would **BFS** be very inefficient?
- When would **DFS** be very inefficient?

Dijkstra's algorithm

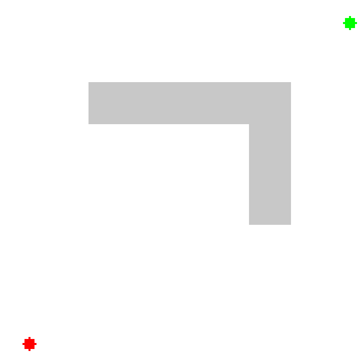
- Main Idea
  - Like BFS but edges can have **different costs**
- Open list is a ***priority queue***,
  - Nodes are sorted according to g(x), where g(x) is the minimum current cost-to-come to x
- g(x) for each node is updated during the search
  - keep a list lowest current cost-to-come to all nodes)





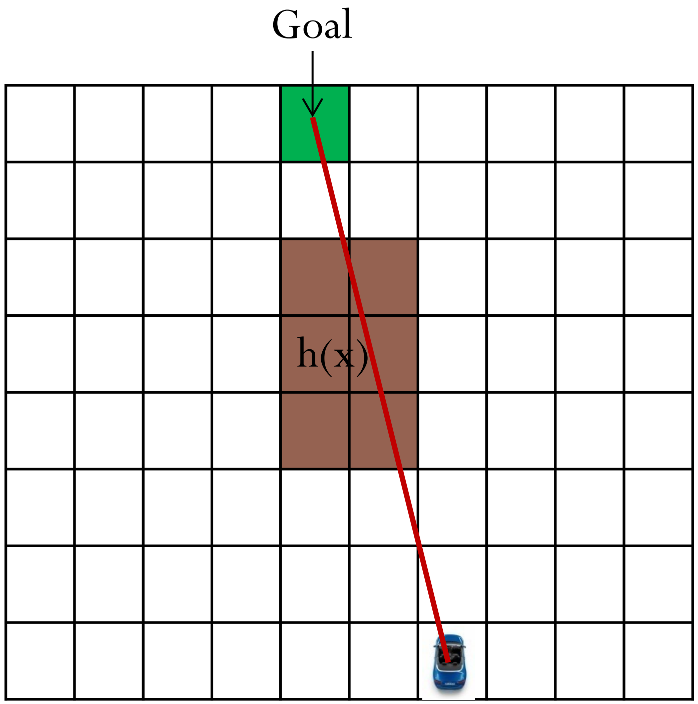
Dijkstra's algorithm

- Result
  - Will find the least-cost path to **all** nodes from a given start
- If planning in a **cartesian grid** and cost is **distance** between grid cell centers, is Dijksta's the same as BSF for
  - 4-connected space?
  - 8-connected space?



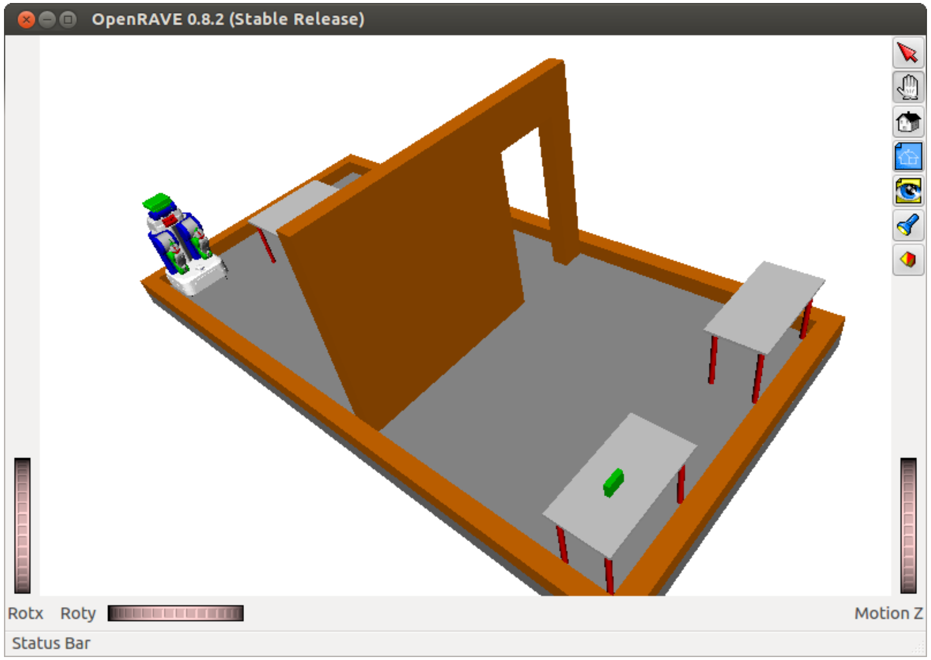
Best-first Search

- Main idea
  - Use Heuristic function h(x) to estimate each node's distance to goal, expand node with minimum h(x)
- Open list is a ***priority queue***,
  - Nodes are sorted according to h(x)



Best-first Search

- Result
  - Works great if **heuristic is a good** estimate
- Does **not** necessarily find least-cost path
- When would this strategy be inefficient?

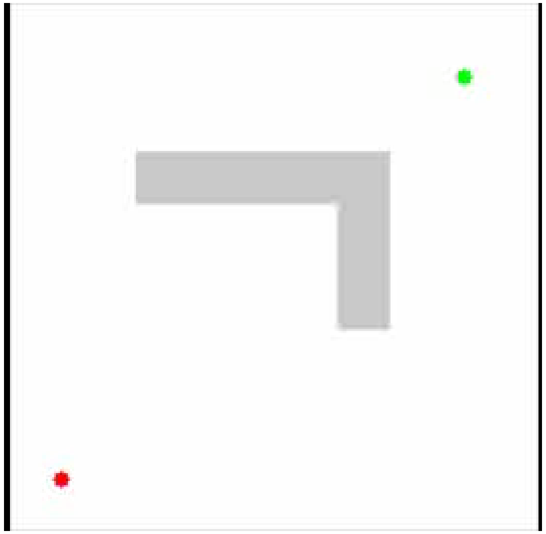


A* Search

- Main idea:
  - Select nodes based on cost-to-come and heuristic:

    $$f(x) = g(x) + h(x)$$

- Open list is a *priority queue*,
  - Nodes are sorted according to f(x)
- g(x) is sum of edge costs from root node to x



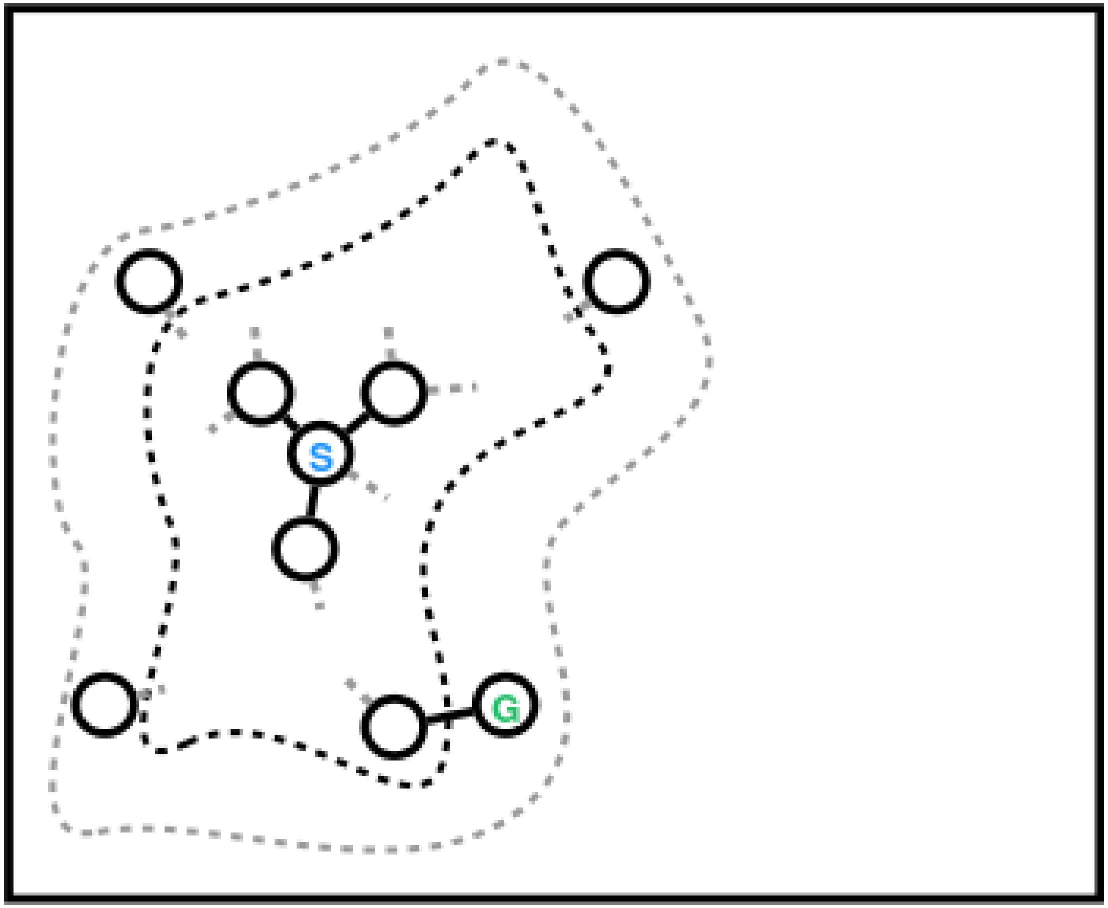
A* Search

- IMPORTANT RESULT:
  - If h(x) is *admissible*, A* will find the least-cost path!
- Admissibility:
  - h(x) must *never overestimate* the true cost to reach the goal from x
    - $h(x) < h^*(x)$, where $h^*(x)$ is the true cost
    - $h(x) > 0$ (so $h(G) = 0$ for goals G)
  - "Inflating" the heuristic may give you faster search, but least-cost path is not guaranteed

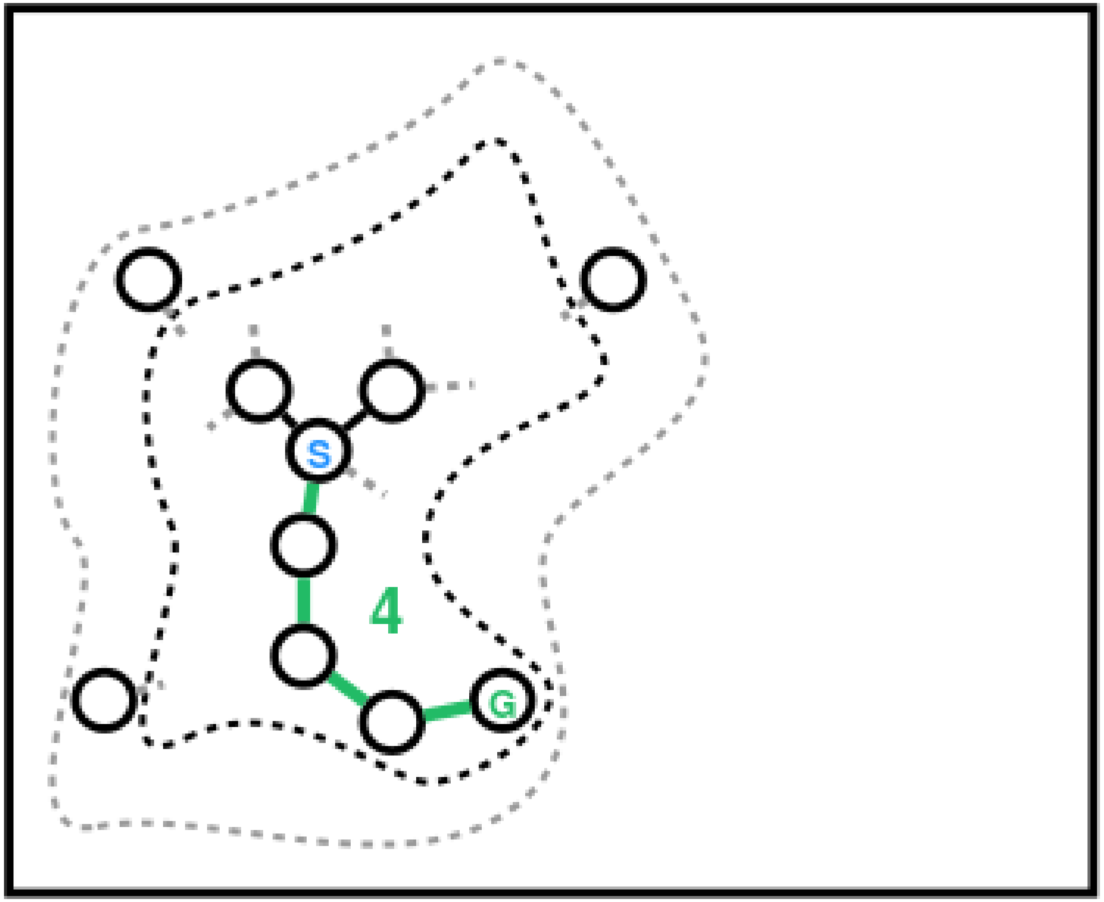
A* Optimality Proof

- Proof by contradiction
- Assumption
  - Heuristic is admissible
  - The path found by A* is sub-optimal

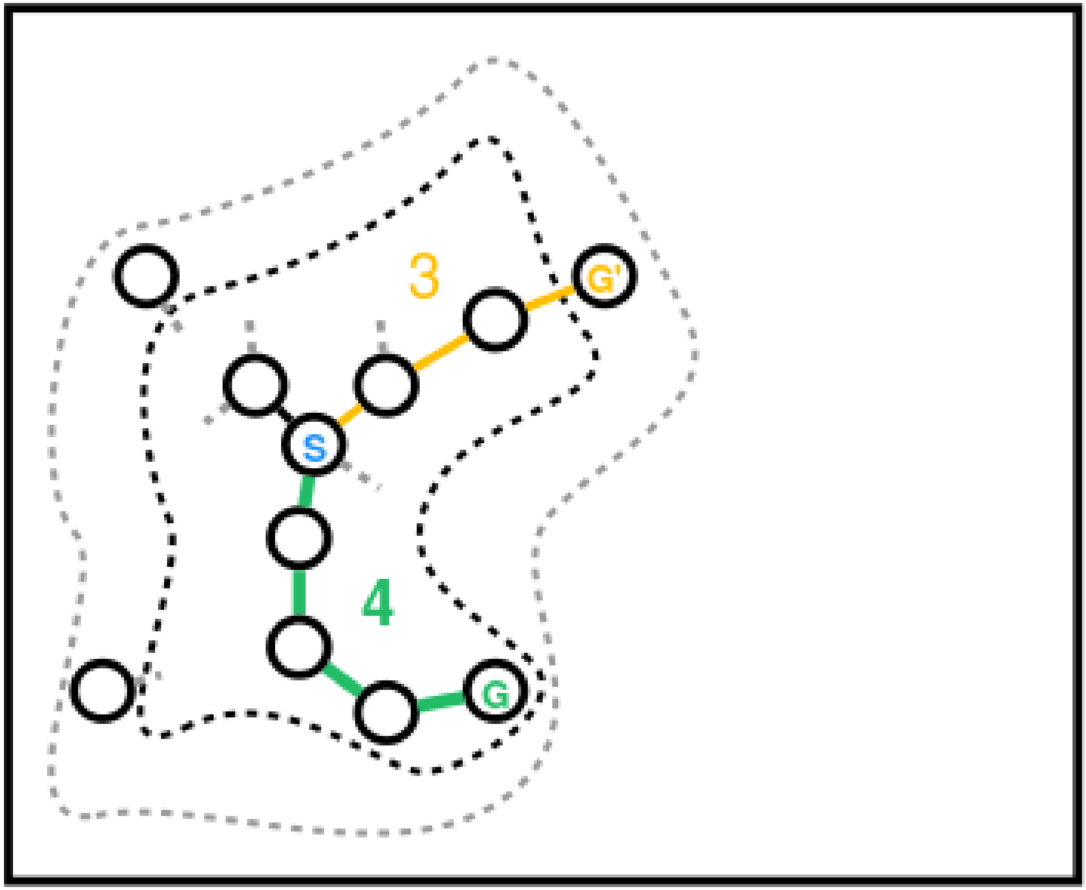
A* Optimality Proof



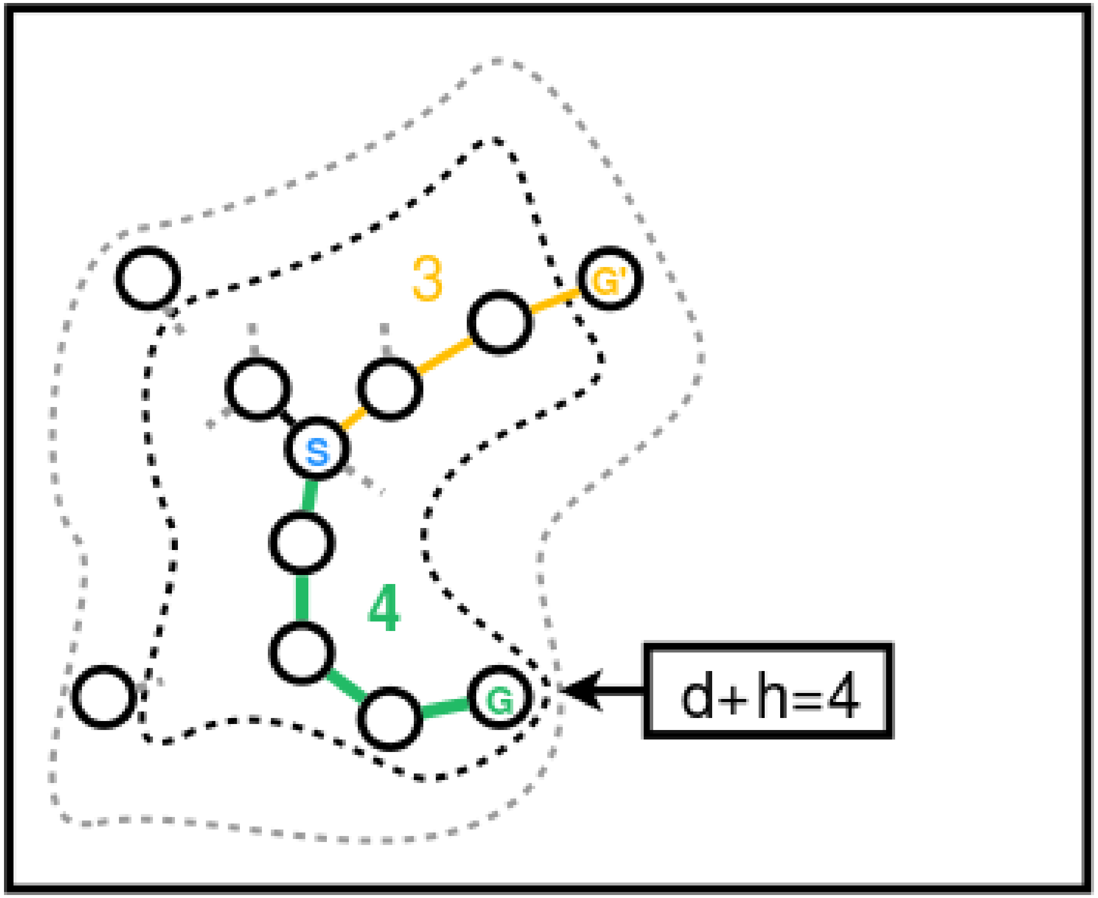
A* Optimality Proof



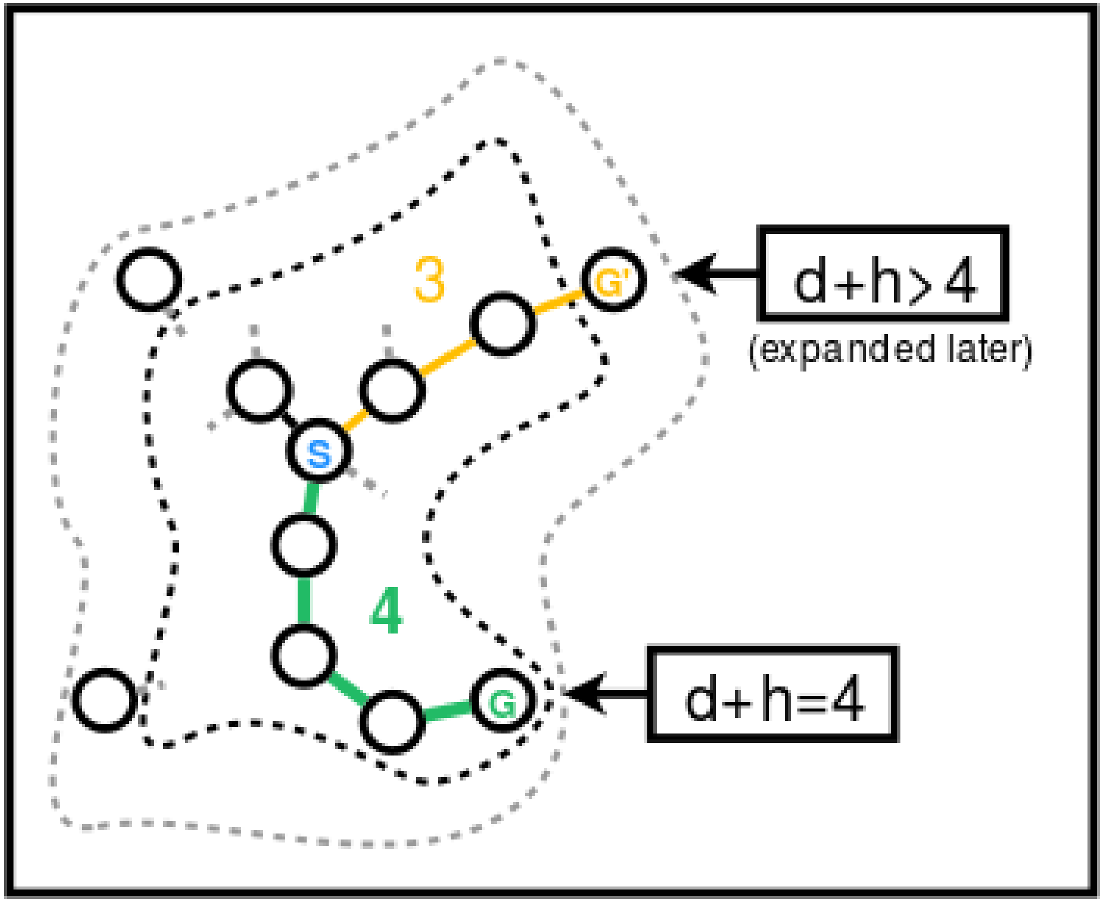
A* Optimality Proof



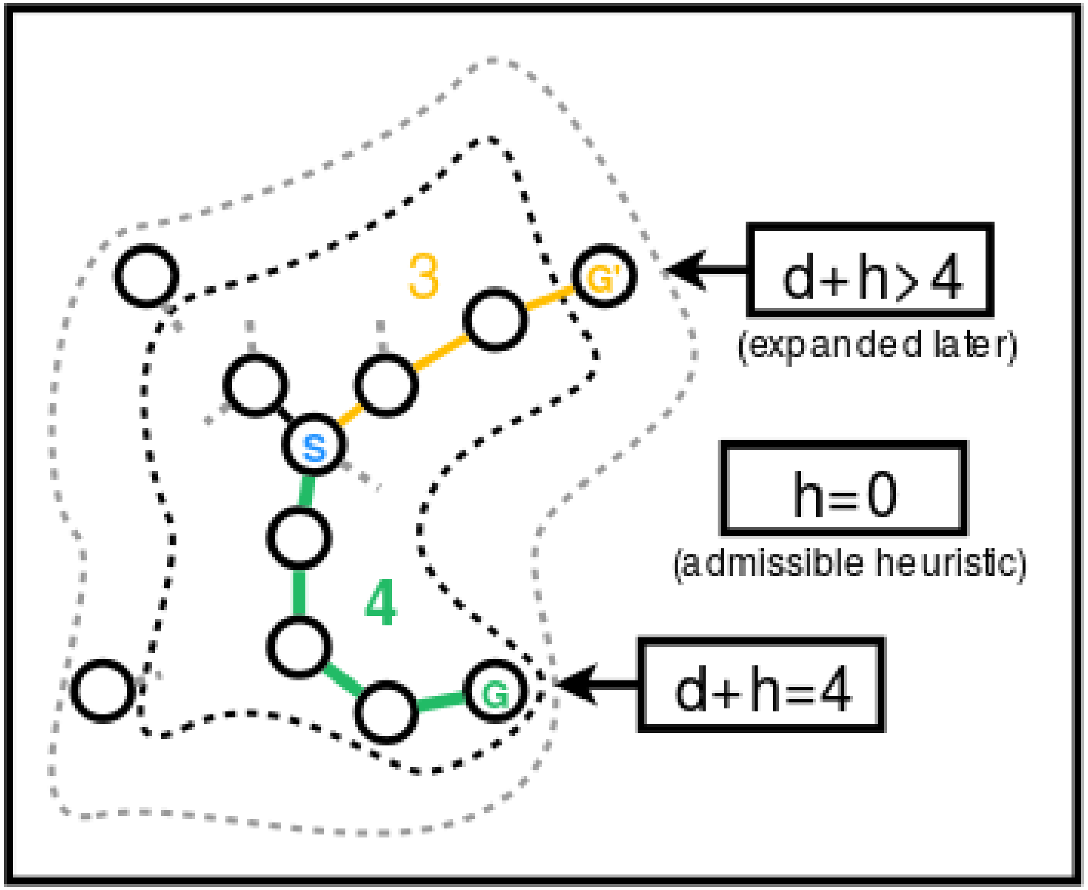
A* Optimality Proof



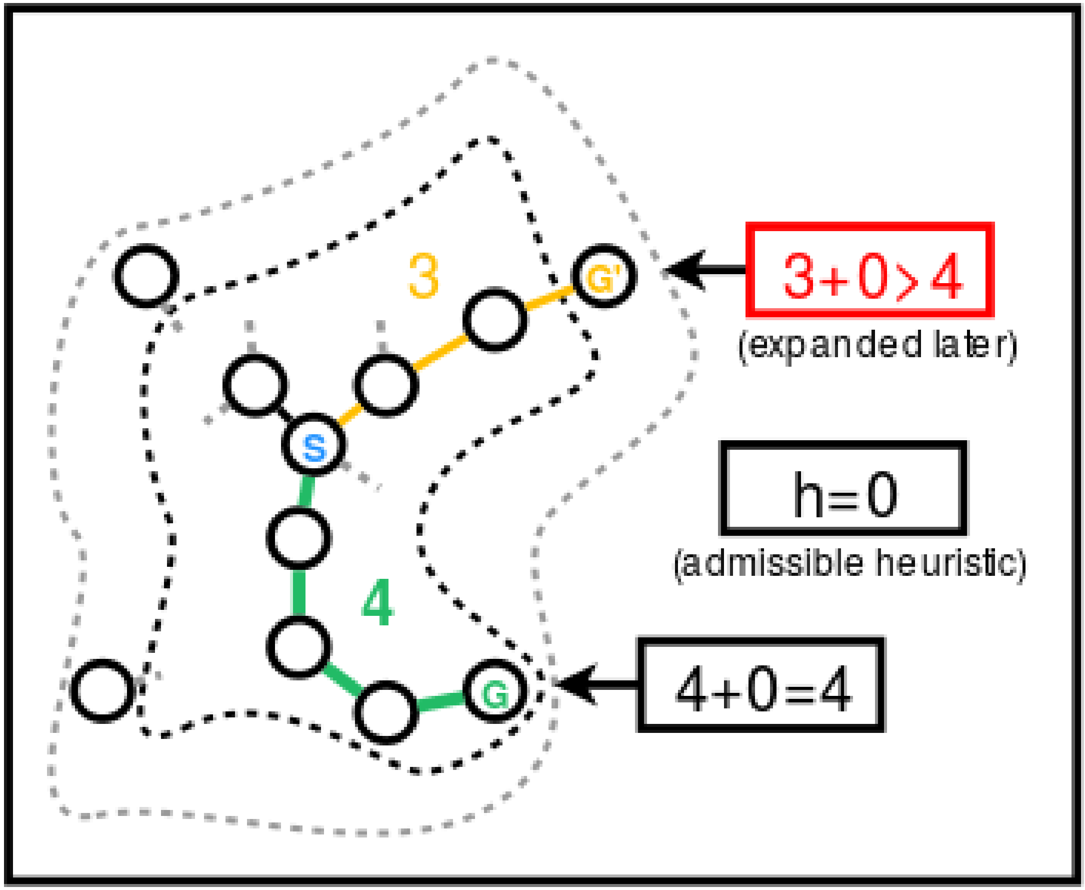
A* Optimality Proof



A* Optimality Proof



A* Optimality Proof



Questions

- If you set $h(x) = 0$ for all x, A* is equivalent to which search strategy?
- If you set $g(x) = 0$ for all x, and $h(x)$ = depth of x, A* is equivalent to which search strategy?
- In the worst case, what percentage of nodes will A* explore?

Variants of A*

- There are many variants of A*, some of the most popular for motion planning are:
  - Anytime Repairing A* (ARA*)
  - Anytime Non-parameteric A* (ANA*)
- Student presentation - Feb 15

Readings

- Principles CH 3.5-3.6, Appendix E
- Homework 1 is out
  - Need help with Installation of Openrave