Motion Planning, Part III
Graph Search, Part I

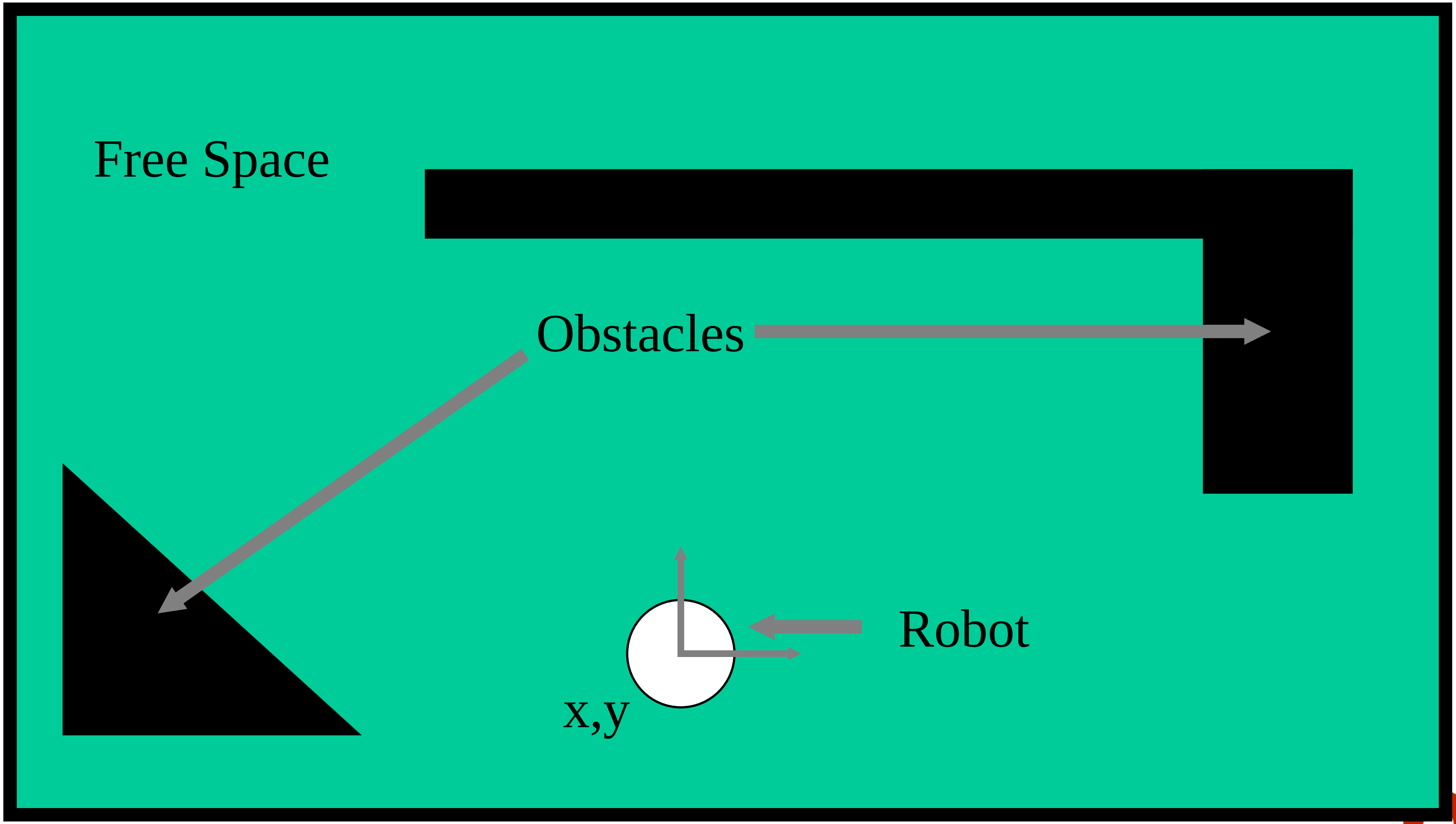
Howie Choset

Happy President's Day

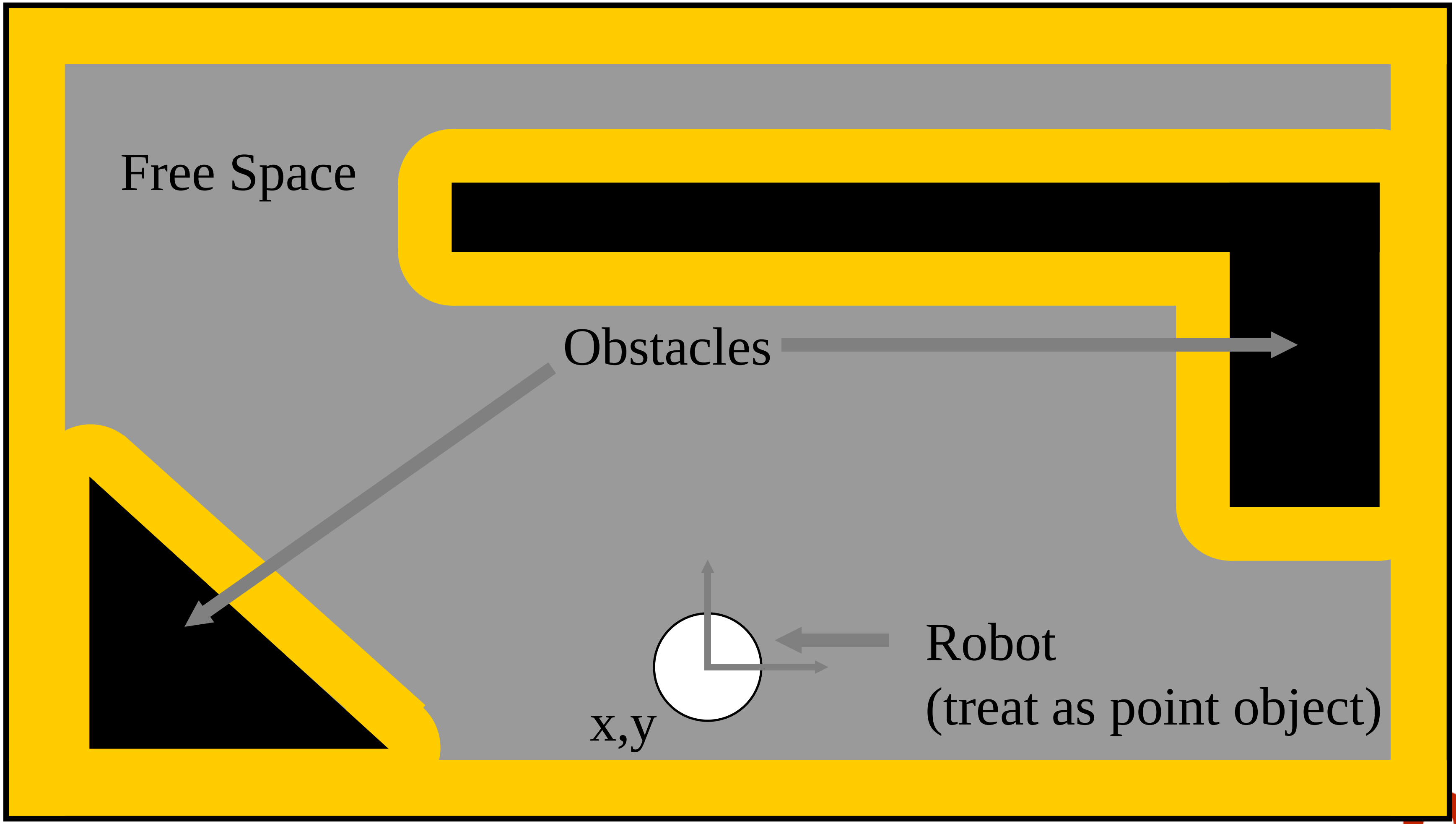
The Configuration Space

- What it is
  - A set of "reachable" areas constructed from knowledge of both the robot and the world
- How to create it
  - First abstract the robot as a point object. Then, enlarge the obstacles to account for the robot's footprint and degrees of freedom
  - In our example, the robot was circular, so we simply enlarged our obstacles by the robot's radius (*note the curved vertices*)

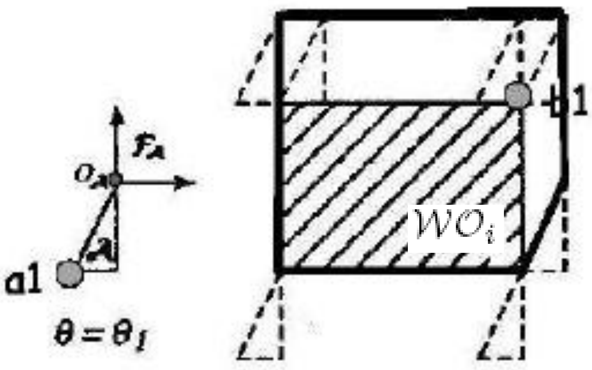
Example of a World (and Robot)

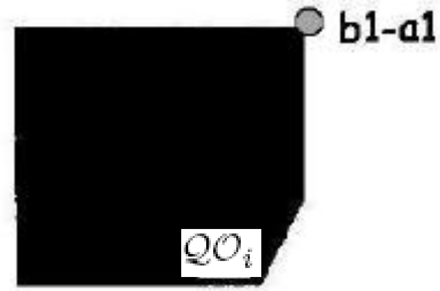


Configuration Space: Accommodate Robot Size



Translate-only, non-circularly

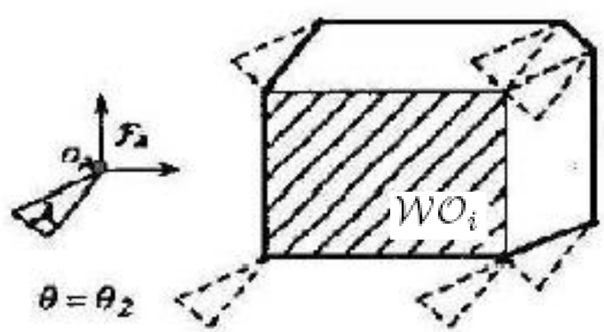


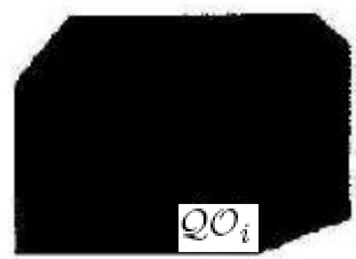


$$\mathcal{QO}_i = \{q \in \mathcal{Q} \mid R(q) \bigcap \mathcal{WO}_i \neq \emptyset\}.$$

Pick a reference point…

Translate-only, non-circularly symmetric

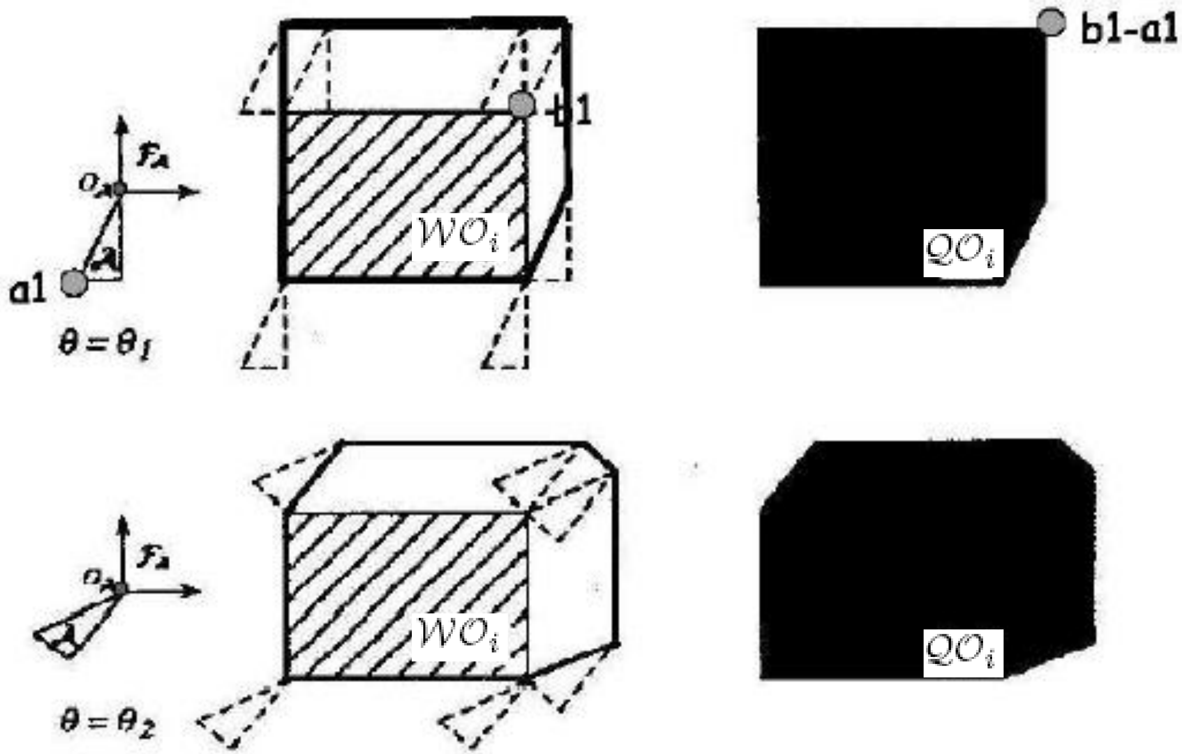




$$\mathcal{QO}_i = \{q \in \mathcal{Q} \mid R(q) \bigcap \mathcal{WO}_i \neq \emptyset\}.$$

Pick a reference point…

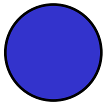
With Rotation: how much distance to rotate

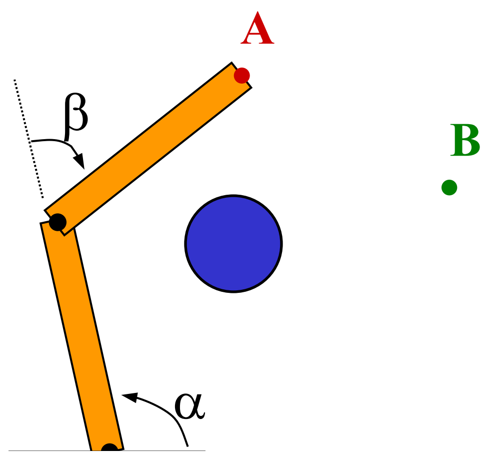


$$\mathcal{QO}_i = \{q \in \mathcal{Q} \mid R(q) \bigcap \mathcal{WO}_i \neq \emptyset\}.$$

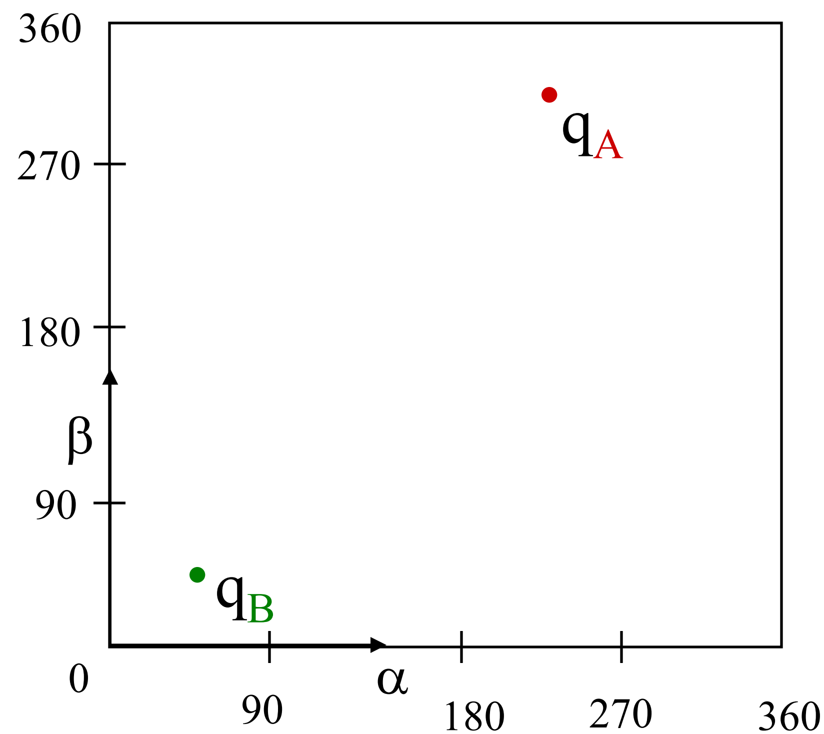
Pick a reference point…

Configuration Space “Quiz”

Where do we put  ?



An obstacle in the robot’s workspace

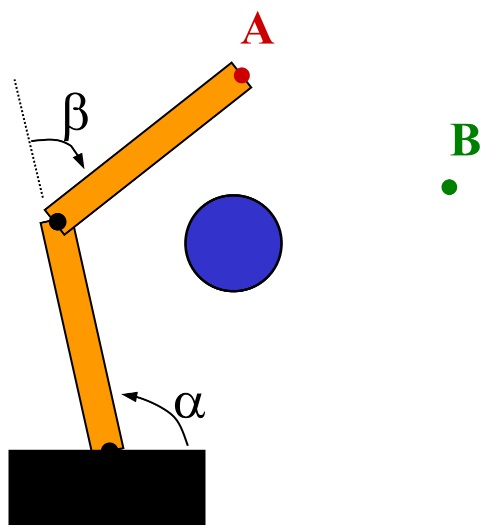


Torus

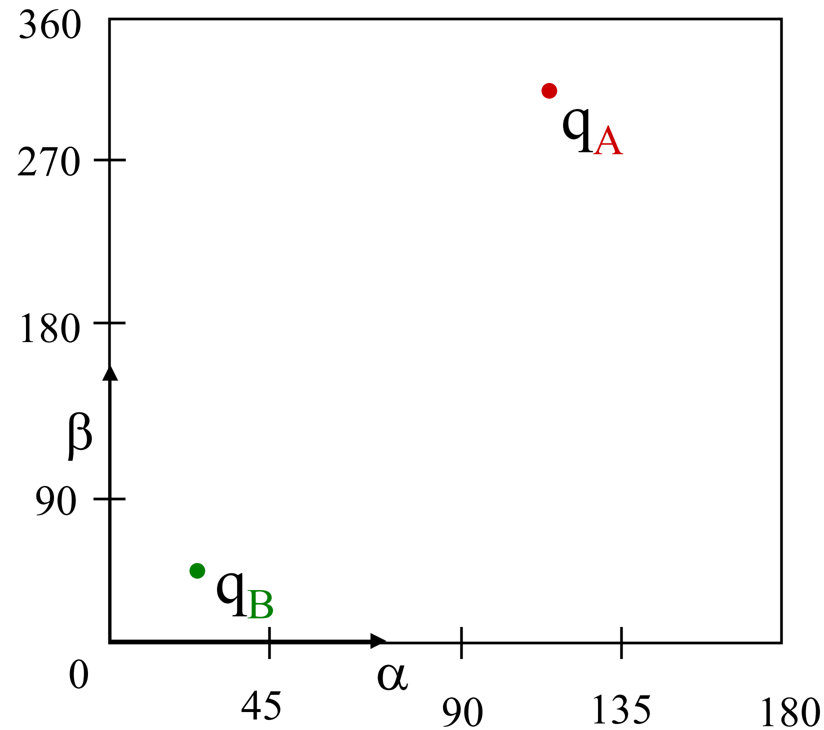
(wraps horizontally and vertically)

Configuration Space "Quiz"

Where do we put ?



An obstacle in the robot's workspace

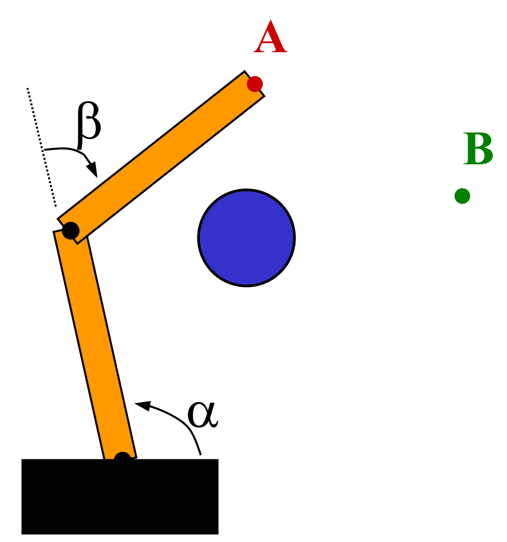


Torus

(wraps vertically)

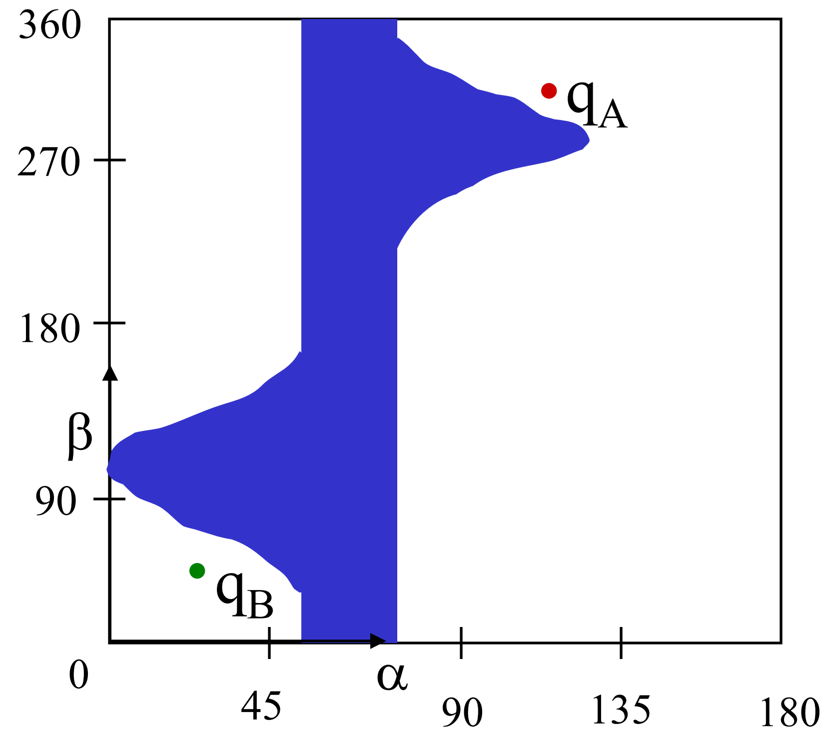
Configuration Space Obstacle

Reference *configuration*



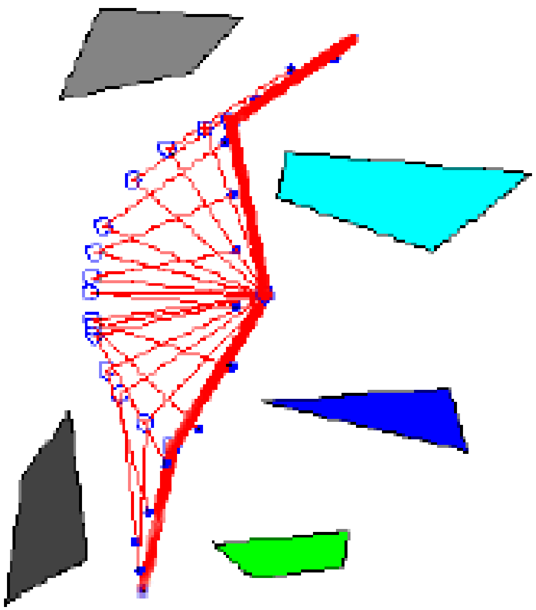
An obstacle in the robot's workspace

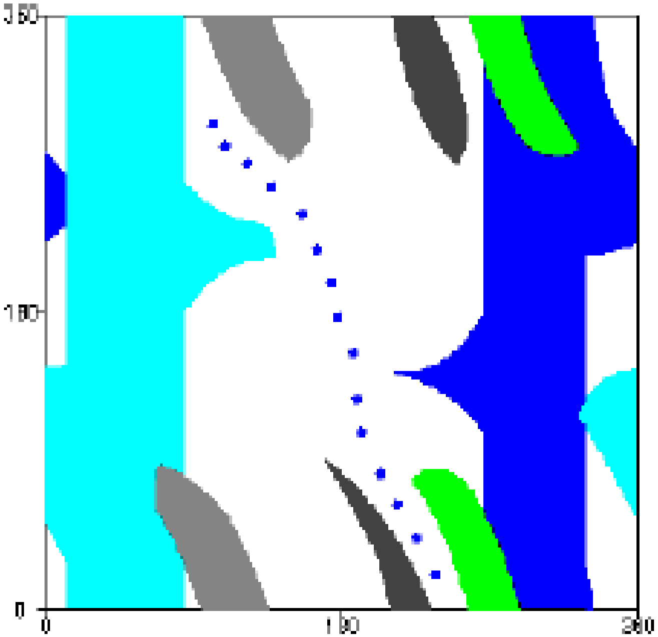
How do we get from A to B ?



The C-space representation of this obstacle…

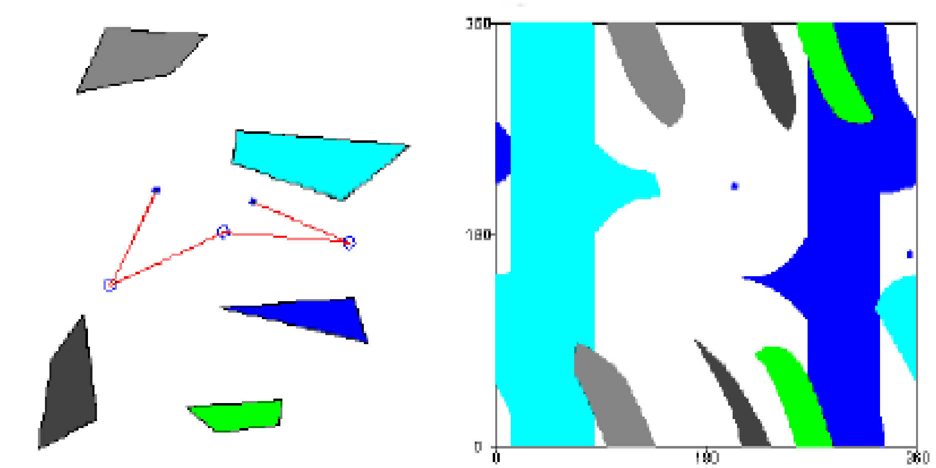
Two Link Path

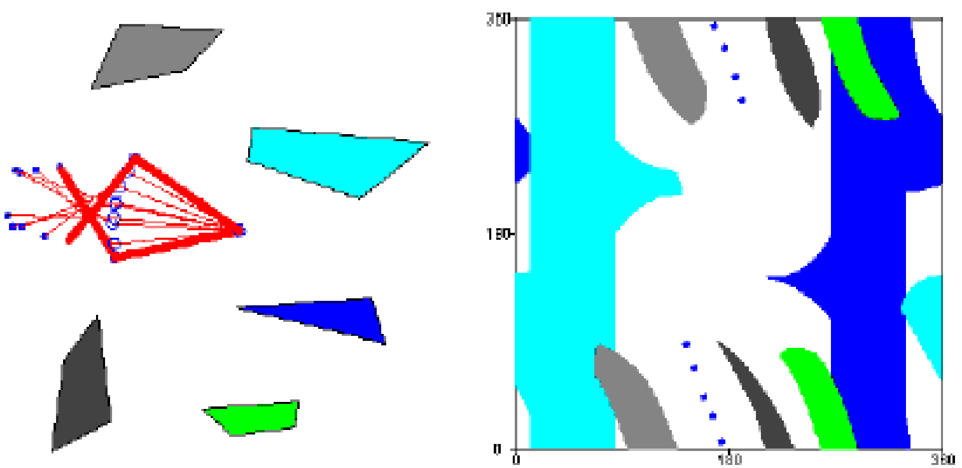




Thanks to Ken Goldberg

Two Link Path

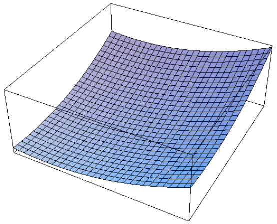




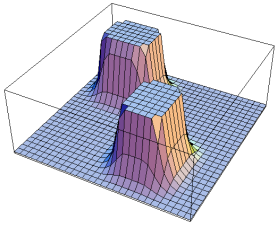
Total Potential Function

$$U(q) = U_{\text{att}}(q) + U_{\text{rep}}(q)$$

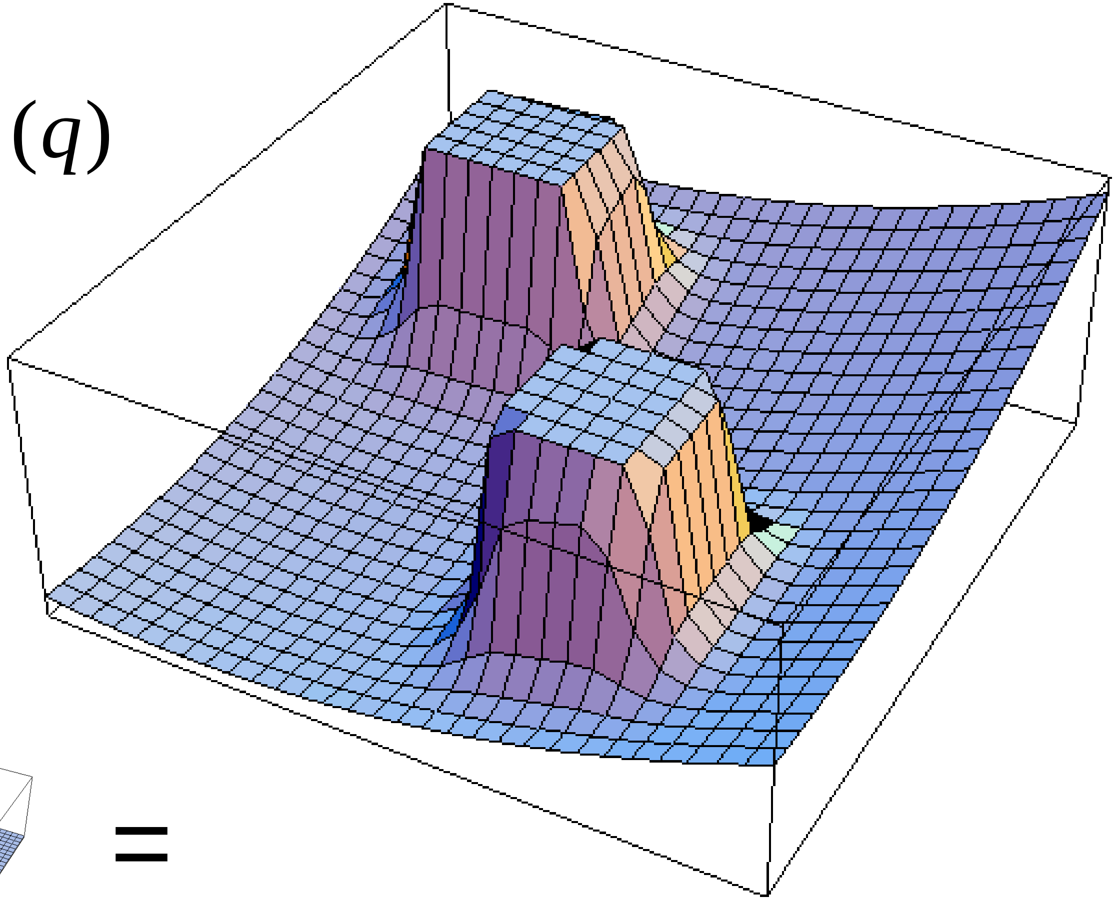
$$F(q) = -\nabla U(q)$$



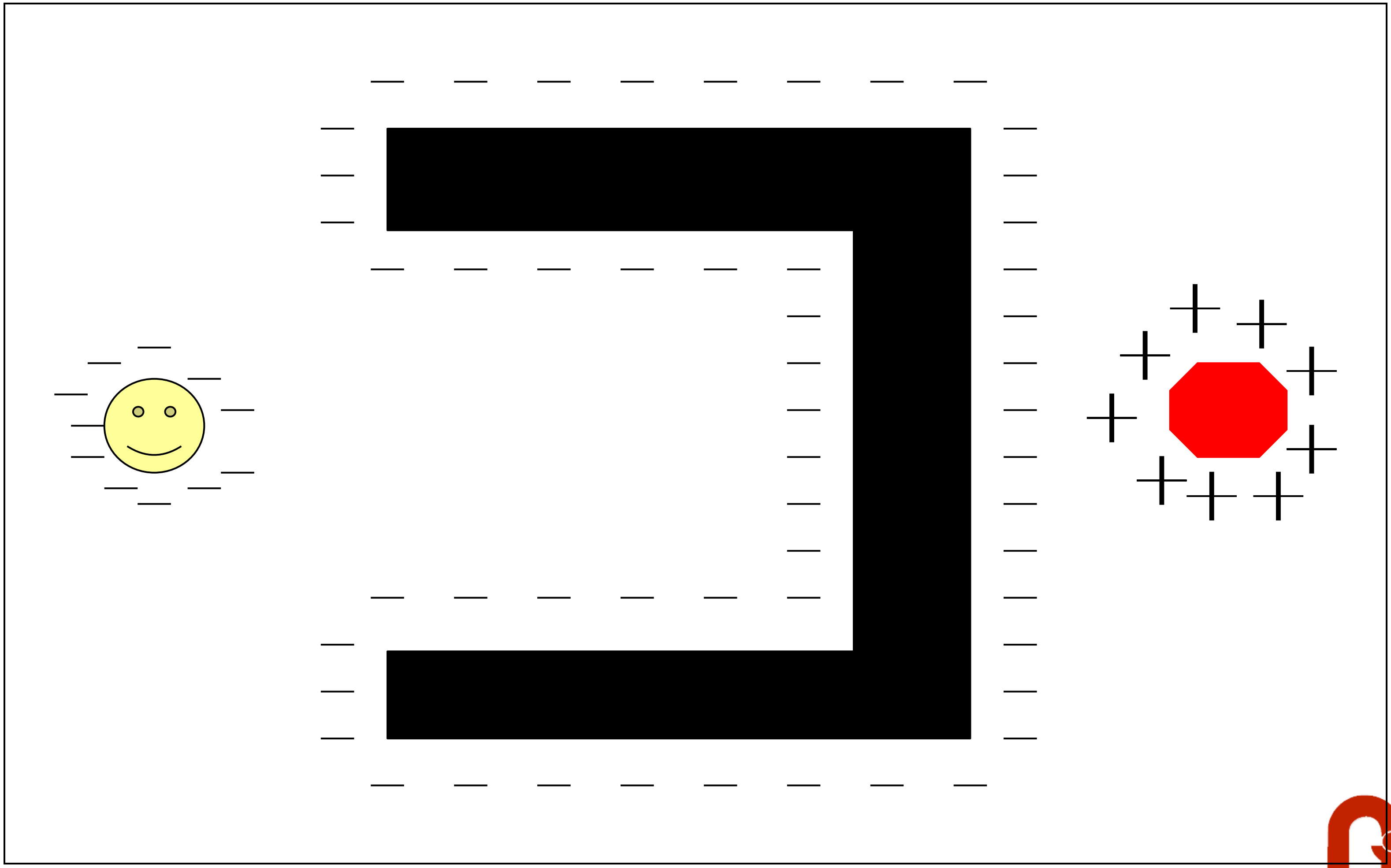
+



=

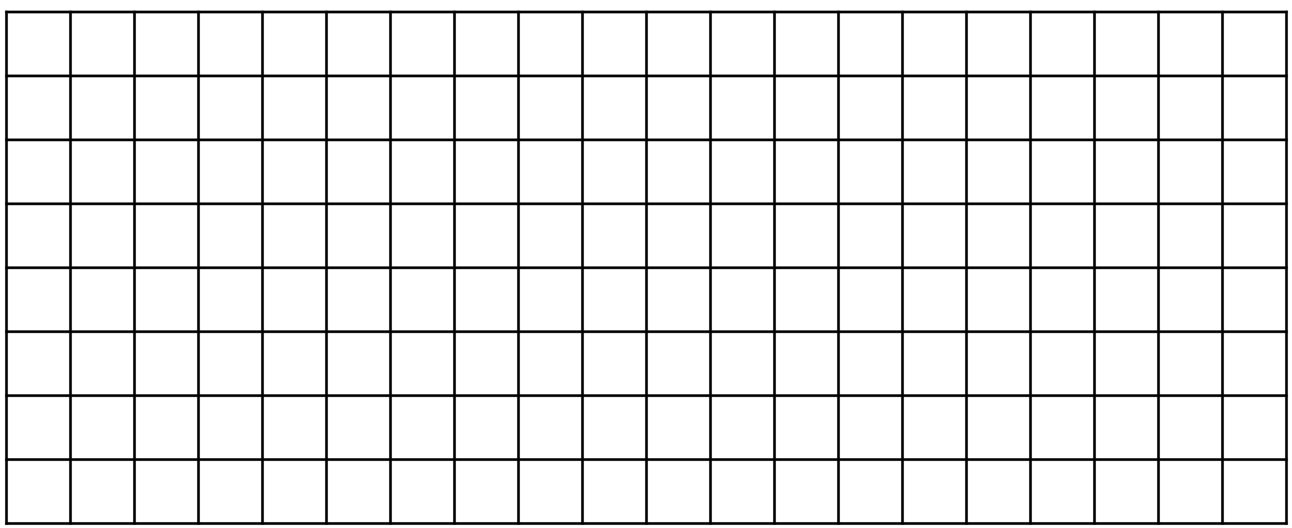


Local Minimum Problem with the Charge Analogy



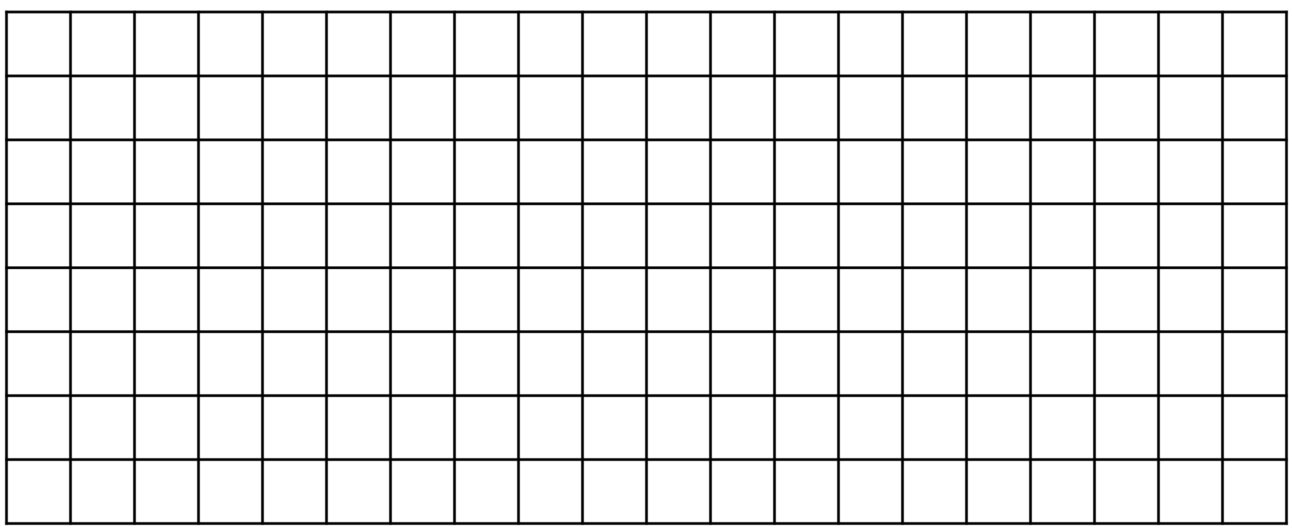
Representations

- World Representation
  - You could always use a large region and distances
  - However, a grid can be used for simplicity



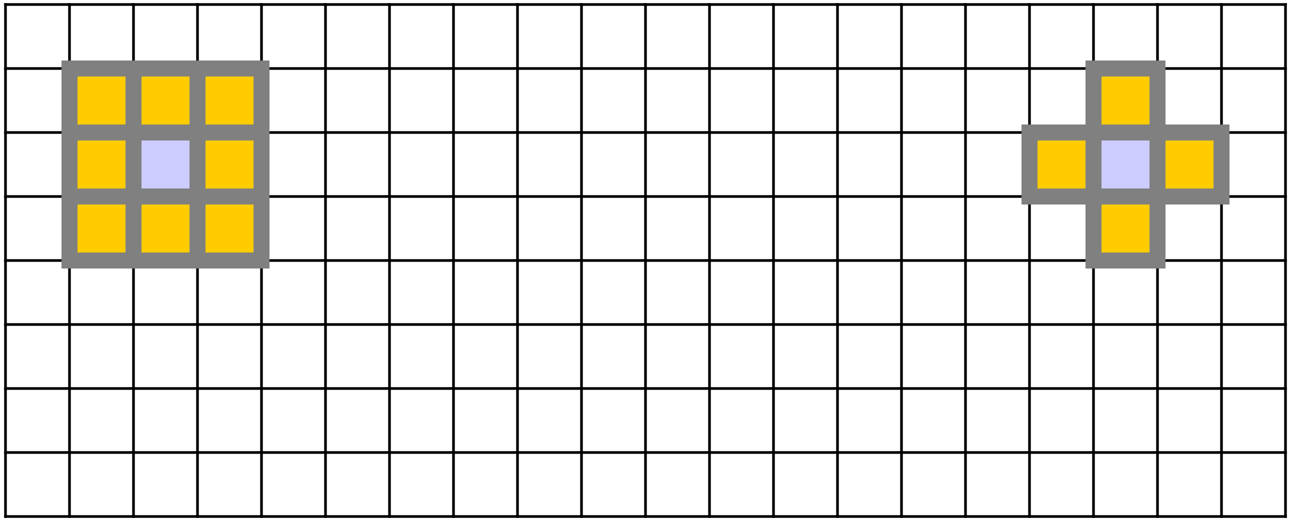
Representations: A Grid

- Distance is reduced to discrete steps
  - For simplicity, we'll assume distance is uniform
- Direction is now limited from one adjacent cell to another
  - Time to revisit Connectivity (Remember Vision?)



Representations: Connectivity

- 8-Point Connectivity
- 4-Point Connectivity
  - *(approximation of the L1 metric)*



The Wavefront Planner: Setup

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
3	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The Wavefront in Action (Part 1)

- Starting with the goal, set all adjacent cells with "0" to the current cell + 1
  - 4-Point Connectivity or 8-Point Connectivity?
  - Your Choice. We'll use 8-Point Connectivity in our example

7	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	**2**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The Wavefront in Action (Part 2)

- Now repeat with the modified cells
  - This will be repeated until no 0's are adjacent to cells with values >= 2
    - 0's will only remain when regions are unreachable

7	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4
1	0	0	0	0	0	0	0	0	0	0	0	0	0	4	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	3	**2**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The Wavefront in Action (Part 3)

- Repeat again...

7	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5	5	5	5
2	0	0	0	0	0	0	0	0	0	0	0	0	5	4	4	4
1	0	0	0	0	0	0	0	0	0	0	0	0	5	4	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	5	4	3	**2**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The Wavefront in Action (Part 4)

- And again...

7	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	6	6	6	6
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5	5	5	5
2	0	0	0	0	0	0	0	0	0	0	0	6	5	4	4	4
1	0	0	0	0	0	0	0	0	0	0	0	6	5	4	3	3
0	0	0	0	0	0	0	0	0	0	0	0	6	5	4	3	**2**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The Wavefront in Action (Part 5)

- And again until…

7	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	7	7	7	7	7
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	6	6	6	6
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5	5	5	5
2	0	0	0	0	0	0	0	0	0	0	7	6	5	4	4	4
1	0	0	0	0	0	0	0	0	0	0	7	6	5	4	3	3
0	0	0	0	0	0	0	0	0	0	0	7	6	5	4	3	**2**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The Wavefront in Action (Done)

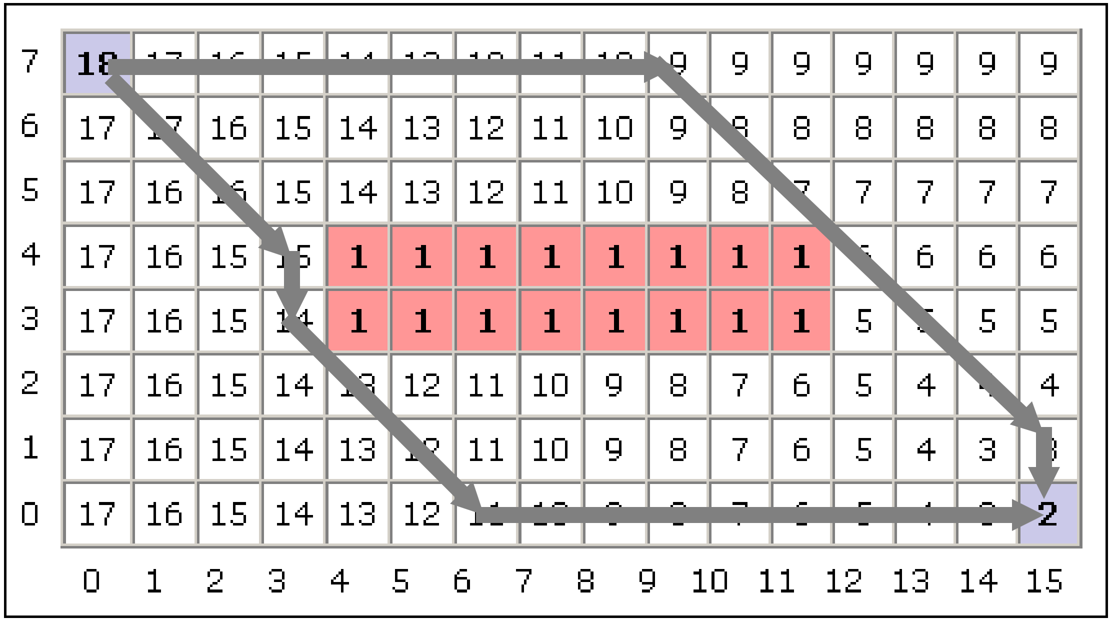
- You're done
  - Remember, 0's should only remain if unreachable regions exist

7	**18**	17	16	15	14	13	12	11	10	9	9	9	9	9	9	9
6	17	17	16	15	14	13	12	11	10	9	8	8	8	8	8	8
5	17	16	16	15	14	13	12	11	10	9	8	7	7	7	7	7
4	17	16	15	15	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	6	6	6	6
3	17	16	15	14	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5	5	5	5
2	17	16	15	14	13	12	11	10	9	8	7	6	5	4	4	4
1	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	3
0	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	**2**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

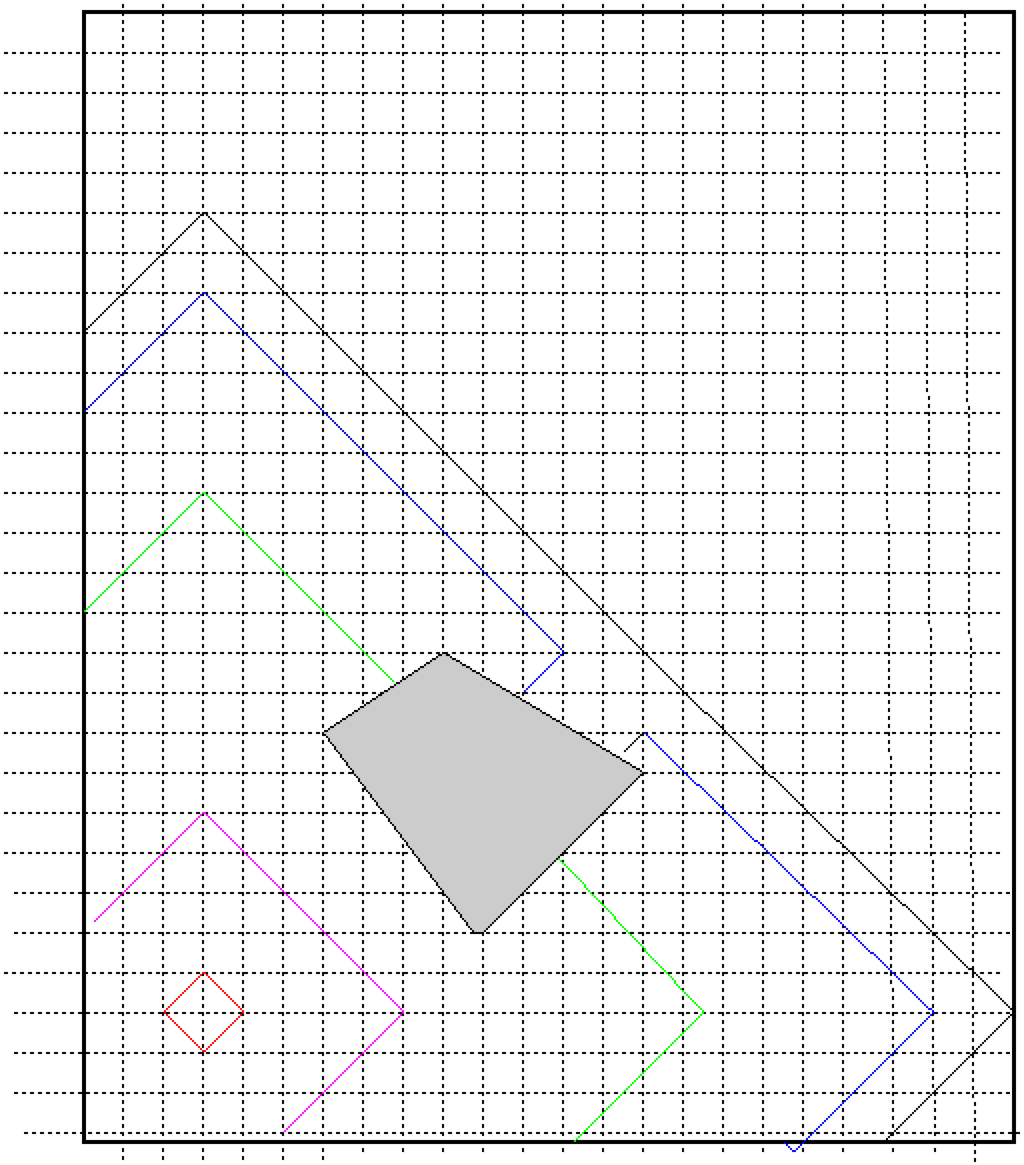
The Wavefront, Now What?

- To find the shortest path, according to your metric, simply always move toward a cell with a lower number
  - The numbers generated by the Wavefront planner are roughly proportional to their distance from the goal

Two possible shortest paths shown



This is really a Continuous Solution

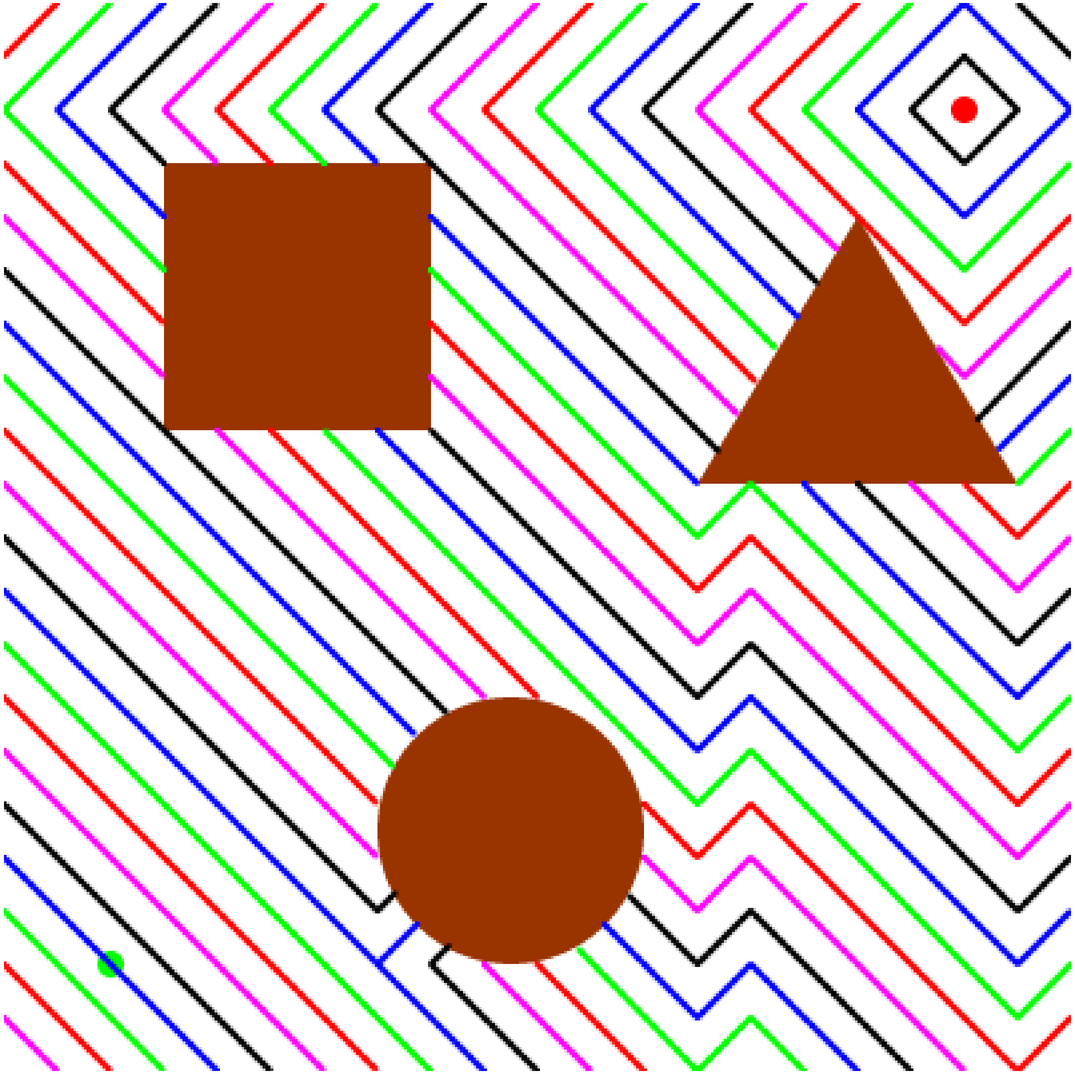


Not pixels

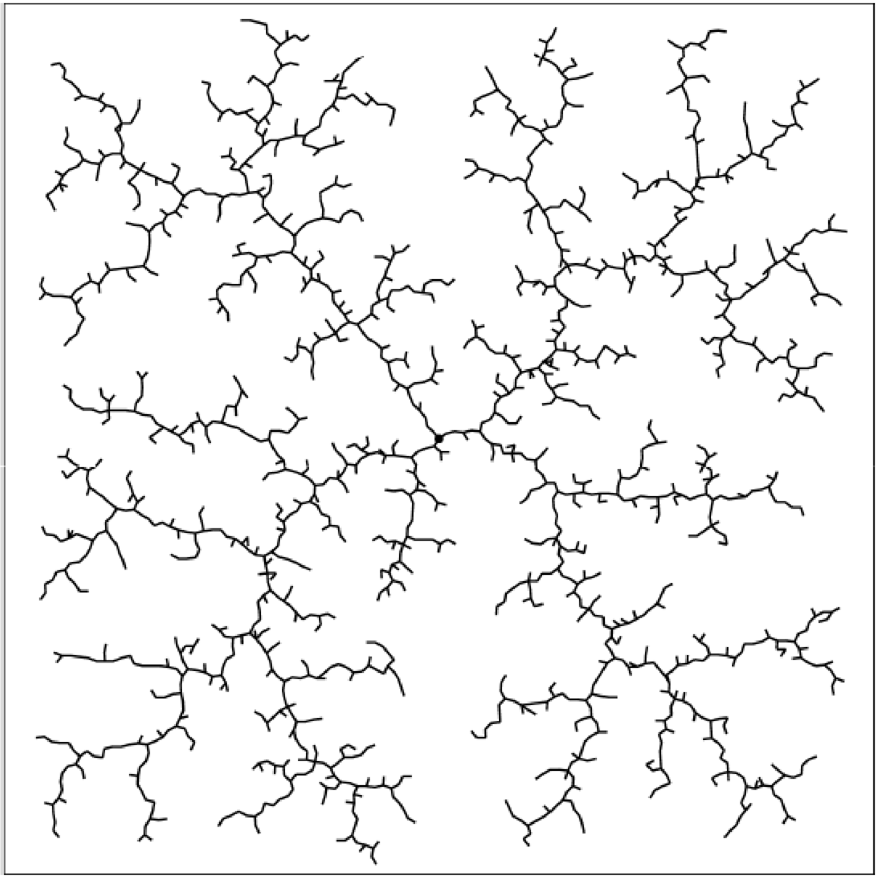
Waves bend

L1 distance

Pretty Wavefront



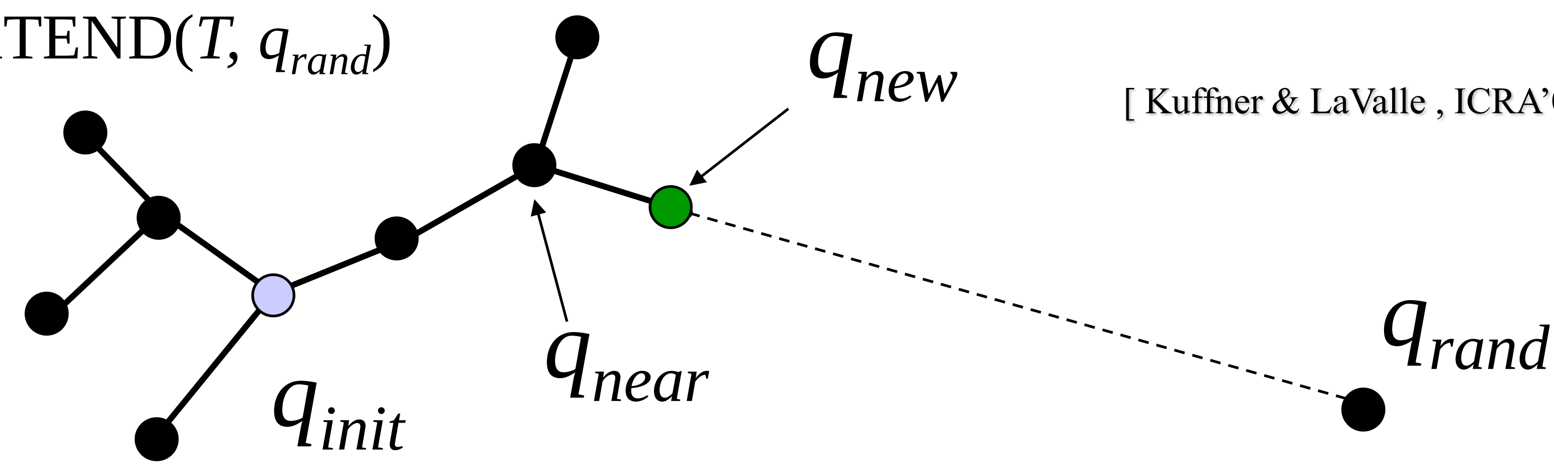
Rapidly-Exploring Random Tree



Path Planning with RRTs (Rapidly-Exploring Random Trees)

BUILD_RRT (q_{init}) {
    T.init(q_{init});
    for *k* = 1 to K do
        q_{rand} = RANDOM_CONFIG();
        EXTEND(*T*, q_{rand})
}

EXTEND(*T*, q_{rand})

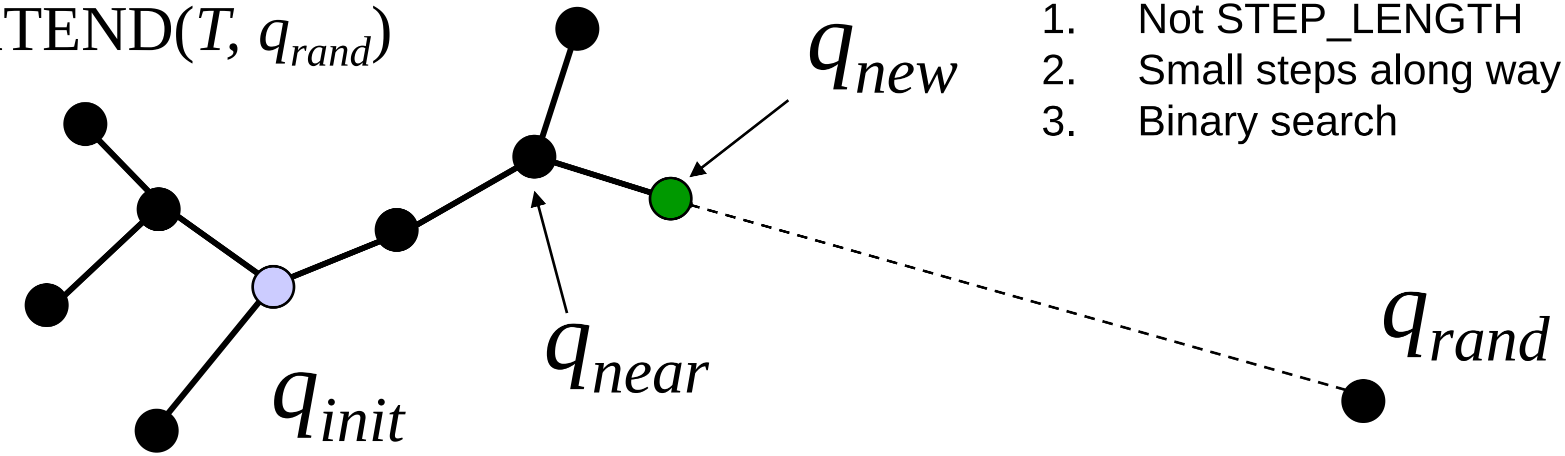


[Kuffner & LaValle , ICRA'00]

Path Planning with RRTs (Some Details)

BUILD_RRT (q_{init}) {
  $T.init(q_{init})$;
  for k = 1 to K do
    q_{rand} = RANDOM_CONFIG();
    EXTEND(T, q_{rand})
}

EXTEND(T, q_{rand})



STEP_LENGTH: How far to sample

1. Sample just at end point
2. Sample all along
3. Small Step

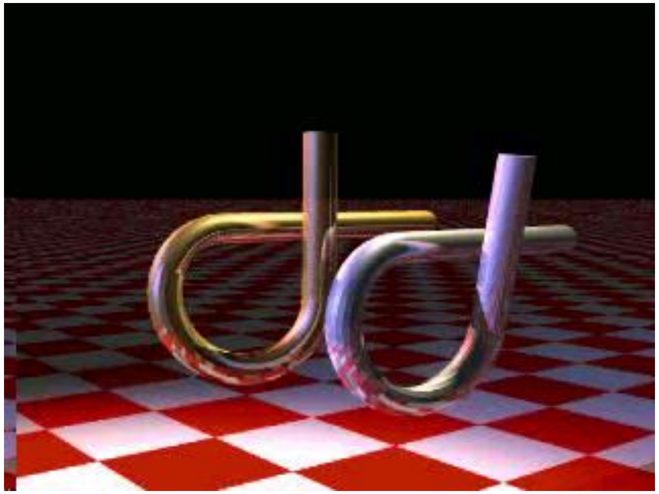
Extend returns

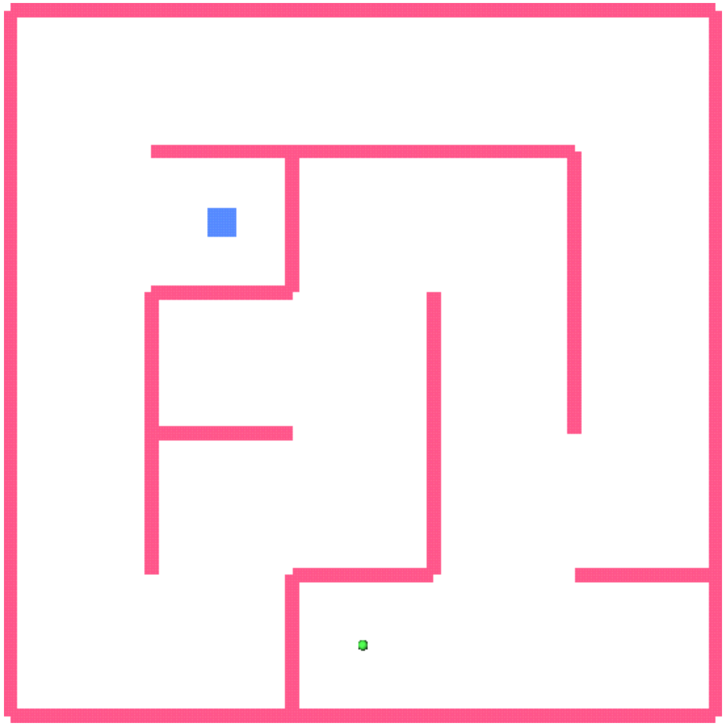
1. Trapped, cant make it
2. Extended, steps toward node
3. Reached, connects to node

STEP_SIZE

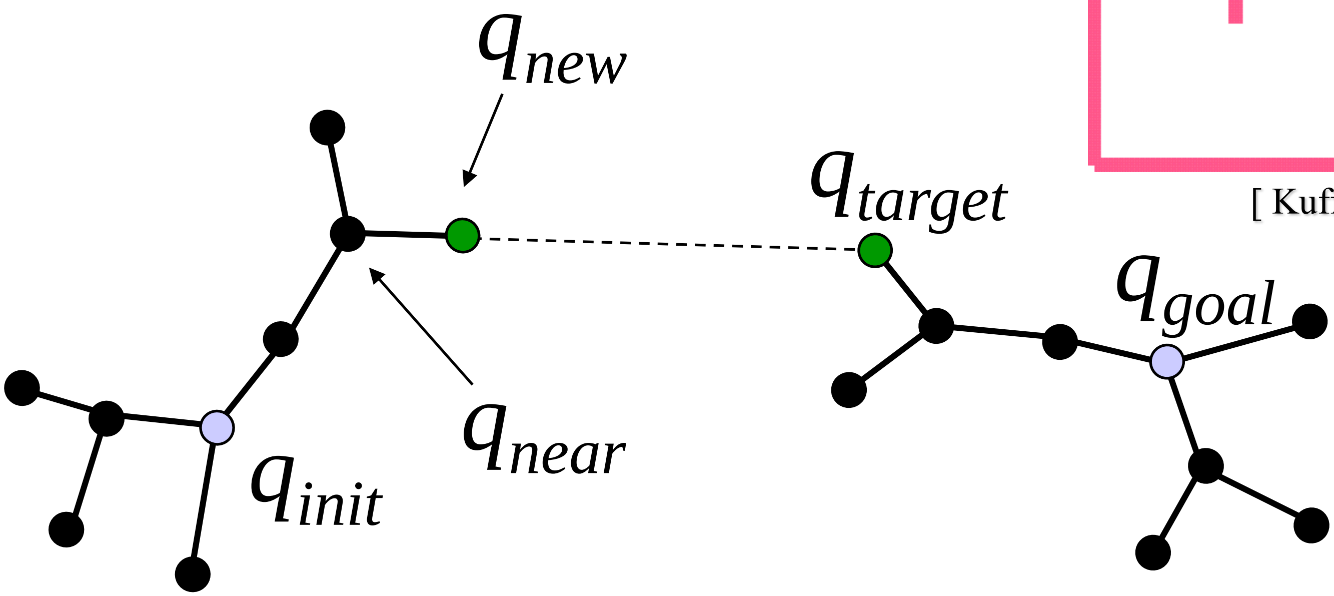
1. Not STEP_LENGTH
2. Small steps along way
3. Binary search

Grow two RRTs towards each other





[Kuffner, LaValle ICRA '00]



Map-Based Approaches: Roadmap Theory

- Properties of a roadmap:
  - Accessibility: there exists a collision-free path from the start to the road map
  - Departability: there exists a collision-free path from the roadmap to the goal.
  - Connectivity: there exists a collision-free path from the start to the goal (on the roadmap).

- **a roadmap exists ⇔ a path exists**
- **Examples of Roadmaps**
  - **Generalized Voronoi Graph (GVG)**
  - **Visibility Graph**

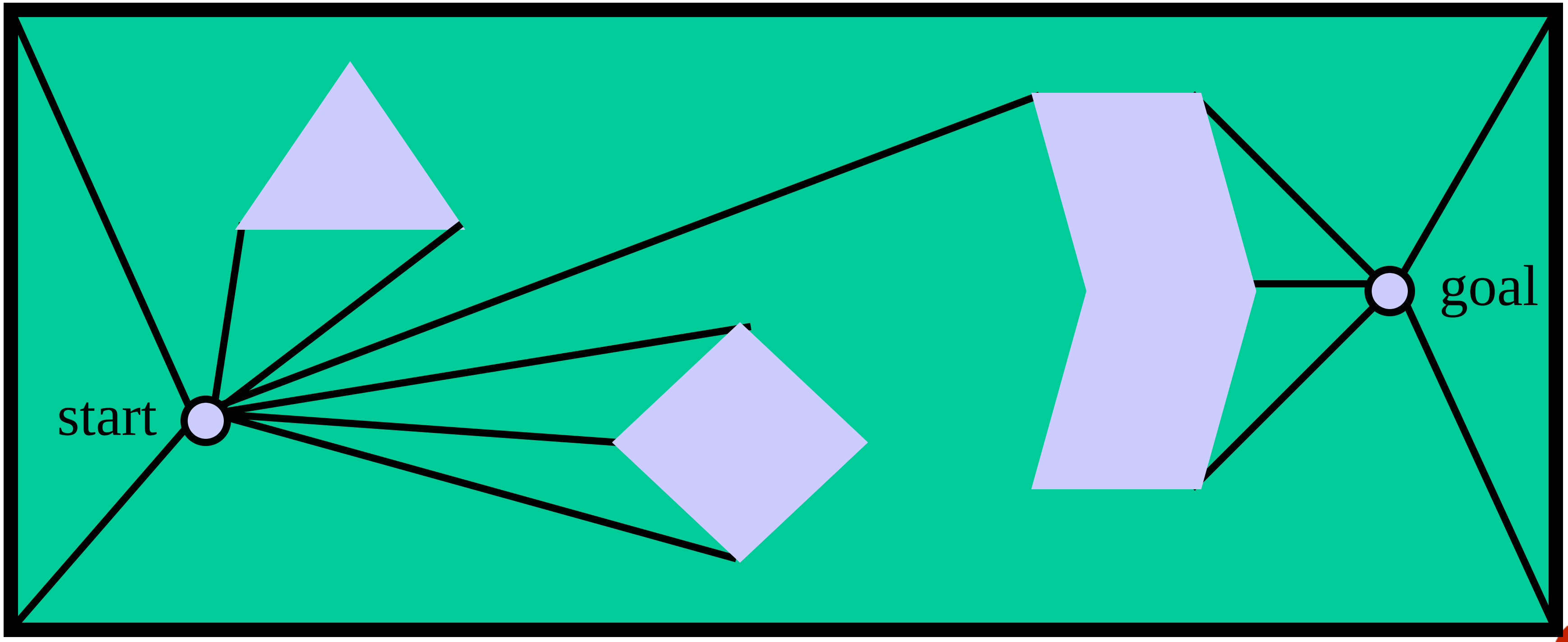
THE ROBOTICS INSTITUTE

Roadmap: Visibility Graph

- Formed by connecting all "visible" vertices, the start point and the end point, to each other
- For two points to be "visible" no obstacle can exist between them
  - Paths exist on the perimeter of obstacles
- In our example, this produces the shortest path with respect to the L2 metric. However, the close proximity of paths to obstacles makes it dangerous

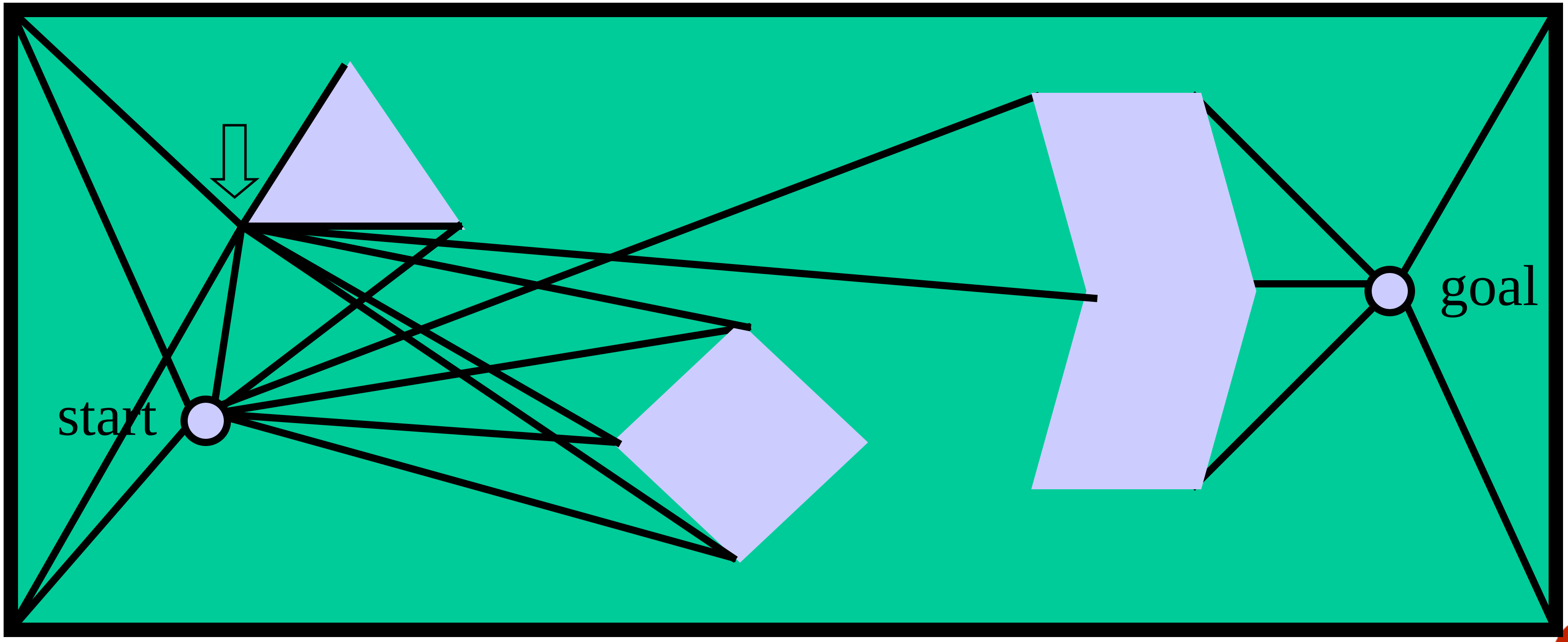
The Visibility Graph in Action (Part 1)

- First, draw lines of sight from the start and goal to all "visible" vertices and corners of the world.



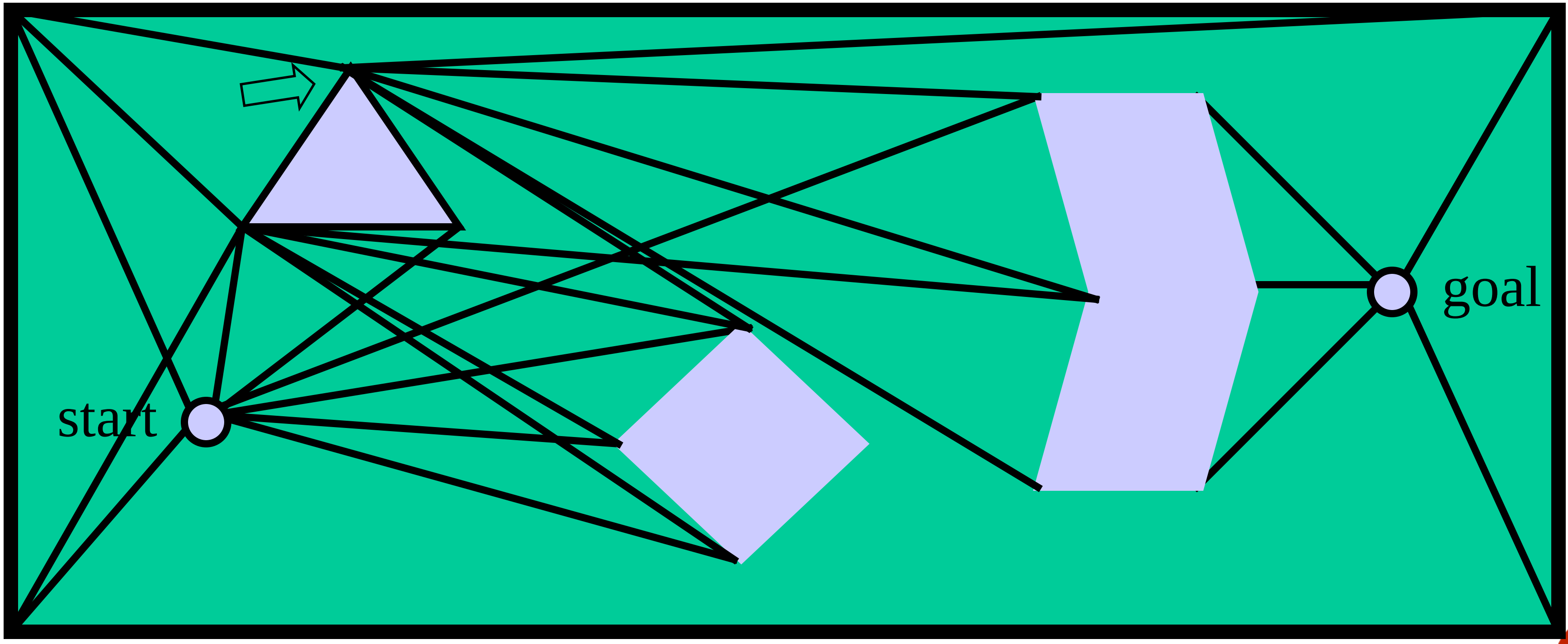
The Visibility Graph in Action (Part 2)

- Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



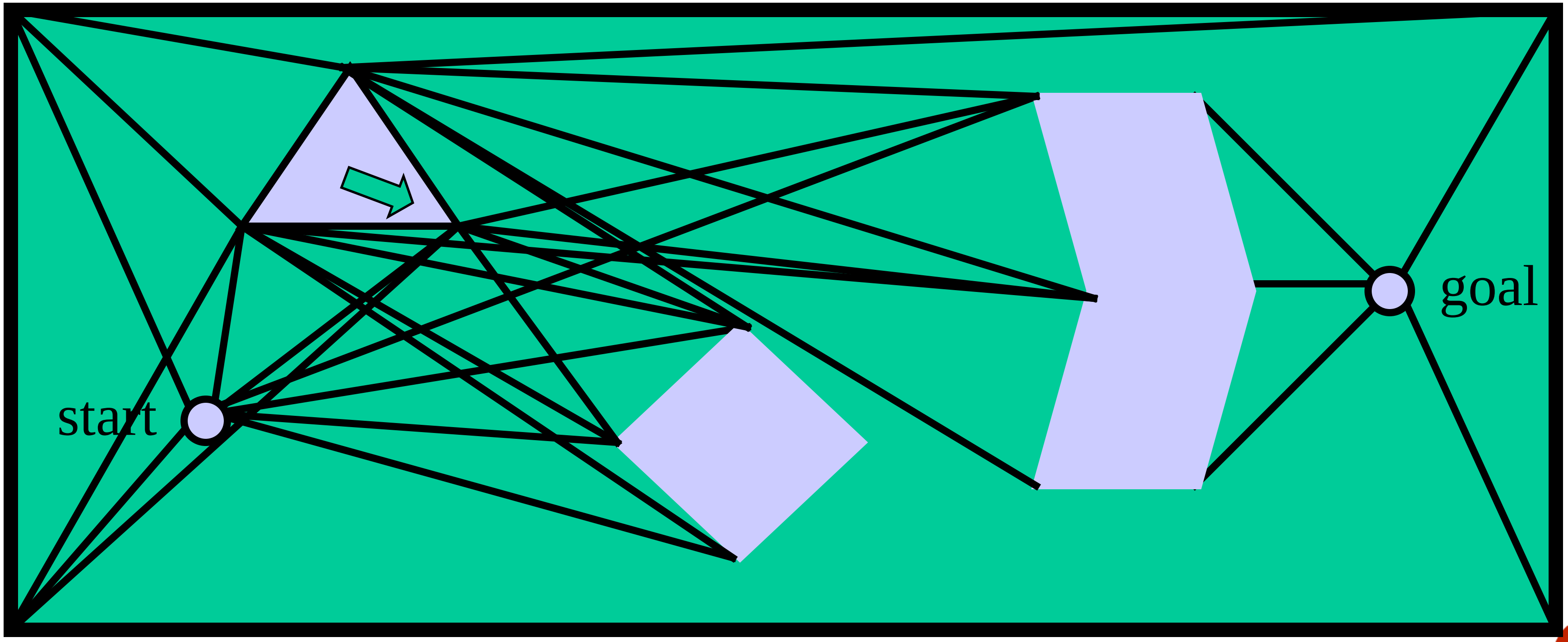
The Visibility Graph in Action (Part 3)

- Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



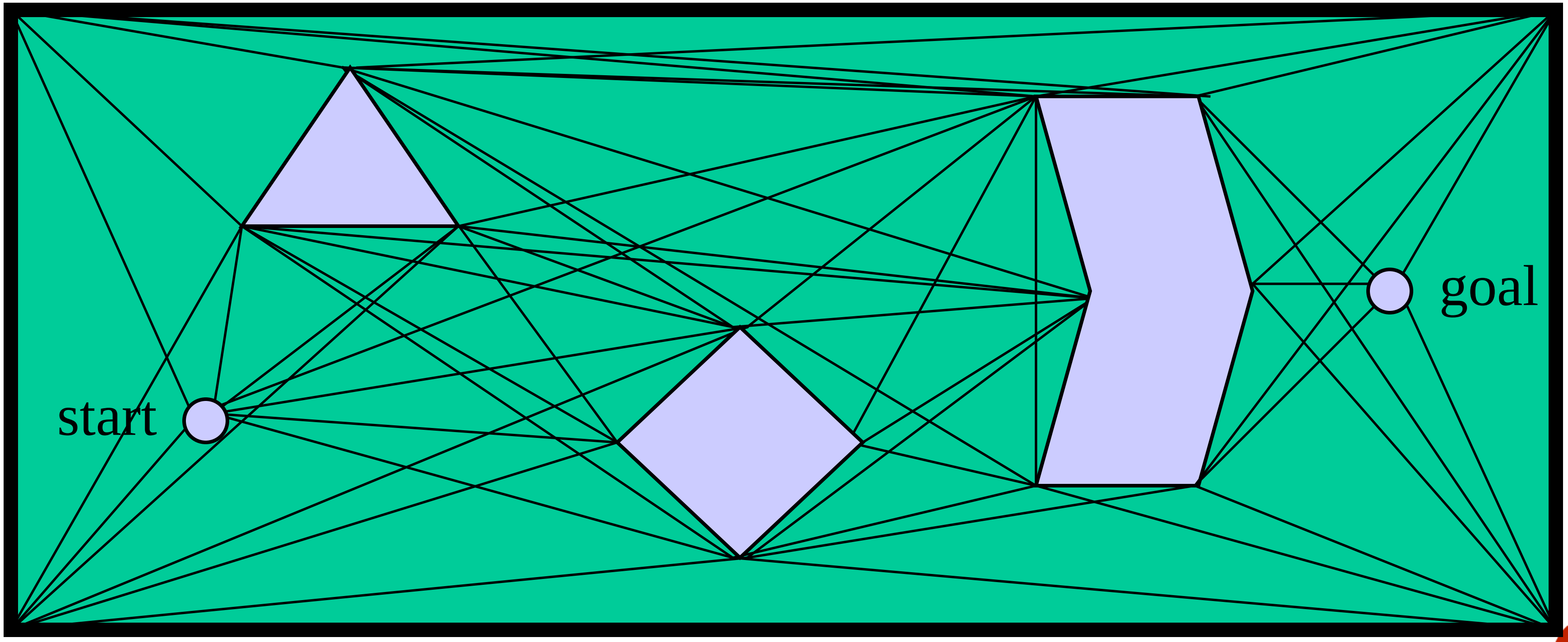
The Visibility Graph in Action (Part 4)

- Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



The Visibility Graph (Done)

- Repeat until you're done.



Visibility Graph Overview

- Start with a map of the world, draw lines of sight from the start and goal to every "corner" of the world and vertex of the obstacles, not cutting through any obstacles.
- Draw lines of sight from every vertex of every obstacle like above. Lines along edges of obstacles are lines of sight too, since they don't pass through the obstacles.
- If the map was in Configuration space, each line potentially represents part of a path from the start to the goal.

Graph Search

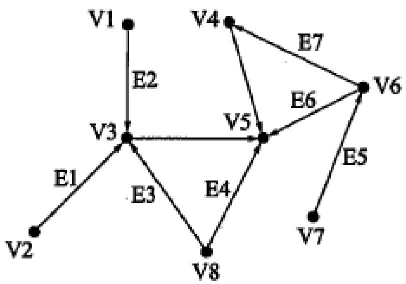
Howie Choset

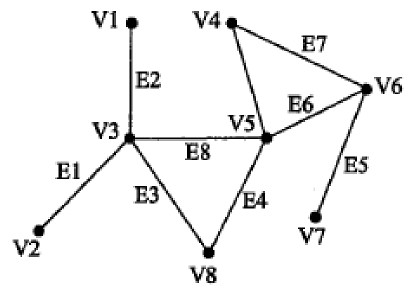
16-311

Outline

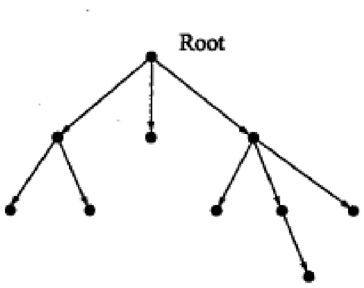
- Overview of Search Techniques
- A* Search

Graphs

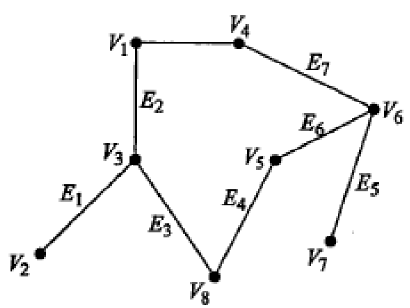




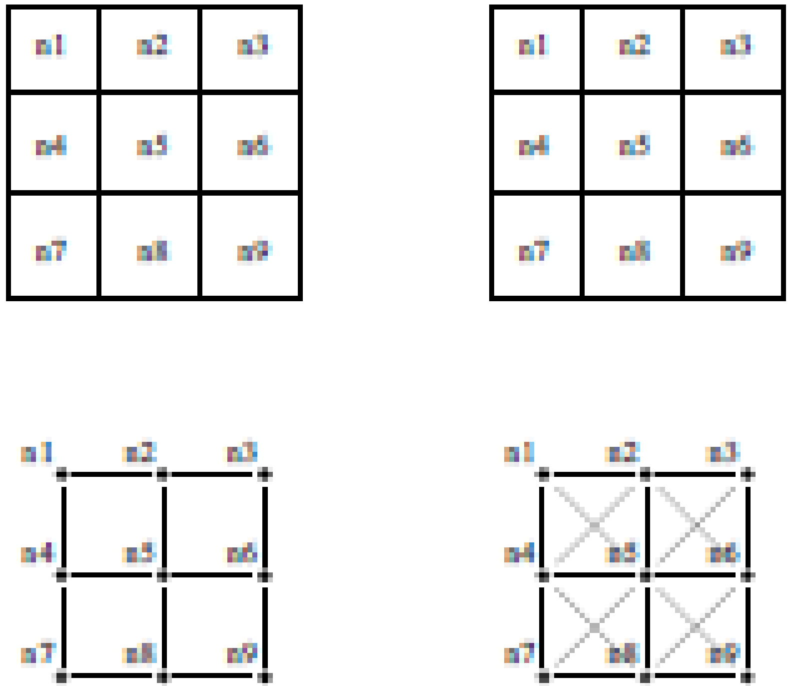
Collection of Edges and Nodes (Vertices)



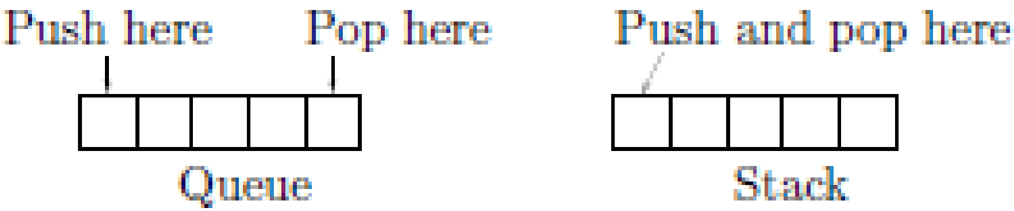
A tree



Grids


n1 n2 n3
n4 n5 n6
n7 n8 n9
n1 n2 n3
n4 n5 n6
n7 n8 n9
n1 n2 n3
n4 n5 n6
n7 n8 n9
n1 n2 n3
n4 n5 n6
n7 n8 n9

Stacks and Queues

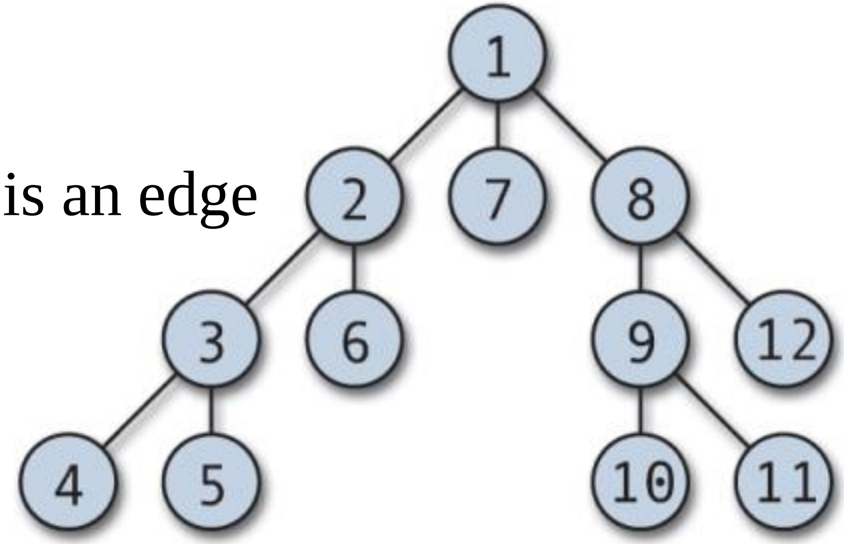


Stack: First in, Last out (FILO)

Queue: First in, First out (FIFO)

Depth First Search

algorithm dft(x)
    visit(x)
    FOR each y such that (x,y) is an edge
        IF y was not visited yet
            THEN dft(y)

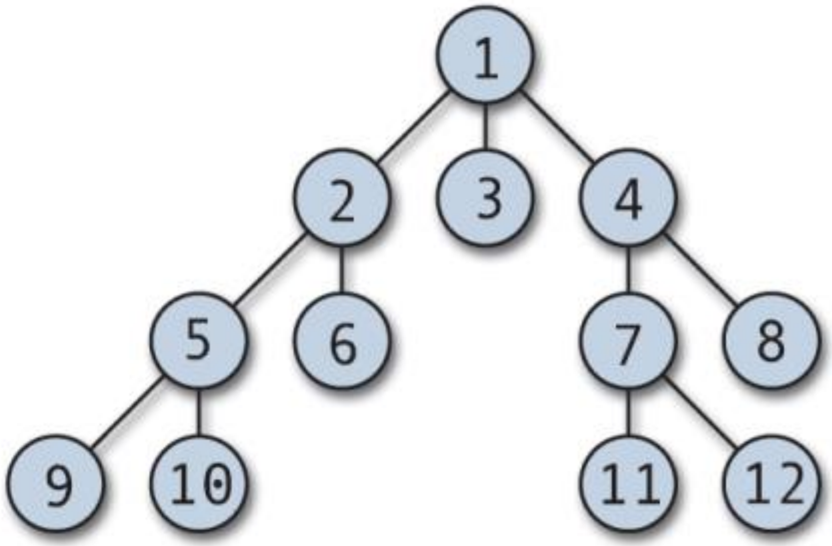


Copied from wikipedia

Worst case performance	$O(\|V\| + \|E\|)$ for explicit graphs traversed without repetition,
Worst case space complexity	$O(\|V\|)$ if entire graph is traversed without repetition, O(longest path length searched) for implicit graphs without elimination of duplicate nodes

Breadth First Search

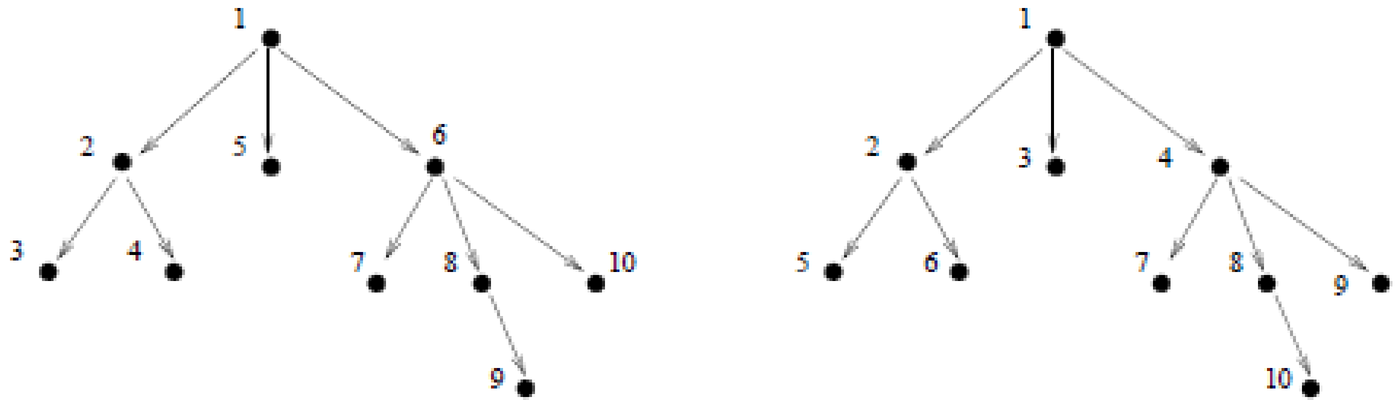
```
visit(start node)
queue <- start node
WHILE queue is nor empty D(
        x <- queue
        FOR each y such that (
                and y has not be
            visit(y)
           queue <- y
          END
END
```



wikipedia

http://www.cse.ohio-state.edu/~gurari/course/cis680/cis680Ch14.html

Depth First and Breadth First



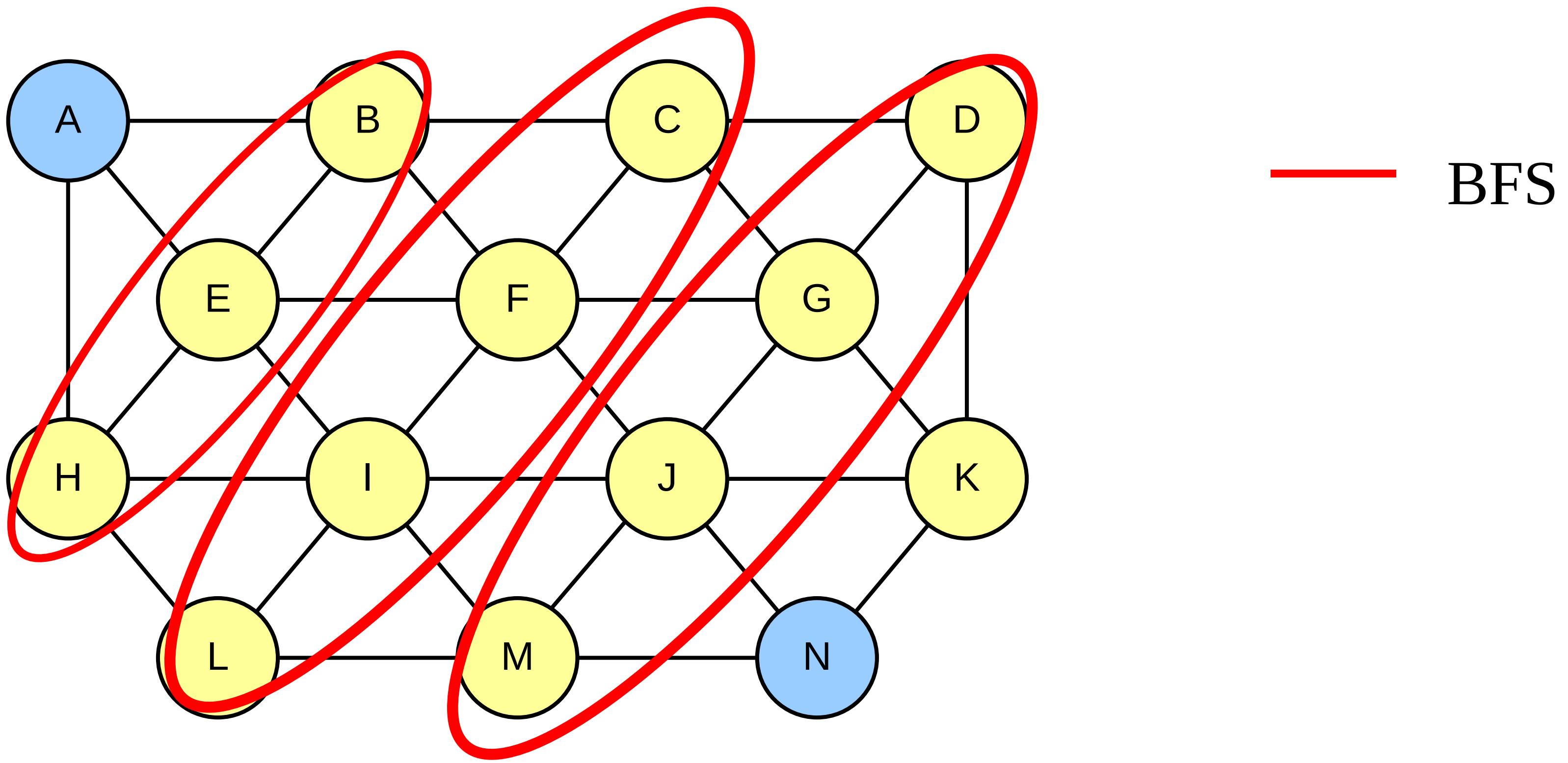
Wavefront Planner: A BFS

Search

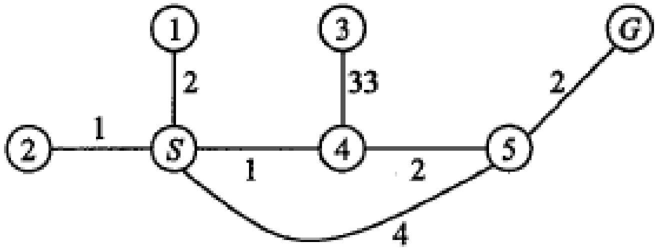
- Uninformed Search
  - Use no information obtained from the environment
  - Blind Search: BFS (Wavefront), DFS
- Informed Search
  - Use evaluation function
  - More efficient
  - Heuristic Search: A*, D*, etc.

Uninformed Search

Graph Search from A to N

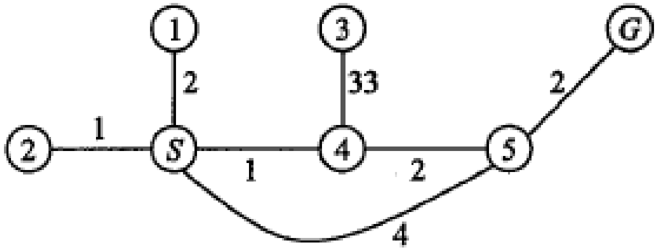


Dijkstra's Search: f(n) = g(n)



Pop lowest f first

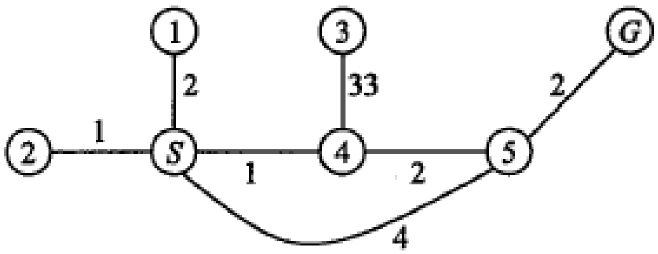
Dijkstra's Search: f(n) = g(n)



Pop lowest f first

1. O = {S}

Dijkstra's Search: f(n) = g(n)

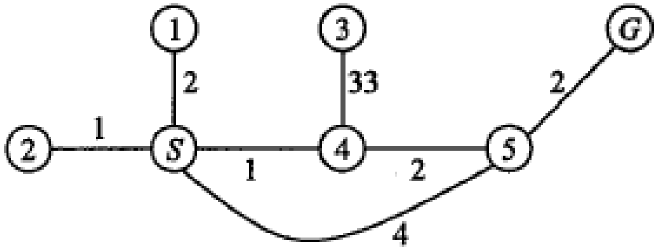


Pop lowest f first

1. O = {S}

2. O = {1, 2, 4, 5}; C = {S} (1,2,4,5 all back point to S)

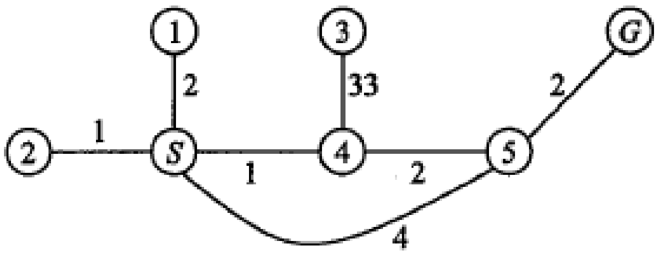
Dijkstra's Search: f(n) = g(n)



Pop lowest f first

1. O = {S}
2. O = {1, 2, 4, 5}; C = {S} (1,2,4,5 all back point to S)
3. O = {1, 4, 5}; C = {S, 2} (there are no adjacent nodes not in C)

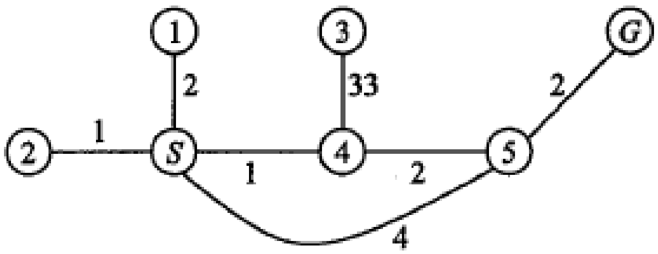
Dijkstra's Search: $f(n) = g(n)$



Pop lowest f first

1. O = {S}
2. O = {1, 2, 4, 5}; C = {S} (1,2,4,5 all back point to S)
3. O = {1, 4, 5}; C = {S, 2} (there are no adjacent nodes not in C)
4. O = {1, 5, 3}; C = {S, 2, 4} (1, 2, 4 point to S; 5 points to 4)

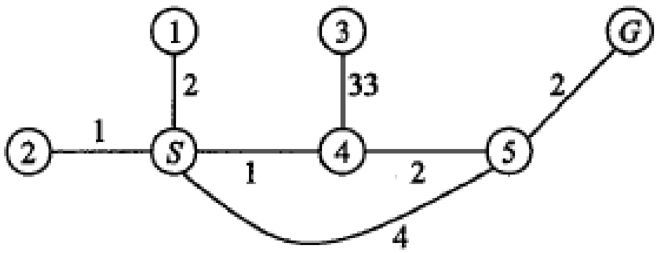
Dijkstra's Search: f(n) = g(n)



Pop lowest f first

1. O = {S}
2. O = {1, 2, 4, 5}; C = {S} (1,2,4,5 all back point to S)
3. O = {1, 4, 5}; C = {S, 2} (there are no adjacent nodes not in C)
4. O = {1, 5, 3}; C = {S, 2, 4} (1, 2, 4 point to S; 5 points to 4)
5. O = {5, 3}; C = {S, 2, 4, 1}

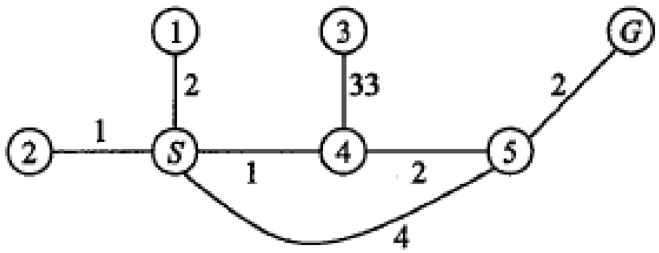
Dijkstra's Search: f(n) = g(n)



Pop lowest f first

1. O = {S}
2. O = {1, 2, 4, 5}; C = {S} (1,2,4,5 all back point to S)
3. O = {1, 4, 5}; C = {S, 2} (there are no adjacent nodes not in C)
4. O = {1, 5, 3}; C = {S, 2, 4} (1, 2, 4 point to S; 5 points to 4)
5. O = {5, 3}; C = {S, 2, 4, 1}
6. O = {3, G}; C = {S, 2, 4 1,5} (goal points to 5 which points to 4 which points to S)

Dijkstra's Search: f(n) = g(n)



Pop lowest f first

1. O = {S}
2. O = {1, 2, 4, 5}; C = {S} (1,2,4,5 all back point to S)
3. O = {1, 4, 5}; C = {S, 2} (there are no adjacent nodes not in C)
4. O = {1, 5, 3}; C = {S, 2, 4} (1, 2, 4 point to S; 5 points to 4)
5. O = {5, 3}; C = {S, 2, 4, 1}
6. O = {3, G}; C = {S, 2, 4 1,5} (goal points to 5 which points to 4 which points to S)
7. O = {3}; C = {S, 2, 4, 1,5} (Path found because Goal was popped)

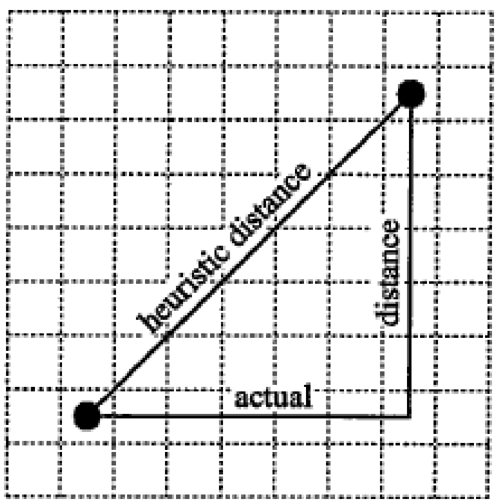
Informed Search: A*

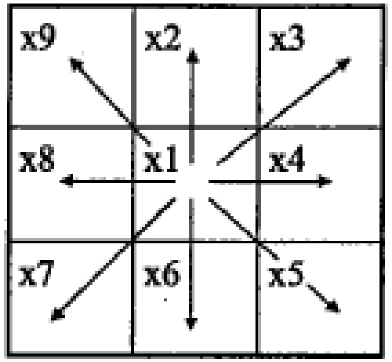
Notation

- n → node/state
- $c(n_1,n_2)$ → the length of an edge connecting between n_1 and n_2
- $b(n_1) = n_2$ → backpointer of a node n_1 to a node n_2.

Informed Search: A*

- Evaluation function, ***f(n) = g(n) + h(n)***
- Operating cost function, ***g(n)***
  - Actual operating cost having been already traversed
- Heuristic function, ***h(n)***
  - Information used to find the promising node to traverse
  - Admissible → never overestimate the actual path cost





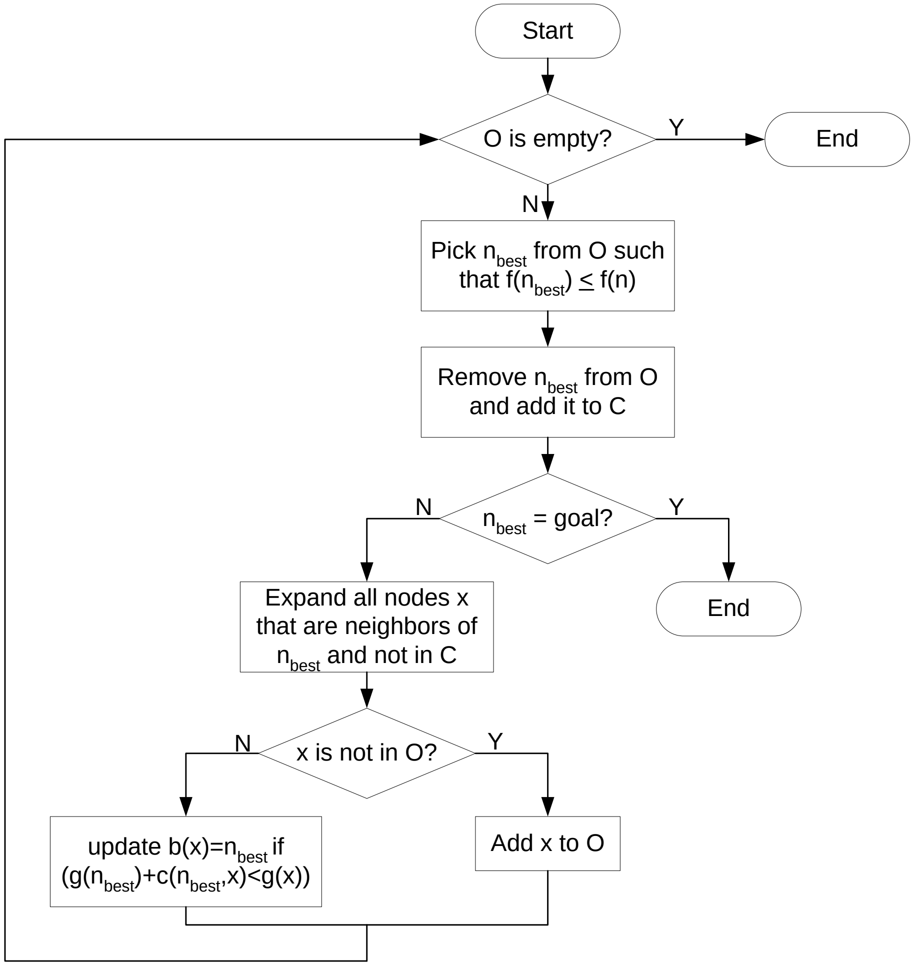
c(x1, x2) = 1
c(x1, x9) = 1.4

c(x1, x8) = 10000, if x8 is in obstacle, x1 is a free cell

c(x1,x9) = 10000.4, if x9 is in obstacle, x1 is a free cell

Cost on a grid

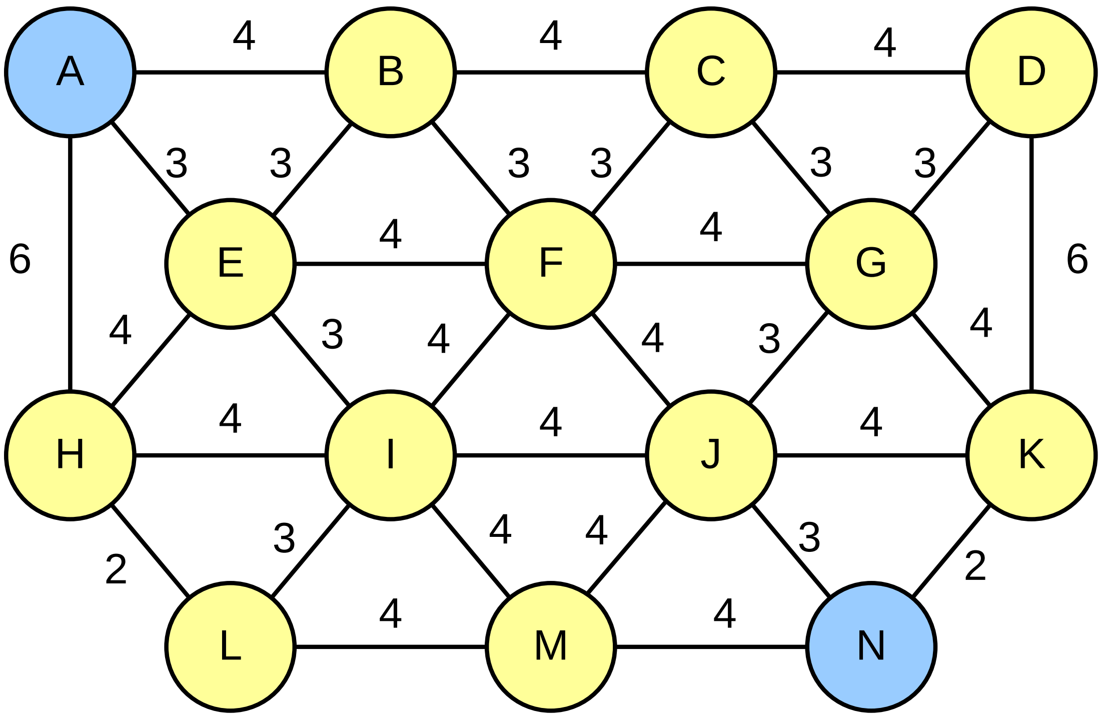
A*: Algorithm



The search requires 2 lists to store information about nodes

1) **Open list (O)** stores nodes for expansions
2) **Closed list (C)** stores nodes which we have explored

A*: Example (1/6)

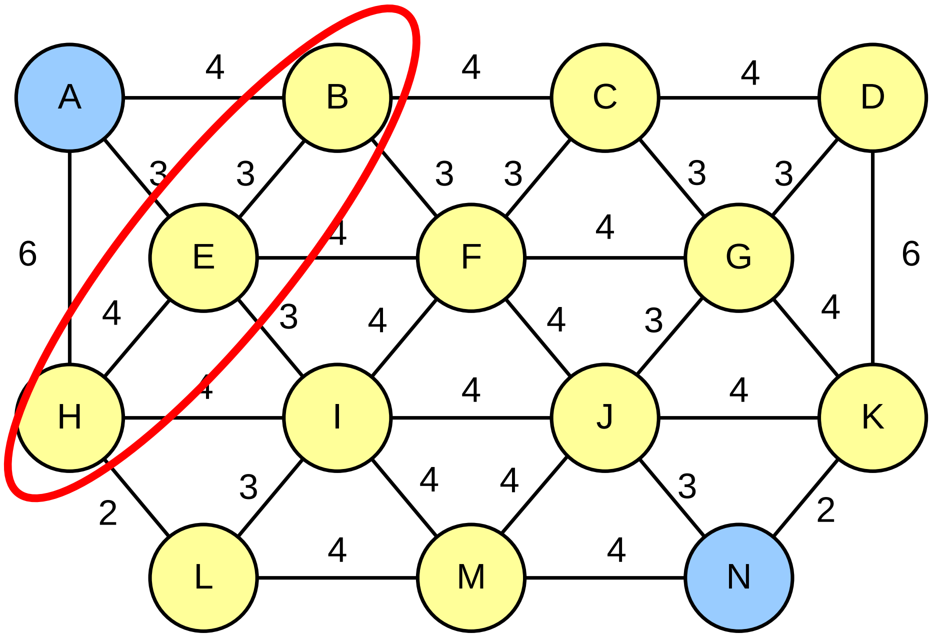


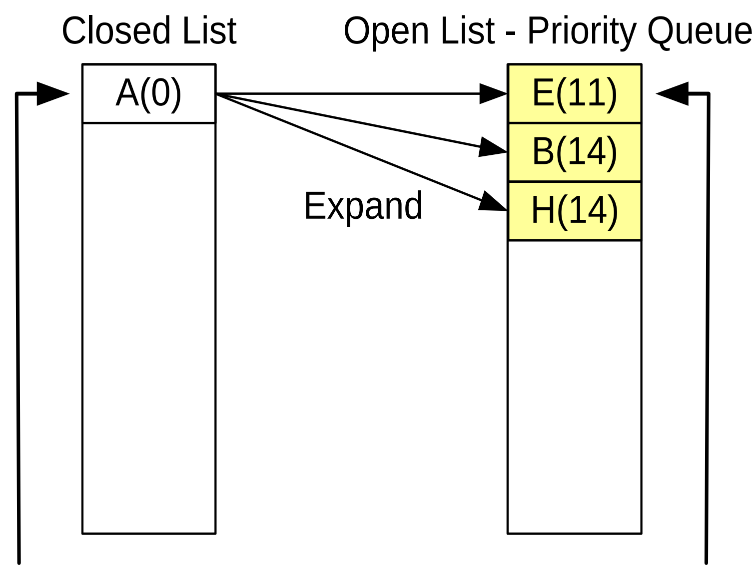
Heuristics

A = 14	H = 8
B = 10	I = 5
C = 8	J = 2
D = 6	K = 2
E = 8	L = 6
F = 7	M = 2
G = 6	N = 0

Legend <u>ope</u>rating cost

A*: Example (2/6)





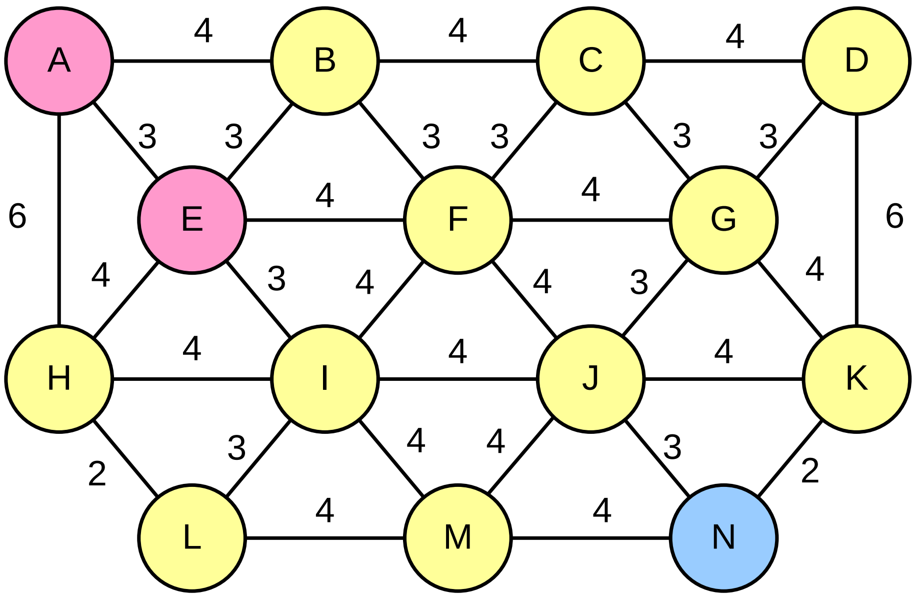
A(x) = A(g(n))

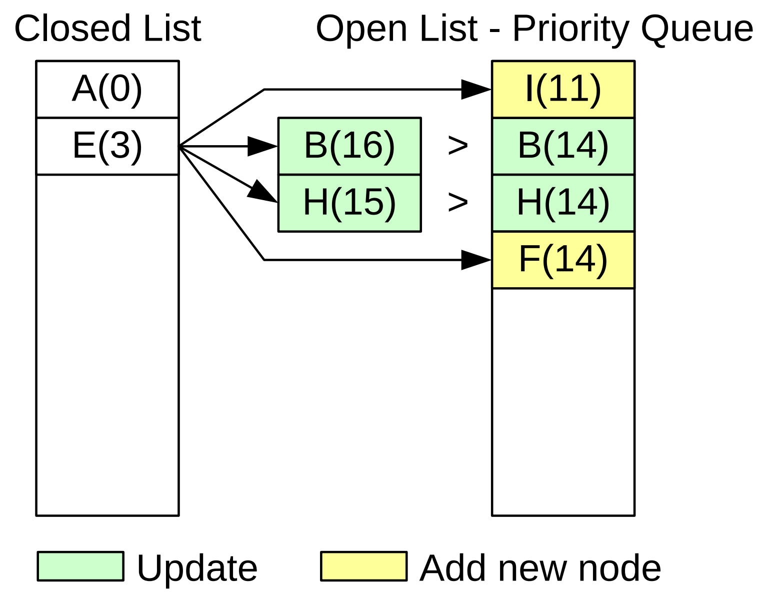
E(y) = E(f(n))
where f(n) = g(n) + h(n)
= 3 + 8 = 11

Heuristics

A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6
H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

A*: Example (3/6)





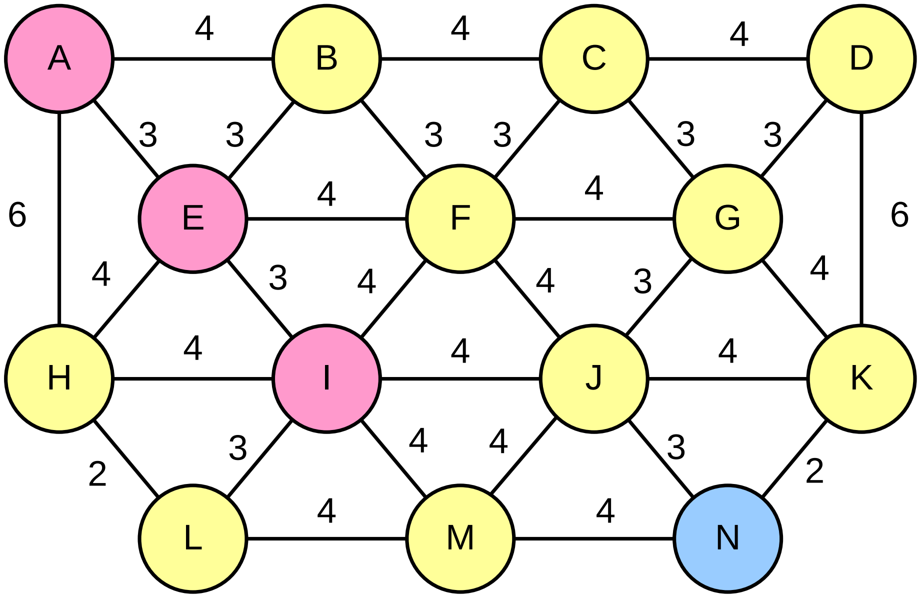
Heuristics

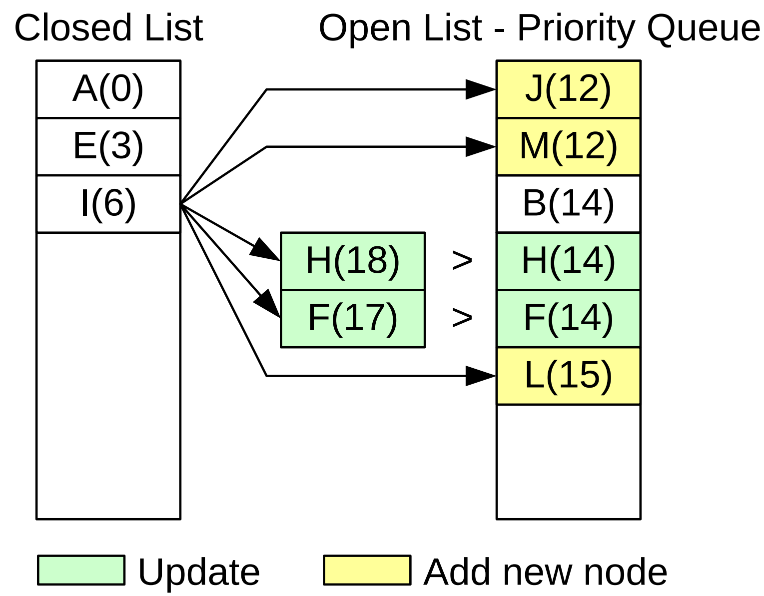
A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6

H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

Since A → B is smaller than A → E → B, the f-cost value of B in an open list needs not be updated

A*: Example (4/6)



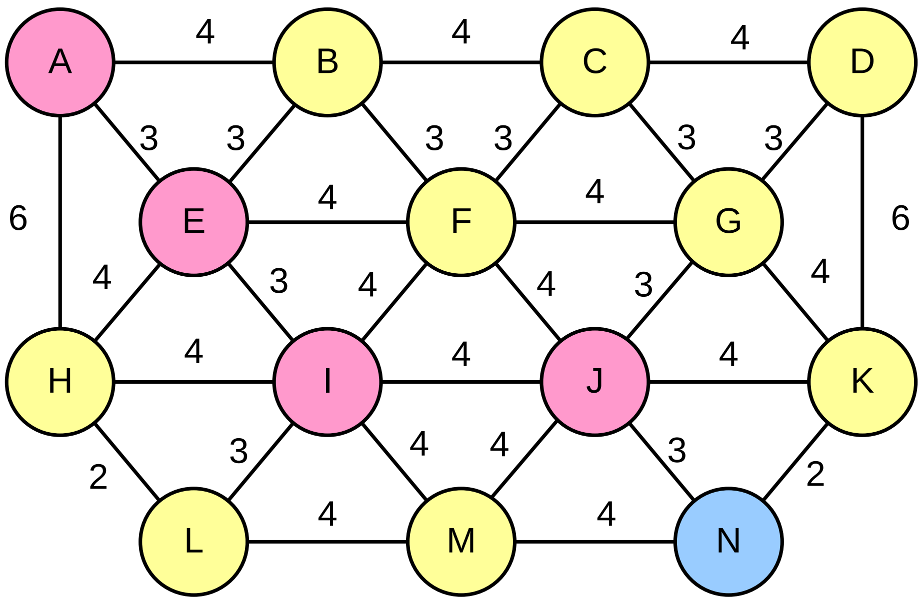


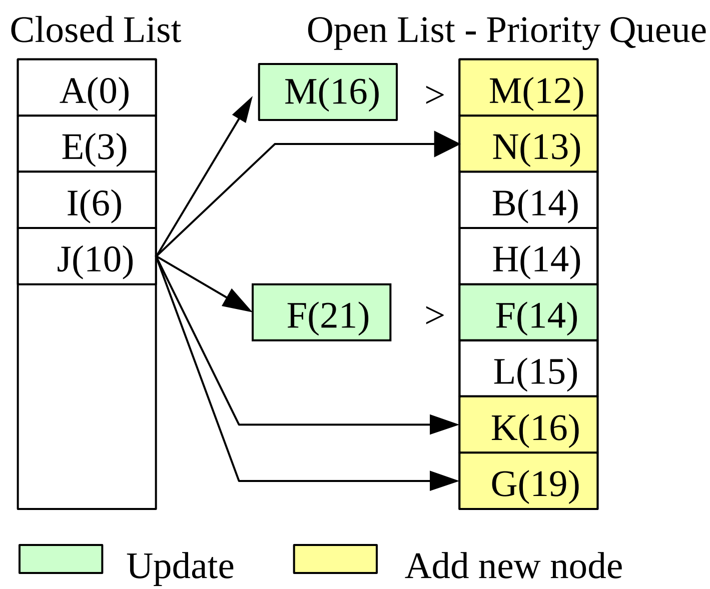
Heuristics

A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6

H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

A*: Example (5/6)



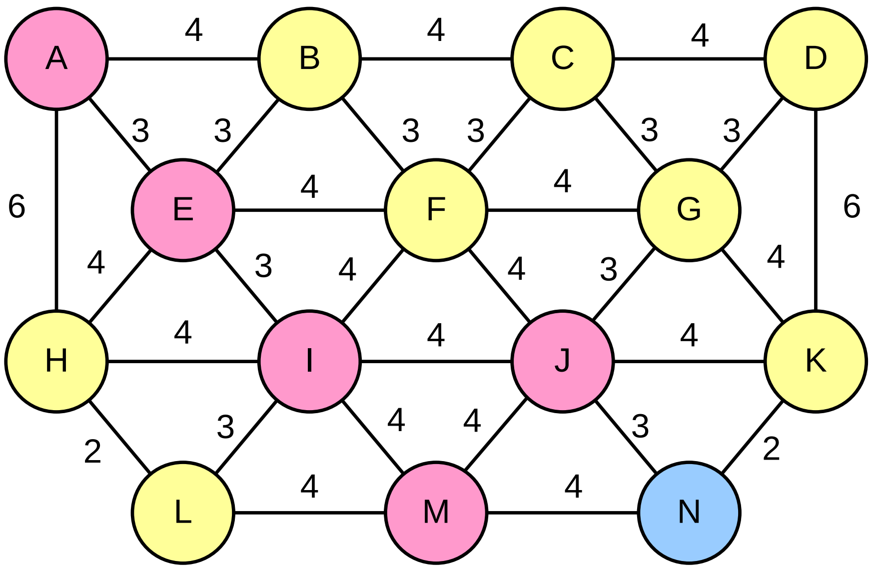


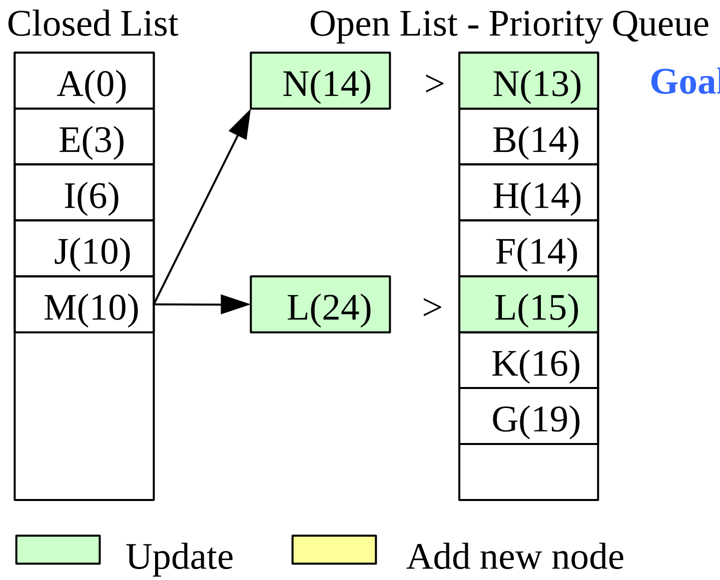
Heuristics

A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6

H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

A*: Example (6/6)





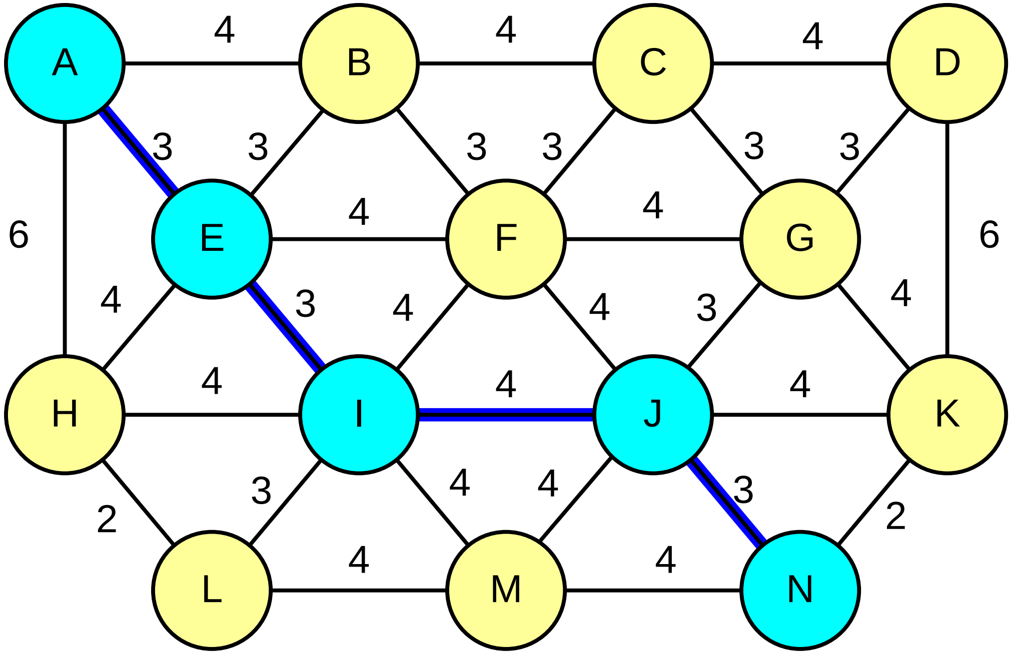
Heuristics

A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6

H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

Since the path to N from M is greater than that from J, **the optimal path to N is the one traversed from J**

A*: Example Result



Generate the path from the goal node back to the start node through the back-pointer attribute