AuSRoS

Australian School of Robotic Systems

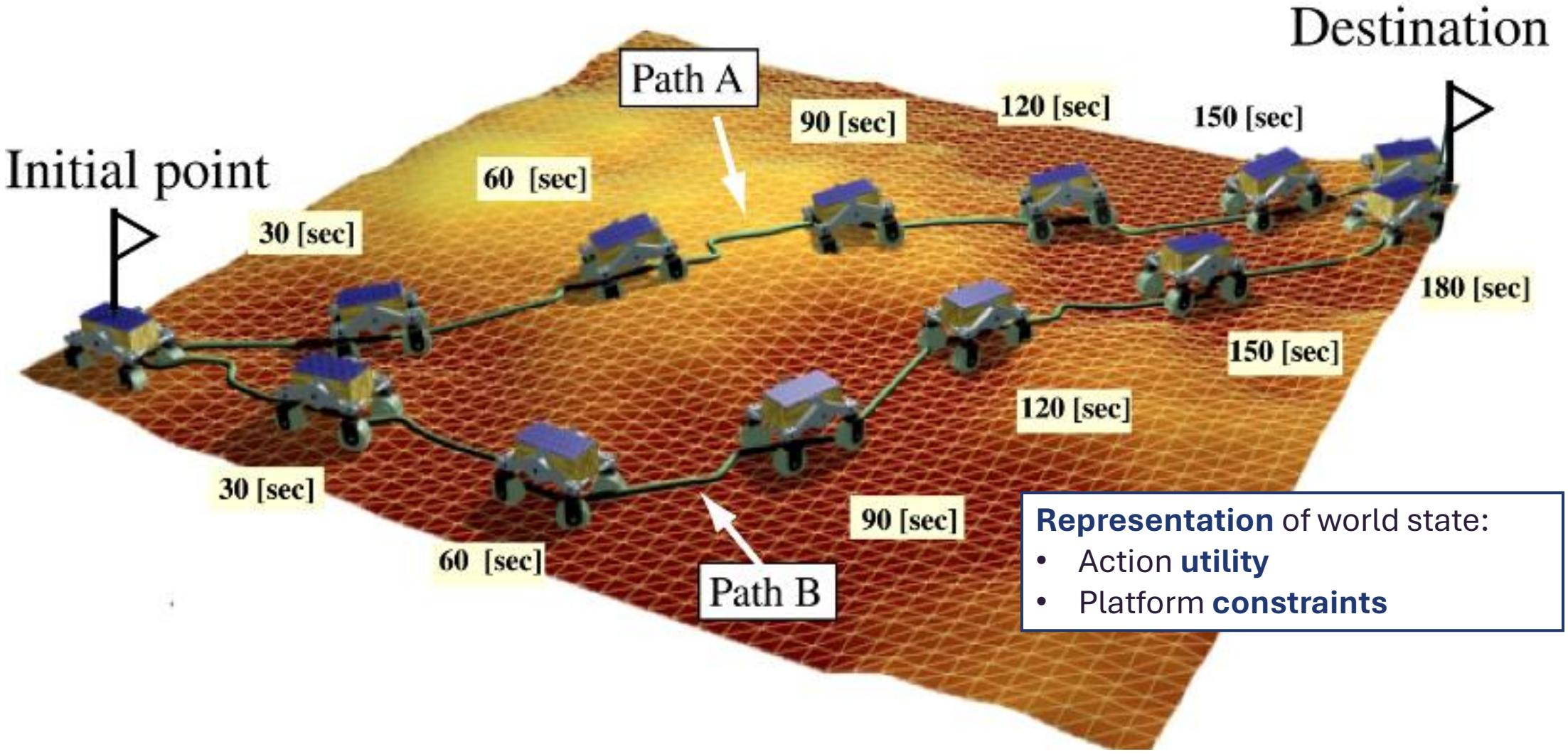
B1: Motion Planning Essentials

A/Prof. Jen Jen Chung

The University Of Queensland
AUSTRALIA

With slides adapted from Dr Nicholas Lawrance

What is Robot Motion Planning?

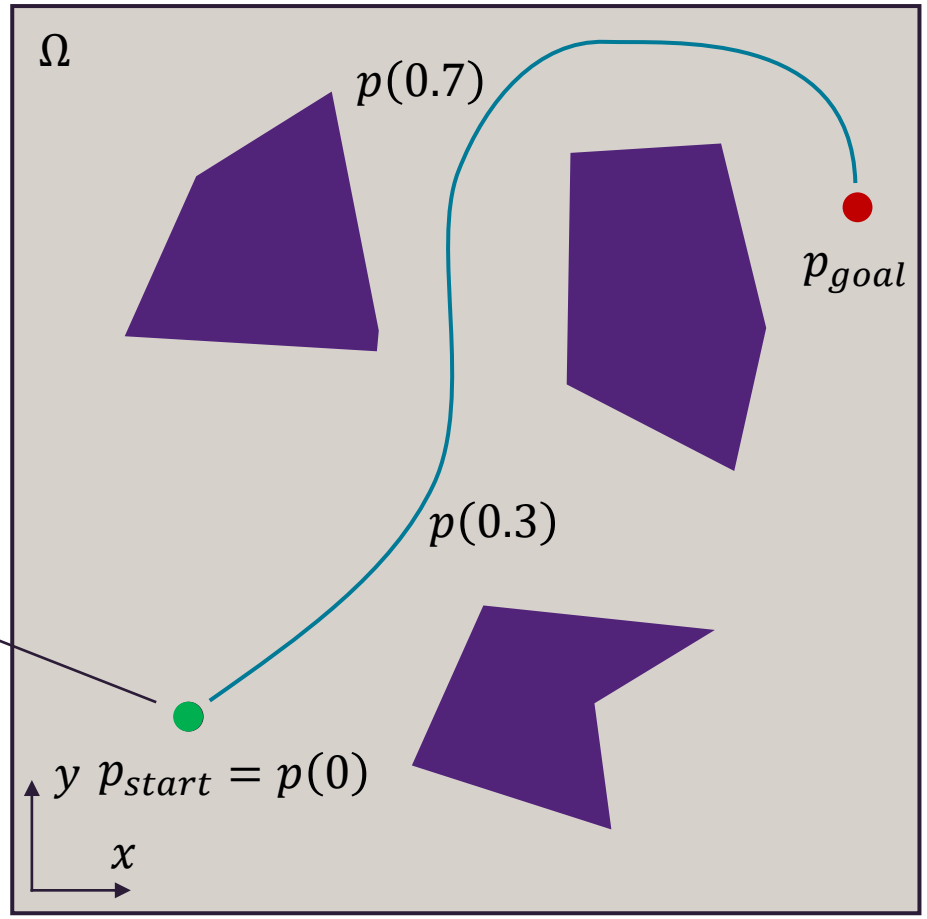


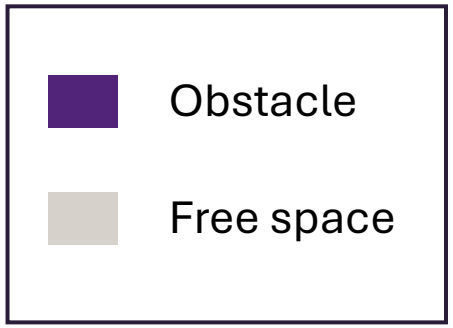
Representation of world state:

- Action **utility**
- Platform **constraints**

Motion Planning Representations

Workspace and Paths





$$\omega_{free} = \Omega - \omega_{obs}$$

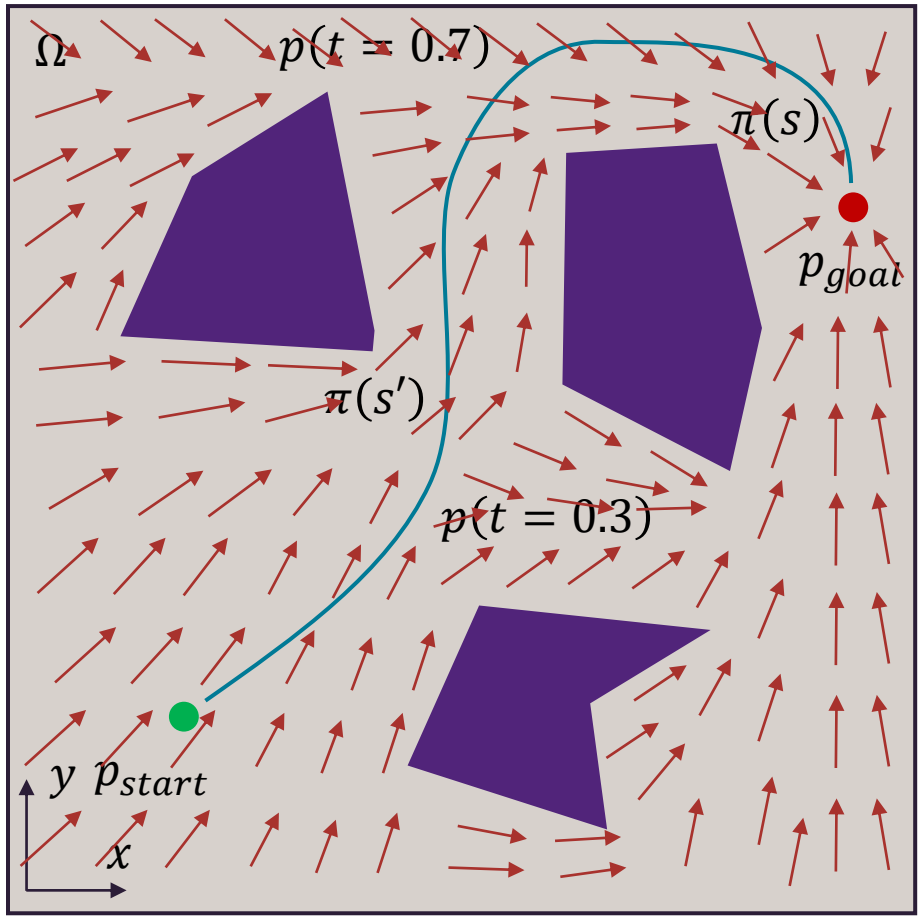
$$p: \ [0,1] \rightarrow \omega_{free}$$

$$p(t): \ \forall t \in [0,1], \ p(t) \in \omega_{free}$$

$$p(0) = p_{start}$$

$$p(1) = p_{goal}$$

Paths, Trajectories and Policies



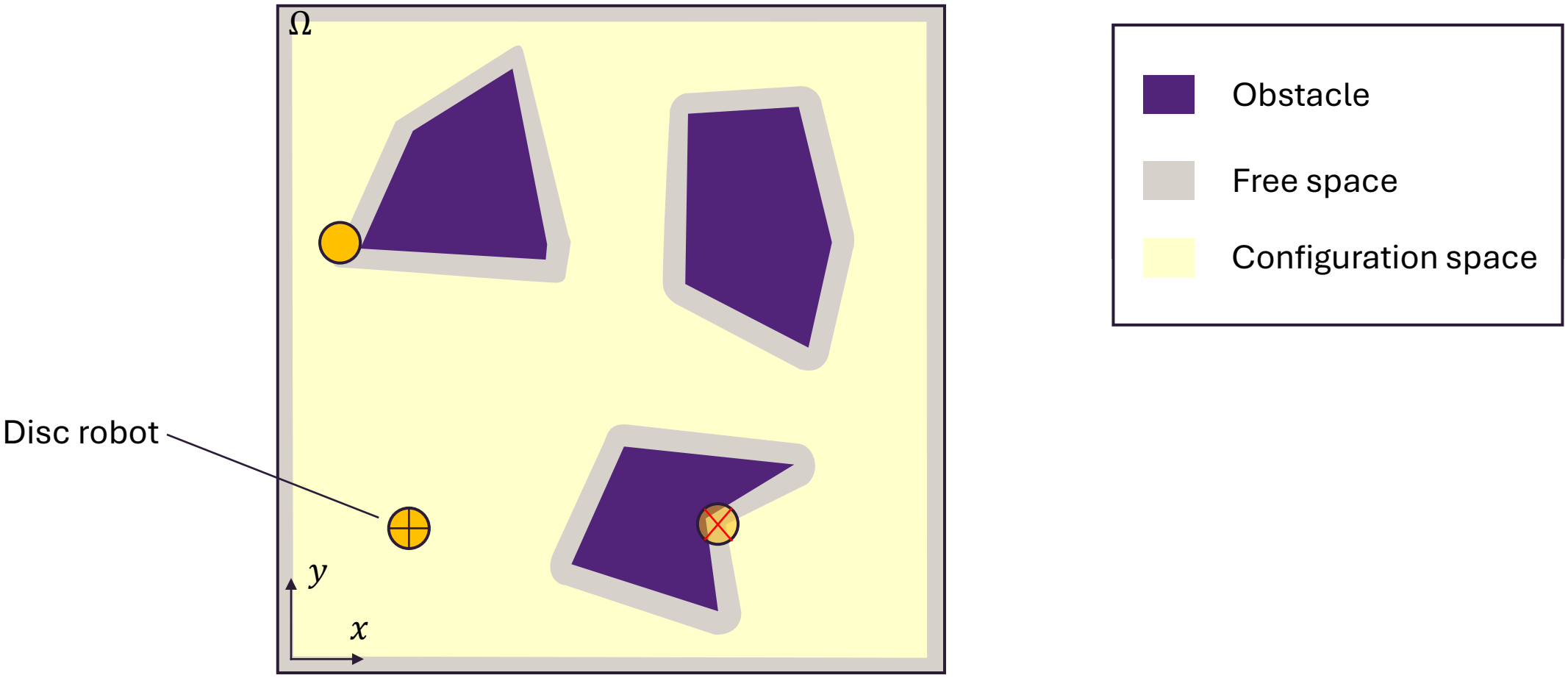
- **Path $\boldsymbol{p} \subset \boldsymbol{\Omega}$:** A set of points (geometric path) that the vehicle will travel along (no time consideration)
- **Trajectory $\boldsymbol{p(t)}: \boldsymbol{T} \rightarrow \boldsymbol{\Omega}$:** Path parameterised by ***time*** – important when considering dynamics constraints (e.g. turning radius as a function of speed)
- **Policy $\boldsymbol{\pi(s)}: \boldsymbol{S} \rightarrow \boldsymbol{A}$:** A function that maps (all valid) states to actions

Workspace vs. Configuration Space

- **Workspace** is often the representation of the world, possibly independent of the robot itself. Often describes some notion of reachability, free vs. occupied

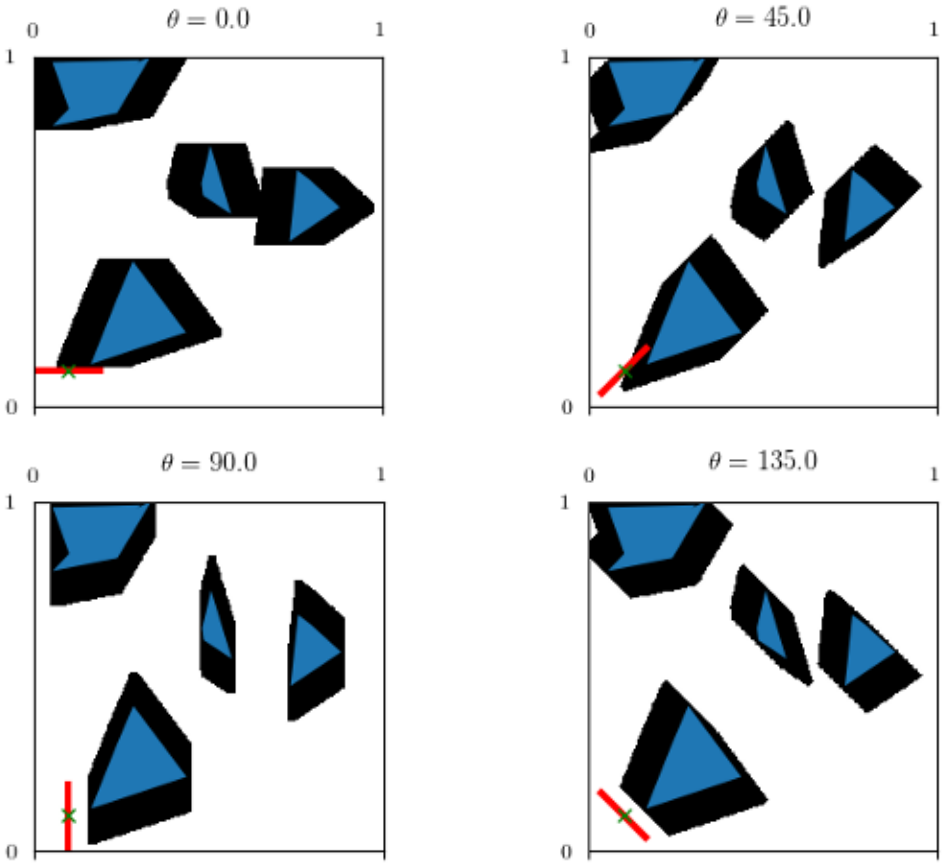
- **Configuration** space describes the full state of the robot in the world (actuator positions, orientation, etc.)

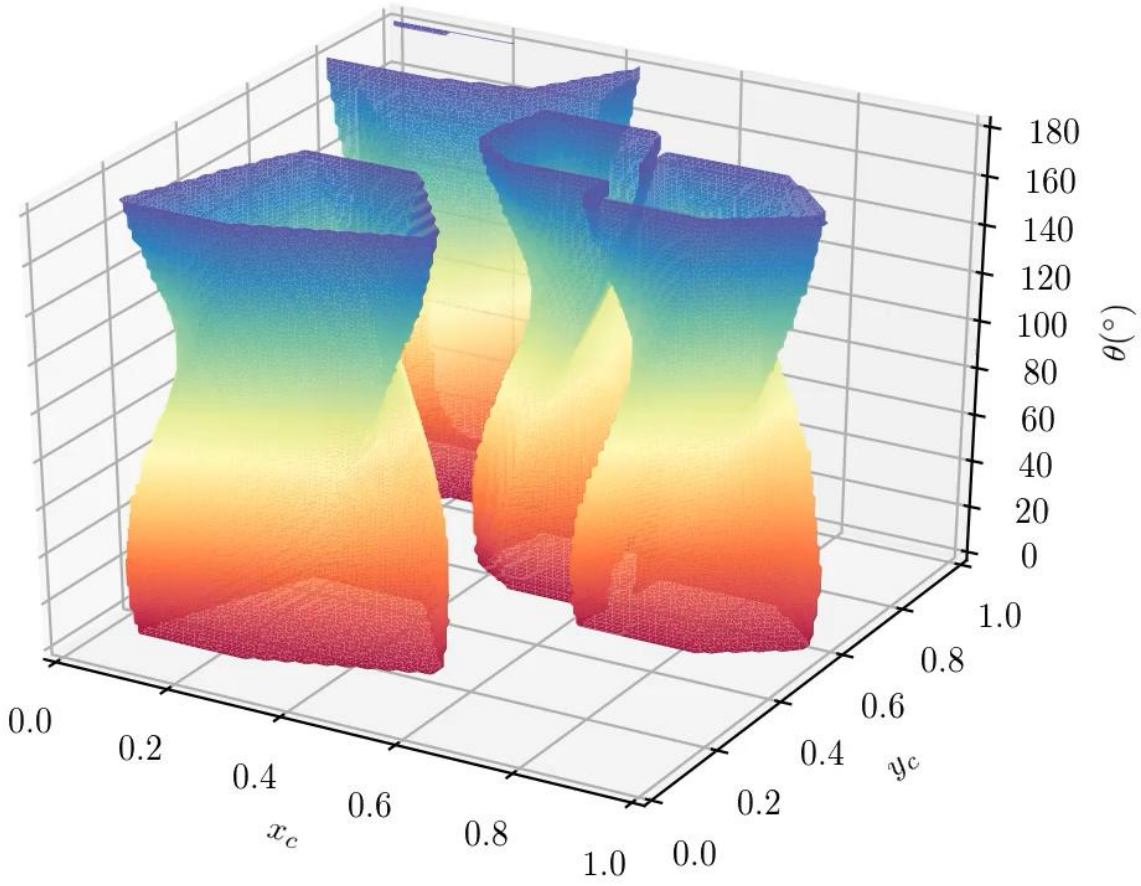
Configuration Space



Configuration Space

A robot without rotational symmetry (x, y, θ)





Configuration Space for Alternative Morphologies

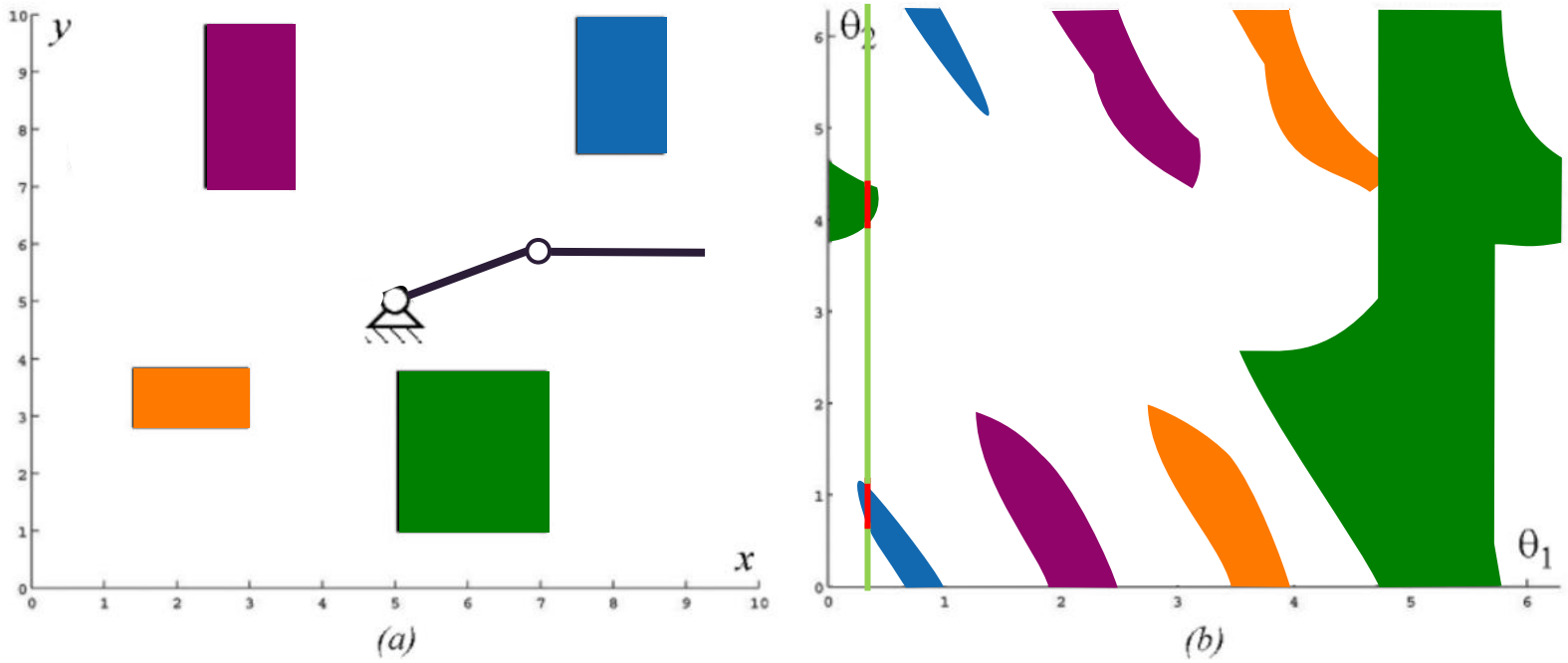
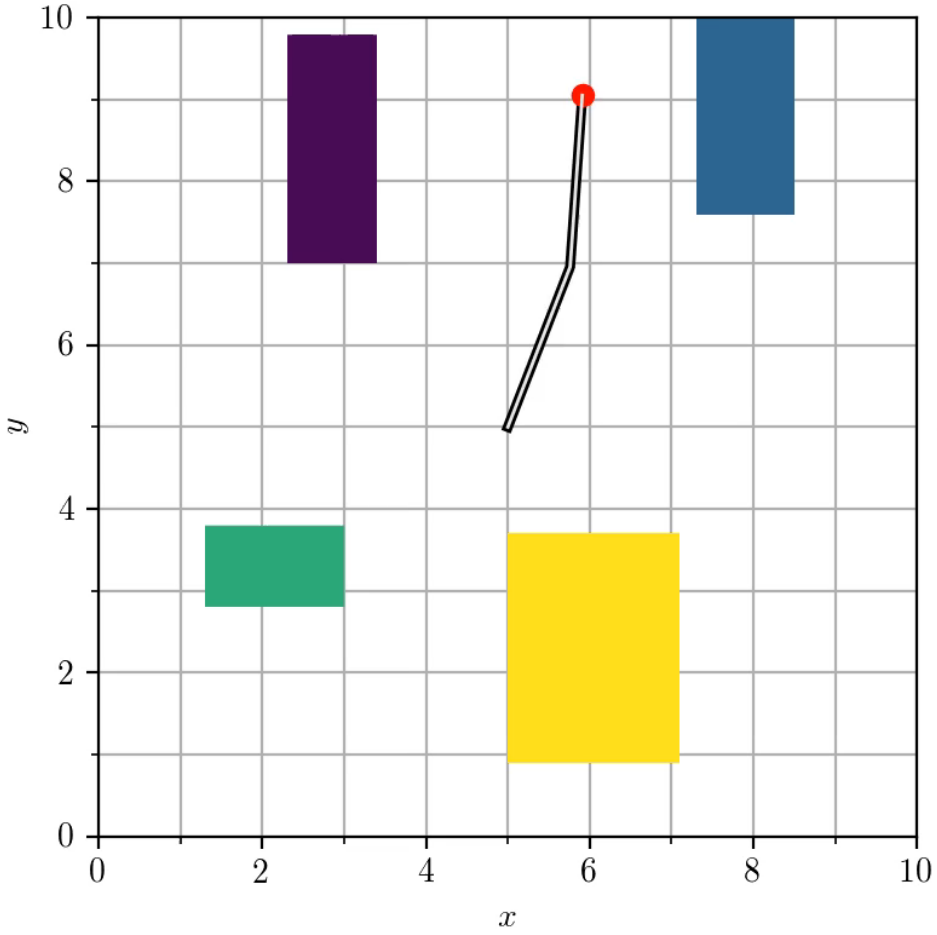
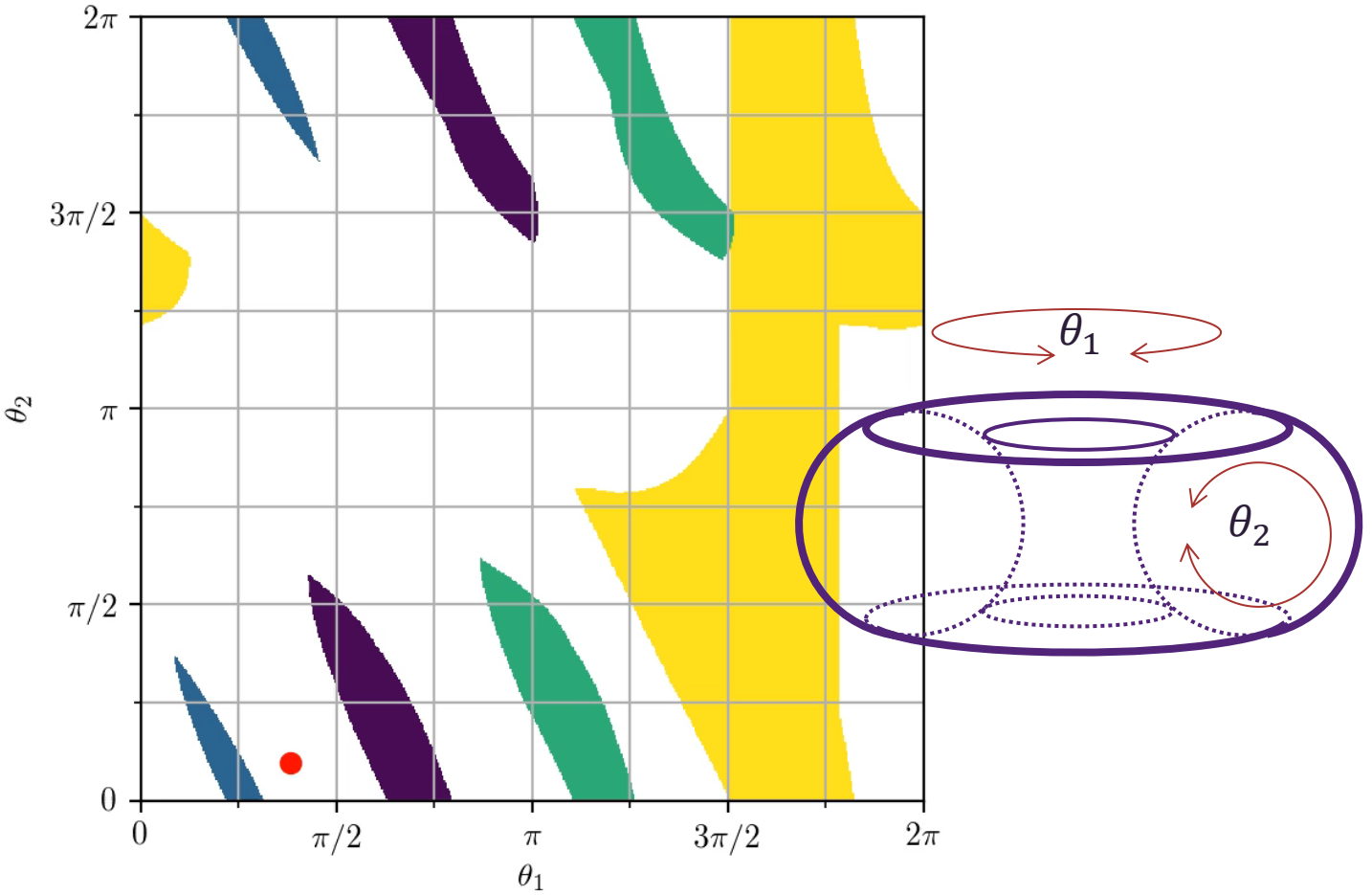


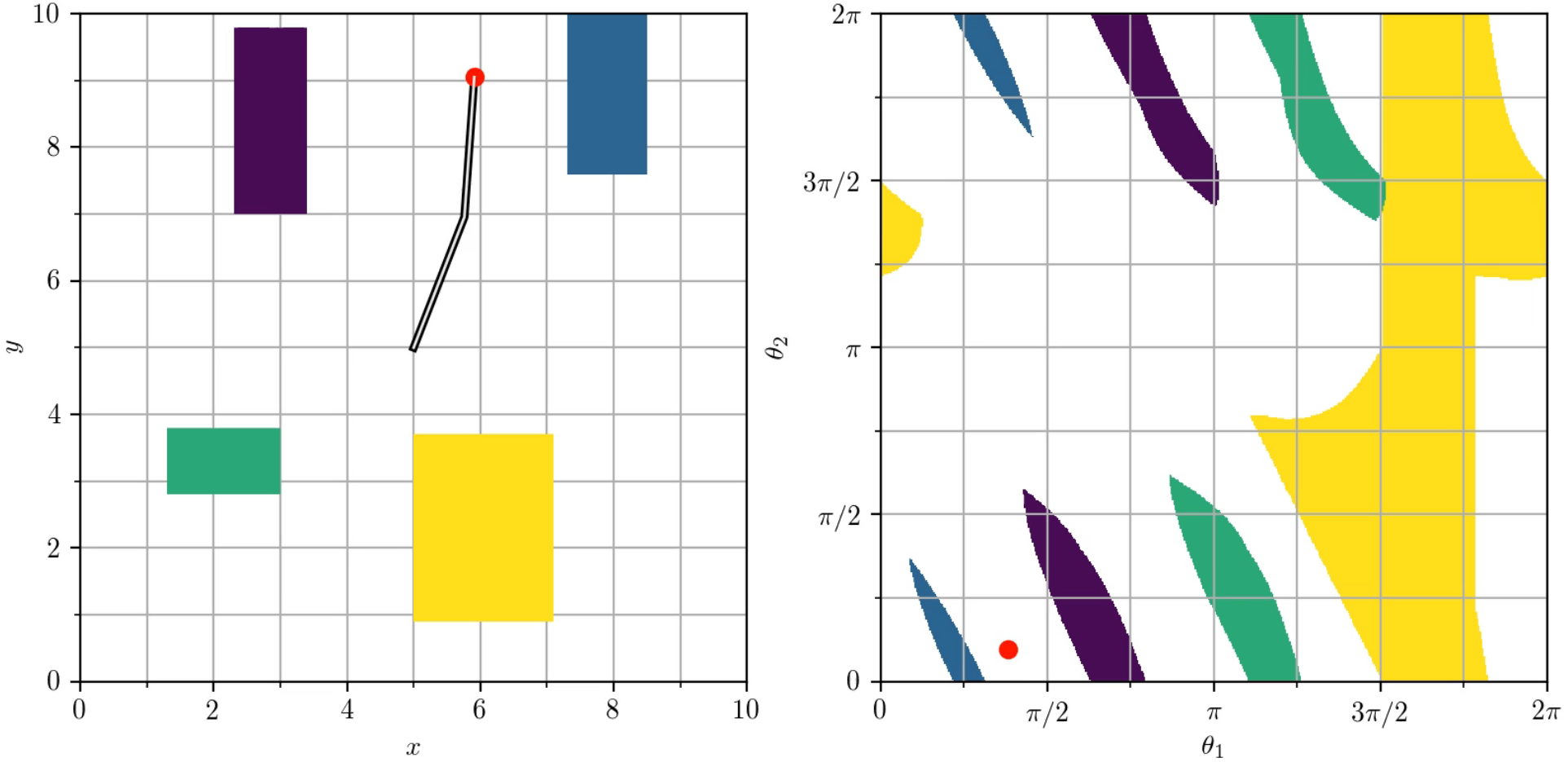
Figure 6.1
Physical space (a) and configuration space (b): (a) A two-link planar robot arm has to move from the configuration *start* to *end*. The motion is thereby constraint by the obstacles 1 to 4. (b) The corresponding configuration space shows the free space in joint coordinates (angle θ_1 and θ_2) and a path that achieves the goal.

Configuration Space for Alternative Morphologies

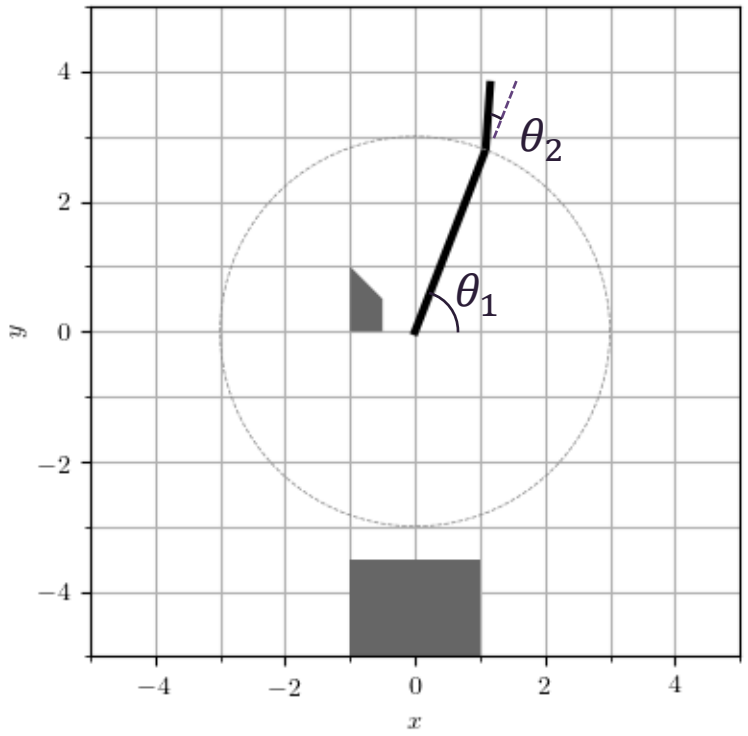




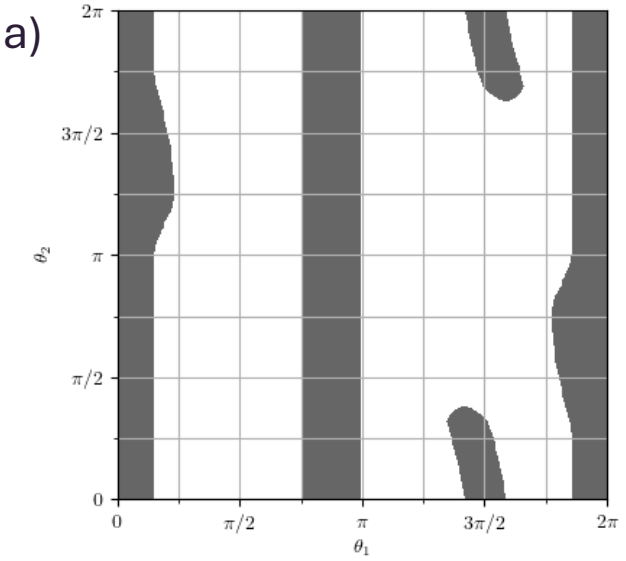
Configuration Space for Alternative Morphologies



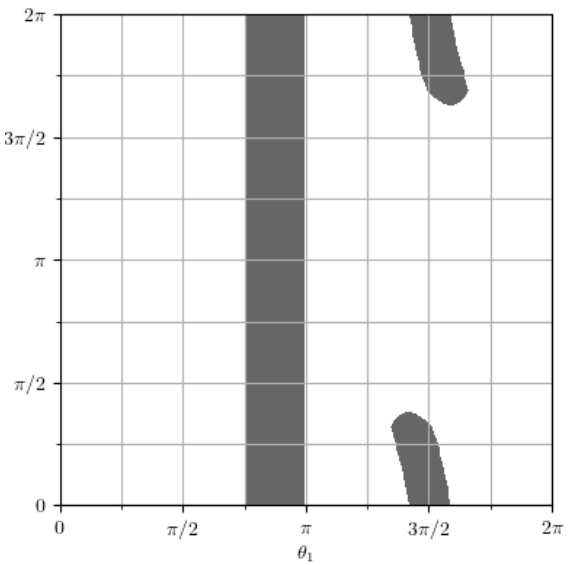
Which is the Matching Configuration Space?



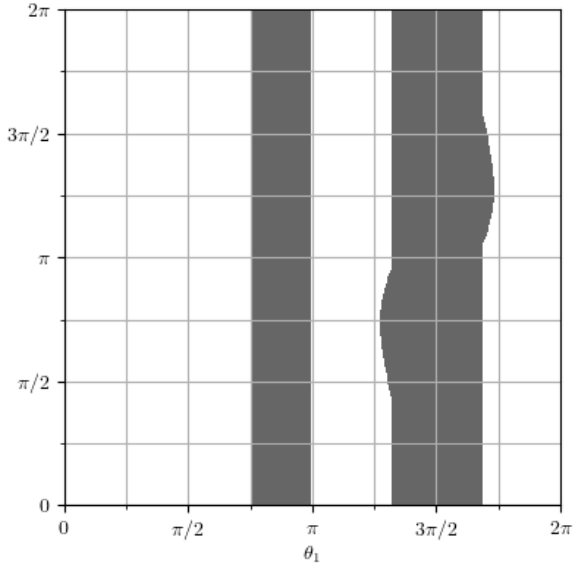
a)

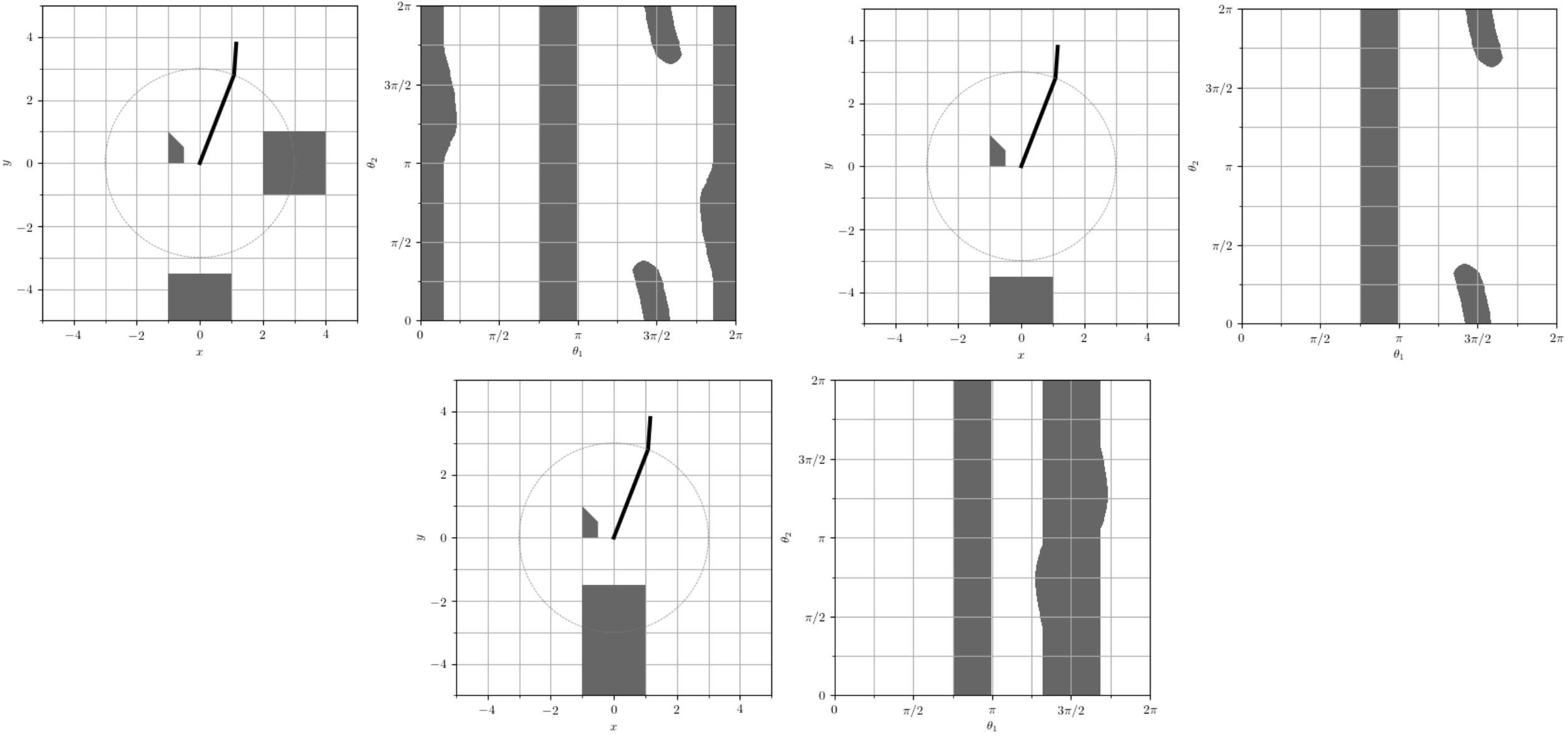


b)



c)




4
2
0
−2
−4
y
−4
−2
0
2
4
x
2π
3π/2
π
π/2
0
θ_2
0
π/2
π
3π/2
2π
θ_1

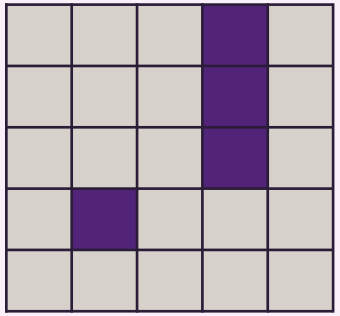
Why use Configuration Space?

- Paths in configuration space correspond to actual state changes of the robot
- Can be easier to solve collision checks and join nearby poses
- Allows a level of abstraction that means solution methods can solve a wider range of problems
- Sometimes helps with wraparound conditions (rotation joints)

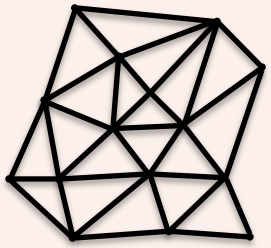
Continuous → Discrete Representations

Structured

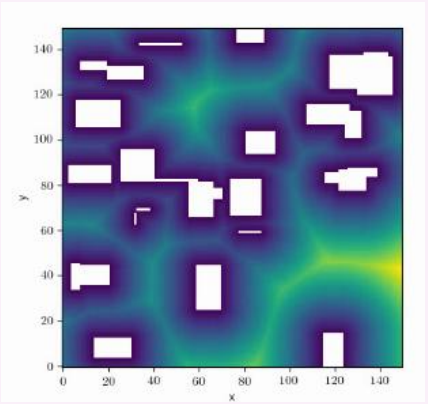
Occupancy grids

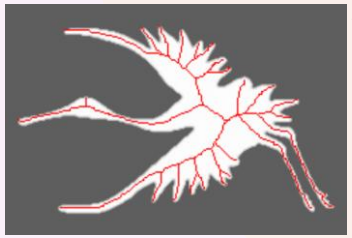


Graphs



Distance fields

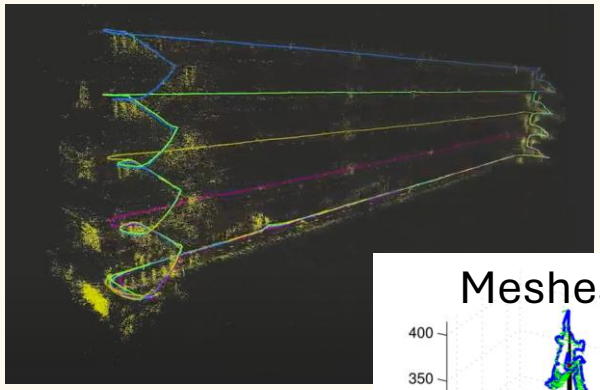




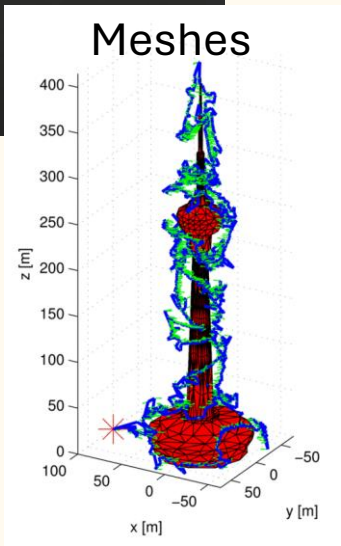
*

Unstructured

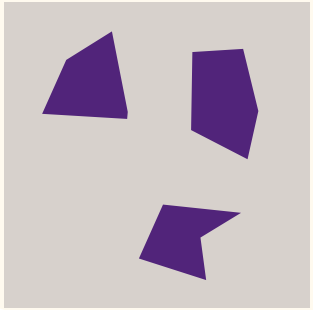
Point clouds



Meshes



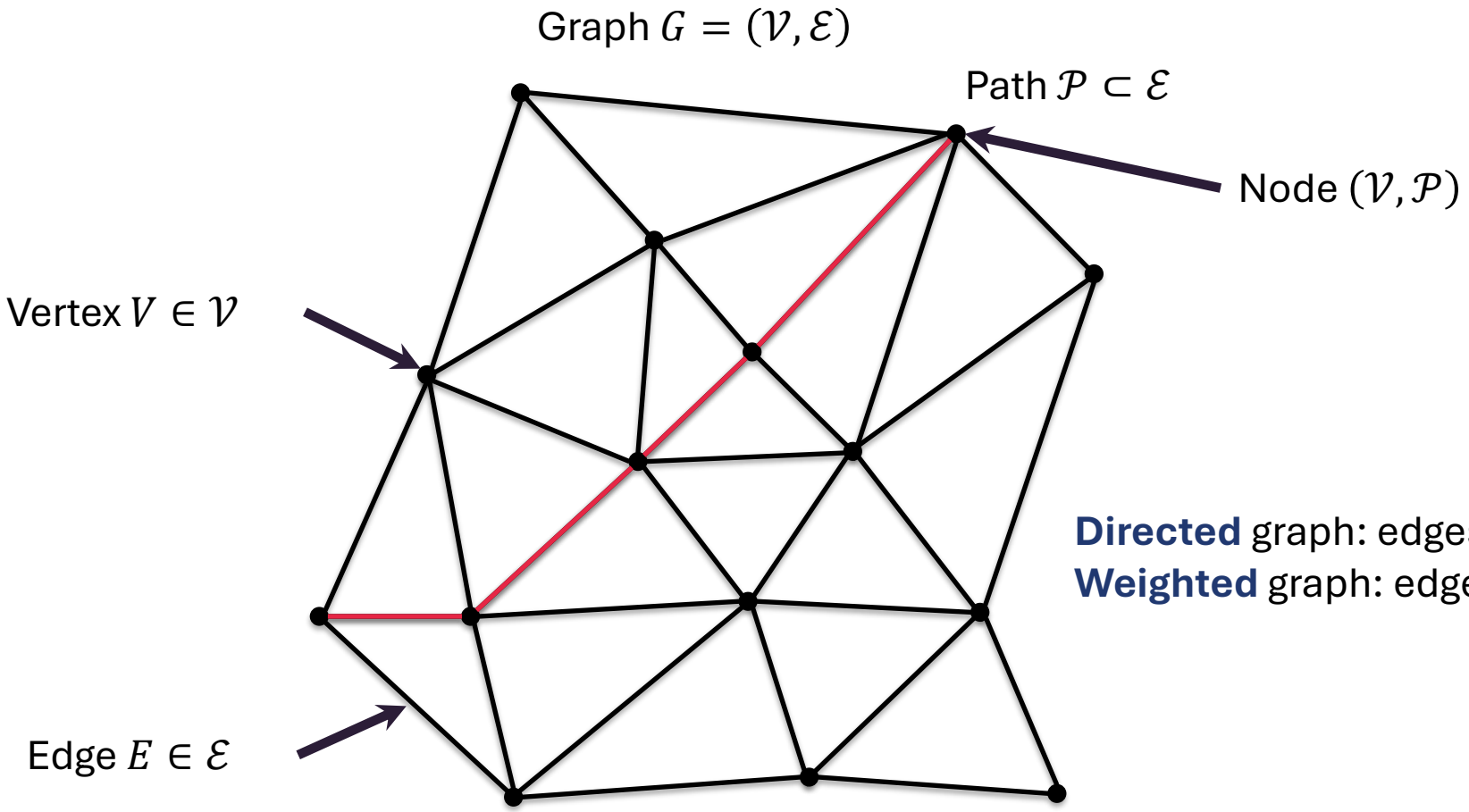
Exact



* Chang, Sukmoon. "Extracting skeletons from distance maps." *International Journal of Computer Science and Network Security* 7.7 (2007): 213-219.

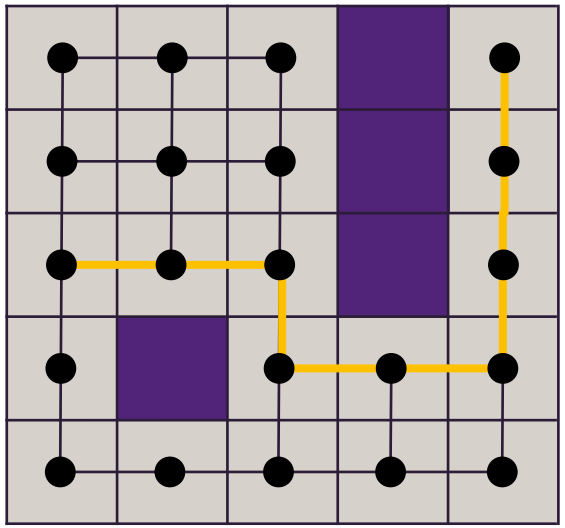
Graph Search Methods

Some Terminology

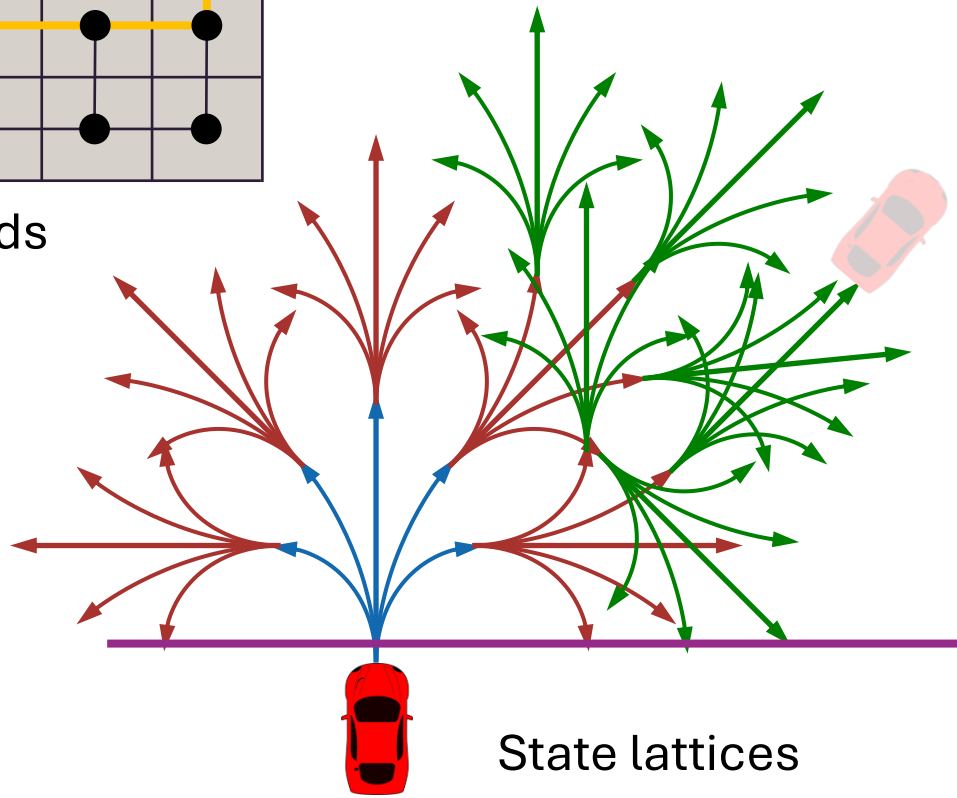


Directed graph: edges have direction
Weighted graph: edges have costs

Graph-based Representations

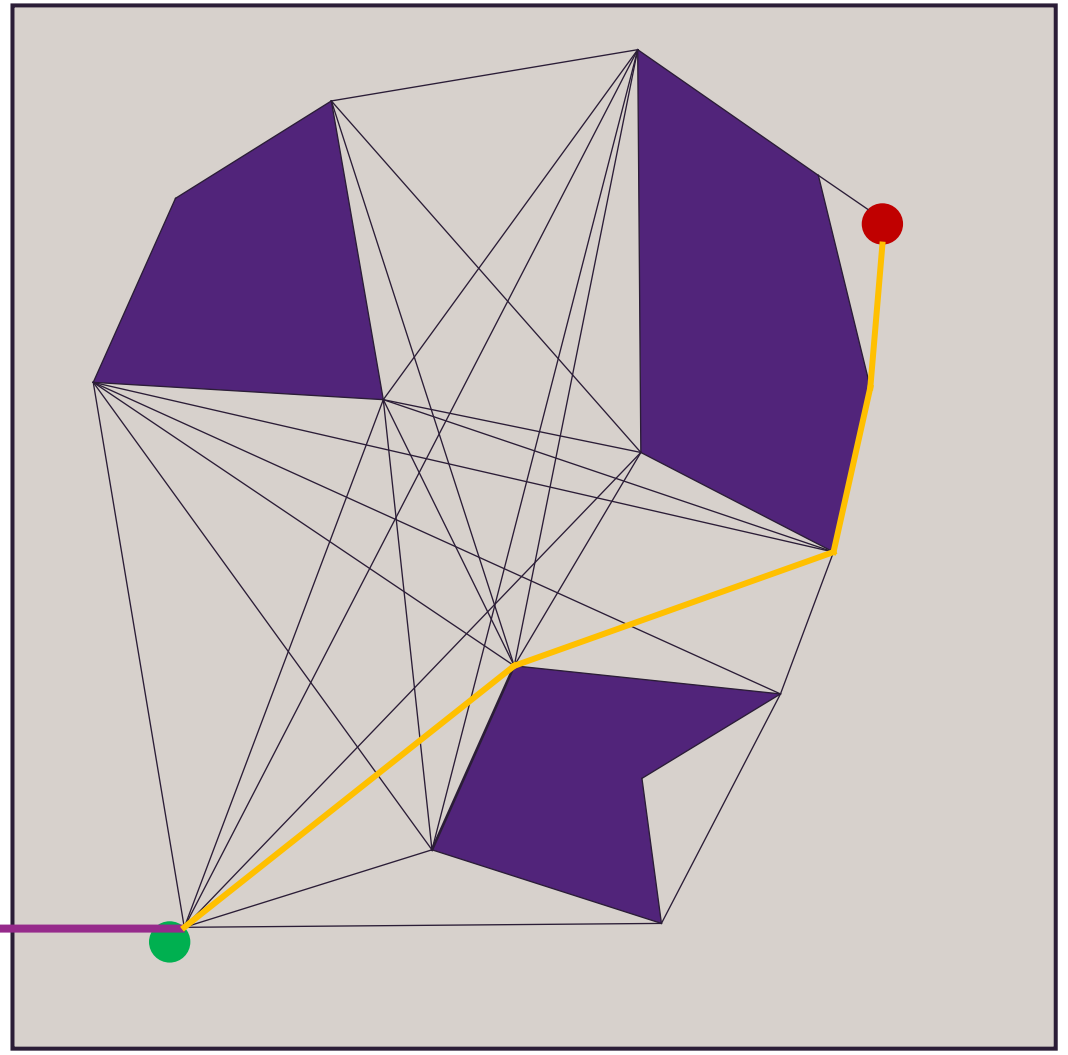


Grids



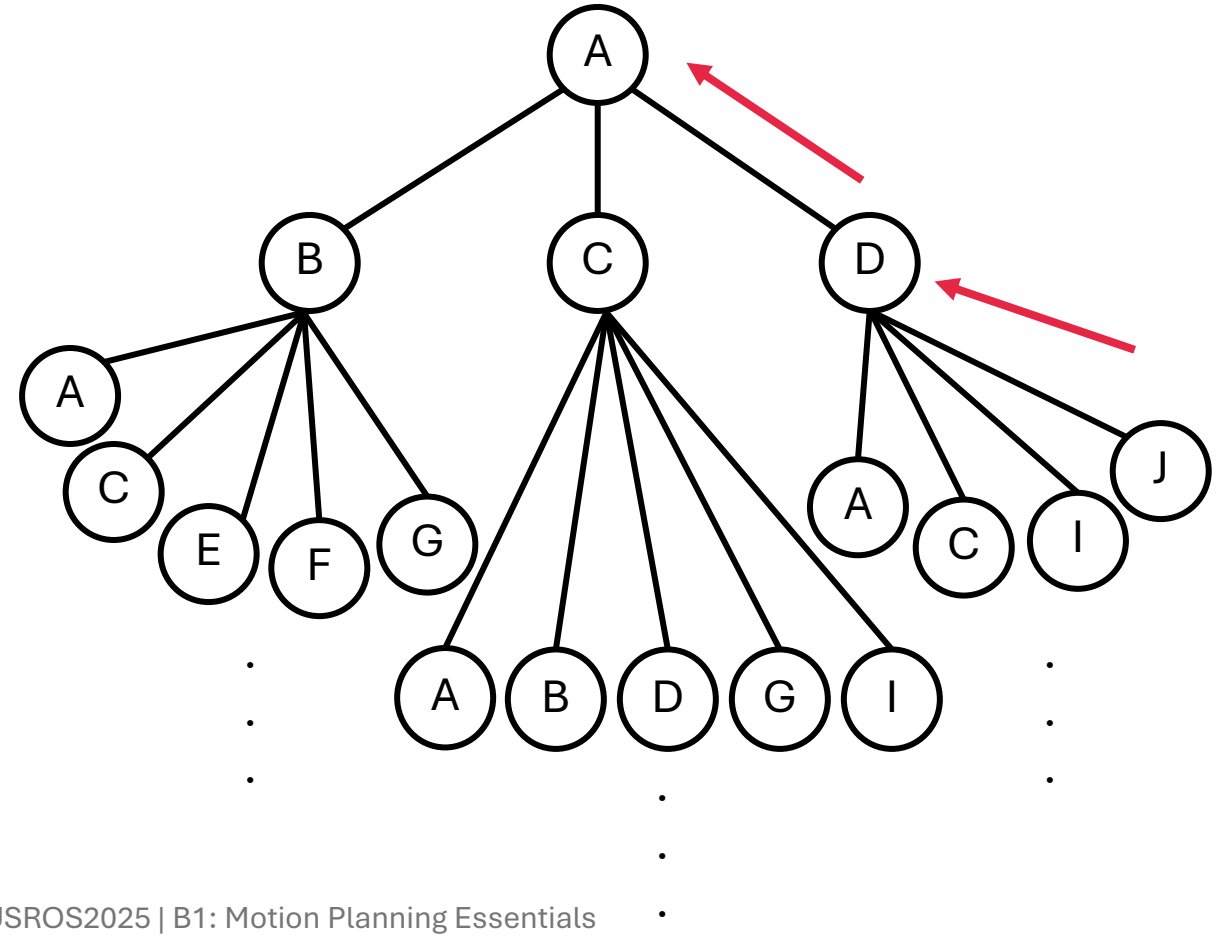
State lattices

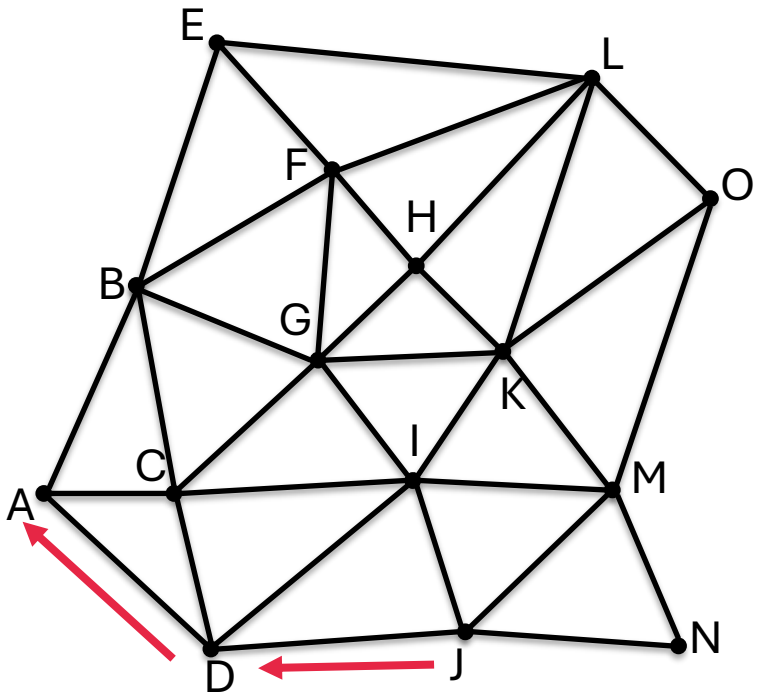
Visibility graphs



Graph Search via Search Tree Construction

- Edges may have non-uniform costs (generally only consider positive costs)
- Construct search tree from root node





Basics of Forward Search

- Start from the start, grow tree until you find a solution (path to goal)
- Expanding a node refers to adding children to the tree, pushing them onto the open set
- Try to expand as few tree nodes as possible

- **Open** set maintains a list of frontier (unexpanded) plans
  - Keeps track of what nodes to expand next
  - Often stored as a priority queue
  - For each node in the open list, we know of at least one path to it from the start

- **Closed** (visited) set keeps track of nodes that have been expanded
  - For each node in the closed list, we've already found the lowest-cost path to it from the start

Forward Search Algorithm

```
FORWARD_SEARCH
 1   Q.Insert(x_I) and mark x_I as visited
 2   while Q not empty do
 3       x ← Q.GetFirst()
 4       if x ∈ X_G
 5           return SUCCESS
 6       forall u ∈ U(x)
 7           x' ← f(x, u)
 8           if x' not visited
 9               Mark x' as visited
10               Q.Insert(x')
11           else
12               Resolve duplicate x'
13   return FAILURE
```

Figure 2.4: A general template for forward search.

LaValle, Steven M. *Planning algorithms*. Cambridge university press, 2006, p. 33

Dijkstra's Algorithm

- Published by Edsger Dijkstra in 1959

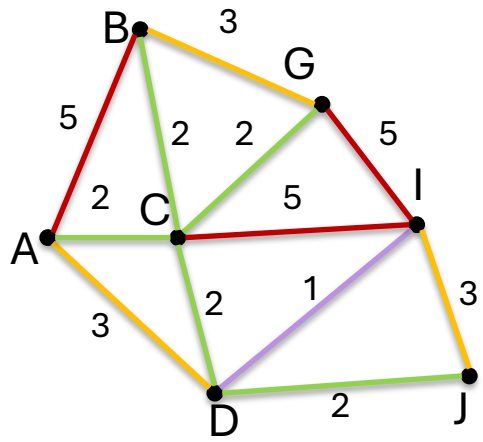
- Basic idea of expanding in order of closest to start (BFS with edge costs)
  - **Open** queue ordered according to currently known best cost to arrive

- One of the most commonly used routing algorithms in graph traversal problems

- Asymptotically the fastest known single-source shortest path algorithm for arbitrary directed graphs

Dijkstra's Algorithm Example

B(0)




```
FORWARD SEARCH
1   Q.Insert(x_I)
2   while Q not empty do
3       x ← Q.GetFirst(), R.Insert(x)
4       if x ∈ X_G
5           return SUCCESS
6       forall u ∈ U(x)
7           x' ← f(x, u)
8           if x' ∉ Q or R
9               Mark x' as visited
10              Q.Insert(x')
11          else
12              Resolve duplicate x'
13  return FAILURE
```

Our Dijkstra queue will be ordered by cost to arrive:

- push (*Q.Insert*) by cost
- pop (*Q.GetFirst*) from the front, and add it to the closed list

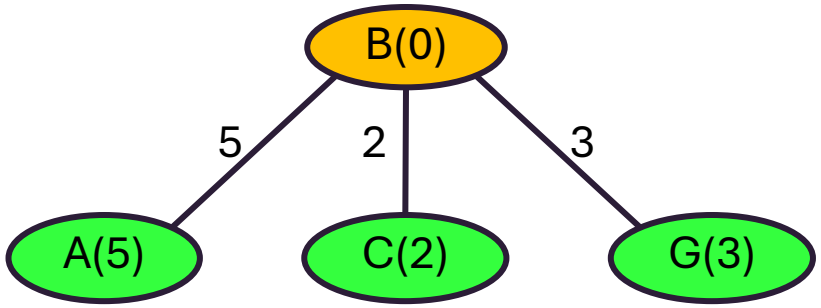
Open (*Q*):

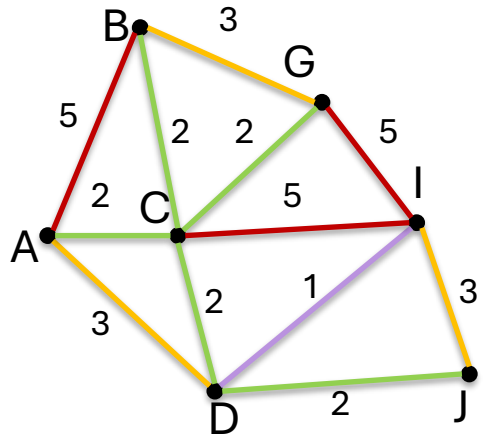
Node	Cost	Parent

Closed (*R*):

Node	Cost	Parent

Dijkstra's Algorithm Example






```
FORWARD_SEARCH
1   Q.Insert(x_I)
2   while Q not empty do
3       x ← Q.GetFirst(), R.Insert(x)
4       if x ∈ X_G
5           return SUCCESS
6       forall u ∈ U(x)
7           x' ← f(x, u)
8           if x' ∉ Q or R
9               Mark x' as visited
10              Q.Insert(x')
11          else
12              Resolve duplicate x'
13  return FAILURE
```

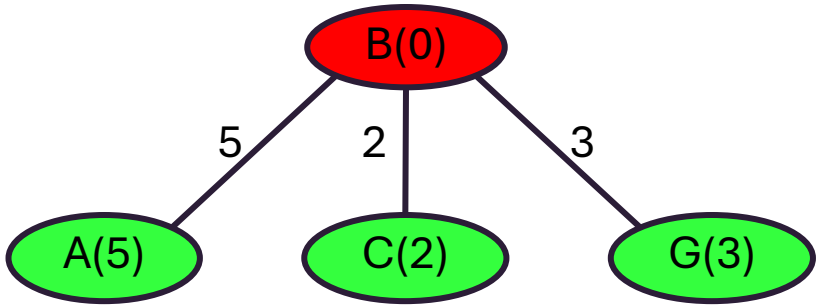
Open (Q):

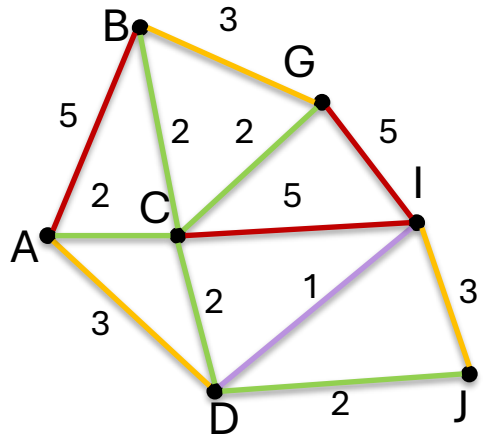
Node	Cost	Parent
C	2	B
G	3	B
A	5	B

Closed (R):

Node	Cost	Parent
B	0	

Dijkstra's Algorithm Example






```
FORWARD_SEARCH
1   Q.Insert(x_I)
2   while Q not empty do
3       x ← Q.GetFirst(), R.Insert(x)
4       if x ∈ X_G
5           return SUCCESS
6       forall u ∈ U(x)
7           x' ← f(x, u)
8           if x' ∉ Q or R
9               Mark x' as visited
10              Q.Insert(x')
11          else
12              Resolve duplicate x'
13  return FAILURE
```

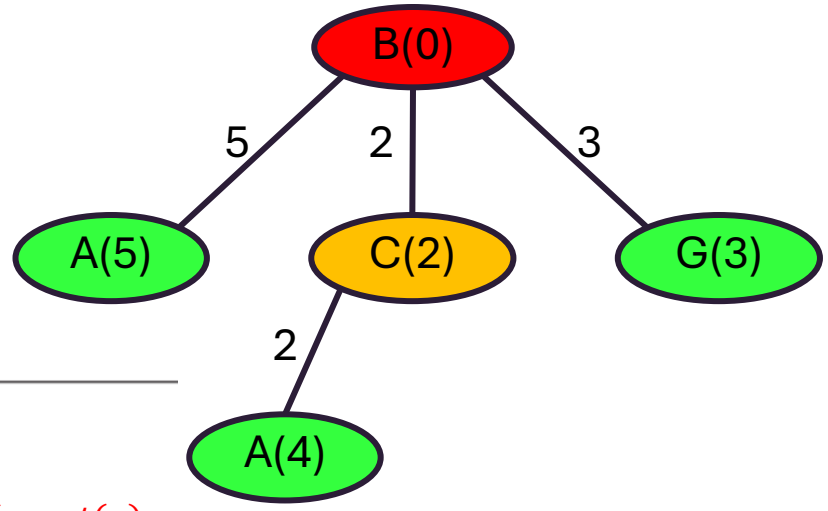
Open (Q):

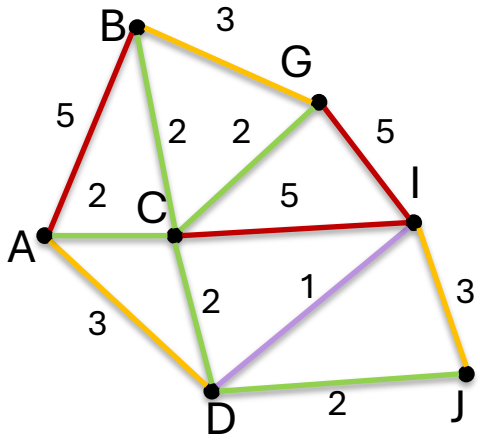
Node	Cost	Parent
G	3	B
A	5	B

Closed (R):

Node	Cost	Parent
B	0	
C	2	B

Dijkstra's Algorithm Example






```
FORWARD_SEARCH
1   Q.Insert(x_I)
2   while Q not empty do
3       x ← Q.GetFirst(), R.Insert(x)
4       if x ∈ X_G
5           return SUCCESS
6       forall u ∈ U(x)
7           x' ← f(x, u)
8           if x' ∉ Q or R
9               Mark x' as visited
10              Q.Insert(x')
11          else
12              Resolve duplicate x'
13  return FAILURE
```

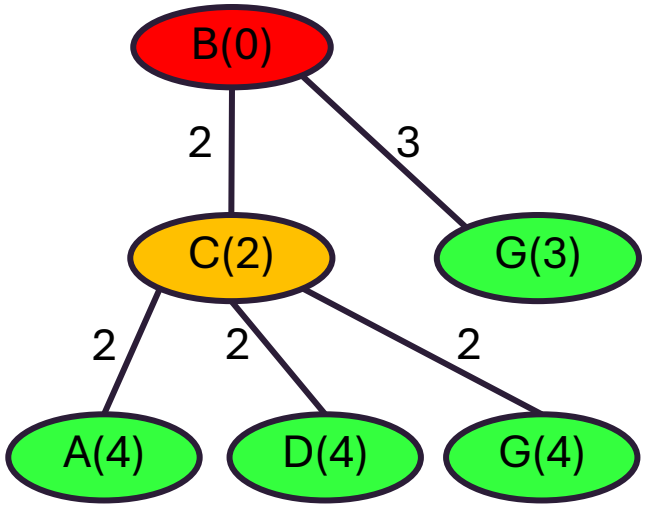
Open (Q):

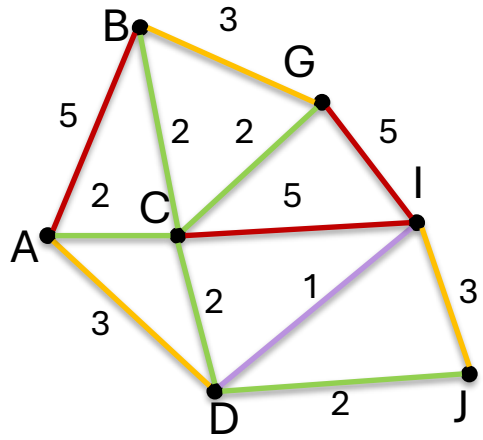
Node	Cost	Parent
G	3	B
A	4	C

Closed (R):

Node	Cost	Parent
B	0	
C	2	B

Dijkstra's Algorithm Example






```
FORWARD_SEARCH
1   Q.Insert(x_I)
2   while Q not empty do
3       x ← Q.GetFirst(), R.Insert(x)
4       if x ∈ X_G
5           return SUCCESS
6       forall u ∈ U(x)
7           x' ← f(x, u)
8           if x' ∉ Q or R
9               Mark x' as visited
10              Q.Insert(x')
11          else
12              Resolve duplicate x'
13  return FAILURE
```

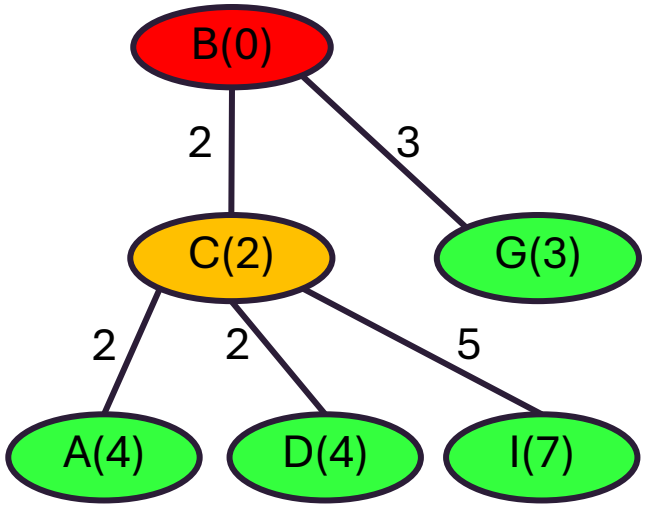
Open (Q):

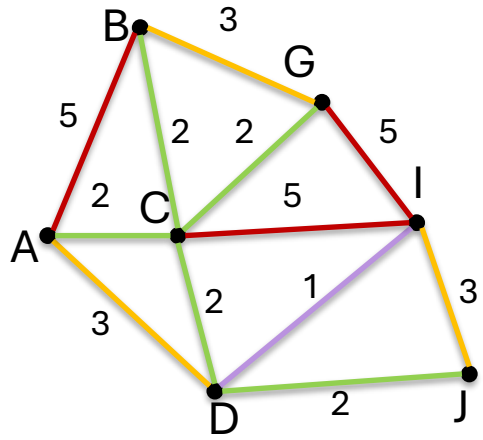
Node	Cost	Parent
G	3	B
A	4	C
D	4	C

Closed (R):

Node	Cost	Parent
B	0	
C	2	B

Dijkstra's Algorithm Example






```
FORWARD_SEARCH
1   Q.Insert(x_I)
2   while Q not empty do
3       x ← Q.GetFirst(), R.Insert(x)
4       if x ∈ X_G
5           return SUCCESS
6       forall u ∈ U(x)
7           x' ← f(x, u)
8           if x' ∉ Q or R
9               Mark x' as visited
10              Q.Insert(x')
11          else
12              Resolve duplicate x'
13  return FAILURE
```

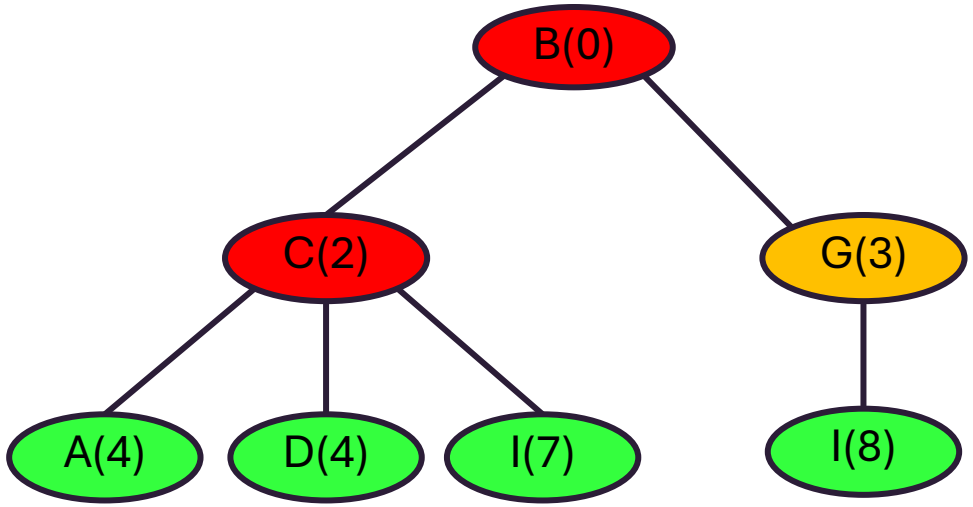
Open (Q):

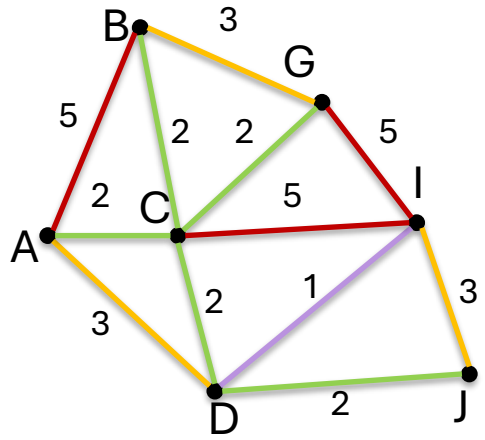
Node	Cost	Parent
G	3	B
A	4	C
D	4	C
I	7	C

Closed (R):

Node	Cost	Parent
B	0	
C	2	B

Dijkstra's Algorithm Example





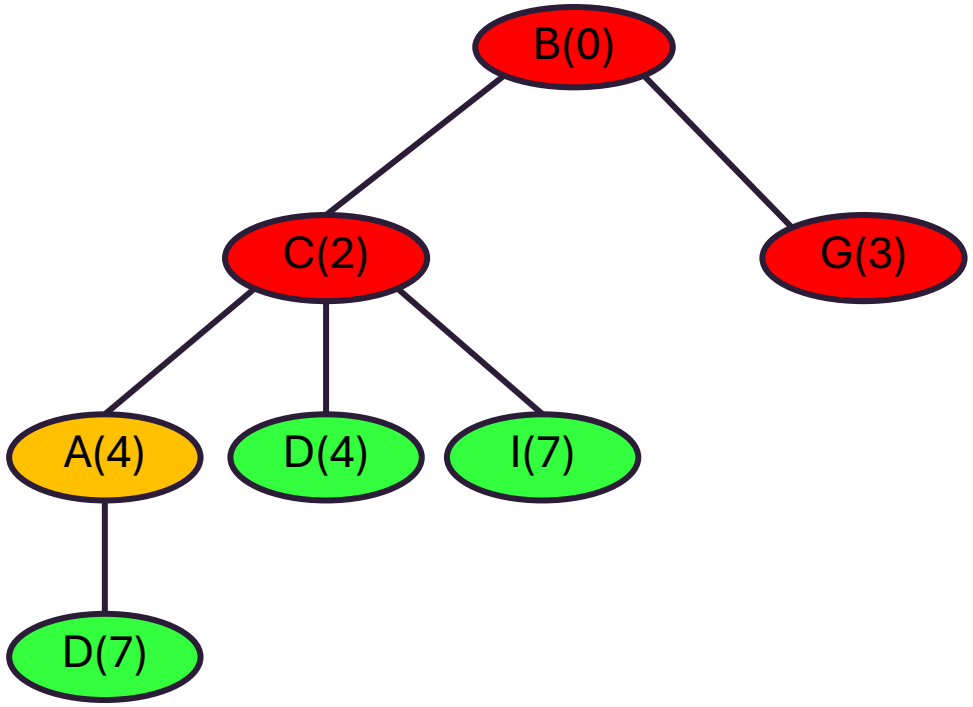
Open (Q):

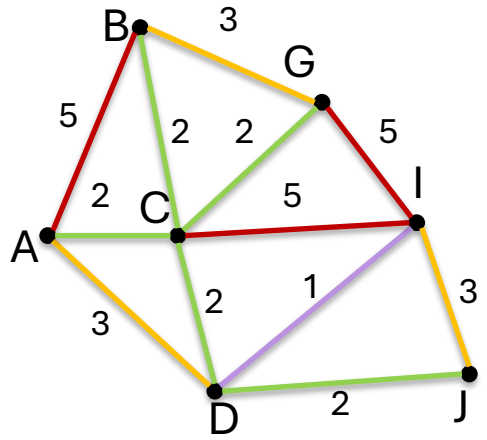
Node	Cost	Parent
A	4	C
D	4	C
I	7	C

Closed (R):

Node	Cost	Parent
B	0	
C	2	B
G	3	B

Dijkstra's Algorithm Example





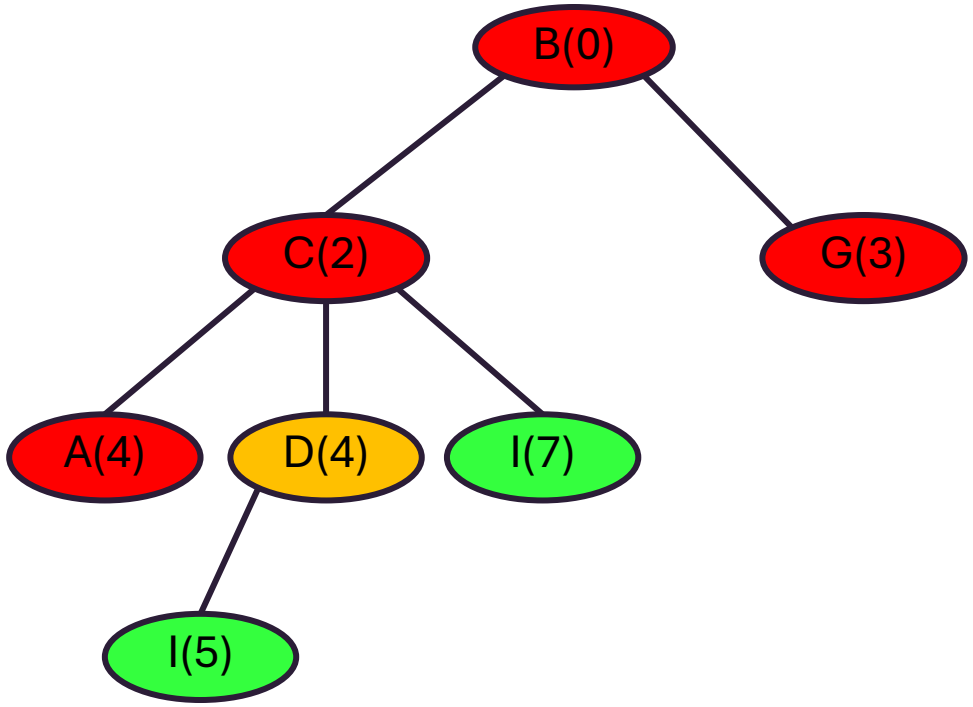
Open (Q):

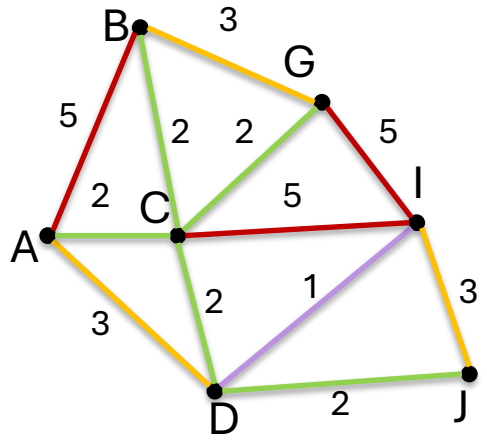
Node	Cost	Parent
D	4	C
I	7	C

Closed (R):

Node	Cost	Parent
B	0	
C	2	B
G	3	B
A	4	C

Dijkstra's Algorithm Example





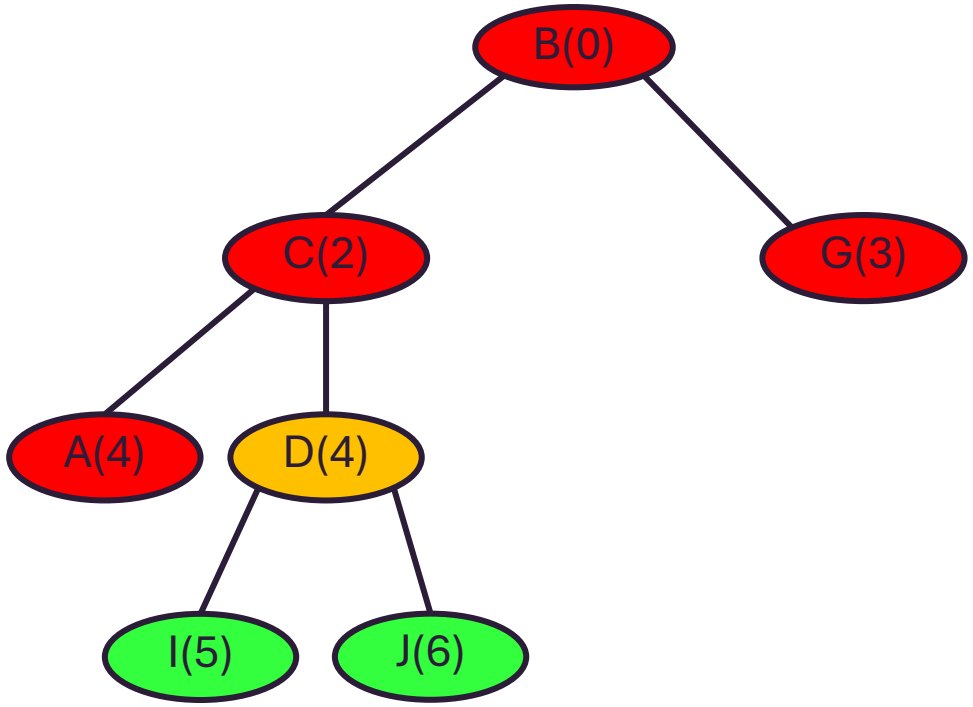
Open (Q):

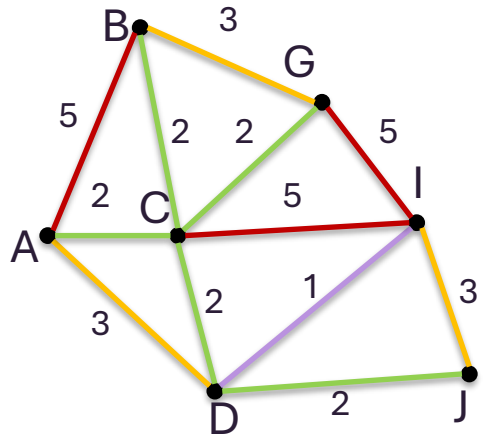
Node	Cost	Parent
I	5	D
I	7	C

Closed (R):

Node	Cost	Parent
B	0	
C	2	B
G	3	B
A	4	C
D	4	C

Dijkstra's Algorithm Example





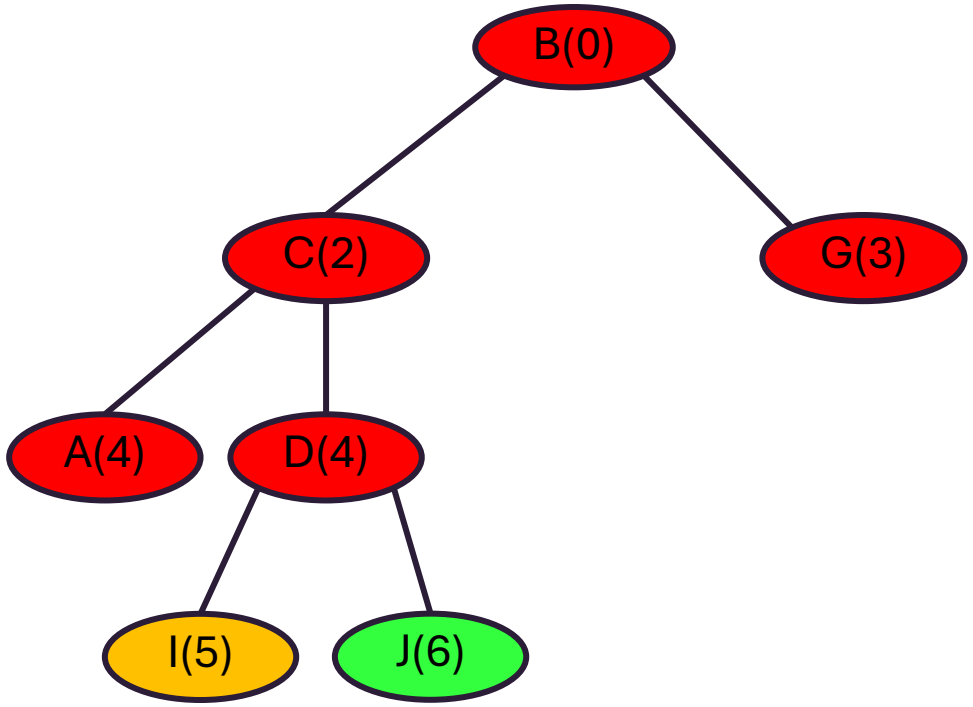
Open (Q):

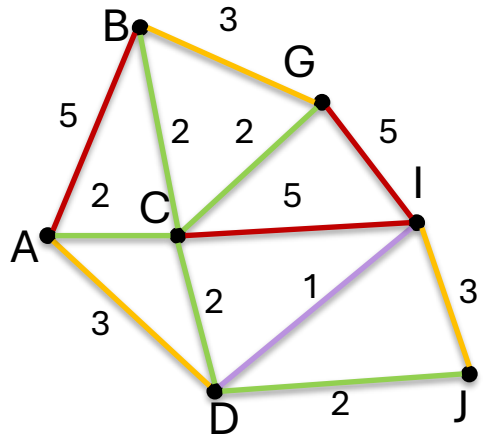
Node	Cost	Parent
I	5	D
J	6	D

Closed (R):

Node	Cost	Parent
B	0	
C	2	B
G	3	B
A	4	C
D	4	C

Dijkstra's Algorithm Example





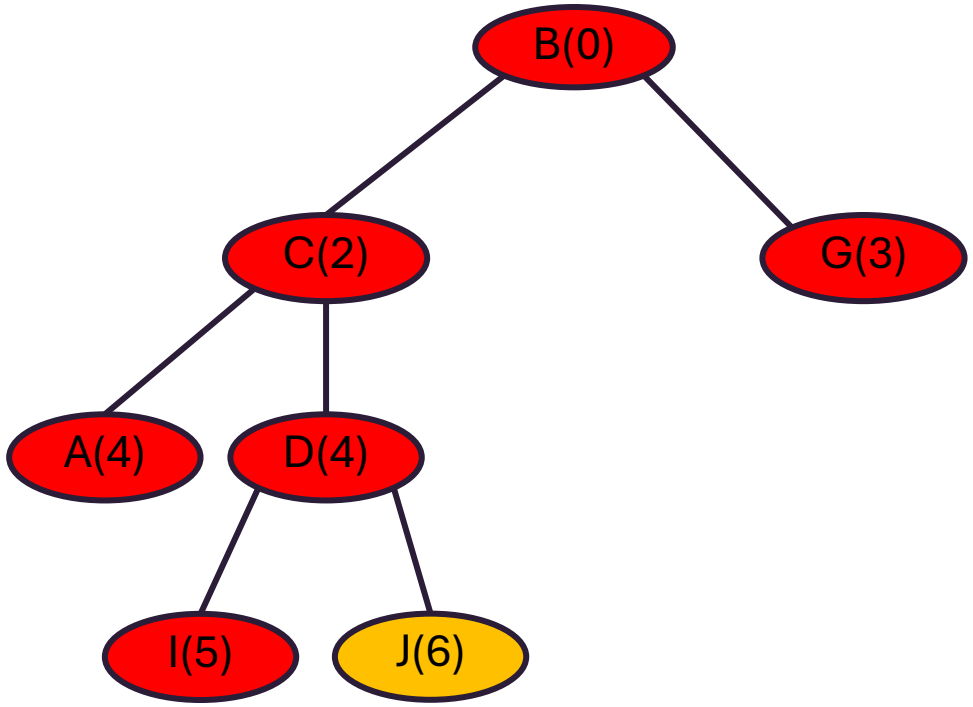
Open (Q):

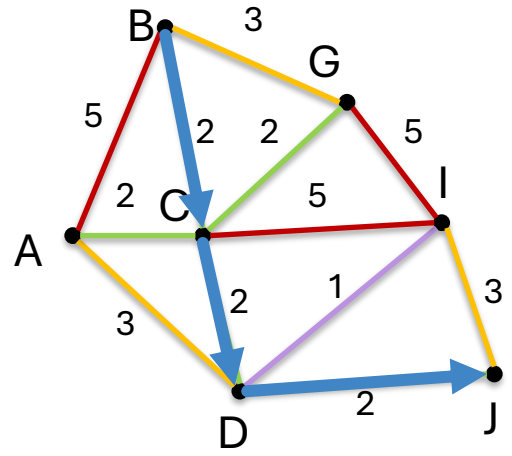
Node	Cost	Parent
J	6	D

Closed (R):

Node	Cost	Parent
B	0	
C	2	B
G	3	B
A	4	C
D	4	C
I	5	D

Dijkstra's Algorithm Example





Frontier (Q):

Node	Cost	Parent

Closed

Node	Cost	Parent
B	0	
C	2	B
G	3	B
A	4	C
D	4	C
I	5	D
J	6	D

Final path solution: B→C→D→J
with path cost 6

A* Heuristic Search

- Heuristic:
  - Any optimistic estimate of how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance

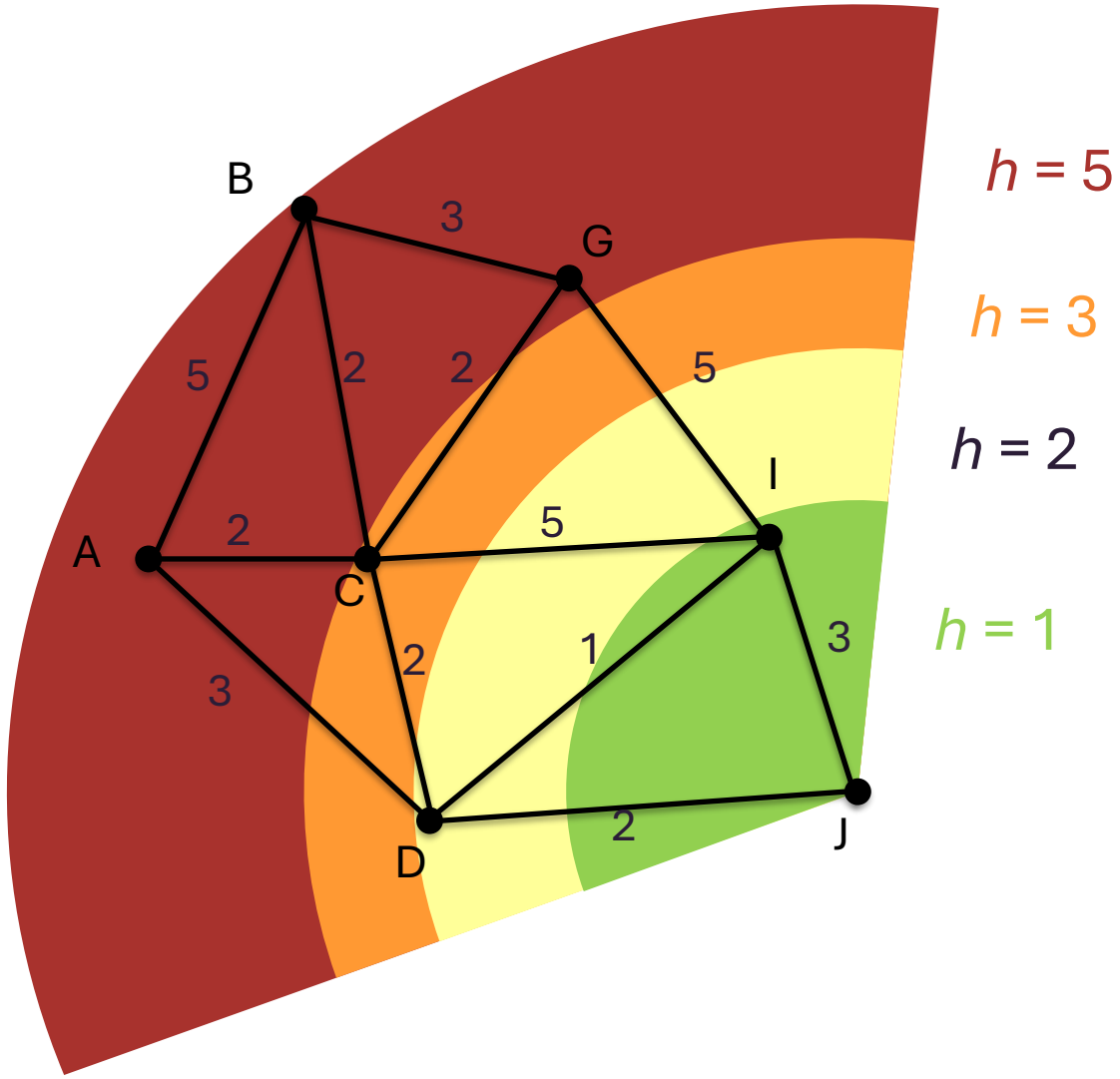
- A* priority:

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

Cost to arrive

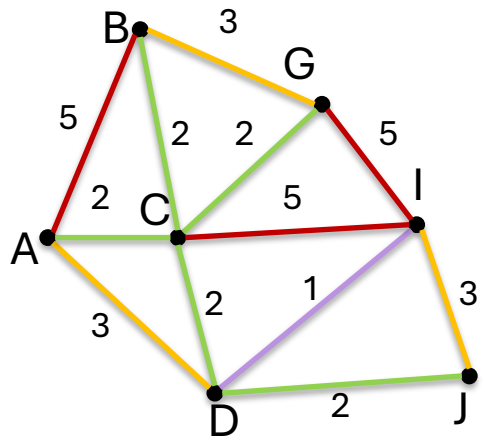
Heuristic cost to goal

A* Heuristic



A* Algorithm Example

B(0)



Our A* queue will be ordered by cost to arrive + heuristic:

- push (*Q.Insert*) by A* priority, $f(n)$
- pop (*Q.GetFirst*) from the front, and add it to the closed list

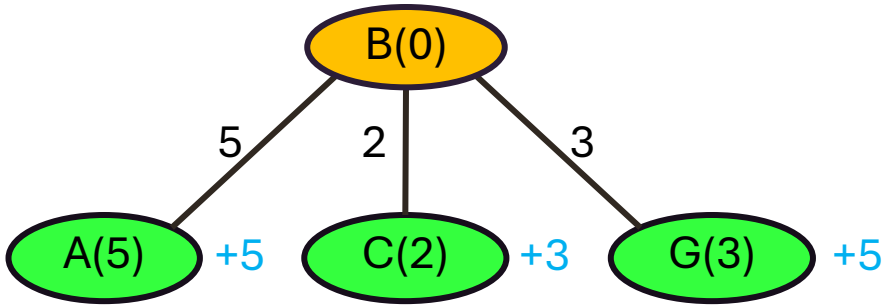
Open (*Q*):

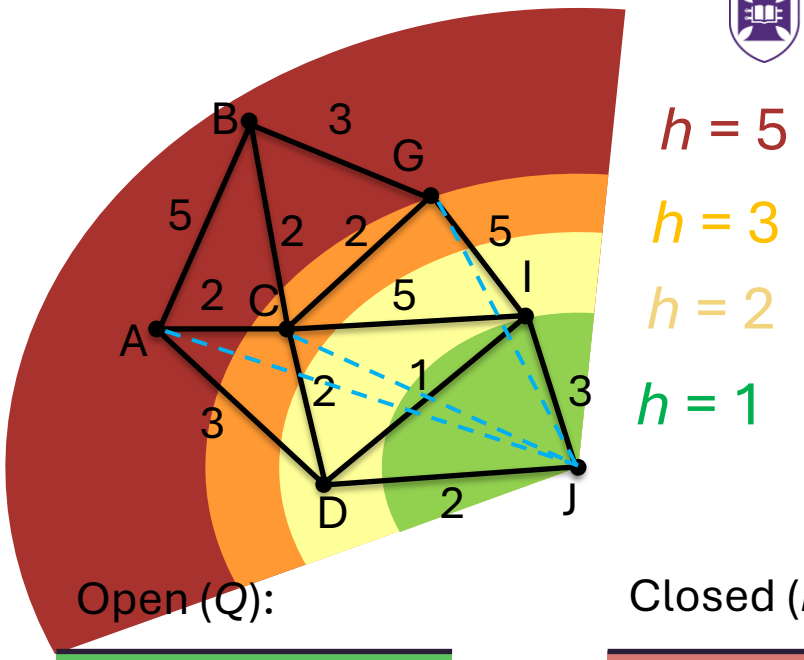
Node	Cost	Parent

Closed (*R*):

Node	Cost	Parent
B	0	

A* Algorithm Example





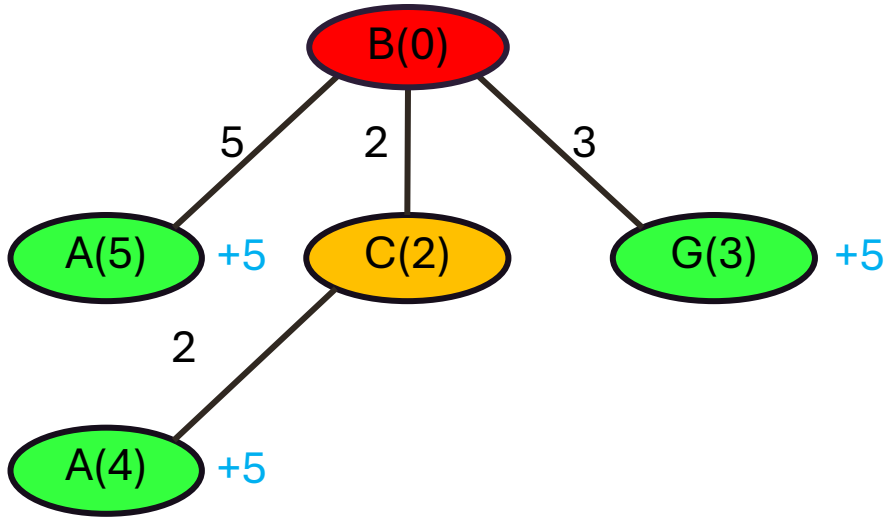
Open (*Q*):

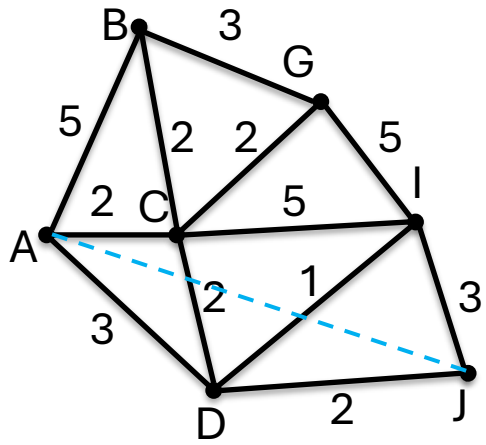
Node	Cost	Parent
C	2+3	B
G	3+5	B
A	5+5	B

Closed (*R*):

Node	Cost	Parent
B	0	

A* Algorithm Example





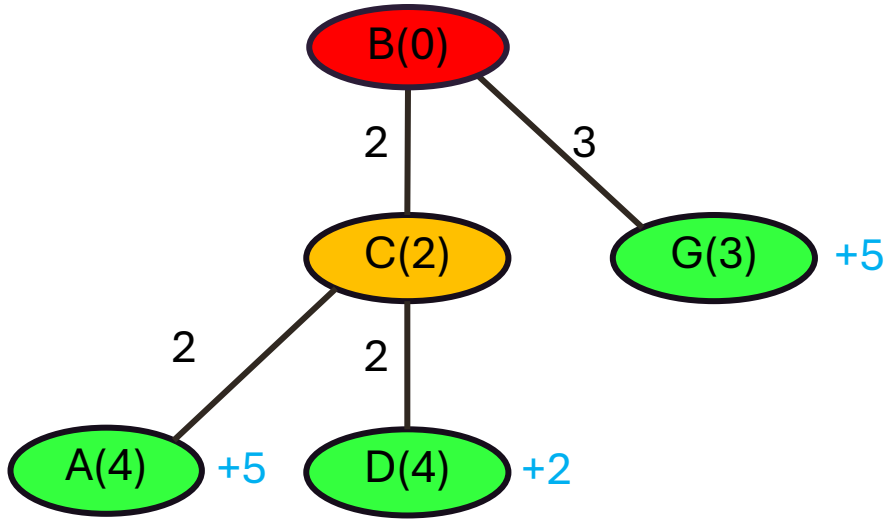
Open (*Q*):

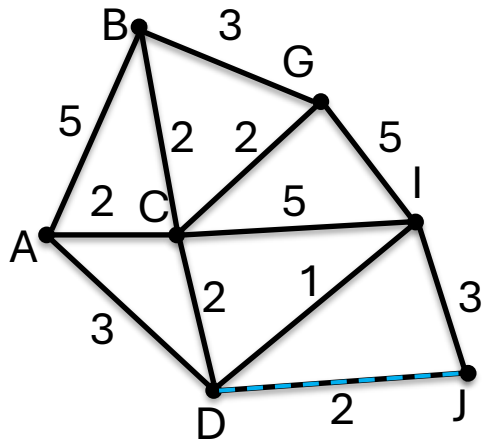
Node	Cost	Parent
G	3+5	B
A	4+5	C

Closed (*R*):

Node	Cost	Parent
B	0	
C	2	B

A* Algorithm Example





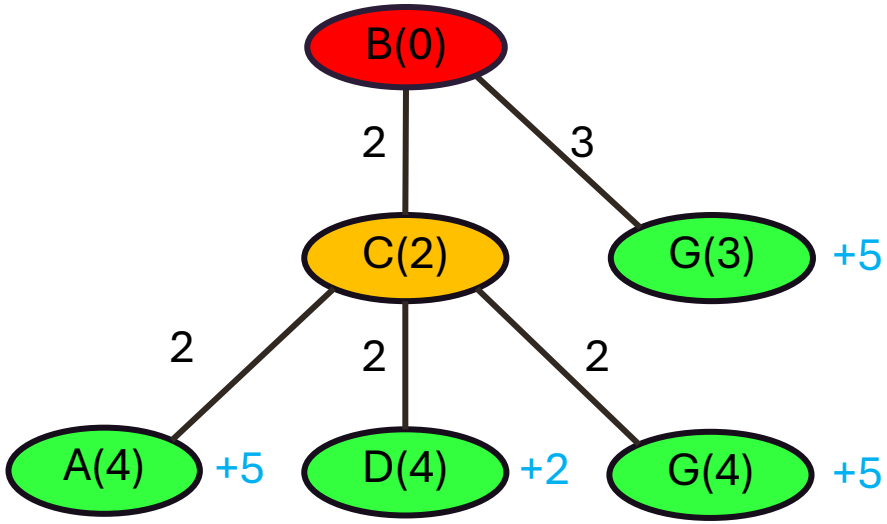
Open (Q):

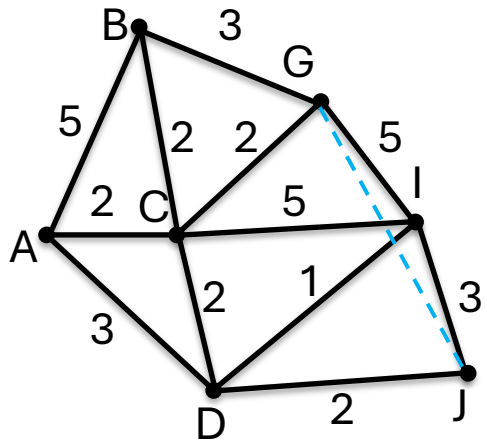
Node	Cost	Parent
~~D~~	~~4+2~~	~~C~~
A	4+5	C

Closed (R):

Node	Cost	Parent
B	0	
C	2	B

A* Algorithm Example





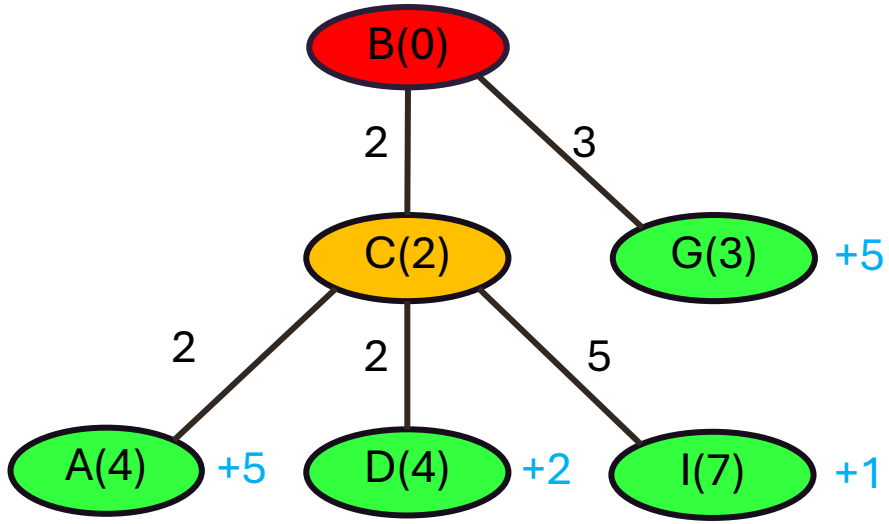
Open (*Q*):

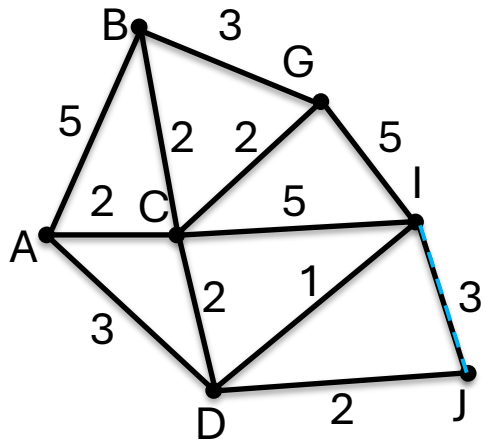
Node	Cost	Parent
D	4+2	C
G	3+5	B
A	4+5	C

Closed (*R*):

Node	Cost	Parent
B	0	
C	2	B

A* Algorithm Example





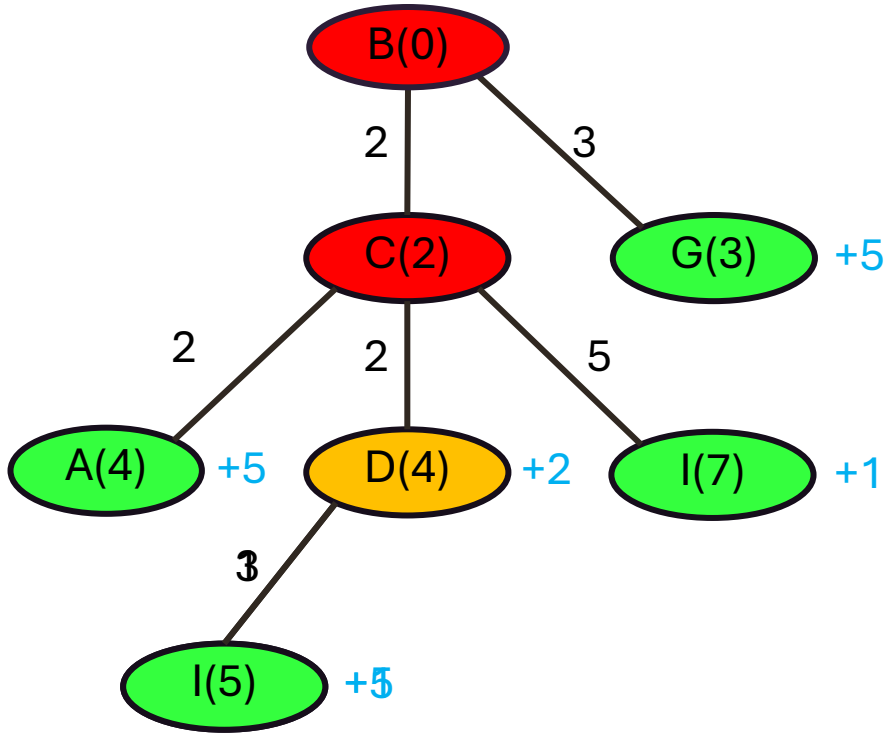
Open (*Q*):

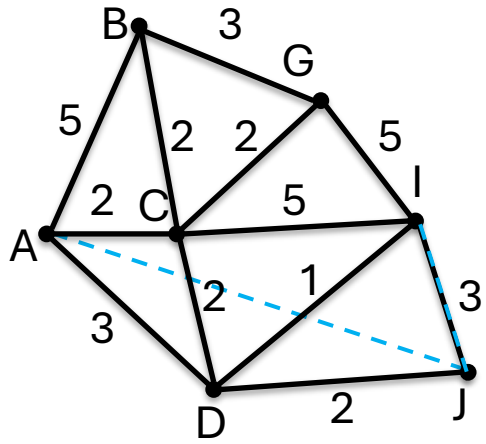
Node	Cost	Parent
D	4+2	C
~~G~~	~~3+5~~	~~B~~
A	4+5	C

Closed (*R*):

Node	Cost	Parent
B	0	
C	2	B

A* Algorithm Example





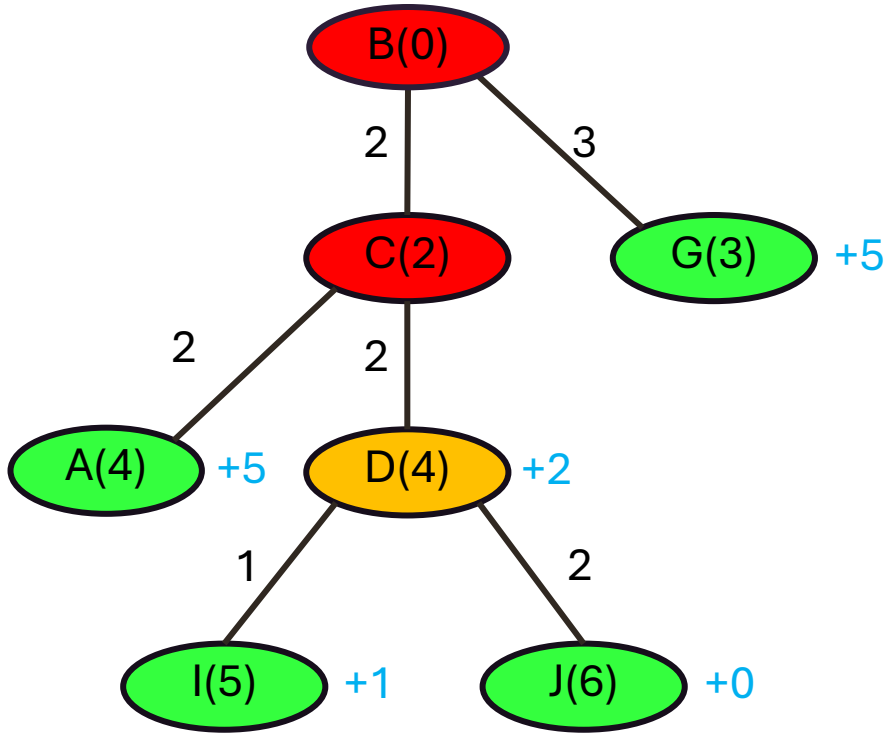
Open (Q):

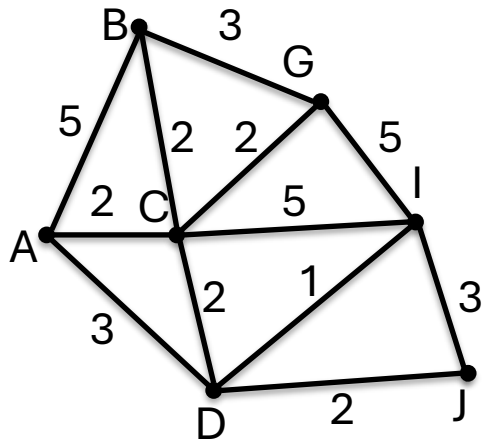
Node	Cost	Parent
I	5+1	D
I	7+1	C
G	3+5	B
A	4+5	C

Closed (R):

Node	Cost	Parent
B	0	
C	2	B
D	4	C

A* Algorithm Example





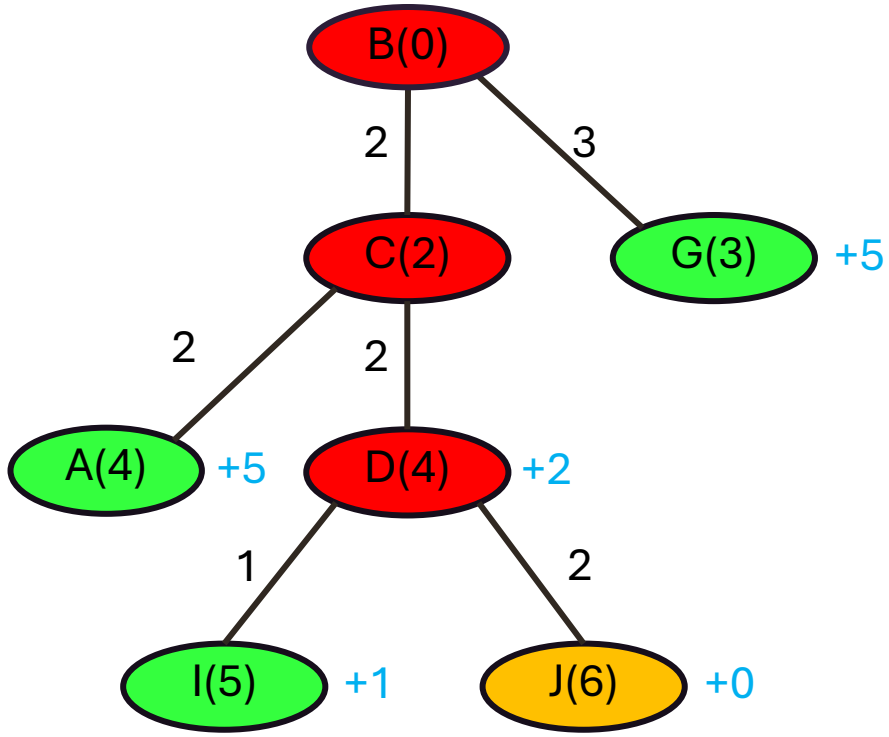
Open (*Q*):

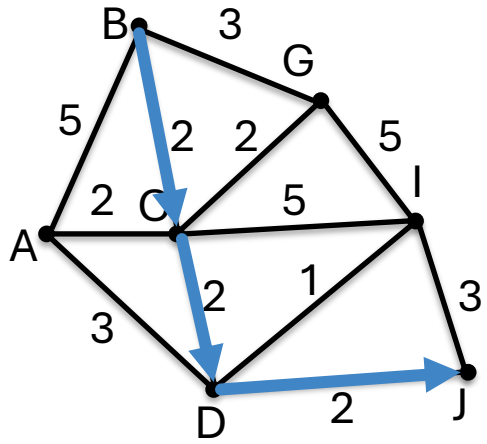
Node	Cost	Parent
~~J~~	~~6+0~~	~~D~~
G	3+5	B
A	4+5	C

Closed (*R*):

Node	Cost	Parent
B	0	
C	2	B
D	4	C

A* Algorithm Example





Open (*Q*):

Node	Cost	Parent
I	5+1	D
G	3+5	B
A	4+5	C

Closed (*R*):

Node	Cost	Parent
B	0	
C	2	B
D	4	C
J	6	D

Final path solution: B→C→D→J
with path cost 6

A* Heuristic

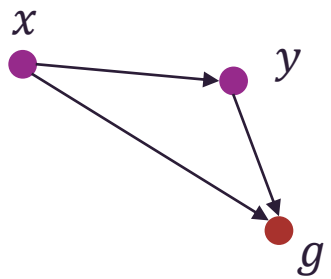
- The heuristic must be **admissible**
  - It never overestimates the cost

True cost to goal

$$h(x) \leq d(x, goal)$$

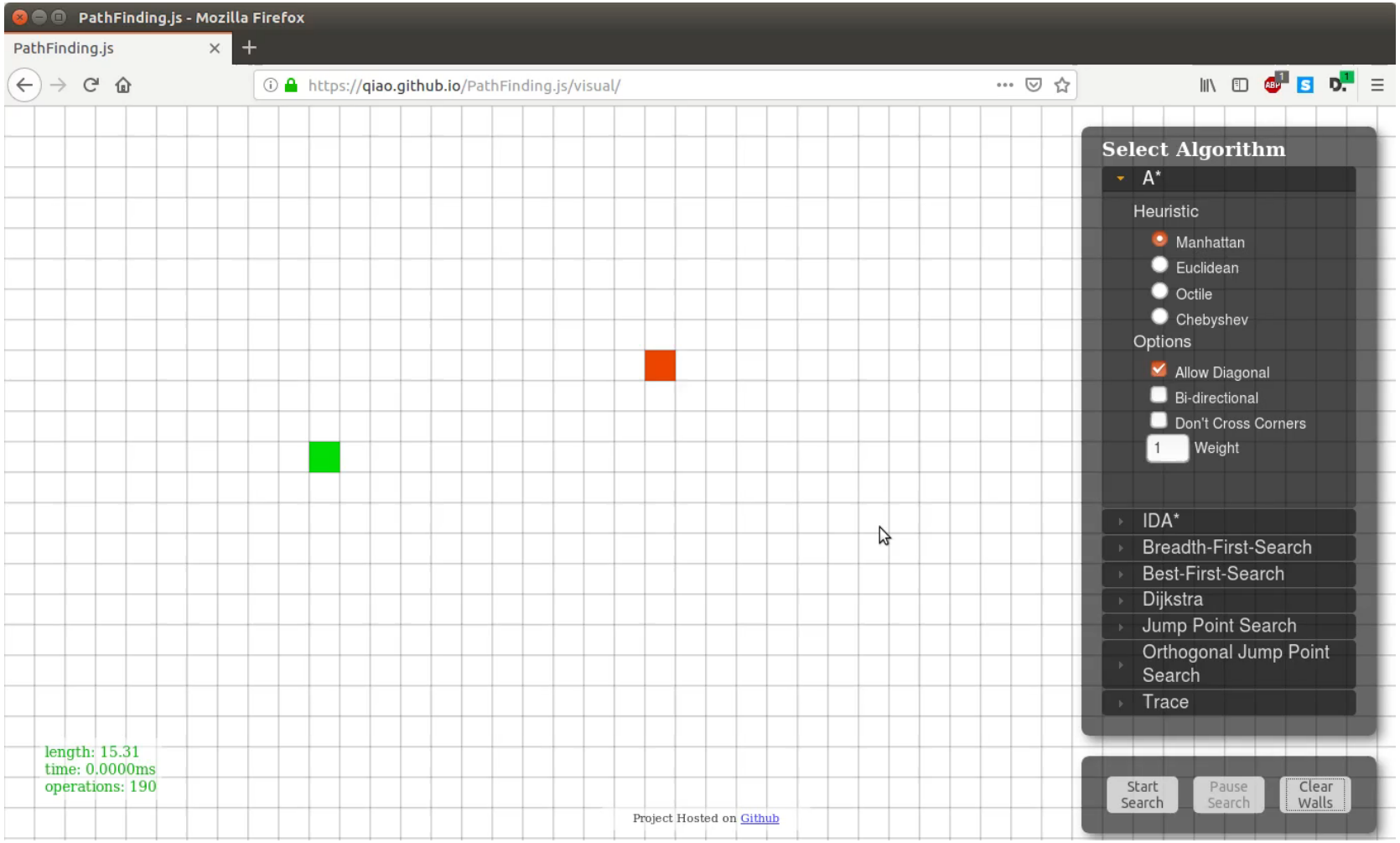
- The heuristic must be **consistent**
  - For any pair of adjacent nodes x and y, where $d(x,y)$ is the cost of edge between them

$$h(x) \leq d(x, y) + h(y)$$



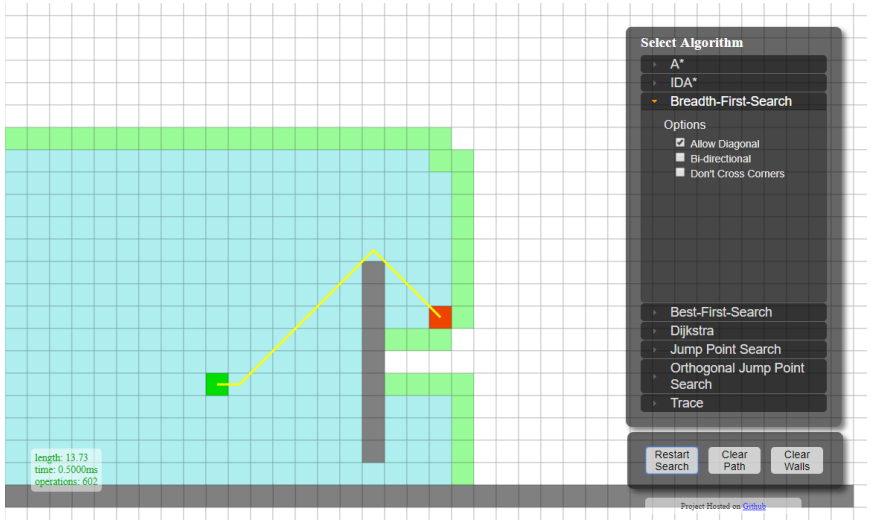
- Typical valid heuristics:
  - Euclidean distance
  - Manhattan distance
  - Zero (Dijkstra's algorithm)

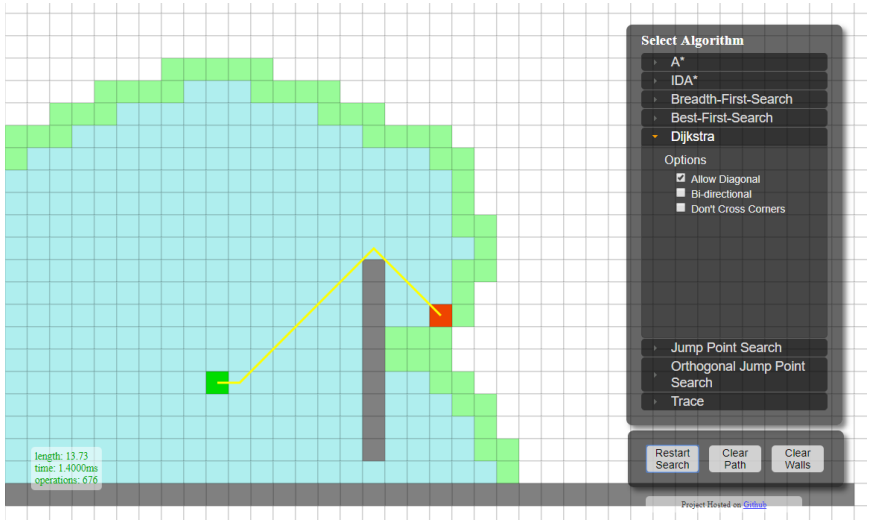
Example: Graph Search

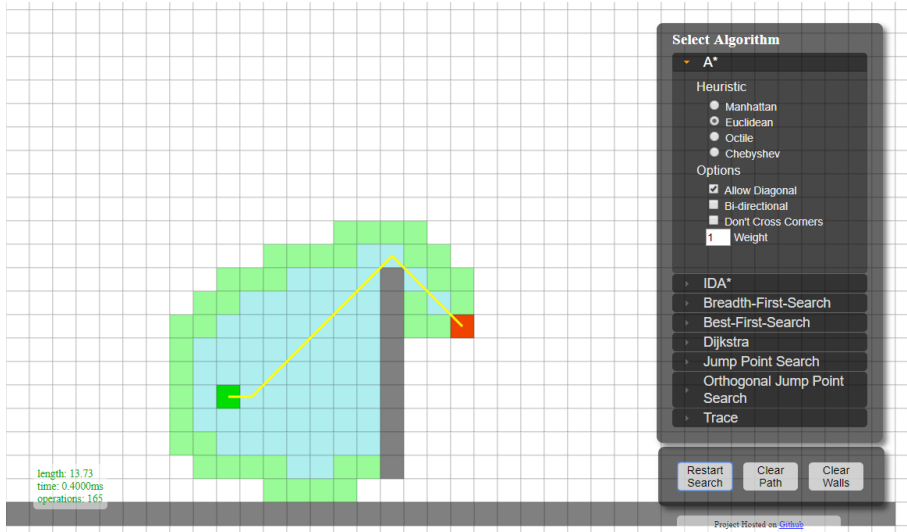


https://qiao.github.io/PathFinding.js/visual/

Example: Graph Search





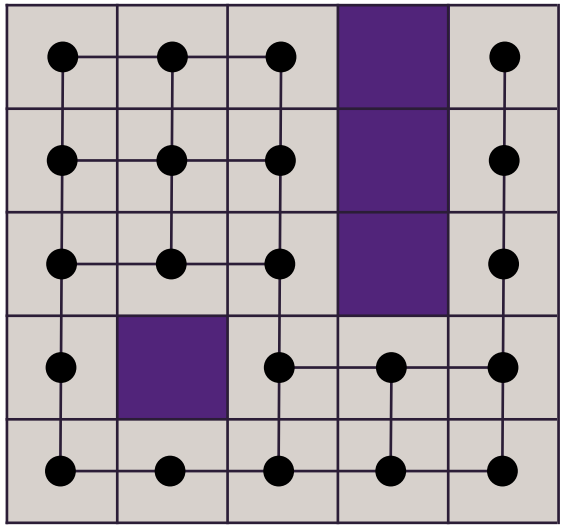


https://qiao.github.io/PathFinding.js/visual/

Sampling-based Methods

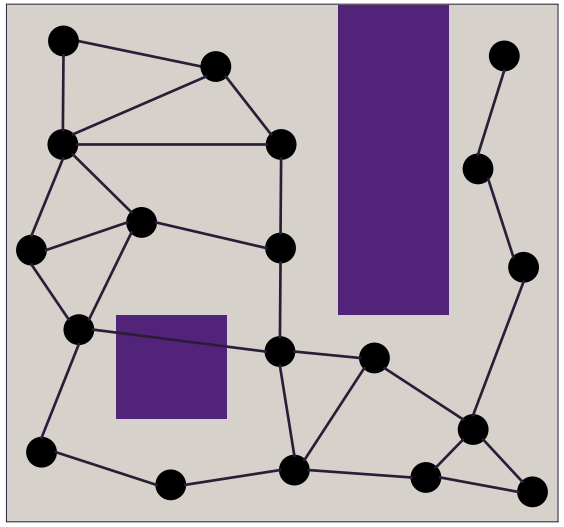
How to Build a Valid Graph

Explicitly use obstacles and grid



- Grid resolution trades off representation precision vs. search time
- Connectivity defined by grid shape
- Suffers from *curse of dimensionality*

Use collision checks



- Representation precision determined by sampling distribution
- Connectivity defined by explicit collision check

Sampling-based Motion Planning

- Probabilistic Roadmaps (PRMs, 1996)
  - Construct roadmap, then search
- Rapidly Exploring Random Trees (RRTs, 1998)
  - Online roadmap construction and search

- General framework:

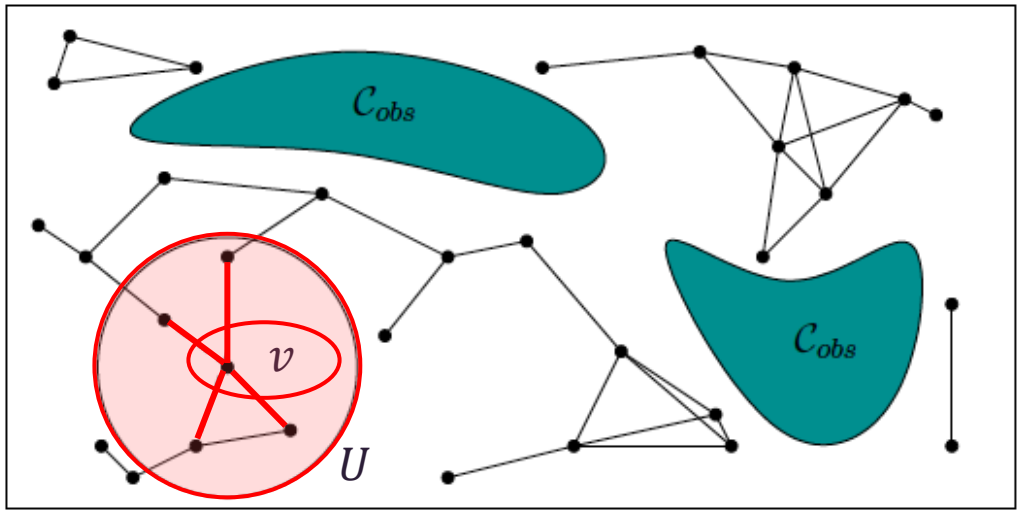
PRM: In parallel!

1. Initialise empty graph $G = (\mathcal{V}, \mathcal{E})$ (may include start node)
2. Generate **random configuration** c and add to $\mathcal{V}$ ➡ Need sampler
3. For each new c, select nodes from $\mathcal{V}$ 'near' c and try to connect ➡ Need distance metric
4. If local path **collision free**, add edge to $\mathcal{E}$ ➡ Need collision detection
5. Check for solution
6. Repeat from 2

PRM: Building the Roadmap

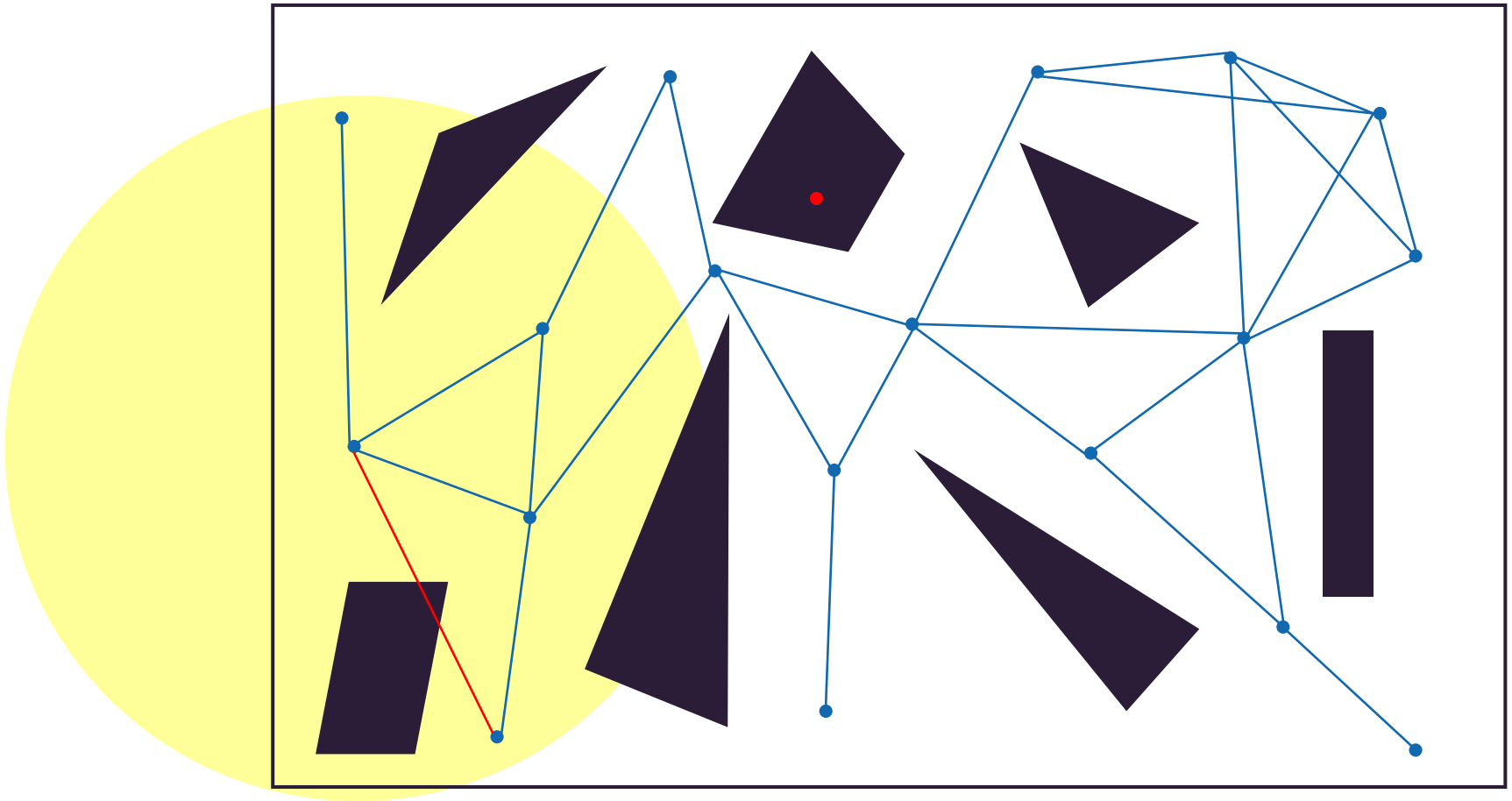
Algorithm 2: sPRM.

1 $V \leftarrow \{x_{\text{init}}\} \cup \{\texttt{SampleFree}_i\}_{i=1,\ldots,n}$; $E \leftarrow \emptyset$;
2 **foreach** $v \in V$ **do**
3 $U \leftarrow \texttt{Near}(G=(V,E),v,r) \setminus \{v\}$;
4 **foreach** $u \in U$ **do**
5 **if** $\texttt{CollisionFree}(v,u)$ **then**
  $E \leftarrow E \cup \{(v,u),(u,v)\}$
6 **return** $G=(V,E)$;

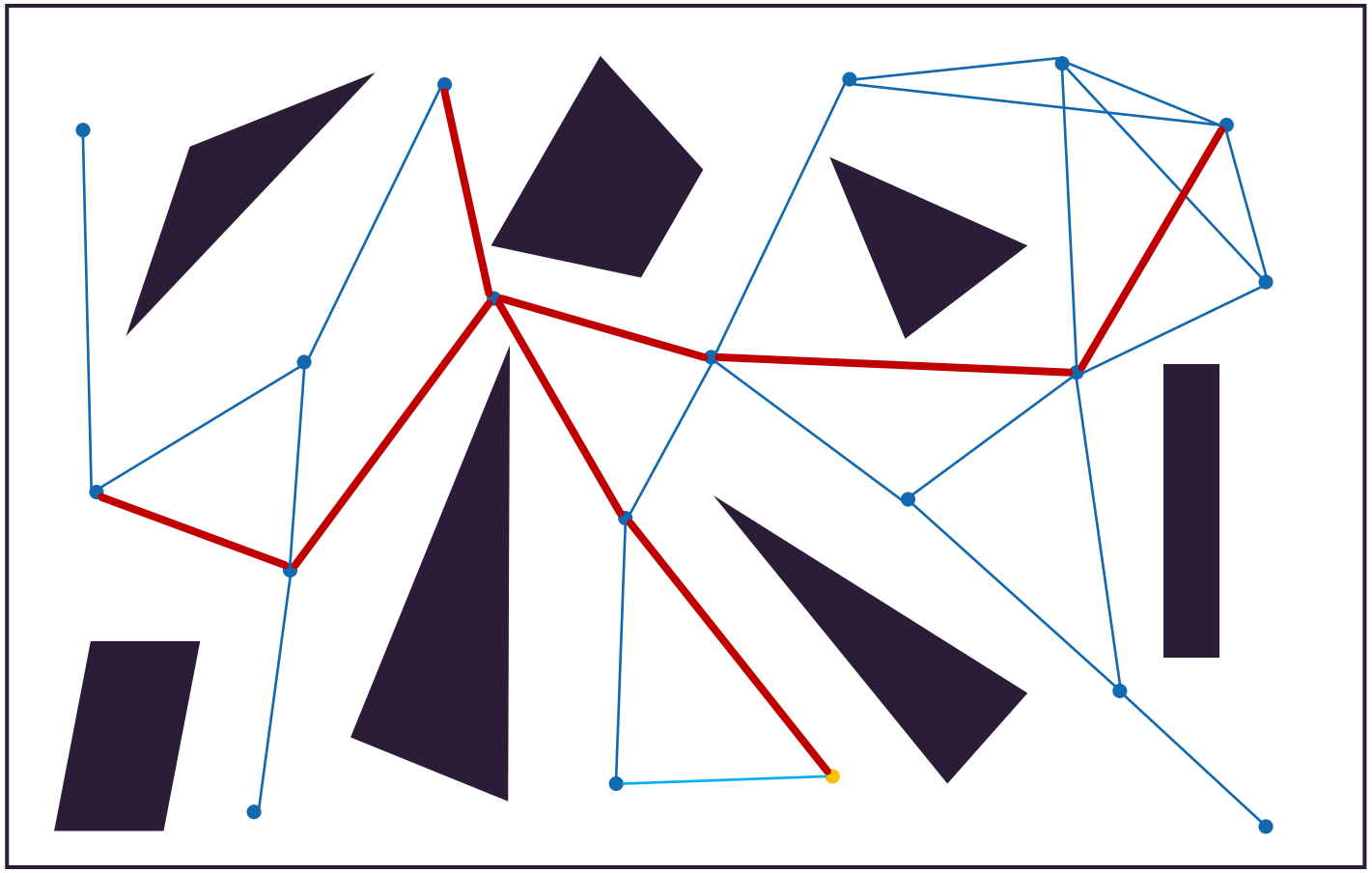


S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning." *The International Journal of Robotics Research* 30.7 (2011): 846-894.

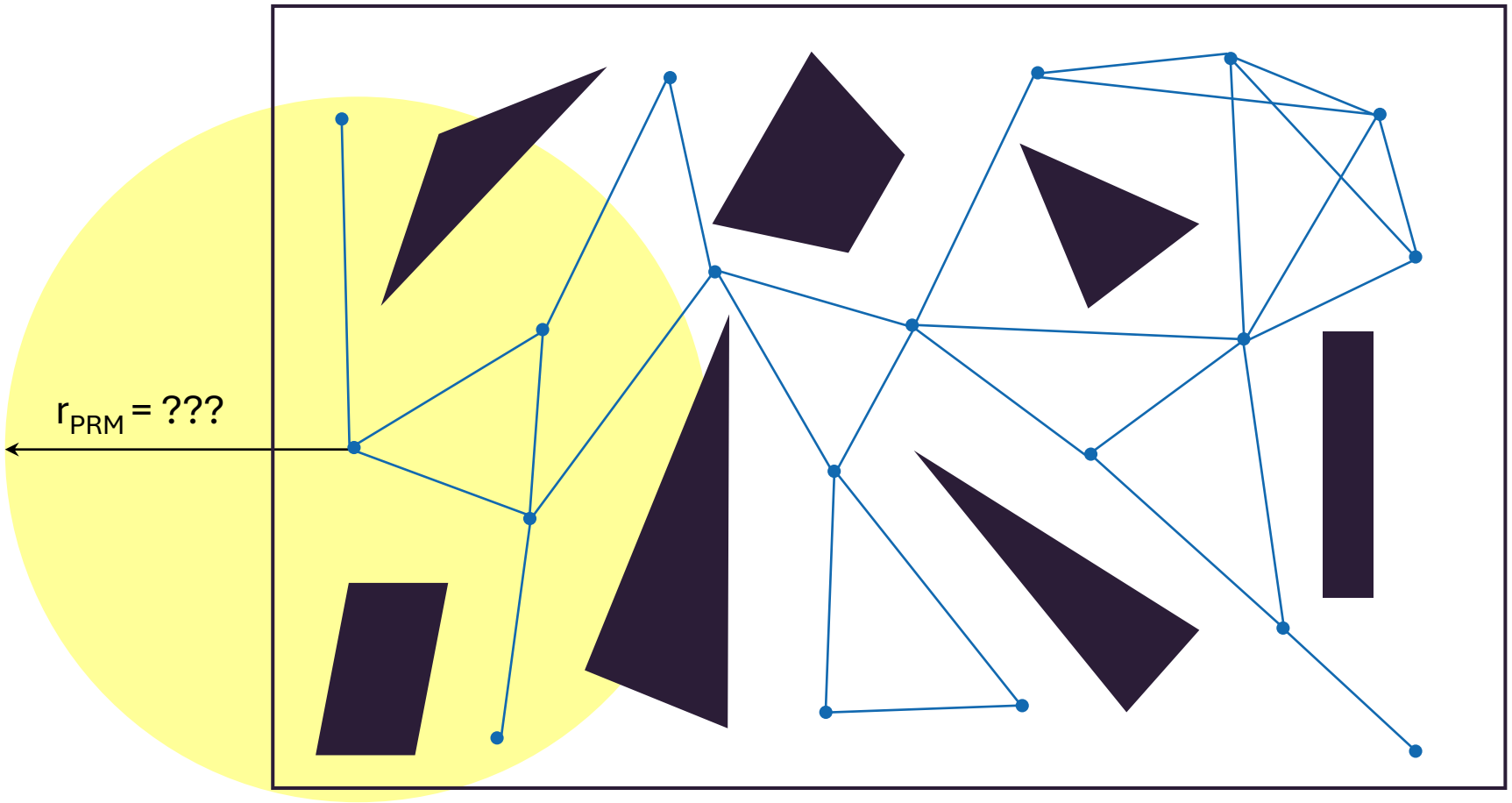
Example: PRM



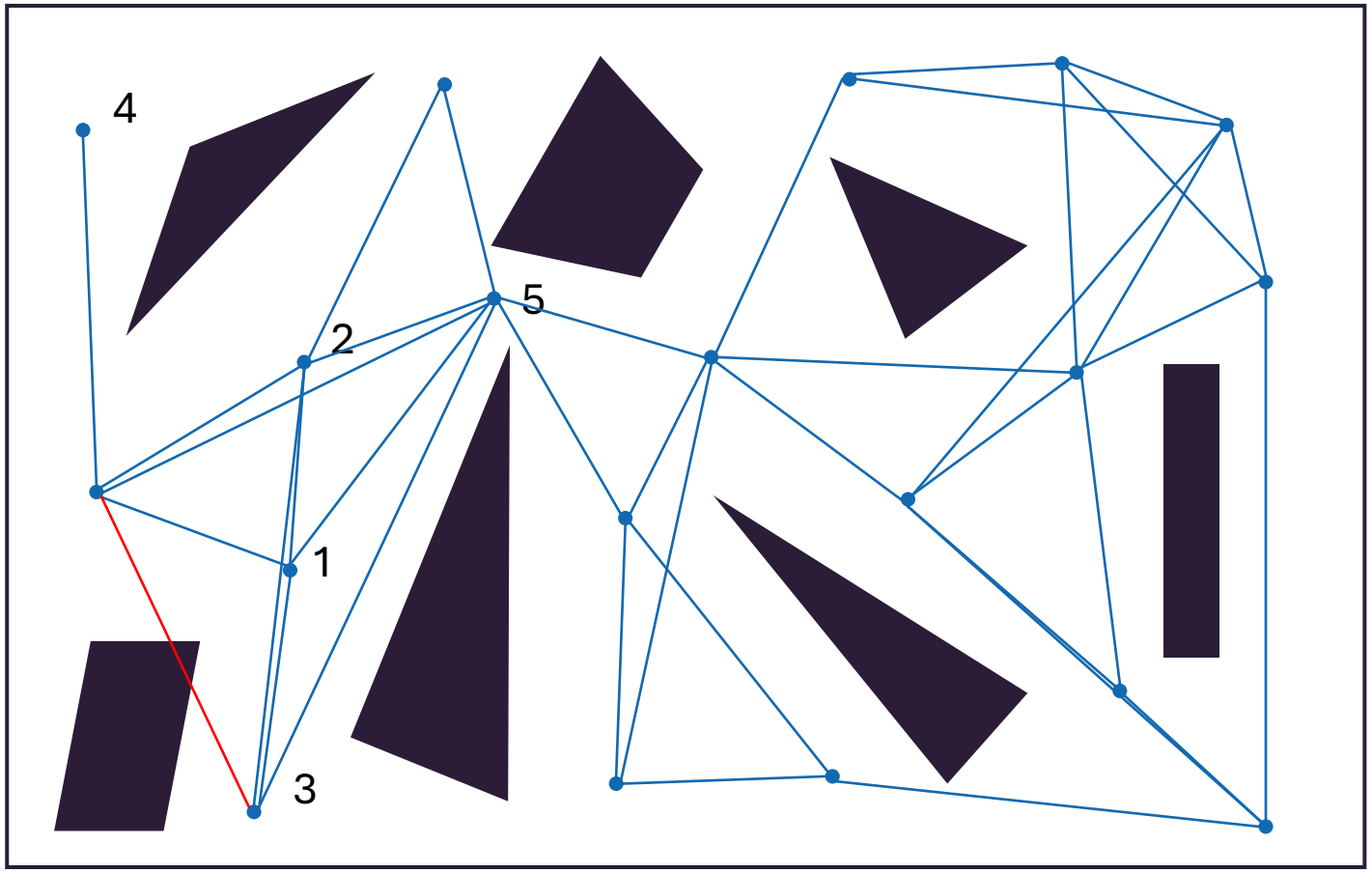
Example: PRM



PRM: How to Connect?



PRM: How to Connect?



Connect to k nearest neighbours

PRM Summary

- Graph constructed by connecting random configurations
- Designed for **multiple queries**
- Underlying assumption is that it's worth performing **substantial pre-computation** on a given environment to enable multiple planning queries
- However, often only interested in an **efficient single query** without pre-computations
  - E.g. in dynamic environments, different robot models
- Idea: combine **exploration and search** into a single method
  - Motivates tree-based planning approaches

Rapidly Exploring Random Trees

- Instead of sampling all our target states at once, start from initial configuration and **incrementally** grow outwards from there, forming a tree of connected, dynamically feasible local paths

- Terminate when we reach a configuration in a goal **region**

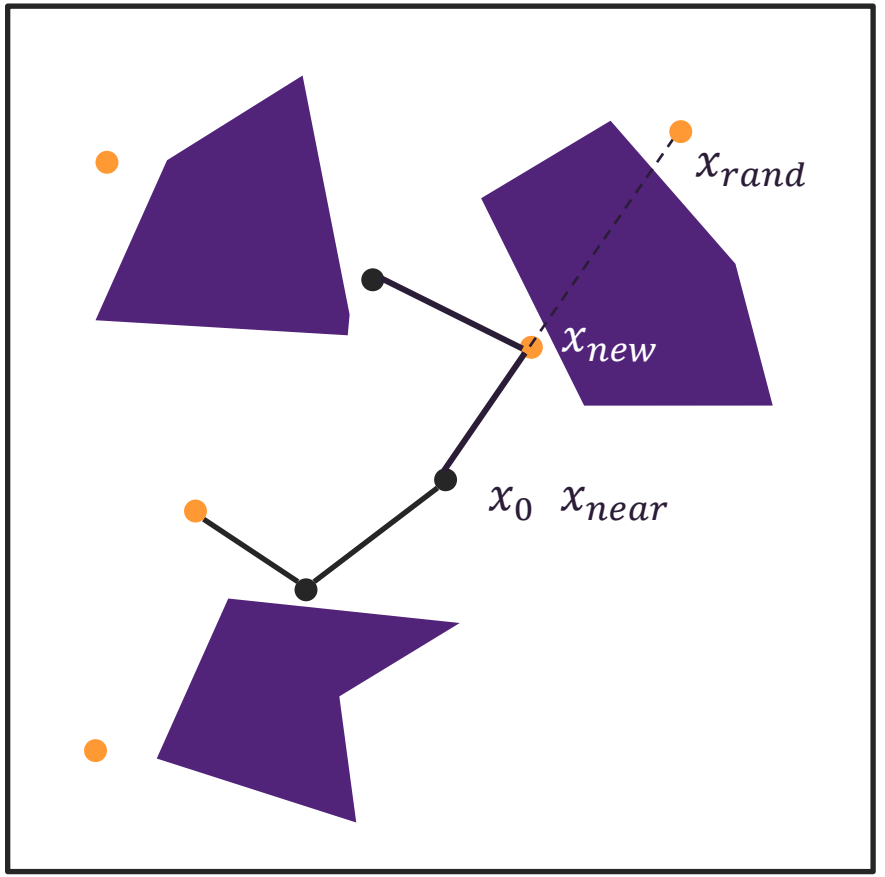
- For **single-query searches**

RRTs

Algorithm 1 RRT (space filling) - Adapted from [1]

Require: Initial pose x_0,

1: $V \leftarrow \{x_0\}, E \leftarrow \emptyset$
2: **for** $i = 1, \ldots, n$ **do**
3: $x_{rand} \leftarrow$ SAMPLEFREESPACE()
4: $x_{near} \leftarrow$ FINDNEAREST(V, x_{rand})
5: $x_{new} \leftarrow$ STEER(x_{rand}, x_{near})
6: **if** OBSTACLEFREE(x_{near}, x_{new}) **then**
7: $V \leftarrow V \cup \{x_{new}\}$
8: $E \leftarrow E \cup \{(x_{near}, x_{new})\}$
9: **return** $\mathcal{G} = (V, E)$

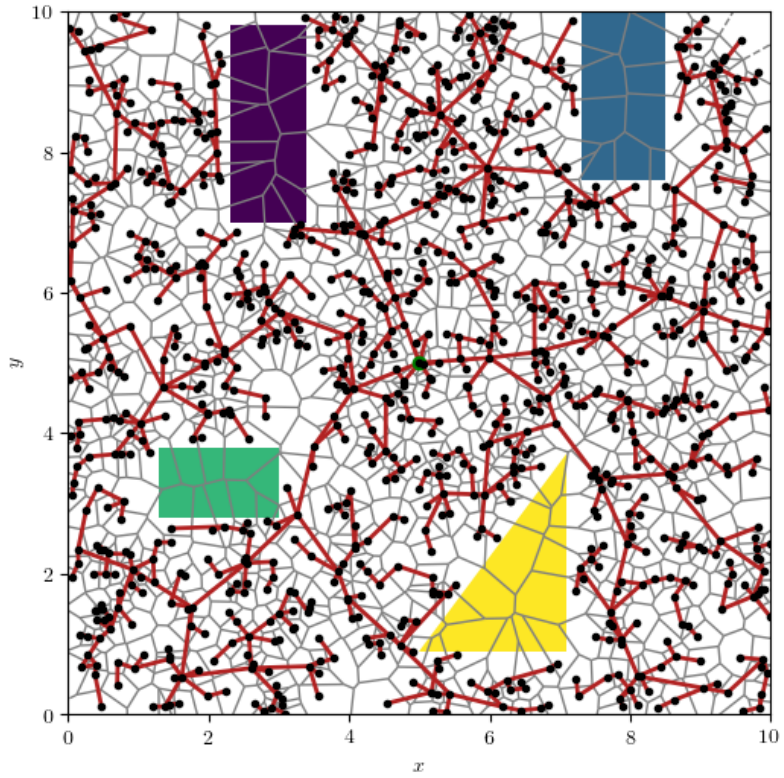


S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning." *Technical Report. Computer Science Department, Iowa State University (TR 98–11).* (1998)

S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning." *The International Journal of Robotics Research* 30.7 (2011): 846-894.

RRTs

- Incrementally explore the C-space
  - In early iterations, the RRT quickly reaches unexplored parts (Voronoi diagram)
  - However, RRT is dense in the limit → will eventually reach all points in the space



45 iterations

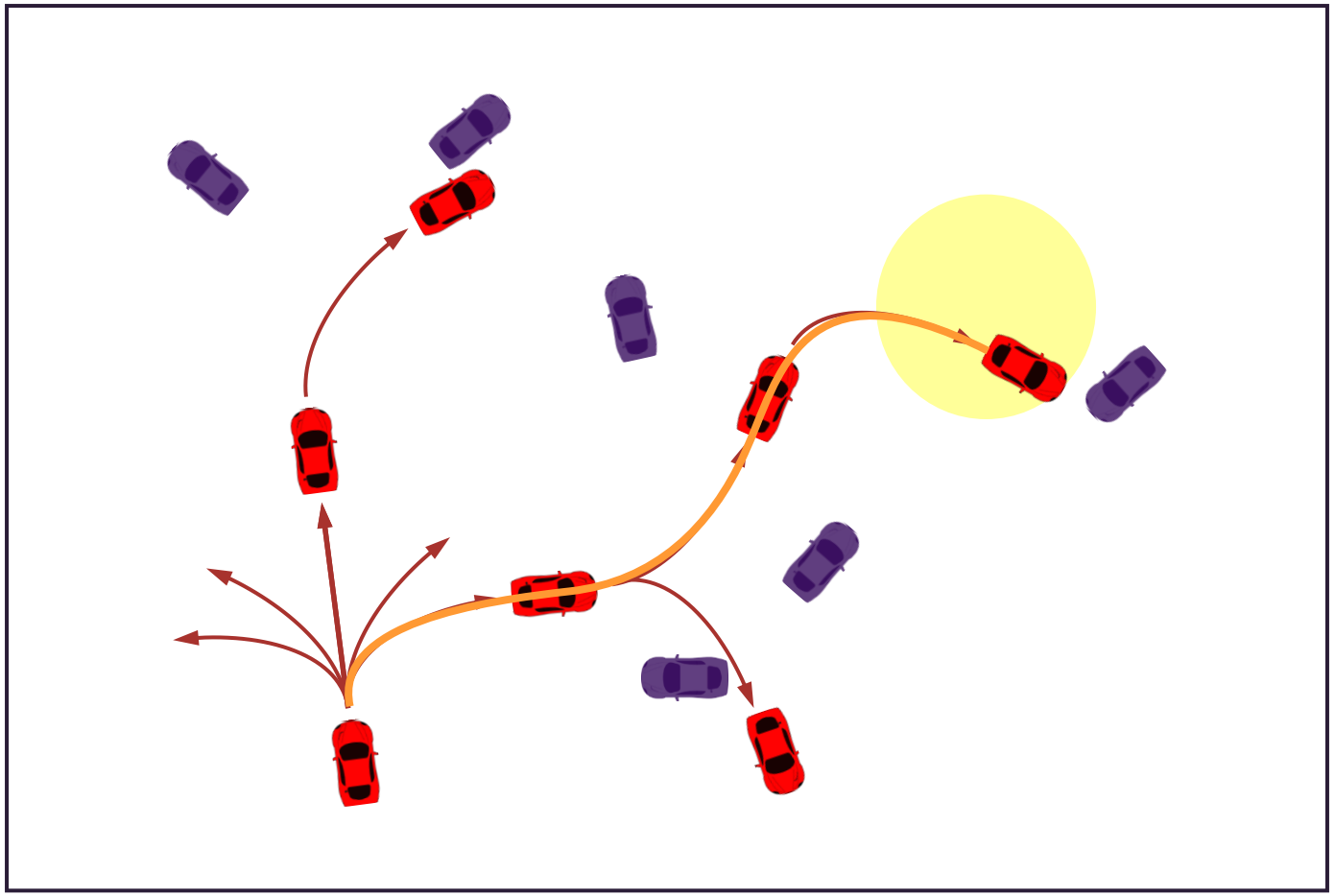
2345 iterations

Kinodynamic Planning

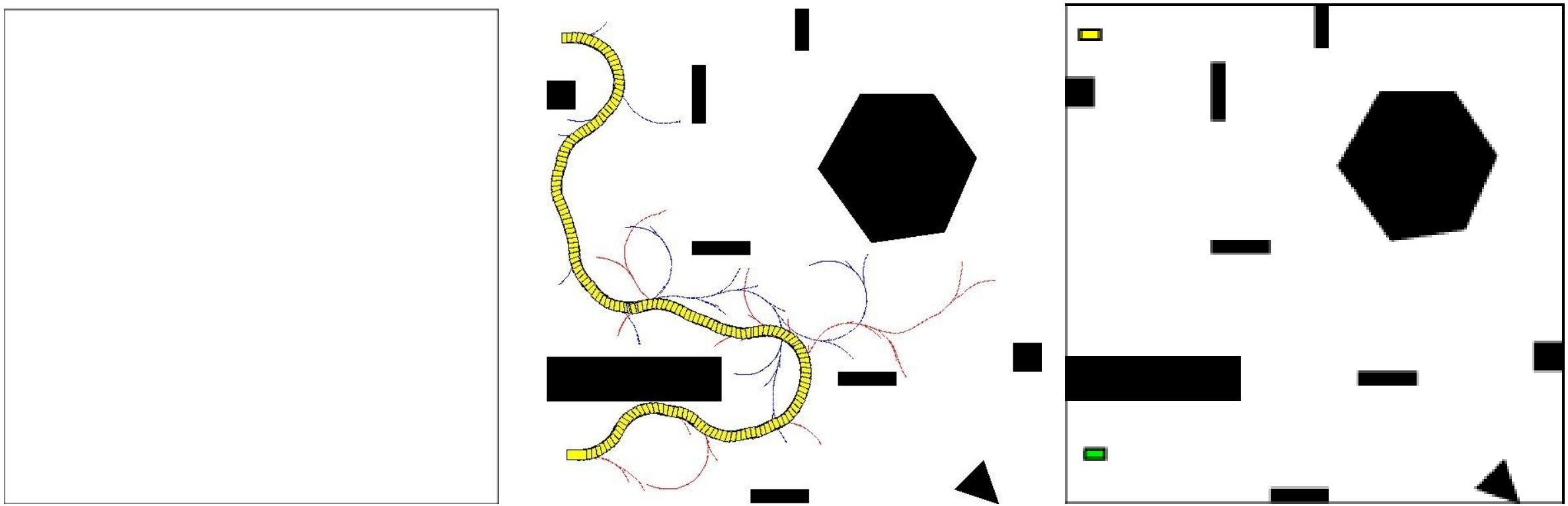
- Specifically designed to handle non-holonomic constraints and dynamic constraints (kinodynamic planning)

- Rather than solve for control inputs to exactly connect two points in C-space, just ‘steer’ in the right direction
  - Sample controls to generate **feasible** local paths and reach **approximate, intermediate** configuration that gets ‘close’ to the target

RRT Steering Function



RRT Examples



Steve LaValle (http://msl.cs.uiuc.edu/rrt/gallery.html)

Theoretical Guarantees

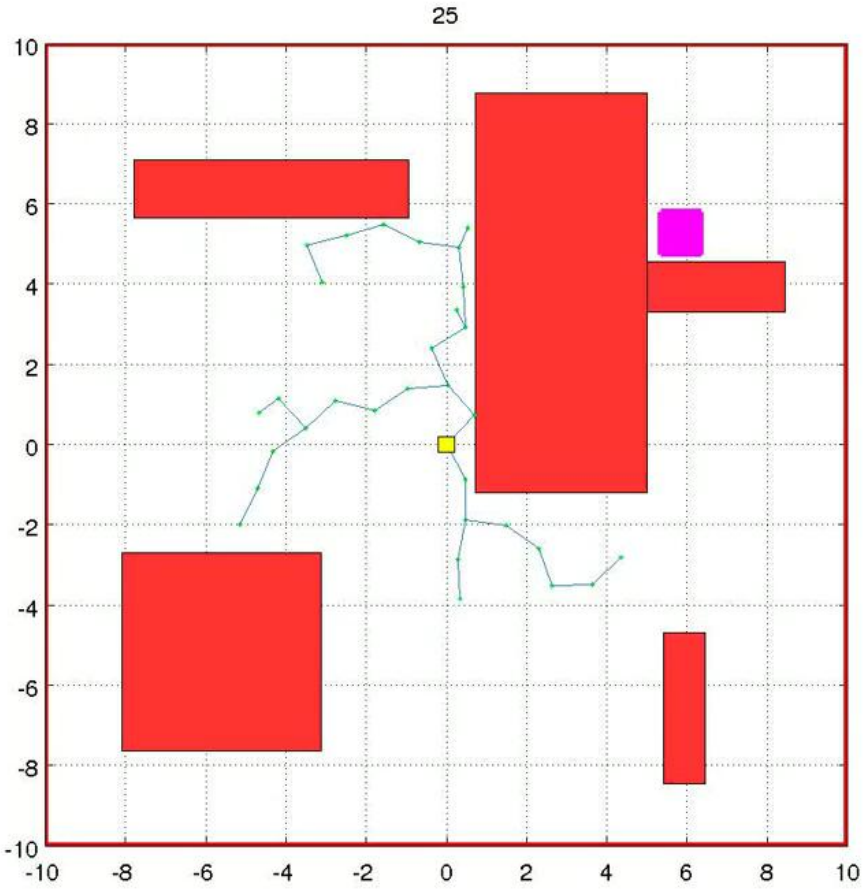
- Early on, PRM (and RRT) were known to be probabilistically complete, i.e.

“If there exists a solution, it will be found with probability → 1 as the number of samples → ∞”

- Will the RRT find the shortest path?

NO! They almost surely converge to a non-optimal solution

RRT



https://www.youtube.com/watch?v=FAFw8DoKvik

A Star Rises

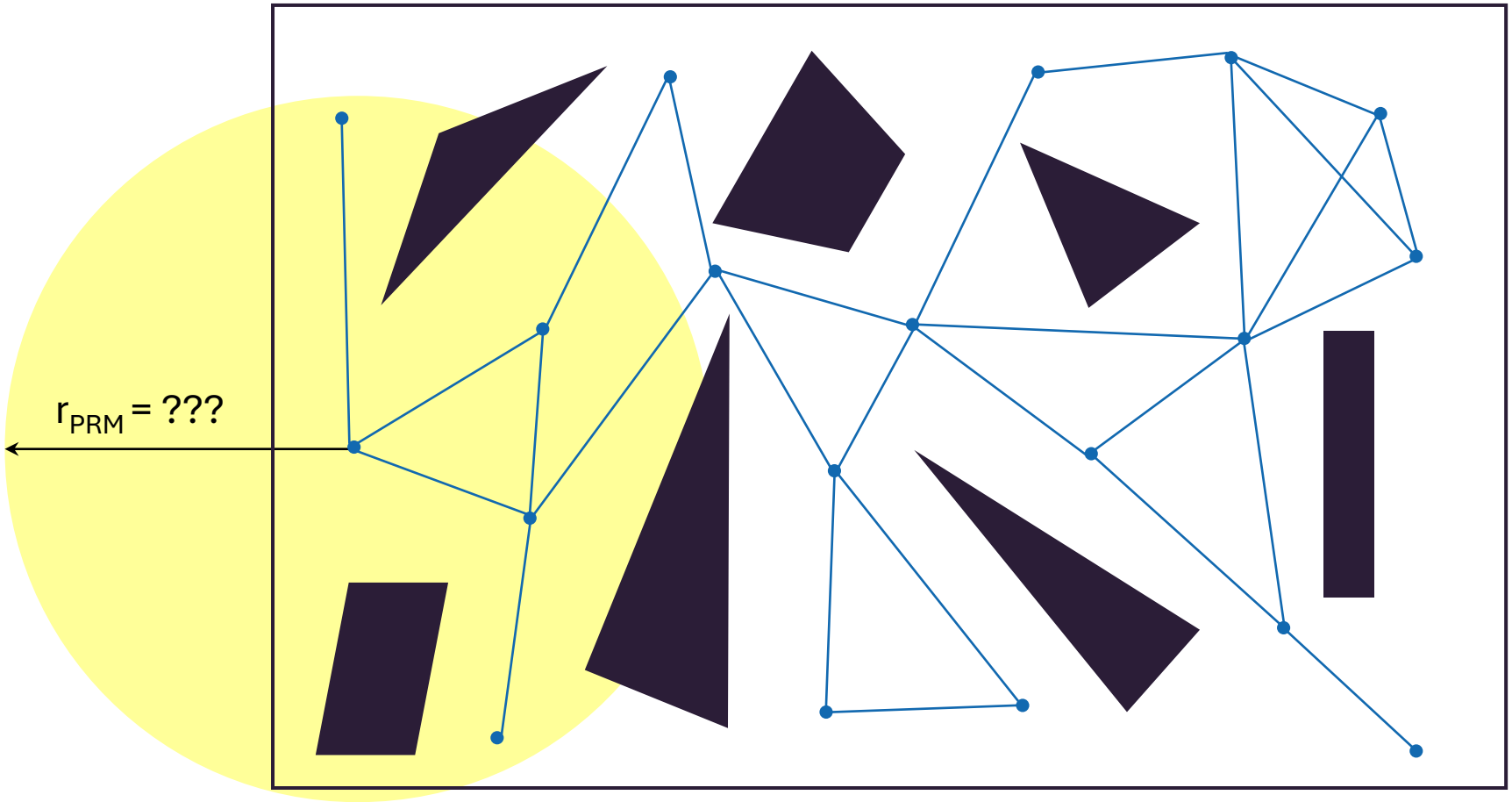
- Sertac Karaman and Emilio Frazzoli worked on incremental versions of PRM and RRT, and developed PRM* and RRT*, **asymptotically optimal** versions of PRM and RRT

“The cost of the returned solution will approach the optimal as the number of samples → ∞”

- ‘Star’ versions use rewiring technique (RRT*) and formal proof to select connection radius (PRM*) as function of sample density

S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning." *The International Journal of Robotics Research* 30.7 (2011): 846-894.

PRM*



number of samples

problem dimensionality

$$r^*_{PRM} = \gamma_{PRM}\left(\frac{\log n}{n}\right)^{\frac{1}{d}}$$

$$\gamma_{PRM} > 2\left(1+\frac{1}{d}\right)^{\frac{1}{d}}\left(\frac{\mu(X_{free})}{\zeta_d}\right)^{\frac{1}{d}}$$

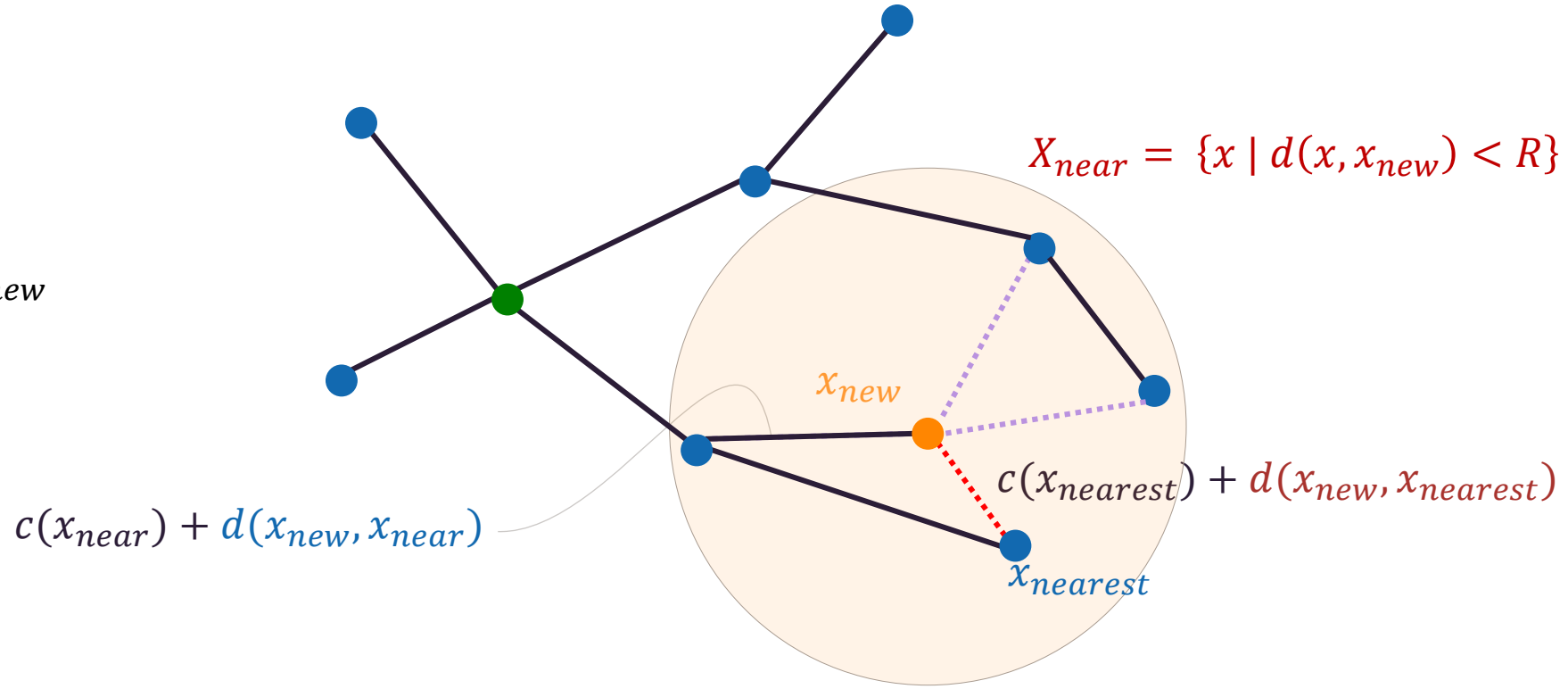
total free space

area of unit ball

RRT*

- Two key features
  - **Near** operations find the set of nodes that might be affected by a new node addition
  - **Rewiring** rebuilds the tree within this set to maintain the minimum cost of arrival tree

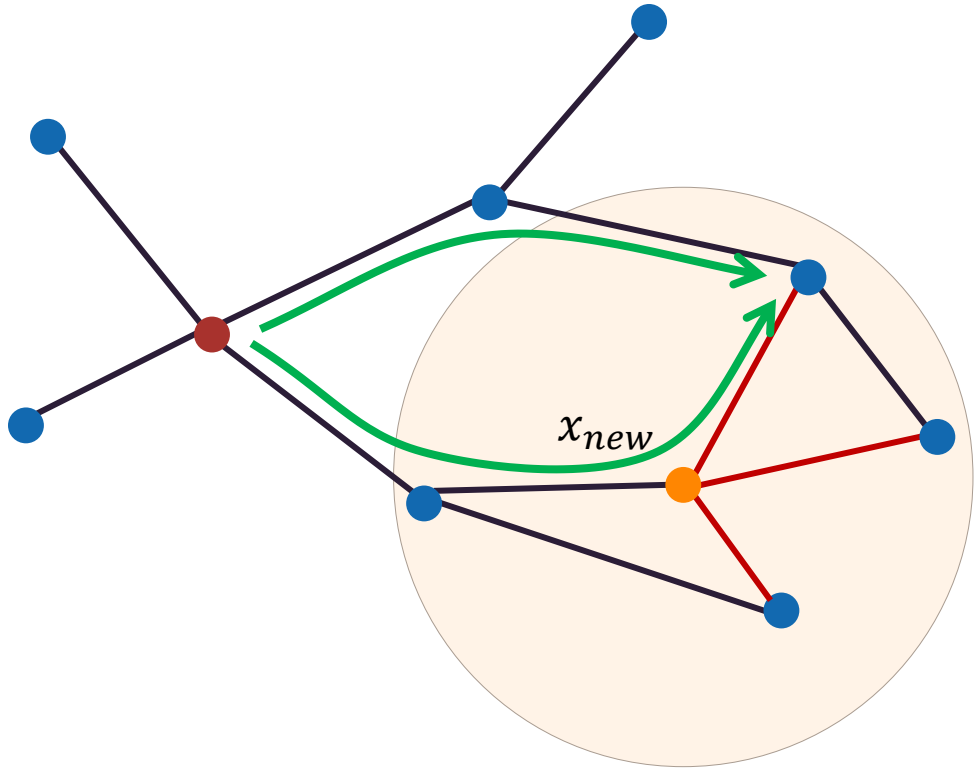
1. Shortest connection ***to*** x_{new}



RRT*

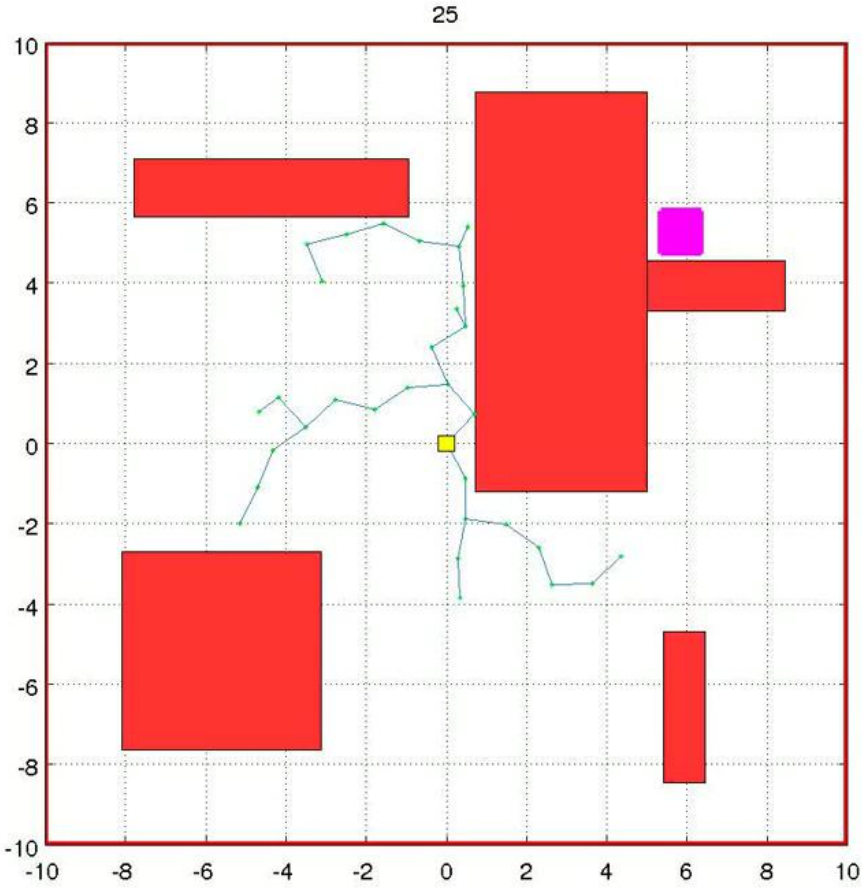
- Two key features
  - **Near** operations find the set of nodes that might be affected by a new node addition
  - **Rewiring** rebuilds the tree within this set to maintain the minimum cost of arrival tree

1. Shortest connection ***to*** x_{new}
2. Shortest connections ***from*** x_{new}



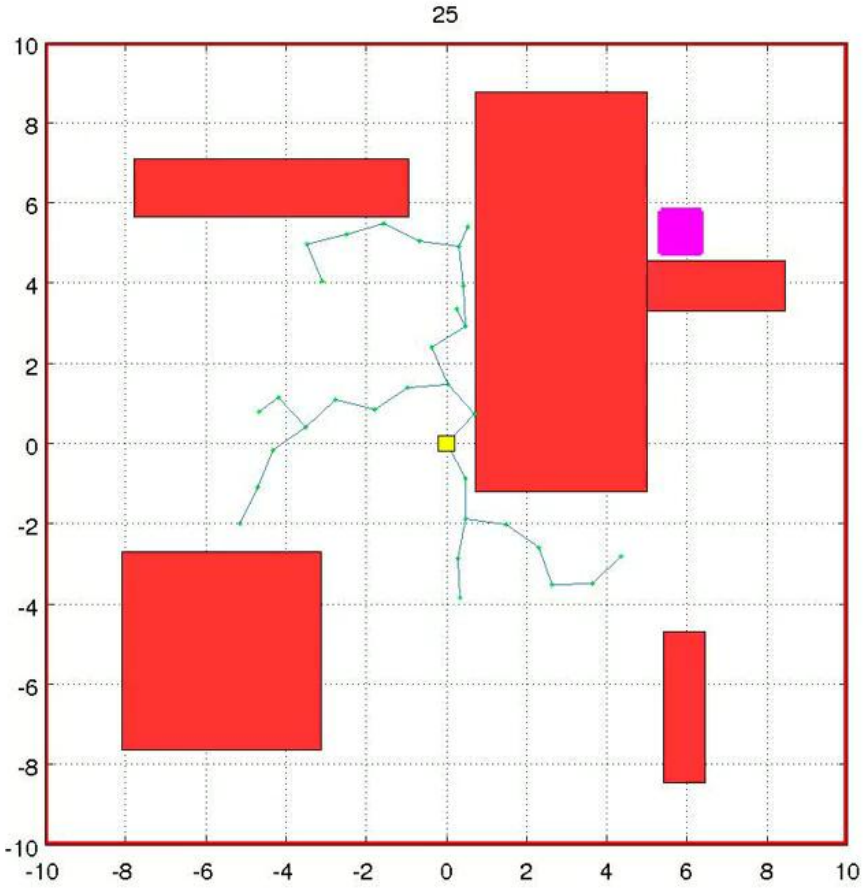
RRT vs RRT*

RRT



https://www.youtube.com/watch?v=FAFw8DoKvik

RRT*



https://www.youtube.com/watch?v=YKiQTJpPFkA

The Many MANY Variants of PRM and RRT

- The popularity of sampling-based planners has led to a huge variety of probabilistic sampling-based planning methods

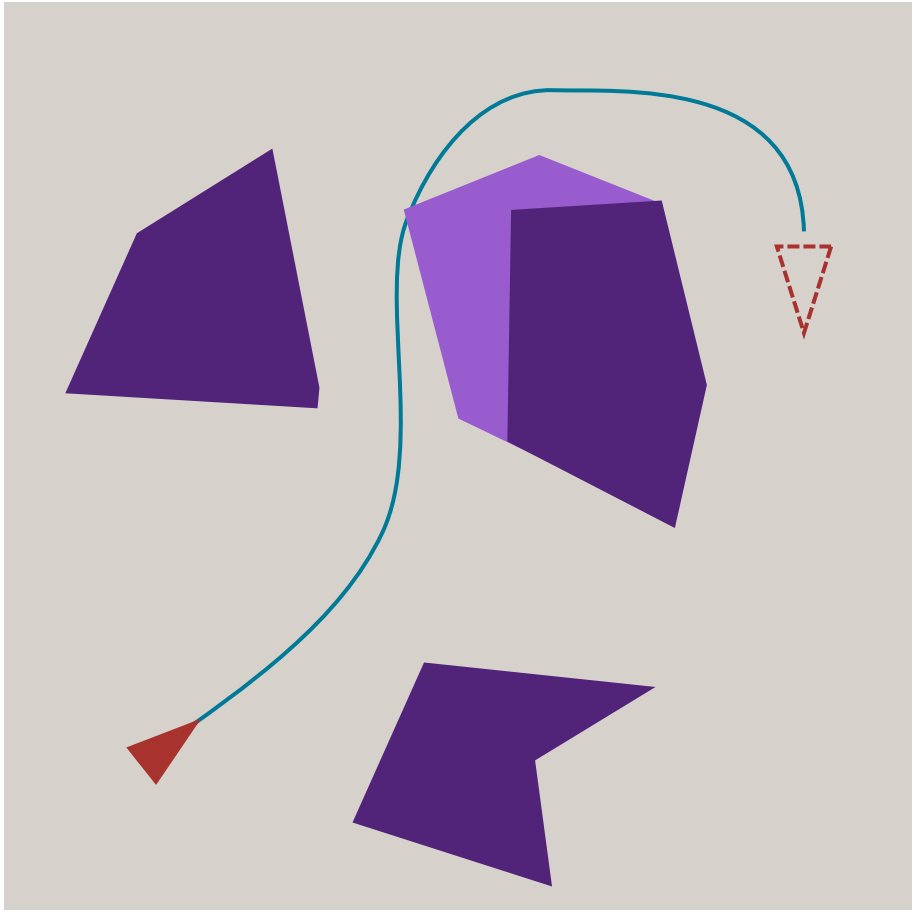
- Most focus of finding more optimal solutions faster, sometimes at the cost of formal guarantees

- Examples include:
  - Searching from start and goal and meshing the two trees
  - Limiting the search space once a plan has been found
  - Varying sampling density based on domain-knowledge
  - Computational improvements (collision checks, bootstrapping search, efficient data structures)

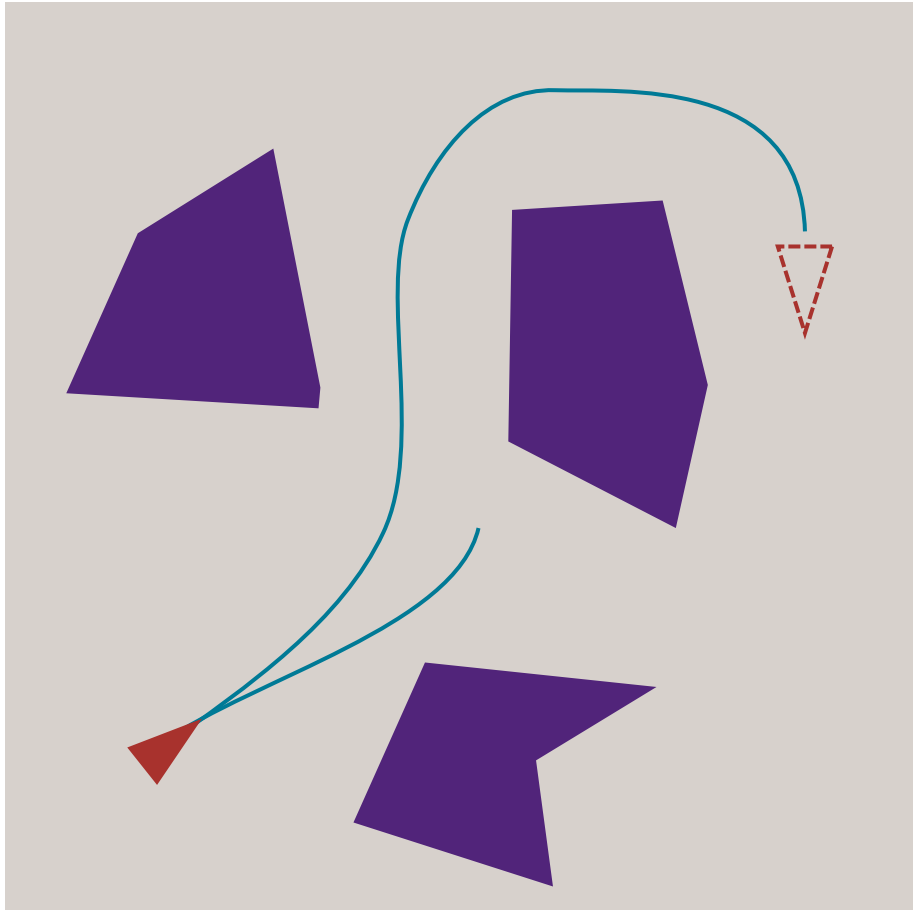
Planning Hierarchies

Sources of Uncertainty

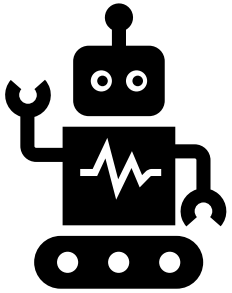
Environment uncertainty



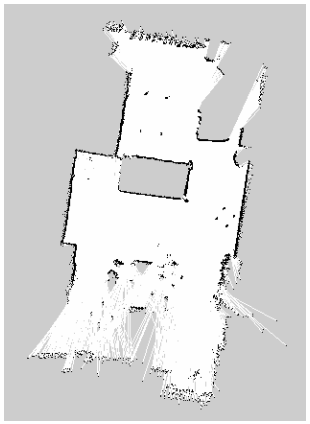
Motion uncertainty



Planning Hierarchy

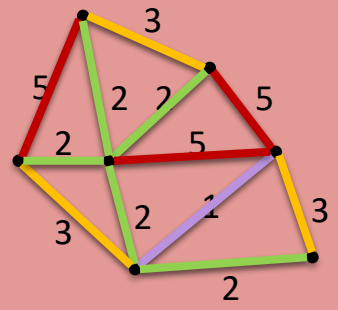


Map

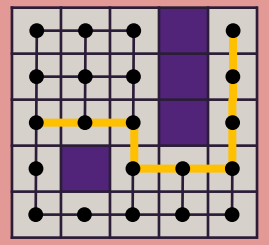


Sensor data

Graph



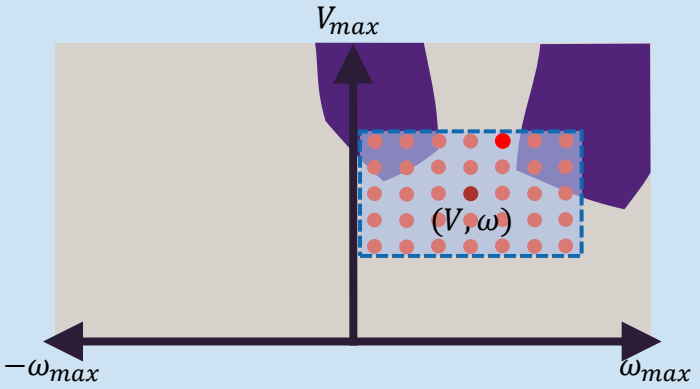
Search

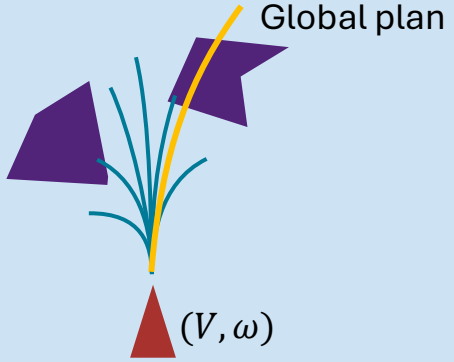


Difficult problem, low-frequency updates

GLOBAL PLANNER

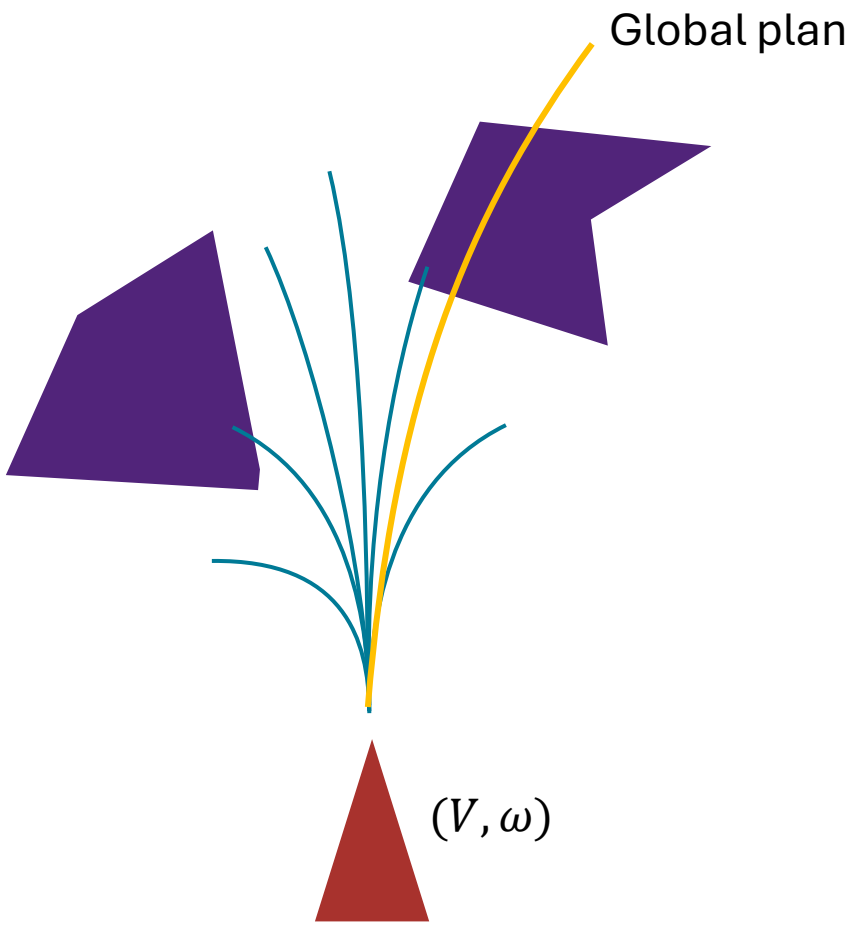
Easier problem, high-frequency updates





LOCAL PLANNER

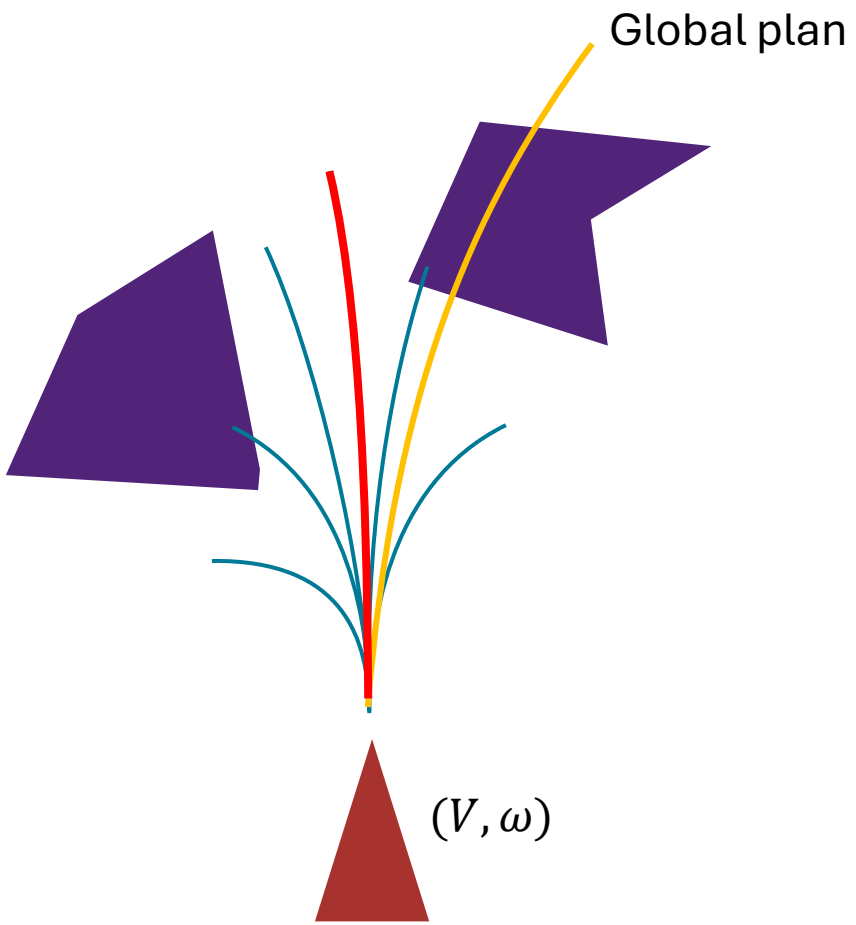
Dynamic Window Collision Avoidance

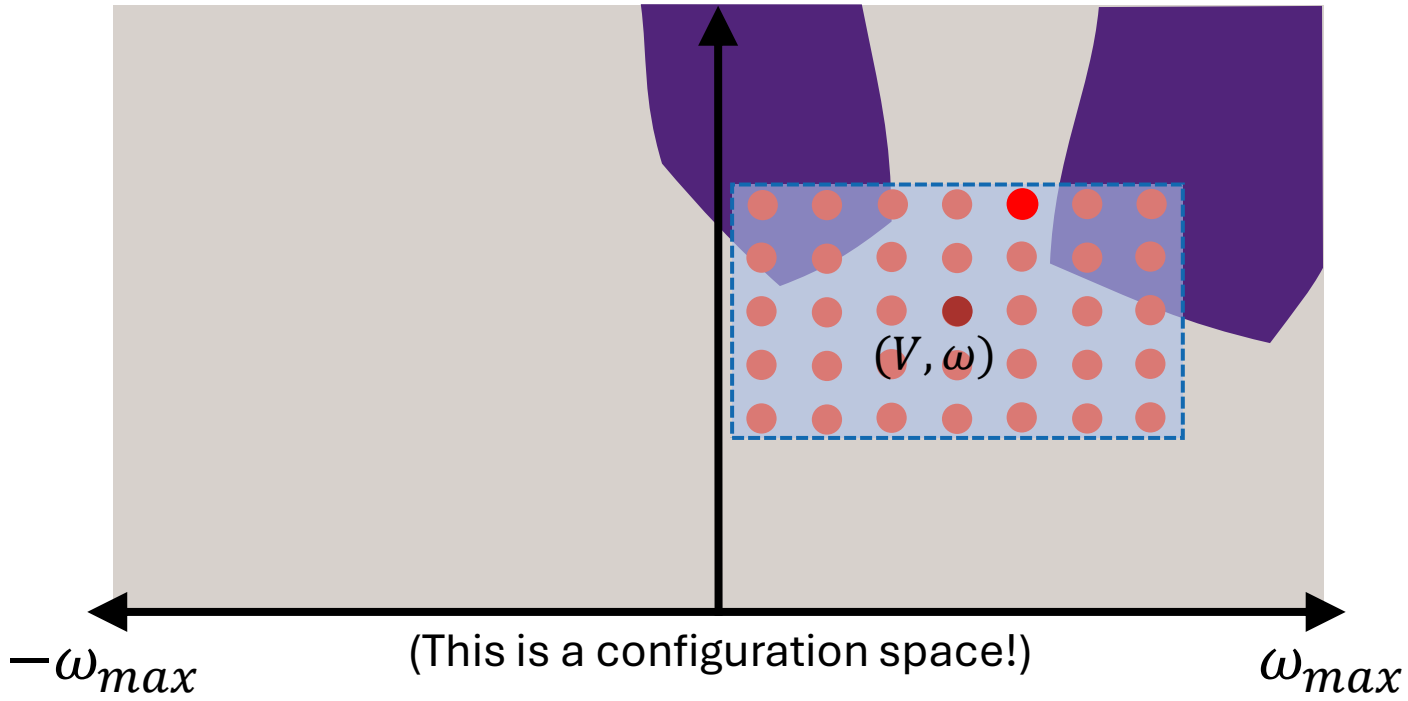


- Generating full time-varying trajectories for $V(t)$ and $\omega(t)$ is still very challenging
- If we assume (V, ω) are constant for a fixed Δt, each local path in the future is a circular arc segment
- This can be easily considered as a type of *velocity configuration space*

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, *4*(1), 23-33.

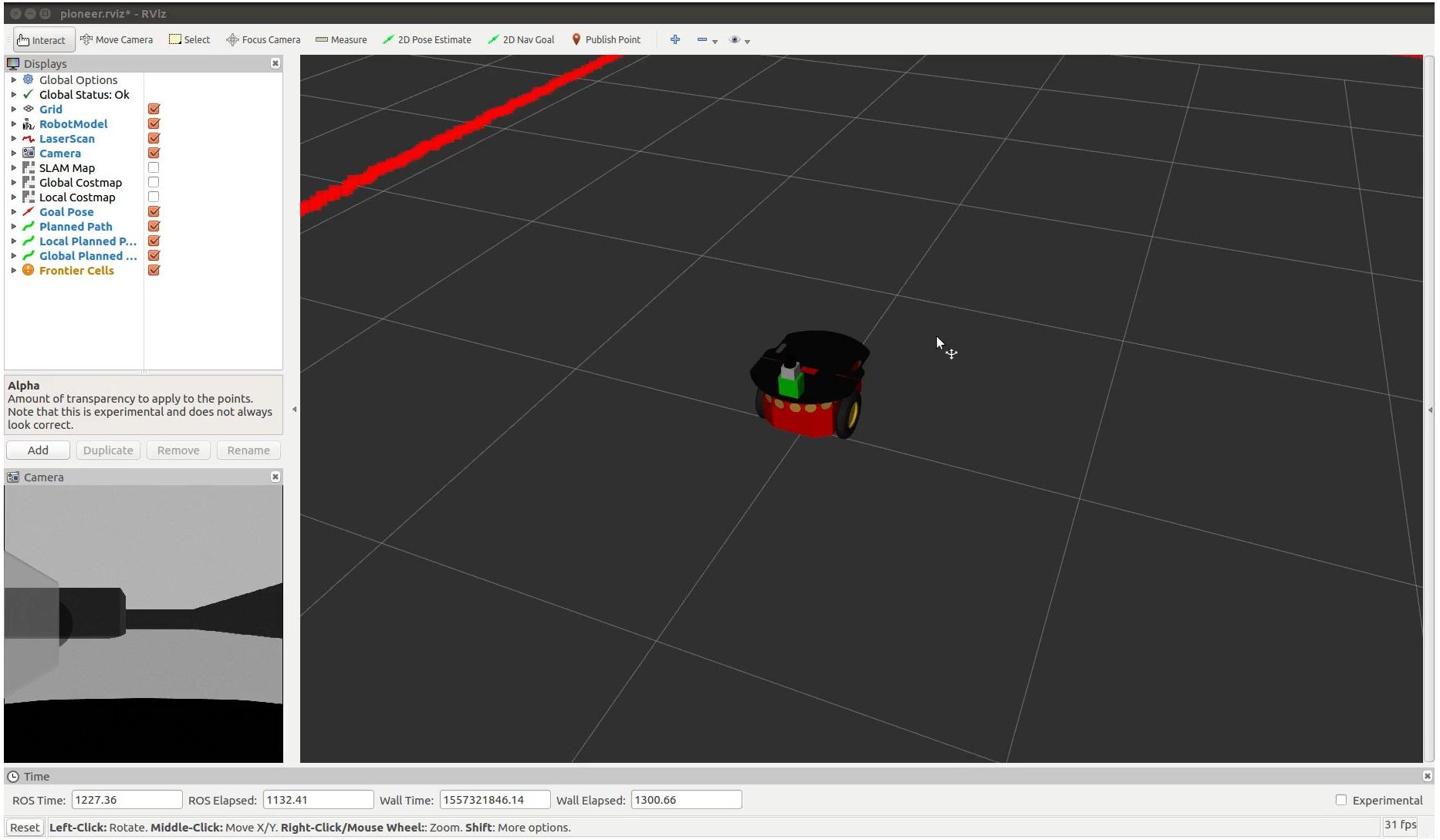
Dynamic Window Collision Avoidance





Maximise utility metric (usually maximise speed, minimise distance to goal, maximise distance from obstacles) across configuration samples

ROS Navigation Stack (A* global, DWA local)



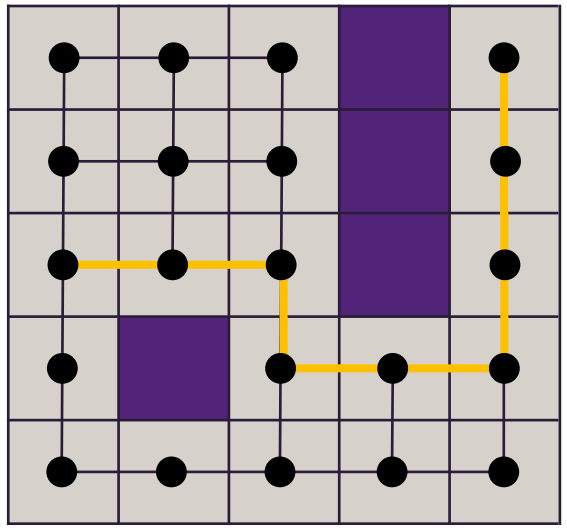
Summary: Motion Planning Essentials

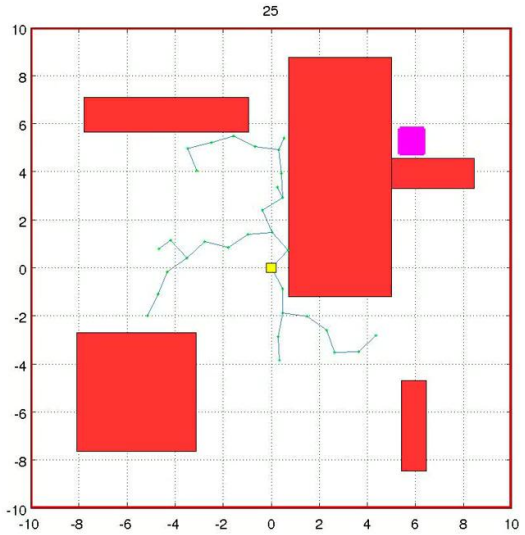
- Motion planning representations:
  - Workspace vs. configuration space

- Graphs to discretise the planning space
  - Graph search methods e.g. Dijkstra's algorithm (non-uniform edge weights), A* (cost-to-go heuristic for faster search)

- Sampling-based planning
  - PRM/PRM*, RRT/RRT*

- Hierarchies for global and local planning





AuSRoS

Australian School of Robotic Systems

B1: Motion Planning Essentials

A/Prof. Jen Jen Chung

The University Of Queensland
AUSTRALIA

With slides adapted from Dr Nicholas Lawrance