INTELLIGENT ROBOTS

CHAPTER 2: MOTION PLANNING AND COLLISION AVOIDANCE

Learning Objectives

1、What are task planning, path planning, and motion planning?

2、What is the potential field method?

3、What is collision avoidance ?

4、What is the dynamic window approach?

5、What is dynamic programming?

6、What is the Dijkstra’s algorithm?

7、What is the A* algorithm?

8、What is the 5D approach?

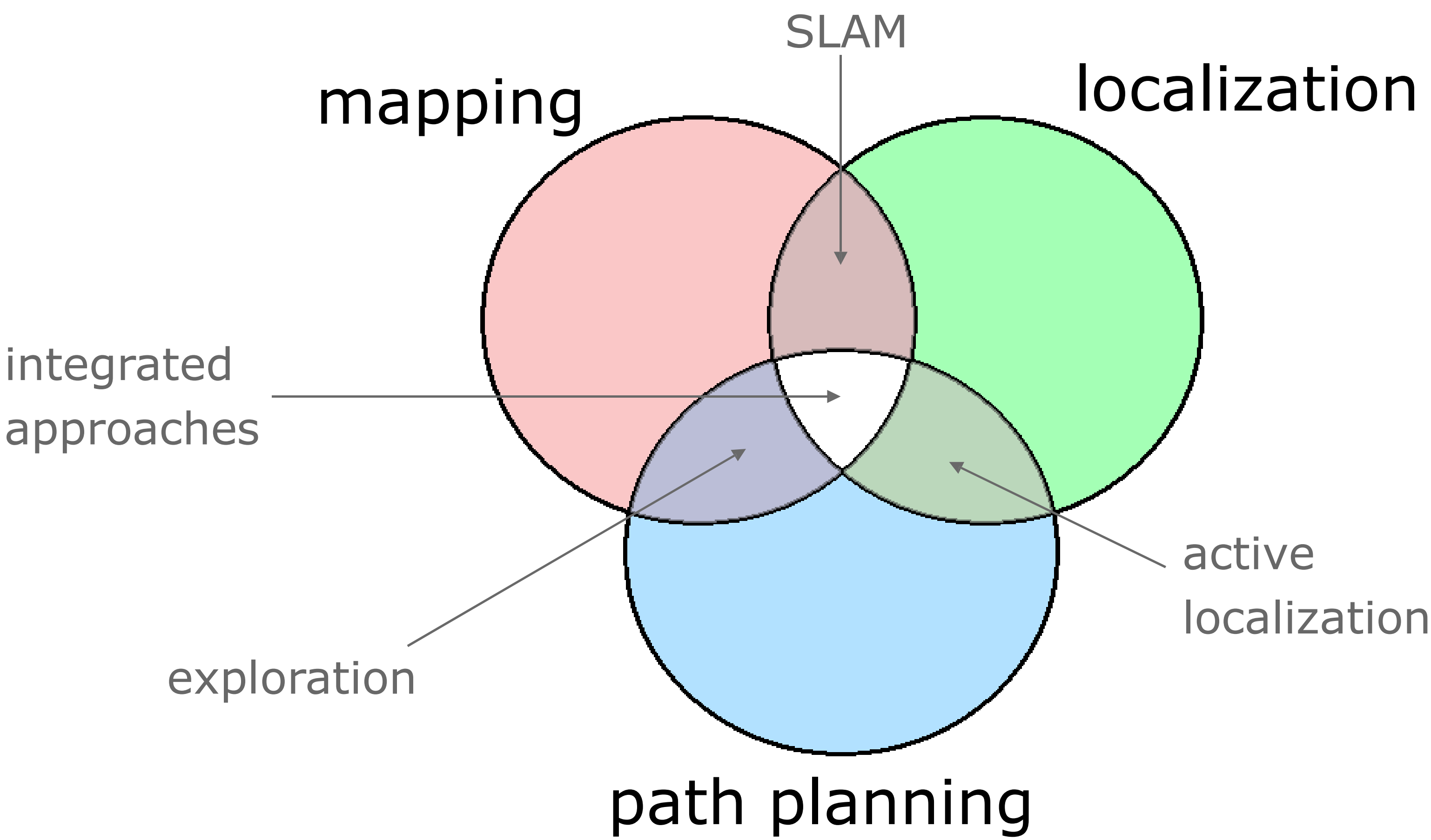
9、What is the Markov decision process?

10、What are value iteration and policy iteration?

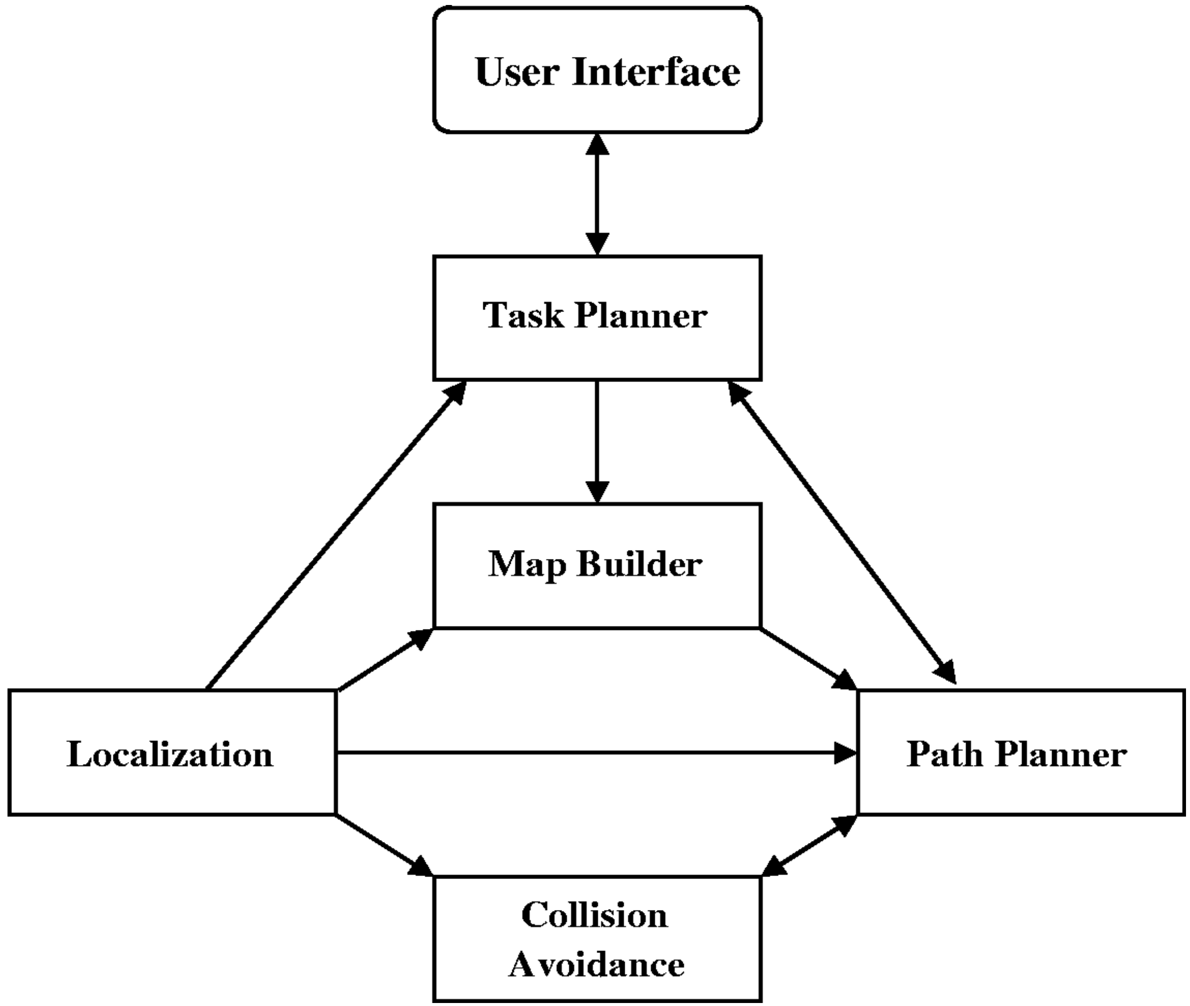
Outline

- Motion Planning Problem
- Potential Field Method
- Dynamic Window Approach
- A* Algorithm for Path Planning
- 5D Motion Planning
- Markov Decision Process

Tasks of Mobile Robots



Mobile Robot System



Motion Planning

Latombe (1991):

"…eminently necessary since, by definition, a robot accomplishes tasks by moving in the real world."

Goals:

- Collision-free trajectories.
- Robot should reach the goal location as fast as possible.

Dynamic Environment

- How to react to unforeseen obstacles?
  - efficiency
  - reliability

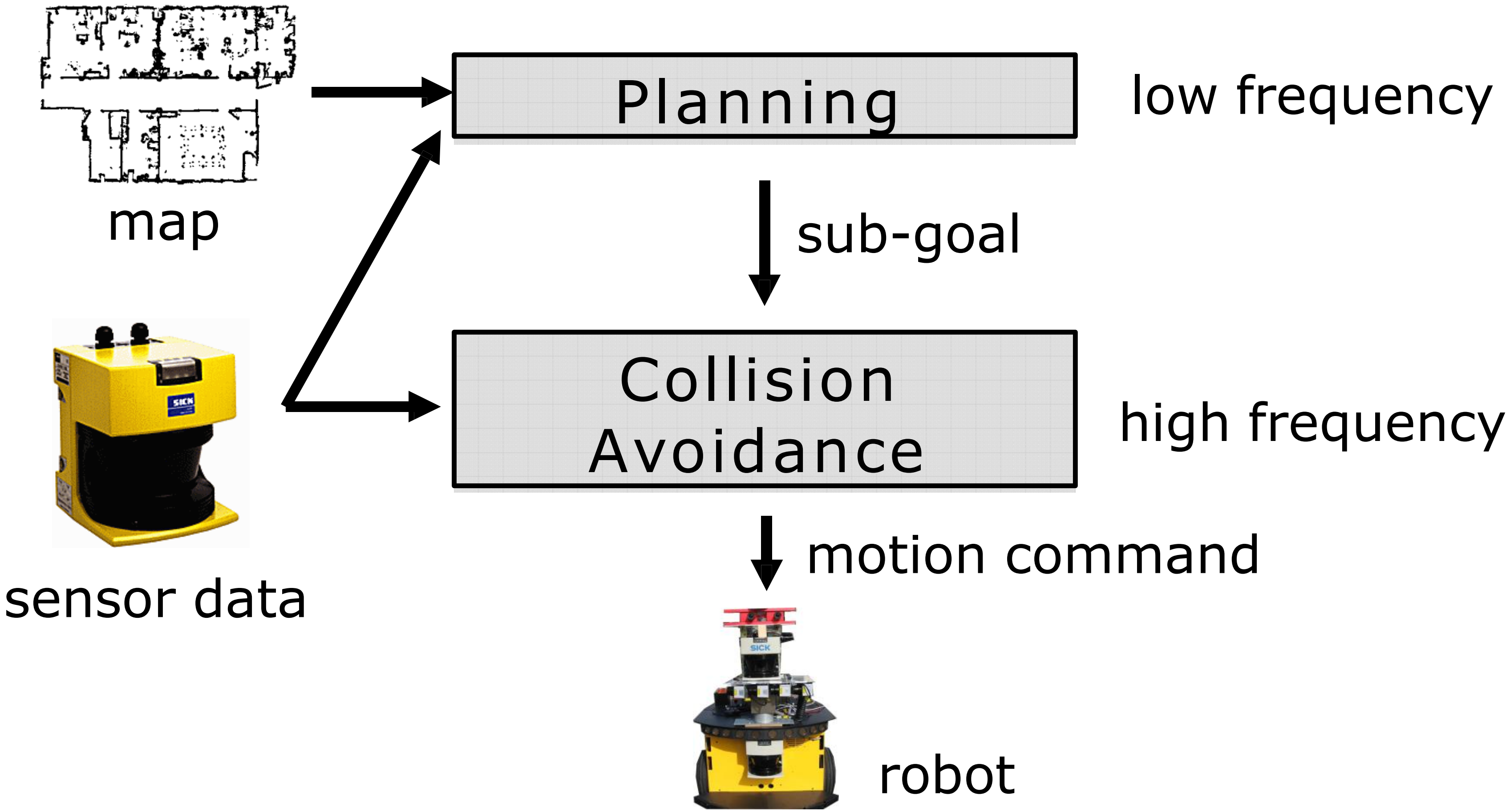
- Dynamic Window Approaches
  [Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]
- Grid map based Planning
  [Konolige, 00]
- Nearness Diagram Navigation
  [Minguez at al., 2001, 2002]
- Vector-Field-Histogram+
  [Ulrich & Borenstein, 98]
- A*, D*, D* Lite, ARA*, …
- Potential Field Method, Fuzzy Logic, Rapid Random Tree
- Markov Decision Process, Reinforcement Learning

Two Challenges

- Calculate the optimal path taking potential uncertainties in the actions into account

- Quickly generate actions in the case of unforeseen objects

Classic Two Layered Architecture



Approaches

- Optimization Based Planning
  - Dynamic window approach (collision avoidance)
  - A*, D*, Lite D* (path planning)
  - 5D planning (motion planning)

- Behavior Based Planning (motion planning)
  - Potential Field Method
  - Markov Decision Process
  - Partially Observable MDP (POMDP)
  - Reinforcement Learning
    neural networks for policy and value

Outline

- Motion Planning Problem
- Potential Field Method
- Dynamic Window Approach
- A* Algorithm for Path Planning
- 5D Motion Planning
- Markov Decision Process

Robotics Paradigms

Classical Robotics (mid-70’ s)
- exact models
- no sensing necessary

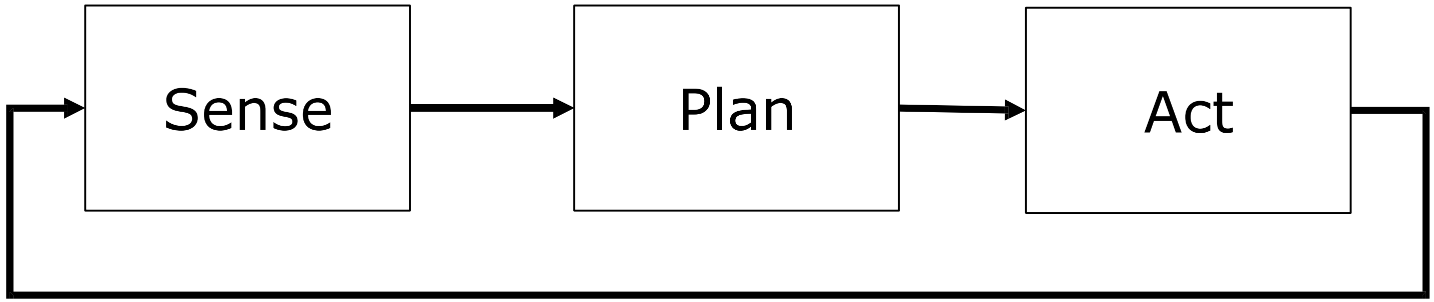
Reactive Paradigm (mid-80’ s)
- no models
- relies heavily on good sensing

Hybrids (since 90’ s)
- model-based at higher levels
- reactive at lower levels

Probabilistic Robotics (since mid-90’ s)
- integration of models and sensing
- inaccurate models, inaccurate sensors

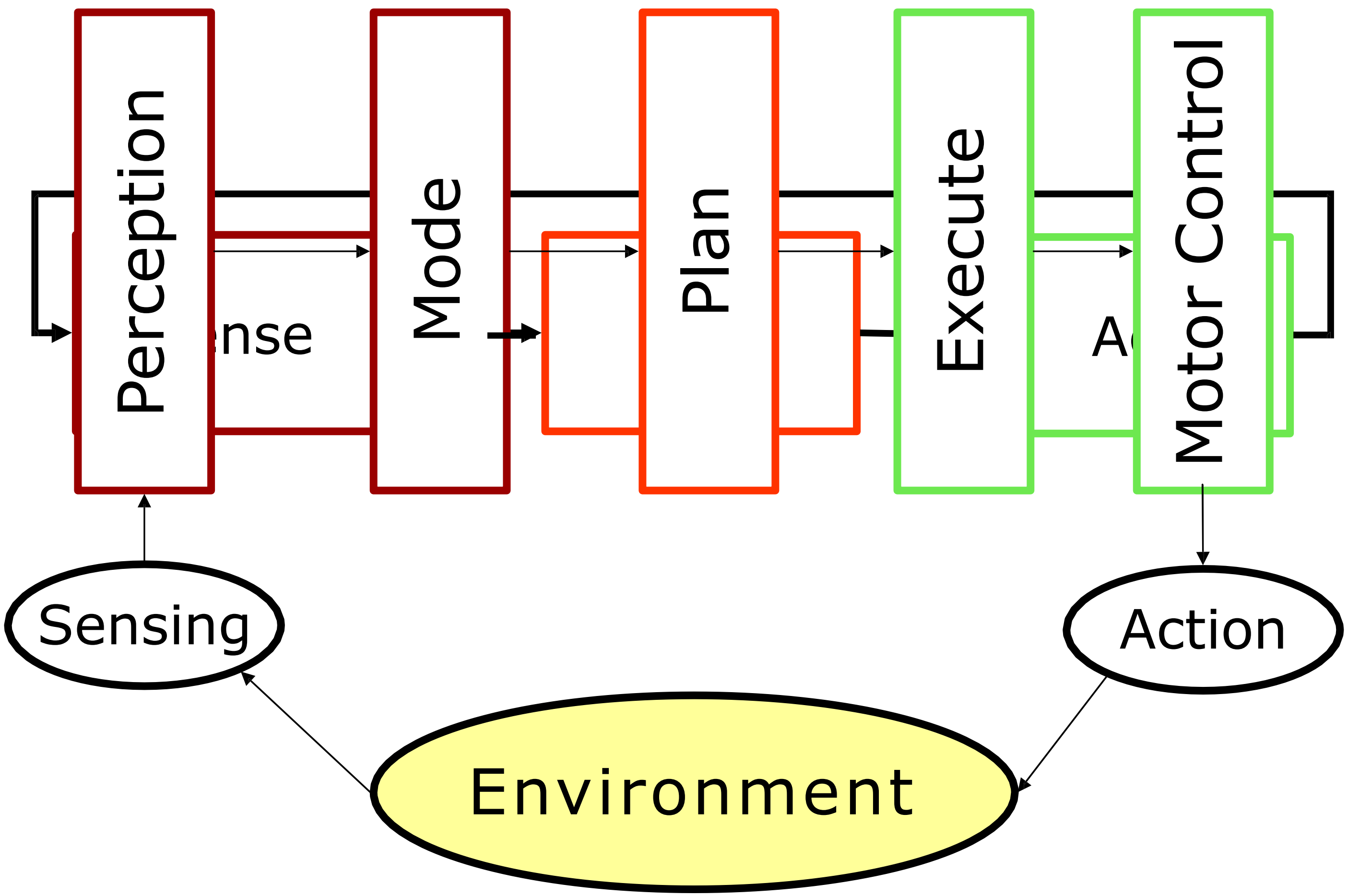
Intelligent Robotics (since 00’ s)
- deep learning and deep reinforcement learning
- vision based sensing

Classical / Hierarchical Paradigm

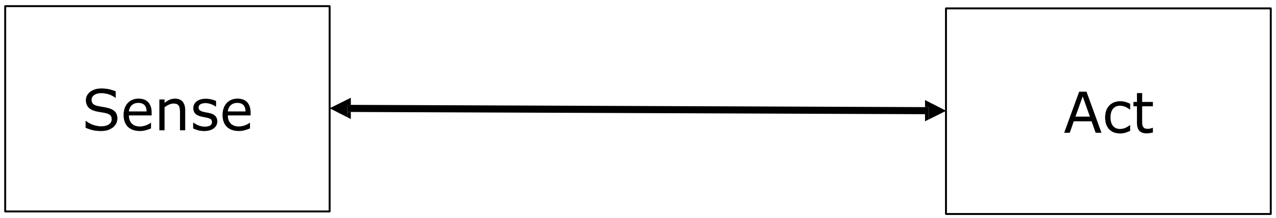


- 70's
- Focus on automated reasoning and knowledge representation
- STRIPS (Stanford Research Institute Problem Solver): Perfect world model, closed world assumption
- Find boxes and move them to designated position

Classical Paradigm: Horizontal Decomposition

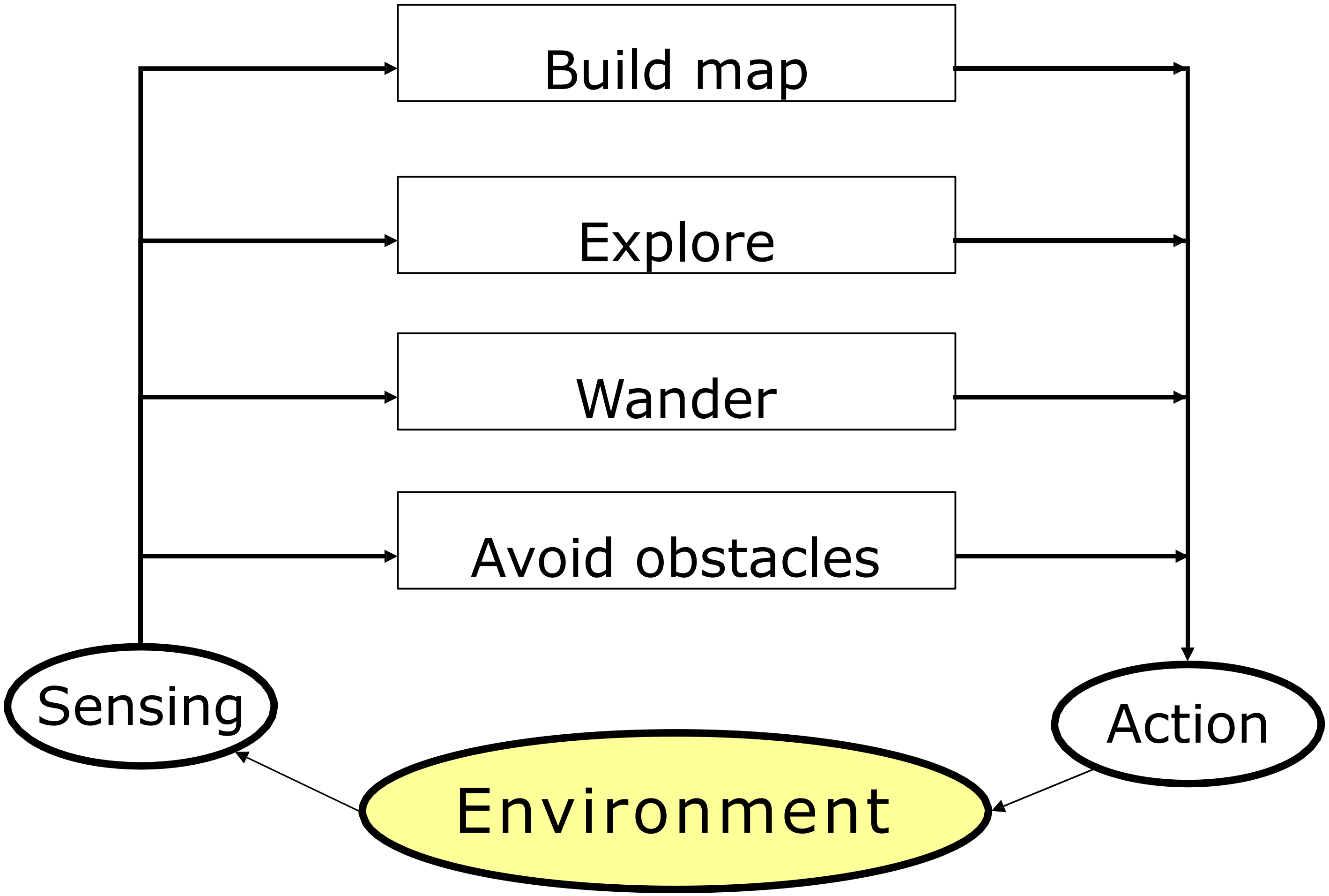


Reactive / Behavior-based Paradigm

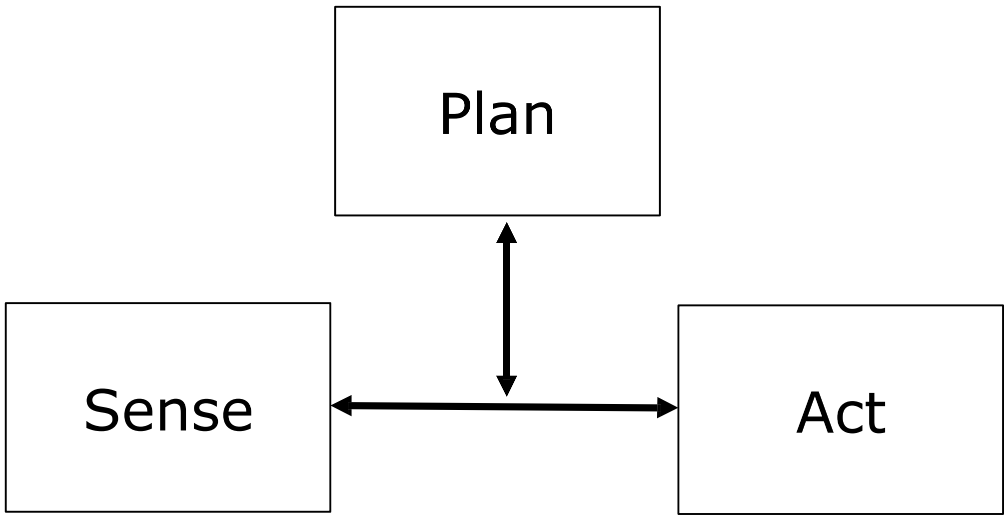


- No models: The world is its own, best model
- Easy successes, but also limitations
- Investigate biological systems

Reactive Paradigm: Vertical Decomposition



Hybrid Deliberative/Reactive Paradigm



- Combines advantages of previous paradigms
  - World model used for planning
  - Closed loop, reactive control

Characteristics of Reactive Paradigm

- Situated agent, robot is an integral part of the world.
- No memory, controlled by what is happening in the world.
- Tight coupling between perception and action via behaviors.
- Only local, behavior-specific sensing is permitted (ego-centric representation).

Behaviours

- a direct mapping of sensory inputs to a pattern of motor actions that are then used to achieve a task.

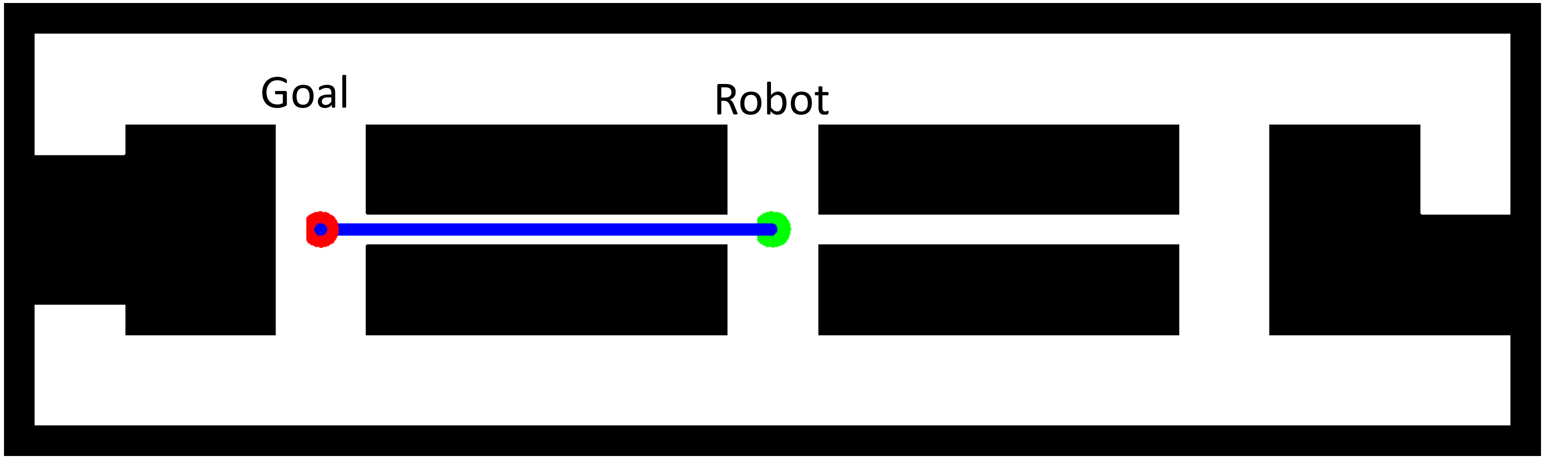
- serve as the basic building block for robotics actions, and the overall behavior of the robot is emergent.

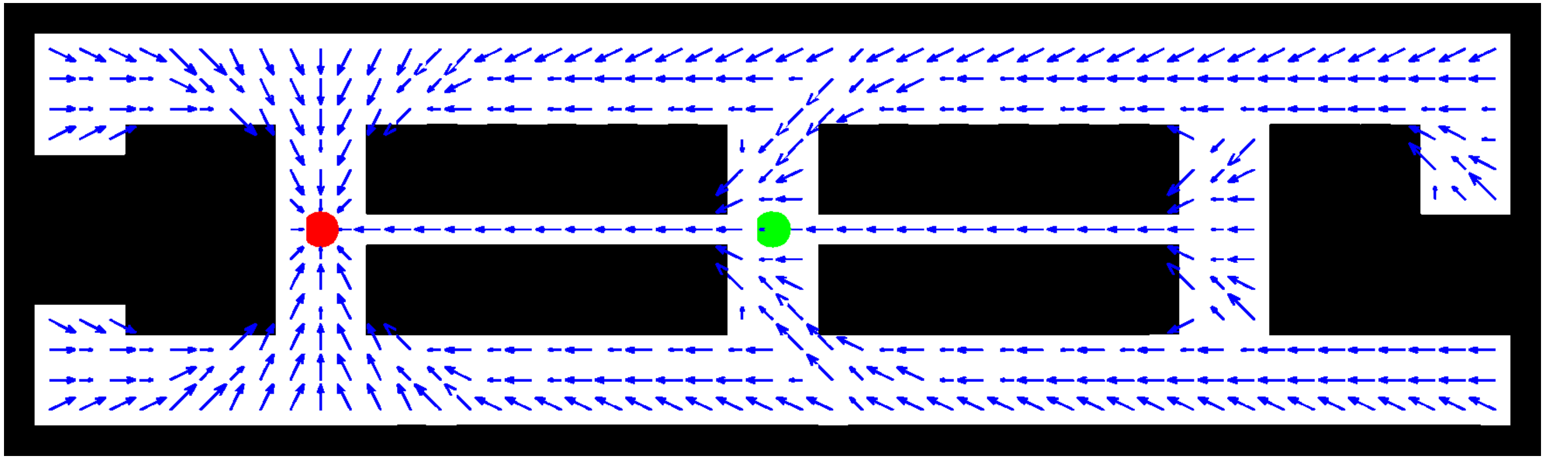
- support good software design principles due to modularity.

Potential Field Methodologies

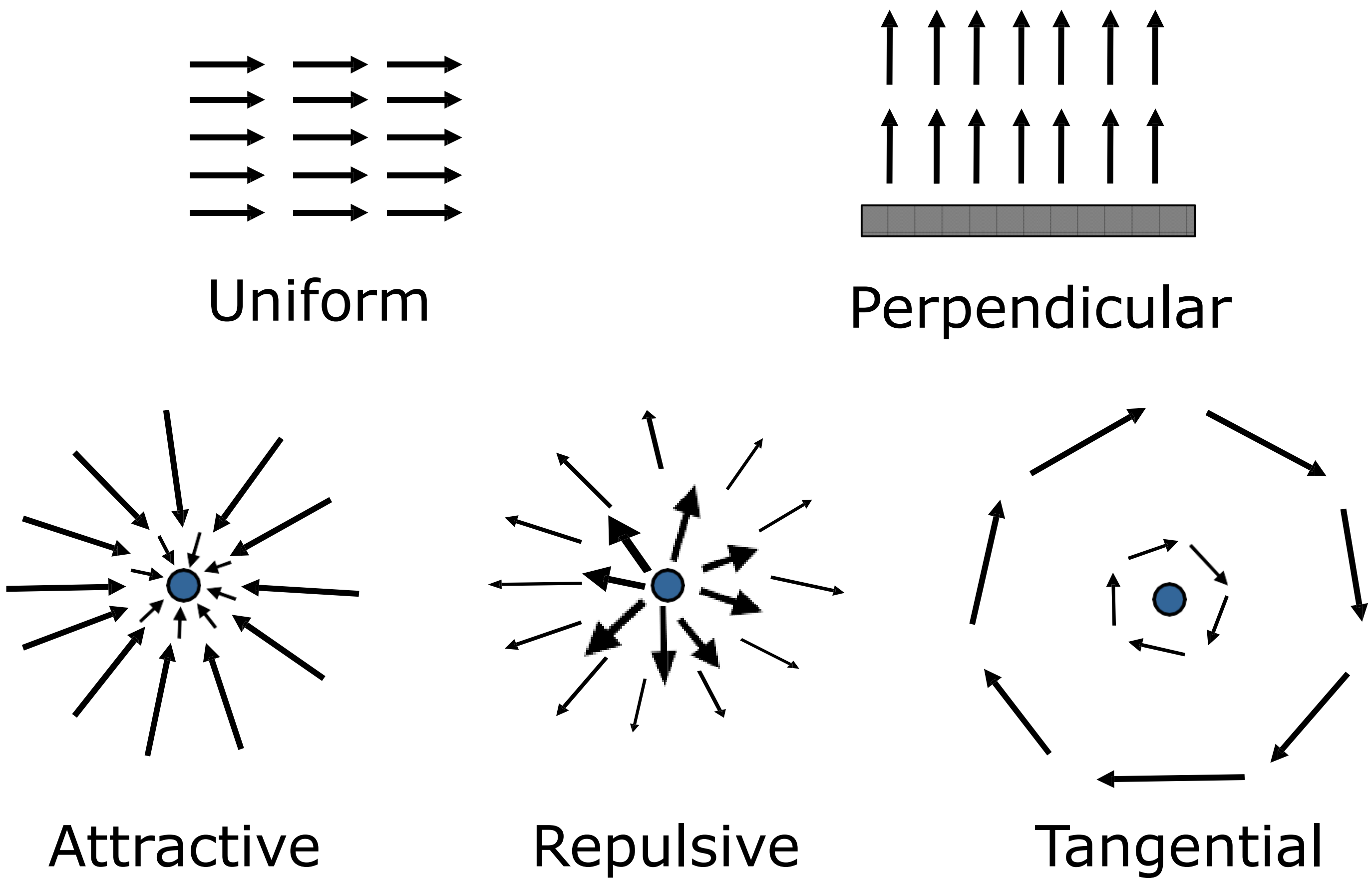
- Treat robot as a particle acting under the influence of a potential field
- Robot travels along the derivative of the potential
- Field depends on obstacles, desired travel directions and targets
- Resulting field (vector) is given by the summation of primitive fields
- Strength of field may change with distance to obstacle/target
- **A potential field is equivalent to a velocity field**

Robot Navigation Problem



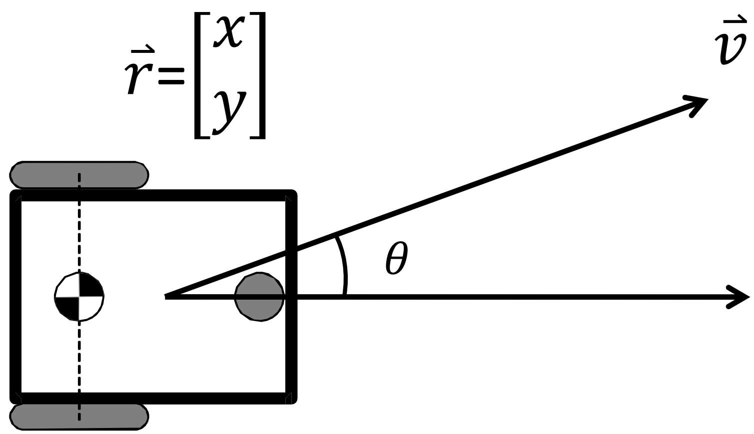


Primitive Potential Fields

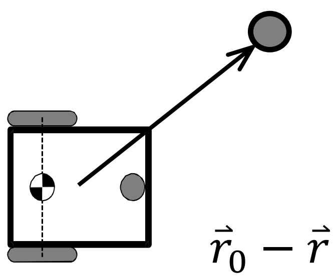


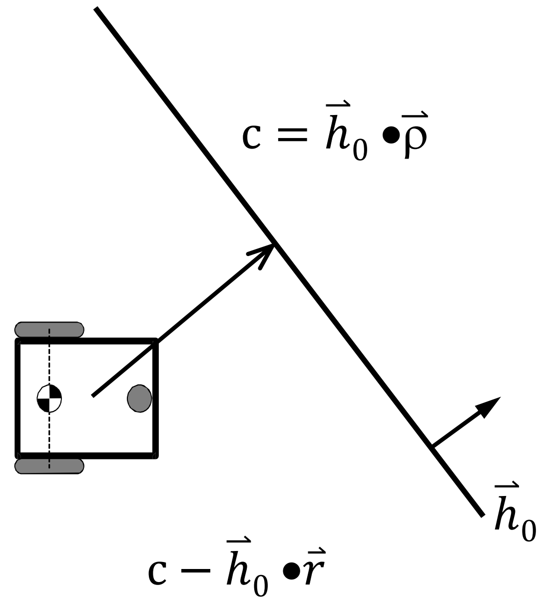
Robot Model and Parameters

$$\boxed{\vec{r}\,' = \vec{r} + \vec{v}\Delta t}$$



Robot-to-Object Distance

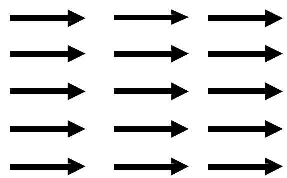




Distance between
the robot and the point

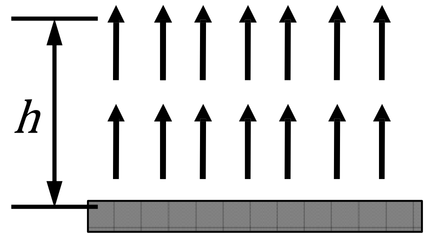
Distance between
the robot and the line

Primitive Potential (Velocity) Fields



$\vec{v} = \vec{v}_0$

Uniform



$\vec{v} = \eta \vec{h} \dfrac{v_p}{h^3}$

Perpendicular

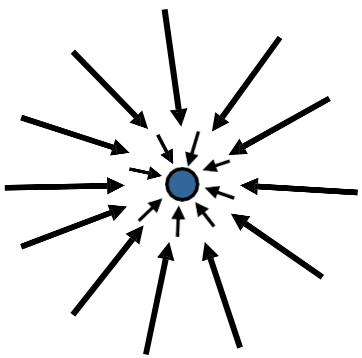
ξ is the attractive gain

η is the repulsive gain

$v = v_0$

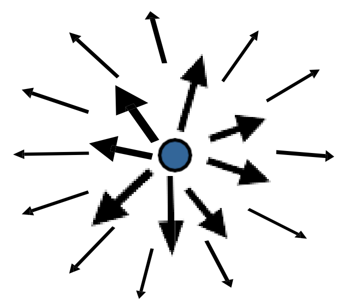
$\theta \sim \mathrm{rand}[0, 360^o]$

Random



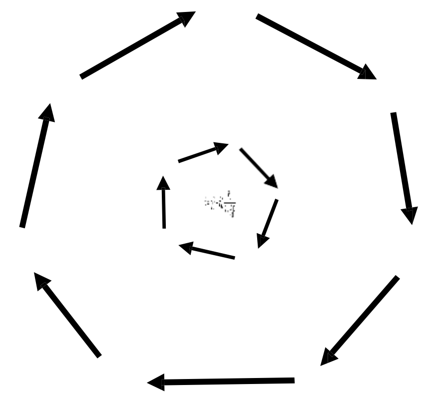
$\vec{v} = \xi(\vec{r}_0 - \vec{r})|\vec{r} - \vec{r}_0|v_0$

Attractive



$\vec{v} = \eta(\vec{r} - \vec{r}_0)\dfrac{v_0}{|\vec{r} - \vec{r}_0|^3}$

Repulsive



$\vec{v} = \vec{\omega}_0 \times (\vec{r} - \vec{r}_0)$

Tangential

Corridor Following With Potential Fields

- Level 0 (collision avoidance)
  is done by the repulsive fields of detected obstacles.

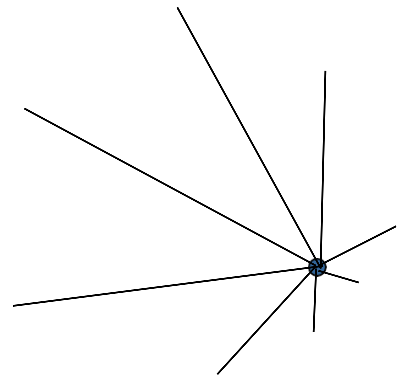
- Level 1 (wander)

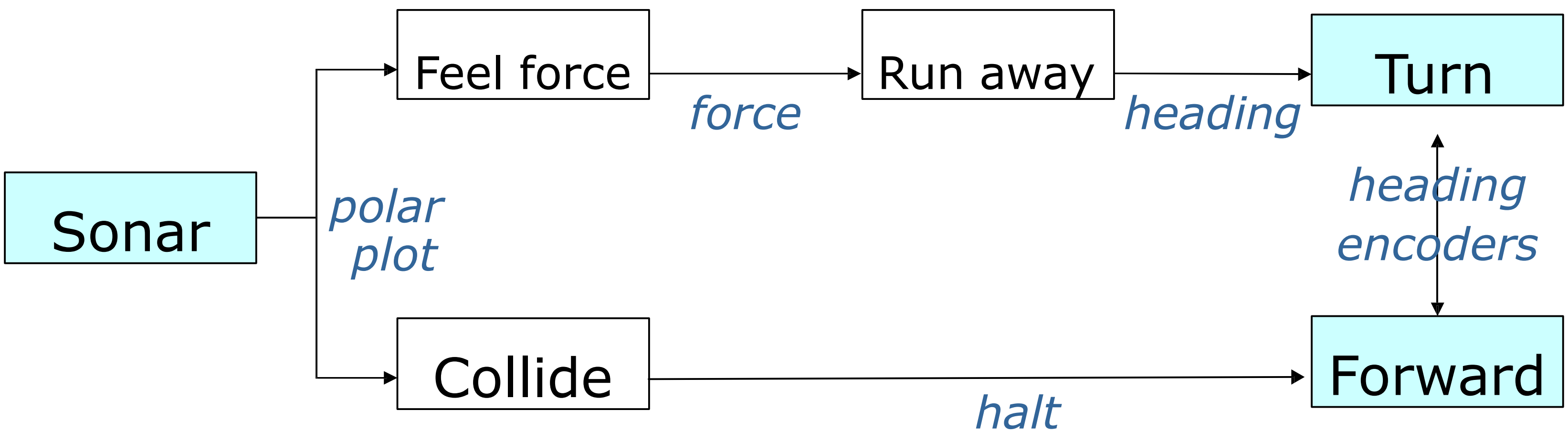
  adds a uniform field.

- Level 2 (corridor following)
  replaces the wander field by three fields (two perpendicular, one uniform).

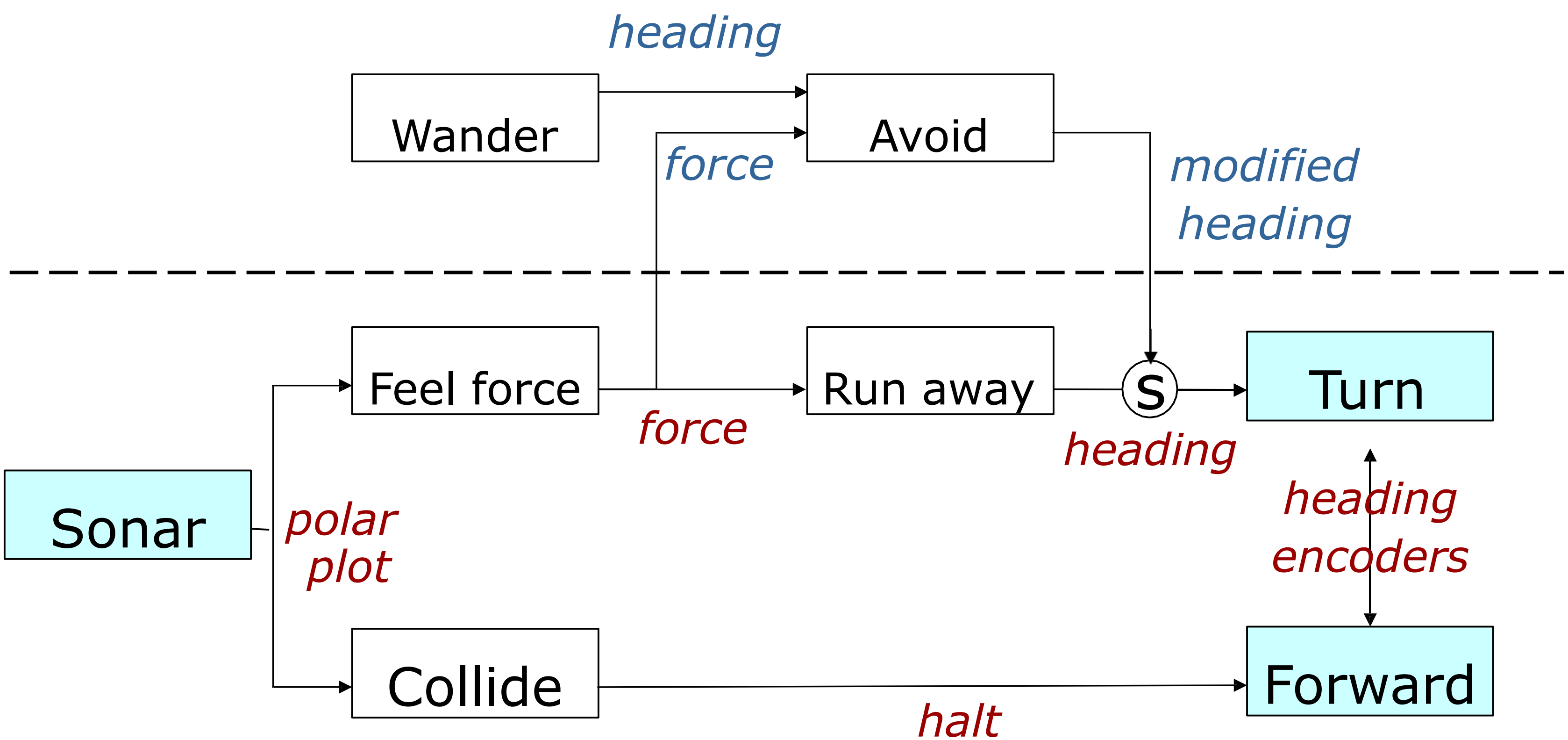
Level 0: Avoid

Polar plot of sonars

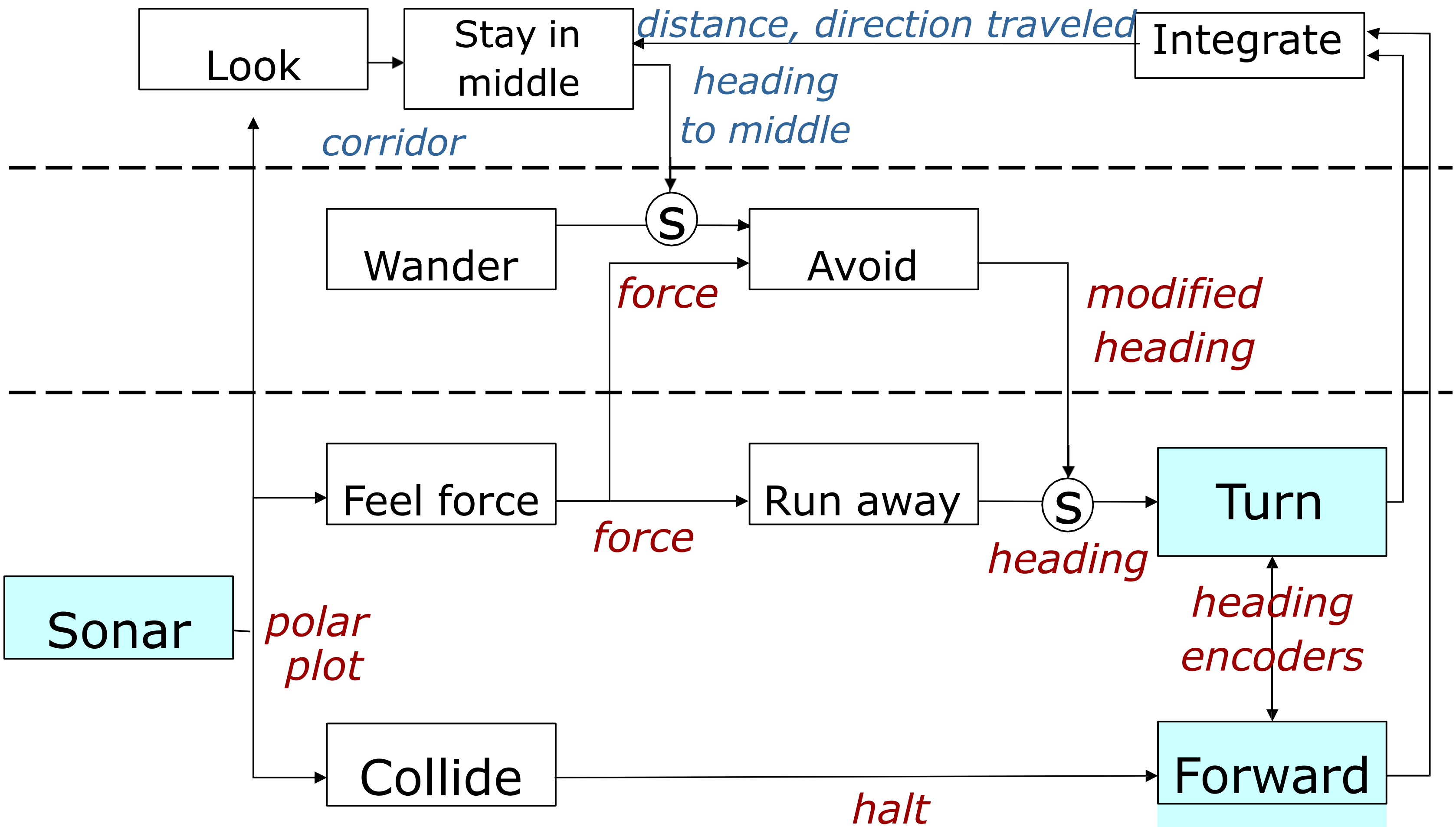




Level 1: Wander

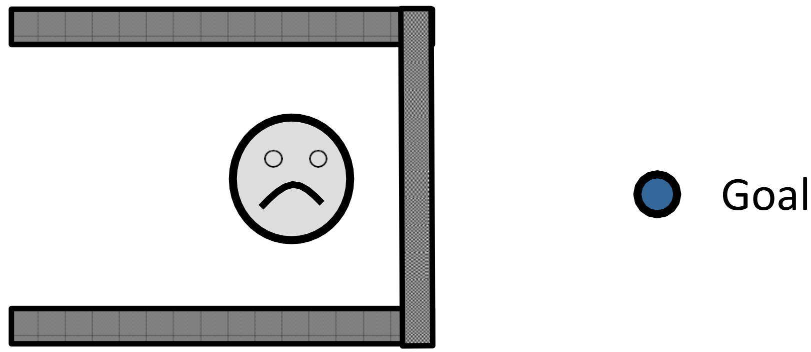


Level 2: Follow Corridor



Characteristics of Potential Fields

- Suffer from local minima



- Backtracking(Enumeration method)
- Random motion to escape local minimum
- Procedural planner such as wall following
- Increase potential of visited regions
- Avoid local minima by harmonic functions

Characteristics of Potential Fields

- No preference among layers
- Easy to visualize
- Easy to combine different fields
- High update rates necessary
- Parameter tuning important

Outline

- Motion Planning Problem
- Potential Field Method
- Dynamic Window Approach
- A* Algorithm for Path Planning
- 5D Motion Planning
- Markov Decision Process

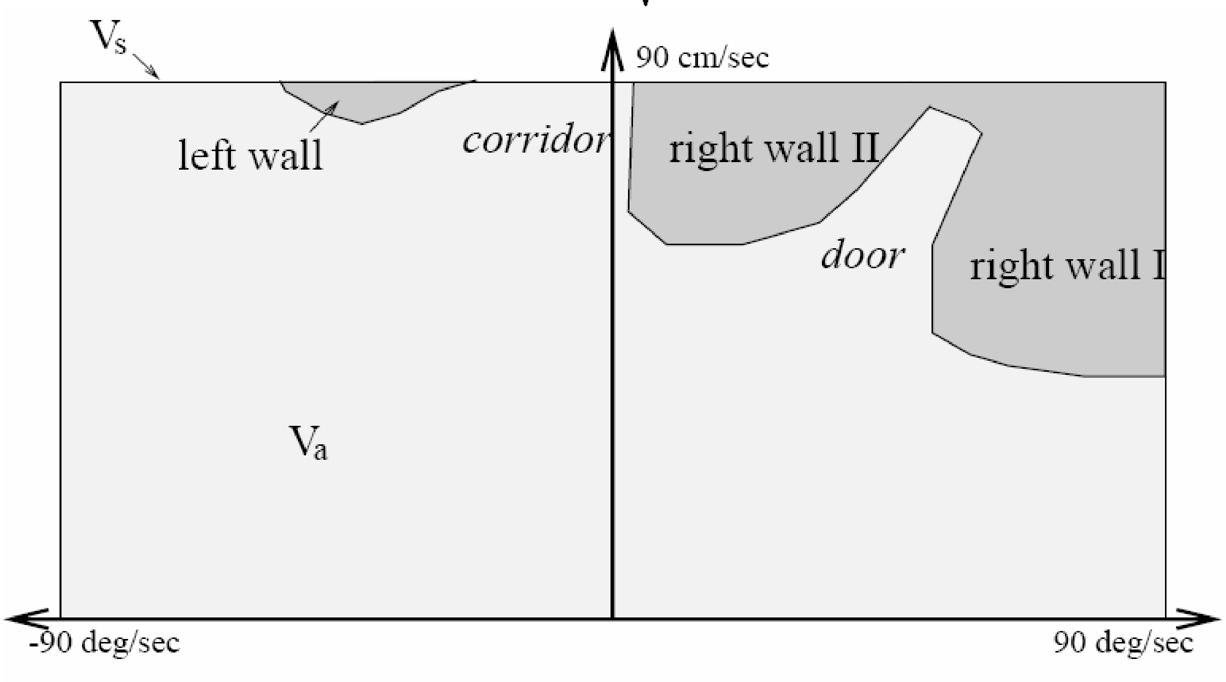
Dynamic Window Approach

- Collision avoidance: determine collision-free trajectories using geometric operations
- Here: robot moves on circular arcs
- Motion commands (v,ω)
- Which (v,ω) are admissible and reachable?

Admissible Velocities

- Speeds are admissible if

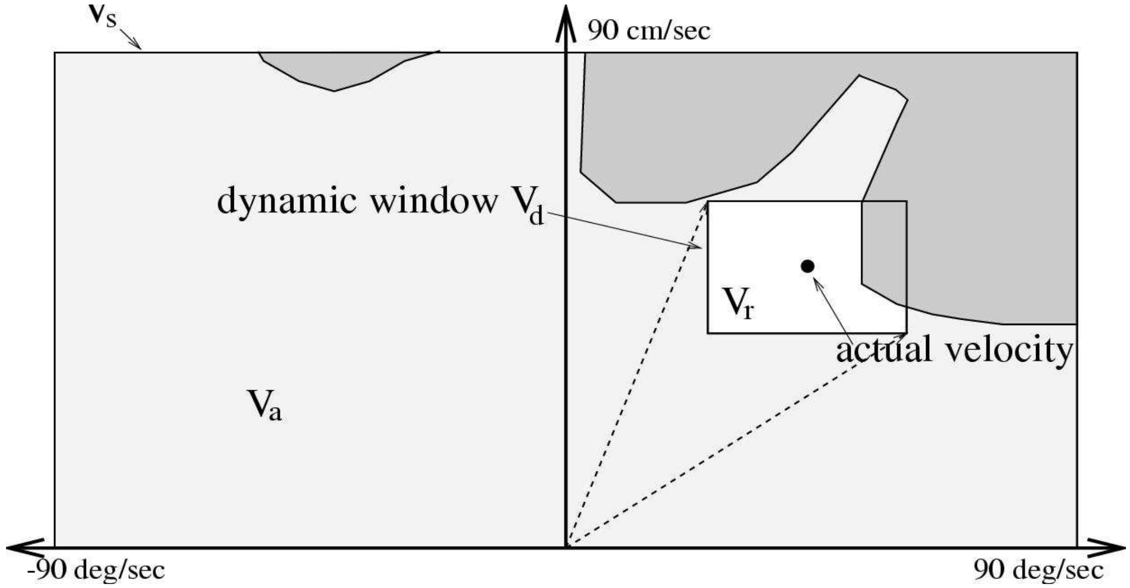
$$V_a = \{(v,\omega) \;\; | \;\; v \leq \sqrt{2\text{dist}(v,\omega)a_{trans}} \;\; \wedge$$
$$\omega \leq \sqrt{2\text{dist}(v,\omega)a_{rot}}\}$$



Reachable Velocities

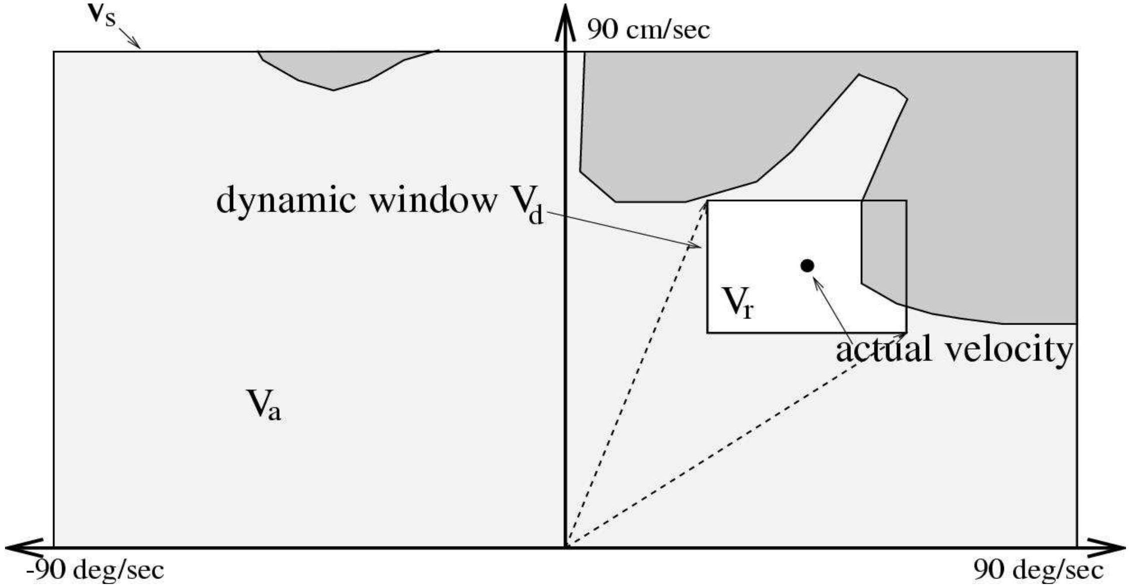
- Speeds are admissible if

$$V_d = \{(v, \omega) \ \mid \ v \in [v - a_{trans}t, v + a_{trans}t] \ \wedge \\ \omega \in [\omega - a_{rot}t, \omega + a_{rot}t]\}$$



DWA Search Space

- Example search-space:

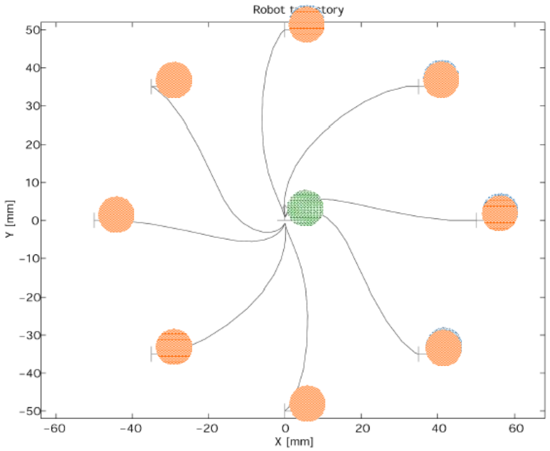


  - V_s = all possible speeds of the robot.
  - V_a = obstacle free area.
  - V_d = speeds reachable within a certain time frame based on possible accelerations.

$$Space = V_s \cap V_a \cap V_d$$

Dynamic Window Approach

- How to choose <v,ω>?
- Steering commands are chosen by a heuristic navigation function.
- This function tries to minimize the travel-time by: "driving fast in the right direction."



which one is optimal?

Dynamic Window Approach

- Heuristic navigation function.
- Planning restricted to <x,y>-space.
- No planning in the velocity space.

Navigation Function: [Brock & Khatib, 99]

$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Dynamic Window Approach

- Heuristic navigation function.
- Planning restricted to <x,y>-space.
- No planning in the velocity space.

Navigation Function: [Brock & Khatib, 99]

$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Maximizes velocity.

Dynamic Window Approach

- Heuristic navigation function.
- Planning restricted to <x,y>-space.
- No planning in the velocity space.

Navigation Function: [Brock & Khatib, 99]

$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Maximizes velocity.

Considers cost to reach the goal.

Dynamic Window Approach

- Heuristic navigation function.
- Planning restricted to <x,y>-space.
- No planning in the velocity space.

Navigation Function: [Brock & Khatib, 99]

$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Maximizes velocity.

Considers cost to reach the goal.

Follows grid based path computed by A*

Dynamic Window Approach

- Heuristic navigation function.
- Planning restricted to <x,y>-space.
- No planning in the velocity space.

Goal Nearness

Navigation Function: [Brock & Khatib, 99]

$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Maximizes velocity.

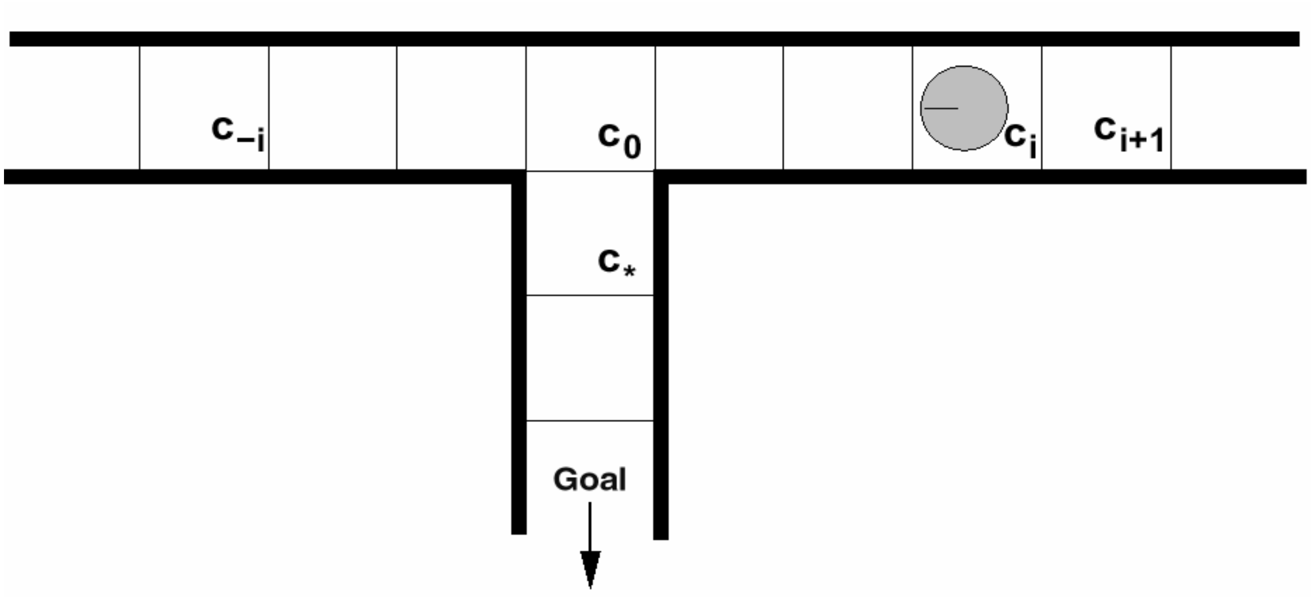
Considers cost to reach the goal.

Follows grid based path computed by A*

Dynamic Window Approach

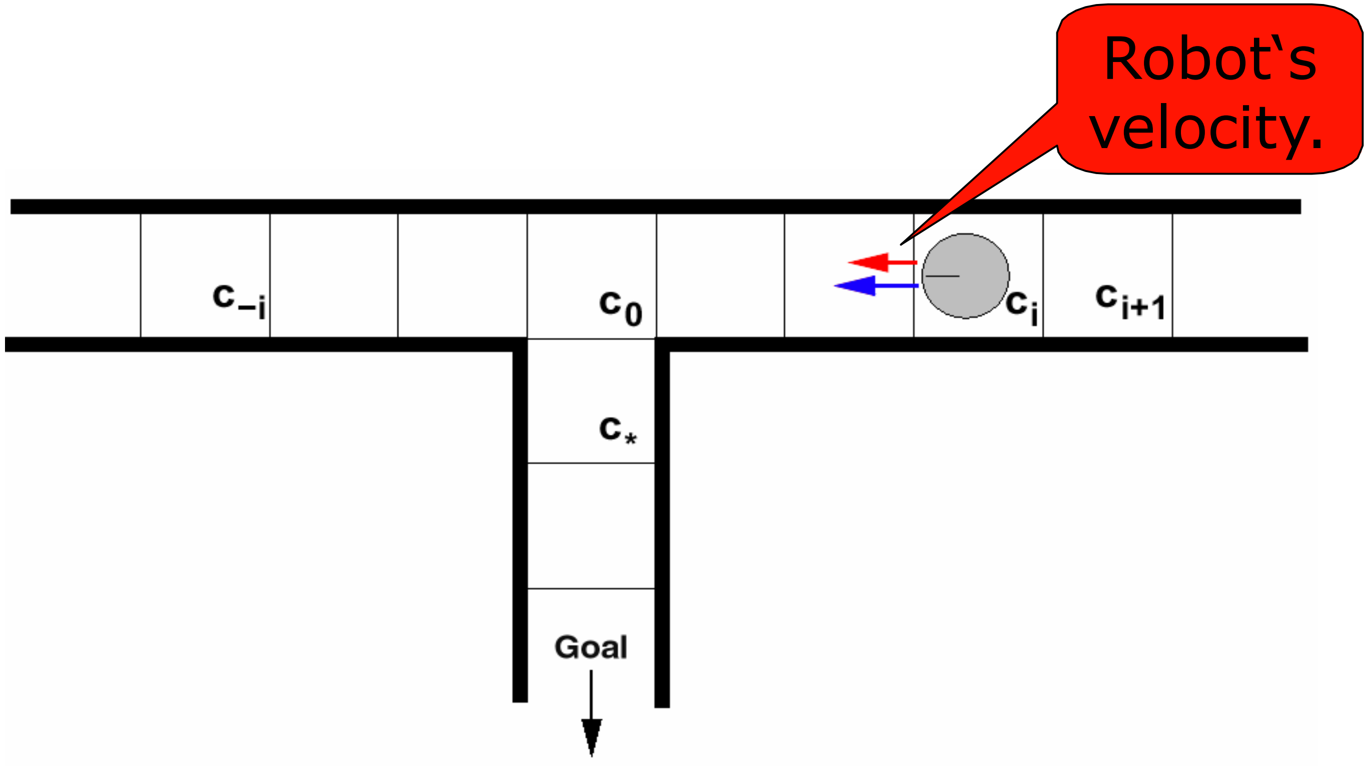
- Reacts quickly.
- Low CPU power requirements.
- Guides a robot on a collision free path.
- Successfully used in a lot of real-world scenarios.
- Resulting trajectories sometimes sub-optimal.
- Local minima might prevent the robot from reaching the goal location.

Problems of DWAs



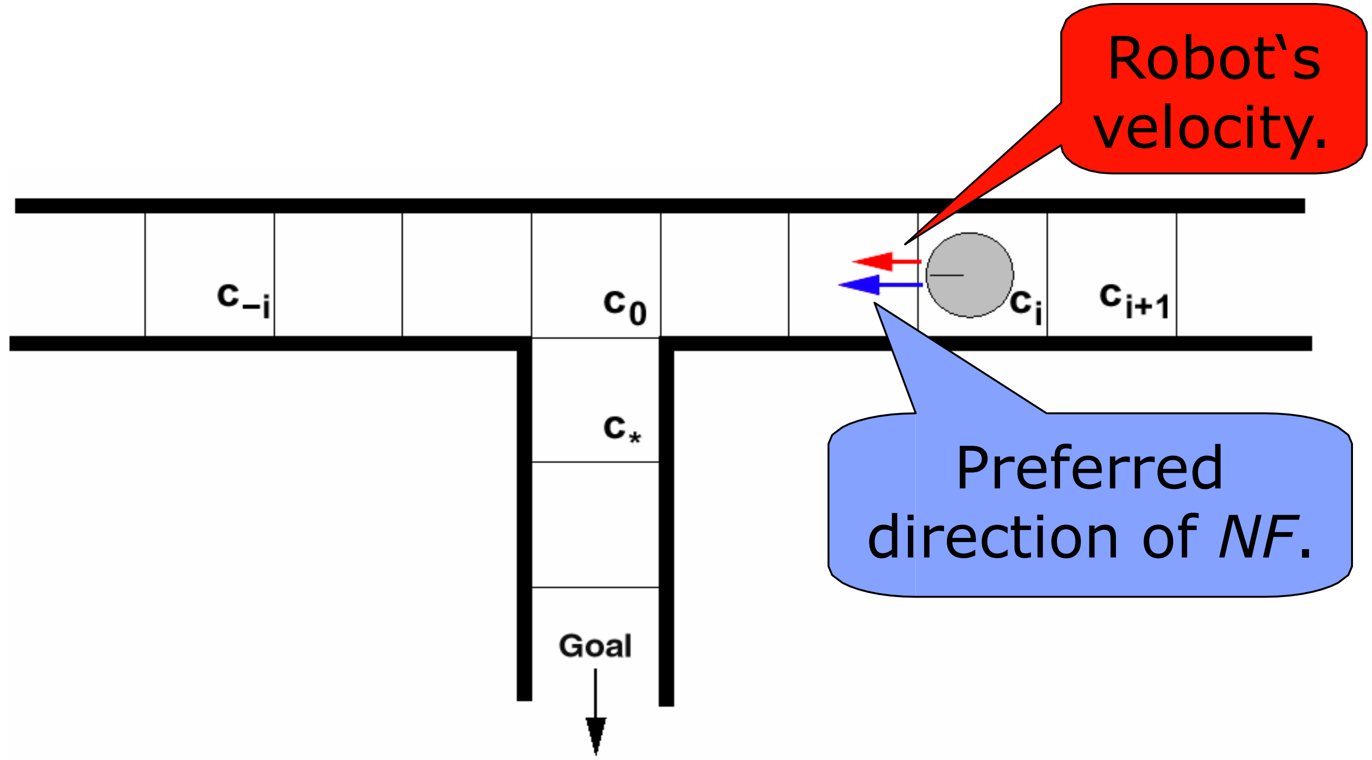
$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Problems of DWAs



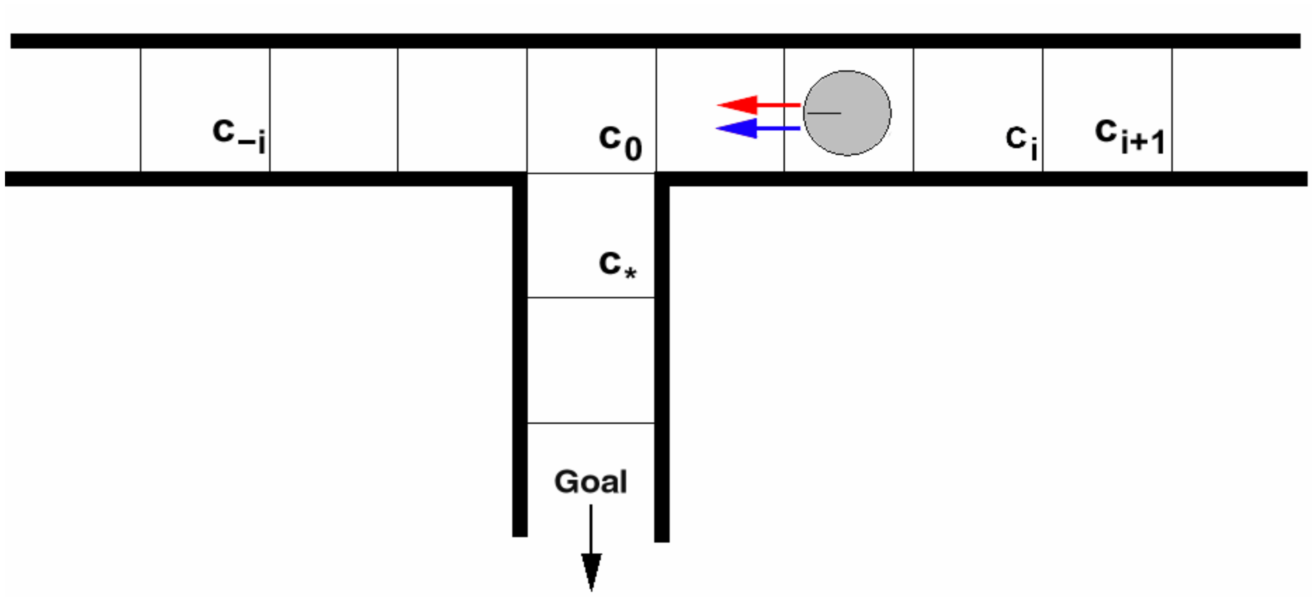
$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Problems of DWAs



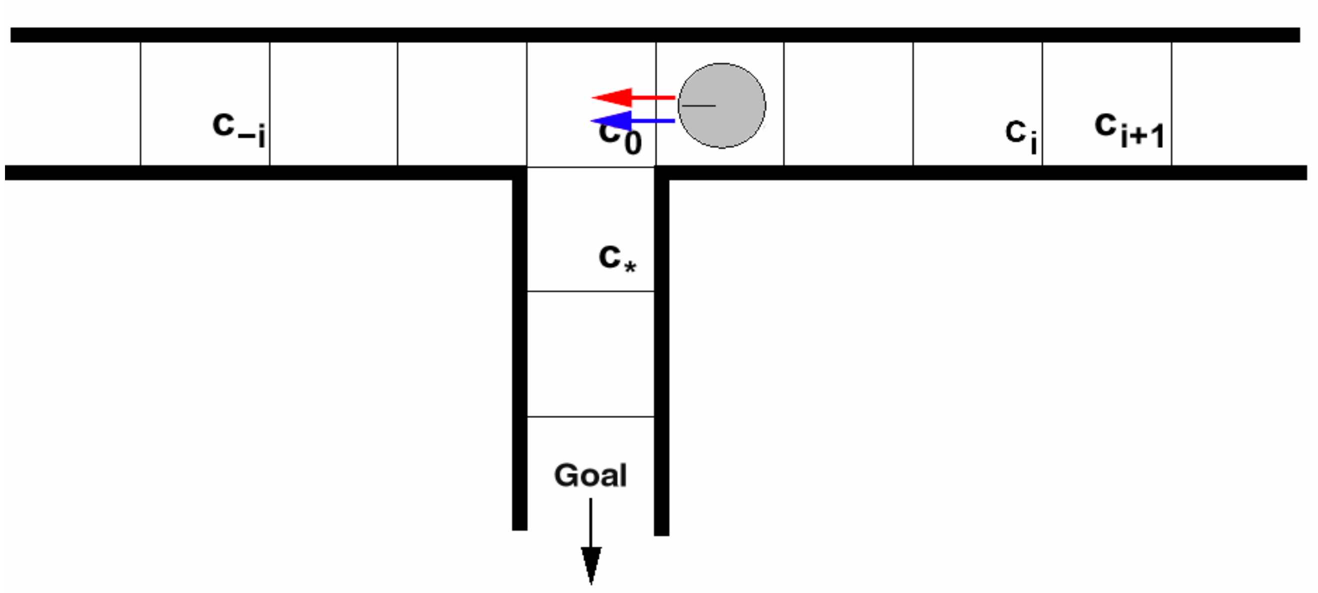
$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Problems of DWAs



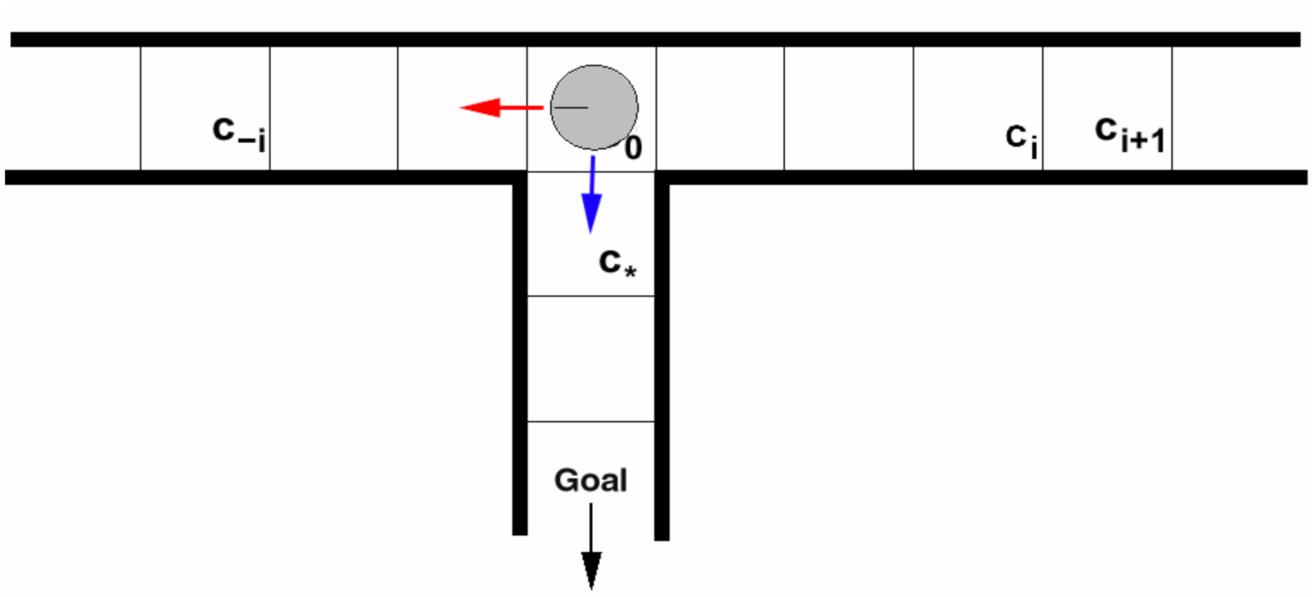
$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Problems of DWAs



$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

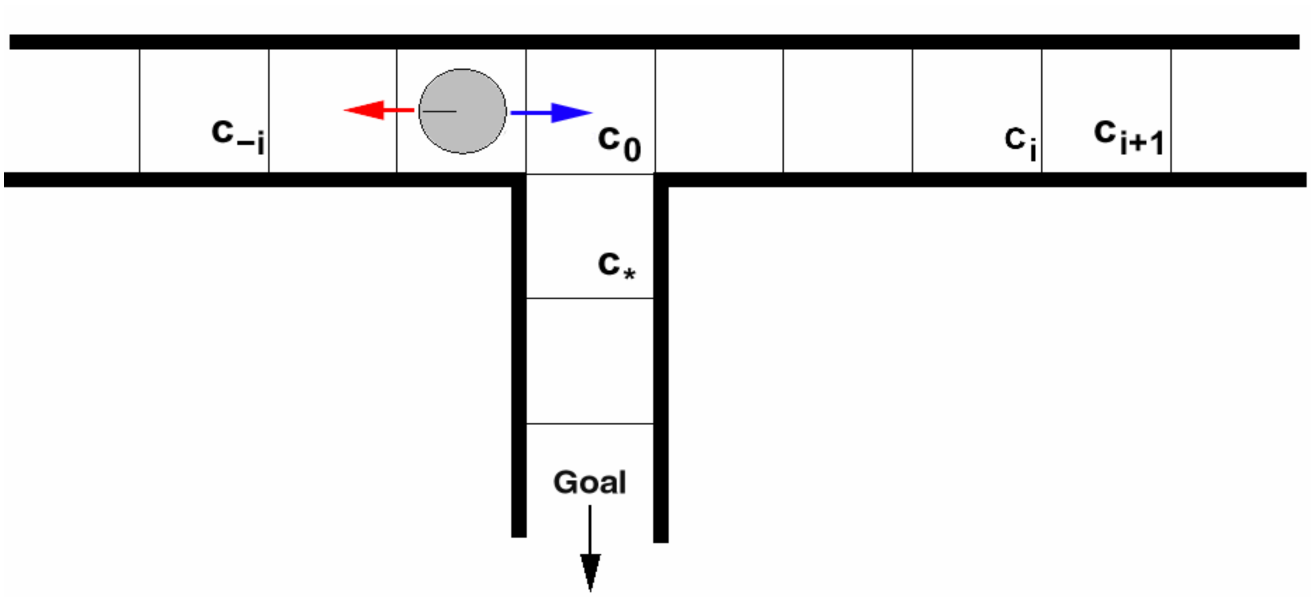
Problems of DWAs



$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

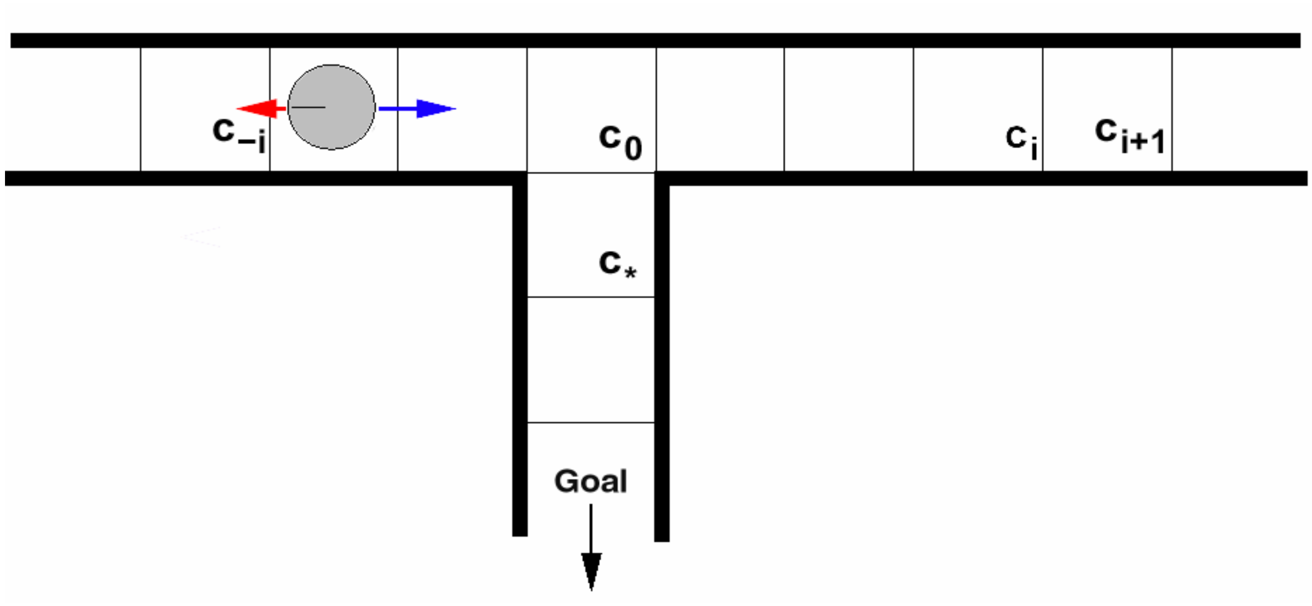
- The robot drives too fast at c_0 to enter corridor facing south.

Problems of DWAs



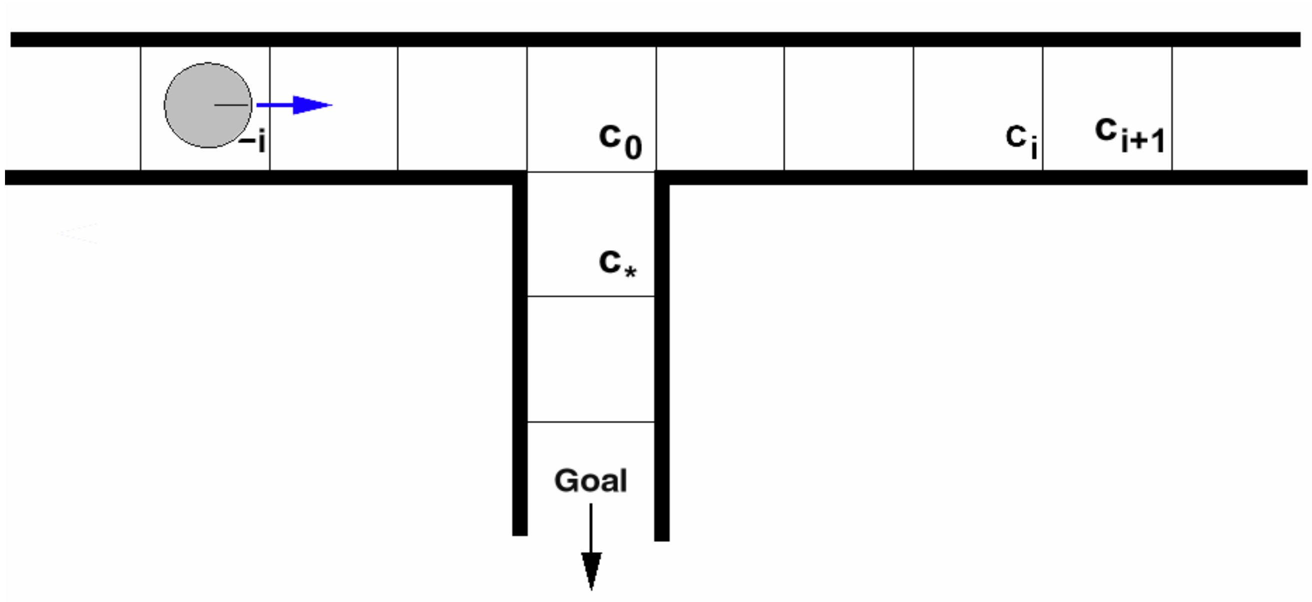
$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Problems of DWAs



$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Problems of DWAs



- Same situation as in the beginning.

  DWAs have problems to reach the goal

Problems of DWAs

- Typical problem in a real world situation:

- Robot does not slow down early enough to enter the doorway.

Outline

- Motion Planning Problem
- Potential Field Method
- Dynamic Window Approach
- A* Algorithm for Path Planning
- 5D Motion Planning
- Markov Decision Process

Dynamic Programing

- To solve a complex problem by breaking it down to simpler sub-problems

(1) Define sub-problems

(2) Find the recurrence that relates sub-problems

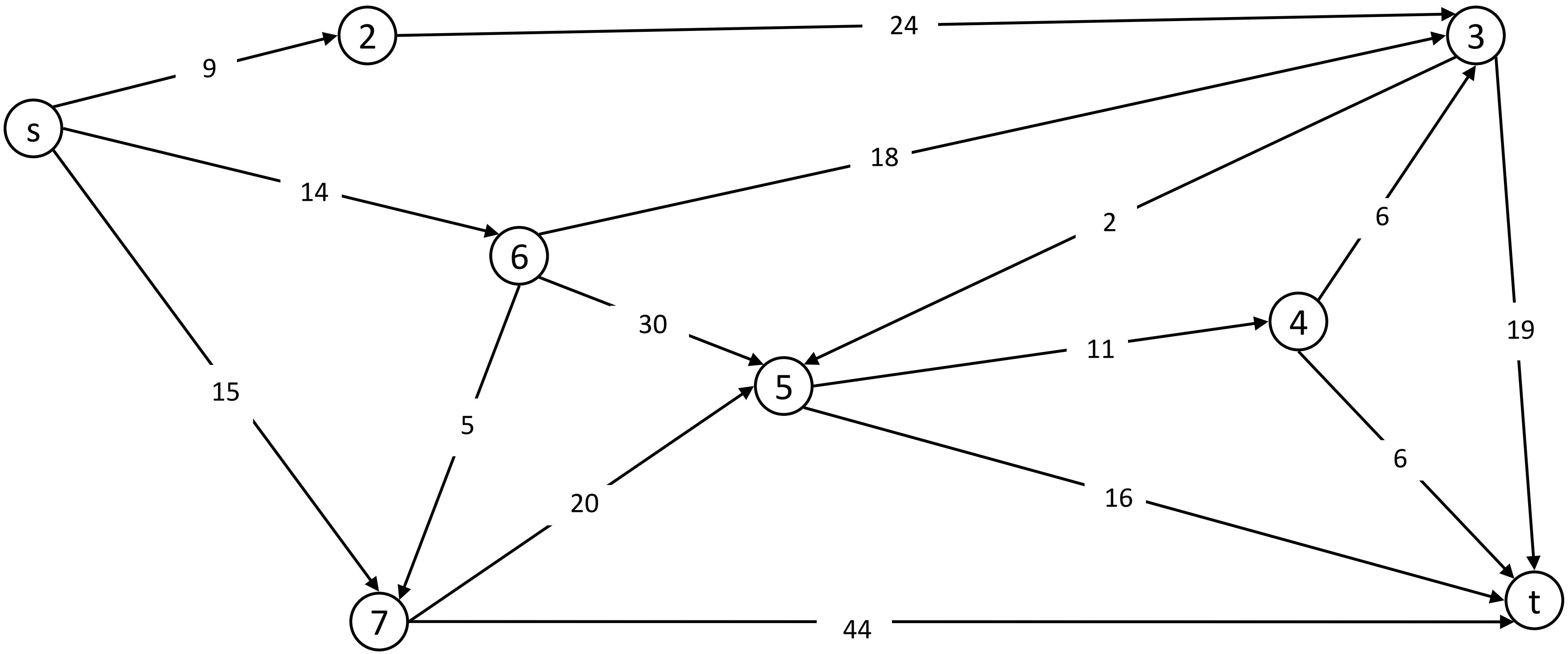
(3) Recognize and solve the base cases

(4) Perform iterations

(5) Trace back the solution

Dijkstra’s Shortest Path Algorithm

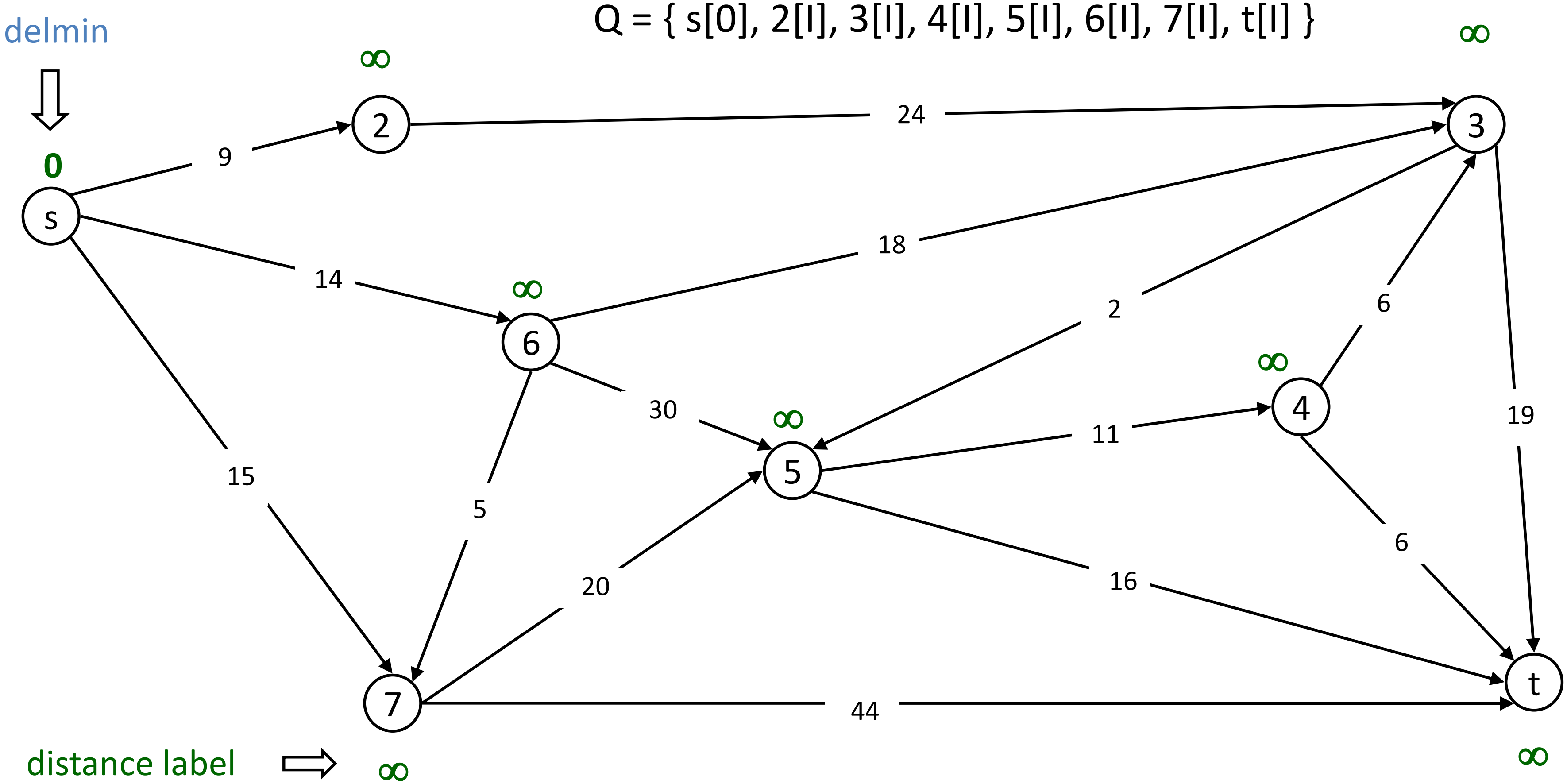
How to find the shortest path between ***s*** and ***t***?



Dijkstra’s Shortest Path Algorithm

S = { }

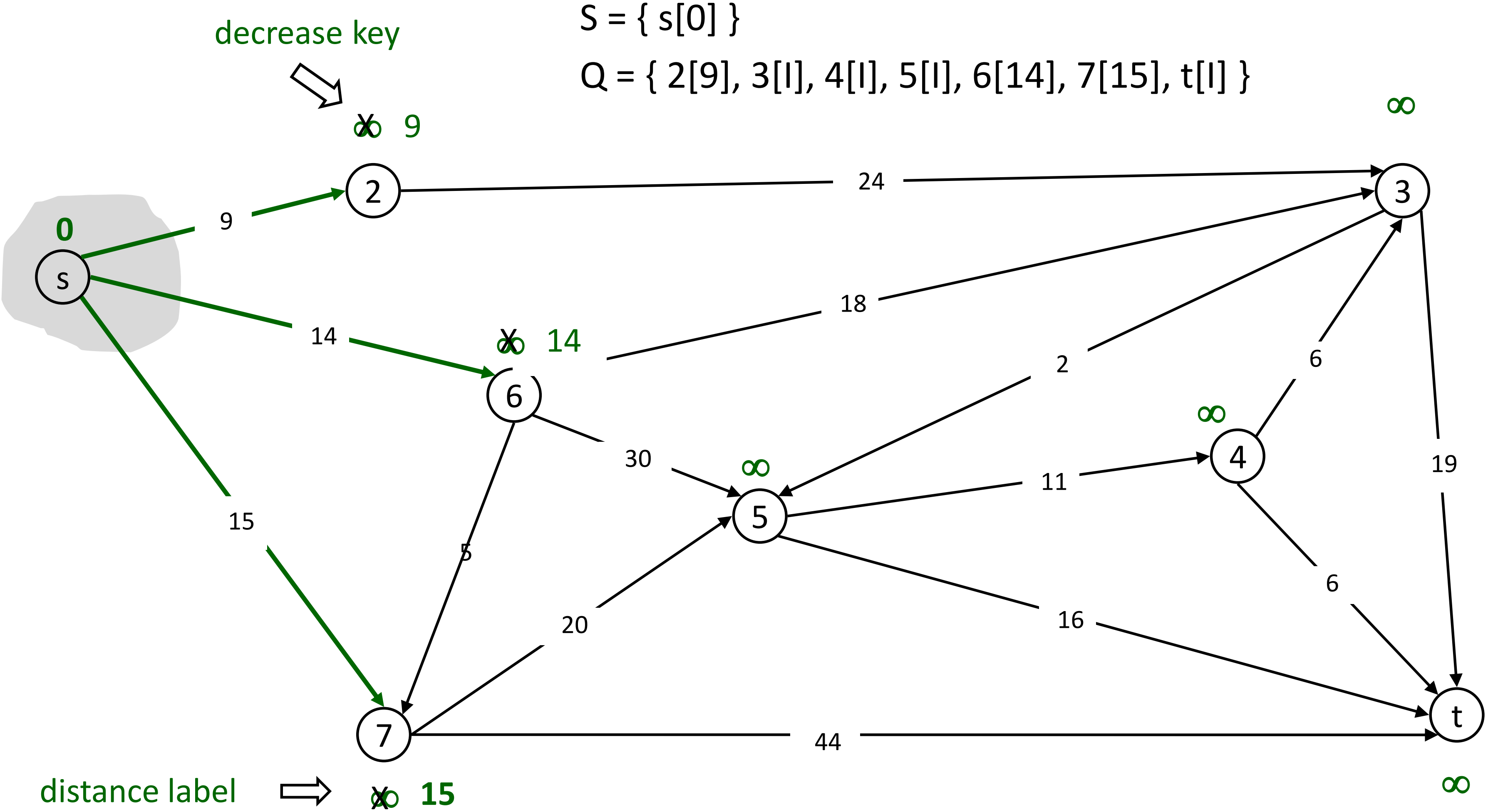
Q = { s[0], 2[I], 3[I], 4[I], 5[I], 6[I], 7[I], t[I] }



Dijkstra's Shortest Path Algorithm

S = { s[0] }

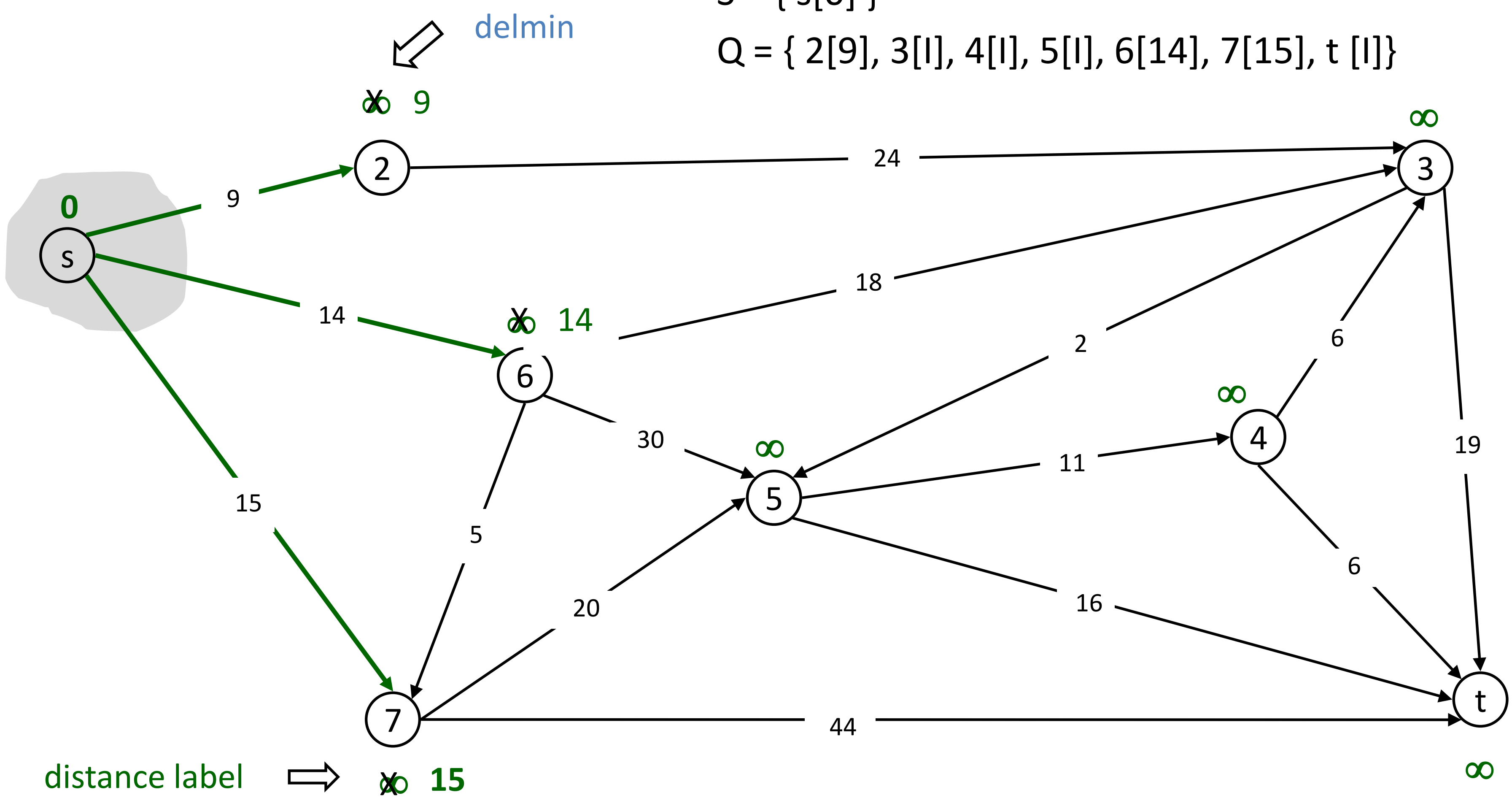
Q = { 2[9], 3[I], 4[I], 5[I], 6[14], 7[15], t[I] }



Dijkstra's Shortest Path Algorithm

S = { s[0] }

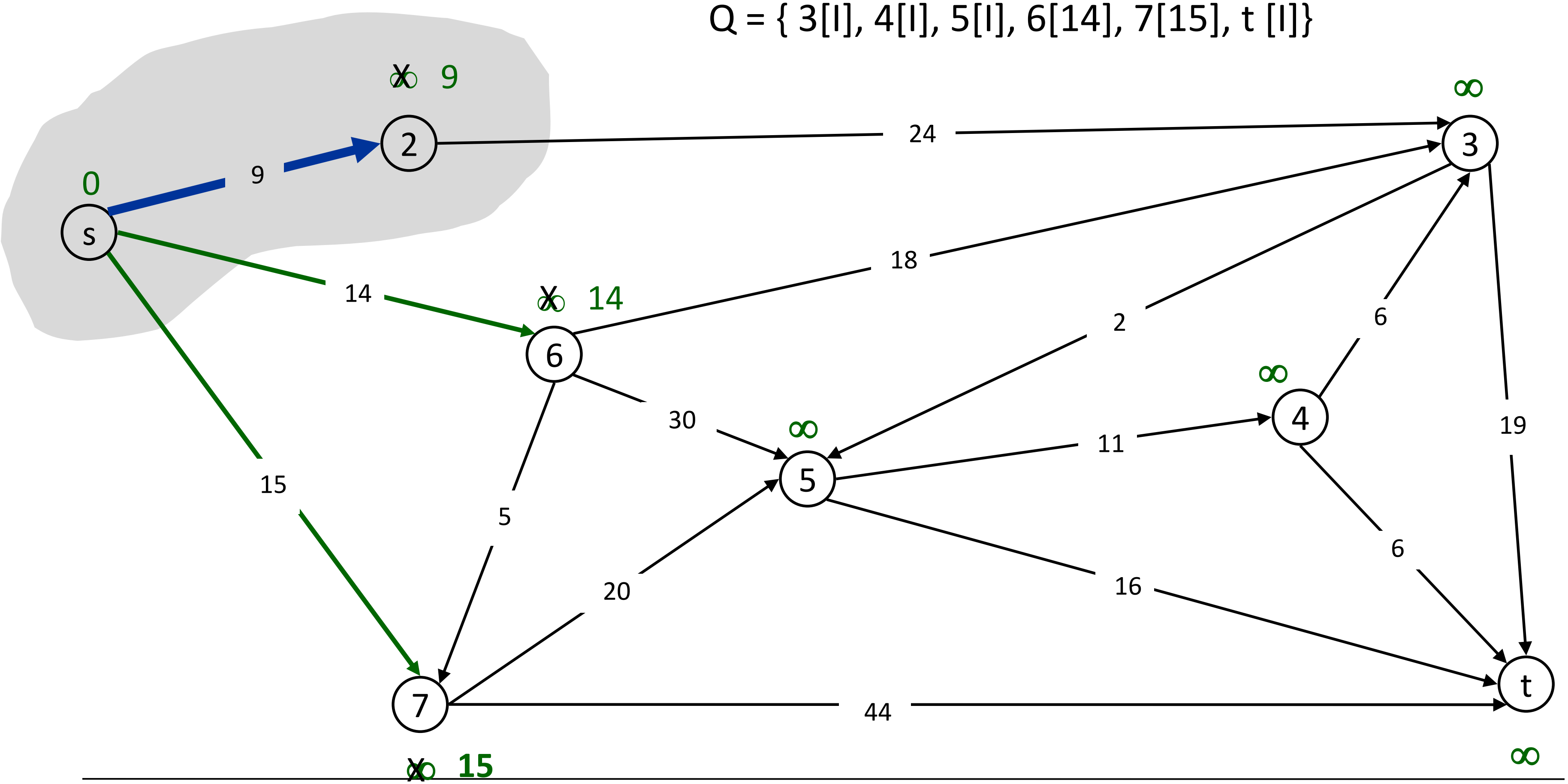
Q = { 2[9], 3[I], 4[I], 5[I], 6[14], 7[15], t [I]}



Dijkstra's Shortest Path Algorithm

S = { s[0], 2[9] }

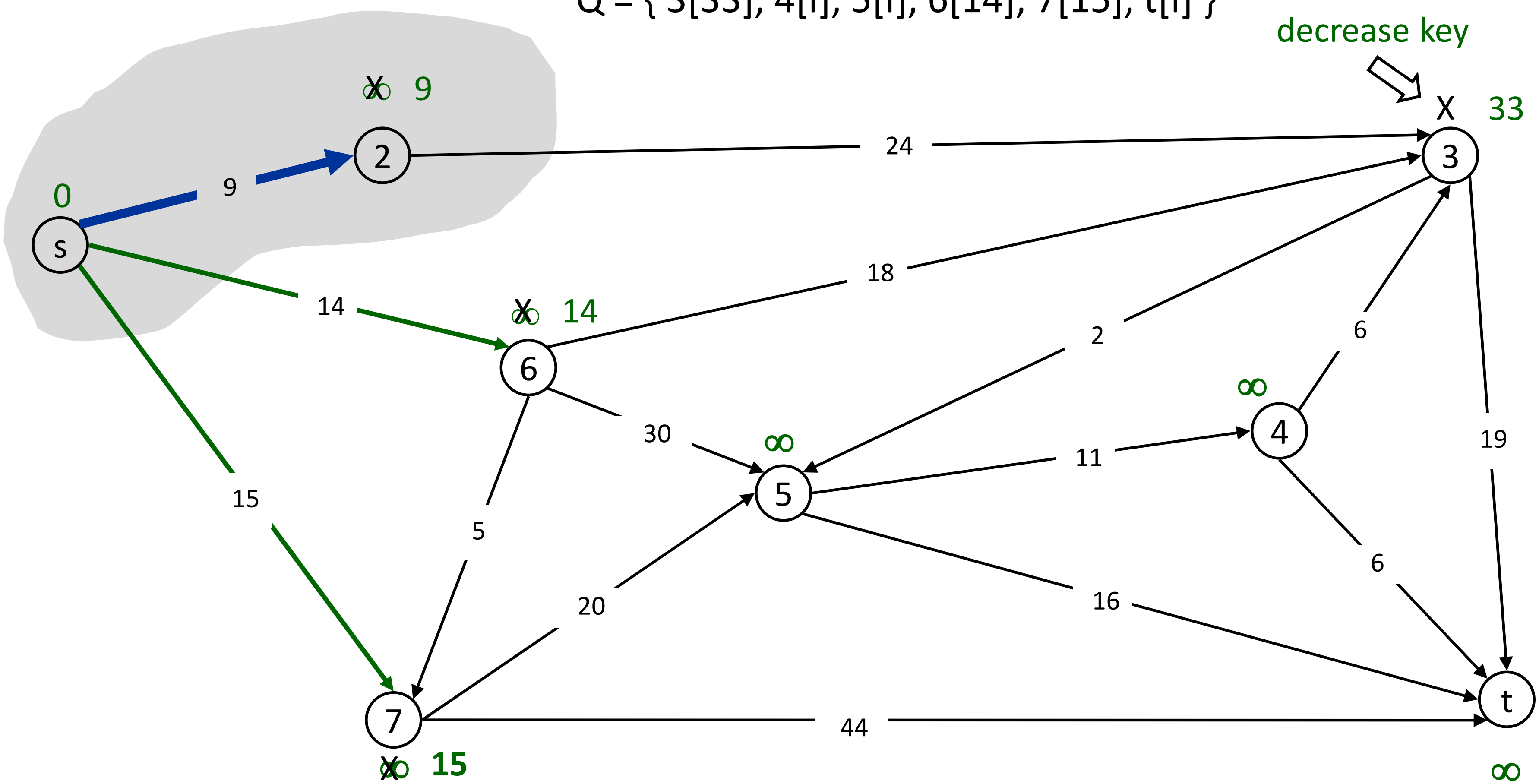
Q = { 3[I], 4[I], 5[I], 6[14], 7[15], t [I]}



Dijkstra's Shortest Path Algorithm

S = { s[0], 2[9] }

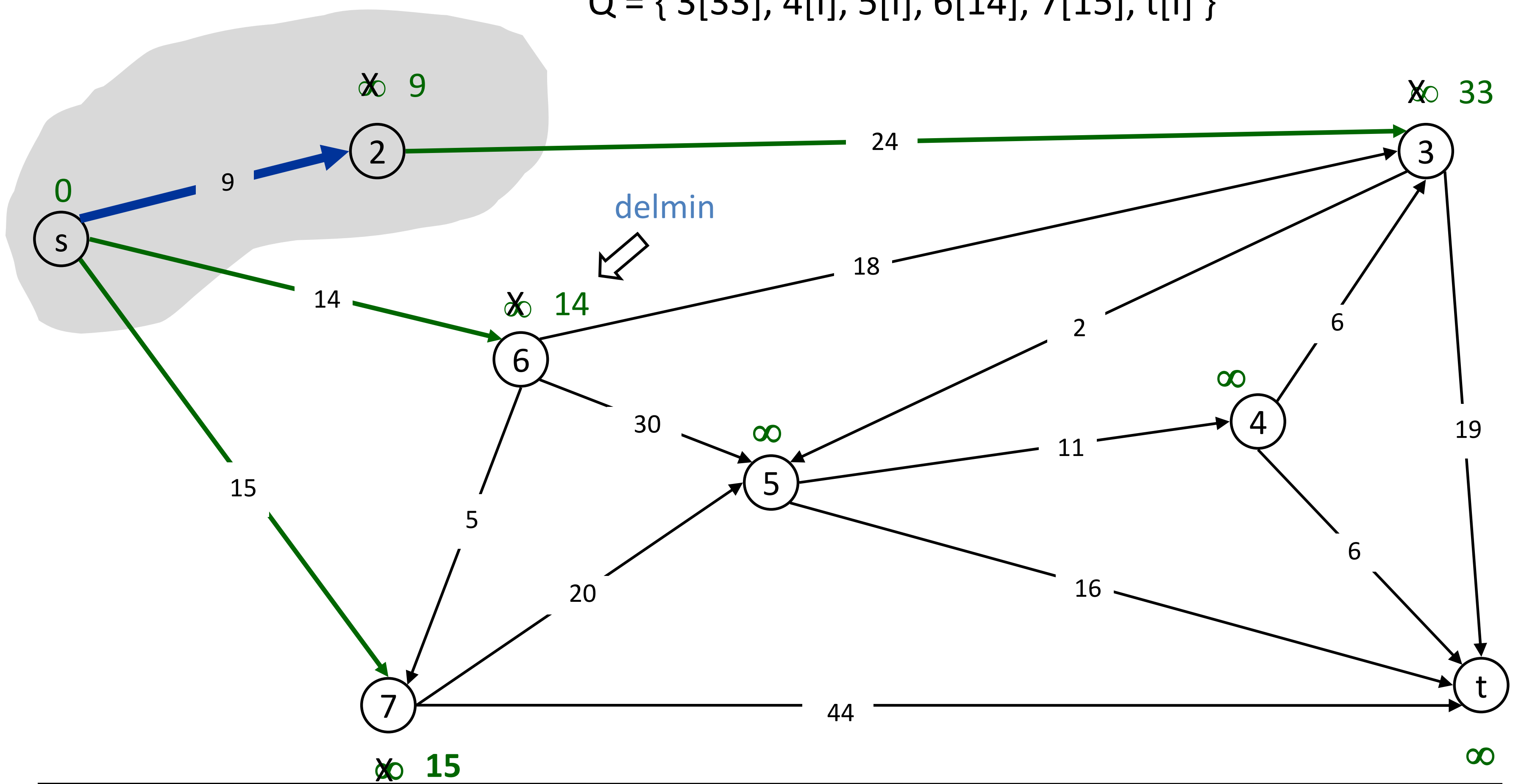
Q = { 3[33], 4[I], 5[I], 6[14], 7[15], t[I] }



Dijkstra's Shortest Path Algorithm

S = { s[0], 2[9] }

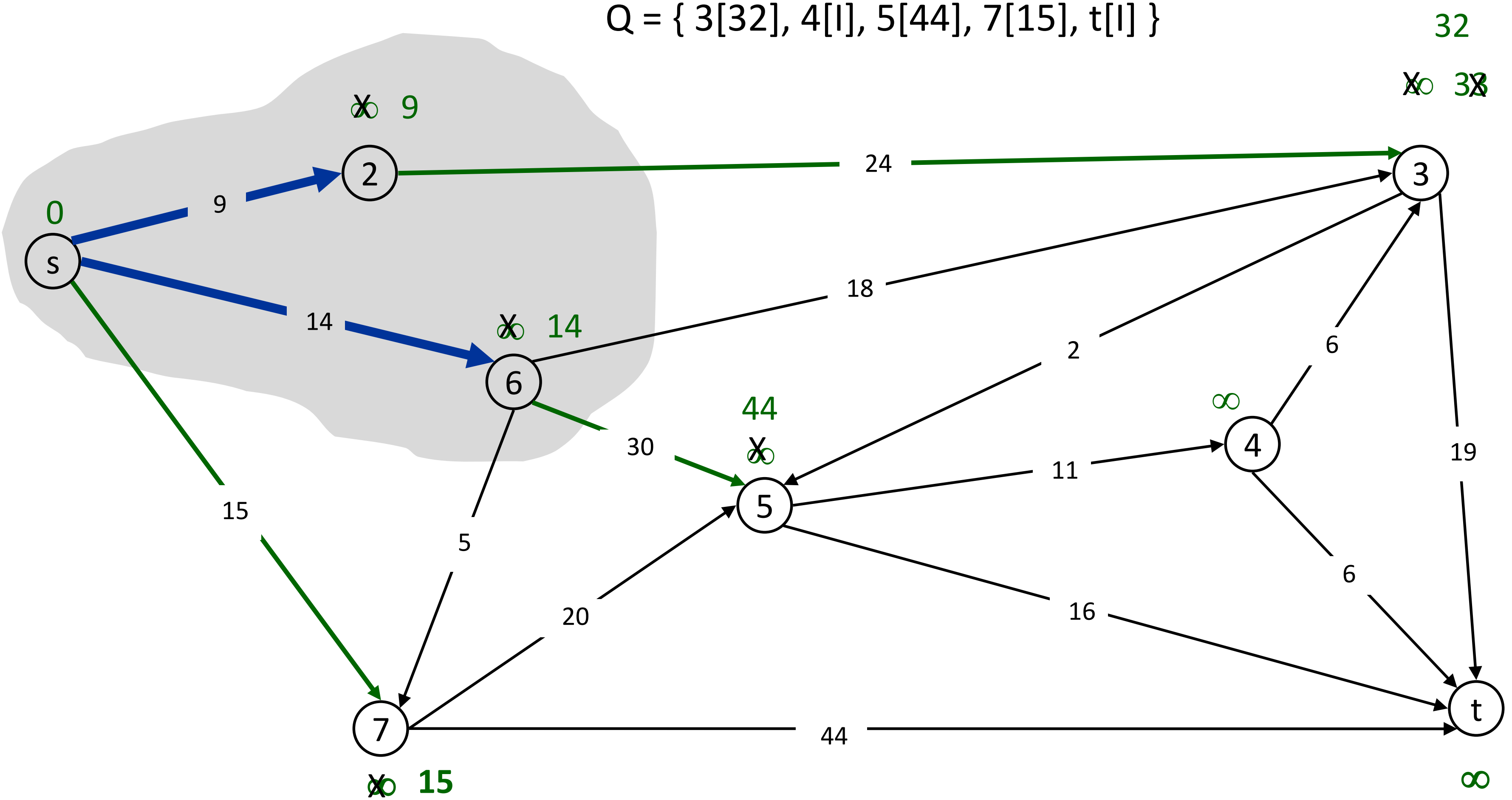
Q = { 3[33], 4[I], 5[I], 6[14], 7[15], t[I] }



Dijkstra's Shortest Path Algorithm

S = { s[0], 2[9], 6[14] }

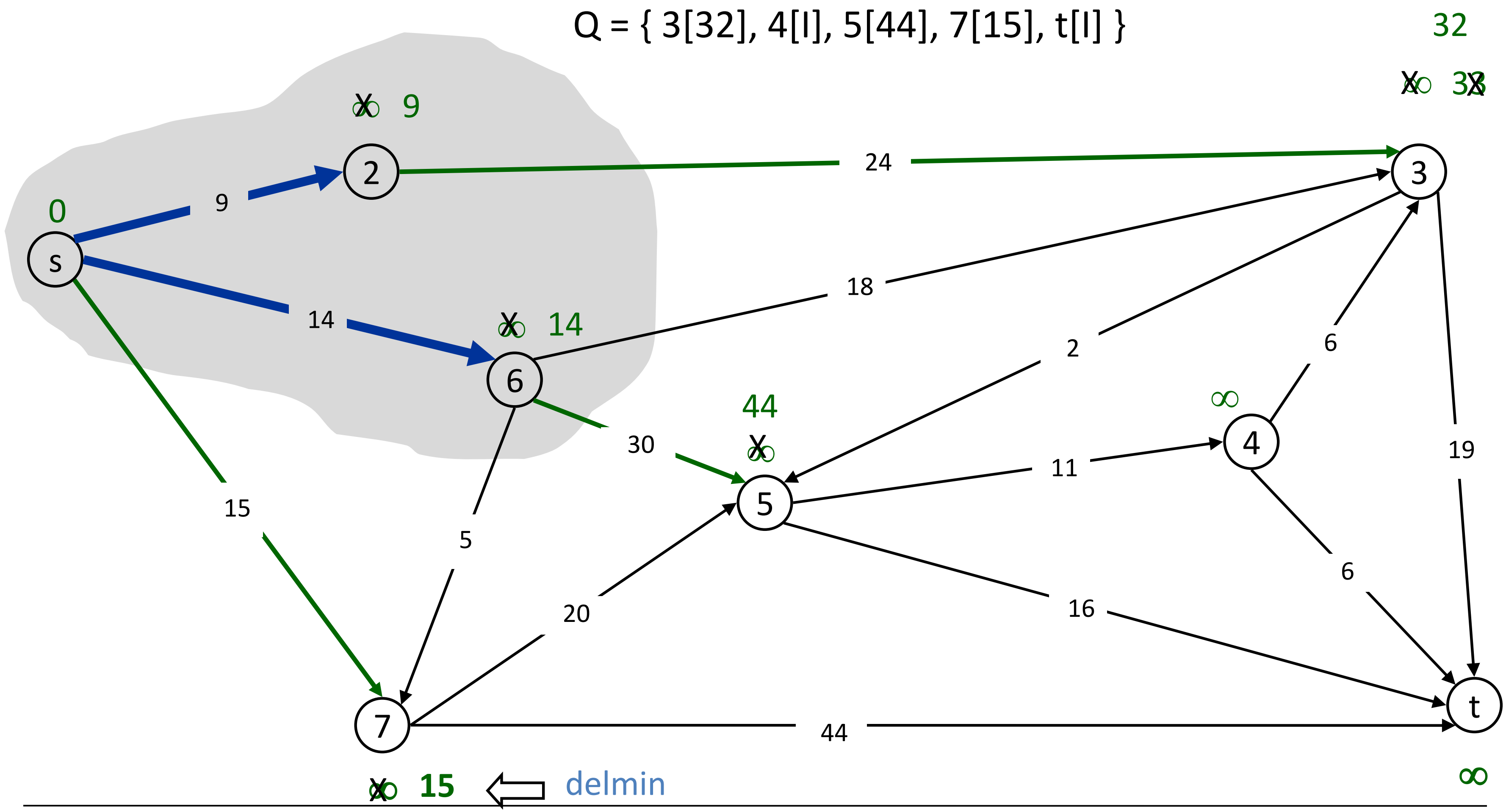
Q = { 3[32], 4[I], 5[44], 7[15], t[I] }



Dijkstra's Shortest Path Algorithm

S = { s[0], 2[9], 6[14] }

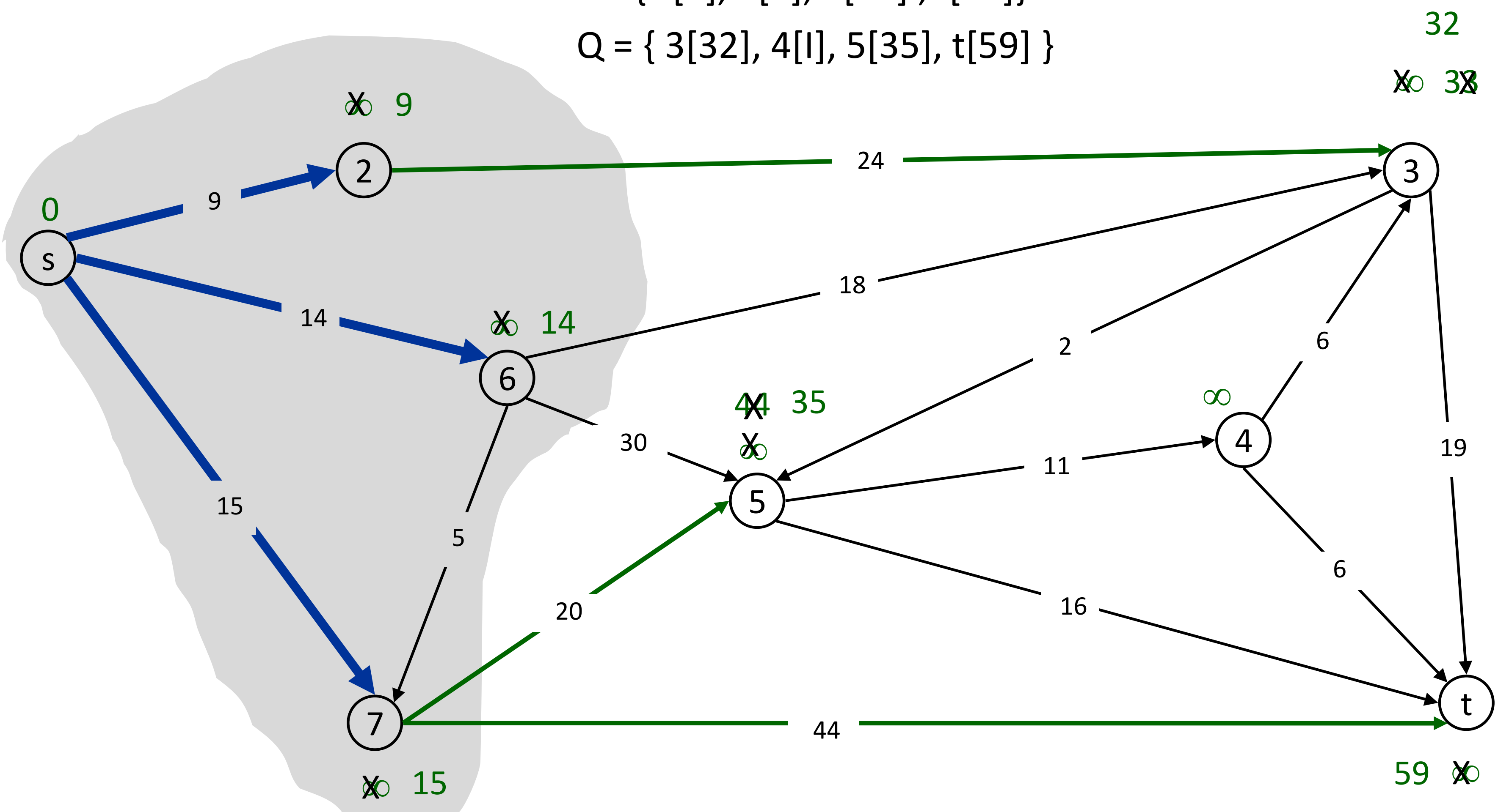
Q = { 3[32], 4[I], 5[44], 7[15], t[I] }



Dijkstra's Shortest Path Algorithm

S = { s[0], 2[9], 6[14] ,7[15]}

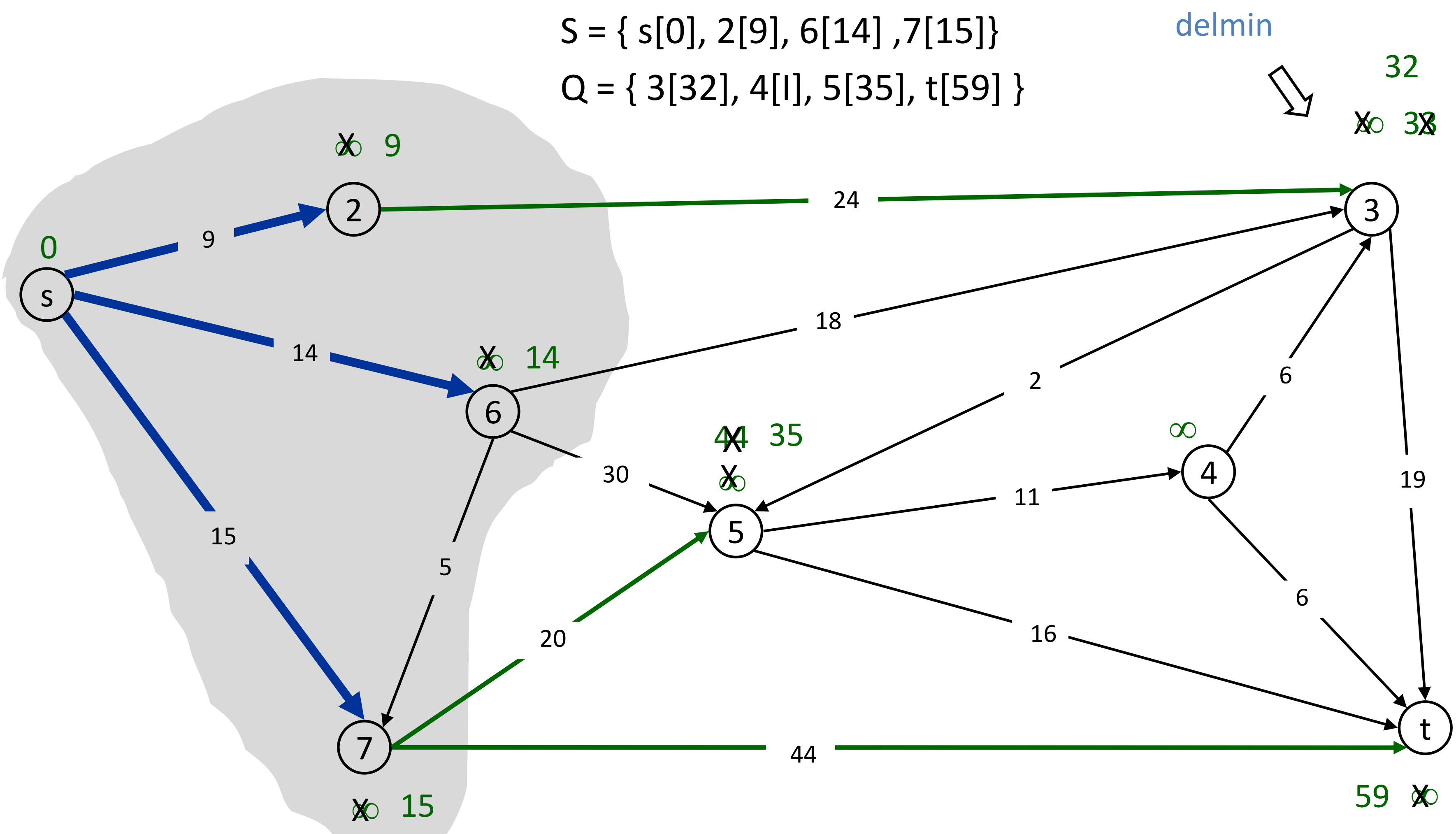
Q = { 3[32], 4[I], 5[35], t[59] }



Dijkstra's Shortest Path Algorithm

S = { s[0], 2[9], 6[14] ,7[15]}

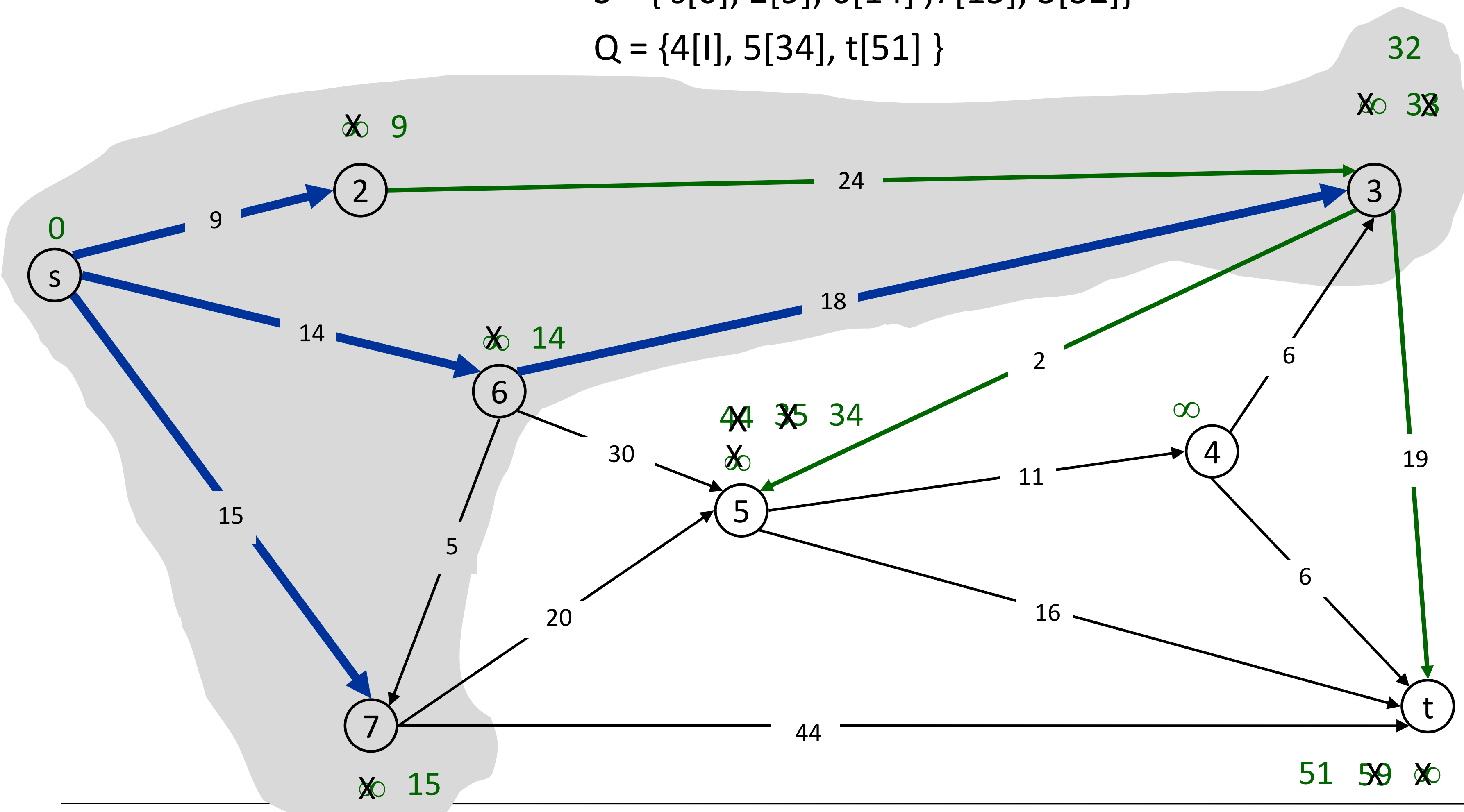
Q = { 3[32], 4[I], 5[35], t[59] }



Dijkstra's Shortest Path Algorithm

S = { s[0], 2[9], 6[14] ,7[15], 3[32]}

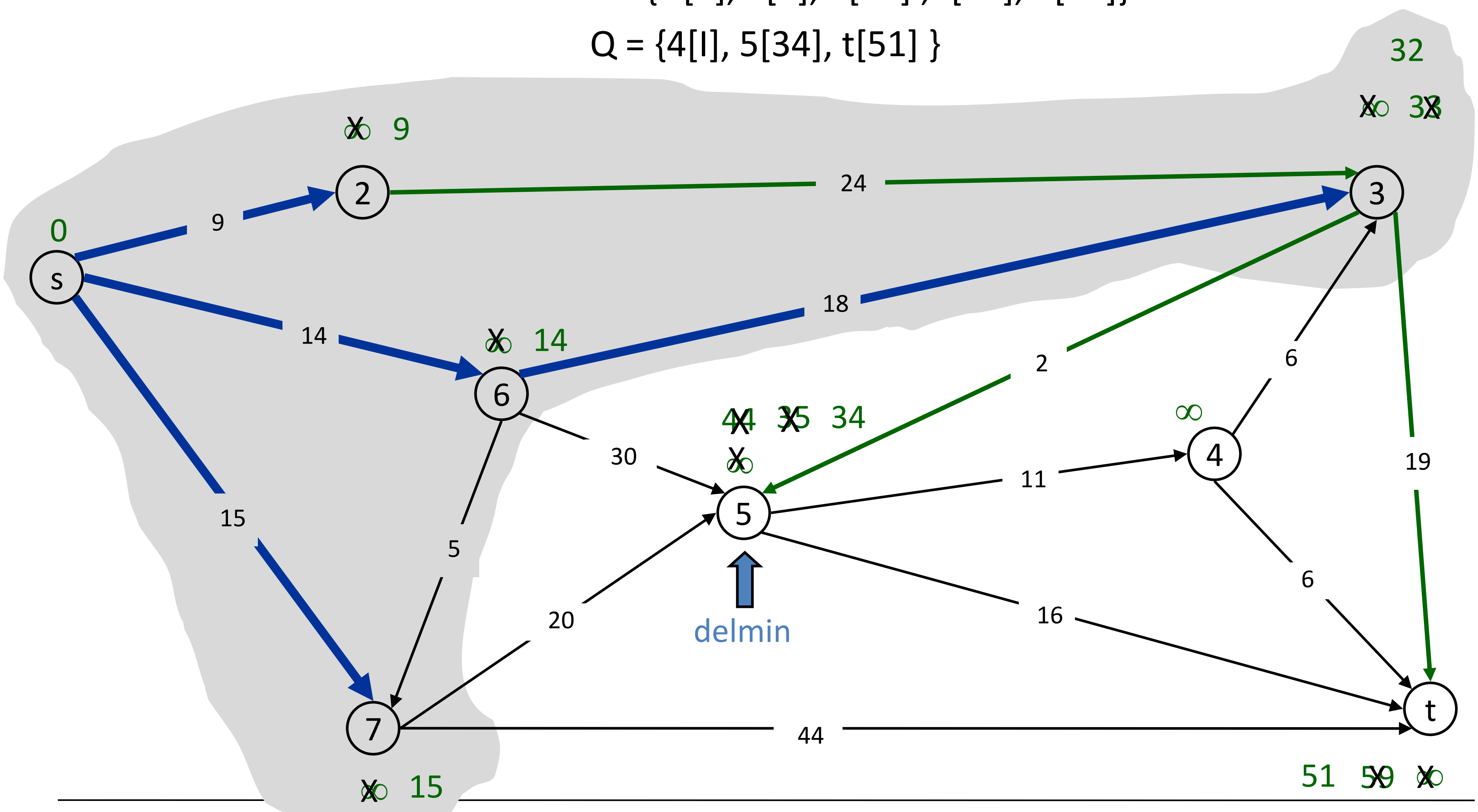
Q = {4[l], 5[34], t[51] }



Dijkstra's Shortest Path Algorithm

S = { s[0], 2[9], 6[14] ,7[15], 3[32]}

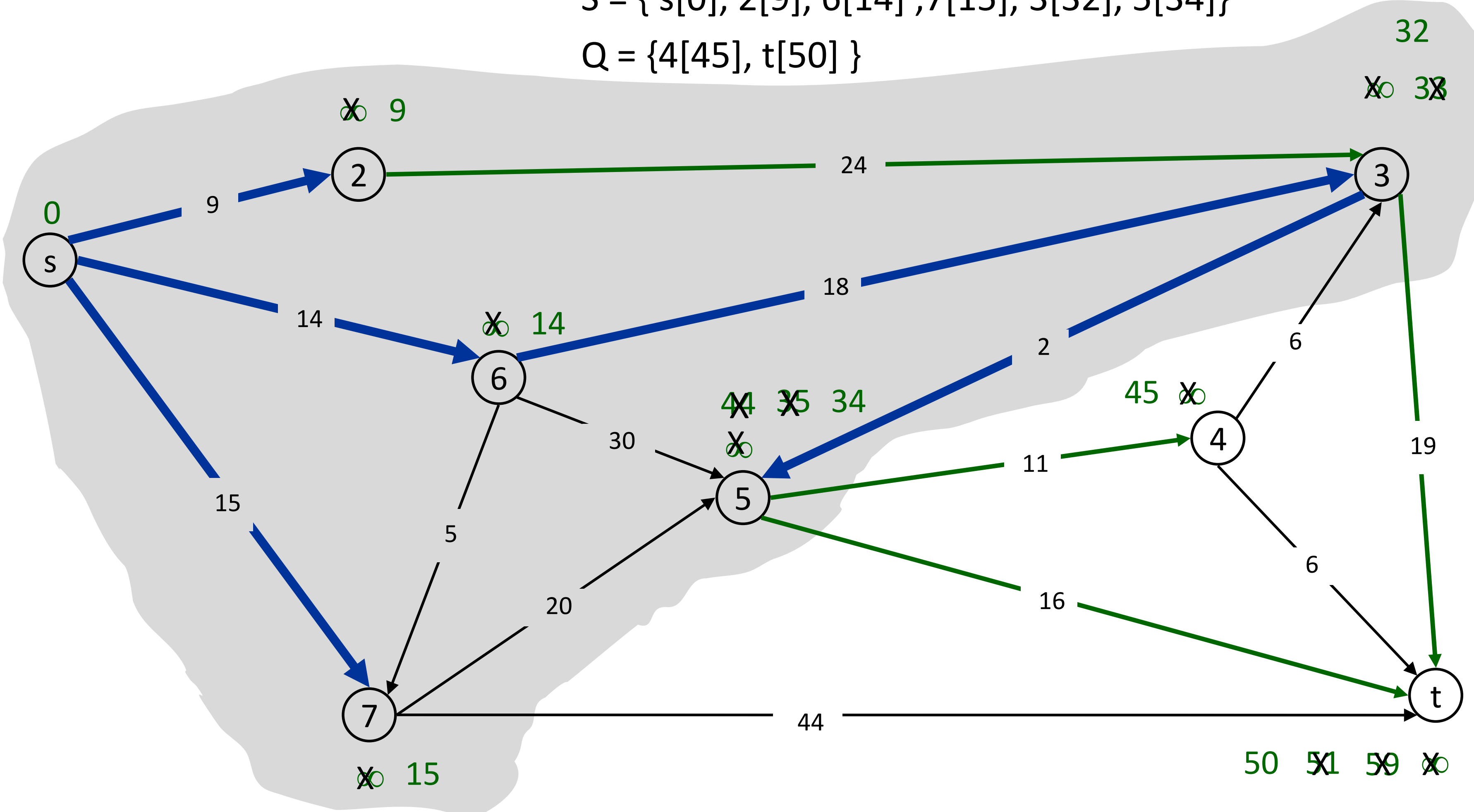
Q = {4[I], 5[34], t[51] }



Dijkstra's Shortest Path Algorithm

S = { s[0], 2[9], 6[14] ,7[15], 3[32], 5[34]}

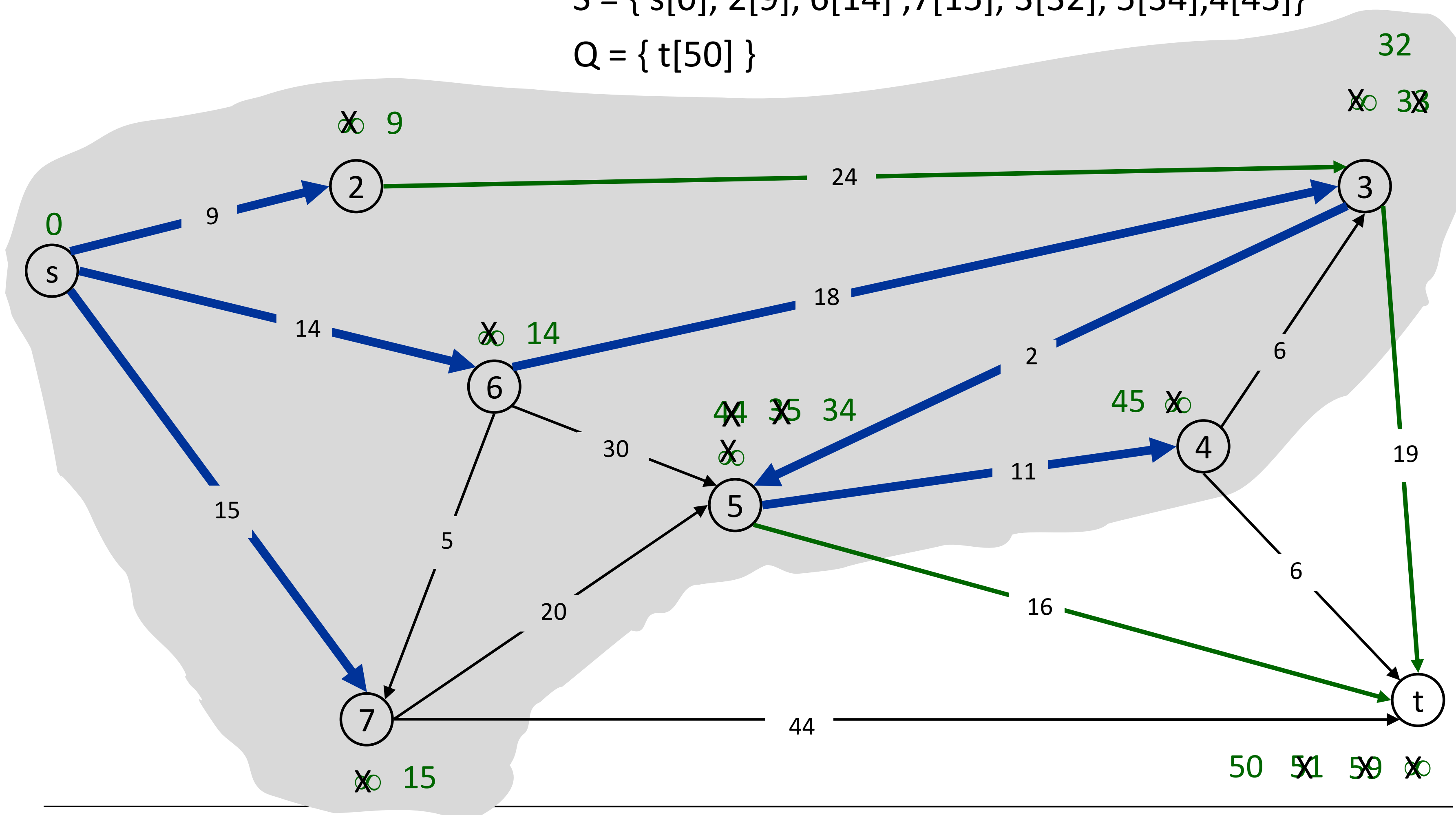
Q = {4[45], t[50] }



Dijkstra's Shortest Path Algorithm

S = { s[0], 2[9], 6[14] ,7[15], 3[32], 5[34],4[45]}

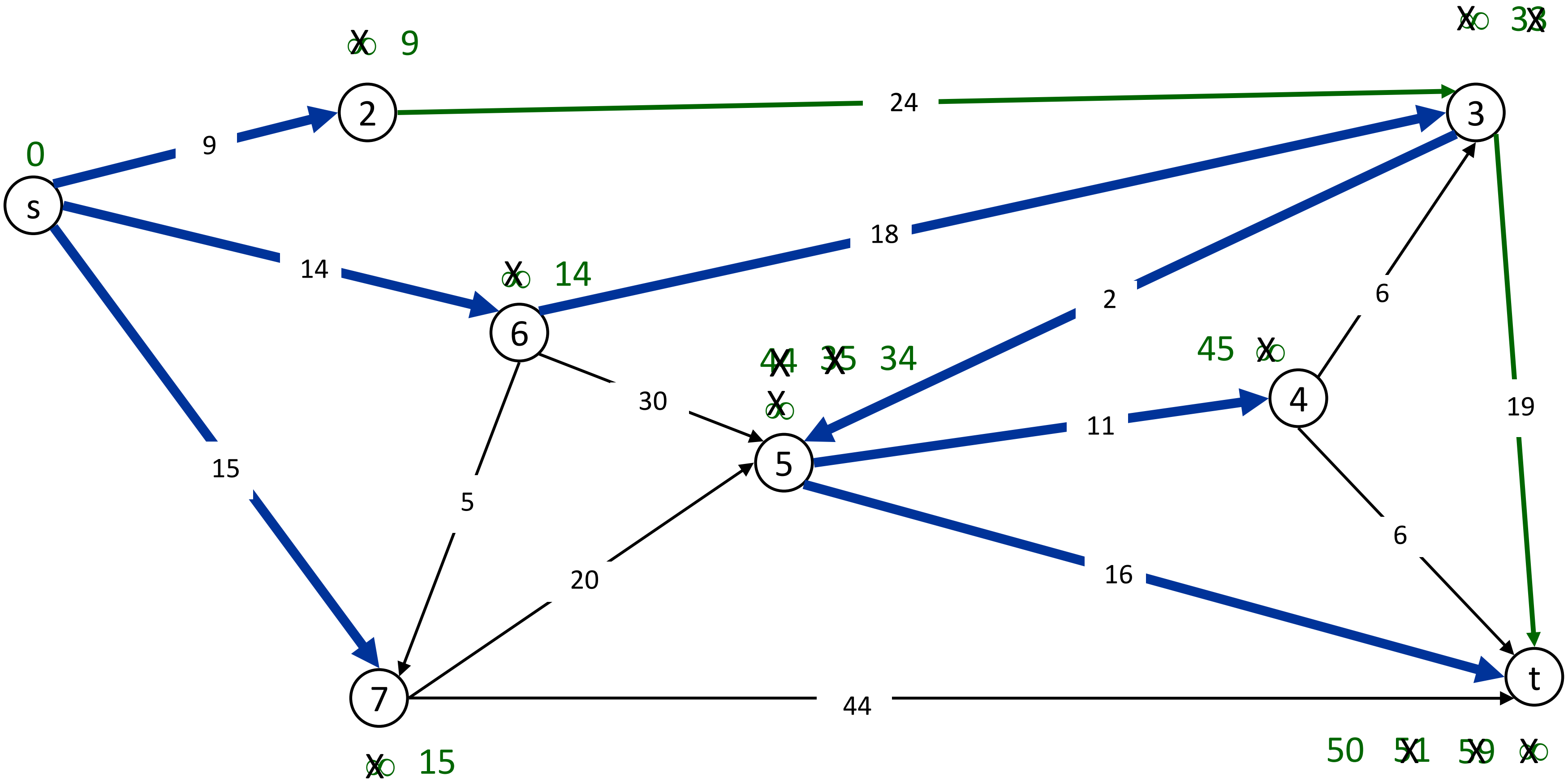
Q = { t[50] }



Dijkstra’s Shortest Path Algorithm

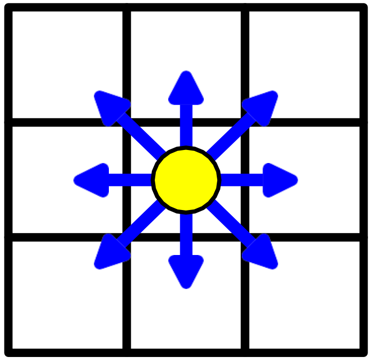
S = { s[0], 2[9], 6[14] ,7[15], 3[32], 5[34],4[45], t[50]}

Q = { }

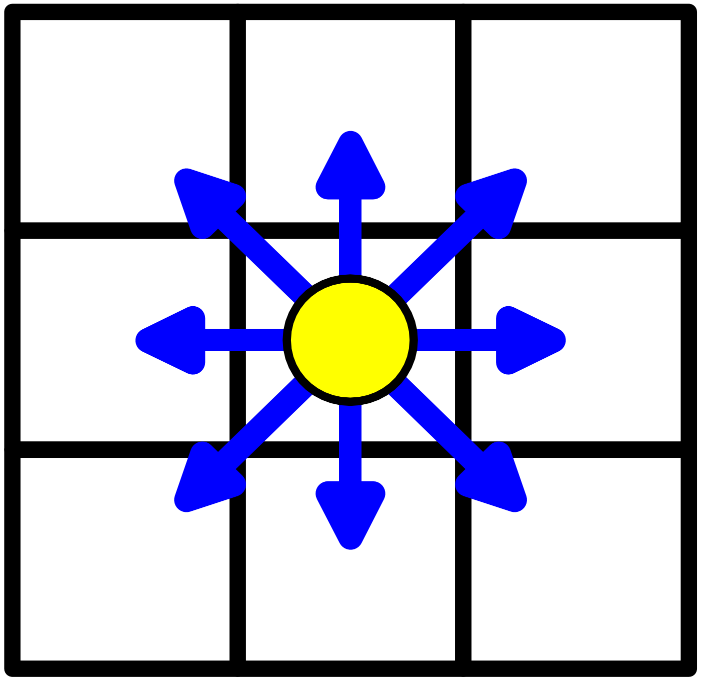


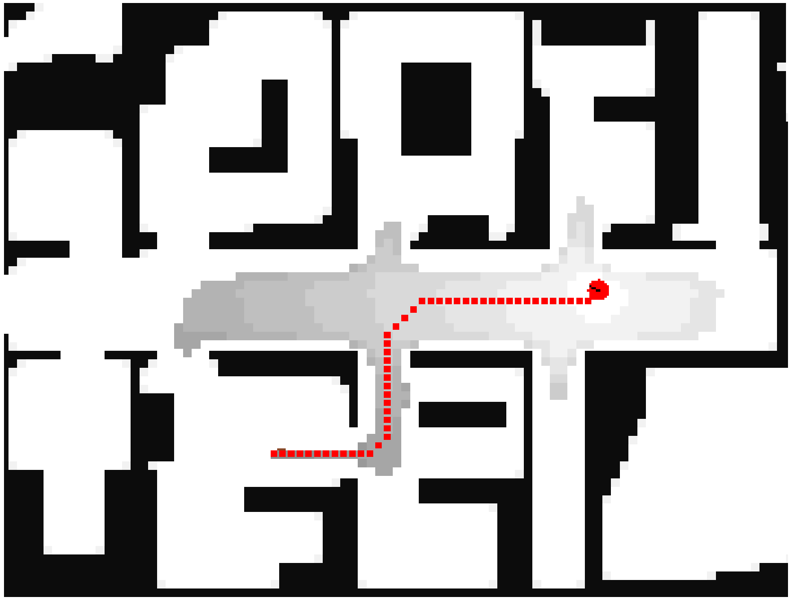
Path Planning with A*

- What about using A* to plan the path of a robot?
- Finds the shortest path
- Requires a graph structure
- Limited number of edges
- In robotics: planning on a 2d occupancy grid map



Path Planning in a Grid-World

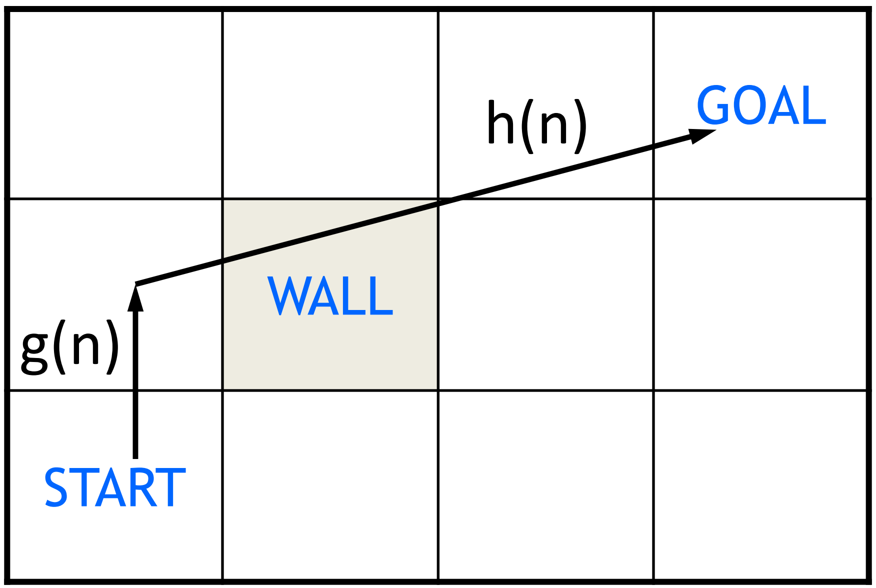




A*: Minimize the Estimated Path Costs

- $g(n)$ = actual cost from the initial state to n.
- $h(n)$ = estimated cost from n to the next goal.
- $f(n) = g(n) + h(n)$, the estimated cost of the cheapest solution through n.
- Let h*(n) be the actual cost of the optimal path from n to the next goal.
- h is admissible if the following holds for all n :
  - $h(n) \leq h^*(n)$
- We require that for A*, h is admissible (the straight-line distance is admissible in the Euclidean Space).

Path Cost



g(n) = 1
h(n) = sqrt(1^2+3^2)

g(n) is estimated
for admissible positions
h(n) is estimated without
considering obstacles

Dijkstra’s Algorithm: find a path that minimizes g(n)
A*: find a path that minimizes g(n) + h(n)

Deterministic Value Iteration

- To compute the shortest path from every state to one goal state (h(n)), use (deterministic) value iteration.
- Very similar to Dijkstra's Algorithm (g(n)).
- Such a cost distribution is the optimal heuristic for A* (h(n)+g(n)).

Typical Assumptions in A*

- A robot is assumed to be localized.
- Often a robot has to compute a path based on an occupancy grid.
- Often the correct motion commands are executed (but no perfect map).

Is this always true?

Problems

- What if the robot is slightly delocalized?
- Moving on the shortest path guides often the robot on a trajectory close to obstacles.
- Trajectory aligned to the grid structure.

Convolution of the Grid Map

- Convolution blurs the map.
- Obstacles are assumed to be bigger than in reality.
- Perform an A* search in such a convolved map.
- The robot increases distance to obstacles and moves on a short path!

Convolution

- Consider an occupancy map, then the convolution is defined as:

$$P(occ_{x_i,y}) = \frac{1}{4} \cdot P(occ_{x_{i-1},y}) + \frac{1}{2} \cdot P(occ_{x_i,y}) + \frac{1}{4} \cdot P(occ_{x_{i+1},y})$$

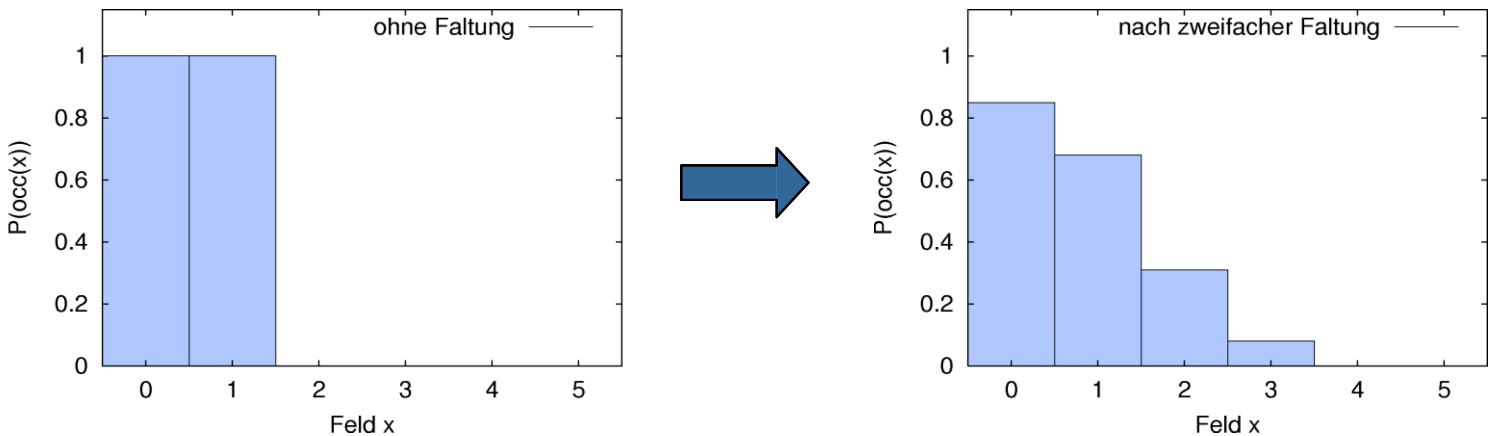
$$P(occ_{x_0,y}) = \frac{2}{3} \cdot P(occ_{x_0,y}) + \frac{1}{3} \cdot P(occ_{x_1,y})$$

$$P(occ_{x_{n-1},y}) = \frac{1}{3} \cdot P(occ_{x_{n-2},y}) + \frac{2}{3} \cdot P(occ_{x_{n-1},y})$$

- This is done for each row and each column of the map.
- "Gaussian blur"

Map Convolution

- 1-d environment, cells $c_0, \ldots, c_5$



- Cells before and after 2 convolution runs.

A* in Convolved Maps

- The costs are a product of path length and occupancy probability of the cells.
- Cells with higher probability (e.g., caused by convolution) are avoided by the robot.
- Thus, it keeps distance to obstacles.
- This technique is fast and quite reliable.

Outline

- Motion Planning Problem
- Potential Field Method
- Dynamic Window Approach
- A* Algorithm for Path Planning
- 5D Motion Planning
- Markov Decision Process

5D Planning

- Plans in the full <x,y,θ,v,ω>-configuration space using A*.

  considers the robot's kinematic constraints.

- Generates a sequence of steering commands to reach the goal location.

- Maximizes trade-off between driving time and distance to obstacles.

The Search Space (I)

- What is a state in this space?
  $<x,y,\theta,v,\omega>$ = current position and
  speed of the robot

- How does a state transition look like?
  $<x_1,y_1,\theta_1,v_1,\omega_1> \longrightarrow <x_2,y_2,\theta_2,v_2,\omega_2>$
  with motion command (v_2,ω_2) and
  $|v_1-v_2| < a_v$, $|\omega_1-\omega_2| < a_\omega$.

  <u>Robot Pose: a result of the motion equations</u>

The Search Space (II)

Idea: search in the discretized $<x,y,\theta,v,\omega>$-space.

Problem: the search space is too huge to be explored within the time constraints (.25 secs for online control).

Solution: restrict the full search space.

Main Steps of the Algorithm

1. Update (static) grid map based on sensory input.

2. Use A^* to find a trajectory in the $<x,y>$-space using the updated grid map.

3. Determine a restricted 5d-configuration space based on step 2.

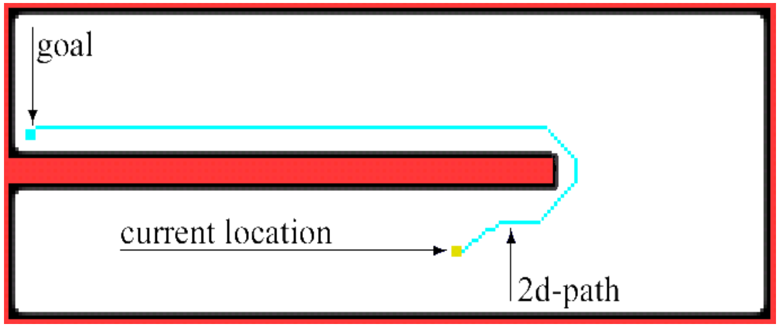
4. Find a trajectory by planning in the restricted $<x,y,\theta,v,\omega>$-space.

Outline

- The environment is represented as a 2d-occupency grid map.
- Convolution of the map increases security distance.
- Detected obstacles are added.
- Cells discovered free are cleared.

Find a Path in 2D-Space

- Use A* to search for the optimal path in the 2d-grid map.
- Use heuristic based on a deterministic value iteration within the static map.



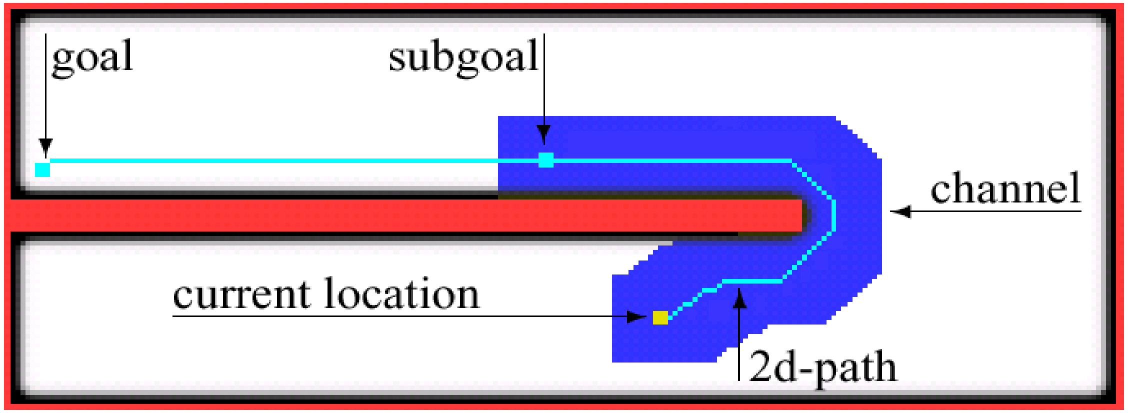
Restricting the Search Space

Assumption: the projection of the 5d-path onto the <x,y>-space lies close to the optimal 2d-path.

Therefore: construct a restricted search space (channel) based on the 2d-path.

Space Restriction

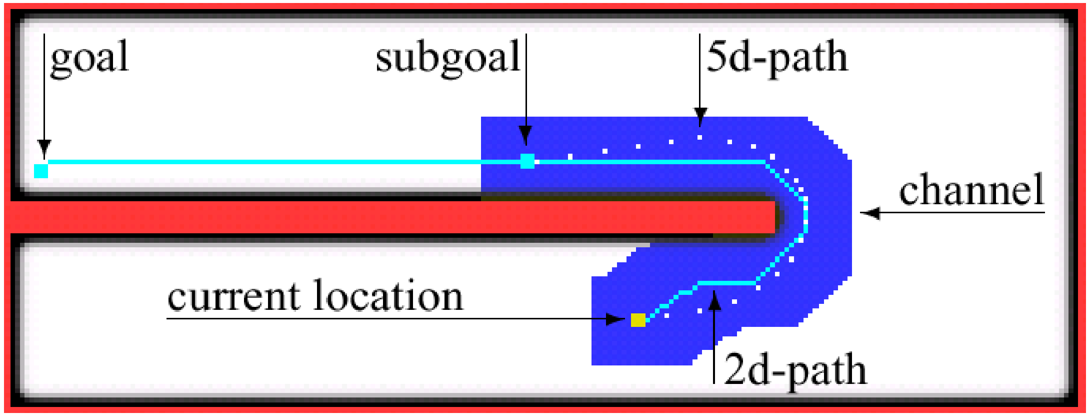
- Resulting search space =
  <x, y, θ, v, ω> with (x,y) ϵ channel.
- Choose a sub-goal lying on the 2d-path within the channel.

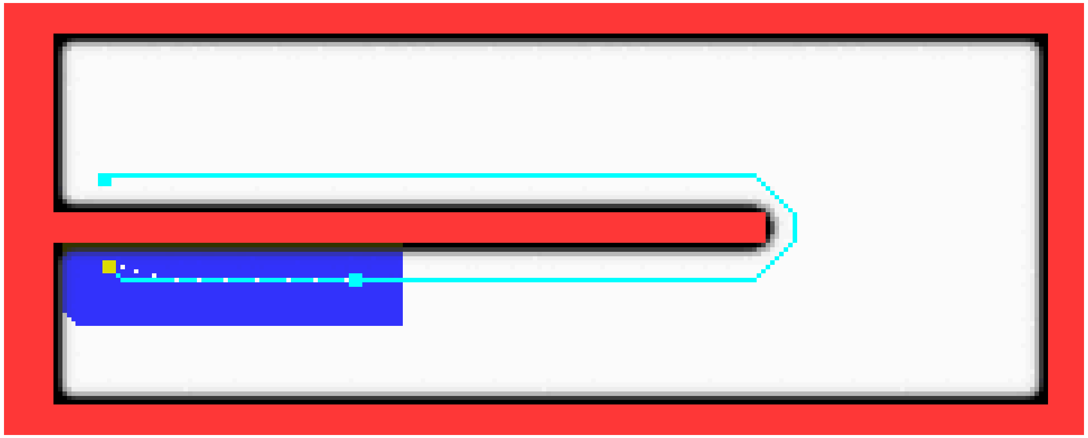


Find a Path in the 5d-Space

- Use A^* in the restricted 5d-space to find a sequence of steering commands to reach the sub-goal.
- To estimate cell costs: perform a deterministic 2d-value iteration within the channel.

Results





Timeouts

- Steering a robot online requires to set a new steering command every .25 secs.

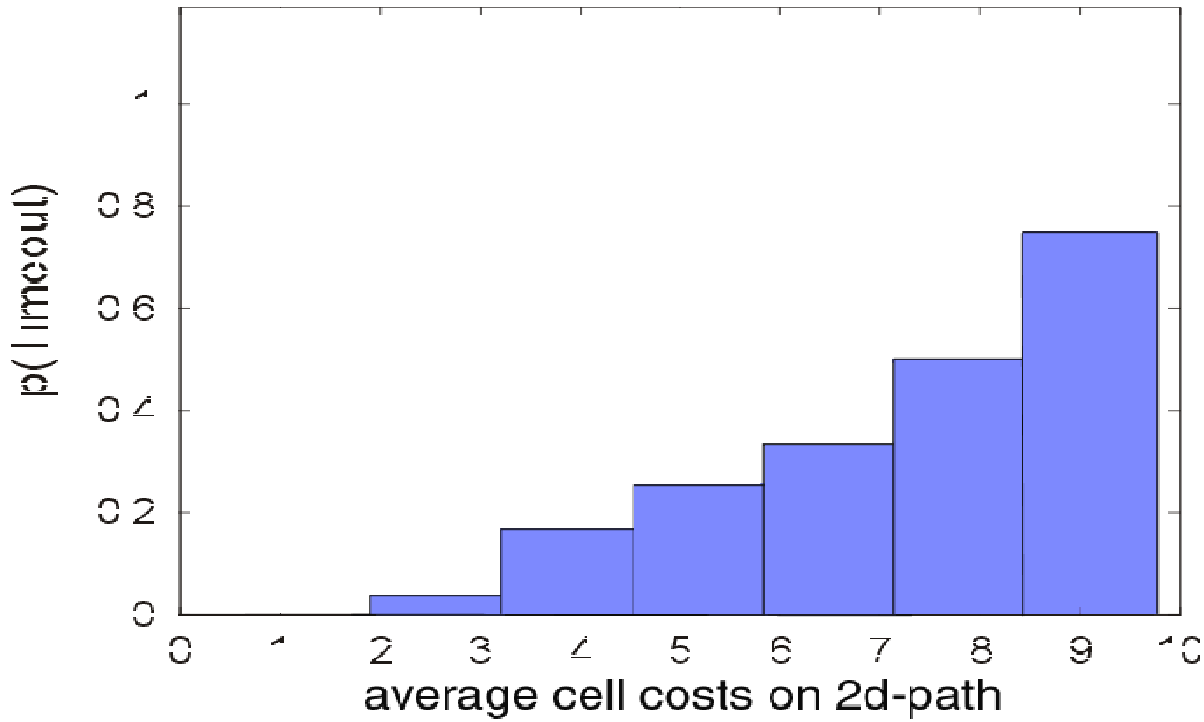
- Abort search after .25 secs.

How to find an admissible steering command?

Alternative Steering Command

- Previous trajectory still admissible?
- If not, drive on the 2d-path or use DWA to find new command.

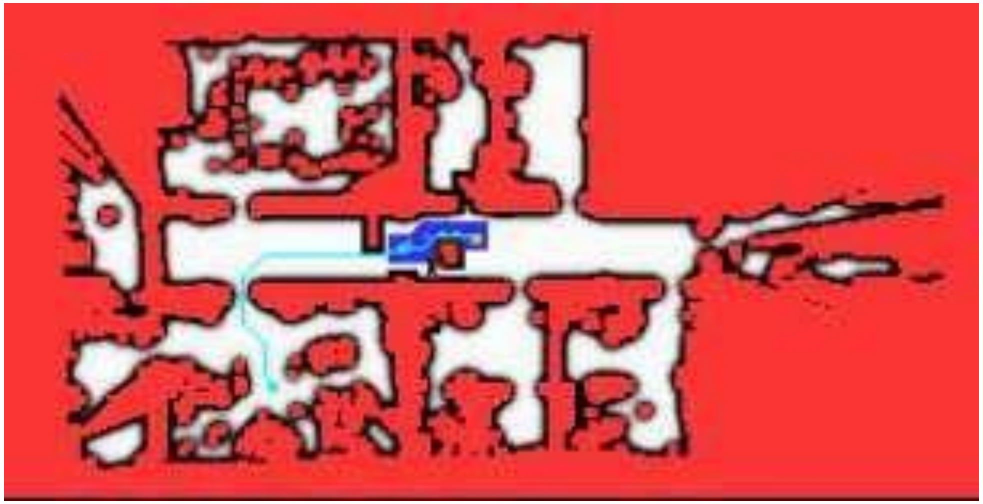
Timeout Avoidance



Reduce the size of the channel the 2d-path that has high cost.

Indoor Planning

Robot Albert



Planning state

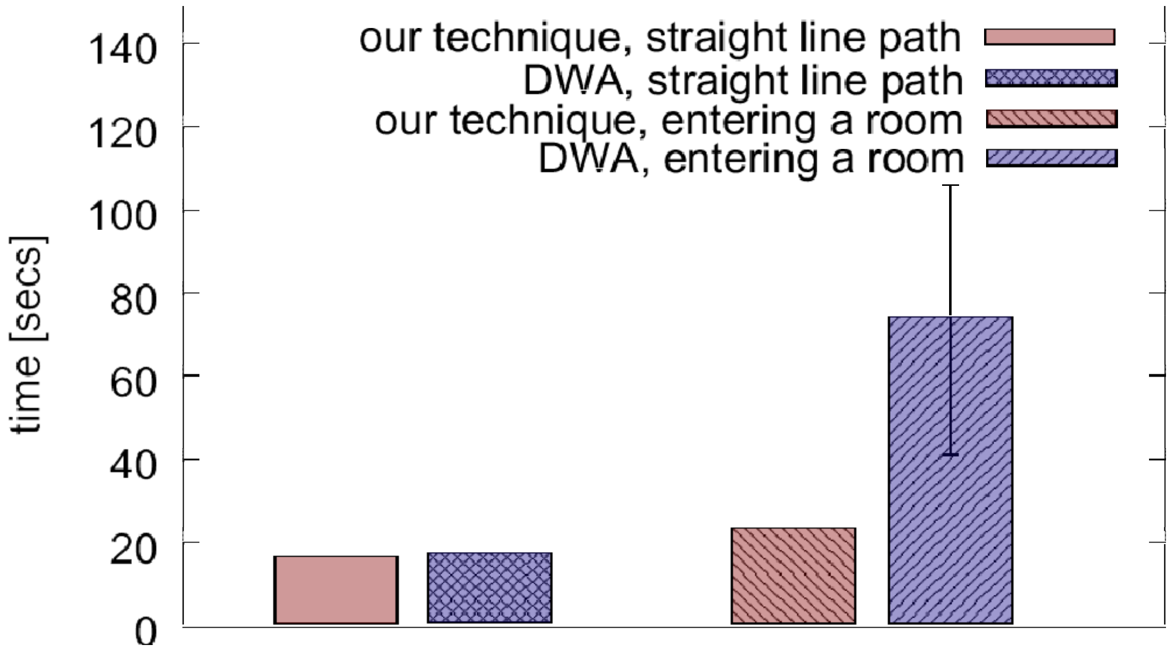
Comparison to the DWA (I)

- DWAs often have problems entering narrow passages.

DWA planned path.

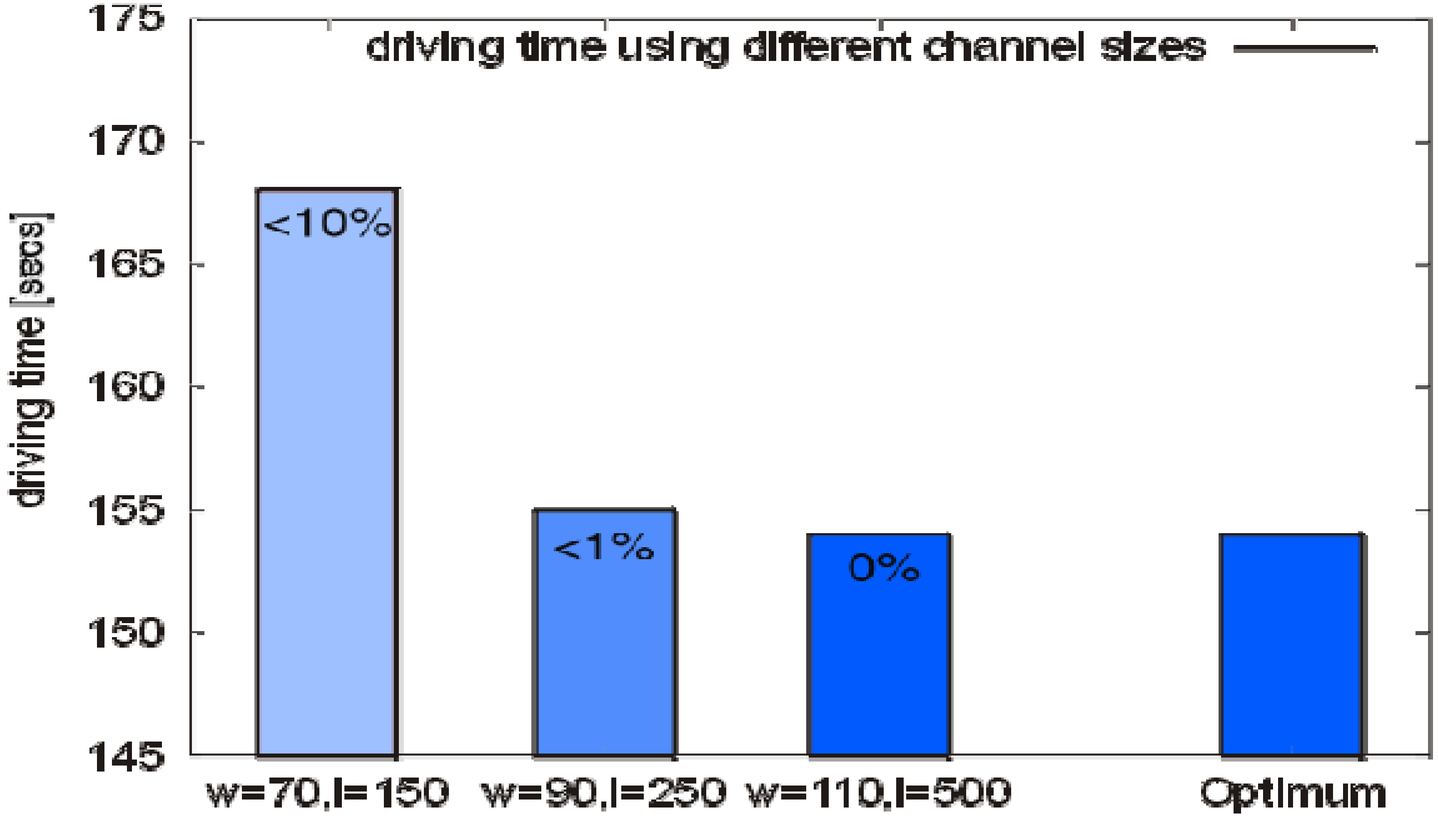
5D approach.

Comparison to the DWA (II)



The presented approach results in significantly faster motion when driving through narrow passages!

Comparison to the Optimum



Channel: with length=5m, width=1.1m
Resulting actions are close to the optimal solution.

Summary

- Robust navigation requires combined path planning & collision avoidance
- Approaches need to consider robot's kinematic constraints and plans in the velocity space.
- Combination of search and reactive techniques show better results than the pure DWA in a variety of situations.
- Using the 5D-approach the quality of the trajectory scales with the performance of the underlying hardware.
- The resulting paths are often close to the optimal ones.

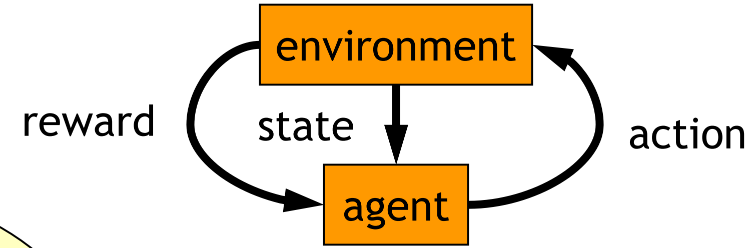
More

- More complex vehicles (e.g., cars).
- Moving obstacles, motion prediction.
- …

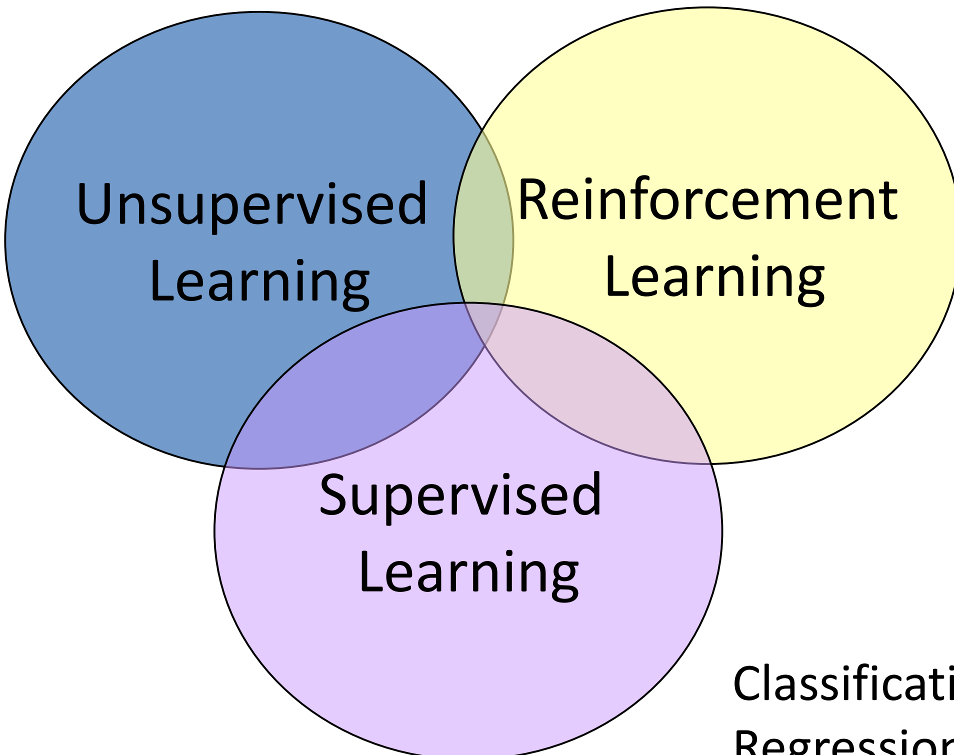
Outline

- Motion Planning Problem
- Potential Field Method
- Dynamic Window Approach
- A* Algorithm for Path Planning
- 5D Motion Planning
- Markov Decision Process

Machine Learning

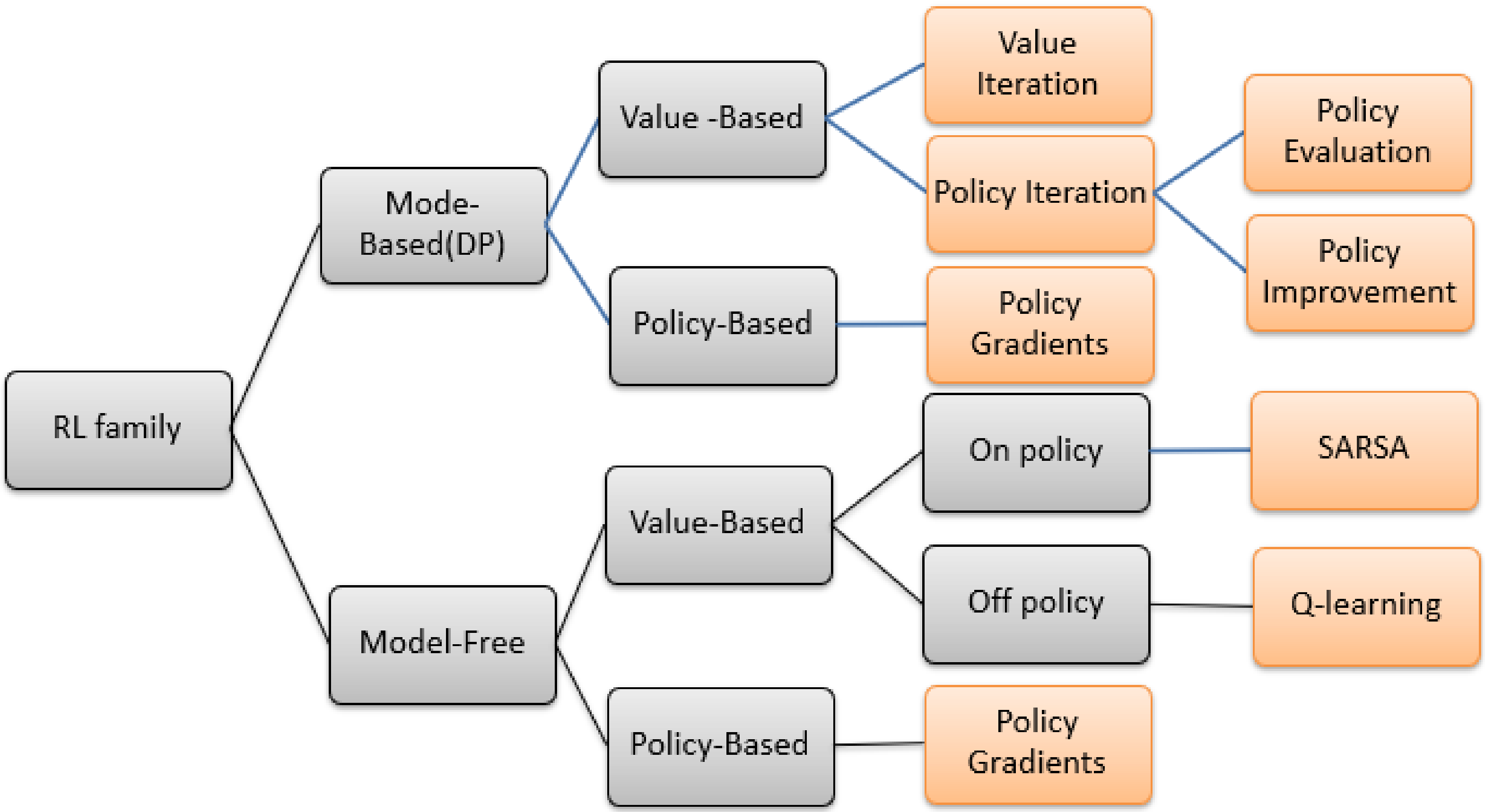


Clustering
Dimension reduction

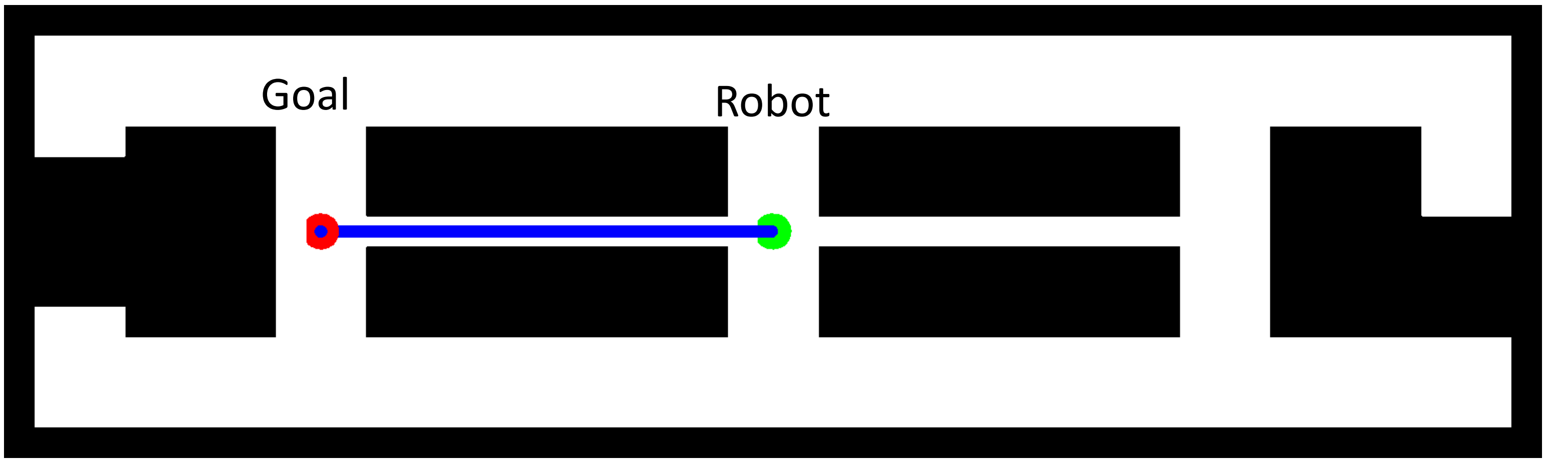


Classification
Regression

Reinforcement Learning

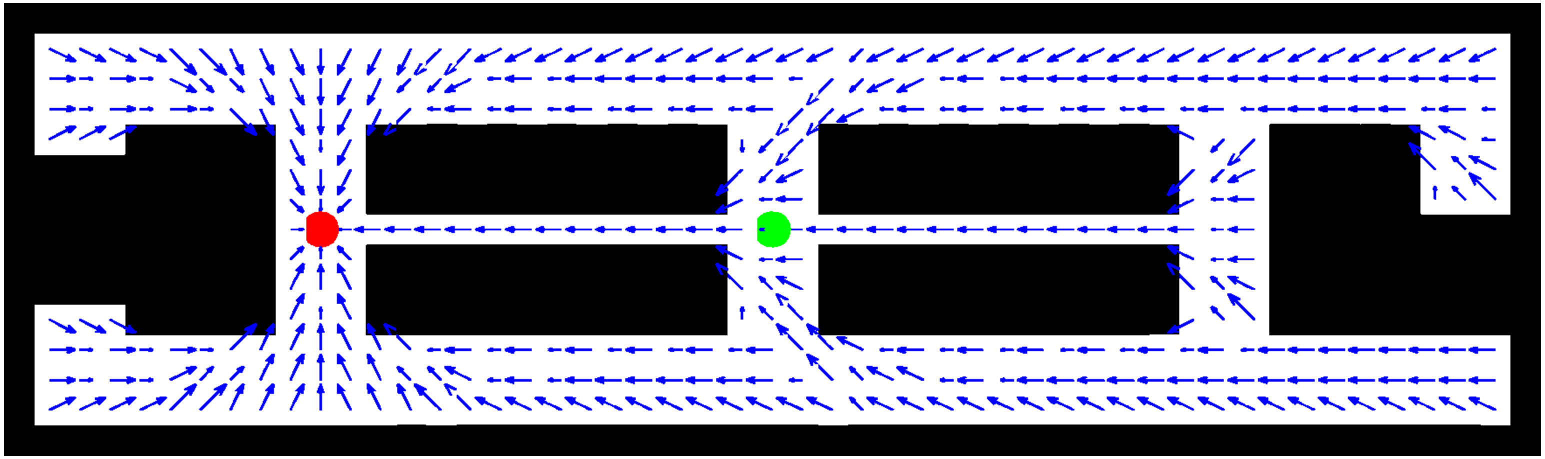


Robot Navigation Problem

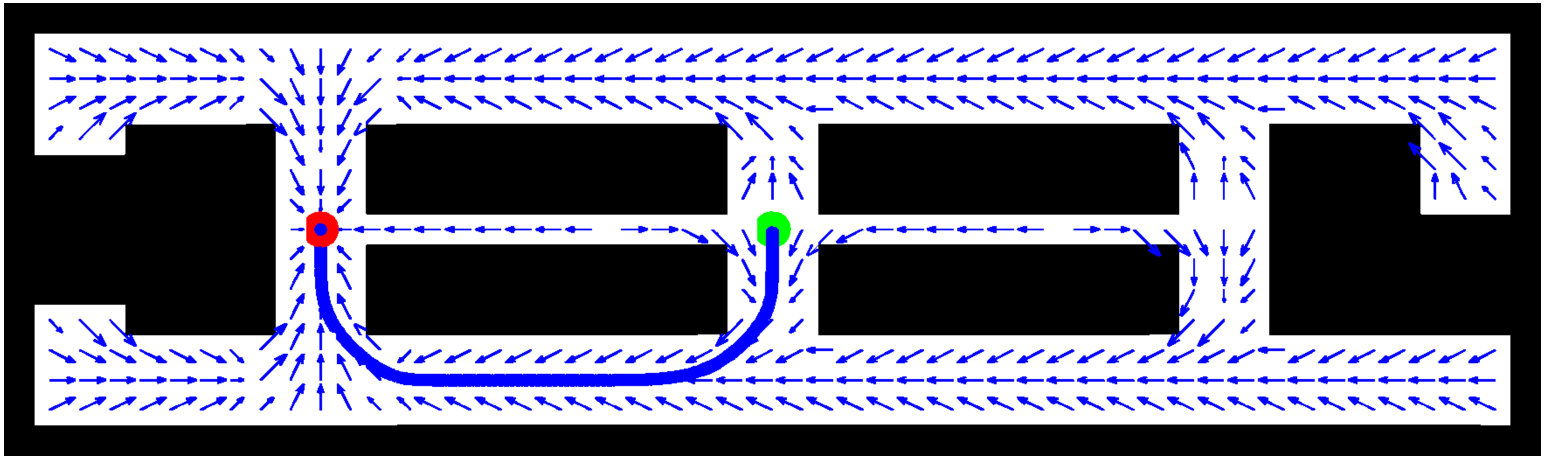


Uncertainty in Motion

without any uncertainty

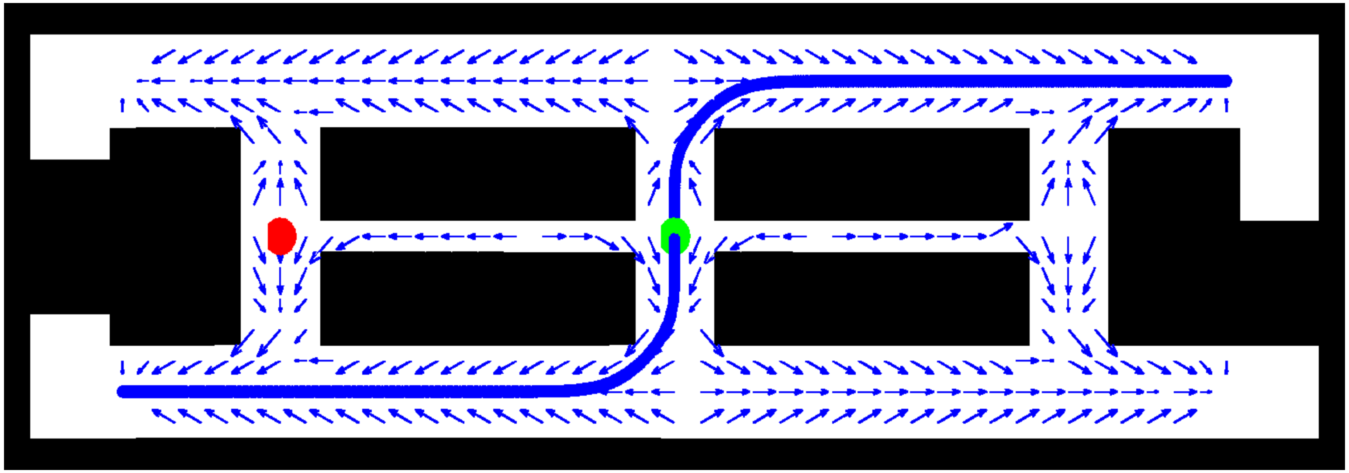


Given motion uncertainty, chose a path with larger space, avoiding the narrow corridor

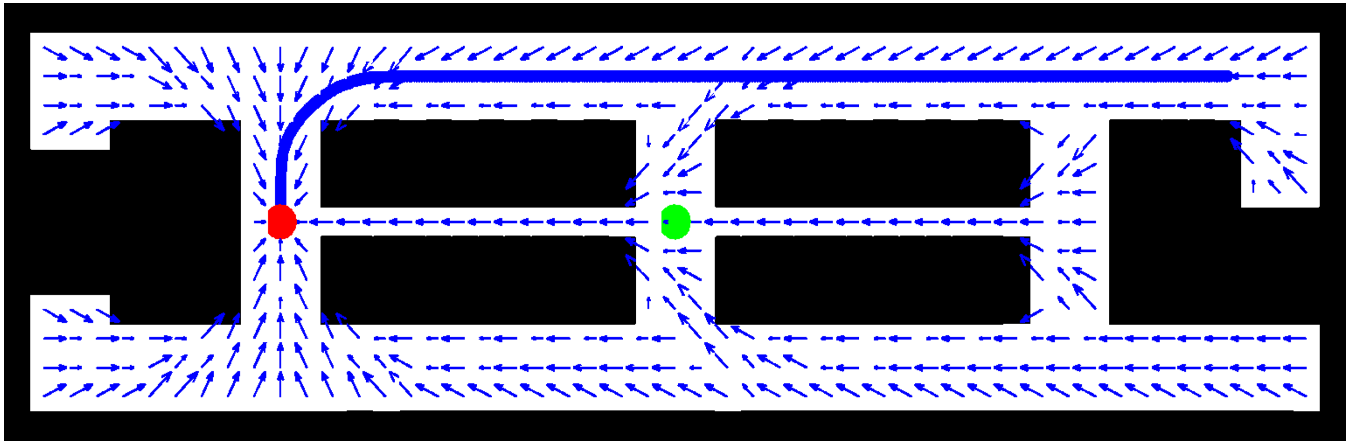


Uncertainty in Motion and Observation

1st Step:
Go to a state
with less
uncertainty
in observation



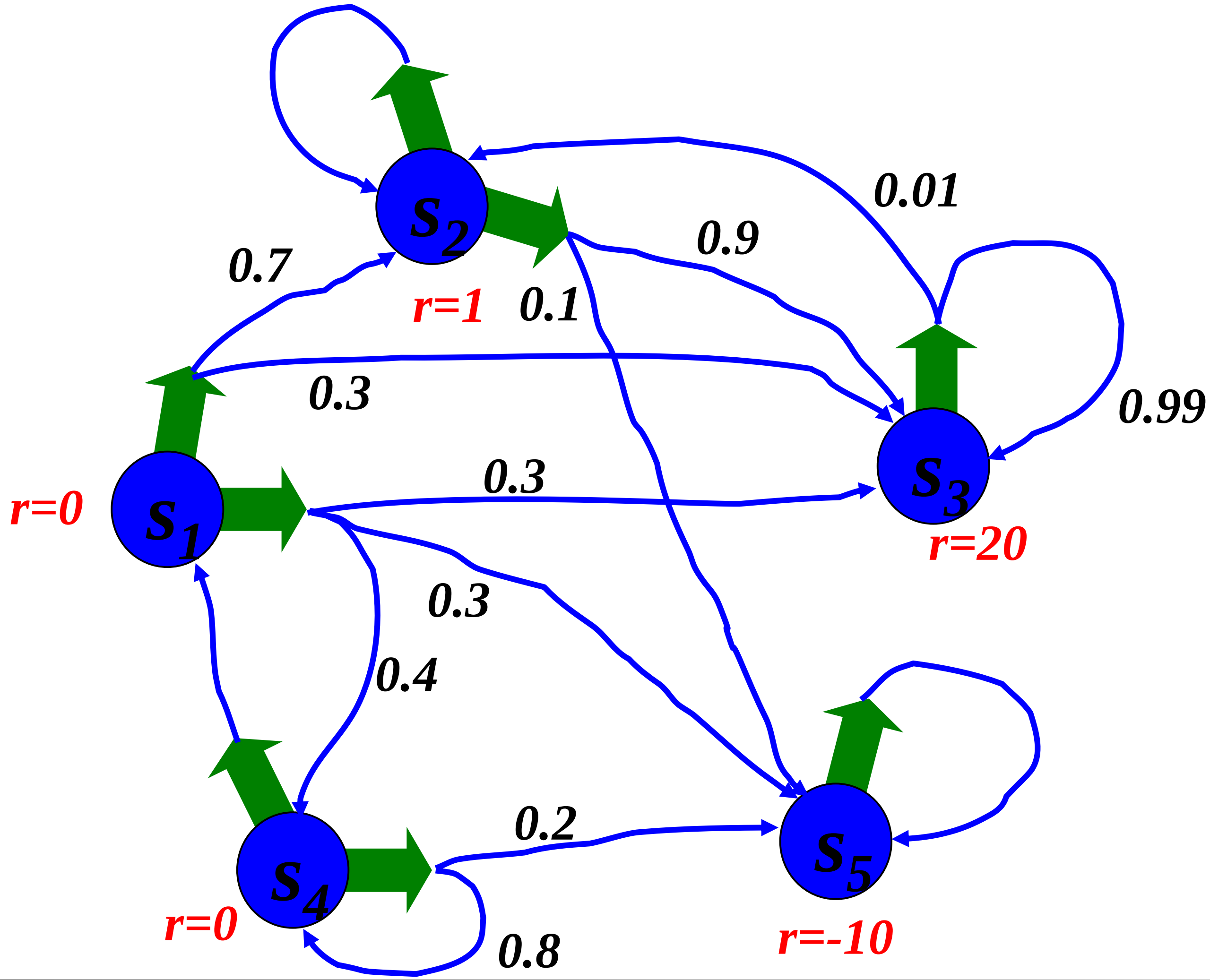
2nd Step:
Go to the goal
along a path
accommodating
more
uncertainty



Markov Decision Process

			RIGHT GOAL
	OBSTACLE		WRONG GOAL
START POSITION			

Markov Decision Process



Markov Decision Process Setup

- Given：

  States x, Actions u

  Transition probabilities $p(x'|u, x)$

  Reward function $r(x, u)$

- Wanted：

  Policy $\pi(x)$ that maximizes the future expected reward

Policy and Cumulative Reward

- Policy (general case): $\pi: \quad z_{1:t-1}, u_{1:t-1} \to \quad u_t$

- Policy (fully observable case): $\pi: \quad x_t \to \quad u_t$

- Expected cumulative reward: $R_T \quad = \quad E\left[\sum_{\tau=1}^{T} \gamma^\tau r_{t+\tau}\right]$

$$\boxed{R_\infty \quad \leq \quad r_{\max} + \gamma r_{\max} + \gamma^2 r_{\max} + \gamma^3 r_{\max} + \ldots \quad = \quad \frac{r_{\max}}{1-\gamma}}$$

T=1 : greedy policy
T>1 : finite horizon case, typically no discount
T=infinity: infinite-horizon case, finite reward if discount < 1

Optimal Policy

- Expected cumulative reward of policy:

$$R_T^\pi(x_t) \quad = \quad E\left[\sum_{\tau=1}^{T} \gamma^\tau r_{t+\tau} \mid u_{t+\tau} = \pi(z_{1:t+\tau-1}, u_{1:t+\tau-1})\right]$$

- Optimal policy:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \quad R_T^\pi(x_t)$$

1-Step Policy and Value Function

- 1-step optimal policy:

$$\pi_1(x) = \underset{u}{\operatorname{argmax}} \; r(x,u)$$

- Optimal value function of 1-step optimal policy:

$$V_1(x) = \gamma \max_u \; r(x,u)$$

2-Step Policy and Value Function

- 2-step optimal policy:

$$\pi_2(x) \quad = \quad \operatorname*{argmax}_u \left[\underbrace{r(x,u)}_{\boxed{\text{Current Reward}}} + \underbrace{\int V_1(x')\, p(x' \mid u, x)\, dx'}_{\text{Predicted Value}} \right]$$

- 2-step optimal value function:

$$V_2(x) \quad = \quad \gamma \max_u \left[\underbrace{r(x,u)}_{\boxed{\text{Current Reward}}} + \underbrace{\int V_1(x')\, p(x' \mid u, x)\, dx'}_{\text{Predicted Value}} \right]$$

T-Step Policy and Value Function

- T-step optimal policy:

$$\pi_T(x) \quad = \quad \operatorname*{argmax}_u \left[\underbrace{r(x,u)}_{\text{Current Reward}} + \underbrace{\int V_{T-1}(x')\, p(x' \mid u, x)\, dx'}_{\text{Predicted Value}} \right]$$

- T-step optimal value function:

$$V_T(x) \quad = \quad \gamma \max_u \left[\underbrace{r(x,u)}_{\text{Current Reward}} + \underbrace{\int V_{T-1}(x')\, p(x' \mid u, x)\, dx'}_{\text{Predicted Value}} \right]$$

Infinite Horizon (Bellman Equation)

- Optimal policy:

$$V_\infty(x) \;=\; \gamma \max_u \left[\underbrace{r(x,u)}_{\boxed{\text{Current Reward}}} + \underbrace{\int V_\infty(x')\, p(x' \mid u, x)\, dx'}_{\text{Predicted Value}} \right]$$

- Bellman Equation
  - ✓ Fix point is optimal policy
  - ✓ Necessary and sufficient condition

Value Iteration

for all *x* do

$$\hat{V} \longleftarrow r_{\min}$$

endfor

repeat until convergence

for all *x* do

$$\hat{V}(x) \longleftarrow \gamma \max_u \left[r(x,u) + \int \hat{V}(x')\, p(x' \mid u, x)\, dx' \right]$$

endfor

endrepeat

$$\pi(x) = \operatorname*{argmax}_u \left[r(x,u) + \int \hat{V}(x')\, p(x' \mid u, x)\, dx' \right]$$

Value Iteration Algorithm (I)

Algorithm MDP_discrete_value_iteration( ):

for $i = 1$ *to* N *do*

$\quad \hat{V}(x_i) = r_{\min}$

endfor

repeat until convergence

$\quad$ *for* $i = 1$ *to* N *do*

$$\hat{V}(x_i) = \gamma \max_u \left[r(x_i, u) + \sum_{j=1}^{N} \hat{V}(x_j)\, p(x_j \mid u, x_i) \right]$$

$\quad$ *endfor*

endrepeat

return $\hat{V}$

Value Iteration Algorithm (II)

1: **Algorithm policy_MDP($x, \hat{V}$):**

2: *return* $\operatorname*{argmax}_u \left[r(x,u) + \sum_{j=1}^{N} \hat{V}(x_j)\, p(x_j \mid u, x_i) \right]$

Shortest Path

State g's reward=0， other reward=-1.0, γ=1

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

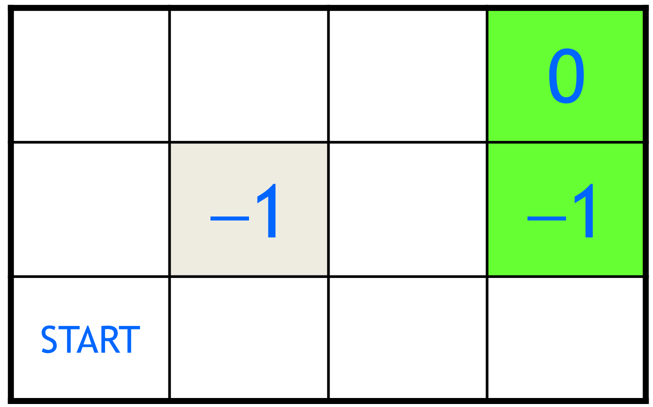
0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

MDP Model



Environment and reward:

a) Green rectangle: destination, reward = 0 for any action
b) Black rectangle : wall, reward = -1
c) reward = - 0.1 for each step in other states
d) action = {up, down, left, right}

MDP Model

0	1	2	3
4	5	6	7
8	9	10	11

a) Position 3: reward = 0 for any action
b) Positions 5 and 7: wall, reward = -1
c) reward = - 0.1 for each step in other states
d) action = {up/0, down/1, left/2, right/3}

transition probabilities:

$$\{x: \{u_1: (x', p(x'|x, u_1), r), u_2: (x', p(x'|x, u_2), r), u_3: (x', p(x'|x, u_3), r), u_4: (x', p(x'|x, u_4), r)\ \}\}$$

```
{0: {0: (0, 1.0, -0.1), 1: (4, 1.0, -0.1), 3: (1, 1.0, -0.1), 2: (0, 1.0, -0.1)}, 1: {0: (1,
1.0, -0.1), 1: (1, 1.0, -1), 3: (2, 1.0, -0.1), 2: (0, 1.0, -0.1)}, 2: {0: (2, 1.0, -0.1), 1:
(6, 1.0, -0.1), 3: (3, 1.0, -0.1), 2: (1, 1.0, -0.1)}, 3: {0: (3, 1.0, 0), 1: (3, 1.0, 0), 3:
(3, 1.0, 0), 2: (3, 1.0, 0)}, 4: {0: (0, 1.0, -0.1), 1: (8, 1.0, -0.1), 3: (4, 1.0, -1), 2: (4,
1.0, -0.1)}, 5: {0: (1, 1.0, -0.1), 1: (9, 1.0, -0.1), 3: (6, 1.0, -0.1), 2: (4, 1.0, -0.1)}, 6:
{0: (2, 1.0, -0.1), 1: (10, 1.0, -0.1), 3: (6, 1.0, -1), 2: (6, 1.0, -1)}, 7: {0: (3, 1.0,
-0.1), 1: (11, 1.0, -0.1), 3: (7, 1.0, -1), 2: (6, 1.0, -0.1)}, 8: {0: (4, 1.0, -0.1), 1: (8,
1.0, -0.1), 3: (9, 1.0, -0.1), 2: (8, 1.0, -0.1)}, 9: {0: (9, 1.0, -1), 1: (9, 1.0, -0.1), 3:
(10, 1.0, -0.1), 2: (8, 1.0, -0.1)}, 10: {0: (6, 1.0, -0.1), 1: (10, 1.0, -0.1), 3: (11, 1.0,
-0.1), 2: (9, 1.0, -0.1)}, 11: {0: (11, 1.0, -1), 1: (11, 1.0, -0.1), 3: (11, 1.0, -0.1), 2:
(10, 1.0, -0.1)}}
```

Value Iteration (I)

Value Function V^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

$\boxed{V^0(0) = 0.0}$ $\boxed{V^1(0) = -0.1}$

$r(0, up) + V^0(0)*p(0|0,up) = -0.1 + (-0.0)*1 = -0.1$

$r(0, do) + V^0(4)*p(4|0,do) = -0.1 + (-0.0)*1 = -0.1$

$\boxed{r(0, rig) + V^0(1)*p(1|0,rig) = -0.1 + (-0.0)*1 = -0.1}$

$r(0, lef) + V^0(0)*p(0|0,lef) = -0.1 + (-0.0)*1 = -0.1$

0	1	2	3
4	5	6	7
8	9	10	11

$\boxed{V^0(1) = 0.0}$ $\boxed{V^1(1) = -0.1}$

$r(1, up) + V^0(1)*p(1|1,up) = -0.1 + (-0.0)*1 = -0.1$

$r(1, do) + V^0(1)*p(1|1,do) = -1.0 + (-0.0)*1 = -1.0$

$\boxed{r(1, rig) + V^0(2)*p(2|1,rig) = -0.1 + (-0.0)*1 = -0.1}$

$r(1, lef) + V^0(0)*p(0|1,lef) = -0.1 + (-0.0)*1 = -0.1$

Value Iteration (II)

Value Function V^1

– 0.1	– 0.1	– 0.1	0.0
– 0.1	0.0	– 0.1	0.0
– 0.1	– 0.1	– 0.1	– 0.1

0	1	2	3
4	5	6	7
8	9	10	11

$\boxed{V^1(0) = -0.1}$ $\boxed{V^2(0) = -0.2}$

$r(0, up) + V^1(0)*p(0|0,up) = -0.1 + (-0.1)*1 = -0.2$

$r(0, do) + V^1(4)*p(4|0,do) = -0.1 + (-0.1)*1 = -0.2$

$\boxed{r(0, rig) + V^1(1)*p(1|0,rig) = -0.1 + (-0.1)*1 = -0.2}$

$r(0, lef) + V^1(0)*p(0|0,lef) = -0.1 + (-0.1)*1 = -0.2$

$\boxed{V^1(1) = -0.1}$ $\boxed{V^2(1) = -0.2}$

$r(1, up) + V^1(1)*p(1|1,up) = -0.1 + (-0.1)*1 = -0.2$

$r(1, do) + V^1(1)*p(1|1,do) = -1.0 + (-0.1)*1 = -1.1$

$\boxed{r(1, rig) + V^1(2)*p(2|1,rig) = -0.1 + (-0.1)*1 = -0.2}$

$r(1, lef) + V^1(0)*p(0|1,lef) = -0.1 + (-0.1)*1 = -0.2$

Value Iteration (III)

Value Function V^2

– 0.2	– 0.2	– 0.1	0.0
– 0.2	0.0	– 0.2	0.0
– 0.2	– 0.2	– 0.2	– 0.2

$\boxed{V^2(0) = -0.2}$ $\boxed{V^3(0) = -0.3}$

$r(0, up) + V^2(0)*p(0|0,up) = -0.1 + (-0.2)*1 = -0.3$

$r(0, do) + V^2(4)*p(4|0,do) = -0.1 + (-0.2)*1 = -0.3$

$\boxed{r(0, rig) + V^2(1)*p(1|0,rig) = -0.1 + (-0.2)*1 = -0.3}$

$r(0, lef) + V^2(0)*p(0|0,lef) = -0.1 + (-0.2)*1 = -0.3$

0	1	2	3
4	5	6	7
8	9	10	11

$\boxed{V^2(1) = -0.2}$ $\boxed{V^3(1) = -0.2}$

$r(1, up) + V^2(1)*p(1|1,up) = -0.1 + (-0.2)*1 = -0.3$

$r(1, do) + V^2(1)*p(1|1,do) = -1.0 + (-0.2)*1 = -1.2$

$\boxed{r(1, rig) + V^2(2)*p(2|1,rig) = -0.1 + (-0.1)*1 = -0.2}$

$r(1, lef) + V^2(0)*p(0|1,lef) = -0.1 + (-0.2)*1 = -0.3$

Value Iteration (IV)

Value Function V^3

– 0.3	– 0.2	– 0.1	0.0
– 0.3	0.0	– 0.2	0.0
– 0.3	– 0.3	– 0.3	– 0.3

0	1	2	3
4	5	6	7
8	9	10	11

$\boxed{V^3(0) = -0.2}$ $\boxed{V^4(0) = -0.3}$

$r(0, up) + V^3(0)*p(0|0,up) = -0.1 + (-0.3)*1 = -0.4$

$r(0, do) + V^3(4)*p(4|0,do) = -0.1 + (-0.3)*1 = -0.4$

$\boxed{r(0, rig) + V^3(1)*p(1|0,rig) = -0.1 + (-0.2)*1 = -0.3}$

$r(0, lef) + V^3(0)*p(0|0,lef) = -0.1 + (-0.3)*1 = -0.4$

$\boxed{V^3(1) = -0.2}$ $\boxed{V^4(1) = -0.2}$

$r(1, up) + V^3(1)*p(1|1,up) = -0.1 + (-0.2)*1 = -0.3$

$r(1, do) + V^3(1)*p(1|1,do) = -1.0 + (-0.2)*1 = -1.2$

$\boxed{r(1, rig) + V^3(2)*p(2|1,rig) = -0.1 + (-0.1)*1 = -0.2}$

$r(1, lef) + V^3(0)*p(0|1,lef) = -0.1 + (-0.3)*1 = -0.4$

Value Iteration (V)

Value Function V^4

– 0.3	– 0.2	– 0.1	0.0
– 0.4	0.0	– 0.2	0.0
– 0.4	– 0.4	– 0.3	– 0.4

$V^4(0) = -0.2$ $V^5(0) = -0.3$

$r(0, up) + V^1(0)*p(0|0,up) = -0.1 + (-0.3)*1 = -0.4$

$r(0, do) + V^1(4)*p(4|0,do) = -0.1 + (-0.4)*1 = -0.5$

$\boxed{r(0, rig) + V^1(1)*p(1|0,rig) = -0.1 + (-0.2)*1 = -0.3}$

$r(0, lef) + V^1(0)*p(0|0,lef) = -0.1 + (-0.3)*1 = -0.4$

0	1	2	3
4	5	6	7
8	9	10	11

$V^4(1) = -0.2$ $V^5(1) = -0.2$

$r(1, up) + V^1(1)*p(1|1,up) = -0.1 + (-0.2)*1 = -0.3$

$r(1, do) + V^1(1)*p(1|1,do) = -1.0 + (-0.2)*1 = -1.2$

$\boxed{r(1, rig) + V^1(2)*p(2|1,rig) = -0.1 + (-0.1)*1 = -0.2}$

$r(1, lef) + V^1(0)*p(0|1,lef) = -0.1 + (-0.3)*1 = -0.4$

Stationary Value Function

Stationary Value Function

– 0.3	– 0.2	– 0.1	0.0
– 0.4	0.0	– 0.2	0.0
– 0.5	– 0.4	– 0.3	– 0.4

0	1	2	3
4	5	6	7
8	9	10	11

$\boxed{V(0) = -0.3}$

r(0, up) + V(0)*p(0|0,up) = – 0.1+ (– 0.3)*1= – 0.4

r(0, do) + V(4)*p(4|0,do) = – 0.1+ (– 0.4)*1= – 0.5

$\boxed{r(0, rig) + V(1)*p(1|0,rig) = -0.1+ (-0.2)*1= -0.3}$

r(0, lef) + V(0)*p(0|0,lef) = – 0.1+ (– 0.3)*1= – 0.4

$\boxed{V(1) = -0.2}$

r(1, up) + V(1)*p(1|1,up) = – 0.1+ (– 0.2)*1= – 0.3

r(1, do) + V(1)*p(1|1,do) = – 1.0+ (– 0.2)*1= – 1.0

$\boxed{r(1, rig) + V(2)*p(2|1,rig) = -0.1+ (-0.1)*1= -0.2}$

r(1, lef) + V(0)*p(0|1,lef) = – 0.1+ (– 0.3)*1= – 0.4

Optimal Policy for Value Iteration

Stationary Value Function

–0.3	–0.2	–0.1	0.0
–0.4	–0.0	–0.2	–0.0
–0.5	–0.4	–0.3	–0.4

V(0) = – 0.3

Optimal Action: right →

r(0, up) + V(0)*p(0|0,up) = – 0.1+ (– 0.3)*1= – 0.4

r(0, do) + V(4)*p(4|0,do) = – 0.1+ (– 0.4)*1= – 0.5

r(0, rig) + V(1)*p(1|0,rig) = – 0.1+ (– 0.2)*1= – 0.3

r(0, lef) + V(0)*p(1|0,lef) = – 0.1+ (– 0.3)*1= – 0.4

Optimal Policy

→	→	→	●
↑	□	↑	□
↑ →	→	↑	←

V(1) = – 0.2

Optimal Action: right →

r(1, up) + V(1)*p(1|1,up) = – 0.1+ (– 0.2)*1= – 0.3

r(1, do) + V(1)*p(1|1,do) = – 1.0+ (– 0.0)*1= – 1.0

r(1, rig) + V(2)*p(2|1,rig) = – 0.1+ (– 0.1)*1= – 0.2

r(1, lef) + V(0)*p(0|1,lef) = – 0.1+ (– 0.3)*1= – 0.4

Value Iteration

```
No. of states in grid: 12
No. of action options in each state:4
Best policy with Value Iteration is
[[3. 3. 3. 5.]
 [0. 7. 0. 7.]
 [0. 3. 0. 2.]]
Corresponding Value Function is
[[-0.3 -0.2 -0.1  0. ]
 [-0.4 13.  -0.2 13. ]
 [-0.5 -0.4 -0.3 -0.4]]
Time taken
0.0015388405049634457
Our Value Function:
[[-0.3 -0.2 -0.1  0. ]
 [-0.4 13.  -0.2 13. ]
 [-0.5 -0.4 -0.3 -0.4]]
Index for actions:
0 : up
1 : down
2 : left
3 : right
5 : terminal states (stay)
7 : wall
```

Policy Iteration

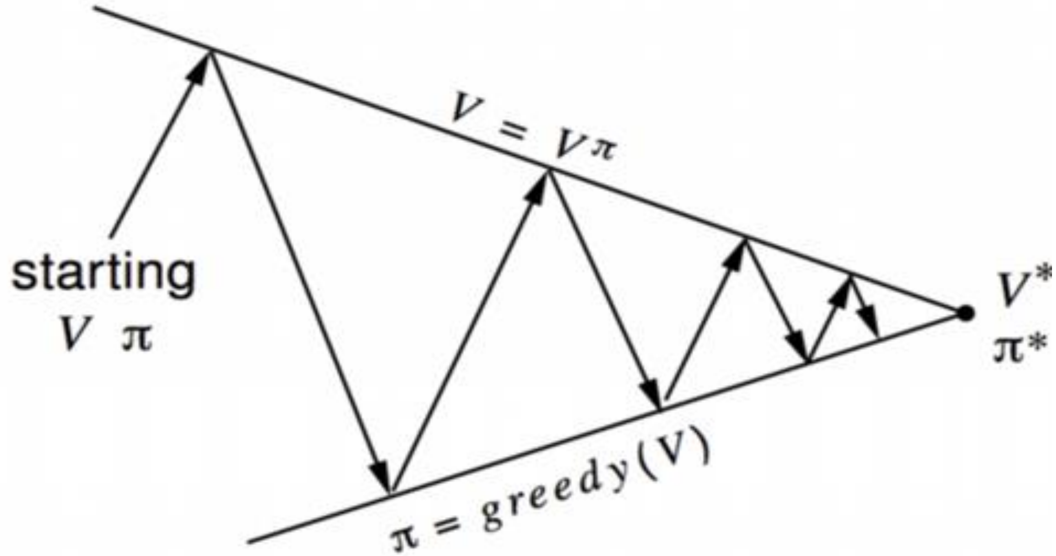
- Often the optimal policy has been reached long before the value function has converged.
- Policy iteration (1) calculates a new policy based on the current value function and (2) then calculates a new value function based on this policy.

(1) Policy improvement $\pi^* = \underset{\pi}{\operatorname{argmax}} \quad R_T^\pi(x_t)$

(2) Policy evaluation
$$R_T^\pi(x_t) = E \left[\sum_{\tau=1}^{T} \gamma^\tau r_{t+\tau} \mid u_{t+\tau} = \pi(z_{1:t+\tau-1} u_{1:t+\tau-1}) \right]$$

- Often converges faster to the optimal policy.

Policy Iteration



Policy evaluation Estimate v_π

Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement

Policy Iteration Algorithm

1. Initialization
   $v(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\quad \Delta \leftarrow 0$
   $\quad$ For each $s \in \mathcal{S}$:
   $\quad\quad temp \leftarrow v(s)$
   $\quad\quad v(s) \leftarrow \sum_{s'} p(s'|s,\pi(s))\left[r(s,\pi(s),s') + \gamma v(s')\right]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |temp - v(s)|)$
   until $\Delta < \theta$ (a small positive number)

Finding value function

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad temp \leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma v(s')\right]$
   $\quad$ If $temp \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return v and π; else go to 2

Value Iteration Algorithm

Finding the optimal value function

Initialize array v arbitrarily (e.g., $v(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $temp \leftarrow v(s)$
    $v(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma v(s')]$
    $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} p(s'|s,a)\Big[r(s,a,s') + \gamma v(s')\Big]$

Extract the policy from the optimal value function

Evaluation of the Random Policy (I)

Target reward=0， other reward=-1.0, γ=1

v_k for the Random Policy

$k = 0$

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	0.0

-1.0 = 0.25*(-1+1.0*0)+0.25*(-1+1.0*0)
+ 0.25*(-1+1.0*0)+0.25*(-1+1.0*0)

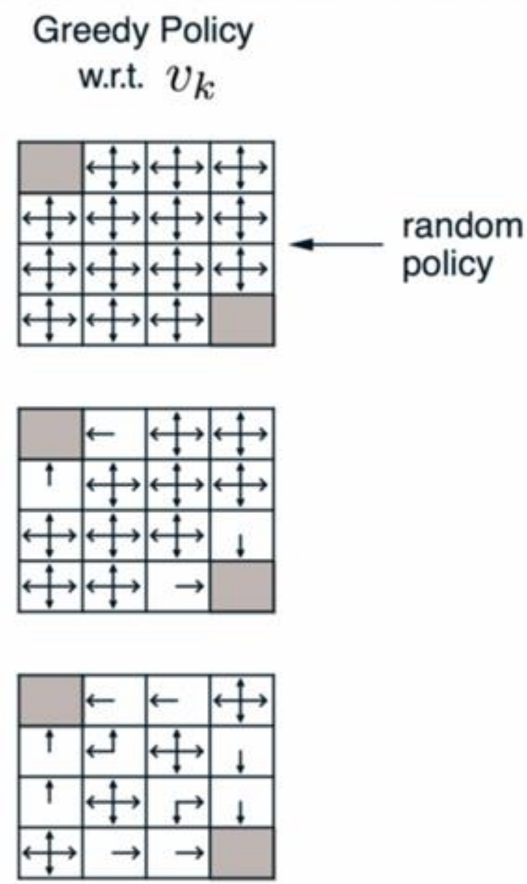
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

-1.7 = 0.25*(-1+1.0*0)+0.25*(-1+1.0*-1)
+ 0.25*(-1+1.0*1)+0.25*(-1+1.0*-1)

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s)\left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s')\right)$$

Evaluation of the Random Policy (II)

-2.4 = 0.25*(-1+1.0*0)+0.25*(-1+1.0*-2.0)
+ 0.25*(-1+1.0*2)+0.25*(-1+1.0*-1.7)

$k = 3$

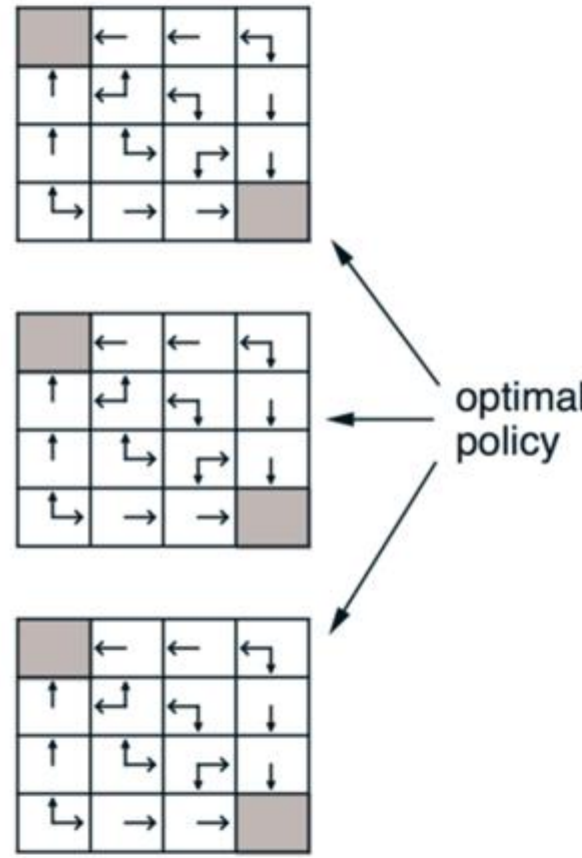
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

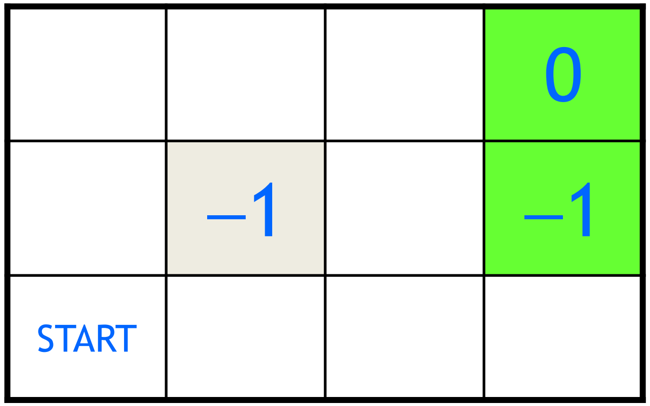
$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s)\left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s')\right)$$

MDP Model



Environment and reward:

a) Green rectangle: destination, reward = 0 for any action
b) Black rectangle : wall, reward = -1
c) reward = - 0.1 for each step in other states
d) action = {up, down, left, right}

MDP Model

0	1	2	3
4	5	6	7
8	9	10	11

a) Position 3: reward = 0 for any action
b) Positions 5 and 7: wall, reward = -1
c) reward = - 0.1 for each step in other states
d) action = {up/0, down/1, left/2, right/3}

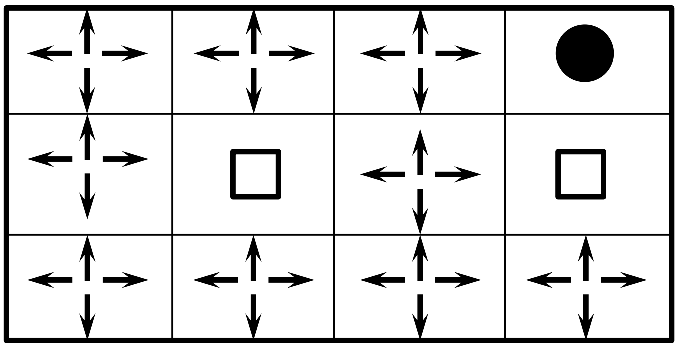
transition probabilities:

$$\{x\text{: }\{u_1\text{: }(x', p(x'|x, u_1), r), u_2\text{: }(x', p(x'|x, u_2), r),\; u_3\text{: }(x', p(x'|x, u_3), r), u_4\text{: }(x', p(x'|x, u_4), r)\ \}\}$$

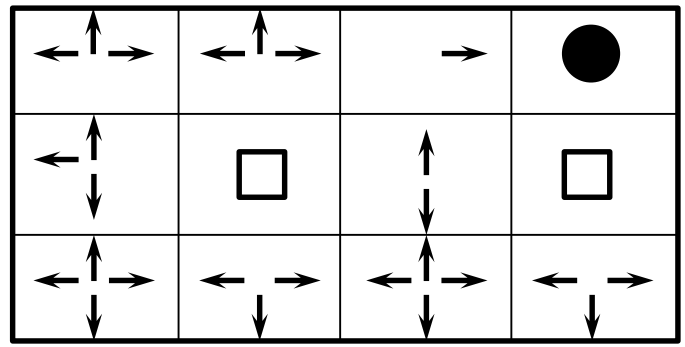
```
{0: {0: (0, 1.0, -0.1), 1: (4, 1.0, -0.1), 3: (1, 1.0, -0.1), 2: (0, 1.0, -0.1)}, 1: {0: (1,
1.0, -0.1), 1: (1, 1.0, -1), 3: (2, 1.0, -0.1), 2: (0, 1.0, -0.1)}, 2: {0: (2, 1.0, -0.1), 1:
(6, 1.0, -0.1), 3: (3, 1.0, -0.1), 2: (1, 1.0, -0.1)}, 3: {0: (3, 1.0, 0), 1: (3, 1.0, 0), 3:
(3, 1.0, 0), 2: (3, 1.0, 0)}, 4: {0: (0, 1.0, -0.1), 1: (8, 1.0, -0.1), 3: (4, 1.0, -1), 2: (4,
1.0, -0.1)}, 5: {0: (1, 1.0, -0.1), 1: (9, 1.0, -0.1), 3: (6, 1.0, -0.1), 2: (4, 1.0, -0.1)}, 6:
{0: (2, 1.0, -0.1), 1: (10, 1.0, -0.1), 3: (6, 1.0, -1), 2: (6, 1.0, -1)}, 7: {0: (3, 1.0,
-0.1), 1: (11, 1.0, -0.1), 3: (7, 1.0, -1), 2: (6, 1.0, -0.1)}, 8: {0: (4, 1.0, -0.1), 1: (8,
1.0, -0.1), 3: (9, 1.0, -0.1), 2: (8, 1.0, -0.1)}, 9: {0: (9, 1.0, -1), 1: (9, 1.0, -0.1), 3:
(10, 1.0, -0.1), 2: (8, 1.0, -0.1)}, 10: {0: (6, 1.0, -0.1), 1: (10, 1.0, -0.1), 3: (11, 1.0,
-0.1), 2: (9, 1.0, -0.1)}, 11: {0: (11, 1.0, -1), 1: (11, 1.0, -0.1), 3: (11, 1.0, -0.1), 2:
(10, 1.0, -0.1)}}
```

Policy Iteration (I)

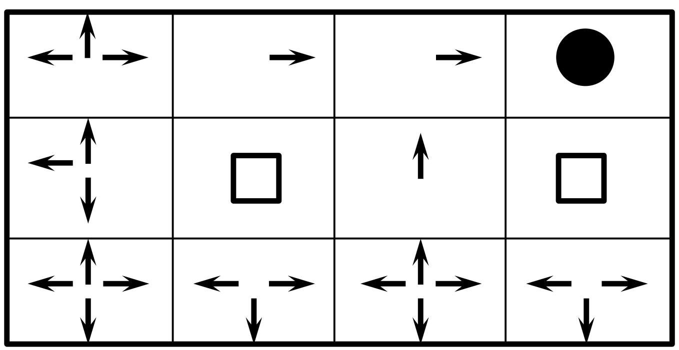
Policy π^0



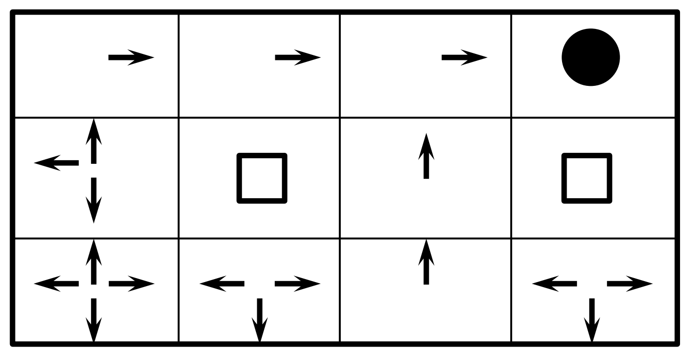
Policy π^1



Policy π^2

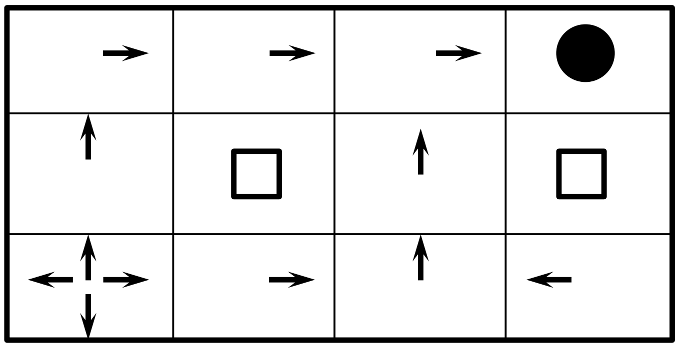


Policy π^3

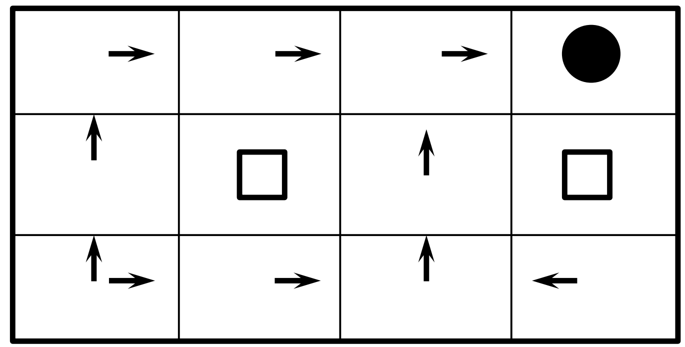


Policy Iteration (II)

Policy π^4



Policy π^5



Value Function

–0.3	–0.2	–0.1	0.0
–0.4	–0.0	–0.2	–0.0
–0.5	–0.4	–0.3	–0.4

Policy Iteration

```
No. of states in grid: 12
No. of action options in each state:4
Index for actions:
0 : up
1 : down
2 : left
3 : right
5 : terminal states (stay)
7 : wall
value:
[-14.92236088 -12.59499541  -8.97031084   0.          -16.85304649
 -15.37209386 -13.91780972 -11.05994835 -17.48748203 -17.7258608
 -16.66812736 -17.96429873]
value:
[-0.3 -0.2 -0.1  0.  -0.4 -0.3 -0.2 -0.1 -0.5 -0.4 -0.3 -0.4]
Best policy with Policy Iteration is
[[3. 3. 3. 5.]
 [0. 7. 0. 7.]
 [0. 3. 0. 2.]]
Corresponding Value Function is
[[-0.3 -0.2 -0.1  0. ]
 [-0.4 13.   -0.2 13. ]
 [-0.5 -0.4 -0.3 -0.4]]
Time taken
0.032080131208203966
```

Summary

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

Homework 2

Problem 1:

- How to generate uniform, perpendicular, attractive, repulse, tangential forces for a robot and obstacles with known positions? (Provide related mathematical formulas)

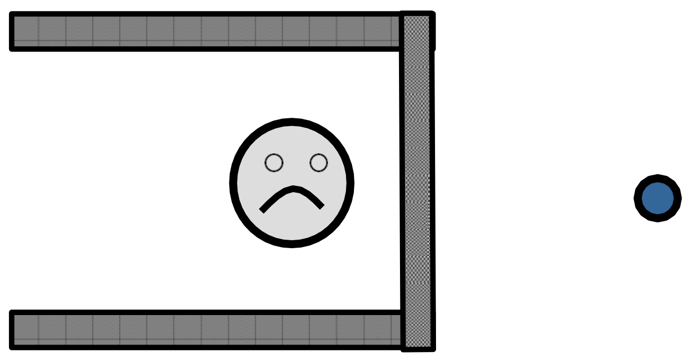
- Please simulate the above force fields, and plot the vector force fields. (provide codes and plots of force fields)

- Please simulate the motions of a robot for given those force fields. (Provide codes and Plots of simulation results)

Homework 2

Problem 2:

- Simulate a robot that can reach the goal without sticking into the a local trap. (Provide codes and simulation results with parameter optimization and analysis)



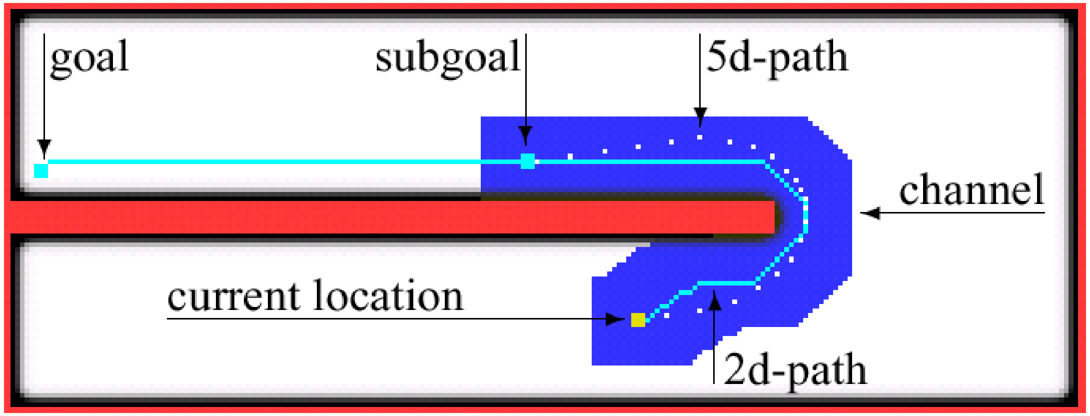
Homework 3

Problem 1: Please use the dynamic window approach to achieve the goal with obstacle avoidance.

			+1
	WALL -1		-1
START			

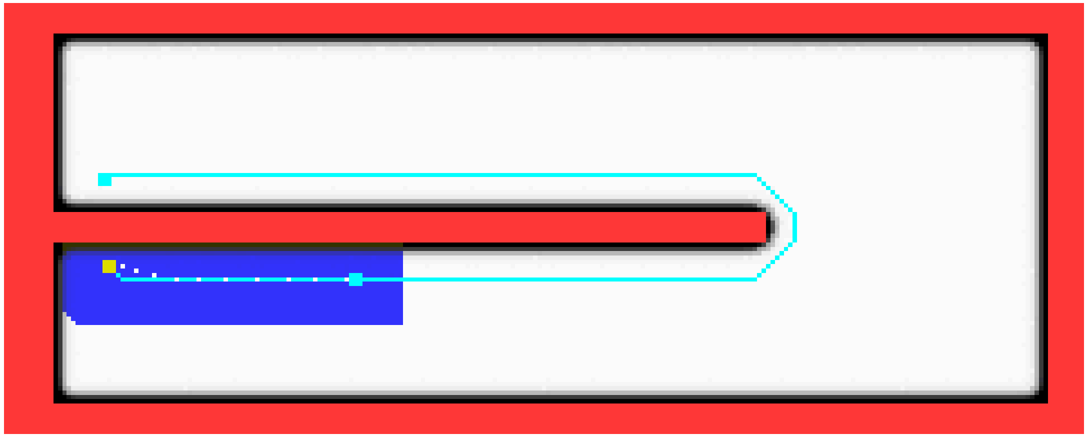
Homework 3

Problem 2: Please use A* and 5d planning to plan the following trajectories, respectively.



Homework 3

Problem 3: Please use the convolution map and A* together to plan the following trajectory.



Homework 4

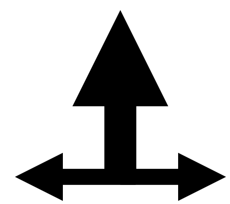
Problem 1: Use value iteration and policy iteration, respectively, to simulate the MDP-based navigation of the robot in the room.

			+1
	WALL -1		-1
START			

actions: UP, DOWN, LEFT, RIGHT

UP

80% move UP
10% move LEFT
10% move RIGHT



reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step