Markov Decision Processes, Reinforcement Learning and Policy Reuse

Manuela Veloso
Co-instructor: Pradeep Ravikumar
Thanks to past instructors

Machine Learning
April 23-25, 2018

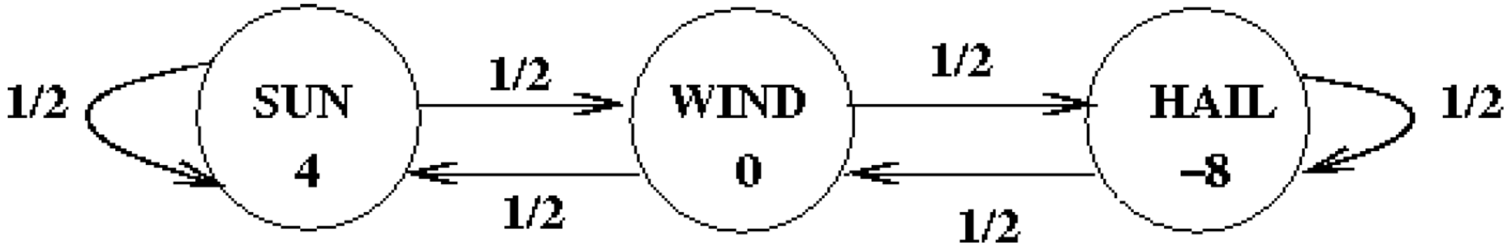
Readings:

- Reinforcement Learning: An Introduction, R. Sutton and A. Barto
- Probabilistic policy reuse in a reinforcement learning agent, Fernando Fernandez and Manuela Veloso. In *Proceedings of AAMAS'06.* (Thanks to Fernando Fernandez)

Markov Models

	Passive	Controlled
Fully Observable	Markov Systems with Rewards	MDP
Hidden State	HMM	POMDP
Time Dependent	Semi-Markov	SMDP

Example – Markov System with Reward



- States
- Rewards in states
- Probabilistic transitions between states
- Markov: transitions only depend on current state

Markov Systems with Rewards

- Finite set of n states, s_i
- Probabilistic state matrix, P, p_{ij}
- "Goal achievement" - Reward for each state, r_i
- Discount factor - γ
- Process/observation:
  - Assume start state s_i
  - Receive immediate reward r_i
  - Move, or observe a move, randomly to a new state according to the probability transition matrix
  - Future rewards (of next state) are discounted by γ

Solving a Markov System with Rewards

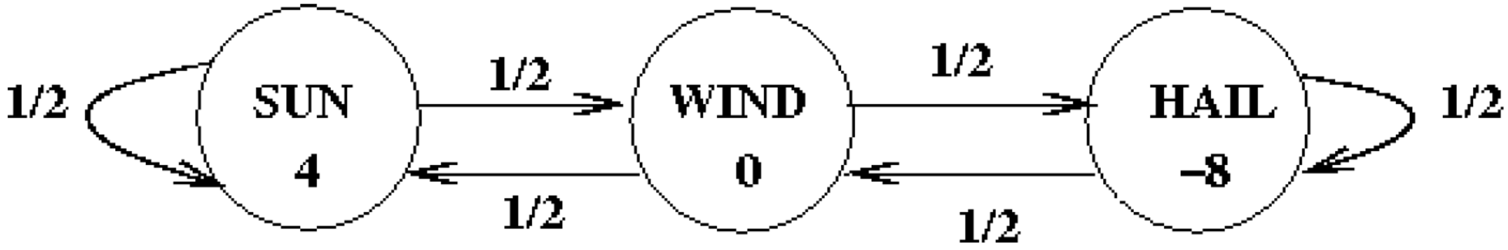
- $V^*(s_i)$ - expected discounted sum of future rewards starting in state s_i

- $V^*(s_i) = r_i + \gamma[p_{i1}V^*(s_1) + p_{i2}V^*(s_2) + \ldots\ p_{in}V^*(s_n)]$

Value Iteration to Solve a Markov System with Rewards

- $V^1(s_i)$ - expected discounted sum of future rewards starting in state s_i *for one step*.
- $V^2(s_i)$ - expected discounted sum of future rewards starting in state s_i *for two steps*.
- …
- $V^k(s_i)$ - expected discounted sum of future rewards starting in state s_i *for k steps*.
- As $k \longrightarrow \infty V^k(s_i) \longrightarrow V^*(s_i)$
- Stop when difference of k + 1 and k values is smaller than some $\in$.

3-State Example



3-State Example: Values $\gamma = 0.5$

Iteration	SUN	WIND	HAIL
0	0	0	0
1	4	0	-8
2	5.0	-1.0	-10.0
3	5.0	-1.25	-10.75
4	4.9375	-1.4375	-11.0
5	4.875	-1.515625	-11.109375
6	4.8398437	-1.5585937	-11.15625
7	4.8203125	-1.5791016	-11.178711
8	4.8103027	-1.5895996	-11.189453
9	4.805176	-1.5947876	-11.194763
10	4.802597	-1.5973969	-11.197388
11	4.8013	-1.5986977	-11.198696
12	4.8006506	-1.599349	-11.199348
13	4.8003254	-1.5996745	-11.199675
14	4.800163	-1.5998373	-11.199837
15	4.8000813	-1.5999185	-11.199919

3-State Example: Values $\gamma = 0.9$

Iteration	SUN	WIND	HAIL
0	0	0	0
1	4	0	-8
2	5.8	-1.8	-11.6
3	5.8	-2.6100001	-14.030001
4	5.4355	-3.7035	-15.488001
5	4.7794	-4.5236254	-16.636175
6	4.1150985	-5.335549	-17.521912
7	3.4507973	-6.0330653	-18.285858
8	2.8379793	-6.6757774	-18.943516
9	2.272991	-7.247492	-19.528683
...	...	...	...
50	-2.8152928	-12.345073	-24.633476
51	-2.8221645	-12.351946	-24.640347
52	-2.8283496	-12.3581295	-24.646532
...	...	...	...
86	-2.882461	-12.412242	-24.700644
87	-2.882616	-12.412397	-24.700798
88	-2.8827558	-12.412536	-24.70094

3-State Example: Values $\gamma = 0.2$

Iteration	SUN	WIND	HAIL
0	0	0	0
1	4	0	-8
2	4.4	-0.4	-8.8
3	4.4	-0.44000003	-8.92
4	4.396	-0.452	-8.936
5	4.3944	-0.454	-8.9388
6	4.39404	-0.45443997	-8.93928
7	4.39396	-0.45452395	-8.939372
8	4.393944	-0.4545412	-8.939389
9	4.3939404	-0.45454454	-8.939393
10	4.3939395	-0.45454526	-8.939394
11	4.3939395	-0.45454547	-8.939394
12	4.3939395	-0.45454547	-8.939394

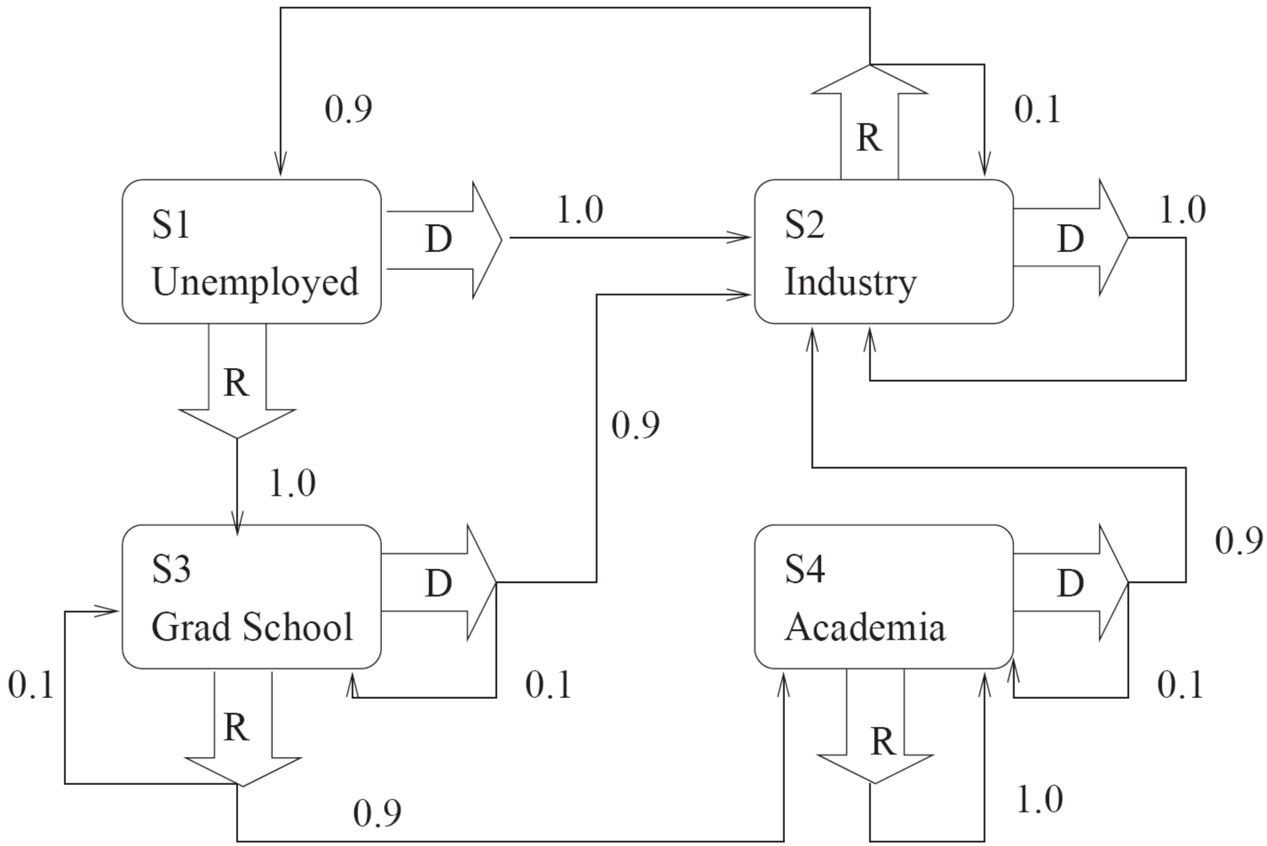
Markov Decision Processes

- Finite set of states, $s_1, \ldots, s_n$
- Finite set of actions, $a_1, \ldots, a_m$
- Probabilistic state,action transitions:

$$p_{ij}^k = \text{prob}\left(\text{next} = s_j \,\middle|\, \text{current} = s_i \text{ and take action } a_k\right)$$

- Markov assumption: State transition function only dependent on current state, not on the "history" of how the state was reached.
- Reward for each state, $r_1, \ldots, r_n$
- Process:
  - Start in state s_i
  - Receive immediate reward r_i
  - Choose action $a_k \in A$
  - Change to state s_j with probability p_{ij}^k .
  - Discount future rewards

Nondeterministic Example



Reward and discount factor to be decided.
Note the need to have a finite set of states and actions.
Note the need to have all transition probabilties.

Solving an MDP

- Find an action to apply to each state.
- A policy is a mapping from states to actions.
- Optimal policy - for every state, there is no other action that gets a higher sum of discounted future rewards.
- For every MDP there exists an optimal policy.
- Solving an MDP is finding an optimal policy.
- A specific policy converts an MDP into a plain Markov system with rewards.

Value Iteration

- $V^*(s_i)$ - expected discounted future rewards, if we start from state $s_{i,}$ and we follow the optimal policy.
- Compute V^* with value iteration:
  - $V^k(s_i)$ = maximum possible future sum of rewards starting from state s_i for k steps.
- Bellman's Equation:

$$V^{n+1}(s_i) = \max_k \{r_i + \gamma \sum_{j=1}^{N} p_{ij}^k V^n(s_j)\}$$

- Dynamic programming

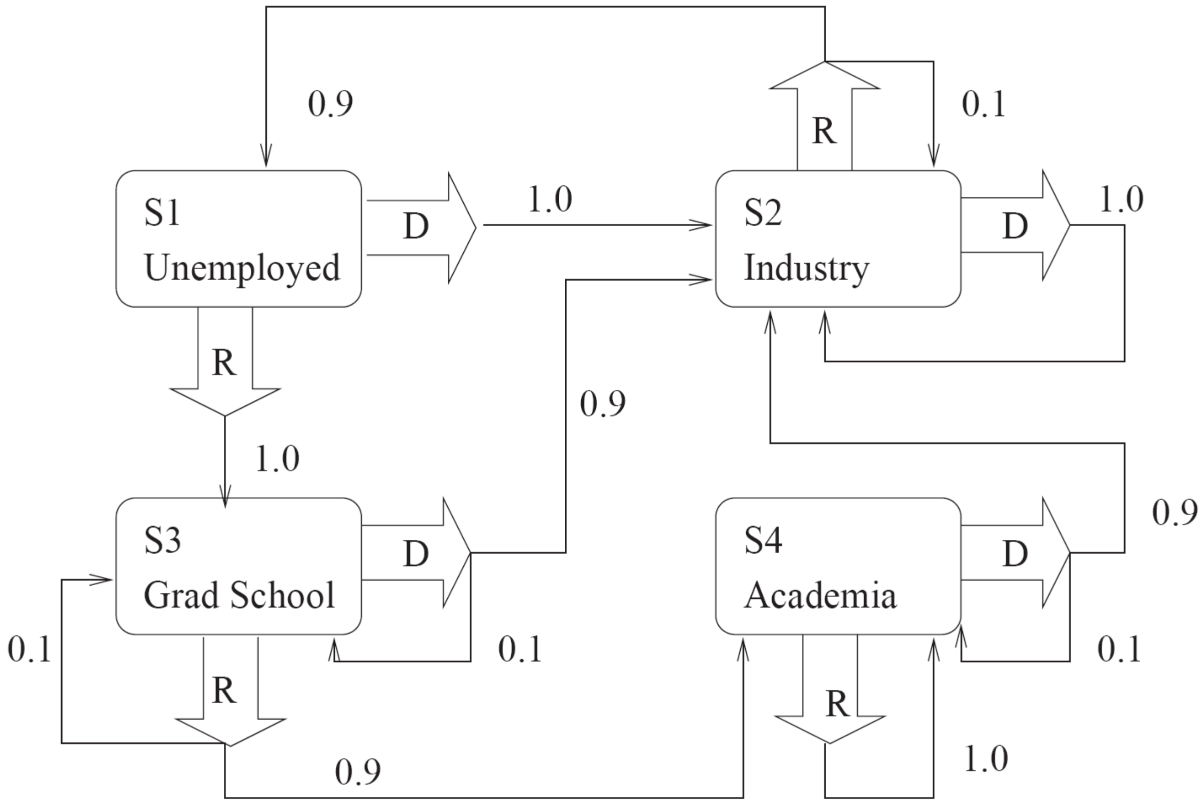
Policy Iteration

- Start with some policy $\pi_0(s_i)$.
- Such policy transforms the MDP into a plain Markov system with rewards.
- Compute the values of the states according to the current policy.
- Update policy:

$$\pi_{k+1}\left(s_i\right) = \arg\max_a \{r_i + \gamma \sum_j p_{ij}^a V^{\pi_k}\left(s_j\right)\}$$

- Keep computing
- Stop when $\pi_{k+1} = \pi_k$.

Nondeterministic Example



REWARD

S1	S2	S3	S4
0	100	0	10

Nondeterministic Example

$\pi^*(s)$ = D, for any s= S1, S2, S3, and S4, $\gamma = 0.9$.

```
V*(S2) = r(S2,D) + 0.9 (1.0 V*(S2))
V*(S2) = 100 + 0.9 V*(S2)
V*(S2) = 1000.

V*(S1) = r(S1,D) + 0.9 (1.0 V*(S2))
V*(S1) = 0 + 0.9 x 1000
V*(S1) = 900.

V*(S3) = r(S3,D) + 0.9 (0.9 V*(S2) + 0.1 V*(S3))
V*(S3) = 0 + 0.9 (0.9 x 1000 + 0.1 V*(S3))
V*(S3) = 81000/91.

V*(S4) = r(S4,D) + 0.9 (0.9 V*(S2) + 0.1 V*(S4))
V*(S4) = 40 + 0.9 (0.9 x 1000 + 0.1 V*(S4))
V*(S4) = 85000/91.
```

Markov Models

	Passive	Controlled
Fully Observable	Markov Systems with Rewards	MDP
Hidden State	HMM	POMDP
Time Dependent	Semi-Markov	SMDP

Tradeoffs

- **MDPs**
  - \+ Tractable to solve
  - \+ Relatively easy to specify
  - – Assumes perfect knowledge of state
- **POMDPs**
  - \+ Treats all sources of uncertainty uniformly
  - \+ Allows for taking actions that gain information
  - – Difficult to specify all the conditional probabilities
  - – *Hugely* intractable to solve optimally
- **SMDPs**
  - \+ General distributions for action durations
  - – Few good solution algorithms

Learning

- Learning from experience
- Supervised learning
  - Labeled examples
- Reward/reinforcement
  - Something good/bad (positive/negative reward) happens
  - An agent gets reward as part of the “input” percept, but it is “programmed” to understand it as reward.
  - Reinforcement extensively studied by animal psychologists.

Reinforcement Learning

- The problem of getting an agent to act in the world so as to maximize its rewards.
- Teaching a dog a new trick:
  - you cannot tell it what to do,
  - but you can reward/punish it if it does the right/wrong thing.
  - Learning: to figure out what it did that made it get the reward/punishment: the **credit assignment** problem.
- RL: a similar method to train computers to do many tasks.

Reinforcement Learning Task

- Assume the world is a Markov Decision Process
  - States and actions known
  - Transitions and rewards unknown
  - Full observability

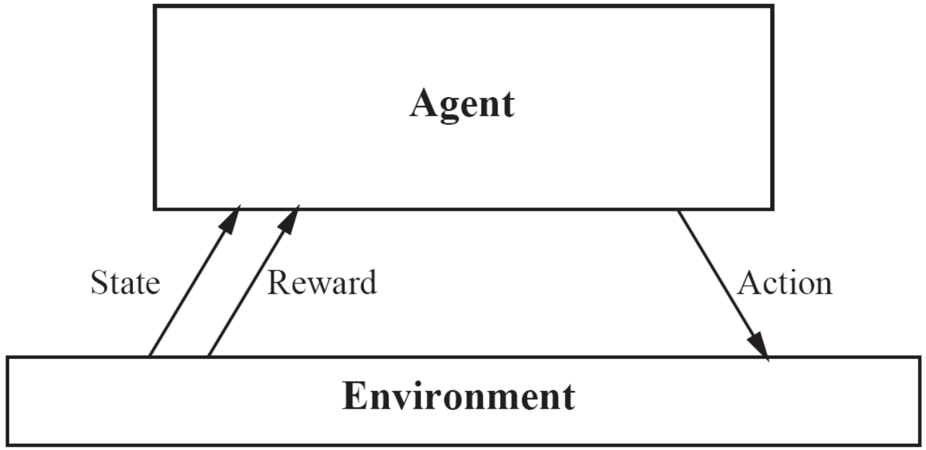
- Objective
  - **Learn action policy** $\pi : S \to A$
  - **Maximize expected reward**

$$E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots]$$

from any starting state in S.

  - $0 \leq \gamma < 1$, discount factor for future rewards

Reinforcement Learning Problem



$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \ldots$$

Agent sees the state, selects and action, and gets reward

Goal: Learn to choose actions that maximize

$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$, where $0 \leq \gamma < 1$

Online Learning Approaches

- Capabilities
  - Execute actions in world
  - Observe state of world
- Two Learning Approaches
  - Model-based
  - Model-free

Model-Based Reinforcement Learning

- Approach
  - Learn the MDP
  - Solve the MDP to determine optimal policy

- Appropriate when model is unknown, but small enough to solve feasibly

Learning the MDP

- Estimate the rewards and transition distributions
  - Try every action some number of times
  - Keep counts (frequentist approach)
    - $R(s,a) = R^a_s/N^a_s$
    - $T(s',a,s) = N^a_{s,s'}/N^a_s$
  - Solve using value or policy iteration
- Iterative Learning and Action
  - Maintain statistics incrementally
  - Solve the model periodically

Model-Free Reinforcement Learning

- Learn policy mapping ***directly***
- Appropriate when model is too large to store, solve, or learn
  - Do not need to try every state/action in order to get good policy
  - Converges to optimal policy

Value Function

- For each possible policy π, define an *evaluation function over states*

$$V^{\pi}(s) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + \ldots$$
$$\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

where r_t, r_{t+1},... are generated by following policy π starting at state s

$\boldsymbol{\pi^* \equiv \mathrm{argmax}_{\pi} V^{\pi}(s), (\forall s)}$

- Learning task: Learn OPTIMAL policy

Learn Value Function

- Learn the evaluation function $V^{\pi *}$ (i.e. V^*)
- Select the optimal action from any state s, i.e., have an optimal policy, by using V^* with one step lookahead:

$$\pi^*(s) = \arg\max_a \left[r(s,a) + \gamma V^*(\delta(s,a)) \right]$$

- But reward and transition functions are unknown

Q Function

- Define new function very similar to V^*

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a))$$

Learn Q function – Q-learning

- If agent learns Q, it can choose optimal action even without knowing δ or r

$$\pi^*(s) = \underset{a}{\arg\max}\left[r(s,a) + \gamma V^*(\delta(s,a))\right]$$

$$\pi^*(s) = \underset{a}{\arg\max}\, Q(s,a)$$

Q-Learning

Q and V^*:

$$V^*(s) = \max_{a'} Q(s, a')$$

We can write Q recursively:

$$\begin{aligned} Q(s_t, a_t) &= r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t)) \\ &= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') \end{aligned}$$

Q-learning actively generates examples.
It "processes" examples by updating its Q values.
While learning, Q values are approximations.

Training Rule to Learn Q (Deterministic Example)

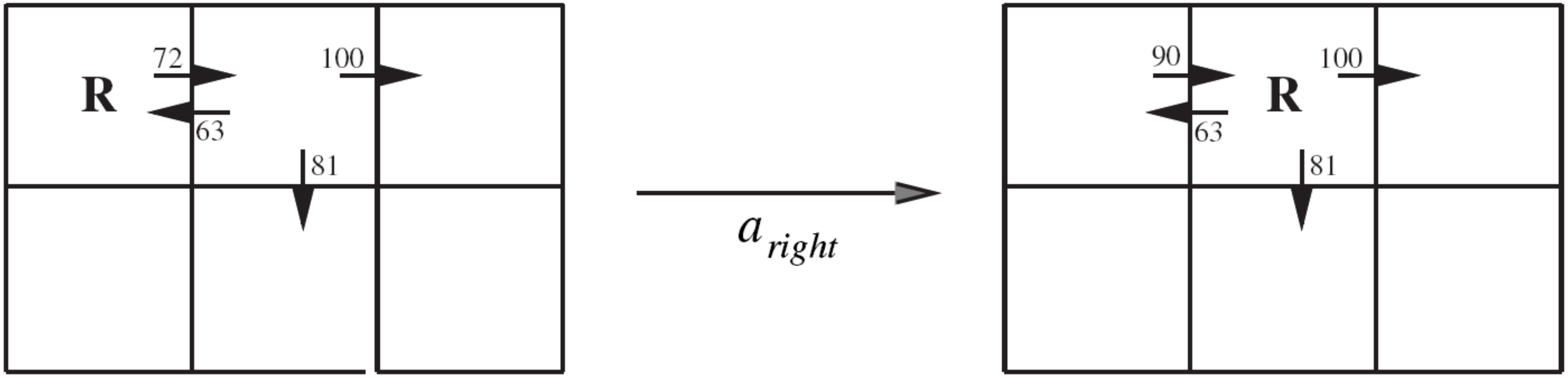
Let $\hat{Q}$ denote current approximation to Q.

Then Q-learning uses the following **training rule**:

$$\boxed{\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')}$$

where s' is the state resulting from applying action a in state s, and r is the reward that is returned.

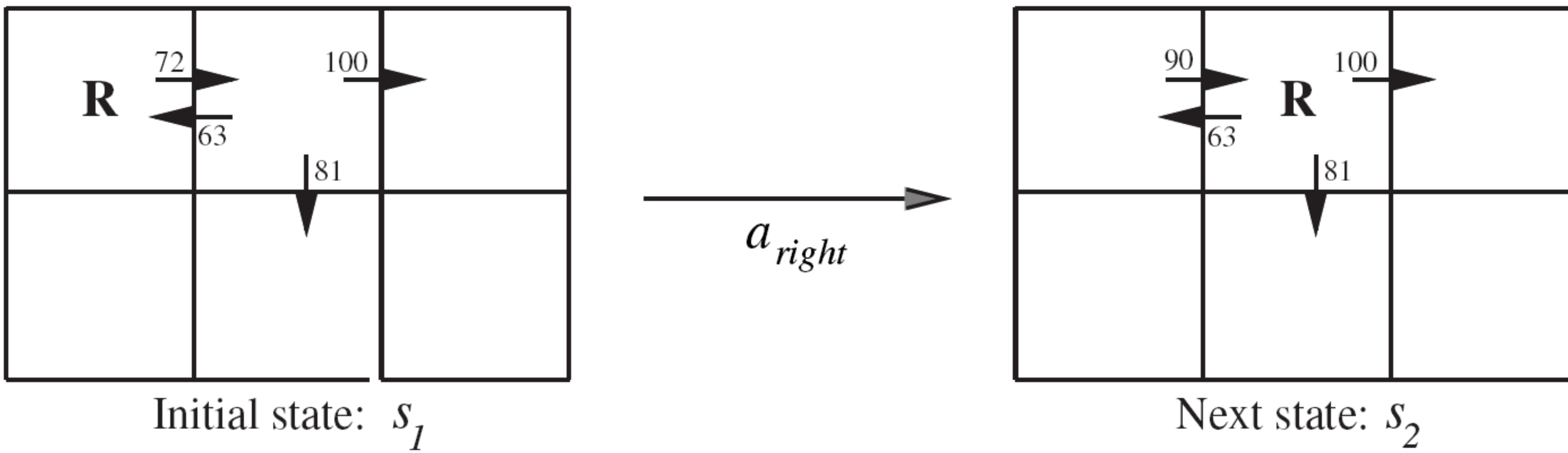
Deterministic Case – Example



Initial state: s_1

Next state: s_2

Deterministic Case – Example

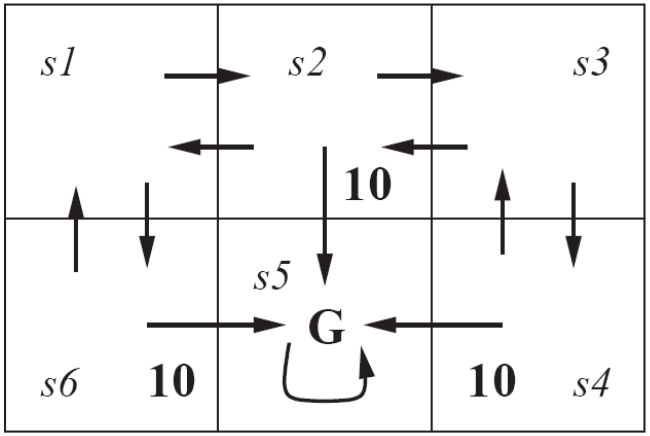


$$\begin{aligned}\hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\ &\leftarrow 0 + 0.9 \max\{63, 81, 100\} \\ &\leftarrow 90\end{aligned}$$

Q Learning Iterations

Start at top left corner with fixed policy – clockwise

Initially $Q(s,a) = 0$; $\gamma = 0.8$



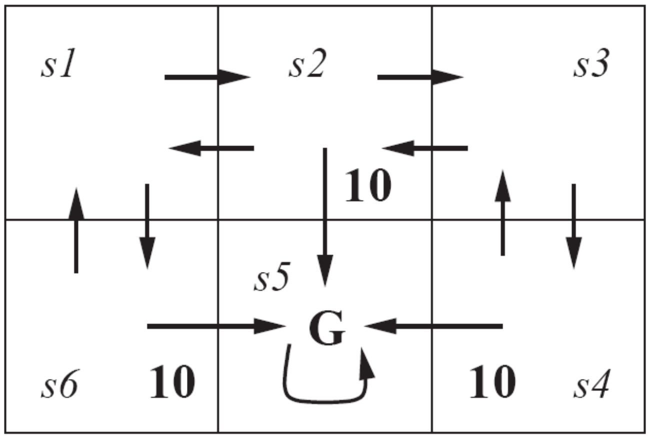
$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$$

Q (s1, E) Q (s2, E) Q (s3, S) Q(s4, W)

Q Learning Iterations

Starts at top left corner with fixed policy – clockwise

Initially $Q(s,a) = 0$; $\gamma = 0.8$



$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$$

Q(s1,E)	Q(s2,E)	Q(s3,S)	Q(s4,W)
0	0	0	r + γ max{Q(s5,loop)}= 10 + 0.8 . 0 = 10
0	0	r + γ max{Q(s4,W),Q(s4,N)}= 0 + 0.8 max{10,0}= 8	10
0	r + γ max{Q(s3,W),Q(s3,S)}= 0 + 0.8 max{0,8}= 6.4	8	10

Nondeterministic Case

- Q learning in nondeterministic worlds
  - Redefine V, Q by taking expected values:

$$V^{\pi}(s) \equiv E\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots\right]$$

$$\equiv E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right]$$

$$Q(s,a) \equiv E\left[r(s,a) + \gamma V^*(\delta(s,a))\right]$$

Nondeterministic Case

- Q learning training rule:

$$\hat{Q}_n(s,a) \leftarrow (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_n\left[r + \gamma \max_{a'} \hat{Q}_{n-1}(s',a')\right],$$

where $\alpha_n = \frac{1}{1+visits_n(s,a)}$, and $s' = \delta(s,a)$.

$\hat{Q}$ still converges to Q^* (Watkins and Dayan, 1992)

Exploration vs Exploitation

- Tension between learning optimal strategy and using what you know, so far, to maximize expected reward
  - Convergence theorem depends on visiting each state sufficient number of times
  - Typically use reinforcement learning while performing tasks

Exploration policy

- Wacky approach: act randomly in hopes of eventually exploring entire environment
- Greedy approach: act to maximize utility using current estimate
- Balanced approach: act “more” wacky when agent has not much knowledge of environment and “more” greedy when the agent has acted in the environment longer
- One-armed bandit problems

Exploration Strategies

- ε-greedy
  - Exploit with probability 1-ε
  - Choose remaining actions uniformly
  - Adjust ε as learning continues

- Boltzman
  - Choose action with probability $$p = \frac{e^{Q(s,a)/t}}{\sum_{a'} e^{Q(s,a')/t}}$$
  - where t "cools" over time (simulated annealing)

All methods sensitive to parameter choices and changes

Policy Reuse

- Impact of change of reward function
  - Does not want to learn from scratch
- Transfer learning
  - Learn macros of the MPD – options
  - Value function transfer
  - Exploration bias
- Reuse complete policies

Episodes

- MDP with absorbing goal states
  - Transition probability from a goal state to the same goal state is 1 (therefore to any other state is 0)
- Episode:
  - Start in random state, end in absorbing state
- Reward per episode (K episodes, H steps each):

$$W = \frac{1}{K} \sum_{k=0}^{K} \sum_{h=0}^{H} \gamma^h r_{k,h} \quad (1)$$

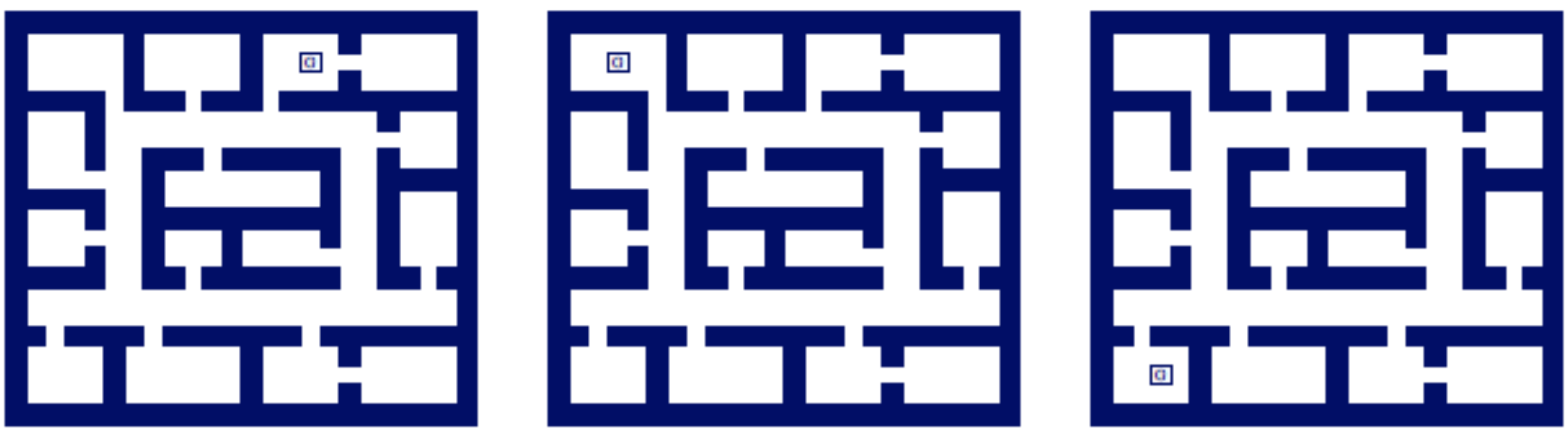
where γ $(0 \leq \gamma \leq 1)$ reduces the importance of future rewards, and $r_{k,h}$ defines the immediate reward obtained in the step h of the episode k, in a total of K episodes.

Domains and Tasks

A **domain** $\mathcal{D}$ is defined as a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T} >$, where $\mathcal{S}$ is the set of all possible states; $\mathcal{A}$ is the set of all possible actions; and $\mathcal{T}$ is a state transition function, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \Re$

A **task** Ω is defined as a tuple $< \mathcal{D}, \mathcal{R}_\Omega >$, where $\mathcal{D}$ is a domain; and $\mathcal{R}_\Omega$ is the reward function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$

An **action policy** Π_Ω to solve a task Ω is a function $\Pi_\Omega : \mathcal{S} \rightarrow \mathcal{A}$.



(a) Task Ω_1 (b) Task Ω_2 c) Task Ω_3

Policy Library and Reuse

- **Policy Reuse**:
  - ⋆ We need to solve the task Ω, i.e. learn Π_Ω
  - ⋆ We have previously solved the set of tasks $\{\Omega_1, \ldots, \Omega_n\}$ so we have a Policy Library composed of the n policies that solve them respectively, say $L = \{\Pi_1, \ldots, \Pi_n\}$
  - ⋆ How can we use the policy library, L, to learn the new policy, Π_Ω?

π-Reuse Exploration

Need to solve a task Ω, i.e. learn Π_{new}.

Have a Policy Library, say $L = \{\Pi_1, \ldots, \Pi_n\}$

Let's assume that there is a supervisor who, given Ω, tells us which is the most similar policy, say Π_{past}, to Π_{new}. Thus, we know that the policy to reuse is Π_{past}.

Integrate the past policy as a probabilistic bias in the exploration strategy of the new learning process

Define probabilities for exploiting the past policy, perform random exploration, or exploit the ongoing policy

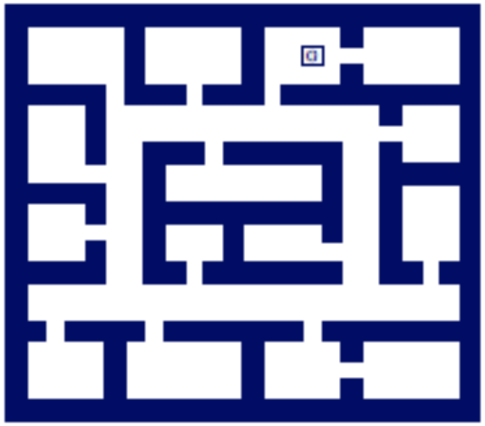
⋆ Select $a = \begin{cases} \Pi_{past}(s) & \text{w/prob. } \psi \\ \Pi_{new}(s)) & \text{w/prob. } (1-\psi)\epsilon \\ Random & \text{w/prob. } (1-\psi)(1-\epsilon) \end{cases}$

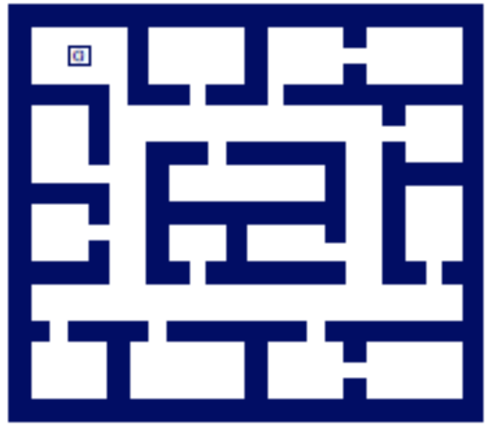
π-Reuse Policy Learning

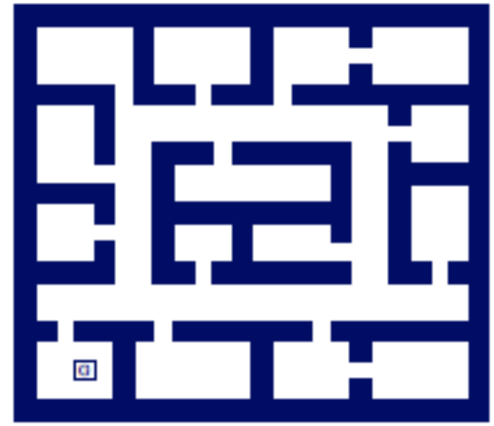
π-reuse $(\Pi_{past}, K, H, \psi, \upsilon, \gamma, \alpha)$.

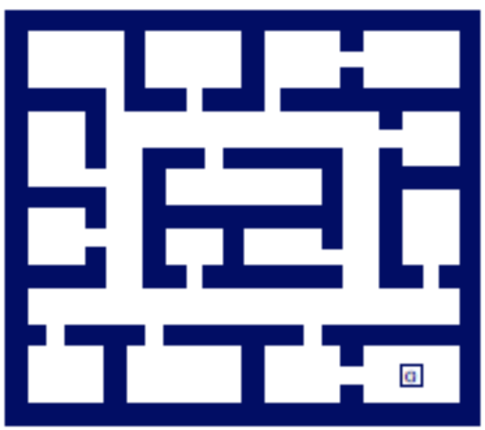
Initialize $Q^{\Pi_{new}}(s,a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$
For $k = 1$ to K
  Set the initial state, s, randomly.
  Set $\psi_1 \leftarrow \psi$
  for $h = 1$ to H
    With a probability of ψ_h, $a = \Pi_{past}(s)$
    With a probability of $1 - \psi_h$, $a = \epsilon\text{-greedy}(\Pi_{new}(s))$
    Receive current state s', and reward, $r_{k,h}$
    Update $Q^{\Pi_{new}}(s,a)$, and therefore, Π_{new}, using the Q-Learning update function:
      $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a')]$
    Set $\psi_{h+1} \leftarrow \psi_h \upsilon$
    Set $s \leftarrow s'$
$W = \frac{1}{K}\sum_{k=0}^{K}\sum_{h=0}^{H}\gamma^h r_{k,h}$
Return W, $Q^{\Pi_{new}}(s,a)$ and Π_{new}

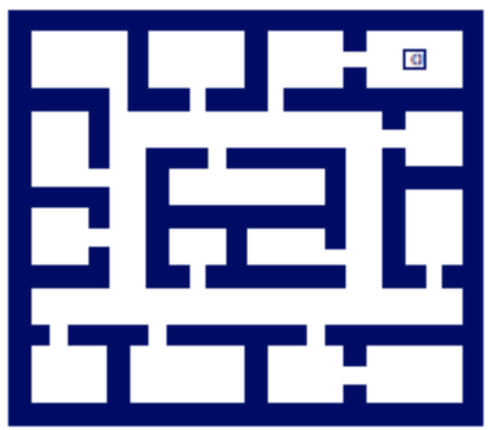
Experimental Results


(a) Task Ω_1


(b) Task Ω_2

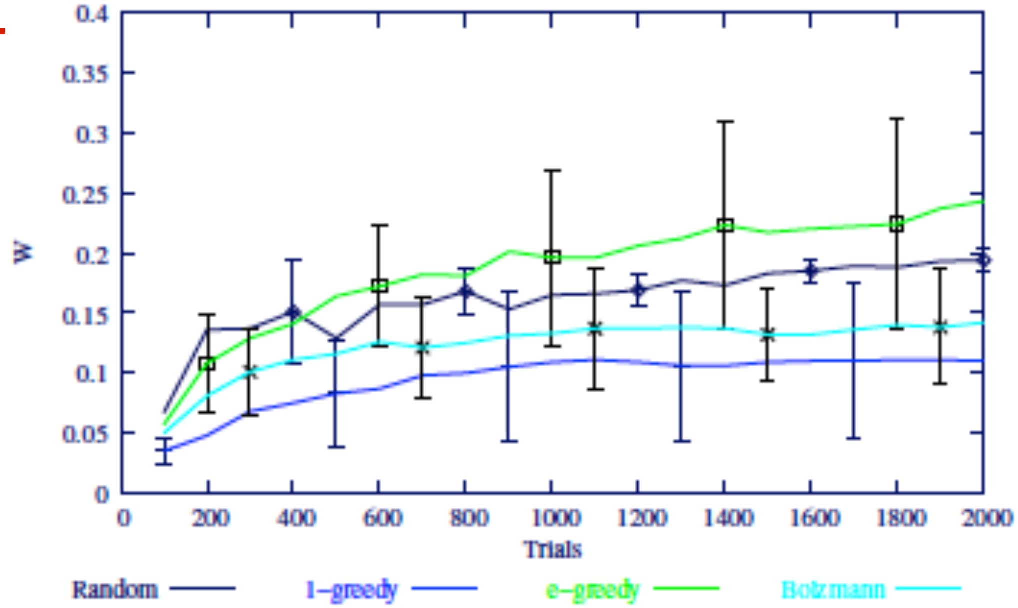

c) Task Ω_3


(d) Task Ω_4

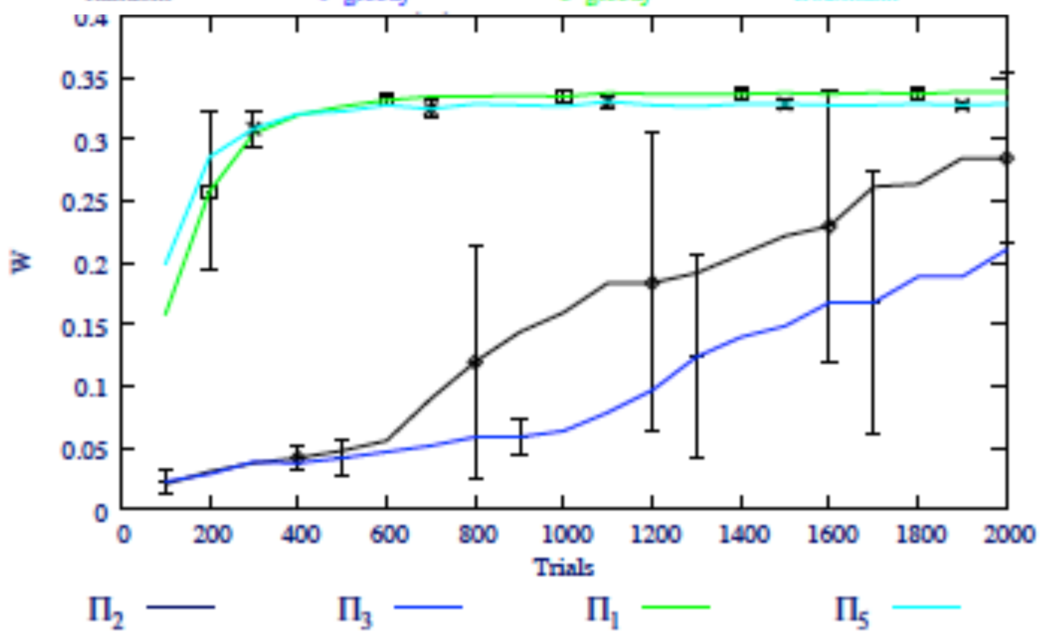

(e) Task Ω_5


(f) Task Ω

Results



W
Trials
Random
1-greedy
e-greedy
Boltzmann



W
Trials
Π_2
Π_3
Π_1
Π_5

Policy Reuse in Q-Learning

- Interestingly, the pi-reuse strategy also contributes a *similarity metric* between policies
  - The gain Wi obtained while executing the pi-reuse exploration strategy, reusing the past policy i.
- Wi is an estimation of how similar the policy i is to the new one!
- The set of Wi values for each of the policies in the library is unknown a priori, but it can be estimated on-line while the new policy is computed in the different episodes.

PRQ-Learning.

PRQ-Learning (Ω, L, K, H)

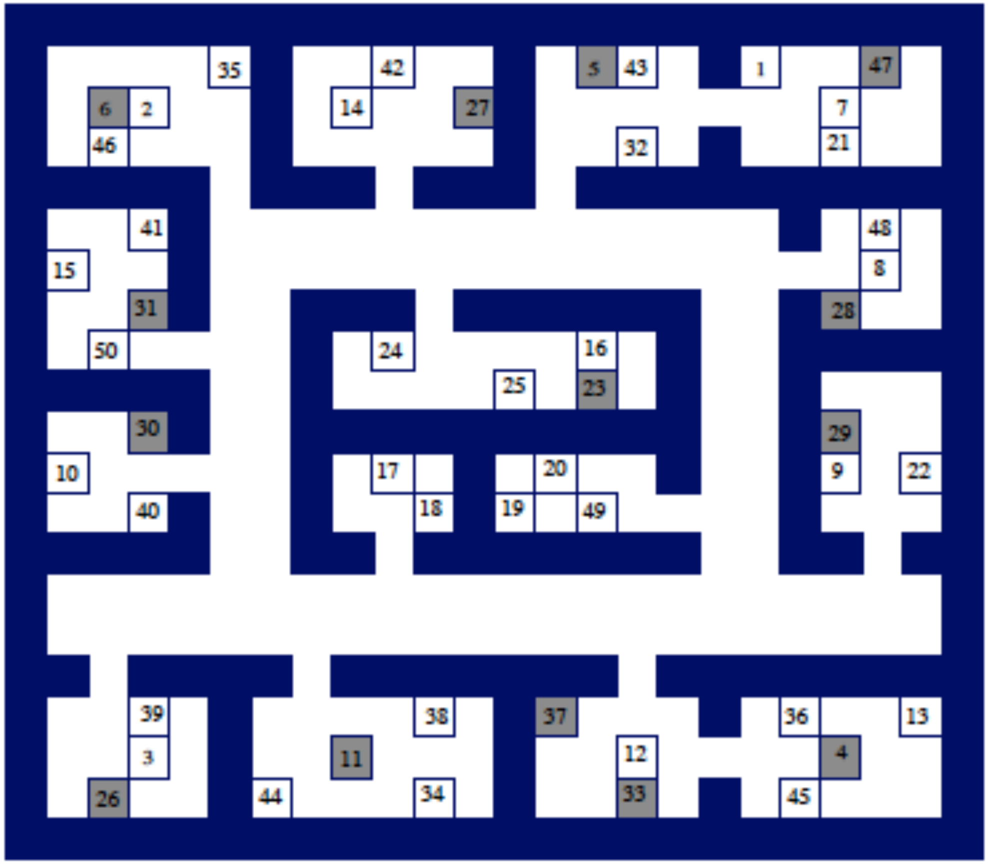
• Given:
(1) A new task Ω we want to solve.
(2) A Policy Library $L = \{\Pi_1, \ldots, \Pi_n\}$.
(3) A maximum number of episodes to execute, K.
(4) A maximum number of steps per episode, H.
• Initialize:
(1) $Q_\Omega(s, a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$.
(2) $W_\Omega = W_i = 0$, for $i = 1, \ldots, n$.
• For $k = 1$ to K do
 – Choose an action policy, Π_k, assigning to each policy the probability of being selected computed by the following equation:

$$P(\Pi_j) = \frac{e^{\tau W_j}}{\sum_{p=0}^{n} e^{\tau W_p}}$$

 where W_0 is set to W_Ω.
 – Execute the learning episode k.
  If $\Pi_k = \Pi_\Omega$, execute a Q-Learning episode following a fully greedy strategy.
  Otherwise, call π-reuse (Π_k, 1, H, ψ, υ).
  In any case, receive the reward obtained in that episode, say R, and the updated Q function, $Q_\Omega(s, a)$.
 – Recompute W_k using R.
• Return the policy derived from $Q_\Omega(s, a)$.

Learning to Use a Policy Library

- Similarity between policies can be learned
- Gain of using each policy
- Explore different policies
- Learn domain structure: “eigen” policies



Summary

- Reinforcement learning
  - Q-learning
- Policy Reuse

- Next class:
  - Other reinforcement learning algorithms
  - (There are many…)