Reinforcement Learning:
From Foundations to Advanced Topics

Prasad Tadepalli
Sridhar Mahadevan
Vivek Borkar

Outline

1. Markov Decision Processes (MDPs) (Tadepalli and Borkar)
   - Introduction to Reinforcement Learning (30 m)
   - Stochastic Approximation Theory (60 m)
2. Scaling Issues (Tadepalli) (60 m)
   - Function Approximation
   - Hierarchical Reinforcement Learning
   - Approximate Policy Iteration
3. Learning Representations (Mahadevan) (90 m)
   - Spectral Methods
   - Solving MDPs using Spectral Methods

Reinforcement Learning (RL)

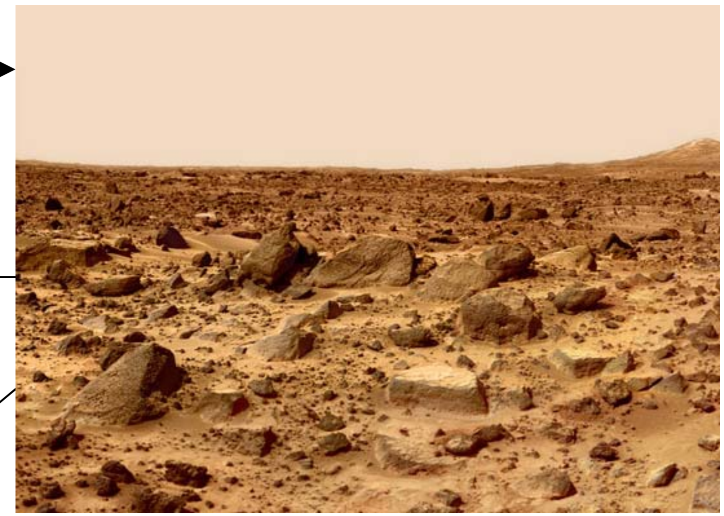
Agent

World

Actions

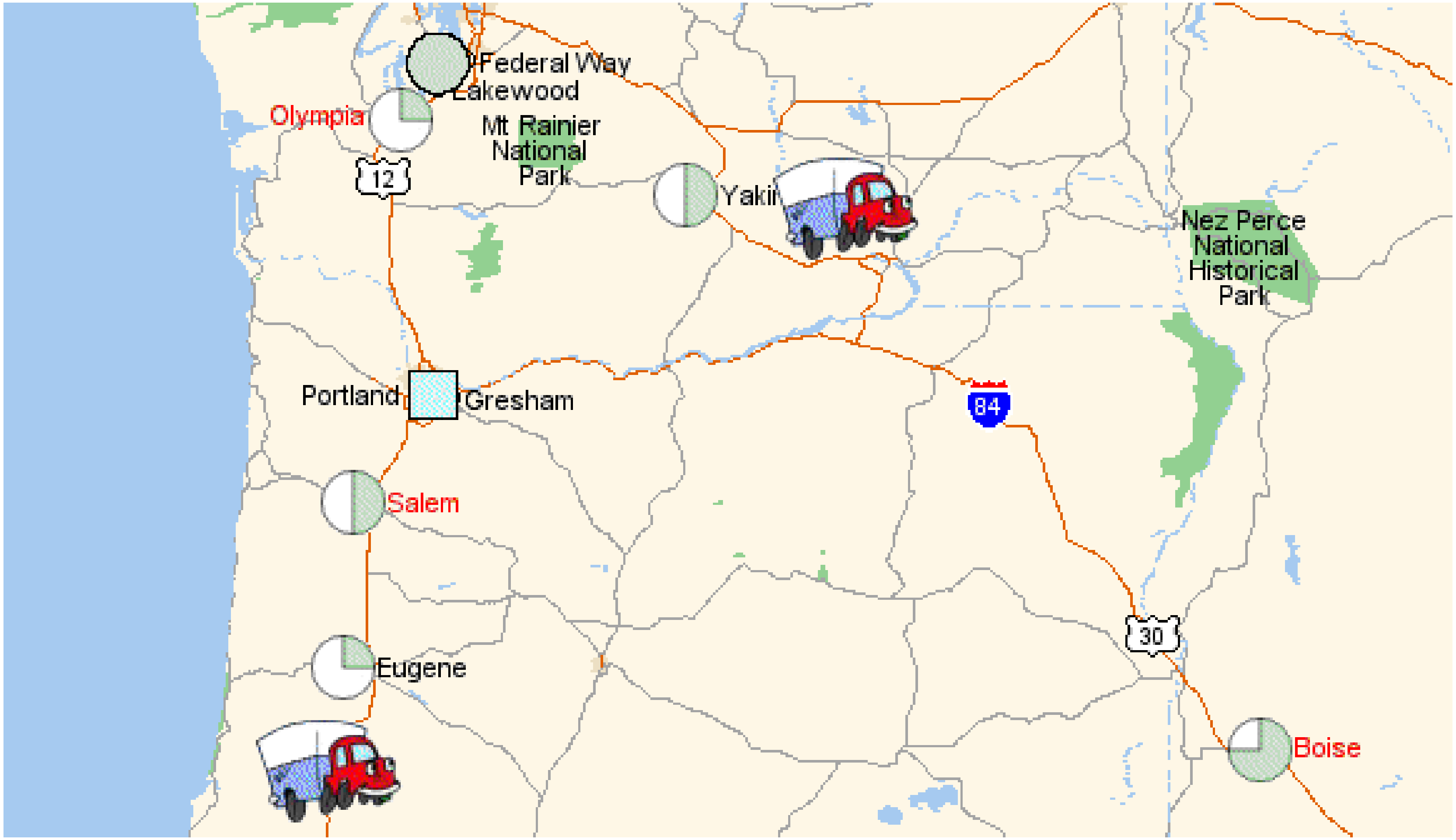
Percepts

Rewards



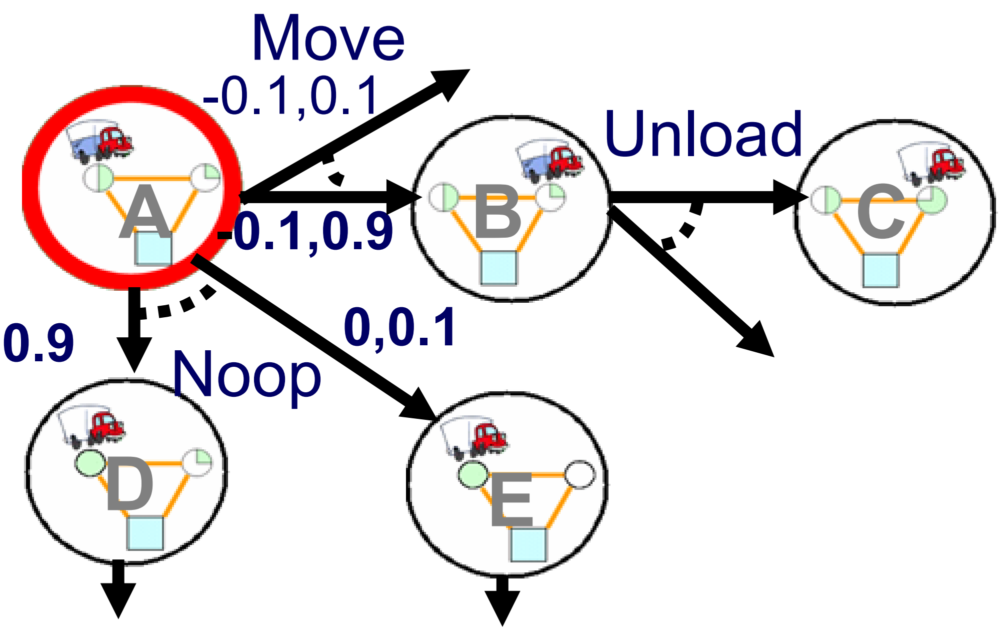
The goal is to act to optimize a performance measure, e.g., expected total reward received

Vehicle Routing & Product Delivery



Markov Decision Processes

- A *Markov Decision Process* (MDP) consists of a set of states *S*, actions *A*, Rewards *R(s,a),* and a stochastic state-transition function *P(s'|s,a)*
- A *policy* is a mapping from States to Actions.
- The goal is to find a policy π* that maximizes the total expected reward until some termination state – the "*optimal policy*."

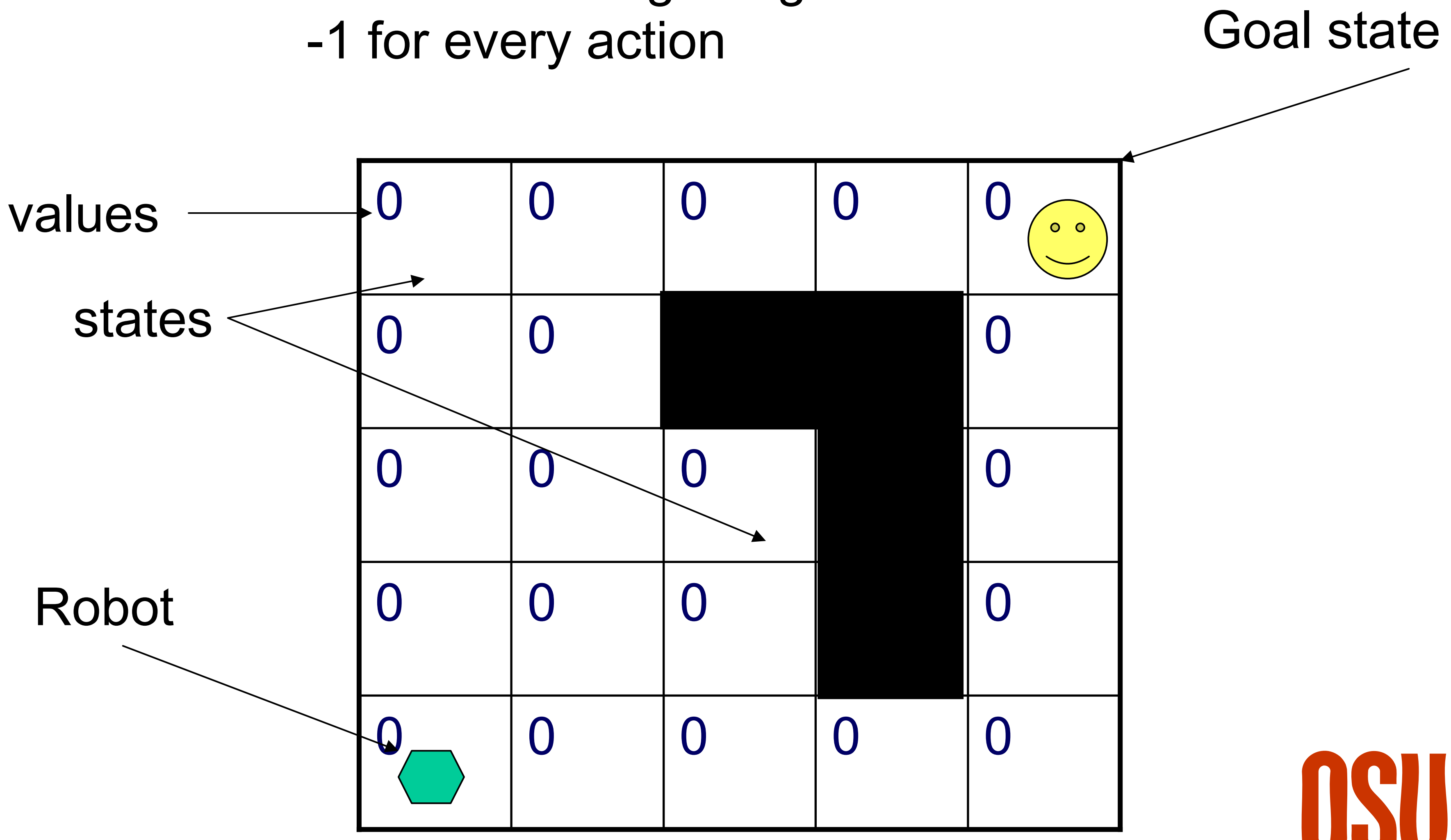


MDP Theory

- For a fixed policy π, there is a real-valued function V^π that satisfies $V^\pi(s) = R(s,\pi(s))+E(V^\pi(s'))$ (BellmanEqn) $V^\pi(s)$ represents the expected total reward of π from s
- Theorem: $\exists$ an optimal policy π^* : $\forall s\ \forall\pi\ V^{\pi^*}(s) \geq V^\pi(s)$
- An optimal policy π^* satisfies the Bellman Equation:
  $V^{\pi*}(s) = Max_a \{R(s,a) + \sum_{s'} P(s'|s,a) (V^{\pi*}(s'))\}$
- Value iteration: Solve the equations for V by iteratively replacing the l.h.s with the r.h.s for all states
- Temporal Difference Learning: Update V for states encountered along a trajectory
- If every state is updated infinitely often, V converges to $V^{\pi*}$
- Assumption: All policies terminate, i.e., reach an absorbing state.

A Grid Example

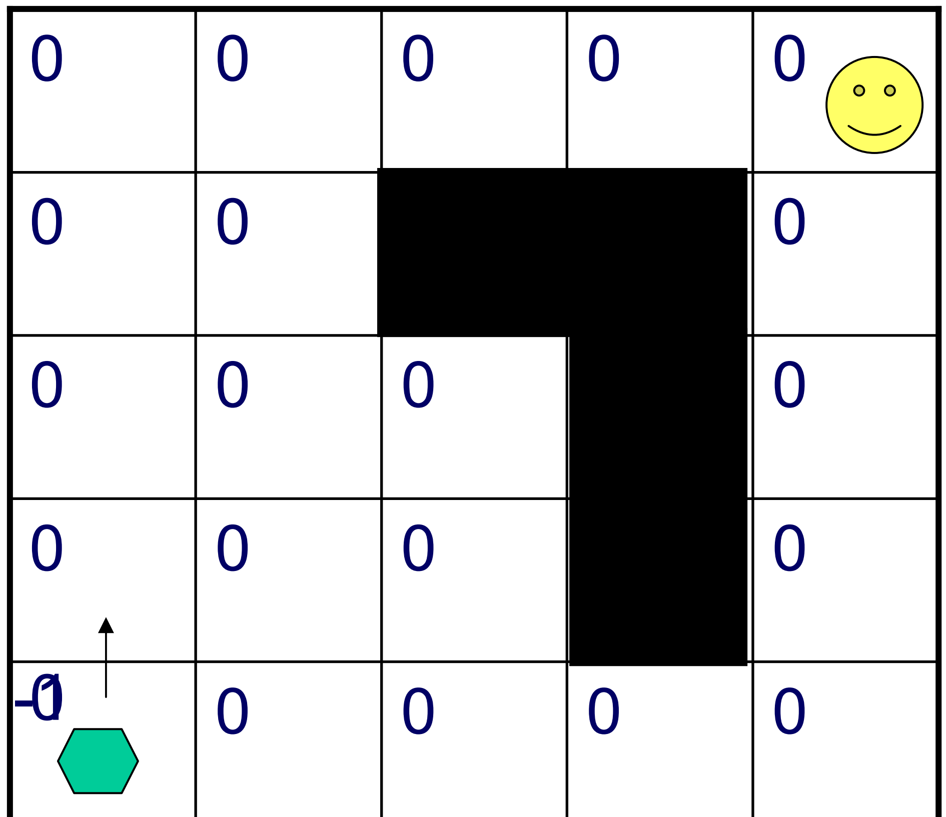
Rewards: 10 for reaching the goal state
-1 for every action



A Grid Example

Choose an action $a = argmax_a\ (R(s,a)+V(s'))$

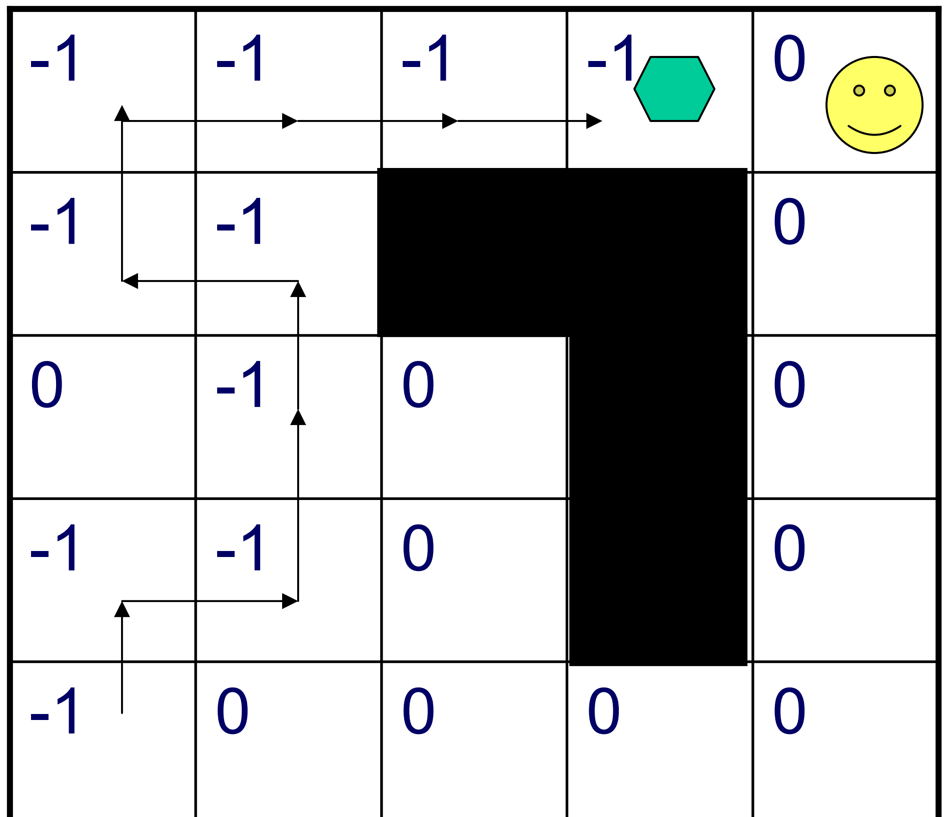
Update $V(s) \leftarrow Max_a\ R(s,a)+V(s')$



A Grid Example

Choose an action $a = argmax_a\ R(s,a)+V(s'))$

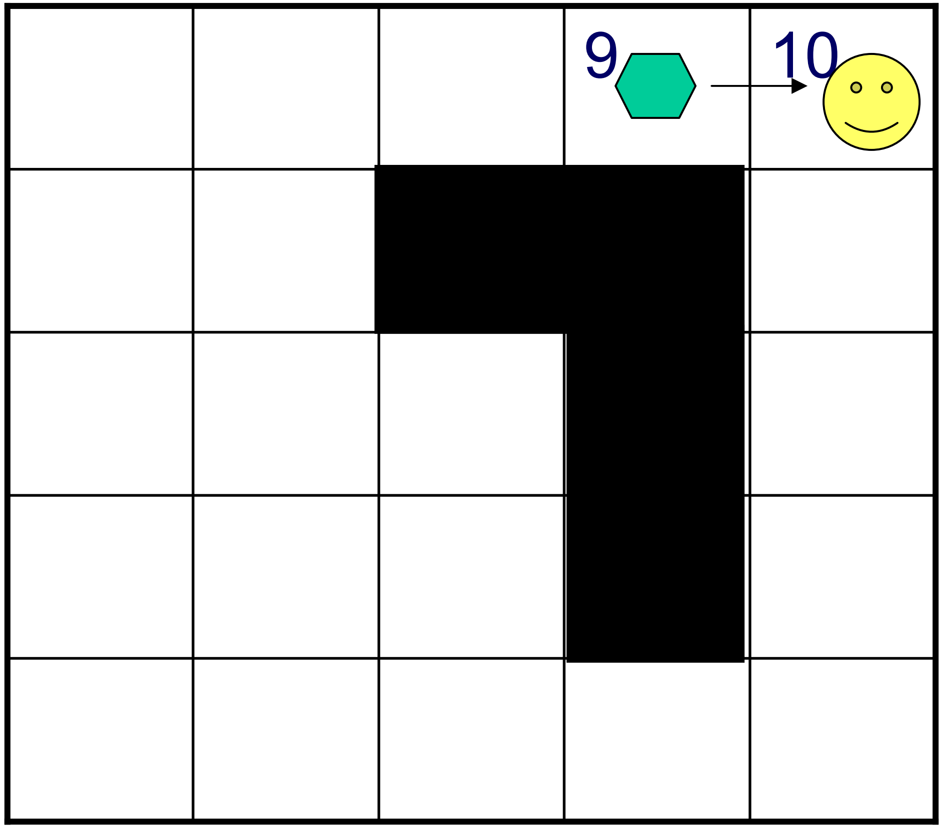
Update $V(s) \leftarrow Max_a\ R(s,a)+V(s')$



A Grid Example

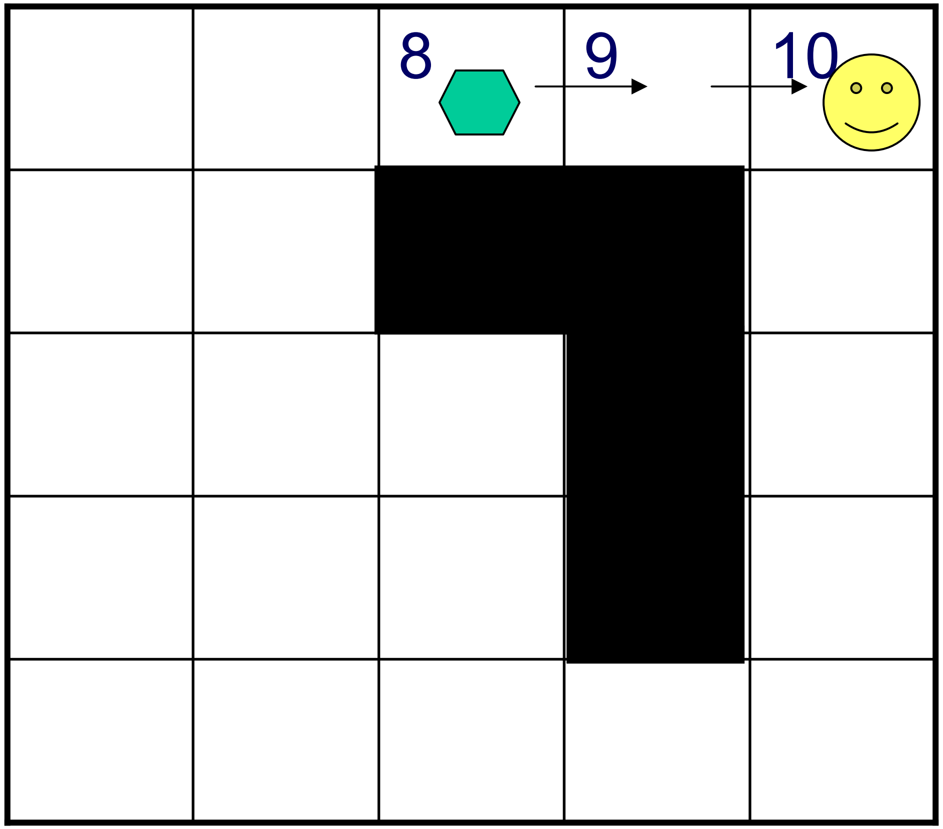
Rewards: 10 for reaching the goal state

-1 for every action



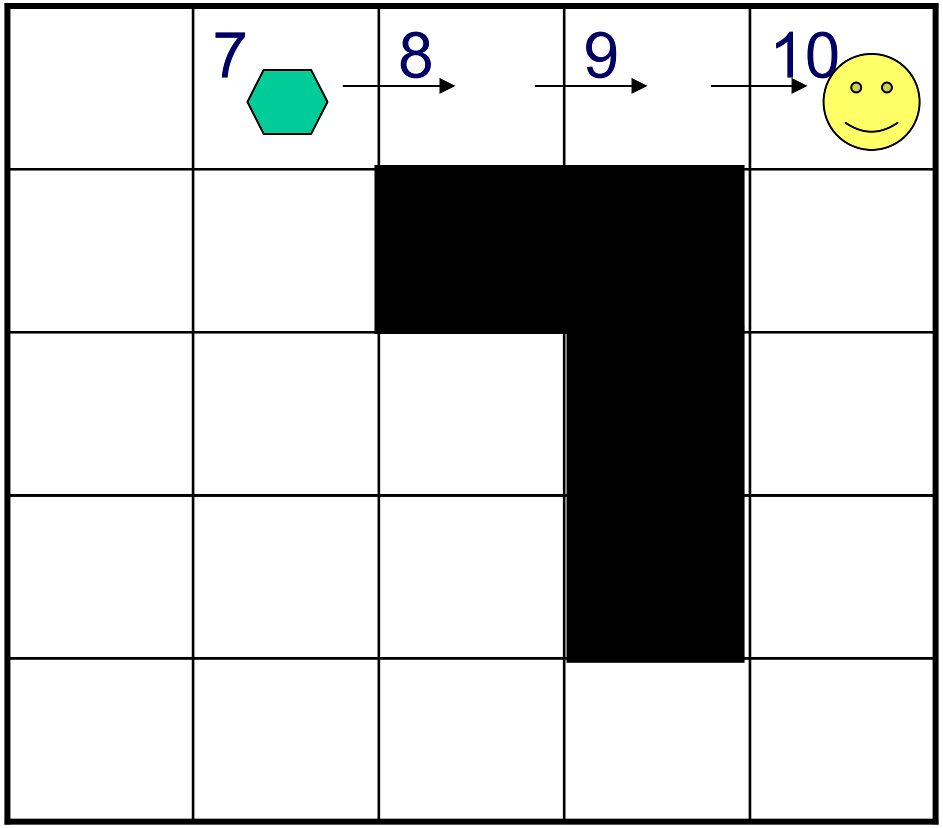
A Grid Example

Update: $V(s) \leftarrow Max_a\ R(s,a)+V(s')$



A Grid Example

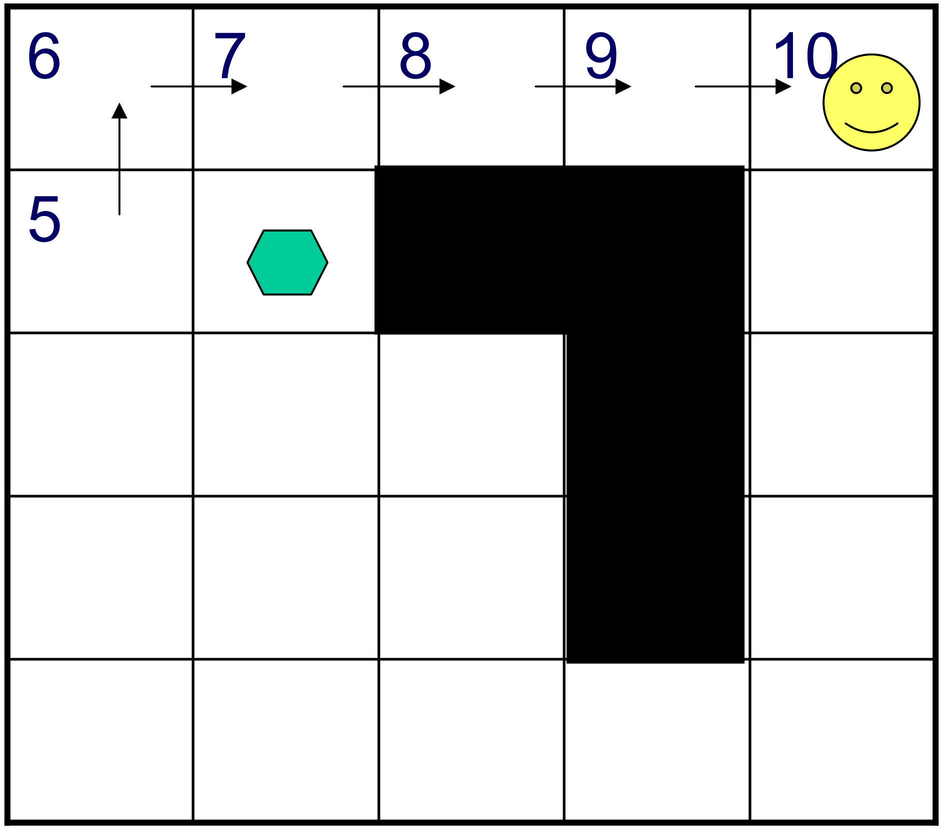
Update: $V(s) \leftarrow Max_a\, R(s,a)+V(s')$



A Grid Example

Choose an action $a = argmax_a\ R(s,a)+V(s')$

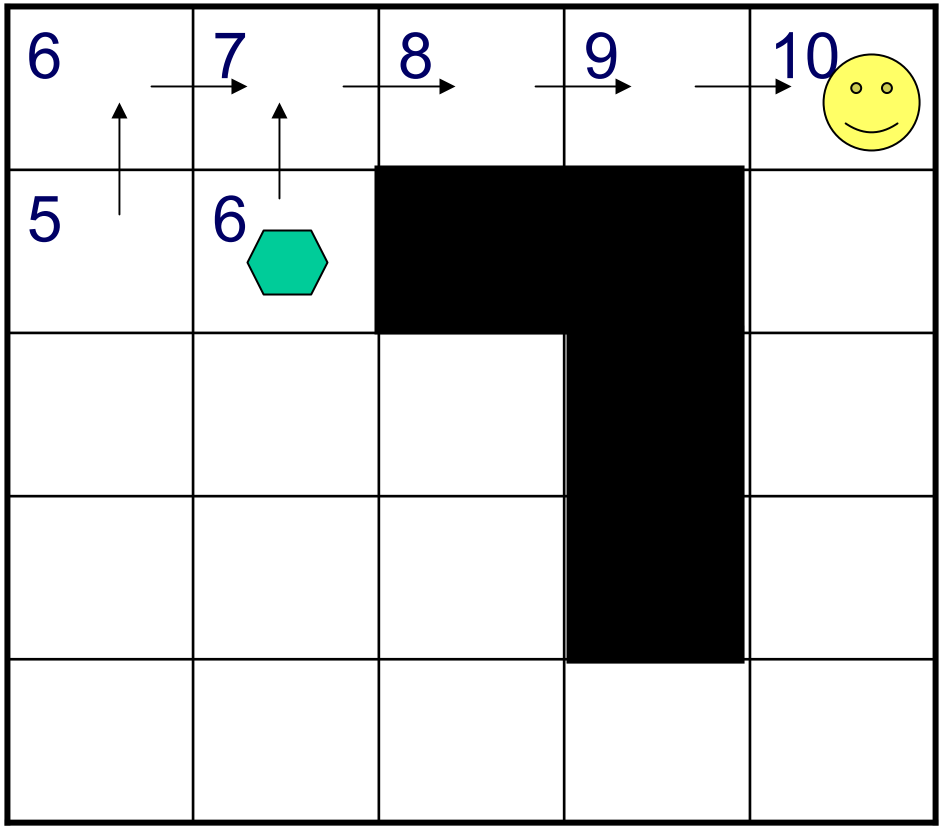
Update $V(s) \leftarrow Max_a\ R(s,a)+V(s')$



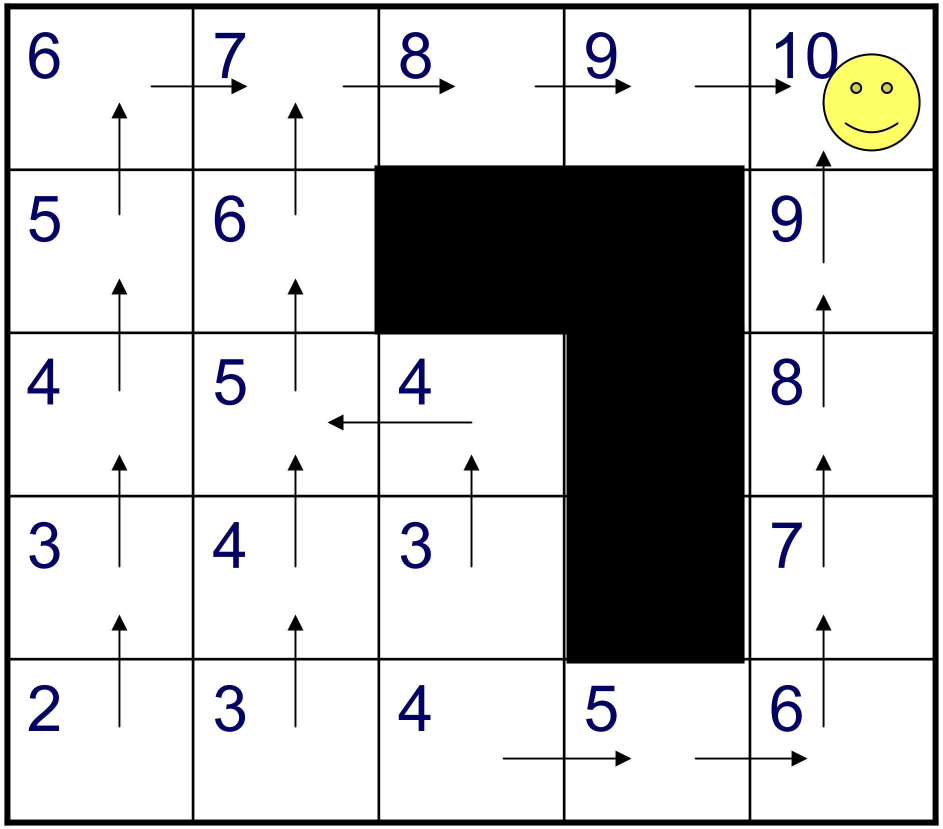
A Grid Example

Choose an action $a = argmax_a\ R(s,a)+V(s')$

Update $V(s) \leftarrow Max_a\ R(s,a)+V(s')$



A Grid Example



The values converge after a few trials if every action is exercised infinitely often in every state

Stochastic Case

- Taking action *a* from the same state *s*, results in possibly different next states *s'* with probability $P(s'|s,a)$
- Choose $a = \text{argmax}_a [R(s,a) + \sum_{s'} P(s'|s,a)V(s')]$
- Update $V(s) \leftarrow Max_a R(s,a) + \sum_{s'} P(s'|s,a) V(s')$
- Converges to the optimal policy if every action is exercised in every state infinitely often
- **Problem:** To choose an action, one needs to know not only $V(.)$ but also the action models:
  - Immediate reward $R(.,.)$
  - State transition function $P(.|.,.)$
- Method is called "model-based."

Q-Learning

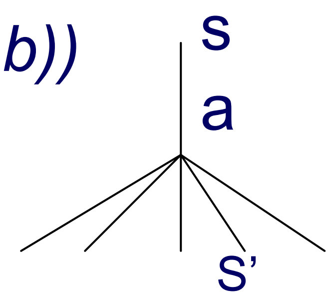
- Motivation: What if $R(s,a)$ and $P(s'|s,a)$ are unknown?
- An optimal policy π^* satisfies the Bellman Equation:

$$V^{\pi^*}(s) = Max_a \{R(s,a) + \sum_{s'} P(s'|s,a) V^{\pi^*}(s')\}$$
$$= Max_a\ Q(s,a),$$
$$\text{where } Q(s,a) \equiv R(s,a) + \sum_{s'} P(s'|s,a) V^{\pi^*}(s')$$
$$\equiv R(s,a) + \sum_{s'} P(s'|s,a)\ Max_b\ Q(s',b)$$

- $\pi^*(s) = Argmax_a Q(s,a)$
- If we know the Q-function, we know the optimal policy!
- But, we still need *P* and *R* to update *Q* – or do we?
- Use sample update instead of full model update!

Q-Learning

$$Q(s,a) = R(s,a) + \sum_{s'} P(s'|s,a)\,(Max_b\, Q(s',b))$$



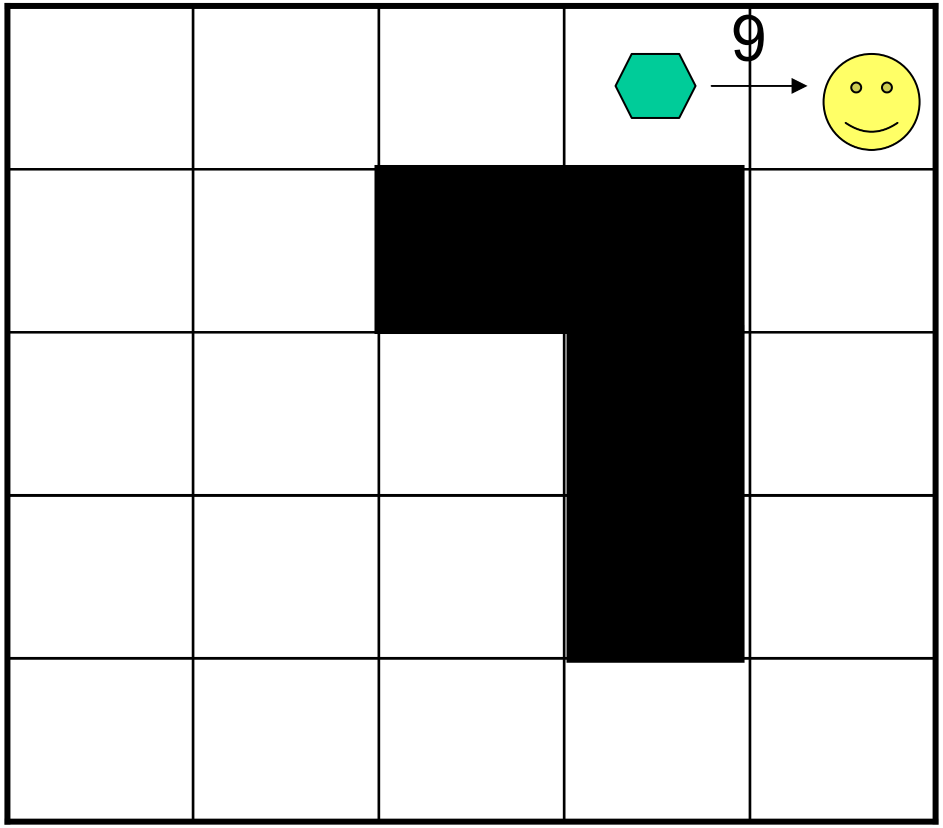
- Initialize *Q*-values arbitrarily
- When in state *s*, take some action *a*
  - Usually a greedy action $argmax_a\ Q(s,a)$
  - With some probabiity explore different actions
- Observe immediate reward *r* and next state *s'*
- *r* is a sample of *R(s,a); s'* is a sample of next state
- *Q(s,a)* is updated towards $r + Max_b\ Q(s',b)$ *(stochastic approximation or sample update instead of a full model update)*
  - $Q(s,a) \leftarrow (1-\alpha)\ Q(s,a) + \alpha\ (r + Max_b\ Q(s',b))$, where α is a learning rate
- If every *Q(s,a)* is updated infinitely often, the *Q*-values converge to their true values.

A Grid Example

Rewards: 10 for reaching the goal state

-1 for every action. α is set to 1 for simplicity.

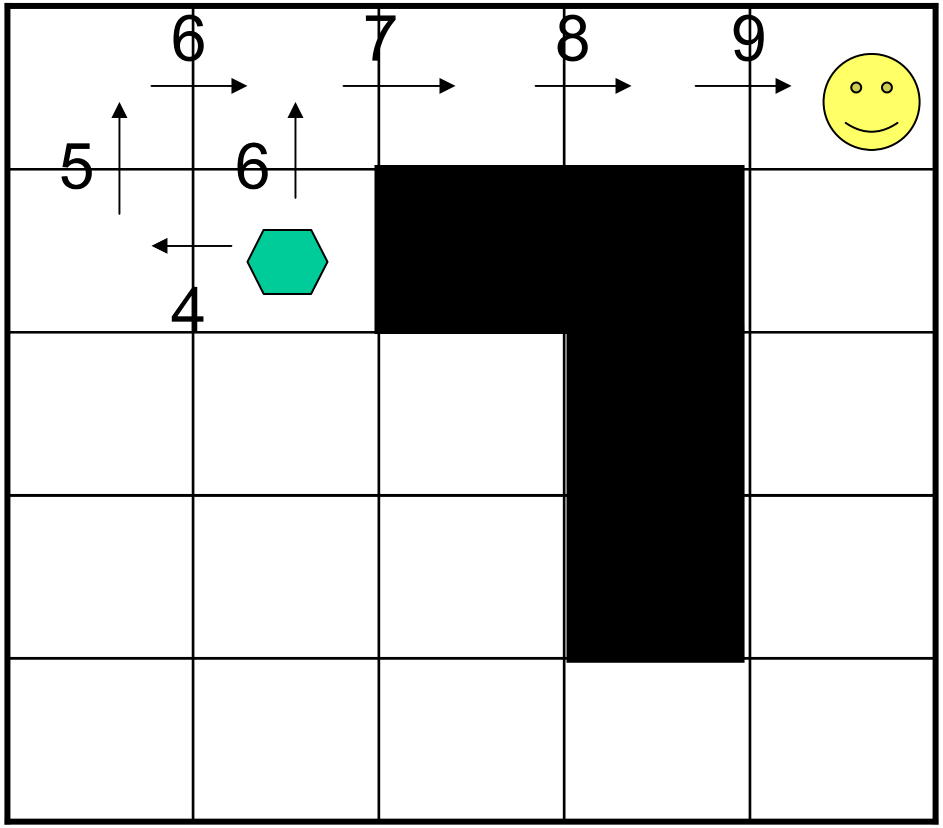
Update: $Q(s,a) = r + Max_b\,(Q(s',b))$



A Grid Example

Choose an action $a = argmax_a\ Q(s,a)$ reaching s'

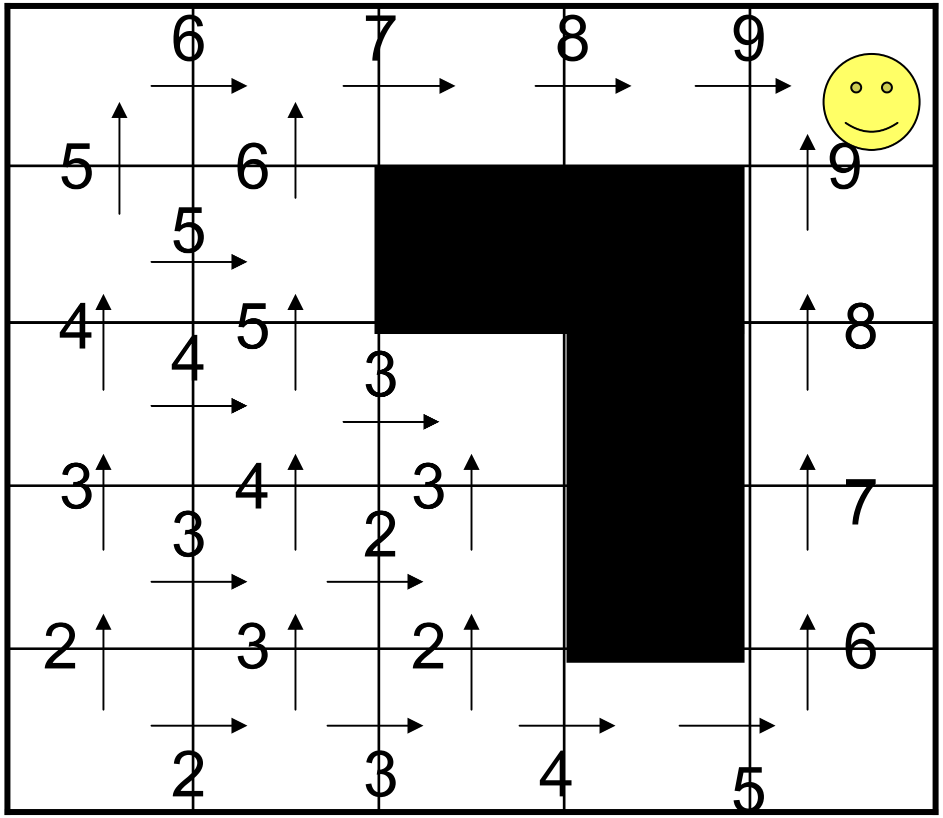
Update $Q(s,a) = r + Max_b\ Q(s',b)$



A Grid Example

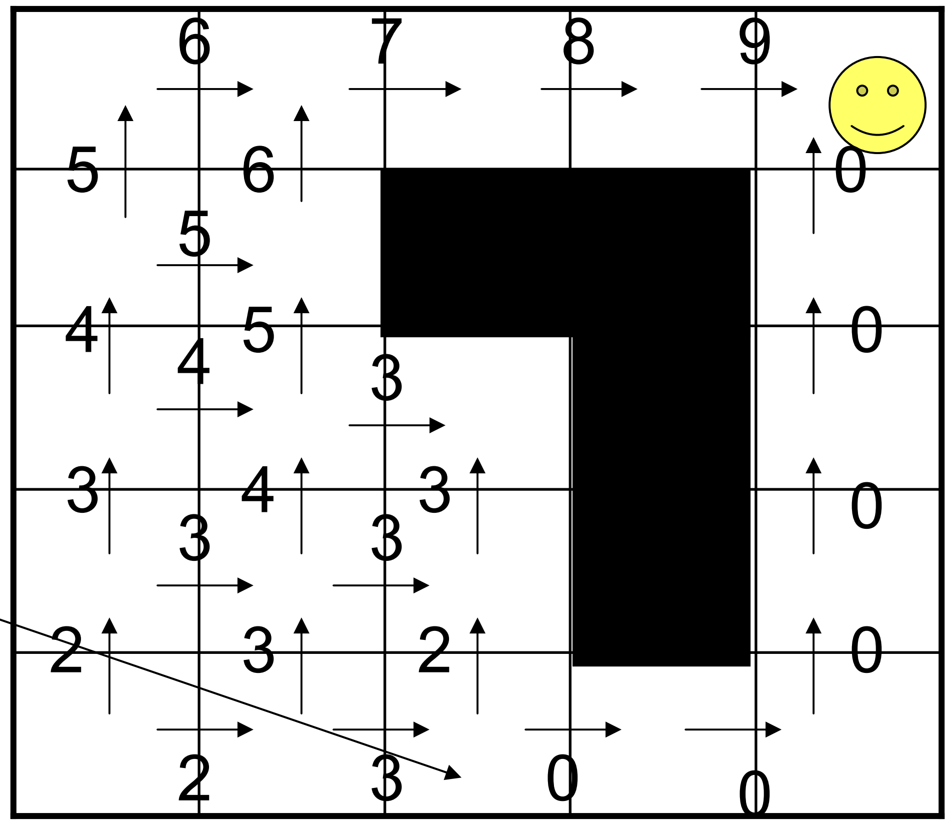
Choose an action $a = argmax_a\ Q(s,a)$ reaching s'

Update $Q(s,a) = r + Max_b\ Q(s',b)$



Exploration-Exploitation Dilemma

Exploitation: Take the best action according to the current value function (which may not have converged).
Exploration: Take the most informative action (which may not be very good).

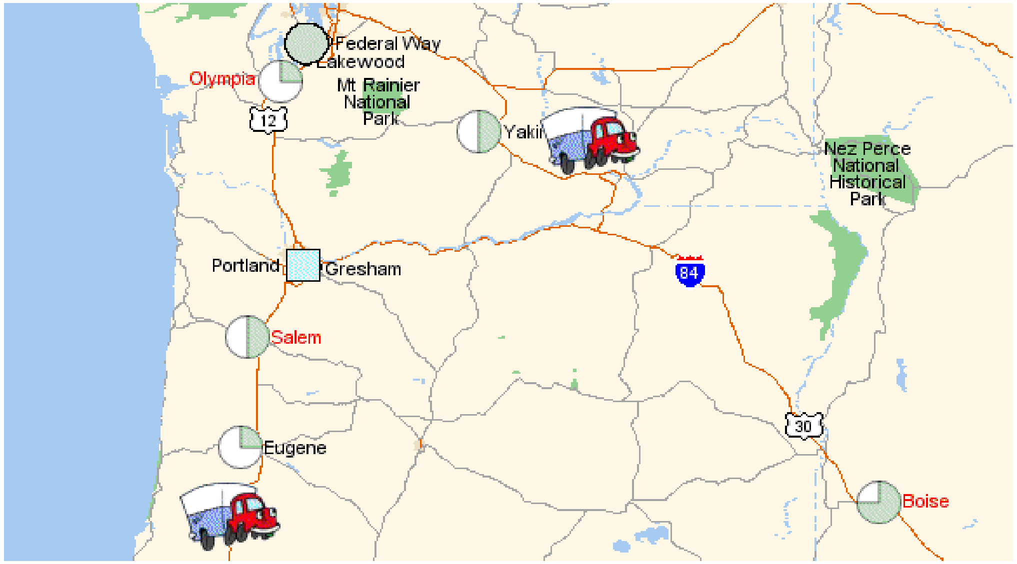


Which way to go?

Solutions to the Dilemma

- ***Epsilon-Greedy Exploration***: Choose greedy action with 1-ε probability. With ε probability pick randomly among all actions.
- ***Optimism under uncertainty***: Initialize the Q-values with maximum possible value Rmax. Choose actions greedily. "Delayed Q-Learning" guarantees polynomial-time convergence.
- ***Explicit Explore and Exploit (E^3):*** Learns models and solves them offline. Explicitly chooses between following optimal policy for the known MDP and reaching an unknown part of MDP. Guarantees polynomial-time convergence.
- ***RMAX:*** Model-based version of optimism under uncertainty – *Implicit* Explore and Exploit

The Curse of Dimensionality



- Number of states is exponential in the number of shops and trucks
- 10 locations, 5 shops, 2 trucks = $(10^2)(5^5)(5^2)$ = 7,812,500 states
- Table-based RL scales exponentially with the problem size (number of state variables)

Solutions to the Curse

- Function Approximation
  - Represent value function compactly using a parameterized function
- Hierarchical Reinforcement Learning
  - Decompose the value function into simpler components
- Approximate Policy Iteration
  - Represent the policy compactly using approximation

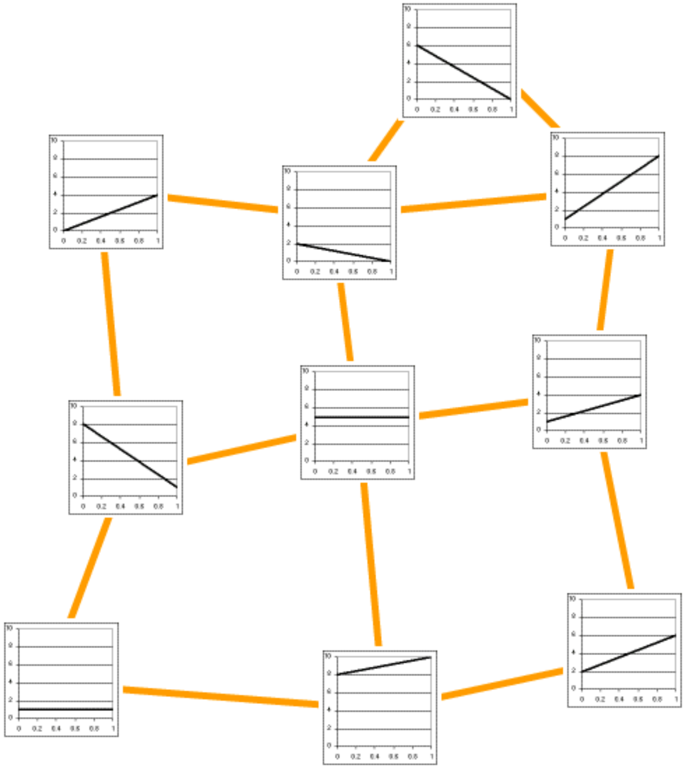
Function Approximation

- Idea: Approximate the value function V(s) or Q(s,a) using a compact function
  - A linear function of carefully designed features
  - A neural network
  - Tabular linear functions
- Compute the temporal difference error in *s* (TD-error)
  - $TD(s) = Max_a\,(R(s,a) + V(s')) \quad - V(s)$
  - $TD(s,a) = R(s,a) + Max_b\, Q(s',b) - Q(s,a)$
- Adjust the parameters of the value function to reduce the (squared) temporal difference error
  - $W \leftarrow W + \alpha\ \ TD(s)\ \{\partial V(s)/\partial W\}$
  - $W \leftarrow W + \alpha\ \ TD(s,a)\ \{\partial Q(s,a)/\partial W\}$

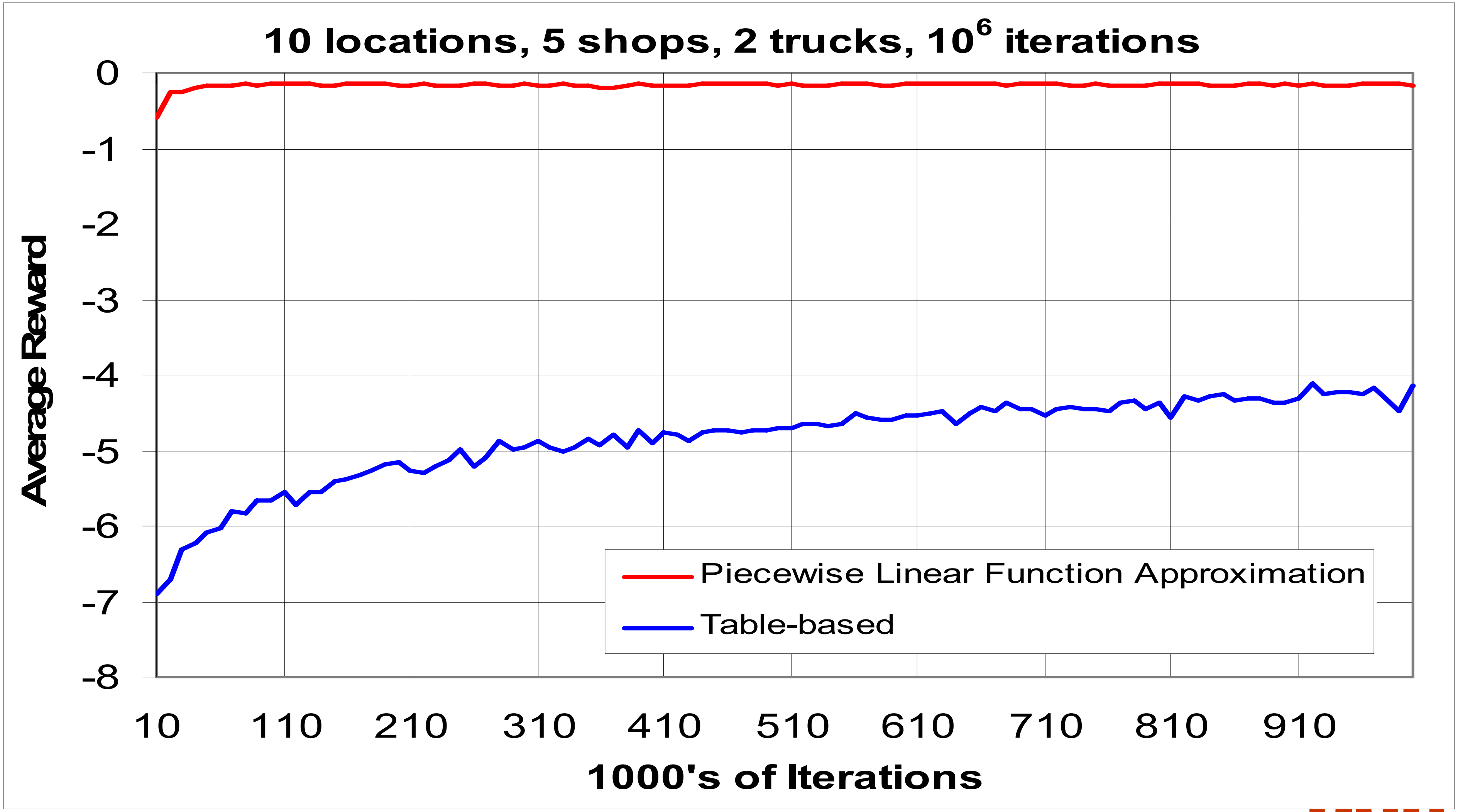
Tabular Linear Function Approximation

- Use a different linear function for each possible 5-tuple of locations $l_1, \ldots, l_5$ of trucks
- Each function is linear in truck loads and shop inventories
- Every function represents

  10 million states
- Million-fold reduction in the number of learnable parameters
- $W \leftarrow W + \alpha \; TD(s) \; \{\partial V(s)/\partial W\}$
- $W_i \leftarrow W_i + \alpha \; TD(s) \; F_{i,k}(s)$, where s belongs to the k^{th} linear function, and $F_{I,k}(s)$ is its i^{th} feature value



Tabular linear function approximation vs. table-based

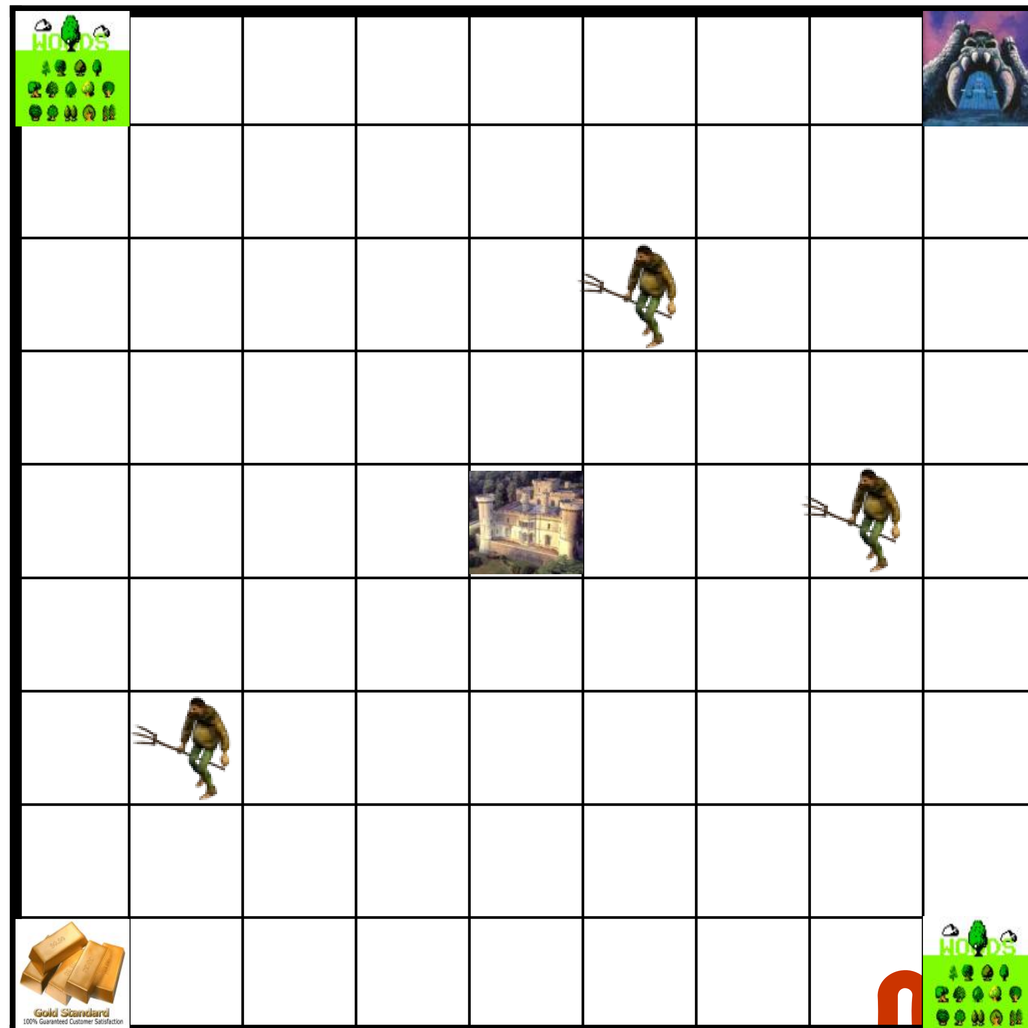


Hierarchical Reinforcement Learning

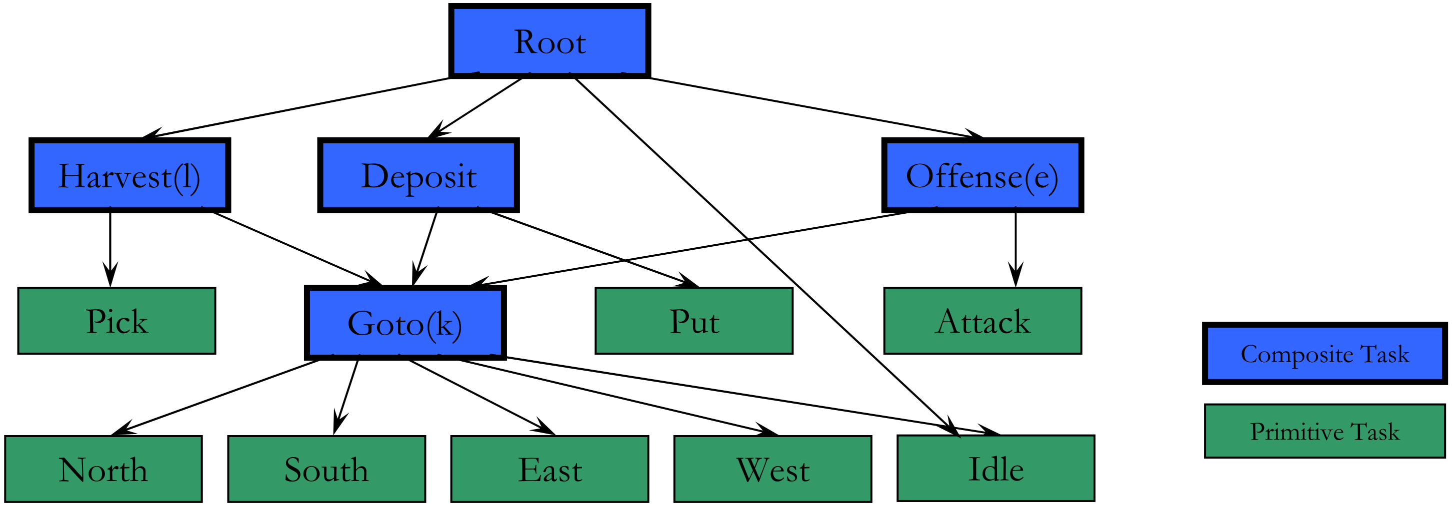
- Many domains are hierarchically organized.
- Tasks have subtasks, subtasks have sub-subtasks and so on.
- Searching the policy space at the lowest level of the action space may be intractable.
- How can we exploit task hierarchies to learn efficiently?
- Many formalisms exist
  - Options (Precup , Sutton, and Singh)
  - MAXQ (Dietterich)
  - ALisp (Andre, Murthy, Russell)

Resource Gathering Domain

- Grid world domain
- Multiple peasants harvest resources (wood, gold) to replenish the home
- Attack the enemy's base any time it pops up
- Number of states exponential in the number of peasants and resources



MAXQ Task Hierarchy

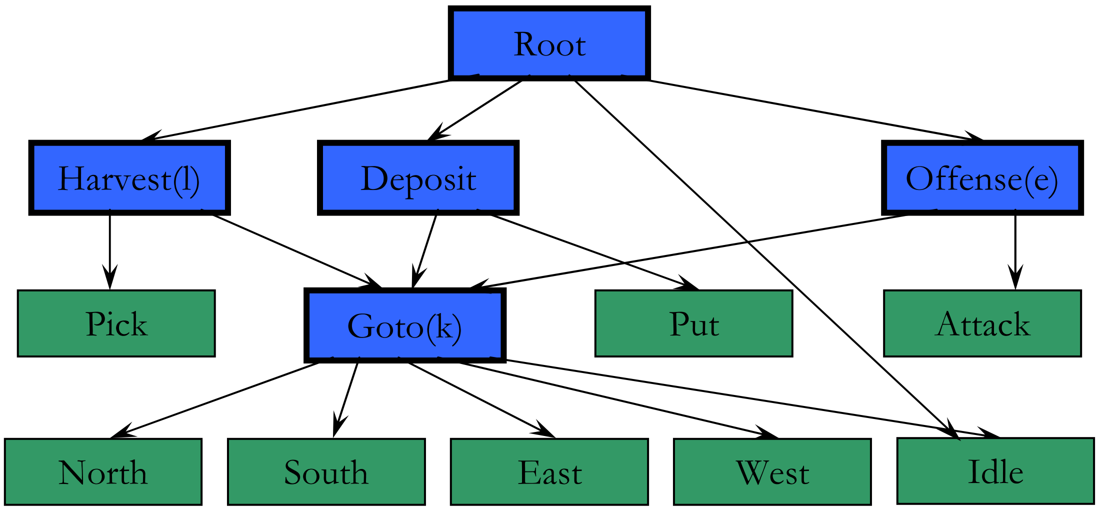


- Each subtask M_i is defined by Termination (goal) predicate G_i, Actions A_i, and State Abstraction B_i
- The subtasks of task M_i are its available actions or subroutines it can call.
- Control returns to the task M_i when its subtask finishes.
- Each M_i learns a policy π: S $\rightarrow$ Subtasks(M_i)

Value Function Decomposition

- $V_i(s)$ = Total optimal expected reward during task *i* when starting from state *s*
- $Q_i(s,j)$ = Total expected reward during task *i*, when starting from state *s* and task *j* and acting optimally
- $V_i(s) = Max_j\ Q_i(s,j)$
- $C_i(s,j)$ = Completion reward = Total expected reward to complete task *i* after *j* is done in *s.*
- $Q_i(s,j) = V_j(s) + C_i(s,j)$

Choosing Actions



$$Q_{root}(harvest) = V_{harvest} + C_{root}(harvest)$$

$$= Max_a\,[V_a + C_{harvest}(a)] + C_{root}(harvest)$$

$$= Max_a\,[Max_b\,[V_b + C_a(b)] + C_{harvest}(a)] + C_{root}(harvest)$$

MAXQ-Q Learning

Learn completion functions $C_i(j)$ for internal nodes and value functions V_j for the leaf nodes. Let *s* be current state and s' be the state after subtask *j*

– $C_i(s,j) \leftarrow (1-\alpha)\, C_i(s,j) + \alpha\, V_i(s')$

$\leftarrow (1-\alpha)\, C_i(s,j) + \alpha\, Max_k\, Q_i(s',k)$

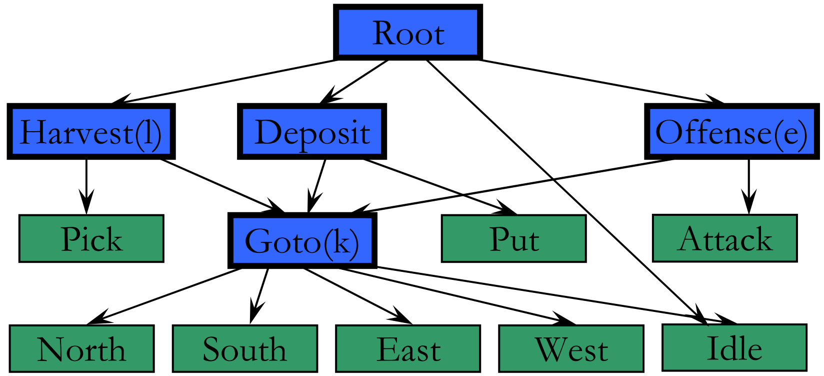
$\leftarrow (1-\alpha)\, C_i(s,j) +$

$\alpha\, Max_k \{V_k(s') + C_i(s',k)\}$

How does the hierarchy help?

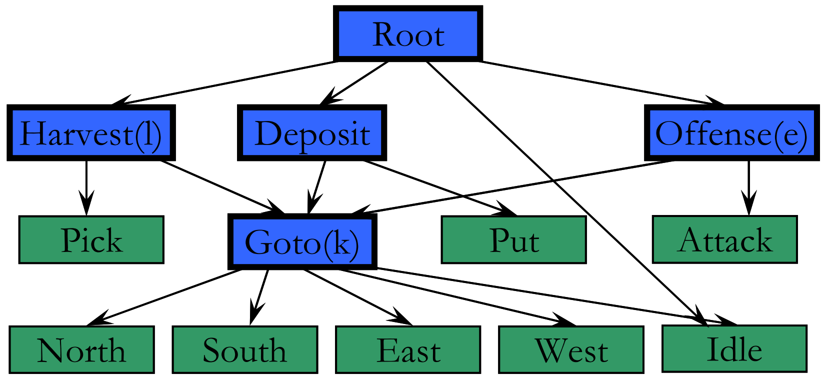
- Temporal Abstraction
  - Reduces the number of decision/update points (search depth)
- State Abstraction
  - What the peasants carry is irrelevant to the Goto actions
  - Other agents' locations are irrelevant to the Goto and the Deposit actions
- Funneling
  - The high level tasks, e.g., Harvest, are considered only in a small number of states (special locations)
- Subtask Sharing
  - The same subtask, e.g., Goto, is called by several other tasks; hence knowledge transfers between the tasks
- Sharing among multiple agents

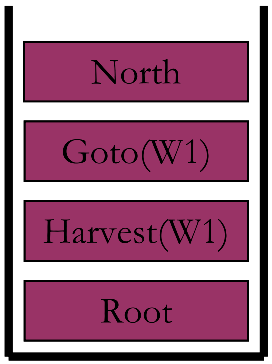
Single Hierarchical Agent (effector)



- The agent has a decomposed value function
- The agent has a task stack
- Each subtask is given appropriate abstraction

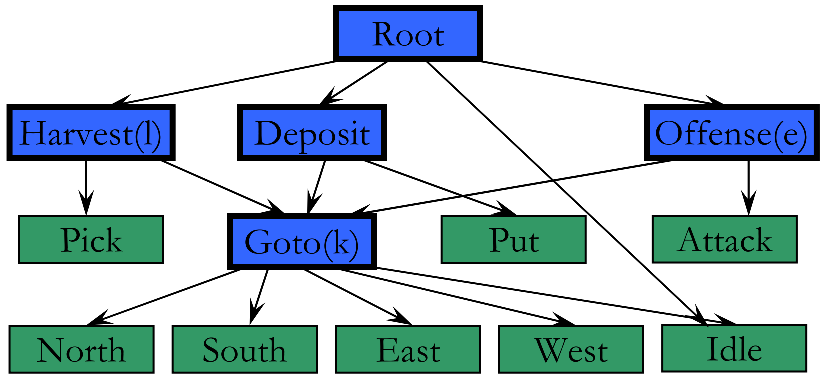
Simple Multi-Effector Setup

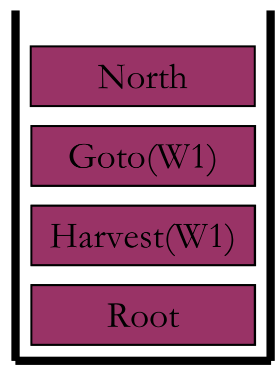




- Every effector has its own decomposed value function (depicted by the separate task hierarchies)
- Every effector has its own task stack

Multiple Agents with Shared Hierarchy (MASH)



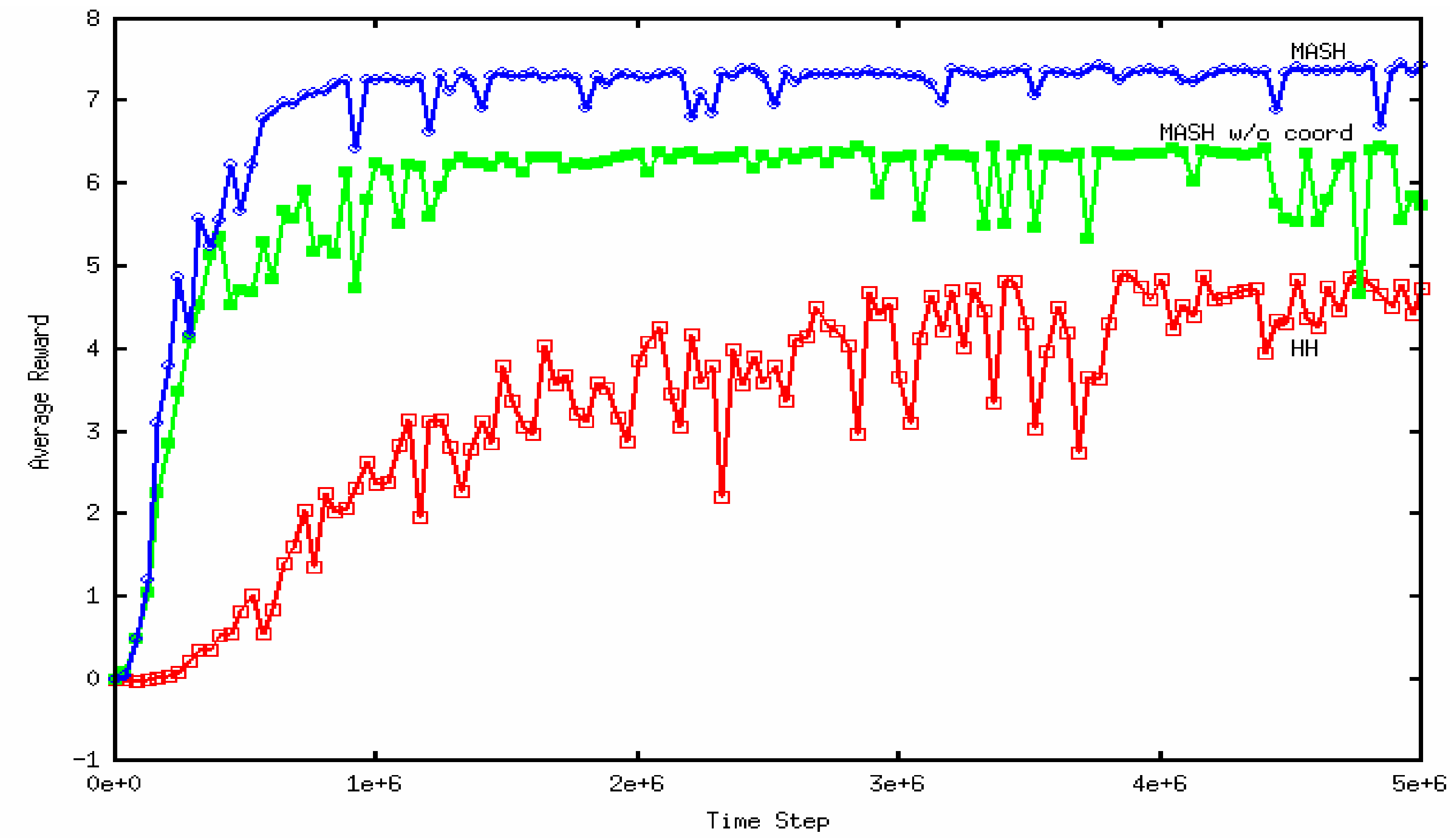


- The effectors share one decomposed value function
- Every effector still has its own task stack (control thread)
- Effectors may coordinate by sharing task information

Resource Gathering Results (Model-based Average-Reward RL)

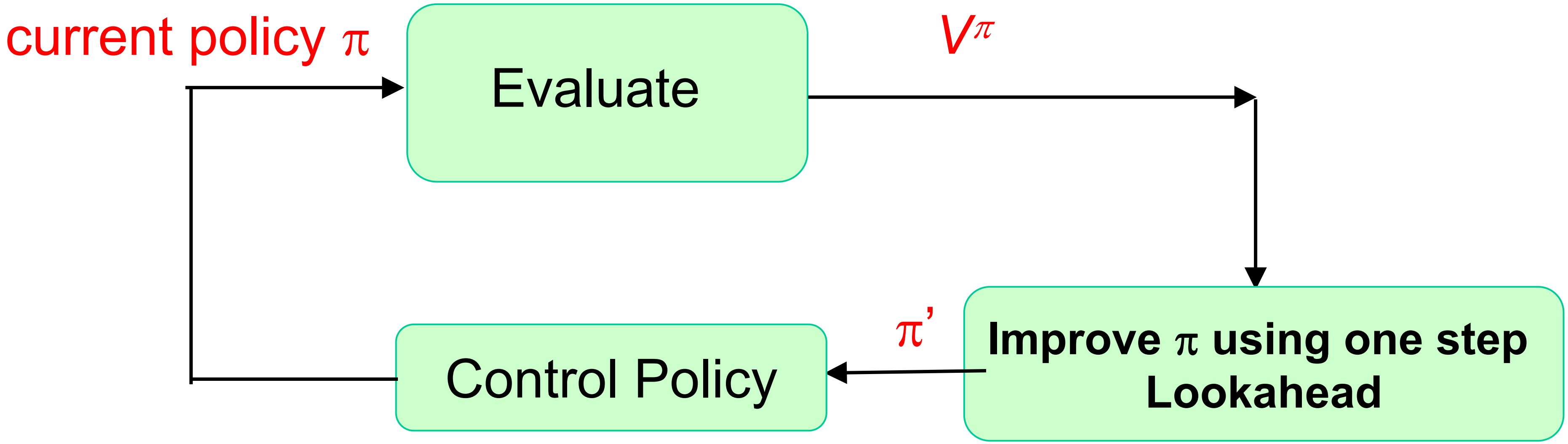
4 agents in a 25 × 25 grid (30 runs):

- Rewards: Deposit = 100, Collision = -5, Offense = 50
- Unable to run this setup for coordinating agents with separate value functions



Approximate Policy Iteration (Fern, Yoon and Givan)

- Based on Policy Iteration
- Converges to globally optimal policies in enumerative state-spaces
- Represents the policy explicitly

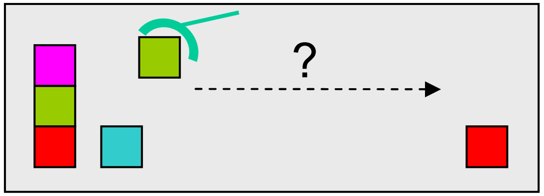


Taxanomic Decision Lists

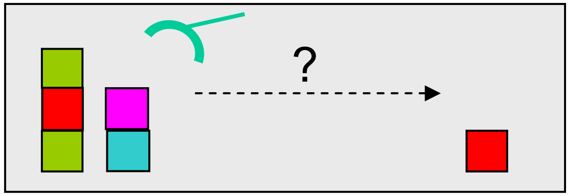
Yoon, Fern, and Givan, 2002

A simple policy to clear a goal block	Taxonomic Syntax
1. Putdown blocks being held	1. holding : putdown
2. Pickup clear blocks above gclear blocks (those that are clear in the goal)	2. clear $\cap$ (on* gclear) : pickup

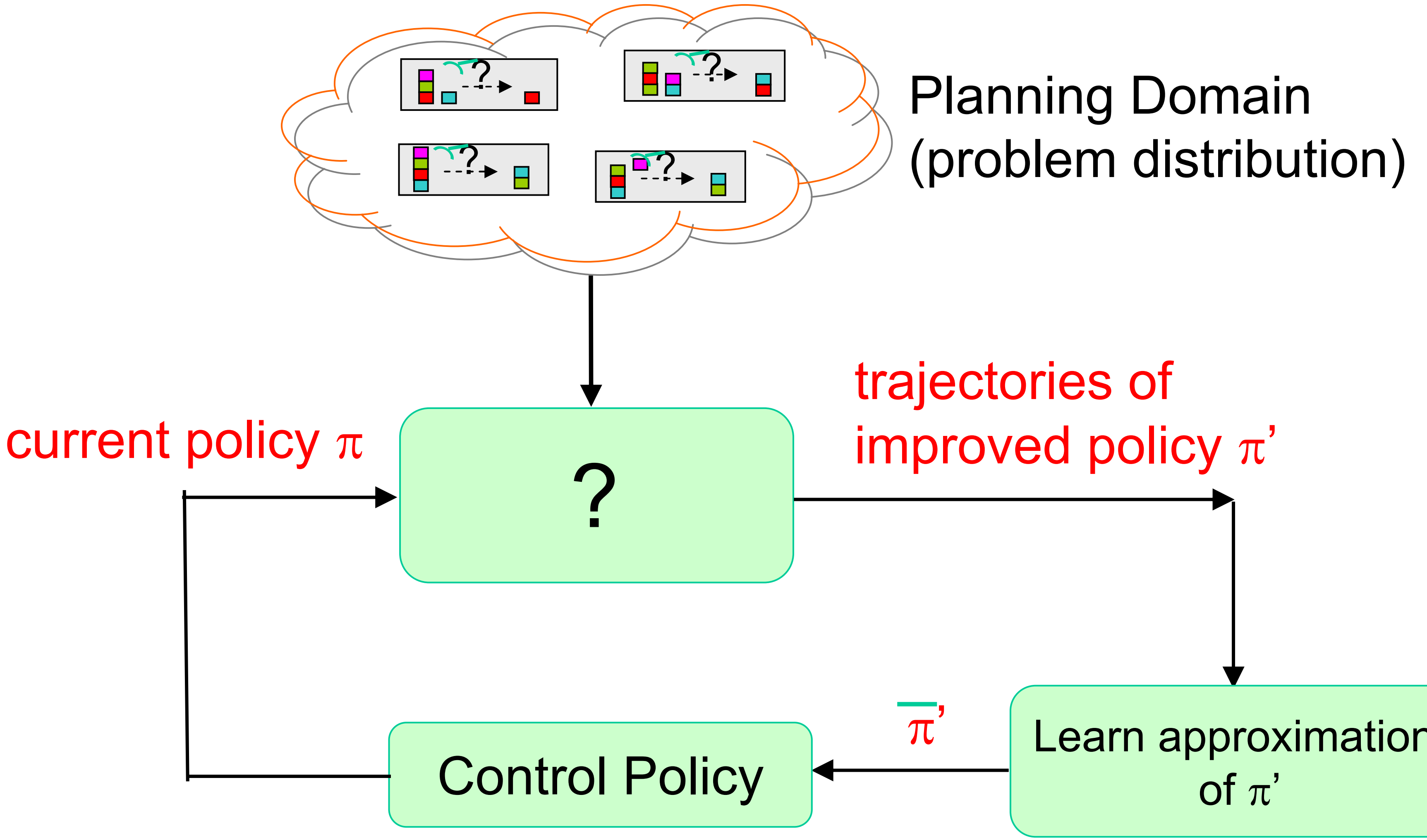
1.



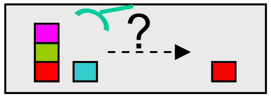
2.



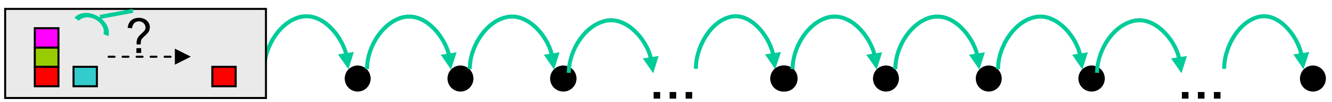
Approximate Policy Iteration (Fern, Yoon and Givan, 2003)



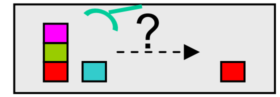
Computing π’ Trajectories from π

Given: current policy π and problem 

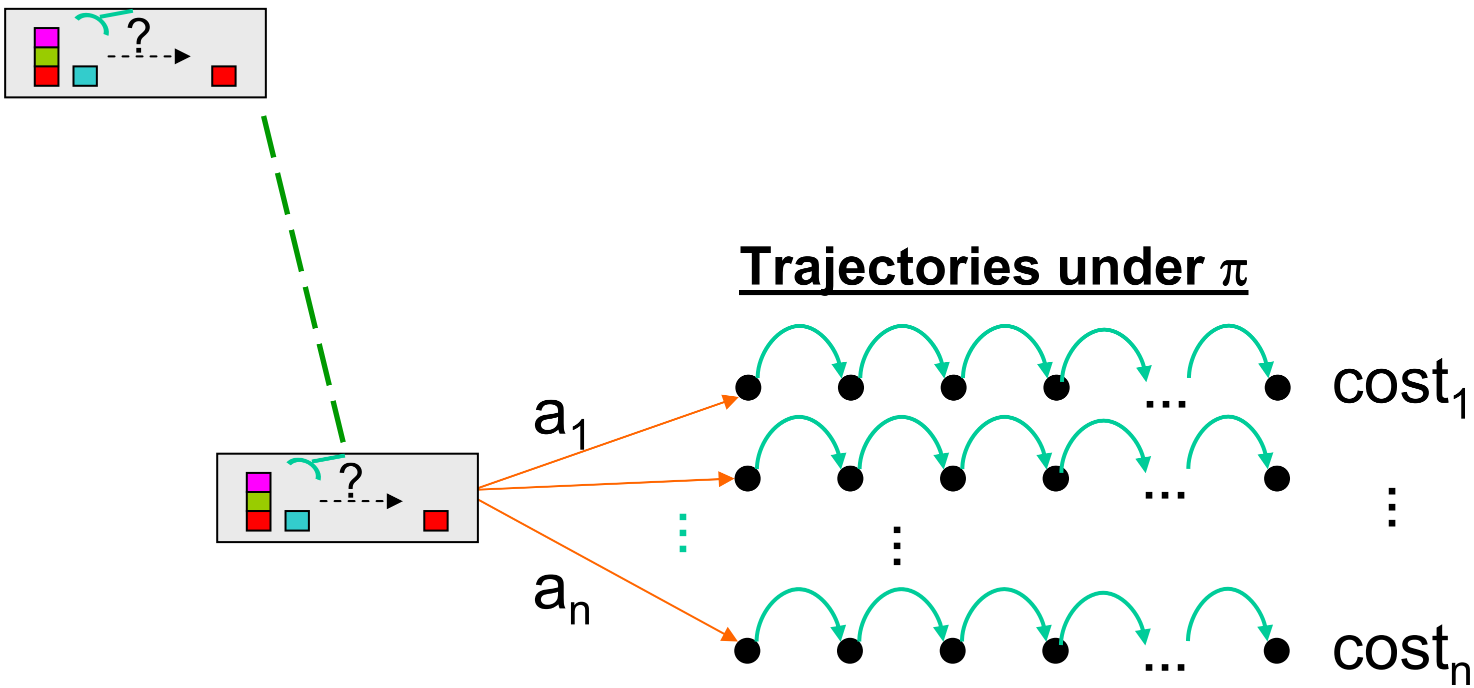
Output: a trajectory under improved policy π’



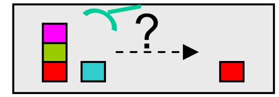
Computing π' Trajectories from π

Given: current policy π and problem 

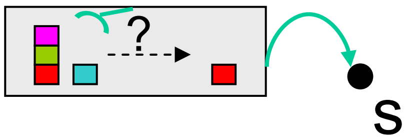
Output: a trajectory under improved policy π'



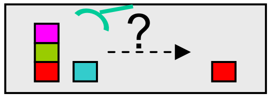
Computing π' Trajectories from π

Given: current policy π and problem 

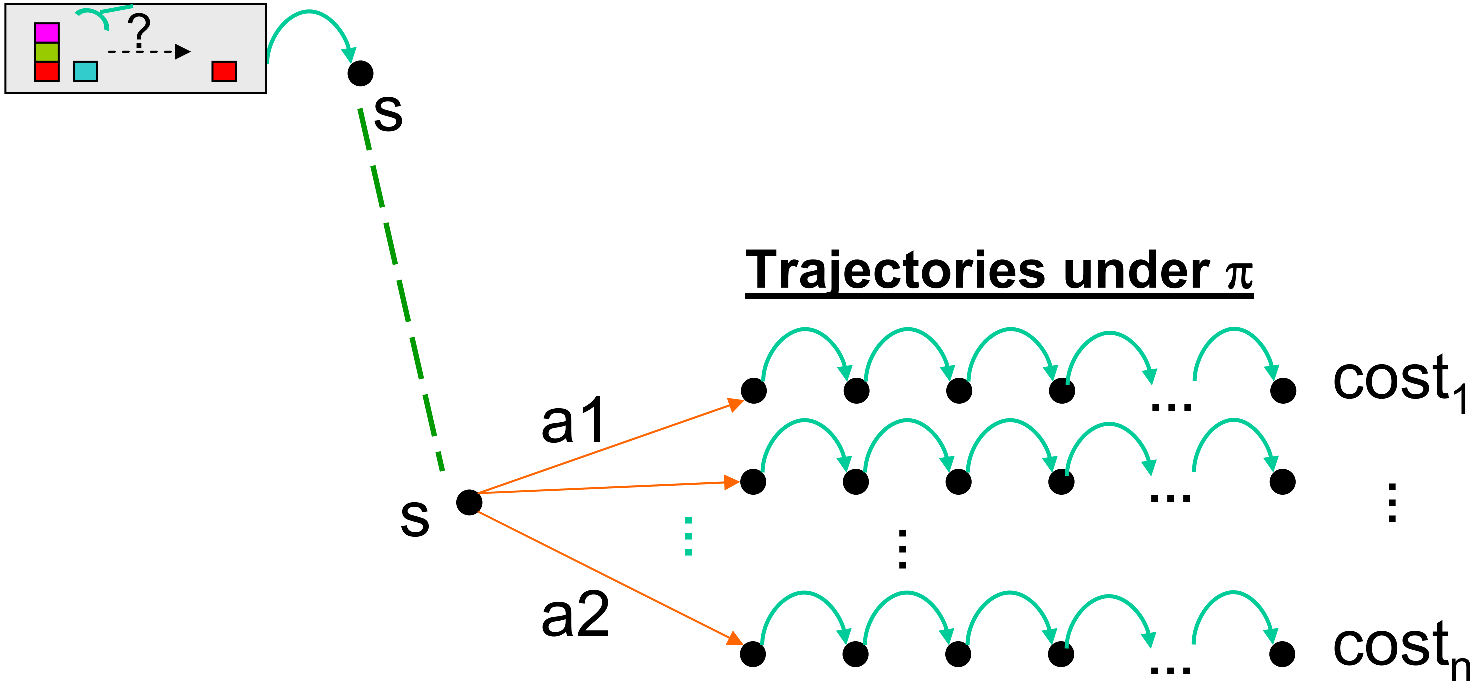
Output: a trajectory under improved policy π'



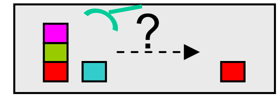
Computing π' Trajectories from π

Given: current policy π and problem 

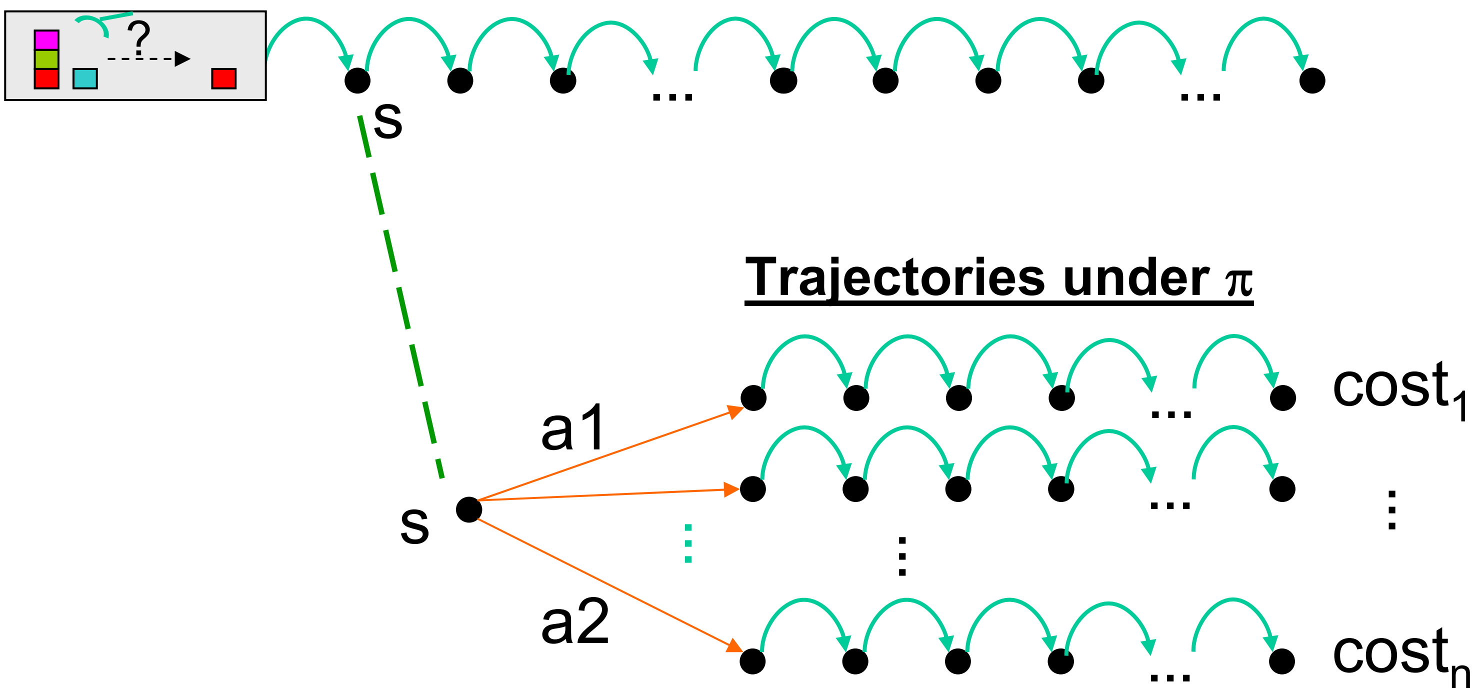
Output: a trajectory under improved policy π'



Computing π' Trajectories from π

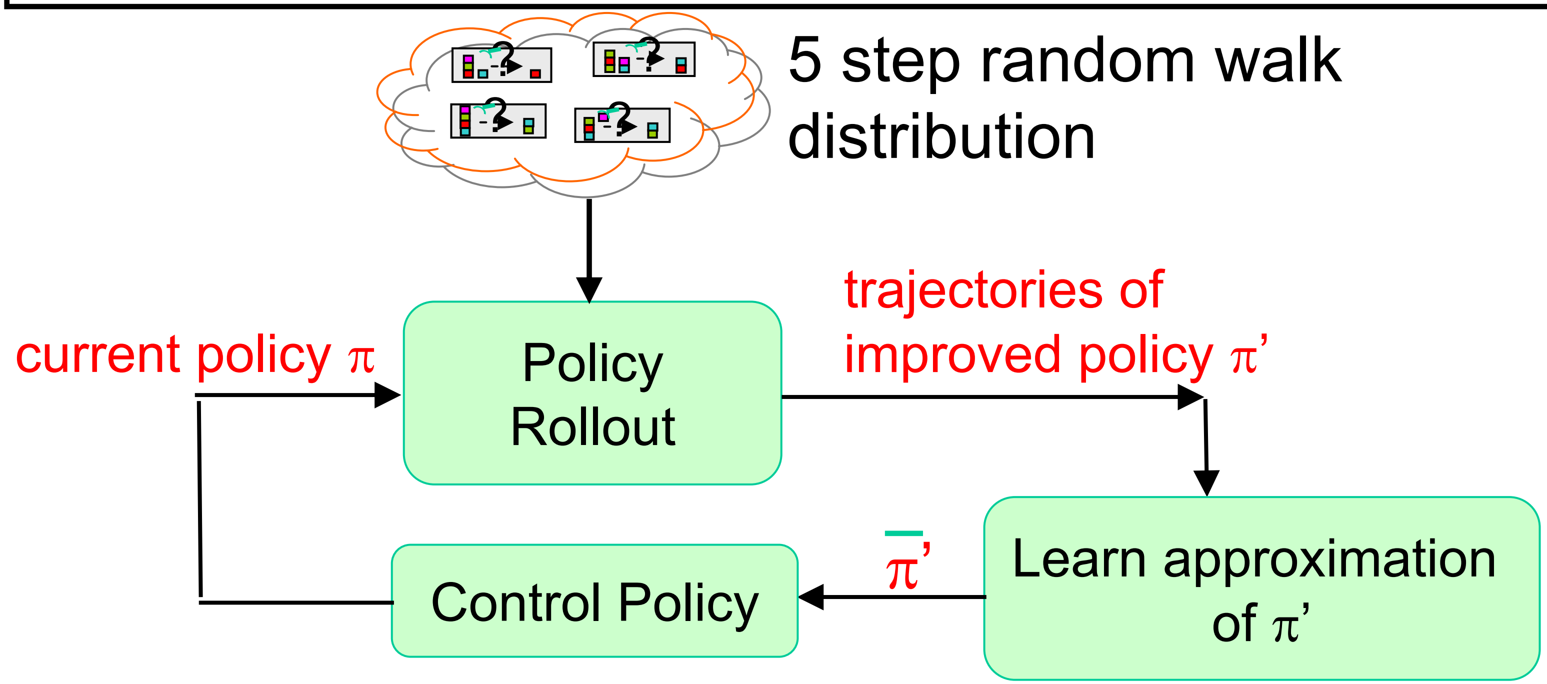
<u>Given</u>: current policy π and problem 

<u>Output</u>: a trajectory under improved policy π'

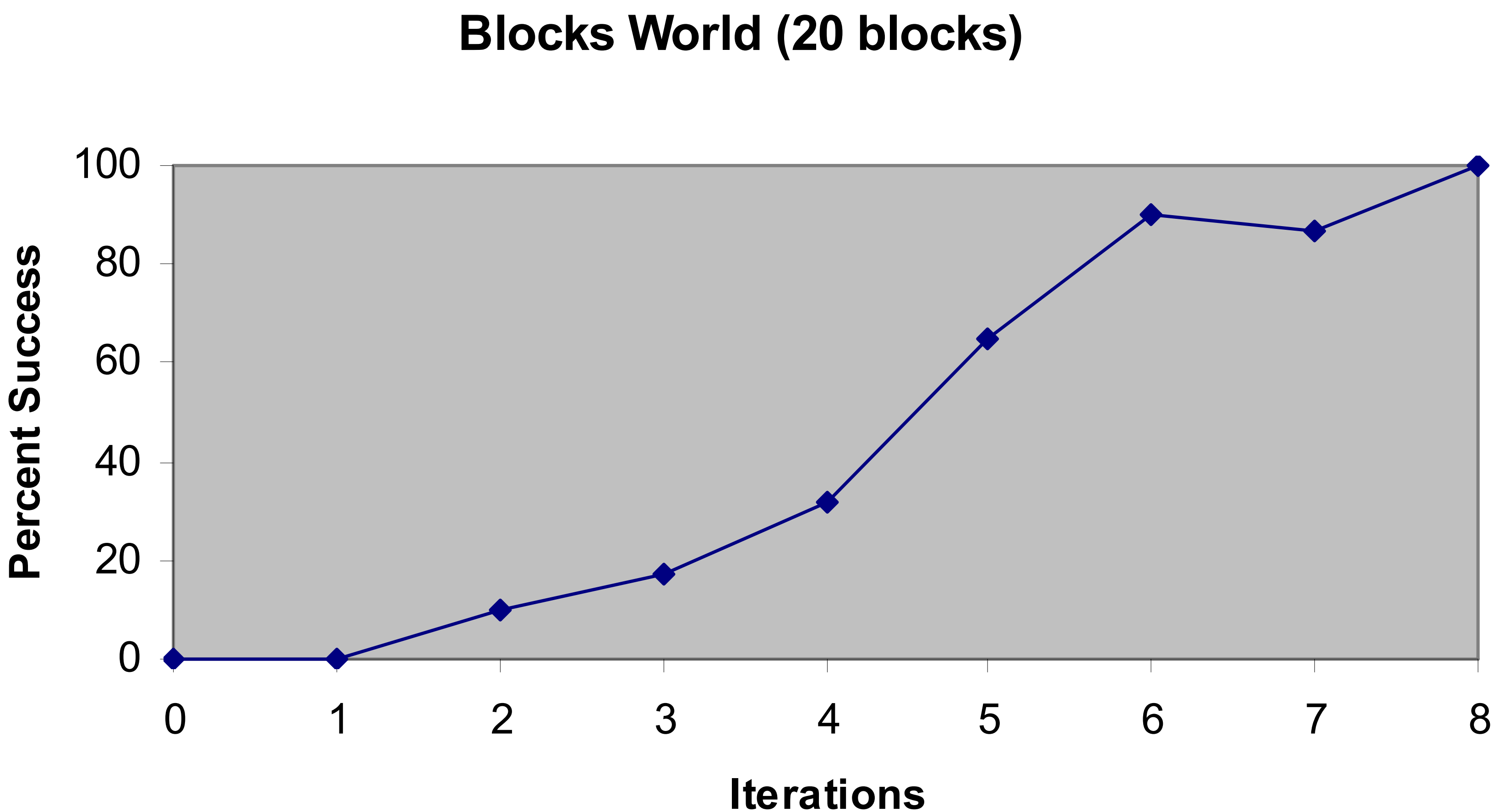


Random Walk Bootstrapping

Objective: Learn policy for long random walk distributions by transferring knowledge from short random walk distributions



Experimental Results



Random walk length: 4 14 54 54 54 54 334 334 334

Domains with Good Policies

Success Percentage

	Blocks	Elevator	Schedule	Briefcase	Gripper
API	100	100	100	100	100
FF-Plan	28	100	100	0	100

Typically our solution lengths are comparable to FF's.

Domains without Good Policies

Success Ratio

	Freecell	Logistics
API	0	0
FF-Plan	47	100

Conclusions

- Function approximation can result in faster convergence if chosen carefully.
  - But there is no guarantee of convergence in most cases.
- Task hierarchies and shared value functions among agents leads to fast learning
  - The hierarchies and abstractions are given and carefully designed.
- Approximate Policy Iteration is effective in many planning domains
  - The policy language must be carefully chosen to contain a good policy

Current Research

- Learning the task hierarchies including termination conditions and state abstraction
- Theory of function approximation – few convergence results are known
- Transfer Learning: How can we learn in one domain and perform well in a related domain?
- Relational Reinforcement Learning
- Combining planning and reinforcement learning
- Partially observable MDPs
- Game playing and assistantship learning