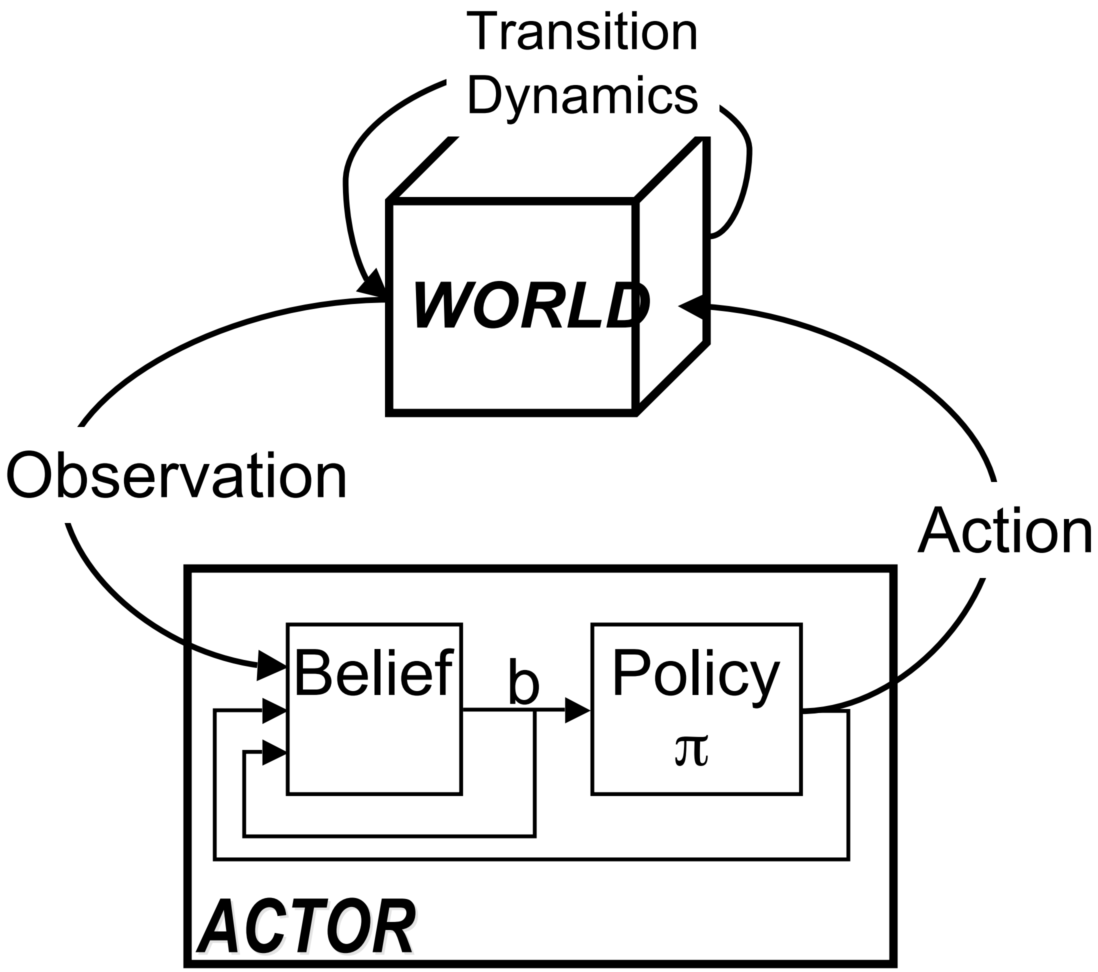
POMDP Tutorial

Preliminaries: Problem Definition

- Agent model, POMDP, Bayesian RL



Markov Decision Process

-X: set of states $[x_s,x_r]$

- *state component*
- *reward component*

--A: set of actions

-T=P(x'|x,a): transition and reward probabilities

-O: Observation function

-b: Belief and info. state

-π: *Policy*

Environment: You are in state 65. You have 4 possible actions.

Agent: I'll take action 2.

Environment: You received a reinforcement of 7 units. You are now in state 15. You have 2 possible actions.

Agent: I'll take action 1.

Environment: You received a reinforcement of -4 units. You are now in state 65. You have 4 possible actions.

Agent: I'll take action 2.

Environment: You received a reinforcement of 5 units. You are now in state 44. You have 5 possible actions.

⋮ ⋮

Sequential decision making with uncertainty in a changing world

- ***Dynamics-*** The world and agent change over time. The agent may have a model of this change process
- ***Observation-*** Both world state and goal achievement are accessed through indirect measurements (sensations of state and reward)
- ***Beliefs -*** Understanding of world state with uncertainty
- ***Goals-*** encoded by *rewards!* => Find a policy that can maximize total reward over some duration
- ***Value Function-*** Measure of goodness of being in a belief state
- ***Policy-*** a function that describes how to select actions in each belief state.

Markov Decision Processes (MDPs)

In RL, the environment is a modeled as an MDP, defined by

S – set of states of the environment

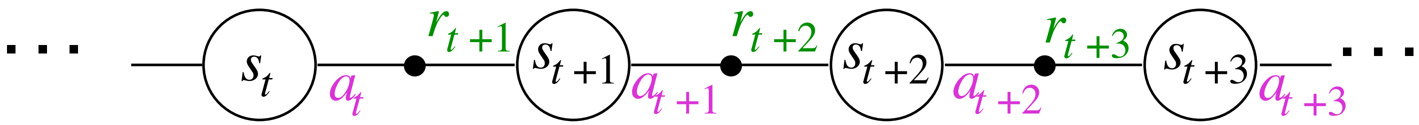
$A(s)$ – set of actions possible in state s within S

$P(s,s',a)$ – probability of transition from s to s' given a

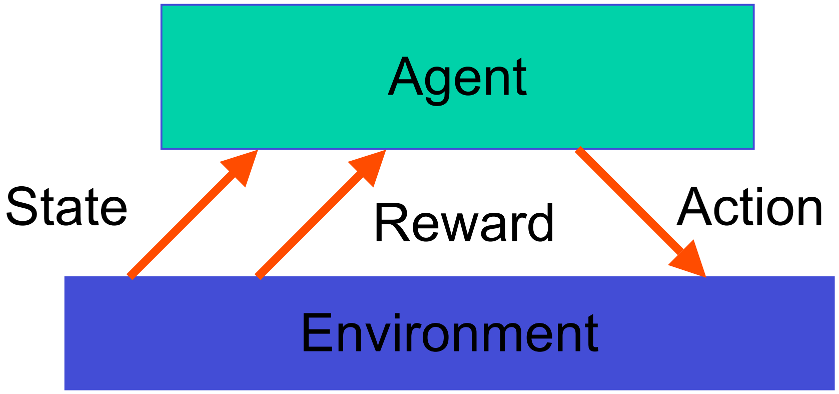
$R(s,s',a)$ – expected reward on transition s to s' given a

g – discount rate for delayed reward

discrete time, $t = 0, 1, 2, \ldots$



MDP Model



$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r1]{a1} s2 \xrightarrow[r2]{a2} s3$$

Process:

- *Observe* state s_t in S
- Choose action at in A_t
- Receive immediate reward r_t
- State changes to s_{t+1}

MDP Model <S, A, T, R>

The Objective is to Maximize Long-term Total Discounted Reward

Find a policy $\pi : \; s \in S \rightarrow a \in A(s)$ **(could be stochastic)**

that maximizes the value (expected future reward) of each s **:**

$$V^{\pi}(s) = E\left\{ r_{t+1} + \gamma\, r_{t+2} + \gamma^{2} r_{t+3} + \;\ldots \middle|\; s_t = s,\ \pi \right\}$$

and each s,a **pair:**

rewards

$$Q^{\pi}(s,a) = E\left\{ r_{t+1} + \gamma\, r_{t+2} + \gamma^{2} r_{t+3} + \;\ldots \middle|\; s_t = s,\ a_t = a,\ \pi \right\}$$

These are called value functions - cf. evaluation functions

Policy & Value Function

- Which action, *a*, should the agent take?
  - In MDPs:
    - Policy is a mapping from *state* to action, π: S → A

- Value Function $V_{\pi,t}(S)$ given a policy π
  - The expected sum of reward gained from starting in state s executing non-stationary policy π for t steps.

- Relation
  - Value function is evaluation for policy based on the long-run value that agent ***expects*** to gain from executing the policy.

Optimal Value Functions and Policies

There exist optimal value functions:

$$V^*(s) = \max_\pi V^\pi(s) \qquad Q^*(s,a) = \max_\pi Q^\pi(s,a)$$

And corresponding optimal policies:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

π^* **is the greedy policy with respect to** Q^*

Optimization (MDPs)

- Recursively calculate expected long-term reward for each *state/belief:*

$$V_t^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_{t-1}^*(s') \right]$$

- Find the action that maximizes the expected reward:

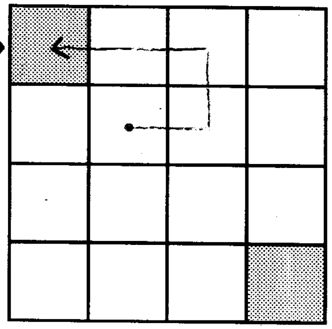
$$\pi_t{*}(s) = \arg\max_a \left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_{t-1}{*}(s') \right]$$

Policies are not actions

- There is an important difference between policies and action choices
- **A Policy** is a (possibly stochastic) mapping between “states” and actions (where states can be beliefs or information vectors). Thus a policy is a *strategy, a* decision rule that can encode action contingencies on context, etc.

Example of Value Functions

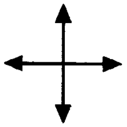
Terminal State



Reward is -1

on all transitions

A Grid-Environment



Actions

▨ 0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	▨ 0

State-Value Function for the random policy

V^{π}

$Q^{\pi}(s,a)$

V^*

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

Optimal State-Value Function

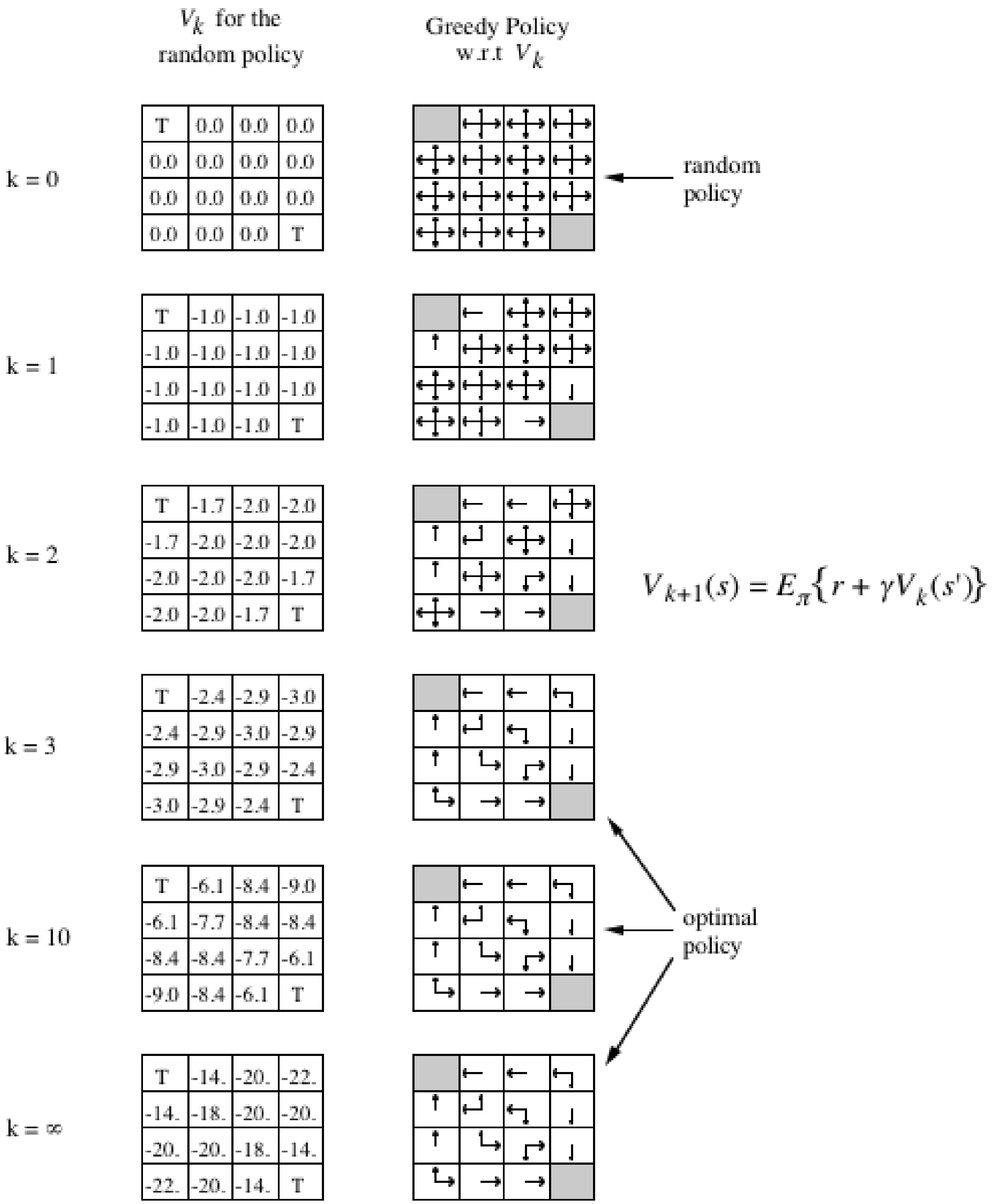
▨	←	←	←↓
↑	←↑	←↑→↓	↓
↑	←↑→↓	↓→	↓
↑→	→	→	▨

Optimal Policies

π^*

Q^*?

Q^{π}?



k = 0

T	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	T

k = 1

T	-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	T

k = 2

T	-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	T

$$V_{k+1}(s) = E_\pi\{r + \gamma V_k(s')\}$$

k = 3

T	-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	T

k = 10

T	-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	T

k = ∞

T	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	T

Policy Iteration

$$\pi_1 \longrightarrow V^{\pi_1} \longrightarrow \pi_2 \longrightarrow V^{\pi_2} \longrightarrow \cdots \pi^* \longrightarrow V^* \longrightarrow \pi^*$$

Policy Eval Step "Expectation"

Greedification Step "Maximization"

Improvement is <u>monotonic</u>

Converges in finite # steps $\leqslant |S|$

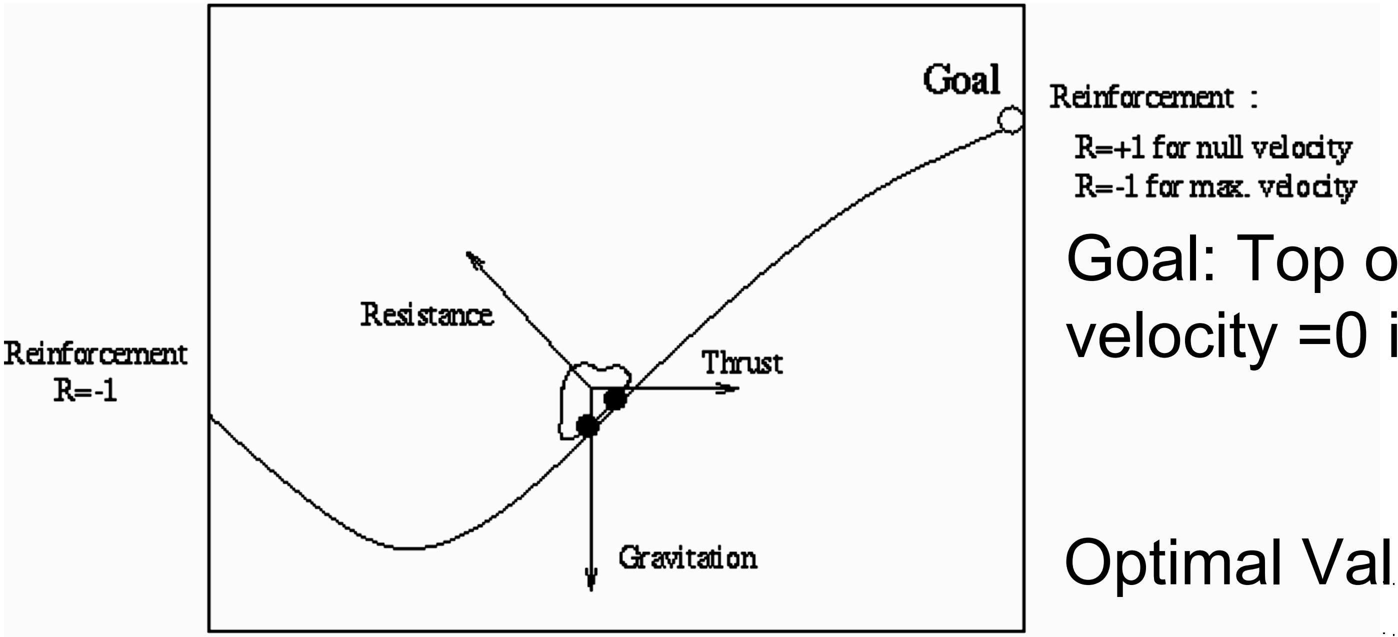
<u>Generalized</u> Policy Iteration:

Q!

Intermix the two steps more finely

state by state, action by action, sample by sample

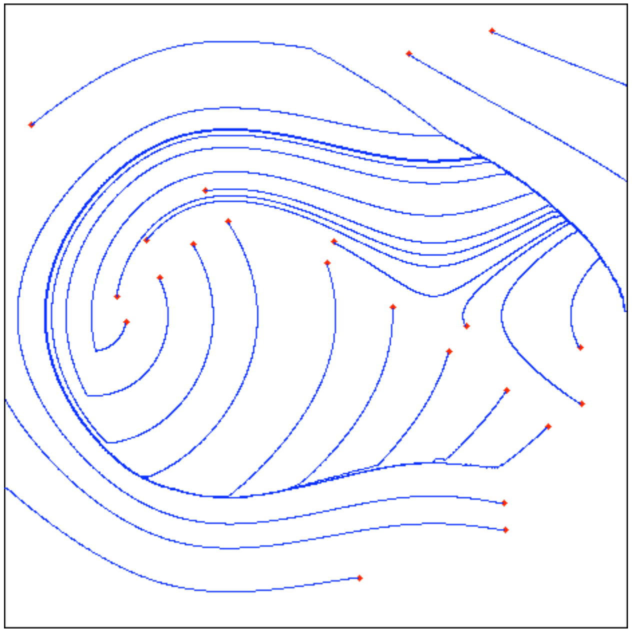
Classic Example: Car on the hill

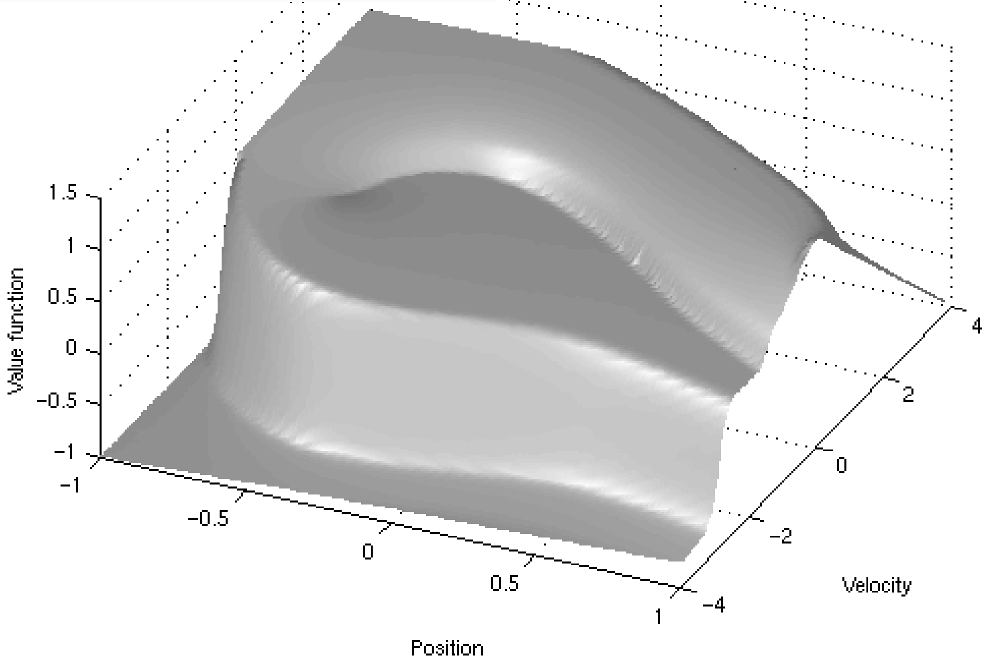


Goal: Top of the hill, final velocity =0 in minimum time

Optimal Value function

Optimal Trajec.





Ball-Balancing

- RL applied to a real physical system-illustrates <u>online</u> learning
- An easy problem...but learns as you watch

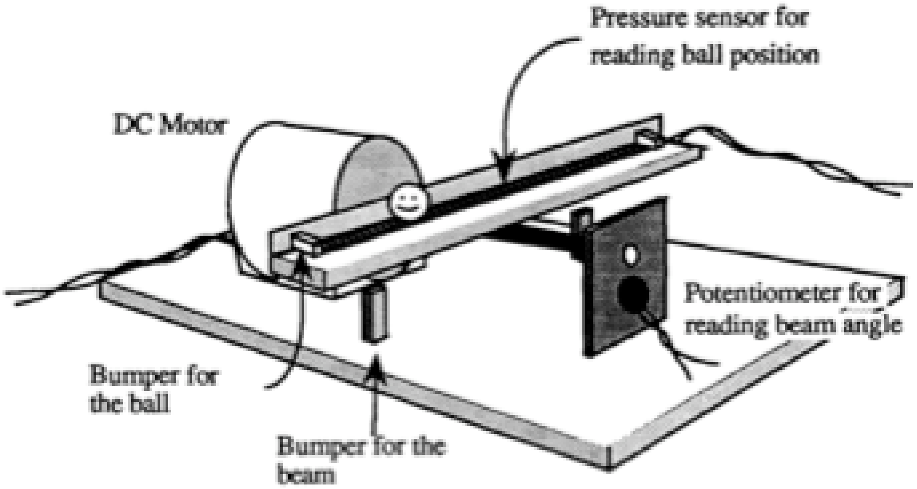

Pressure sensor for reading ball position
DC Motor
Potentiometer for reading beam angle
Bumper for the ball
Bumper for the beam

Figure 1 The ball balancer

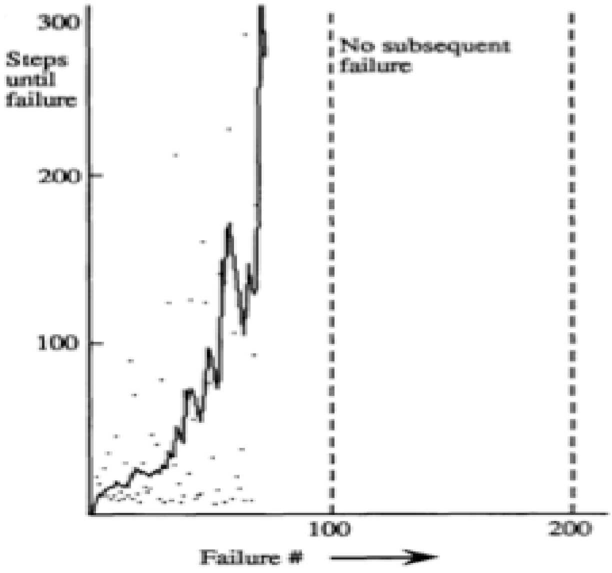
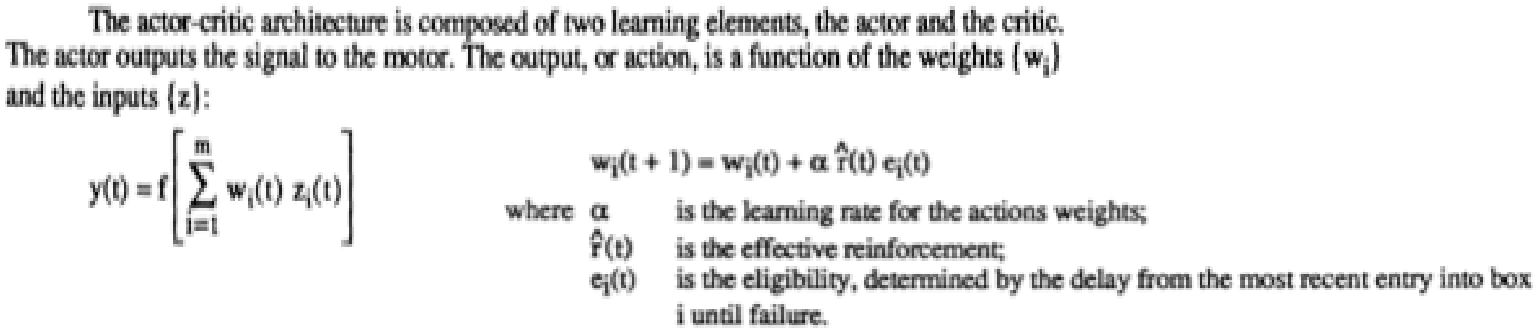


Figure 2 A learning series



The actor-critic architecture is composed of two learning elements, the actor and the critic. The actor outputs the signal to the motor. The output, or action, is a function of the weights $\{w_i\}$ and the inputs $\{z\}$:

$$y(t) = f\left[\sum_{i=1}^{m} w_i(t)\ z_i(t)\right]$$

$$w_i(t+1) = w_i(t) + \alpha\,\hat{r}(t)\,e_i(t)$$

where α is the learning rate for the actions weights;

$\hat{r}(t)$ is the effective reinforcement;

$e_i(t)$ is the eligibility, determined by the delay from the most recent entry into box i until failure.

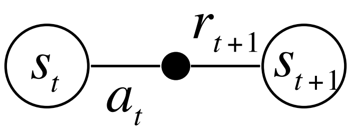
Smarts Demo

- Rewards ?
- Environment state space?
- Action state space ?
- Role of perception?
- Policy?

1-Step Model-free Q-Learning

A model of the environment is not necessary to learn to act in a stationary domain

On each state transition:



Update:

$$\underbrace{Q(s_t,a_t)}_{\text{a table entry}} \leftarrow Q(s_t,a_t) + \alpha \underbrace{\left[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t) \right]}_{\text{TD Error}}$$

$$\lim_{t \to \infty} Q(s,a) \to Q^*(s,a)$$

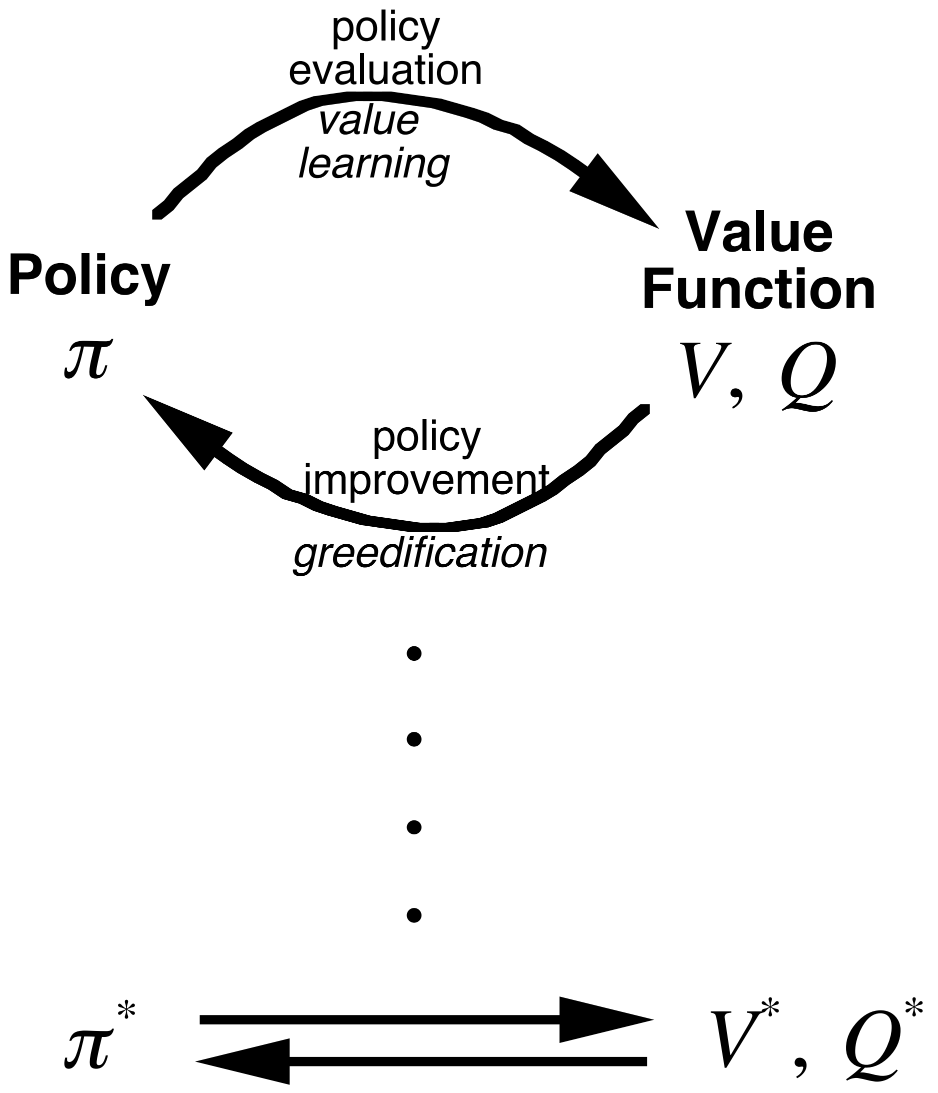
Optimal behavior found without a model of the environment!

$$\lim_{t \to \infty} \pi_t \to \pi^*$$

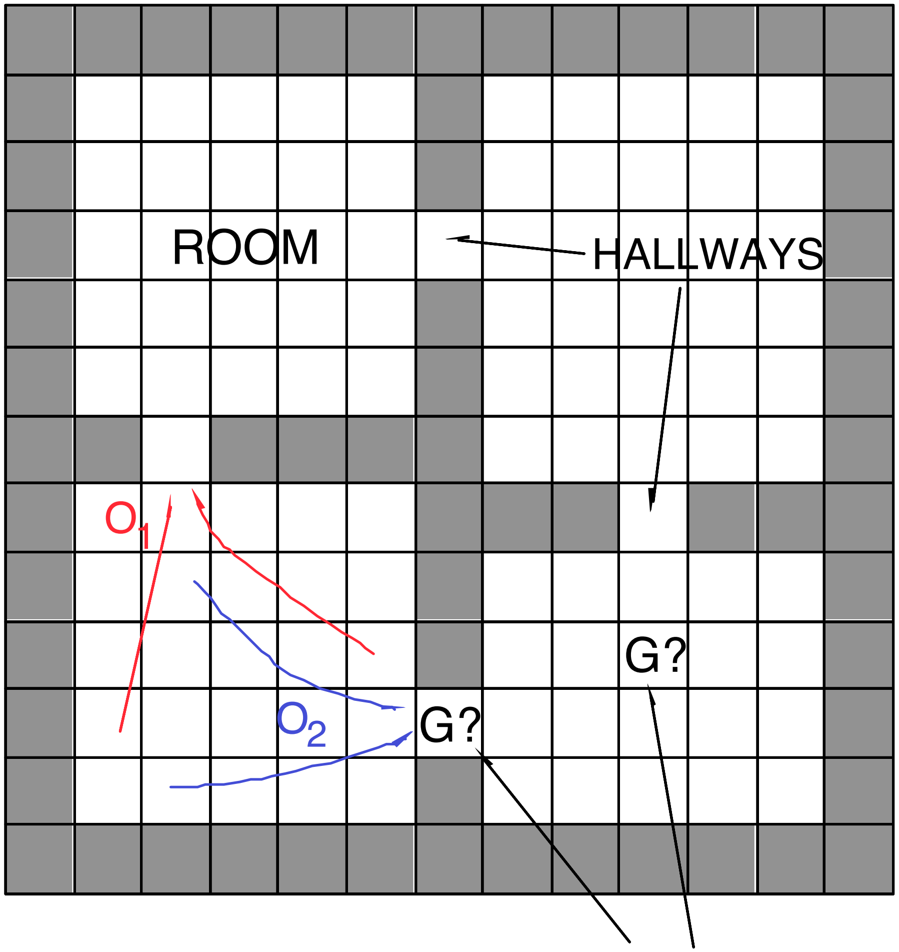
(Watkins, 1989)

Assumes finite MDP

RL Theory: Generalized Policy Iteration



Rooms Example

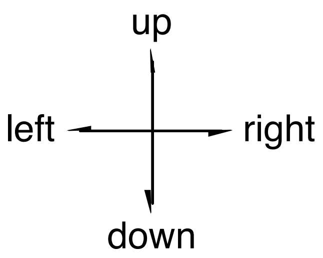


Goal states are given
a terminal value of 1

4 rooms

4 hallways

4 unreliable
primitive actions



Fail 33%
of the time

8 multi-step options

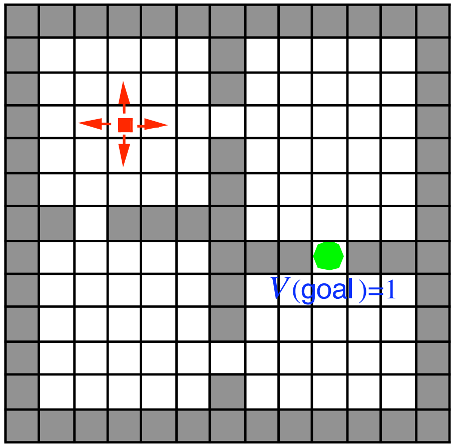
(to each room's 2 hallways)

Given goal location,
quickly plan shortest route

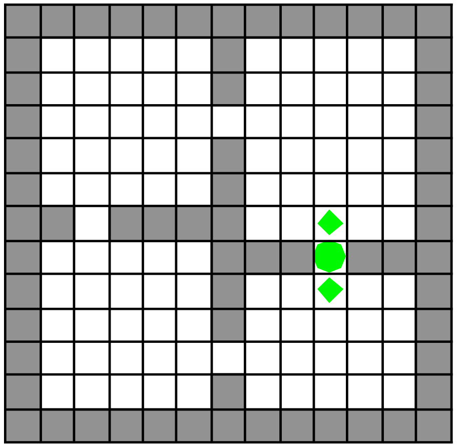
All rewards zero
$\gamma = .9$

Action representation matters: Value functions learned faster for macro-actions

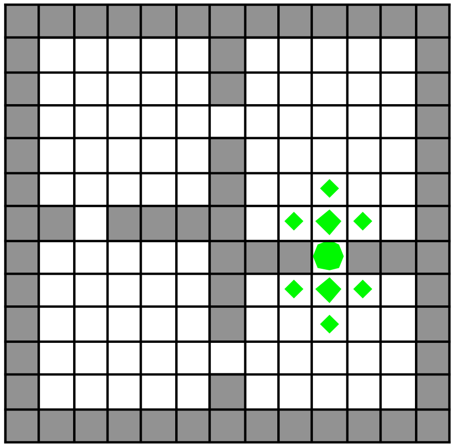
with primitive actions (cell-to-cell)



Iteration #1

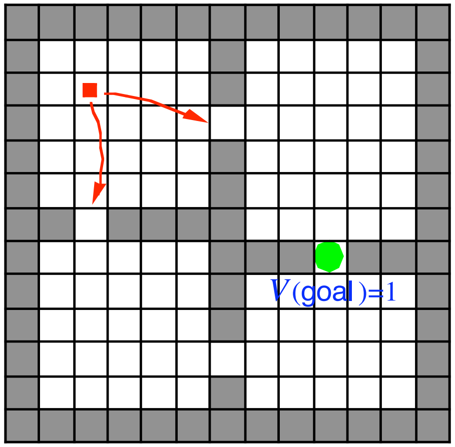


Iteration #2

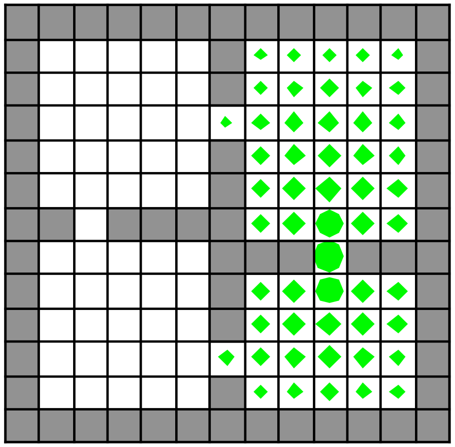


Iteration #3

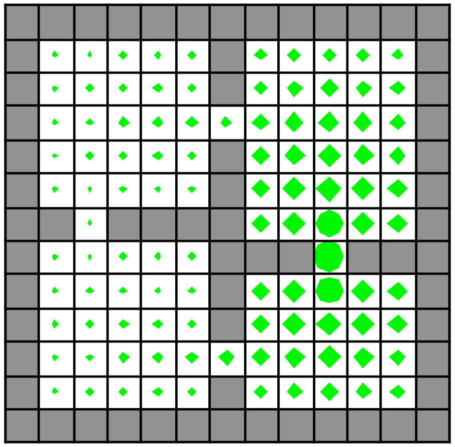
with behaviors (room-to-room)



Iteration #1

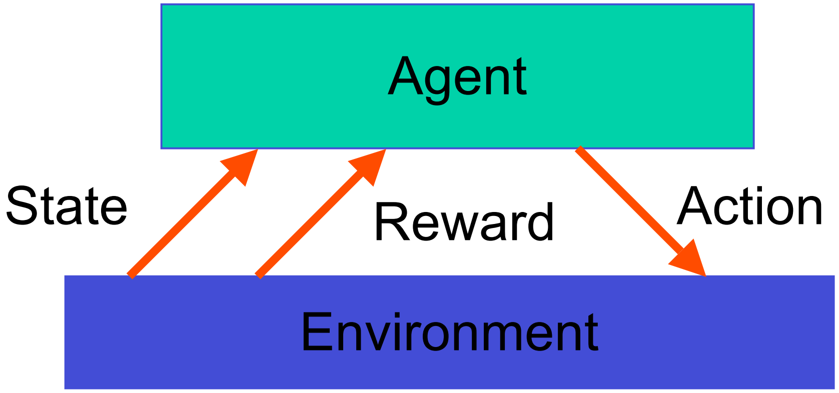


Iteration #2



Iteration #3

MDP Model



$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r1]{a1} s2 \xrightarrow[r2]{a2} s3$$

Process:

- *Observe* state s_t in S
- Choose action at in A_t
- Receive immediate reward r_t
- State changes to s_{t+1}

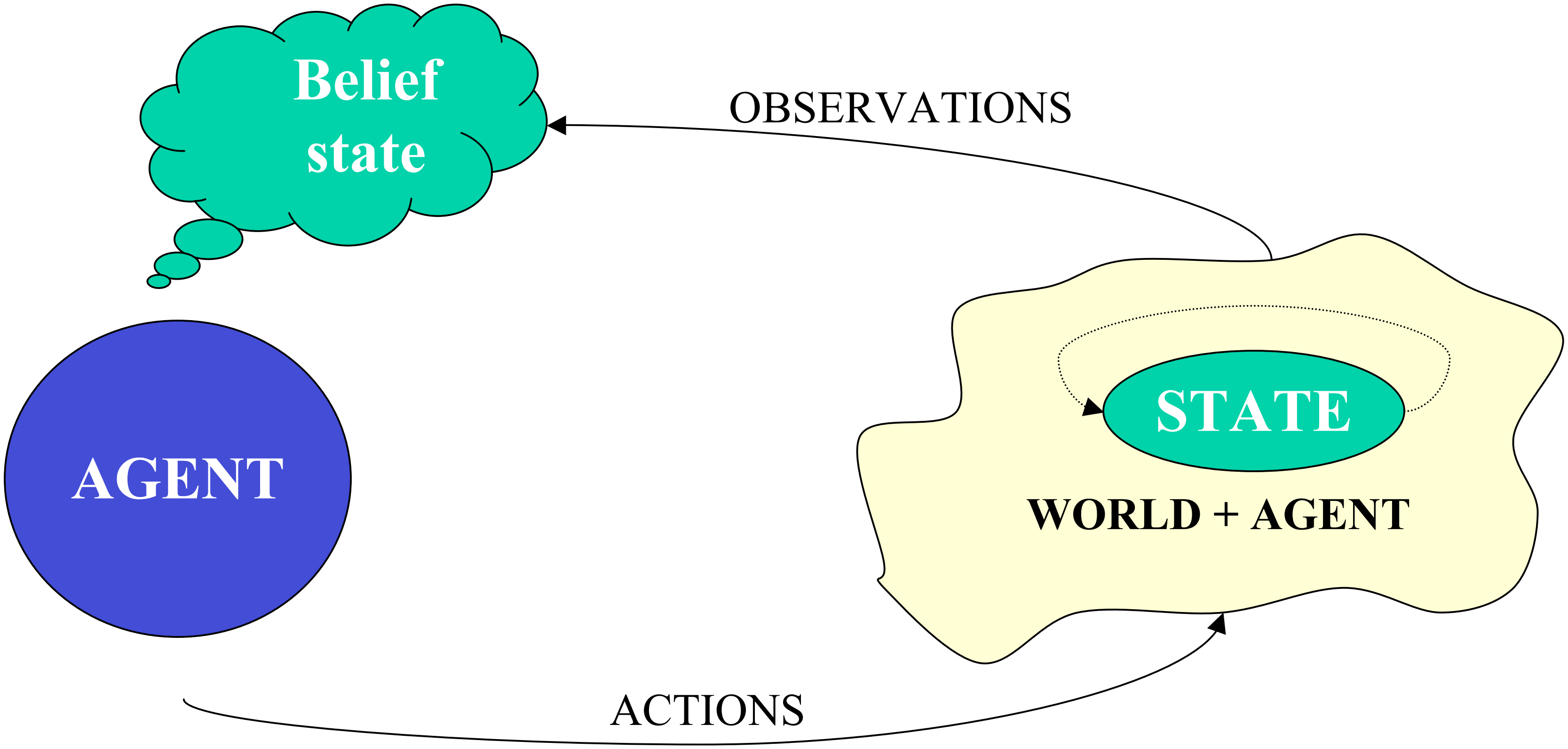
MDP Model <S, A, T, R>

POMDP: UNCERTAINTY

Case #1: Uncertainty about the action outcome

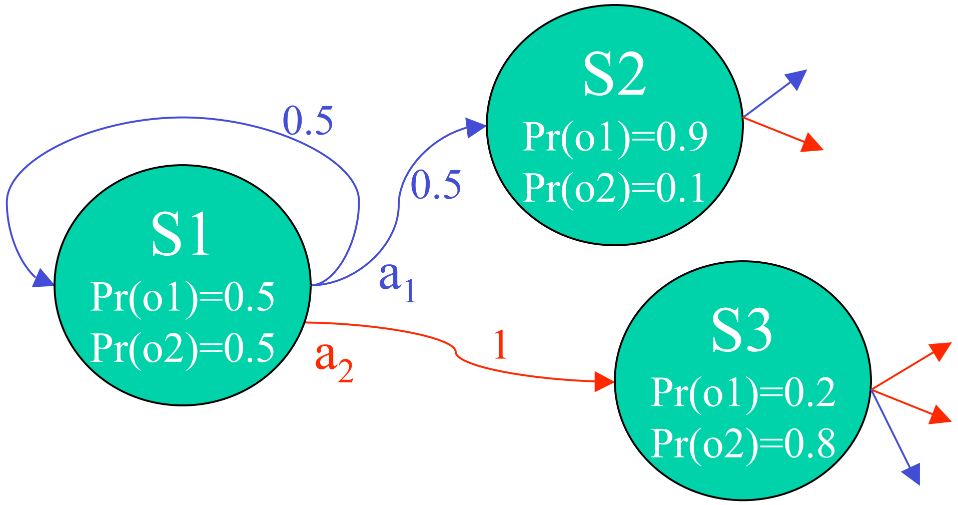
Case #2: Uncertainty about the world state due to imperfect (partial) information

A broad perspective



GOAL = Selecting appropriate actions

What are POMDPs?



Components:

Set of states: s∈S

Set of actions: a∈A

Set of observations: o∈Ω

POMDP parameters:

Initial belief: $b_0(s)$=Pr(S=s)

Belief state updating: b'(s')=Pr(s'|o, a, b)

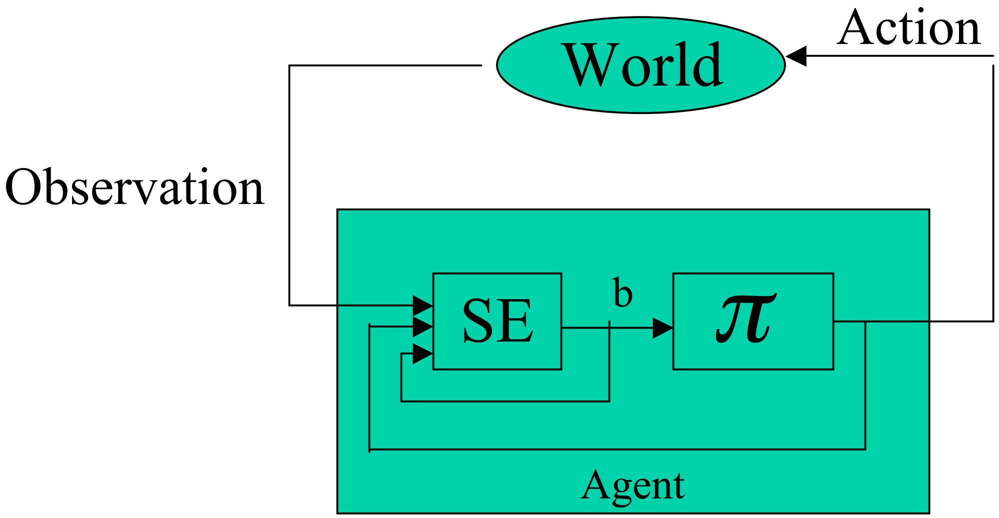
Observation probabilities: O(s',a,o)=Pr(o|s',a)

Transition probabilities: T(s,a,s')=Pr(s'|s,a)

Rewards: R(s,a)

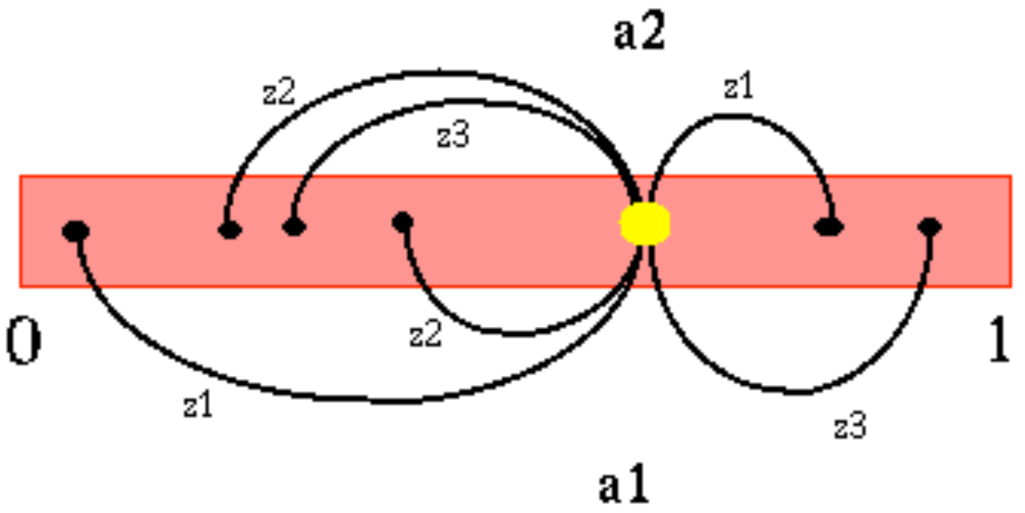
} MDP

Belief state



- Probability distributions over states of the underlying MDP
- The agent keeps an internal ***belief state,*** b, that summarizes its experience. The agent uses a ***state estimator***, SE, for updating the belief state b' based on the last action a_{t-1}, the current observation o_t, and the previous belief state b.
- Belief state is a sufficient statistic (it satisfies the Markov proerty)

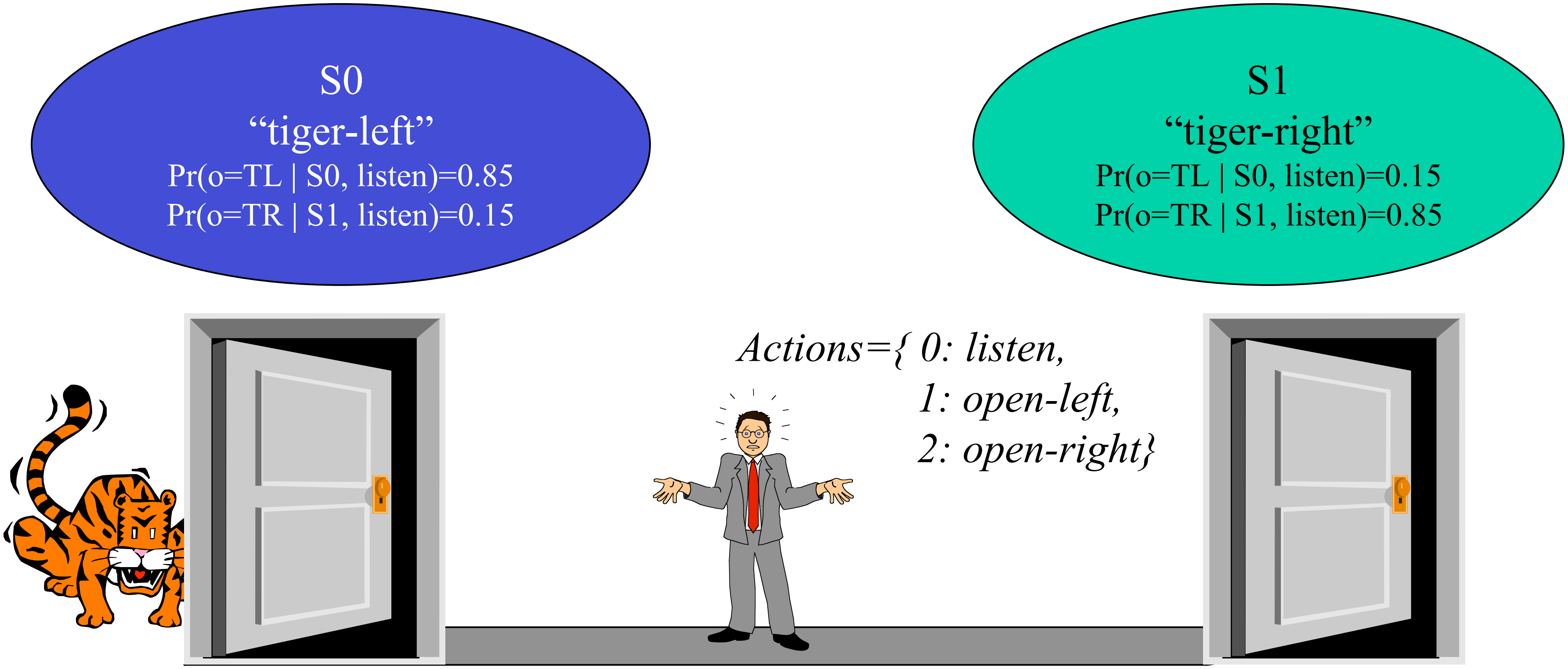
1D belief space for a 2 state POMDP with 3 possible observations



Z_i = Observations

$$b'(s_j) = \mathrm{P}(s_j \mid o,a,b) = \frac{P(o \mid s_j,a)\sum_{s_i \in S} P(s_j \mid s_i,a)\,b(s_i)}{\sum_{s_j \in S} P(o \mid s_j,a)\sum_{s_i \in S} P(s_j \mid s_i,a)\,b(s_i)}$$

A POMDP example: The tiger problem



S0
“tiger-left”
Pr(o=TL | S0, listen)=0.85
Pr(o=TR | S1, listen)=0.15

S1
“tiger-right”
Pr(o=TL | S0, listen)=0.15
Pr(o=TR | S1, listen)=0.85

Actions={ 0: listen,
1: open-left,
2: open-right}

Reward Function

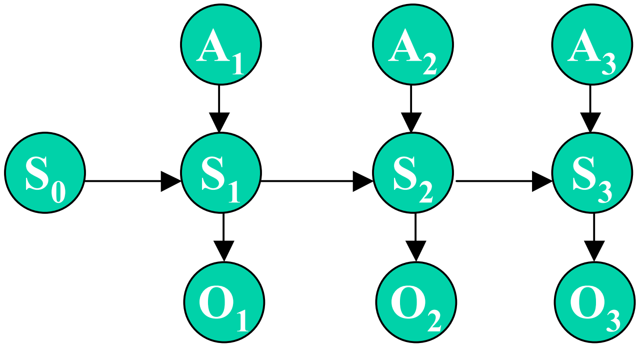
- *Penalty for wrong opening: -100*
- *Reward for correct opening: +10*
- *Cost for listening action: -1*

Observations

- *to hear the tiger on the left (TL)*
- *to hear the tiger on the right(TR)*

POMDP ≡ Continuous-Space Belief MDP

- a POMDP can be seen as a continuous-space "belief MDP", as the agent's belief is encoded through a continuous "belief state".



- We may solve this belief MDP like before using value iteration algorithm to find the optimal policy in a continuous space. However, some adaptations for this algorithm are needed.

Belief MDP

- The policy of a POMDP maps the current belief state into an action. As the belief state holds all relevant information about the past, the optimal policy of the POMDP is the the solution of (continuous-space) belief MDP.
- A belief MDP is a tuple <B, A, ρ, P>:

B = infinite set of belief states

A = finite set of actions

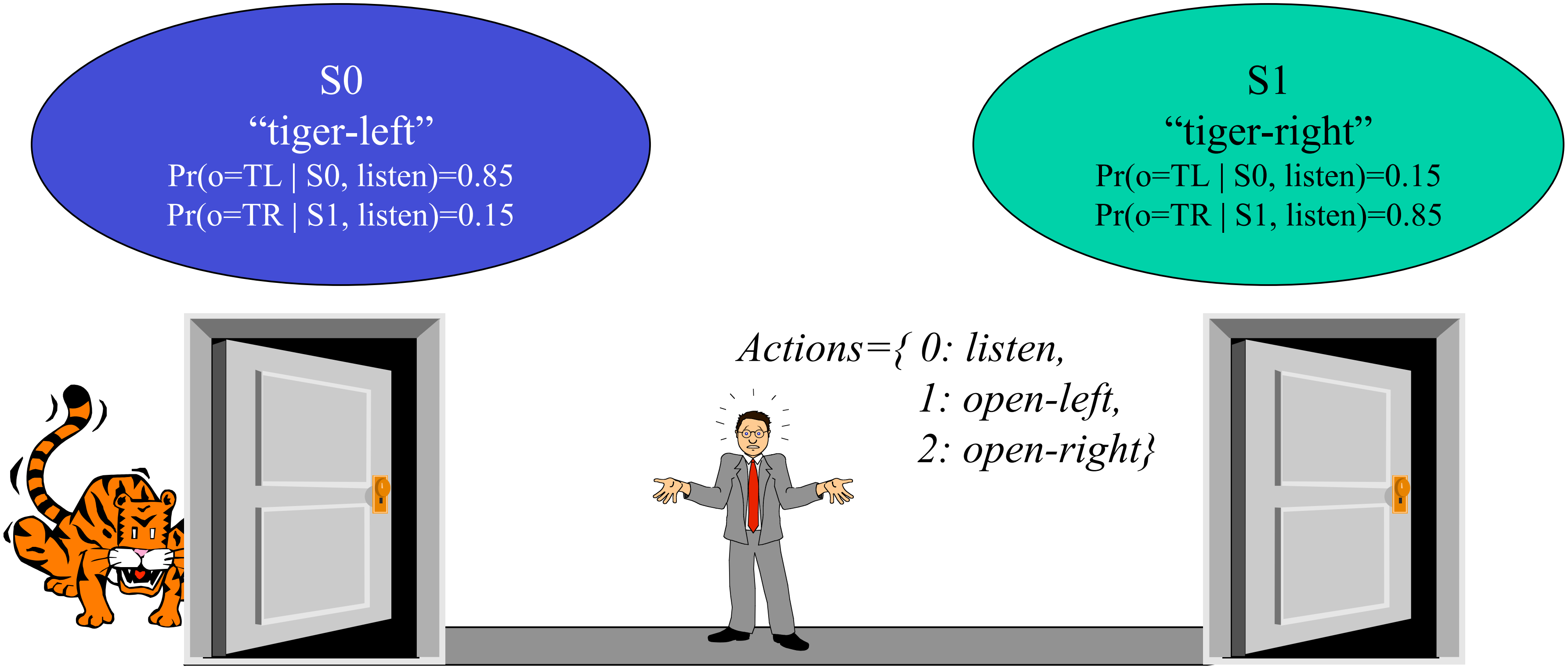
$\rho(b, a) = \sum_{s \in S} b(s) R(s, a)$ (reward function)

$P(b'|b, a) = \sum_{o \in O} P(b' | b, a, o) P(o | a, b)$ (transition function)

Where $P(b'|b, a, o) = 1$ if $SE(b, a, o) = b'$,

$P(b'|b, a, o) = 0$ otherwise;

A POMDP example: The tiger problem



S0
"tiger-left"
Pr(o=TL | S0, listen)=0.85
Pr(o=TR | S1, listen)=0.15

S1
"tiger-right"
Pr(o=TL | S0, listen)=0.15
Pr(o=TR | S1, listen)=0.85

Actions={ 0: listen,
1: open-left,
2: open-right}

Reward Function

- *Penalty for wrong opening: -100*
- *Reward for correct opening: +10*
- *Cost for listening action: -1*

Observations

- *to hear the tiger on the left (TL)*
- *to hear the tiger on the right(TR)*

Tiger Problem (Transition Probabilities)

Prob. (LISTEN)	Tiger: left	Tiger: right
Tiger: left	1.0	0.0
Tiger: right	0.0	1.0

Doesn’t change

Tiger location

Prob. (LEFT)	Tiger: left	Tiger: right
Tiger: left	0.5	0.5
Tiger: right	0.5	0.5

Problem reset

Prob. (RIGHT)	Tiger: left	Tiger: right
Tiger: left	0.5	0.5
Tiger: right	0.5	0.5

Tiger Problem (Observation Probabilities)

Prob. (LISTEN)	O: TL	O: TR
Tiger: left	0.85	0.15
Tiger: right	0.15	0.85

Prob. (LEFT)	O: TL	O: TR
Tiger: left	0.5	0.5
Tiger: right	0.5	0.5

Any observation

Without the listen action

Is uninformative

Prob. (LEFT)	O: TL	O: TR
Tiger: left	0.5	0.5
Tiger: right	0.5	0.5

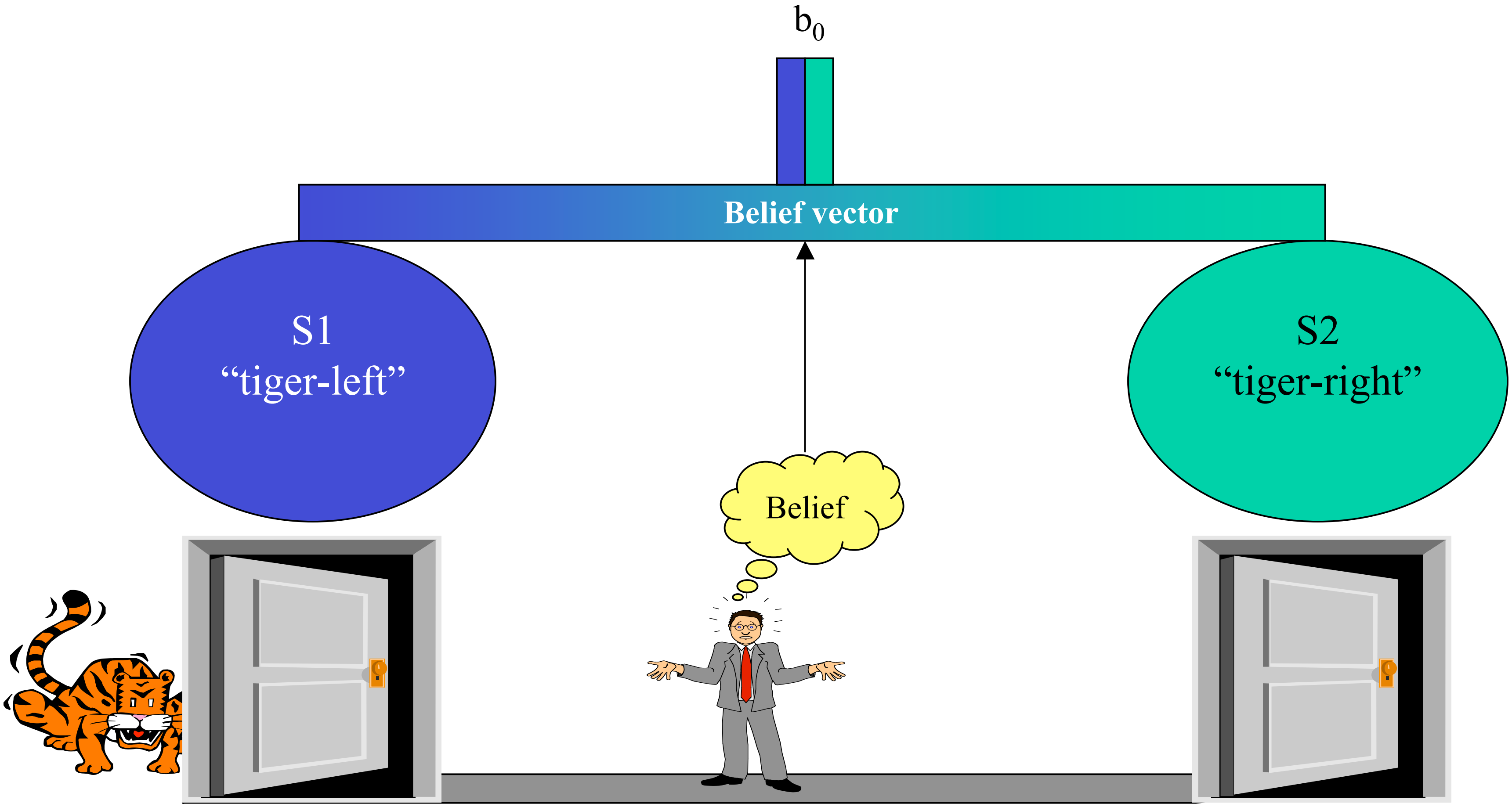
Tiger Problem (Immediate Rewards)

Reward (LISTEN)	
Tiger: left	-1
Tiger: right	-1

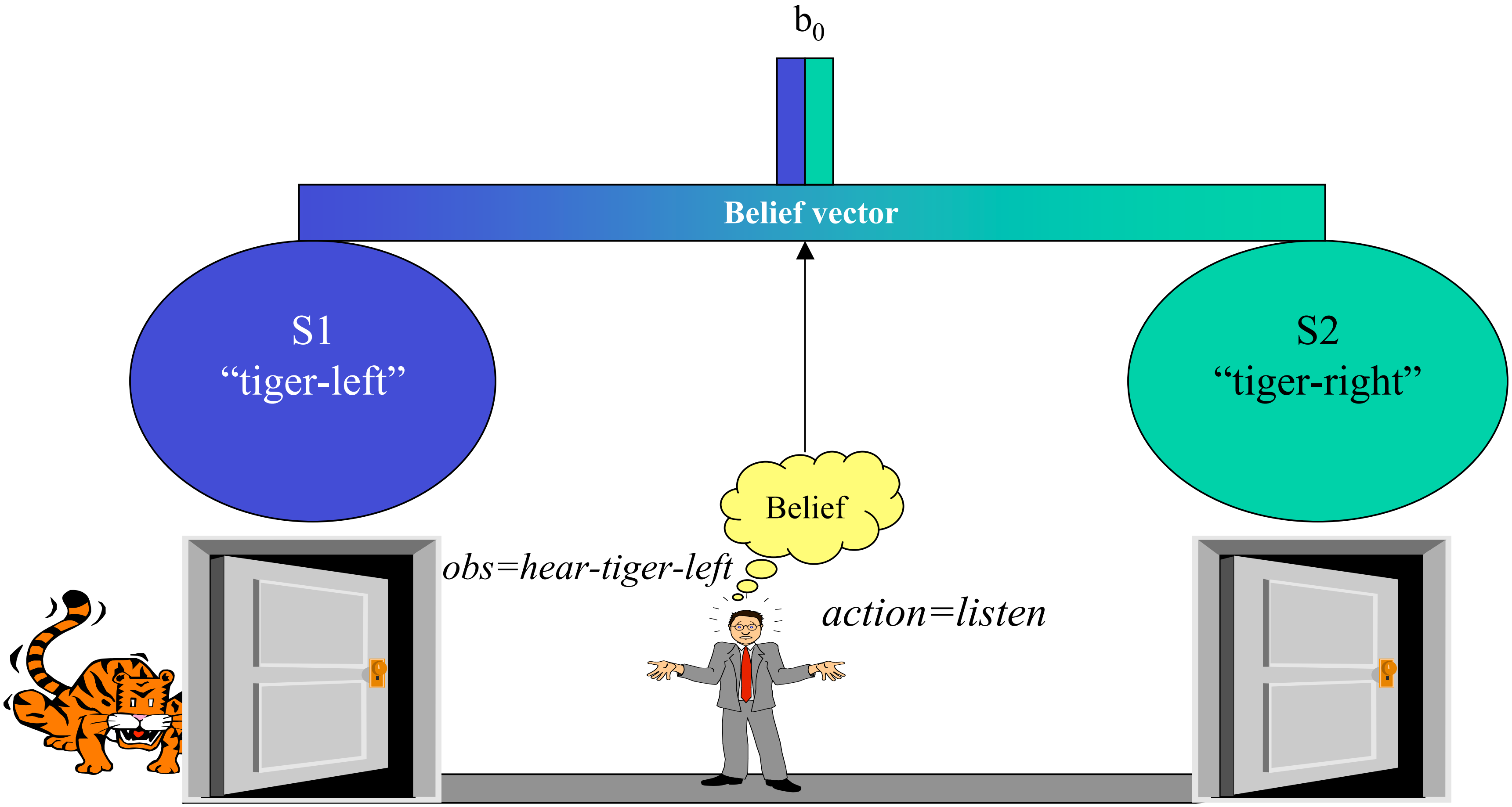
Reward (LEFT)	
Tiger: left	-100
Tiger: right	+10

Reward (RIGHT)	
Tiger: left	+10
Tiger: right	-100

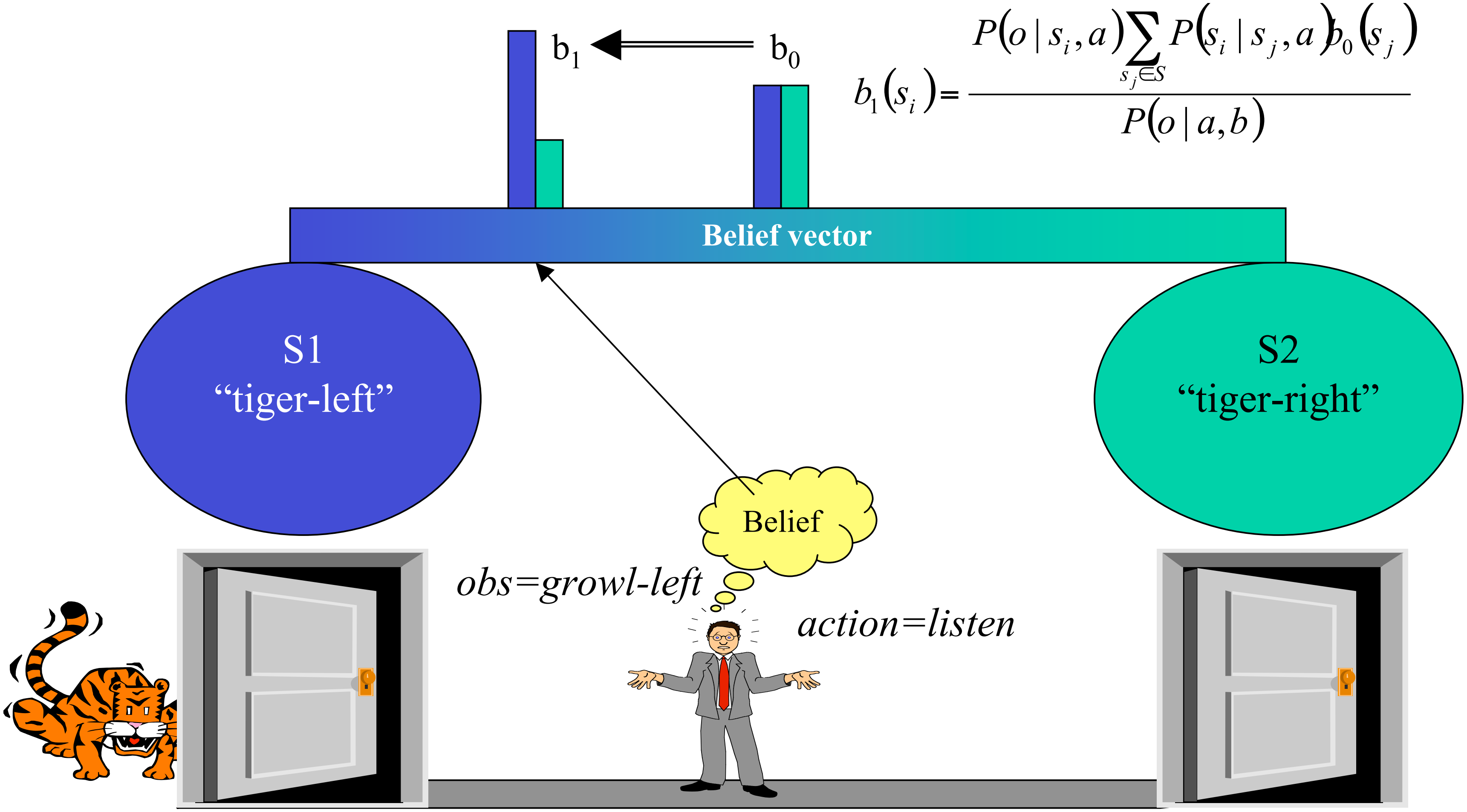
The tiger problem: State tracking



The tiger problem: State tracking



The tiger problem: State tracking

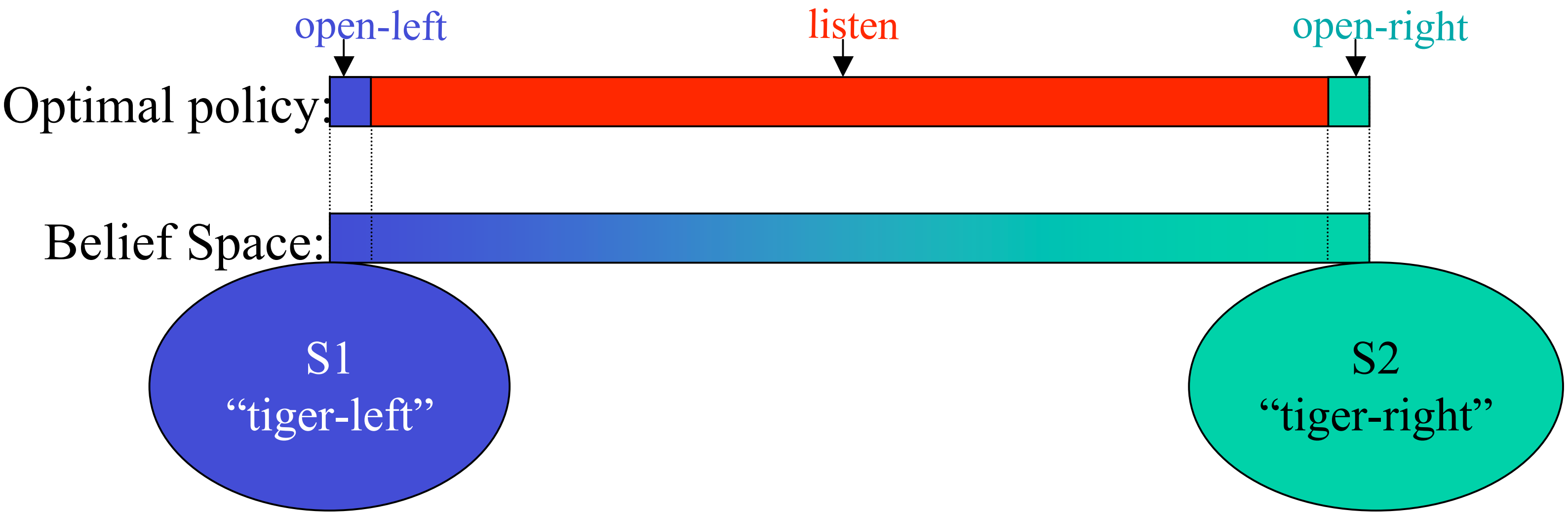


$$b_1(s_i) = \frac{P(o \mid s_i, a) \sum_{s_j \in S} P(s_i \mid s_j, a)\, b_0(s_j)}{P(o \mid a, b)}$$

Tiger Example Optimal Policy t=1

- Optimal Policy for t=1

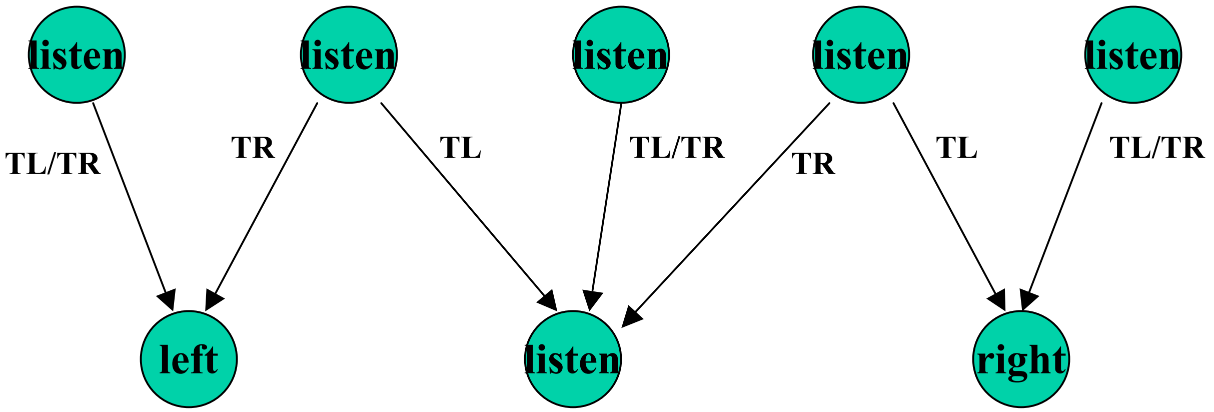
$\alpha^0(1)$=(-100.0, 10.0)	$\alpha^1(1)$=(-1.0, -1.0)	$\alpha^0(1)$=(10.0, -100.0)
left	**listen**	**right**
[0.00, 0.10]	[0.10, 0.90]	[0.90, 1.00]



Tiger Example Optimal Policy for t=2

- For t=2

[0.00, 0.02] [0.02, 0.39] [0.39, 0.61] [0.61, 0.98] [0.98, 1.00]



Can we solving this Belief MDP?

- The Bellman equation for this belief MDP is

$$V^*(b) = \max_{a \in A}\left[\sum_{s \in S} b(s) R(s,a) + \gamma \sum_{o \in O} \Pr(o \mid b, a) V^*\left(b_o^a\right)\right]$$

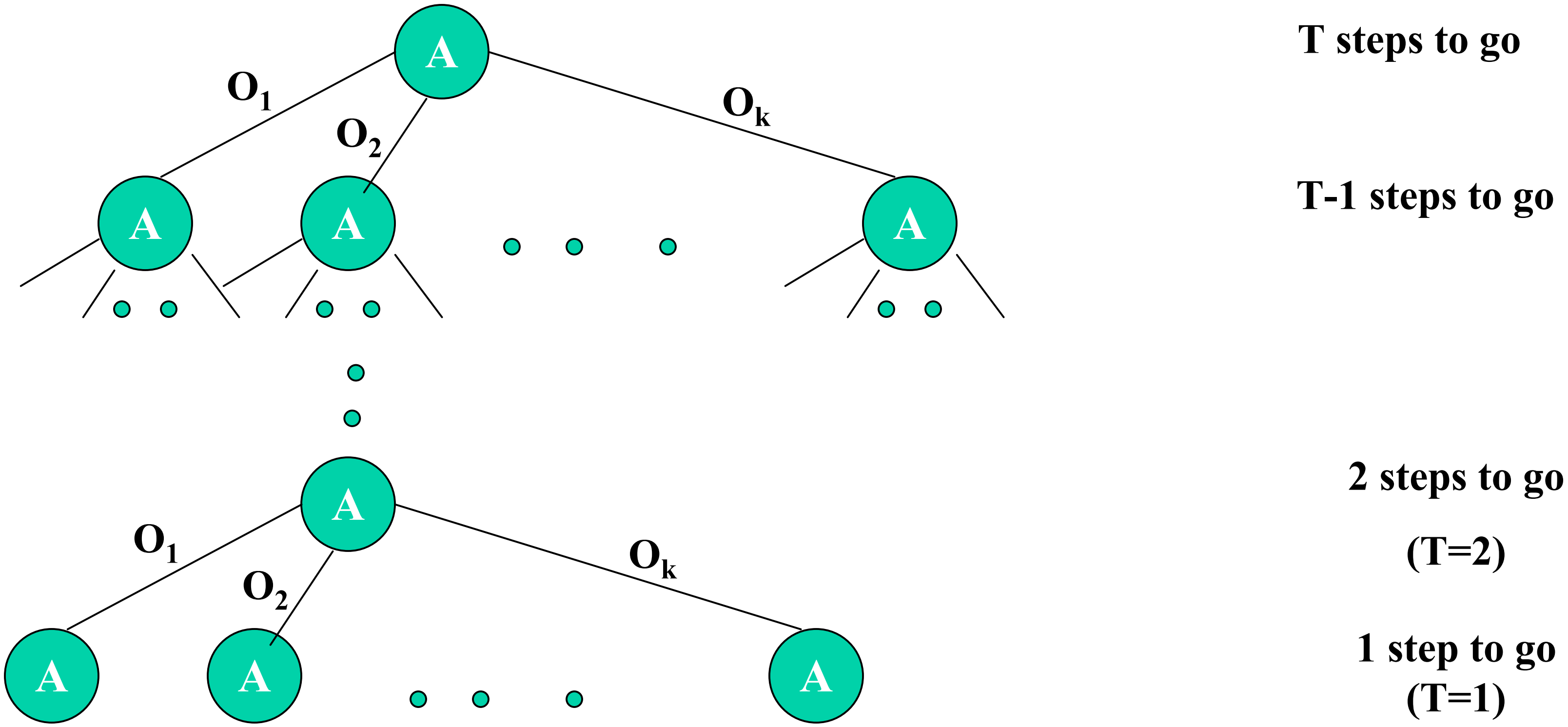
- In general case: Very Hard to solve continuous space MDPs. Unfortunately, DP updates cannot be carried out because there are uncountably many of belief states. One cannot enumerate every equation of value function. To conduct DP steps implicitly, we need to discuss the properties of value functions.

- Some special properties can be exploited first to simplify this problem
  - Policy Tree
  - Piecewise linear and convex property

- And then find an approximation algorithm to construct the optimal t-step discounted value function over belief space using value iteration…

Policy Tree

- With one step remaining, the agent must take a single action. With 2 steps to go, it takes an action, make an observation, and makes the final action. In general, an agent t-step policy can be represented as a policy tree.



T steps to go

T-1 steps to go

2 steps to go

(T=2)

1 step to go
(T=1)

Value Function for policy tree p

- If p is one-step policy tree, then the value of executing that action in state s is

  Vp(s) = R(s, a(p)).

- More generally, if p is a t-step policy tree, then

  Vp(s) = R(s, a(p)) + r (Expected value of the future)

  $$= R(s, a(p)) + r \sum_{s'\in S} T(s' \mid s, a(p), s') \sum_{o_i \in \Omega} T\left(s', a(p), o_i\right) V_{o_i(p)}(s')$$

- Thus, Vp(s) can be thought as a *vector* associated with the policy trees p since its dimension is the same as the number of states. We often use notation αp to refer to this vectors.

$$\alpha_p = \left\langle V_p(s_1), V_p(s_2), \ldots, V_p(s_n) \right\rangle$$

Value Function Over Belief Space

- As the exact world cannot be observed, the agent must compute an expectation over world states of executing policy tree p from belief state b:

  Vp(b) = $\sum_{s \in S} b(s) V_p(s)$

- If we let $\alpha_p = \left\langle V_p(s_1), V_p(s_2), \ldots, V_p(s_n) \right\rangle$, then

$$V_p(b) = b\ \alpha_p$$

- To construct an optimal t-step policy, we must maximize over all t-step policy trees P:

$$V_t(b) = \max_{p \in P} b\ \alpha_p$$

- As Vp(b) is linear in b for each p∈P, Vt(b) is the upper surface of those functions. That is, Vt(b) is piecewise linear and convex.

Illustration: Piecewise Linear Value Function

- Let $V_{p1}V_{p2}$ and V_{p3} be the value functions induced by policy trees p1, p2, and p3. Each of these value functions are the form

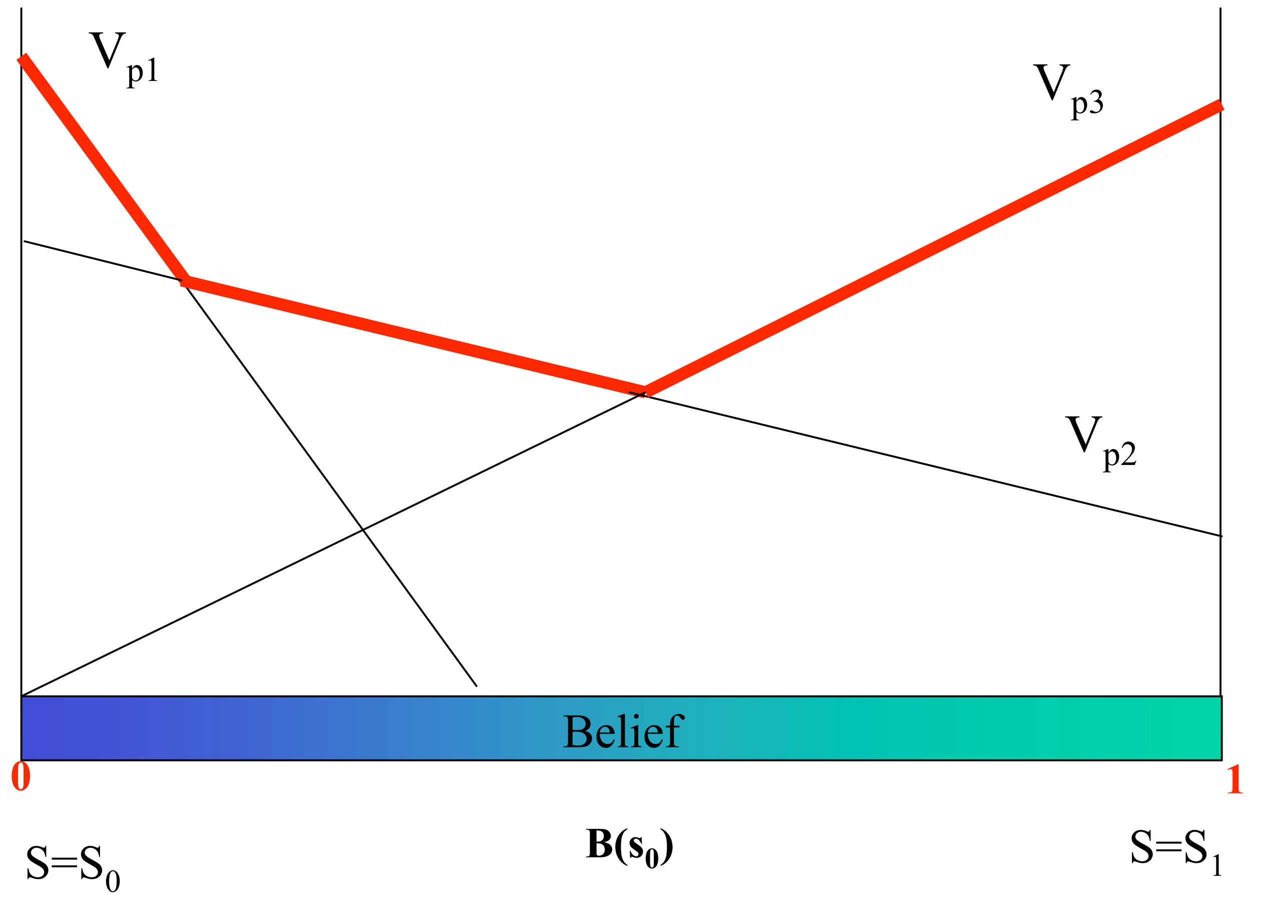
$$V_{p_i}(b) = b\ \alpha_{p_i}$$

- Which is a multi-linear function of b. Thus, for each value function can be shown as a line, plane, or hyperplane, depending on the number of states, and the optimal t-step value

- $$V_t(b) = \max_{p_i \in P} b\ \alpha_{p_i}$$

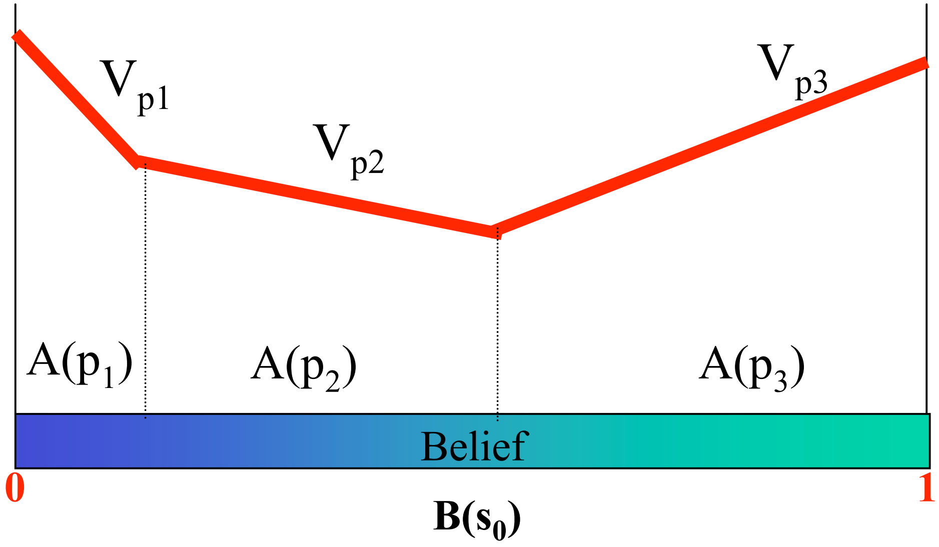
- Example– in the case of two states (b(s1) = 1- b(s2)) – be illustrated as the upper surface in two dimensions:

Picture: Optimal t-step Value Function



Optimal t-Step Policy

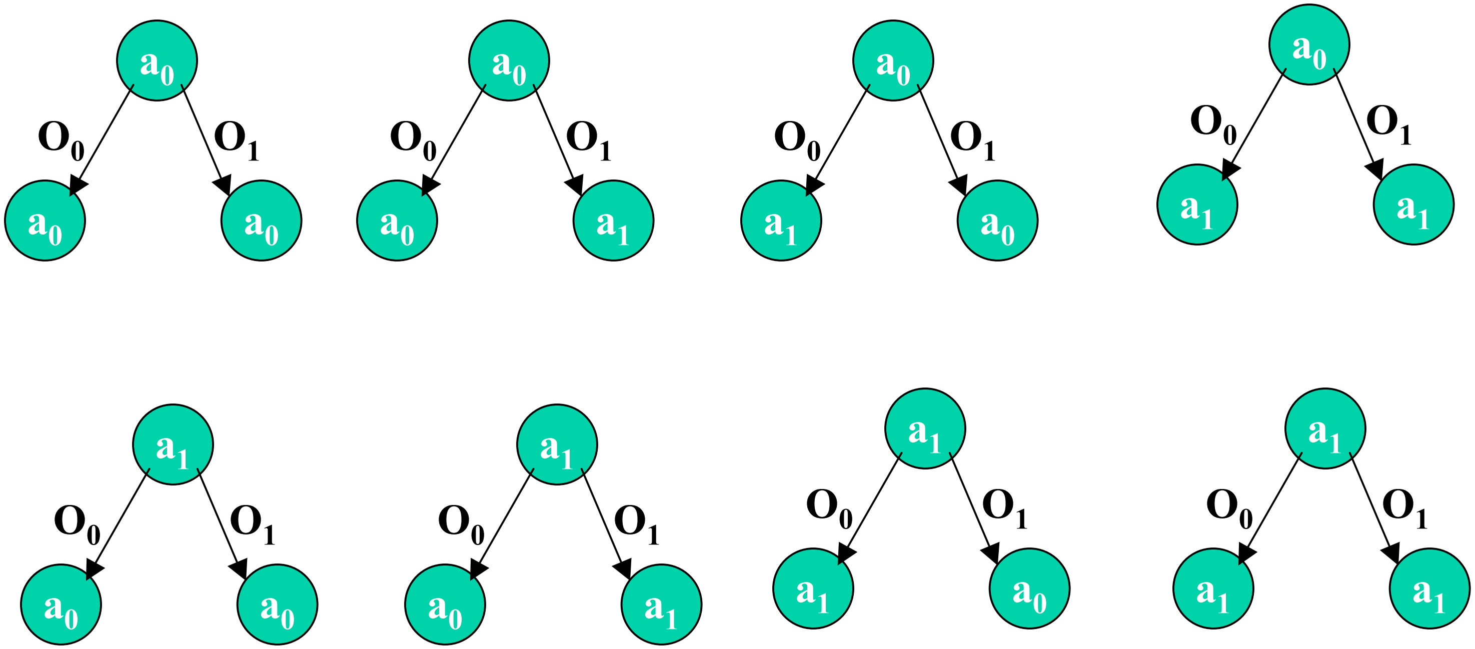
- The optimal t-step policy is determined by projecting the optimal value function back down onto the belief space.



- The projection of the optimal t-step value function yields a partition into regions, within each of which there is a single policy tree, p, such that is maximal over the entire region. The optimal action in that region a(p), the action in the root of the policy tree p.

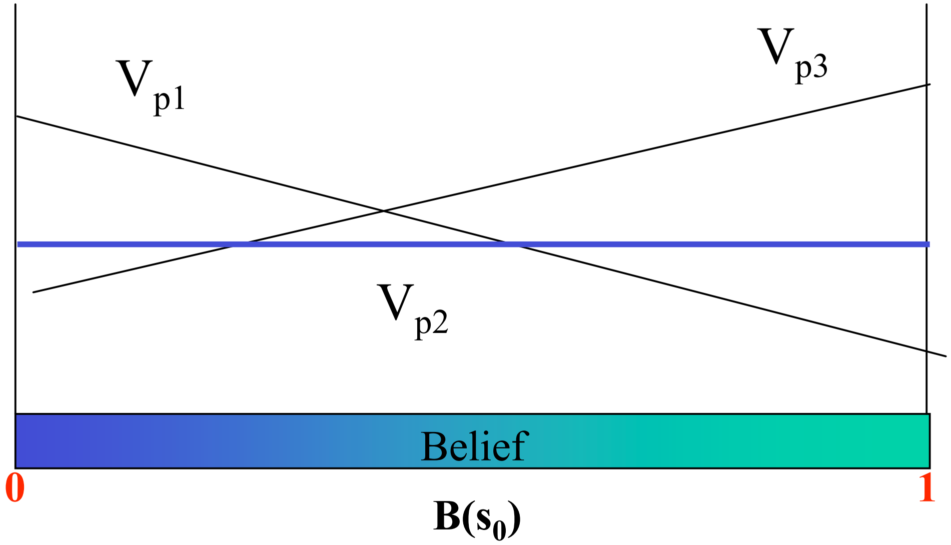
First Step of Value Iteration

- One-step policy trees are just actions: a_0 a_1
- To do a DP backup, we evaluate every possible 2-step policy tree



Pruning

- Some policy trees are dominated and are never useful



- They are pruned and not considered on the next step.
- The key idea is to prune before evaluating (Witness and Incremental Pruning do this)

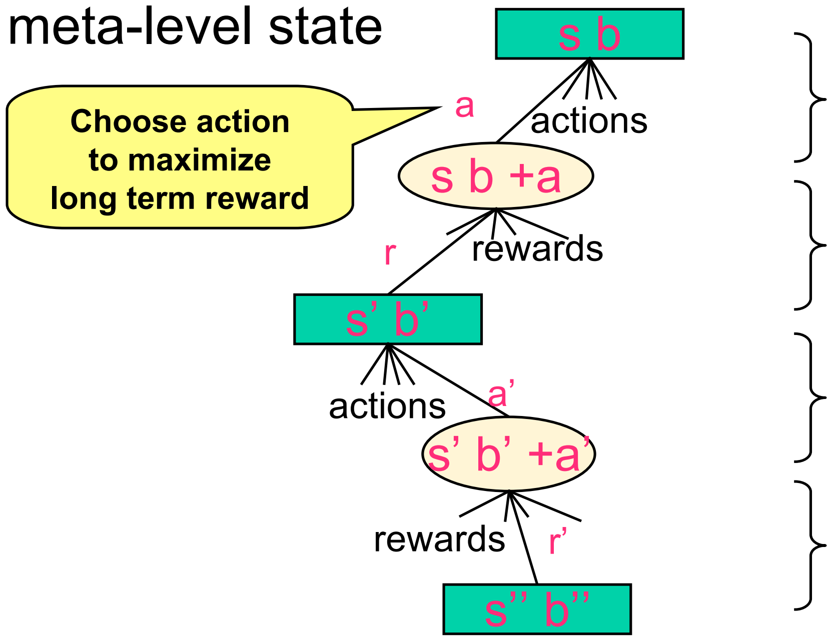
Value Iteration (Belief MDP)

- Keep doing backups until the value function doesn’t change much anymore
- In the worst case, you end up considering every possible policy tree
- But hopefully you will converge before this happens

More General POMDP: Model uncertainty

Belief state $b=P(\boldsymbol{\theta})$

meta-level state



Meta-level MDP

decision
a

outcome
r, s', b'

Meta-level Model
P(r,s'b'|s b,a)

decision
a'

outcome
r', s'', b''

Meta-level Model
P(r',s''b''|s' b',a')

Tree required to compute optimal solution grows exponentially

Multi-armed Bandit Problem

m_1

m_2

m_3

Bandit “arms”

(unknown reward probabilities)

Most complex problem for which optimal solution fully specified - *Gittins index*

-***More general than it looks***

-“Projects” with separable state spaces

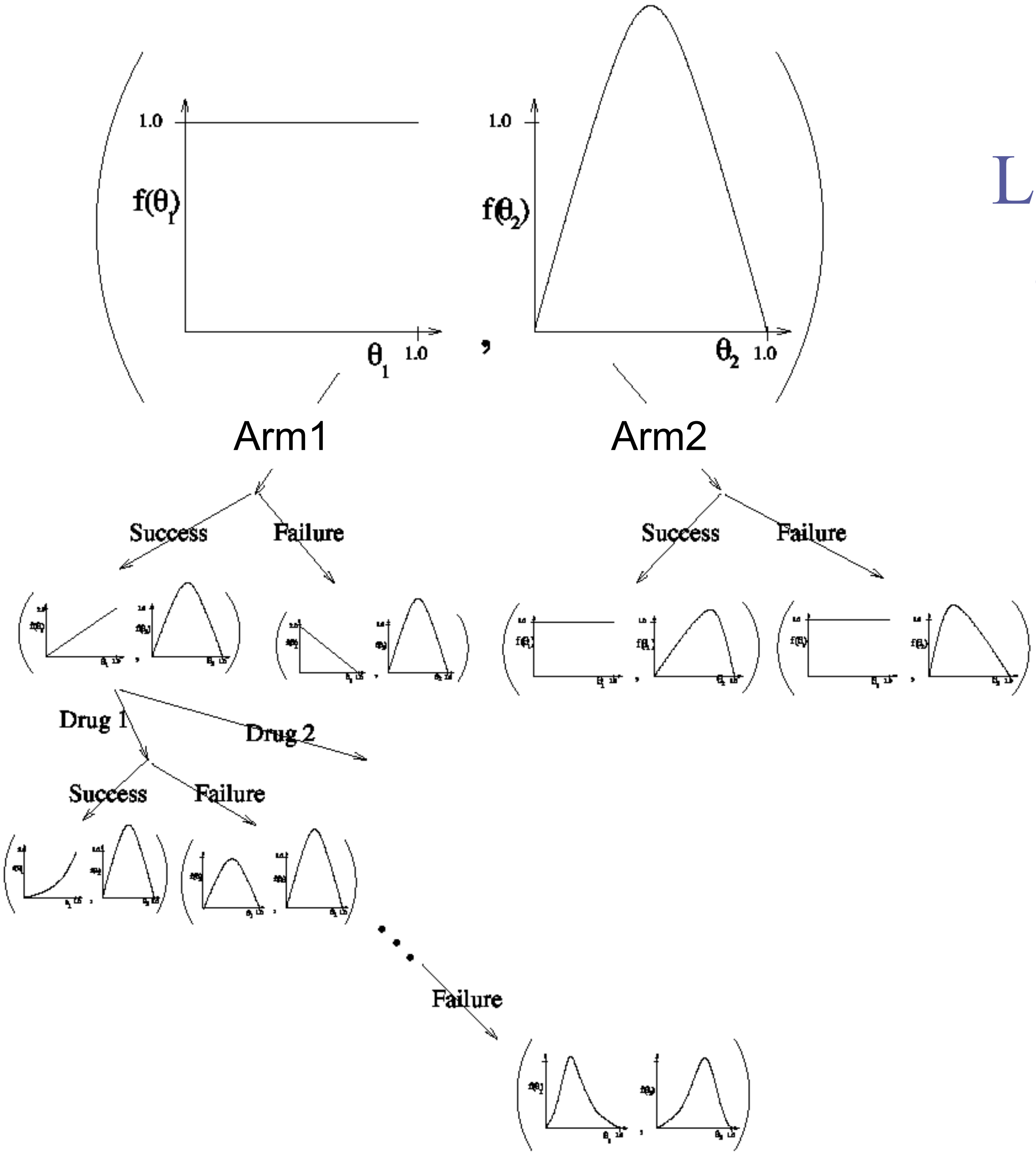
-Projects only change when worked on.

-***Key Properties of Solution***

-Each project’s value can be separately computed

-Optimal policy *permanantly* retires projects less valuable projects

Learning in a 2-armed bandit

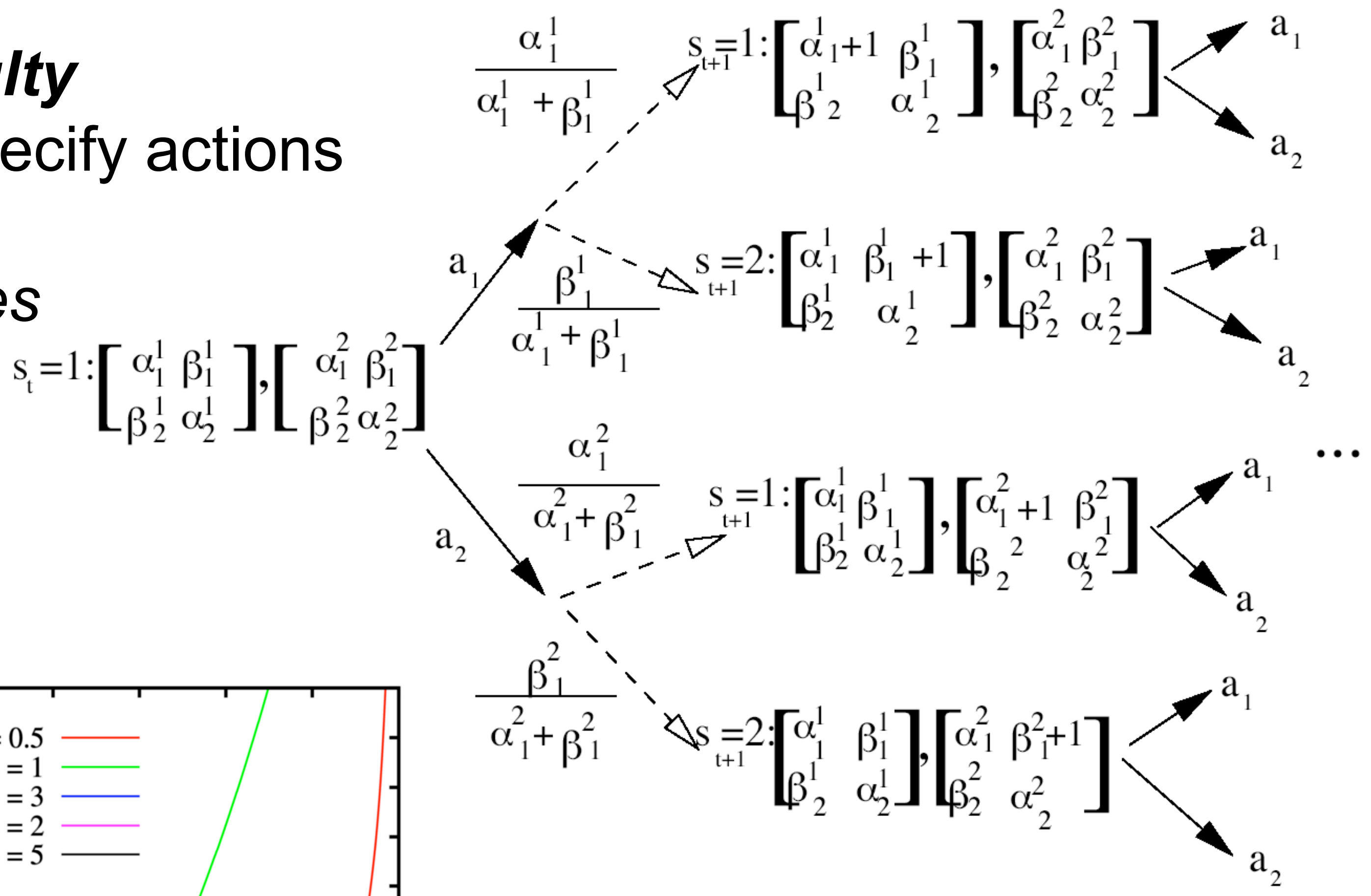


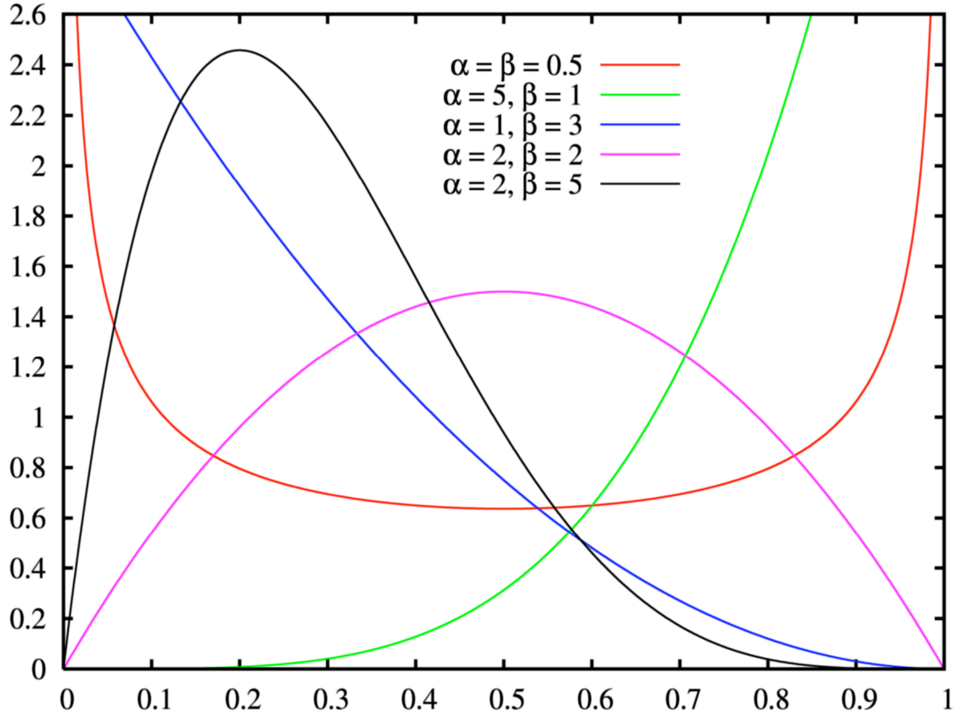
Problem difficulty

Policies must specify actions

For all possible

information states


$s_t=1: \begin{bmatrix} \alpha_1^1 & \beta_1^1 \\ \beta_2^1 & \alpha_2^1 \end{bmatrix}, \begin{bmatrix} \alpha_1^2 & \beta_1^2 \\ \beta_2^2 & \alpha_2^2 \end{bmatrix}$
a_1
$\frac{\alpha_1^1}{\alpha_1^1+\beta_1^1}$
$s_{t+1}=1: \begin{bmatrix} \alpha_1^1+1 & \beta_1^1 \\ \beta_2^1 & \alpha_2^1 \end{bmatrix}, \begin{bmatrix} \alpha_1^2 & \beta_1^2 \\ \beta_2^2 & \alpha_2^2 \end{bmatrix}$
a_1
a_2
$\frac{\beta_1^1}{\alpha_1^1+\beta_1^1}$
$s_{t+1}=2: \begin{bmatrix} \alpha_1^1 & \beta_1^1+1 \\ \beta_2^1 & \alpha_2^1 \end{bmatrix}, \begin{bmatrix} \alpha_1^2 & \beta_1^2 \\ \beta_2^2 & \alpha_2^2 \end{bmatrix}$
a_1
a_2
a_2
$\frac{\alpha_1^2}{\alpha_1^2+\beta_1^2}$
$s_{t+1}=1: \begin{bmatrix} \alpha_1^1 & \beta_1^1 \\ \beta_2^1 & \alpha_2^1 \end{bmatrix}, \begin{bmatrix} \alpha_1^2+1 & \beta_1^2 \\ \beta_2^2 & \alpha_2^2 \end{bmatrix}$
a_1
a_2
...
$\frac{\beta_1^2}{\alpha_1^2+\beta_1^2}$
$s_{t+1}=2: \begin{bmatrix} \alpha_1^1 & \beta_1^1 \\ \beta_2^1 & \alpha_2^1 \end{bmatrix}, \begin{bmatrix} \alpha_1^2 & \beta_1^2+1 \\ \beta_2^2 & \alpha_2^2 \end{bmatrix}$
a_1
a_2



Action dependent tree of Beta sufficient statistics

Value Function

Optimal value function for learning while acting for k step look-ahead

$$\begin{aligned} V_k\left((\alpha_1, \beta_1); (\alpha_2, \beta_2)\right) \quad = \quad & \max\Big\{\tfrac{\alpha_1}{\alpha_1+\beta_1}\left[1 + V_{k-1}\left((\alpha_1+1, \beta_1); (\alpha_2, \beta_2)\right)\right] \\ & +\tfrac{\beta_1}{\alpha_1+\beta_1} V_{k-1}\left((\alpha_1, \beta_1+1); (\alpha_2, \beta_2)\right), \\ & \tfrac{\alpha_2}{\alpha_2+\beta_2}\left[1 + V_{k-1}\left((\alpha_1, \beta_1); (\alpha_2+1, \beta_2)\right)\right] \\ & +\tfrac{\beta_2}{\alpha_2+\beta_2} V_{k-1}\left((\alpha_1, \beta_1); (\alpha_2, \beta_2+1)\right)\Big\}, \end{aligned}$$

The reason information is valuable is the ability to use the information to follow the optimal policy: By distinguishing the arms, future best actions have higher expected payoffs.

More reading…

- ***Planning and acting in partially Observable stochastic domains*** Leslie P. Kaelbling
- ***Optimal Policies for partially Observable Markov Decision Processes*** Anthony R.Cassandra 1995
- ***Hierarchical Methods for Planning under Uncertainty*** Joelle Pineau
- ***POMDP's tutorial in Tony's POMDP Page*** http://www.cs.brown.edu/research/ai/pomdp/