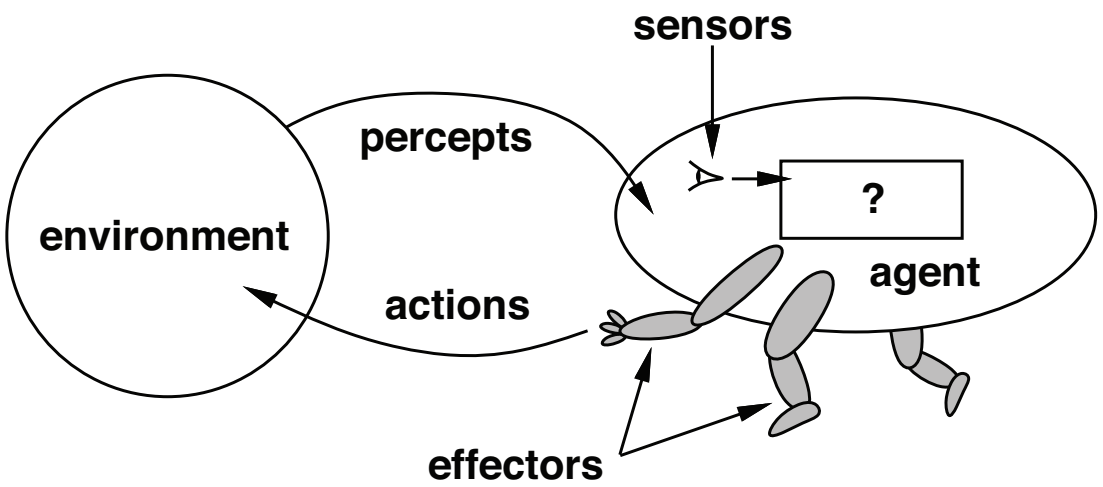
Reactive Agents

Boi Faltings

Laboratoire d'Intelligence Artificielle
boi.faltings@epfl.ch
http://moodle.epfl.ch/

Ideal rational agent



Definition (Russel & Norvig):

> For each possible percept sequence, an ideal rational agent should do whatever action is expected to maximize its performance measure, on the basis of the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Examples of rational agents

Agent Type	Percepts	Actions	Goals	Environment
Medical diagnosis system	Symptoms, findings, patient's answers	Questions, tests, treatments	Healthy patient, minimize costs	Patient, hospital
Satellite image analysis system	Pixels of varying intensity, color	Print a categorization of scene	Correct categorization	Images from orbiting satellite
Part-picking robot	Pixels of varying intensity	Pick up parts and sort into bins	Place parts in correct bins	Conveyor belt with parts
Refinery controller	Temperature, pressure readings	Open, close valves; adjust temperature	Maximize purity, yield, safety	Refinery
Interactive English tutor	Typed words	Print exercises, suggestions, corrections	Maximize student's score on test	Set of students

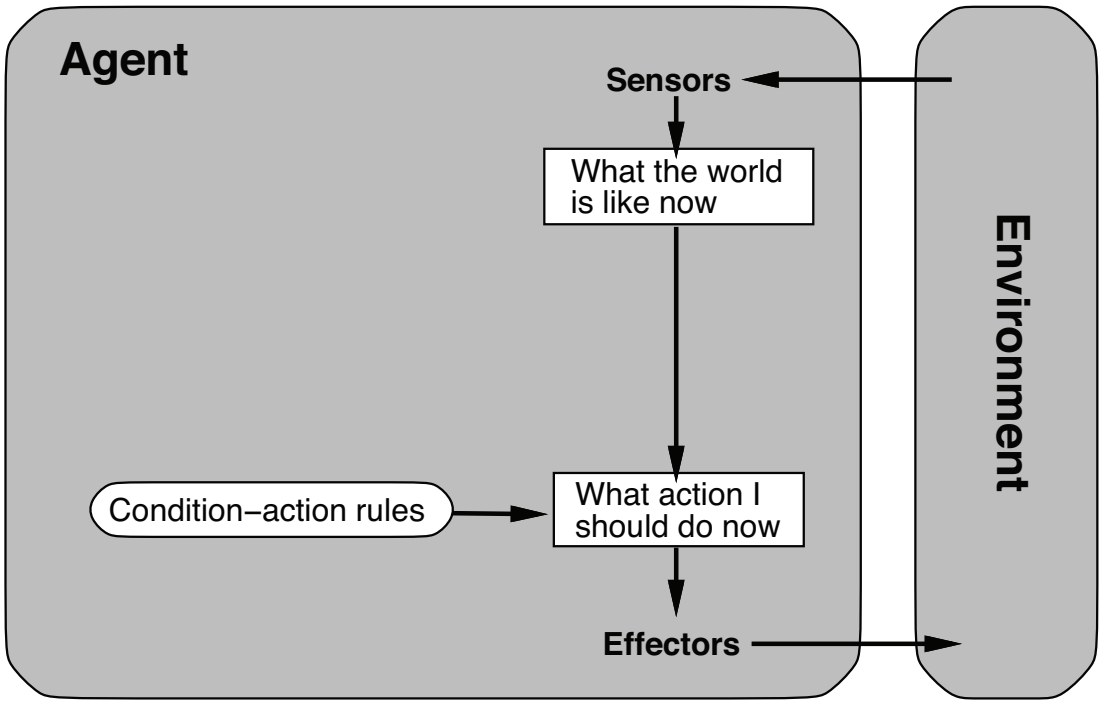
Real-time agents

Agent Type	Percepts	Actions	Goals	Environment
Taxi driver	Cameras, speedometer, GPS, sonar, microphone	Steer, accelerate, brake, talk to passenger	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers

Important:

- many solutions: optimize!
- real-time: acting too late may be worse than acting wrong
- many inputs: cannot program all cases
- learning: world evolves dynamically

Reactive agent



Purely reactive architecture

Behavior = stimulus-response table

No memory

Features:

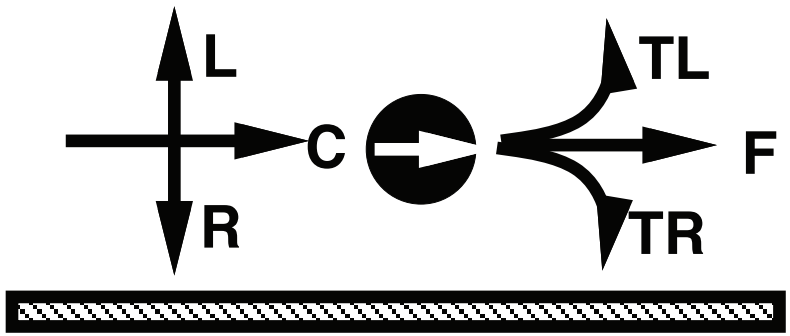
- no planning over time
- no model of the environment

Examples:

- thermostat regulates temperature
- robot following a wall
- ...

Complex behaviors from simple rules

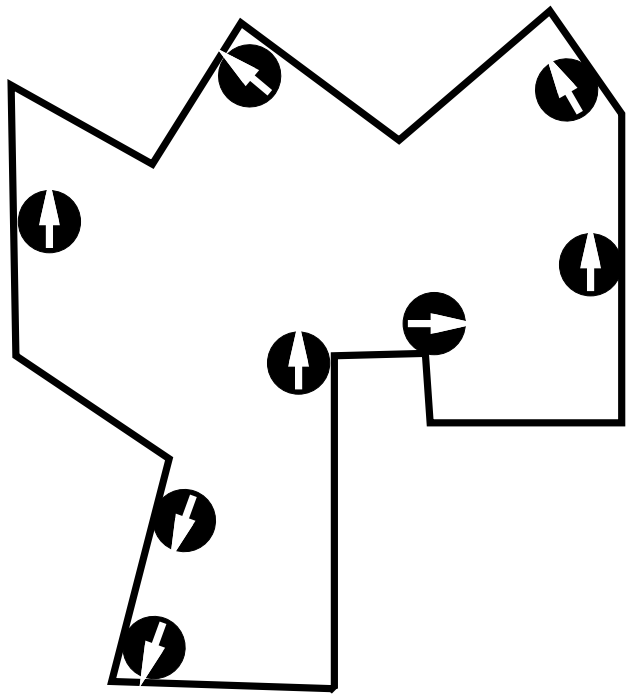
- Simple rules + complex environment $\Rightarrow$ complex behavior
- Example: robot following a wall
- Sensor = obstacle within field of view? (L/C/R/N)
  L overides C overides R
- Action = turn left/forward/turn right (TL/F/TR)



Simple rules + complex environment ⇒ complex behavior

Define behavior so that robot will follow a wall of arbitrary shape:

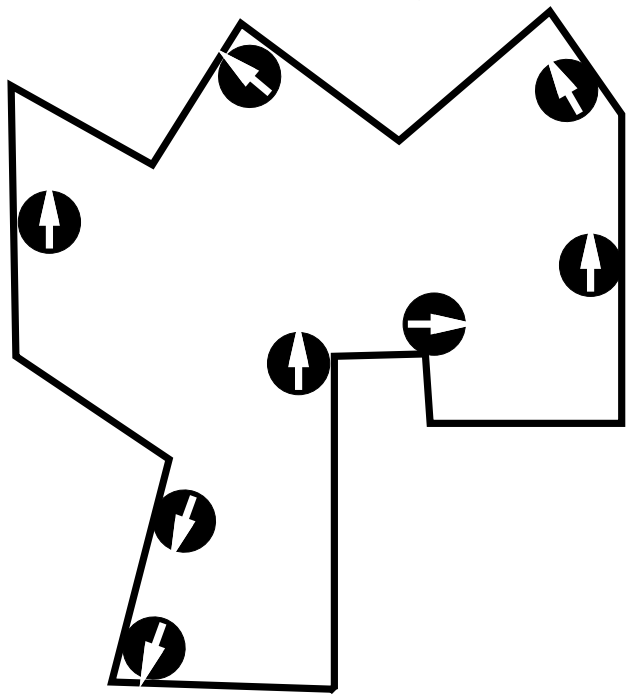
Sensor	L	C	R	N
Action	?	?	?	?



Simple rules + complex environment ⇒ complex behavior

Sensor	L	C	R	N
Action	TR	TL	F	TR

⇒ Robot will follow a wall of arbitrary shape:



Programming stateless behavior

Rational agent: maximize performance measure:

- Define a performance measure, called *reward*
- E.g.: how far is the robot from being aligned with the wall?
- Reward function depends on action and inputs $R(a, I) \rightarrow \Re$
- For each combination of inputs, choose the action that maximizes the reward function

Properties

- Table-lookup is instantaneous $\Rightarrow$ no problem with real-time
- Allows continuous world: behavior = f(inputs)
- Limitations:
  - conflicting behaviors
  - dependence on history

Subsumption architecture

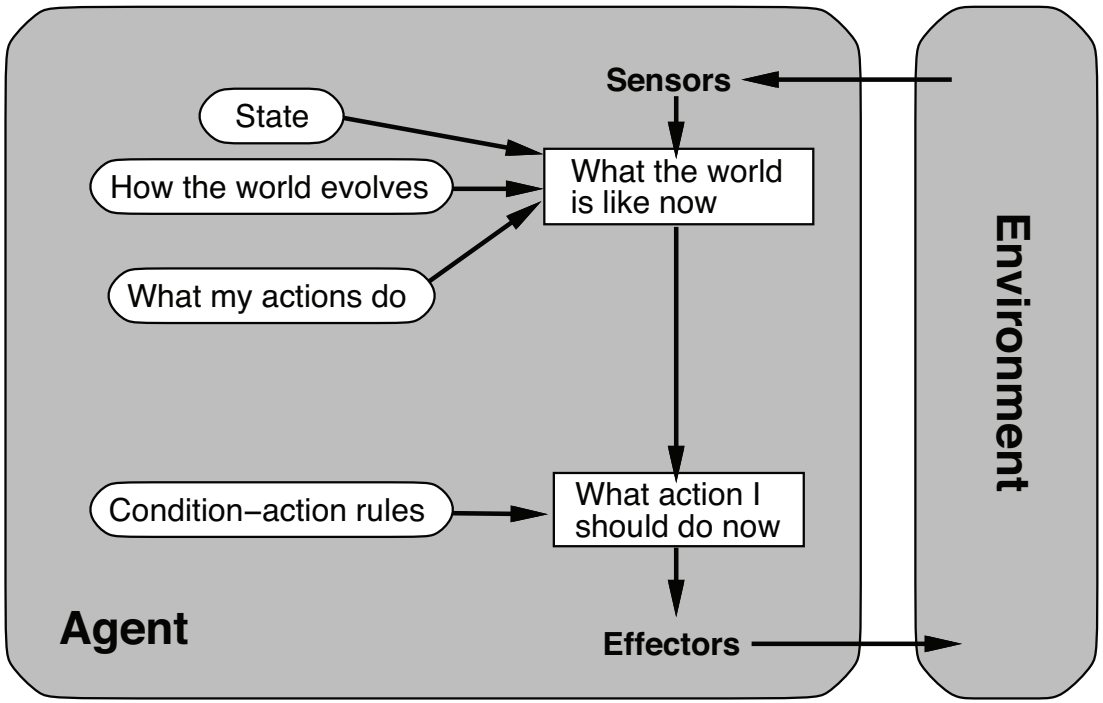
Several behaviors may conflict:

- follow wall $\Rightarrow$ go forward
- charge battery $\Rightarrow$ stop here

Subsumption architecture:

- behaviors turned on/off
- priorities decide on who wins

Agent with internal state



Behaviors with state

- Generalized behavior = state-action table:

S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12
a1	a3	a7	a1	a1	a2	a5	a7	a3	a6	a4	a3

- Called *policy* ($\pi(S)$) or *strategy*
- State models history and measurements
- Agent infers state from past state, action and measurements

Programming stateful behavior

Rational agent: maximize reward

Reward depends on state and action:

Reward function: $R(s, a) -> \Re$

Action leads to another state:

Transition function: $T(s, a) -> S$

Potential for future rewards depends on next state!

Best actions depend on immediate *and* future rewards.

Decision processes

Formalizes knowledge of state transitions and rewards:

	S1	S2	S3	S4
a1	–1	1	1	2
a2	–2	2	2	–2
a3	0	–3	–2	3
a4	2	–1	4	5

Rewards

	S1	S2	S3	S4
a1	S2	S4	S1	S1
a2	S1	S3	S4	S2
a3	S2	S1	S4	S2
a4	S2	S4	S1	S4

Transitions

Reward function: $R(s,a) -> \Re$

Transition function: $T(s,a) -> S$

Types of Decision Processes

Assumptions:

- State transitions are deterministic
  (drop $\Rightarrow$ Markov Decision Processes)
- Current state is known with certainty
  (drop $\Rightarrow$ Partially observable MDP)
- Transition and Payoff matrices are known
  (drop $\Rightarrow$ Q-learning)
- Sets of states and actions are fixed and finite
  (drop $\Rightarrow$ deliberative agents)

Processes $\Rightarrow$ strategies

Optimal strategy = act to maximize immediate reward:

State	s_1	s_2	s_3	s_4
Action	a_4	a_2	a_4	a_4
Reward	2	2	4	5
Next	s_2	s_3	s_1	s_4

Taking the future into account

Dynamic programming ("Bellmann backup"):

Maximize current reward + reward of next state:

State	s_1	s_2	s_3	s_4
Action	a_4	a_2	a_4	a_4
Reward	*2*	*2*	*4*	*5*
Next state	s_2	s_3	s_1	s_4
Value	*2*	*4*	*2*	*5*
Total	*4*	*6*	*6*	*10*

Optimization over time

- Maximize *average* reward:

$$\lim_{h \to \infty} E[\frac{1}{h} \sum_{t=0}^{h} R(s_t, a(s_t))]$$

- difficult to evaluate and optimize over large sum!
- Better to use a recursive formulation.

Infinite horizon

- Value of a state $V(s)$ = potential for rewards from this state onwards
- Iteration on all future states $\Rightarrow$ equilibrium:

$$\begin{aligned} V(s_i) &= R(s_i, a(s_i)) + V(T(s_i, a(s_i))) \\ a(s_i) &= argmax(R(s_i, a(s_i)) + V(T(s_i, a(s_i)))) \end{aligned}$$

- Infinite horizon: all state values are infinite
  $\Rightarrow$ equation for $V(s_i)$ has no solution!

Discounting the future

- Agent might die: future less valuable than present.

$\Rightarrow$ introduce *discount factor* $\gamma \in [0..1)$:

$$V(s_i) = R(s_i, a(s_i)) + \gamma \times V(T(s_i, a(s_i)))$$

- As long as $R(s_i, a(s_i)) \leq R_m$, total reward in bounded:

$$\sum_{i=0}^{\infty} \gamma^i R(s_i, a(s_i)) \leq \sum_{i=0}^{\infty} \gamma^i R_m = \frac{1}{1-\gamma} R_m$$

- Recurrence has a solution!

Computing $V(S)$ by value iteration

```
initialize V(S) arbitrarily
loop until good enough
    loop for s_i ∈ S
        loop for a ∈ A
            Q(s_i, a) ← R(s_i, a) + γ V(T(s_i, a))
        V(s_i) ← max_a Q(s_i, a)
```

Example ($\gamma = 0.5$)

State	s_1	s_2	s_3	s_4
$V(S)_0$	2	2	4	5
$A(S)_0$	a_4	a_2	a_4	a_4
$V(S)_1$	3	4	5	7.5
$A(S)_1$	a_4	a_2	a_2	a_4
$V(S)_2$	4	4.5	5.75	8.75
$A(S)_2$	a_4	a_2	a_2	a_4
$V(S)_3$	4.25	4.875	6.375	9.375
...	...	...	...	...
$A(S)_\infty$	a_4	a_1	a_2	a_4
$V(S)_\infty$	5	6	7	10

	S1	S2	S3	S4
a1	–1	1	1	2
a2	–2	2	2	–2
a3	0	–3	–2	3
a4	2	–1	4	5

Rewards

	S1	S2	S3	S4
a1	S2	S4	S1	S1
a2	S1	S3	S4	S2
a3	S2	S1	S4	S2
a4	S2	S4	S1	S4

Transitions

Stochastic Decision Processes

- In reality, state transitions cannot be predicted with certainty.
- *Markov* decision process (MDP):
  transition table $\Rightarrow$ *state transition function*
  $T(s, a, s') = p(s'|s, a)$ (3-dimensional matrix)
- Markov: Transition probability may not depend on earlier history
- Assumption: Observations determine state with certainty.

Solving a MDP

- Model =
  - reward function $R(s, a)$
  - state transition function $T(s, a, s') = p(s'|s, a)$ (MDP)
- Goal:

  Find a strategy π *mapping* $S \rightarrow A(S)$ *such that average reward is maximized.*

State values

- $V(s)$ gives the rewards that can be reached from state s using the optimal policy.
- Assuming infinite-horizon criterion:

$$\begin{aligned} V(s) &= \max_{\pi} E\left(\sum_{t=0}^{\infty} \gamma^t R(s, a_\pi(t))\right) \\ &= \max_{a}\left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s')\right) \end{aligned}$$

Value iteration

$V(S)$ can be computed by an iteration:

```
initialize V(S) arbitrarily
loop until good enough
    loop for s ∈ S
        loop for a ∈ A
            Q(s,a) ← R(s,a) + γ Σ_{s'∈S} T(s,a,s') V(s')
        V(s) ← max_a Q(s,a)
```

Convergence of value iteration

- Value iteration is guaranteed to converge!
- Stopping criterion: when difference between two successive iterations

$$max_{s \in S} |V'(s) - V(s)| \leq \epsilon$$

then maximum error (compared to true function V^*):

$$max_{s \in S} |V(s) - V^*(s)| \leq \frac{2\epsilon\gamma}{1-\gamma}$$

Choosing the policy

Given value function, choose the policy that maximizes the reward:

$$\pi(s) = \overset{argmax}{a} \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V(s) \right)$$

Policy iteration (Temporal Difference Learning)

Alternative to value iteration: optimize policy directly

choose an arbitrary policy π'
loop
  $\pi \leftarrow \pi'$
  compute the value function V_π of policy π:
   solve the linear equations:
    $V_\pi(S) = R(S, \pi(S)) + \gamma \sum_{S' \in \mathcal{S}} T(S, \pi(S), S') V_\pi(S')$
  improve the policy at each state:
   $\pi'(s) \leftarrow \overset{argmax}{a} \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s')\right)$
until $\pi = \pi'$

Clear stopping criterion: policy no longer changes

Learning with unknown models

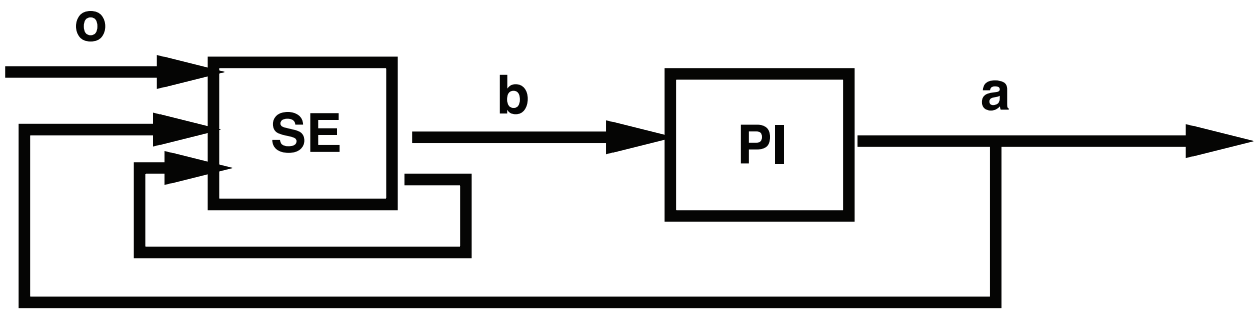
Often, model is not known a-priori
$\Rightarrow$ agent can only observe model from experience!
Approaches:

- learn the model, then the strategy
- learn both at the same time

Will be discussed later...

Partial Observation

State may not be known with certainty:
Partially observable Markov decision process (POMDP)



State Estimator (SE): transforms observations o into belief state b (a vector of probabilities for each state s).

$$SE_{s'}(b, a, o) = Pr(s'|a, o, b)$$

POMDP $\Rightarrow$ *belief MDP*

Use new transition function:

$$\tau(b, a, b') = \sum_{o|SE(b,a,o)=b'} Pr(o|a, b)$$

and reward function:

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

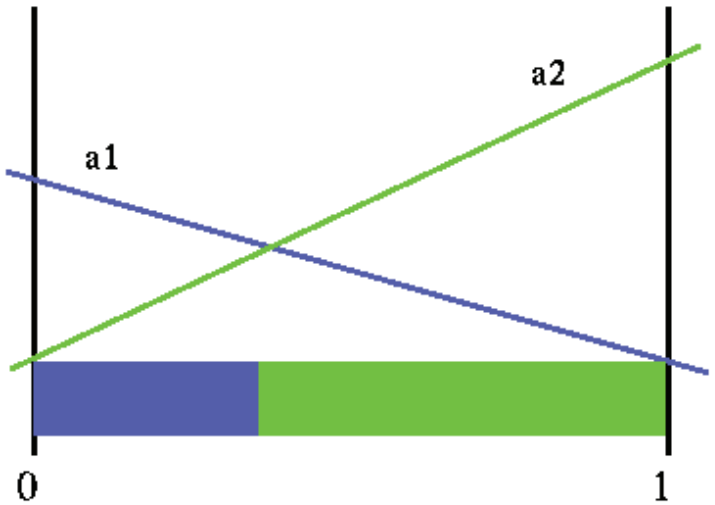
$\Rightarrow$ standard MDP equivalent to original POMDP
(same optimal policy)

Solving belief MDP

- Difficulty: for n states, space of belief states is continuous and n-dimensional.
- Discretization of probabilities in k intervals (e.g. $p \in [0..0.25), [0.25..0.5), [0.5..0.75), [0.75..1.0]$): number of vectors $= k^n$
- Discretization can be generated around most interesting points only
- Can only be applied to very small problems.

Solving belief MDP by Value Iteration

- Let belief state b $= \{Pr(s_1), Pr(s_2), ..., Pr(s_n)\}$
- $\Rightarrow \rho(b, a) = \sum_{i=1}^{n} Pr(s_i)R(s_i, a)$ is a linear function.
- Let a^* be best action in belief state b, then value function is linear around this state.
- $\Rightarrow$ value function is *piecewise* linear.



Solving belief MDP by Value Iteration (2)

Generalize value iteration backup to value function segments:

- Strategy: start with specific sample points to construct best policy, then test this policy at extreme points to see if it holds generally:
  - if another policy is better, then add it as a new segment.
  - drop segments that are never the best.
- Similar to column generation in linear programming.
- Problem: value function can have unbounded number of segments.
- Neural networks can offer an alternative for approximating the space (see later in the course).

Solving belief MDP by Policy Iteration

Policy = finite state controller = directed graph where:

- Nodes correspond to some subspace of beliefs (probability distribution of POMDP states)
- Nodes are associated with an action $\alpha(n)$.
- Arcs are associated with an observation z; different observations lead to different successor nodes.

Stochastic policy = FSC where arcs define a *distribution* of successor states.

Solving belief MDP by Policy Iteration (2)

Policy iteration:

- evaluate current policy by constructing new value function = piecewise linear function of belief state.
- improve policy by
  - adding nodes for new segments of value function.
  - removing nodes for segments that become dominated (action is never optimal).
- Stochastic policy allows to limit node explosion.

Limitations of MDPs and POMDPs

Biggest limitation of decision processes:

space of states, actions must be finite, small

Real world has many features: leads to combinatorial explosion of state space

$\Rightarrow$ partial generation of state space ($\simeq$ deliberative agents)

$\Rightarrow$ factoring the representation (logic)

Applications of MDPs and POMDPs

Small, well-defined problems:

- network routing (Dijkstra's shortest path algorithm!)
- trading agents in financial markets
- autonomous vehicles on highways
- video game playing agents
- simple continuous optimization problems

Summary

Reactive agents = predetermined action depending on measurement or estimate of state
Optimize behavior through dynamic programming, also called *reinforcement learning*:

- value iteration: slow but simple to program
- policy iteration: requires equation solving

Partially observable MDPs (POMDPs) can be transformed into MDPs and then solved.