Learning Objectives

At the end of the class you should be able to:

- build a single-stage decision network for a domain
- compute the optimal decision of a single-stage decision network
- understand MDPs and policies well (or it will come back to haunt you)
- be capable of implementing value and policy iteration

Making Decisions Under Uncertainty

What an agent should do depends on:

- Agent's **ability** — what options are available to it.
- Agent's **beliefs** — the ways the world could be, given the agent's knowledge.
  Sensing updates the agent's beliefs.
- Agent's **preferences** — what the agent wants and tradeoffs when there are risks.

Decision theory specifies how to trade off the desirability and probabilities of the possible outcomes for competing actions.

Single-stage decision

Basic decision theory applied to intelligent agents relies on the following assumptions:

- Agents know what actions they can carry out.
- The effect of each action can be described as a probability distribution over outcomes.
- An agent's preferences are expressed by utilities of outcomes.

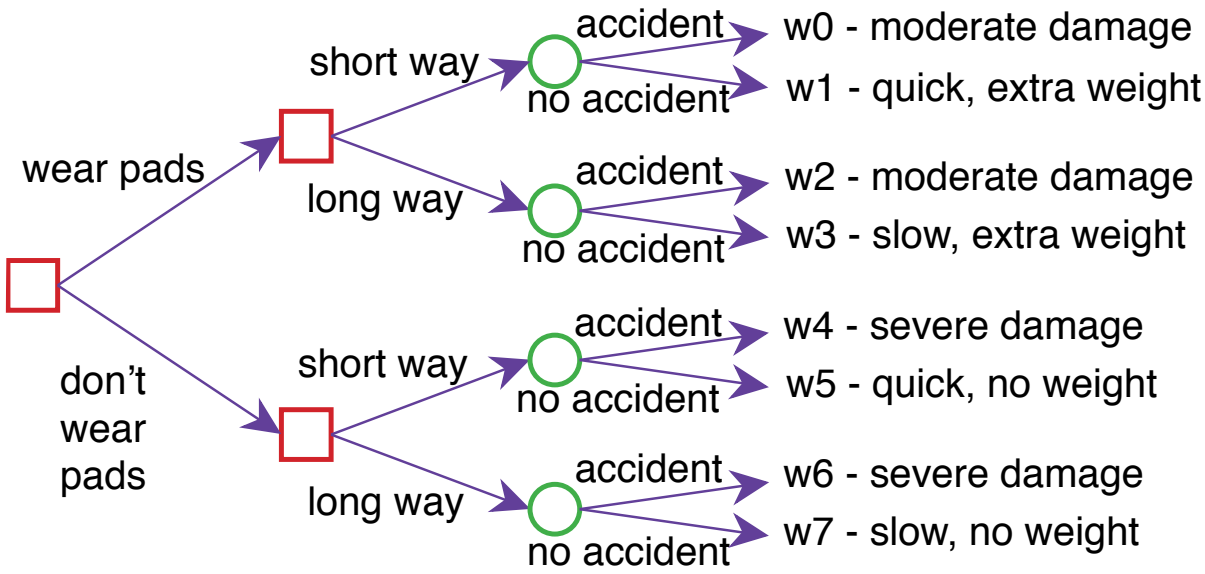
If agents only act for one step, a rational agent should choose an action with the highest expected immediate utility.

Decision Variables

- **Decision variables** are like random variables that an agent gets to choose a value for.
- **Possible world** results from a value assigned to each decision variable and each random variable.
- For each assignment of values to all decision variables, the measure of the set of worlds satisfying that assignment sum to 1.
- Probability of a proposition is undefined unless the agent conditions on the values of all decision variables.

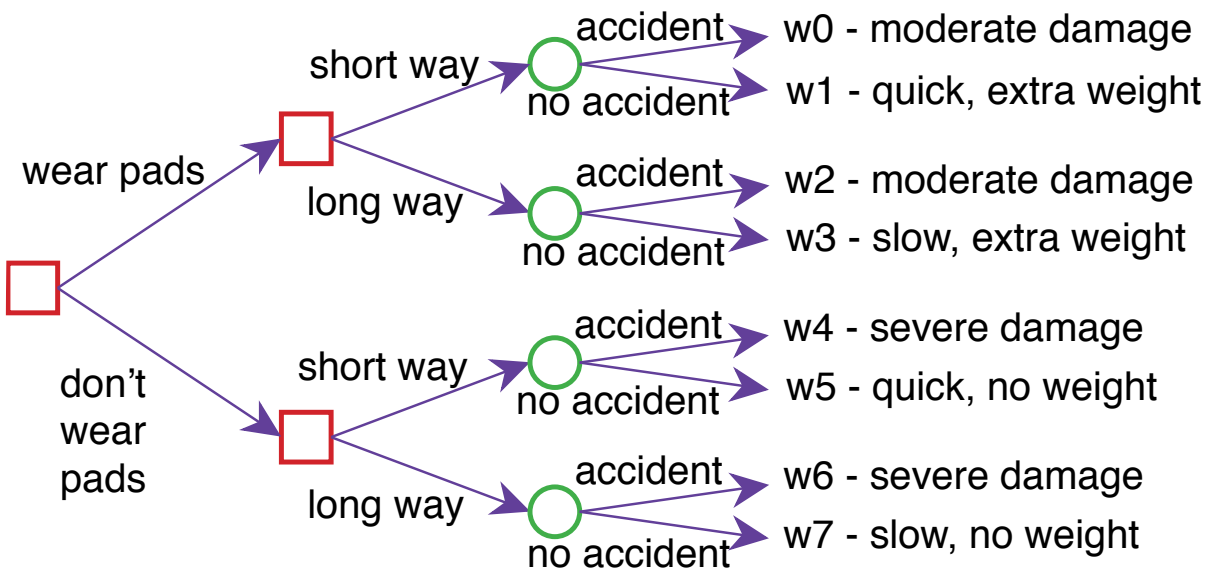
Decision Tree for Delivery Robot

- Two decision variables (squares)
- One random variable of whether there is an accident (circles). The robot can't observe before making its decision.
- Robot can to wear impact-protecting pads or not.
- Robot can go the short way past the stairs, or a long way that reduces chances of a falling accident.



Decision Tree for Delivery Robot

Decide to have utilities in the range [0,100]. First, choose best **outcome**, w_5, and give it a utility of 100. Worst outcome is w_6, so assign utility of 0. For each other world, consider lottery between w_6 and w_5. For example, w_0 may have a utility of 35, meaning indifference between w_0 and $[0.35 : w_5, 0.65 : w_6]$, which is slightly better than w_2, utility 30. w_1 may have utility of 95, because it is only slightly worse than w_5.



Expected Values

- The expected value of a function of possible worlds is its average value, weighting possible worlds by their probability.
- Suppose $f(\omega)$ is the value of function f on world ω.
  - The **expected value** of f is
$$\mathcal{E}(f) = \sum_{\omega \in \Omega} P(\omega) \times f(\omega).$$
  - The **conditional expected value** of f given e is
$$\mathcal{E}(f|e) = \sum_{\omega \models e} P(\omega|e) \times f(\omega).$$

Utility

- Utility is a measure of desirability of worlds to an agent.
- Let $u(\omega)$ be the utility of world ω to the agent.
- Simple goals can be specified by: worlds that satisfy the goal have utility 1; other worlds have utility 0.
- Often utilities are more complicated
- **for example** some function of the amount of damage to a robot, how much energy is left, what goals are achieved, and how much time it has taken.

Single-stage decisions

- Treat all the decision variables as a single composite decision variable
- In a single decision variable, the agent can choose $D = d_i$ for any $d_i \in dom(D)$.
- The **expected utility** of decision $D = d_i$ is $\mathcal{E}(u|D = d_i) = \sum_{\omega \models (D=d_i)} U(\omega)P(\omega)$
- An **optimal single decision** is a decision $D = d_{max}$ whose expected utility is maximal:

$$\mathcal{E}(u|D = d_{max}) = max_{d_i \in dom(D)} \mathcal{E}(u|D = d_i),$$

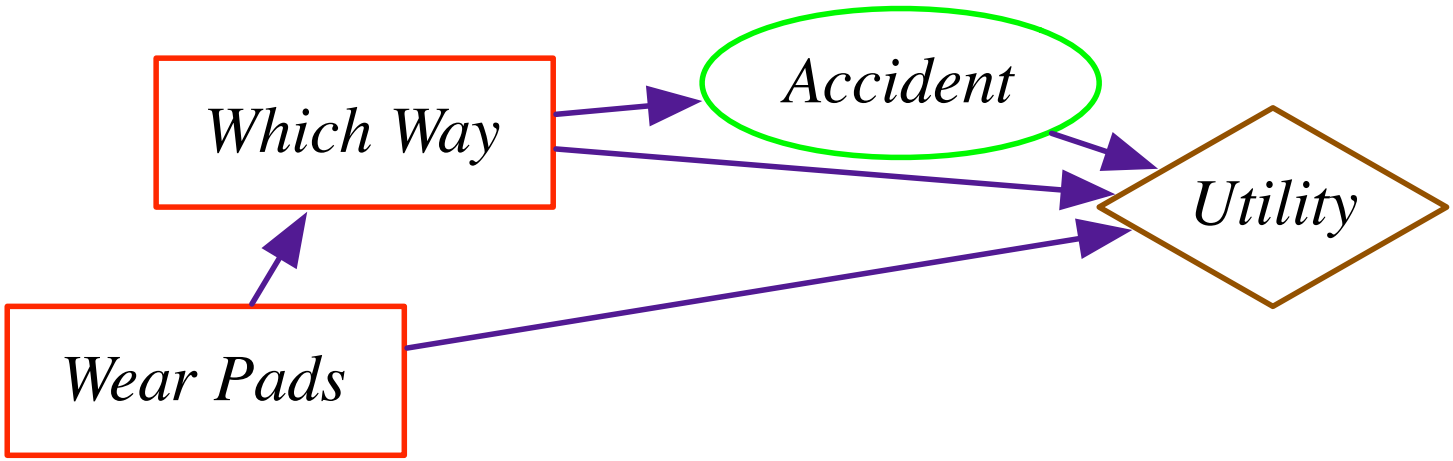
$$d_{max} = argmax_{d_i \in dom(D)} \mathcal{E}(u|D = d_i).$$

- E.g., $E(U|wear\_pads \wedge Which\_Way = short) = P(accident|wear\_pads \wedge Which\_way = short)utility(w_0) + (1 - P(accident|wear\_pads \wedge Which\_way = short))utility(w_1)$

Single-stage decision networks

Rather than state-based tree, to use more efficient feature-based representation, we extend belief networks with:

- **Decision nodes** (D) the agent chooses value from Domain, the set of possible actions (rectangle)
- **Chance nodes**, (ovals) represent random variables.
- **Utility node**, the parents are the variables on which the utility depends (diamond)



Shows explicitly which nodes affect accident

Finding an optimal decision

- Suppose the random variables are $X_1, \ldots, X_n$, and utility depends on $X_{i_1}, \ldots, X_{i_k}$, so via previous methods:

$$\begin{aligned}\mathcal{E}(u|D) &= \sum_{X_1,\ldots,X_n} P(X_1,\ldots,X_n|D) \times u(X_{i_1},\ldots,X_{i_k}) \\ &= \sum_{X_1,\ldots,X_n} \prod_{i=1}^{n} P(X_i|parents(X_i)) \times u(X_{i_1},\ldots,X_{i_k})\end{aligned}$$

To find an optimal decision:

- Create a factor for each conditional probability and for the utility
- Sum out all of the random variables
- This creates a factor on D that gives the expected utility for each D
- Choose the D with the maximum value in the factor.

Initial Factors (supplied by designer)

Which Way	Accident	Value
long	true	0.01
long	false	0.99
short	true	0.2
short	false	0.8

Probabilities

Which Way	Accident	Wear Pads	Value
long	true	true	30
long	true	false	0
long	false	true	75
long	false	false	80
short	true	true	35
short	true	false	3
short	false	true	95
short	false	false	100

Utilities

Remember: factors can be multiplied and variables can be summed out

Summing out Accident: VE


```
1: Procedure OptimizeSSDN(DN)
2:         Inputs
3:             DN a single stage decision network
4:         Output
5:             An optimal policy and the expected utility of that policy.
6:         Prune all nodes that are not ancestors of the utility node.
7:         Sum out all chance nodes.
8:         - at this stage there is a single factor F that was derived from utility
9:         Let v be the maximum value in F
10:          Let d be an assignment that gives the maximum value
11:          return d,v
```

Wear Pads	Which Way	Value
true	short	0.2* 35+ 0.8*95=83
true	long	0.01*30+0.99*75=74.55
false	short	0.2*3+0.8*100=80.6
false	long	0.01*0+0.99*80=79.2

The **optimal policy** is to take the short way and wear impact pads, with expected utility = 83.

Policies and expected utilities

- **Policy** for a single-stage decision network is an assignment of a value to each decision variable.
- Each policy has an **expected utility**, which is the conditional expected value of the utility, conditioned on the policy.
- **Optimal policy** has maximal expected utility; a policy such that no other policy has a higher expected utility.

Sequential Decisions: Decision Networks

- **flat** or modular or hierarchical
- explicit states or **features** or individuals and relations
- static or **finite stage** or indefinite stage or infinite stage
- fully observable or **partially observable**
- deterministic or **stochastic** dynamics
- goals or **complex preferences**
- **single agent** or multiple agents
- **knowledge is given** or knowledge is learned
- **perfect rationality** or bounded rationality

Sequential Decisions

- An optimal agent doesn't carry out a multi-step plan ignoring information it receives between actions.
- A more realistic scenario is where the agent: observes, acts, observes, acts, ...
- Subsequent actions can depend on what is observed. What is observed depends on previous actions.
- Often the sole reason for carrying out an action is to provide information for future actions.
  For example: diagnostic tests, spying.

Sequential decision problems

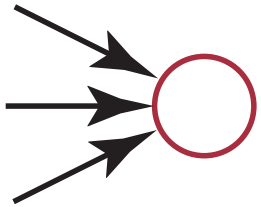
- A **sequential decision problem** consists of a sequence of decision variables $D_1, \ldots, D_n$.
- Each D_i has an **information set** of variables $parents(D_i)$, whose value will be known at the time decision D_i is made.

Decision Networks

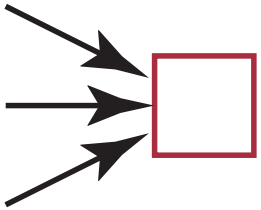
- A **decision network** is a graphical representation of a finite sequential decision problem.
- Decision networks in general extend belief networks to include decision variables and utility.
- Sequential decision networks extend single-stage decision networks by allowing both chance nodes and decision nodes to be parents of decision nodes.
- Decision network specifies what information is available when the agent has to act.
- Decision network specifies which variables the utility depends on.

Decisions Networks

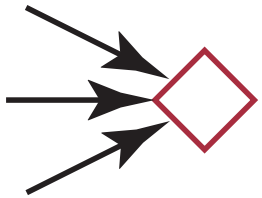
A **decision network**is a directed acyclic graph (DAG) and has 3 types of nodes:



- A **random variable** (ellipse). Arcs into the node represent probabilistic dependence.

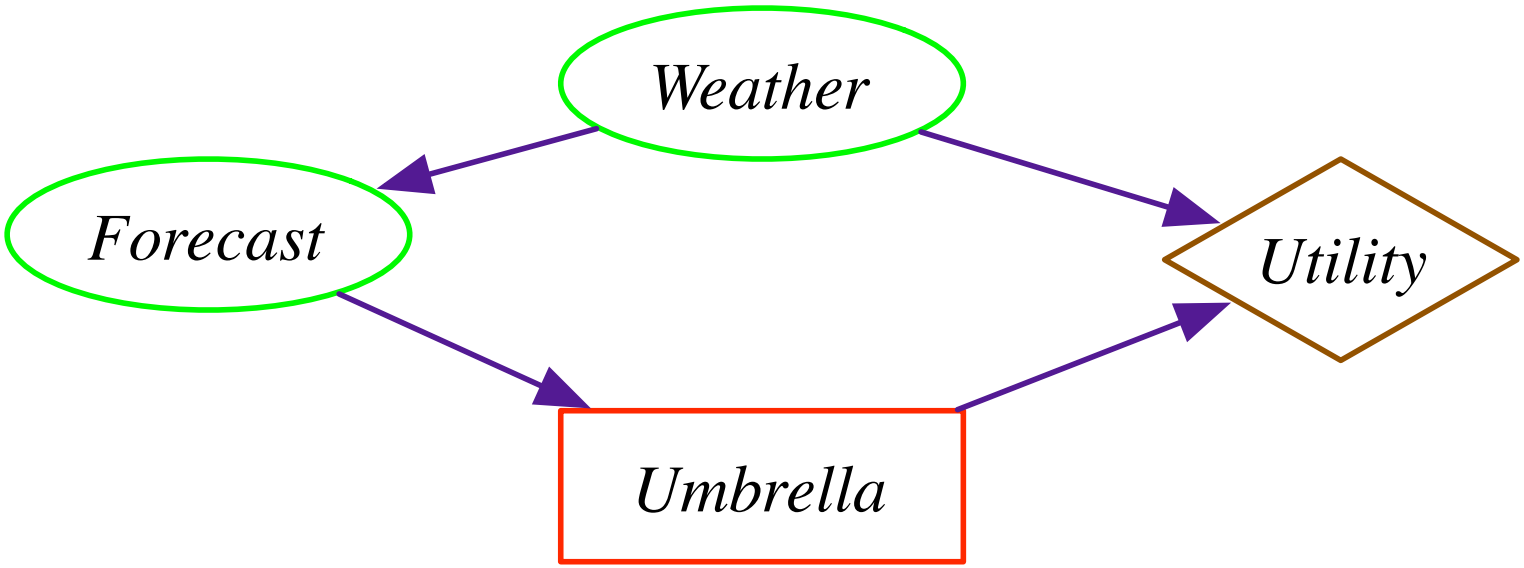


- A **decision variable** (rectangle). Arcs into the node represent information available when the decision is made.



- A **utility node** (diamond). Arcs into the node represent variables that the utility depends on.

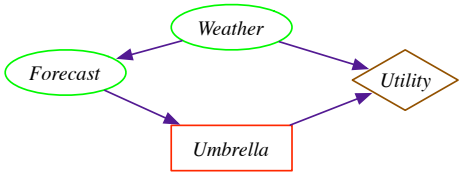
Umbrella Decision Network



You don't (usually or in this example) get to observe the weather when you have to decide whether to take your umbrella. You do get to observe the forecast.

Random variables: Weather has domain: $\{norain, rain\}$; Forecast: $\{sunny, rainy, cloudy\}$, and decision variable Umbrella has domain: $\{takeIt, leaveIt\}$. No domain associated with the utility node. Designer also must specify the probability of all random variables given their parents.

Umbrella Decision Network



Must also specify:

$P(Forecast|Weather)$

Weather	Forecast	Probability
norain	sunny	0.7
norain	cloudy	0.2
norain	rainy	0.1
rain	sunny	0.15
rain	cloudy	0.25

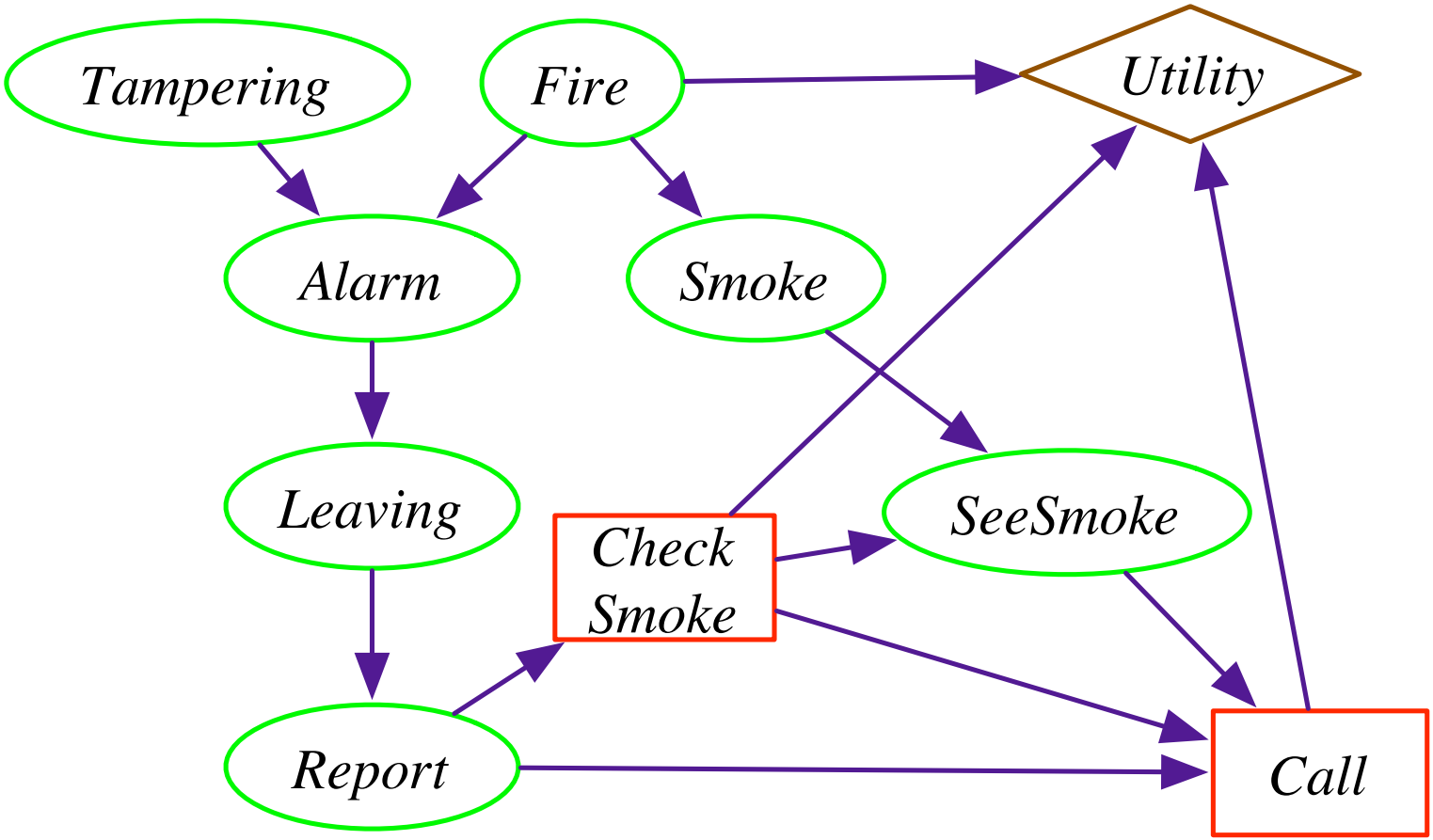
$Utility(Weather, Umbrella)$

Weather	Umbrella	Utility
norain	takeIt	20
norain	leaveIt	100
rain	takeIt	70
rain	leaveIt	0

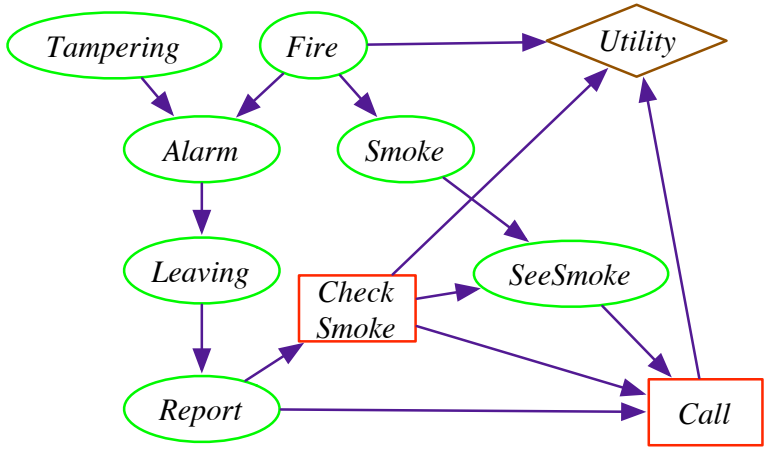
$P(weather)$

Weather	Value
norain	0.7
rain	0.3

Decision Network for the Alarm Problem



Decision Network for the Alarm Problem



In addition to belief network, designer must also specify: $P(SeeSmoke|Smoke, CheckSmoke) = 1$ and utility:

CheckSmoke	Fire	Call	Utility
yes	yes	call	-220
yes	yes	do not call	-5020
yes	no	call	-220
yes	no	do not call	-20
no	yes	call	-200
no	yes	do not call	-5000
no	no	call	-200
no	no	do not call	0

No-forgetting

A **No-forgetting decision network** is a decision network where:

- The decision nodes are totally ordered. This is the order the actions will be taken.
- All decision nodes that come before D_i are parents of decision node D_i. Thus the agent remembers its previous decisions.
- Any parent of a decision node is a parent of subsequent decision nodes. Thus the agent remembers its previous observations and decisions.

What should an agent do?

- What an agent should do at any time depends on what it will do in the future.
- What an agent does in the future depends on what it did before.

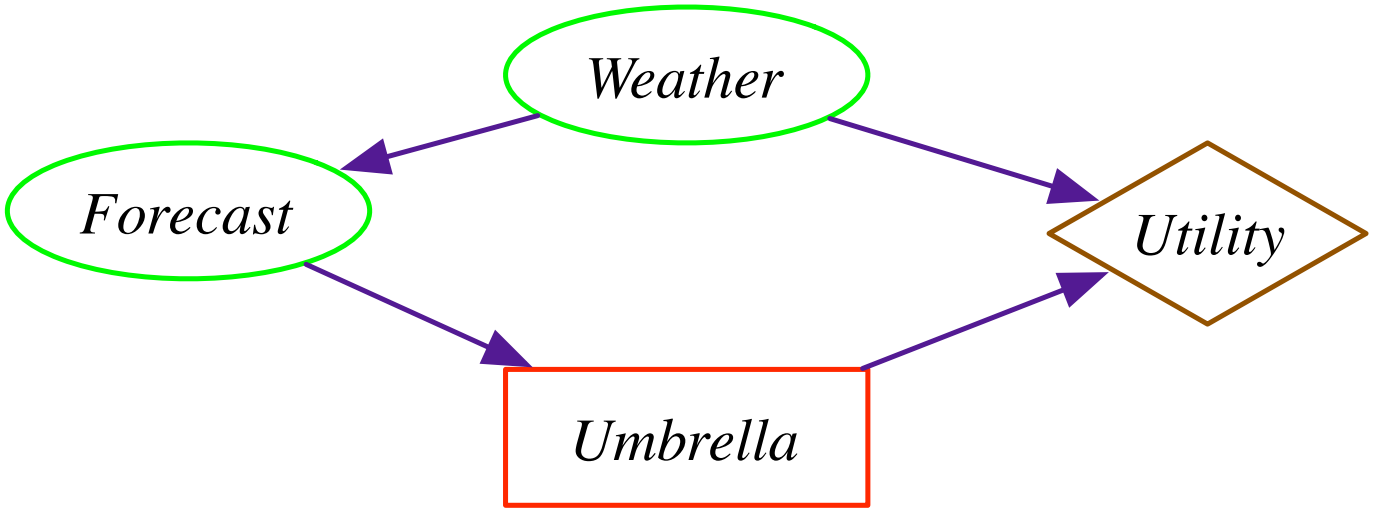
Policies

- A **policy** π is an assignment of a **decision function** to each decision variable D_i where;

$$\delta_i : dom(parents(D_i)) \rightarrow dom(D_i).$$

  This policy means that when the agent has observed $O \in dom(parents(D_i))$, it will do $\delta_i(O)$.
- A decision function for a decision variable is a function that specifies a value for the decision variable for each assignment of values of its parents.
- Policy specifies what the agent will do for each possible value that it could sense.

Policies: Umbrella Decision Network

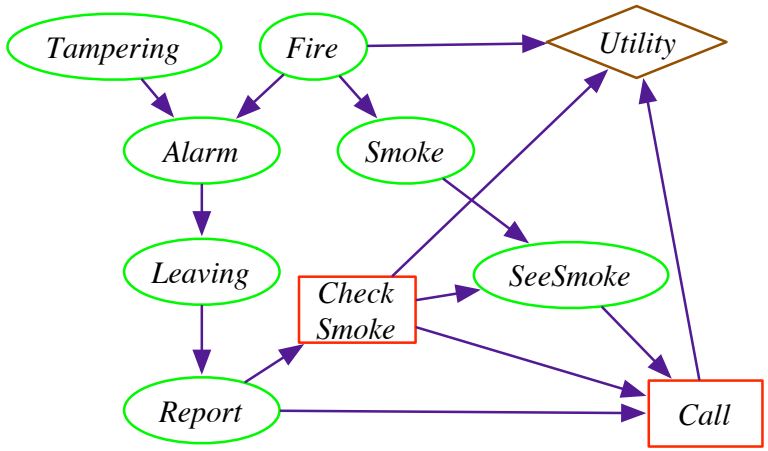


Examples of policies:

- Always bring the umbrella.
- Bring the umbrella only if the forecast is "rainy."
- Bring the umbrella only if the forecast is "sunny."
- ...

In umbrella problem there are 8 different policies, because there are three possible forecasts and there are two choices for each of the forecasts.

Policies: Alarm Problem



Decision function for CheckSmoke and for Call:

- Always check for smoke and never call.
- Always check for smoke, and call only if see smoke.
- Check for smoke if there is a report, and call only if there is a report and it sees smoke.

1,024 different policies. 4 decision functions for CheckSmoke. 28 decision functions for Call; for each of 8 assignments of values to parents of Call, agent can choose to call or not.

Expected Utility of a Policy

- Possible world ω **satisfies** policy π, written $\omega \models \pi$ if the world assigns the value to each decision node that the policy specifies. That is,

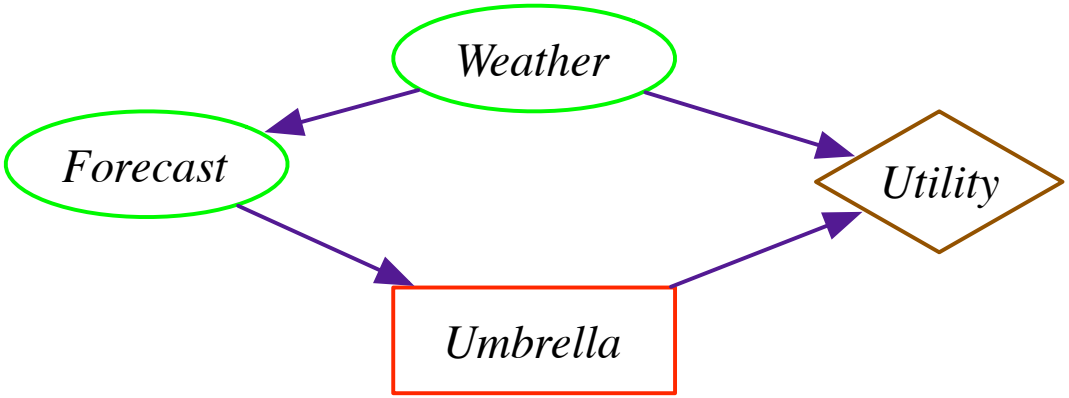
$$\forall \delta_i \in \pi, \omega \models \delta_i(\rho(\mathit{parents}(D_i), \omega)),$$

  where $\rho(\mathit{parents}(D_i), \omega)$ is the value of the parents of D_i in world ω.
- The **expected utility of policy** π is

$$\mathcal{E}(u|\pi) = \sum_{\omega \models \pi} u(\omega) \times P(\omega),$$

- An **optimal policy** (π^*) is the policy with highest expected utility; $\mathcal{E}(\pi^*) \geq \mathcal{E}(\pi)$ for all $\mathcal{E}(\pi)$

Expected utility: Umbrella network policy



Let π_1 be a policy to take the umbrella if the forecast is cloudy and to leave it at home otherwise. Expected utility of π_1 is obtained by averaging the utility over the worlds that satisfy this policy:

$$\begin{aligned}\mathcal{E}(\pi_1) = & P(norain)P(sunny|norain)Utility(norain, leaveIt)\\ & +P(norain)P(cloudy|norain)Utility(norain, takeIt)\\ & +P(norain)P(rainy|norain)Utility(norain, leaveIt)\\ & +P(rain)P(sunny|rain)Utility(rain, leaveIt)\\ & +P(rain)P(cloudy|rain)Utility(rain, takeIt)\\ & +P(rain)P(rainy|rain)Utility(rain, leaveIt)\end{aligned}$$

Problem: Finding an optimal policy

Decision D has k binary parents, and b possible actions:

- 2^k assignments of values to the parents.
- b^{2^k} different decision functions.

If there are multiple decision functions:

- Number of policies is the product of the number decision functions.
- Number of optimizations in the dynamic programming is the sum of the number of assignments of values to parents.
- Dynamic programming algorithm is much more efficient than searching through policy space.

Solution: Finding an optimal policy

- Remove all nodes that aren't ancestors of the utility node
- Create a factor for each conditional probability table and a factor for the utility.
- Repeat:
  - Sum out random variables that are not parents of a decision node.
  - Select a variable D that is only in a factor f with (some of) its parents.
  - Eliminate D by maximizing. This returns:
    - an optimal decision function for D: $argmax_D f$
    - a new factor: $\max_D f$
- until there are no more decision nodes.
- Sum out the remaining random variables. Multiply the factors: this is the expected utility of an optimal policy.

Solution: Variable elimination

1: **Procedure** VE_DN(*DN*):
2: **Inputs**
3: *DN* a decision network
4: **Output**
5: An optimal policy and its expected utility
6: **Local**
7: *DFs*: a set of decision functions, initially empty
8: *Fs*: a set of factors
9: Remove all variables that are not ancestors of the utility node
10: Create a factor in *Fs* for each conditional probability
11: Create a factor in *Fs* for the utility
12: **while** (there are decision nodes remaining)
13: Sum out each random variable that is not a parent of a decision node
14: *// at this stage there is one decision node D that is in a factor F with a subset of its parents*
15: Add max_D *F* to *Fs*.
16: Add $argmax_D$ *F* to *DFs*.
17: Sum out all remaining random variables
18: Return *DFs* and the product of remaining factors

Initial factors for the Umbrella Decision

Weather	Value
norain	0.7
rain	0.3

Weather	Fcast	Value
norain	sunny	0.7
norain	cloudy	0.2
norain	rainy	0.1
rain	sunny	0.15
rain	cloudy	0.25
rain	rainy	0.6

Weather	Umb	Value
norain	take	20
norain	leave	100
rain	take	70
rain	leave	0

Eliminate Weather by multiplying all three factors and summing out Weather, giving a factor on Forecast and Umbrella (next slide)

Eliminating By Maximizing

f:

Fcast	Umb	Val
sunny	take	12.95
sunny	leave	49.0
cloudy	take	8.05
cloudy	leave	14.0
rainy	take	14.0
rainy	leave	7.0

$\max_{Umb} f$:

Fcast	Val
sunny	49.0
cloudy	14.0
rainy	14.0

$\arg\max_{Umb} f$:

Fcast	Umb
sunny	leave
cloudy	leave
rainy	take

Value of Information

- The value of information X for decision D is the utility of the network with an arc from X to D (+ no-forgetting arcs) minus the utility of the network without the arc.
- The value of information is always non-negative.
- It is positive only if the agent changes its action depending on X.
- The value of information provides a bound on how much an agent should be prepared to pay for a sensor. How much is a better weather forecast worth?
- We need to be careful when adding an arc would create a cycle. E.g., how much would it be worth knowing whether the fire truck will arrive quickly when deciding whether to call them?

Value of Control

- The value of control of a variable X is the value of the network when you make X a decision variable (and add no-forgetting arcs) minus the value of the network when X is a random variable.
- You need to be explicit about what information is available when you control X.
- Often control is better than observation, but if you control X without observing, controlling X can be worse than observing X. E.g., controlling a thermometer.
- If you keep the parents the same, the value of control is always non-negative.

Agents as Processes

Agents carry out actions:

- forever **infinite horizon**
- until some stopping criteria is met **indefinite horizon**
- finite and fixed number of steps **finite horizon**

Decision-theoretic Planning

What should an agent do when

- it gets rewards (and punishments) and tries to maximize its rewards received
- actions can be stochastic; the outcome of an action can't be fully predicted
- there is a model that specifies the (probabilistic) outcome of actions and the rewards
- the world is fully observable

Initial Assumptions

- **flat** or modular or hierarchical
- **explicit states** or features or individuals and relations
- static or finite stage or **indefinite stage or infinite stage**
- **fully observable** or partially observable
- deterministic or **stochastic** dynamics
- goals or **complex preferences**
- **single agent** or multiple agents
- **knowledge is given** or knowledge is learned
- **perfect rationality** or bounded rationality

Utility and time

- Would you prefer $1000 today or $1000 next year?
- What price would you pay now to have an eternity of happiness?
- How can you trade off pleasures today with pleasures in the future?

Utility and time

- How would you compare the following sequences of rewards (per week):

  A: \$1000000, \$0, \$0, \$0, \$0, \$0,...

  B: \$1000, \$1000, \$1000, \$1000, \$1000,...

  C: \$1000, \$0, \$0, \$0, \$0,...

  D: \$1, \$1, \$1, \$1, \$1,...

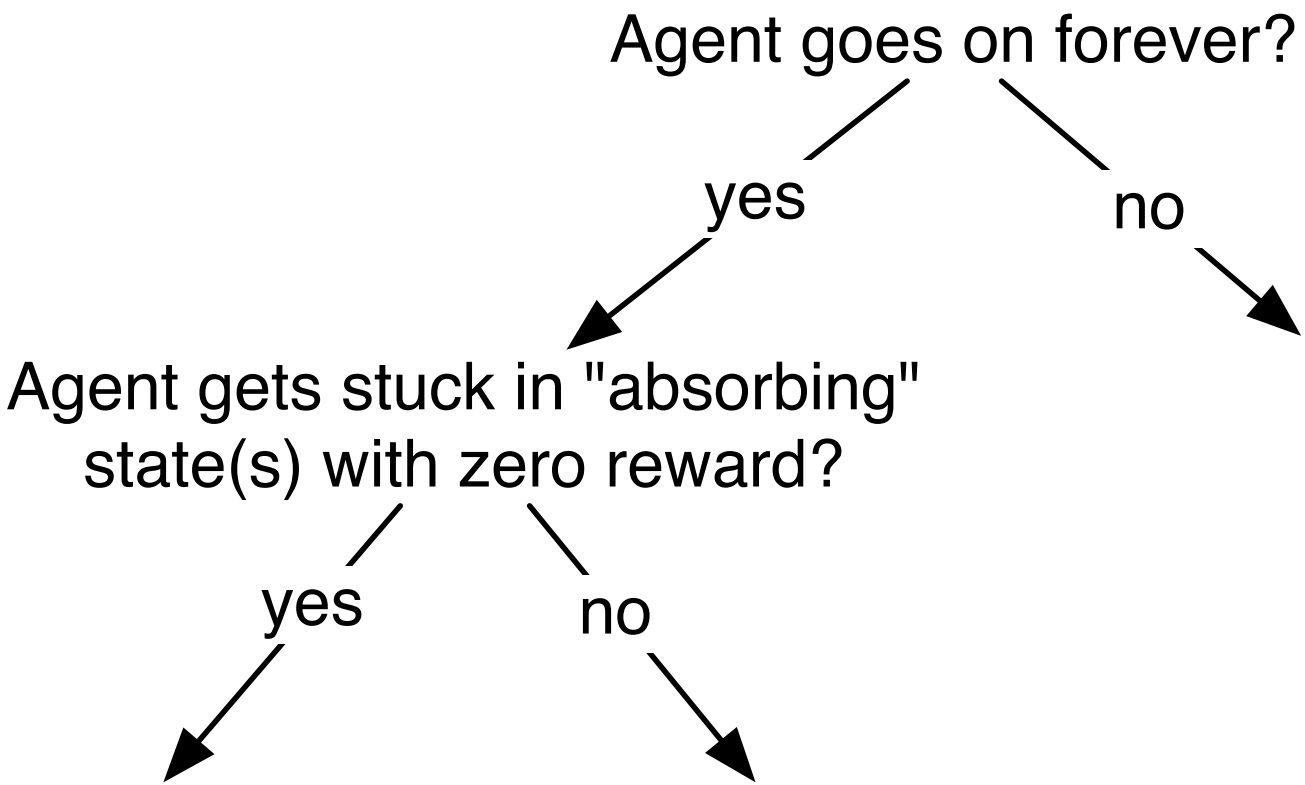
  E: \$1, \$2, \$3, \$4, \$5,...

Rewards and Values

Suppose the agent receives a sequence of rewards $r_1, r_2, r_3, r_4, \ldots$ in time. What utility should be assigned? "Return" or "value"

- **total reward** $V = \sum_{i=1}^{\infty} r_i$
- **average reward** $V = \lim_{n \to \infty} (r_1 + \cdots + r_n)/n$

Average vs Accumulated Rewards



Rewards and Values

Suppose the agent receives the sequence of rewards $r_1, r_2, r_3, r_4, \ldots$. What value should be assigned?

- **total reward** $V = \sum_{i=1}^{\infty} r_i$
- **average reward** $V = \lim_{n \to \infty} (r_1 + \cdots + r_n)/n$
- **discounted reward**
  $V = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \cdots$
  γ is the **discount factor** $0 \leq \gamma \leq 1$.

Properties of the Discounted Rewards

- The discounted return for rewards $r_1, r_2, r_3, r_4, \ldots$ is

$$\begin{aligned} V &= r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \cdots \\ &= r_1 + \gamma(r_2 + \gamma(r_3 + \gamma(r_4 + \ldots))) \end{aligned}$$

- If V_t is the value obtained from time step t

$$V_t = r_t + \gamma V_{t+1}$$

- How is the infinite future valued compared to immediate rewards?
  $1 + \gamma + \gamma^2 + \gamma^3 + \cdots = 1/(1-\gamma)$; Therefore:

$$\frac{\text{minimum reward}}{1-\gamma} \leq V_t \leq \frac{\text{maximum reward}}{1-\gamma}$$

- We can approximate V with the first k terms, with error:

$$V - (r_1 + \gamma r_2 + \cdots + \gamma^{k-1} r_k) = \gamma^k V_{k+1}$$

World State

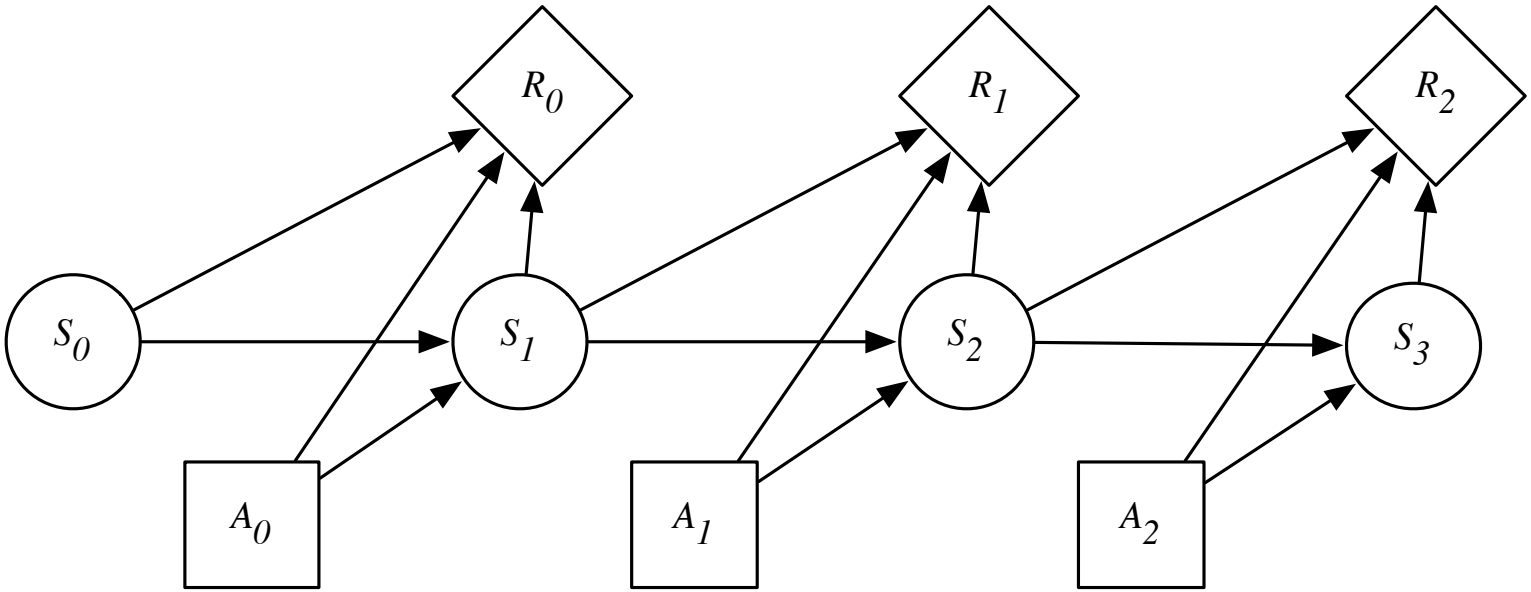
- The world state is the information such that if the agent knew the world state, no information about the past is relevant to the future: **Markovian assumption**.
- S_i is state at time i, and A_i is the action at time i:

$$P(S_{t+1}|S_0, A_0, \ldots, S_t, A_t) = P(S_{t+1}|S_t, A_t)$$

  $P(s'|s, a)$ is the probability that the agent will be in state s' immediately after doing action a in state s.
- The dynamics is **stationary** if the distribution is the same for each time point.

Decision Processes

- A **Markov decision process** augments a Markov chain with actions and values:



Markov Decision Processes

An MDP consists of:

- set S of states.
- set A of actions.
- $P(S_{t+1}|S_t, A_t)$ specifies the dynamics.
- $R(S_t, A_t, S_{t+1})$ specifies the reward at time t. $R(s, a, s')$ is the expected reward received when the agent is in state s, does action a and ends up in state s'.
- γ is discount factor.

Example: to exercise or not?

Each week *Sam* has to decide whether to exercise or not:

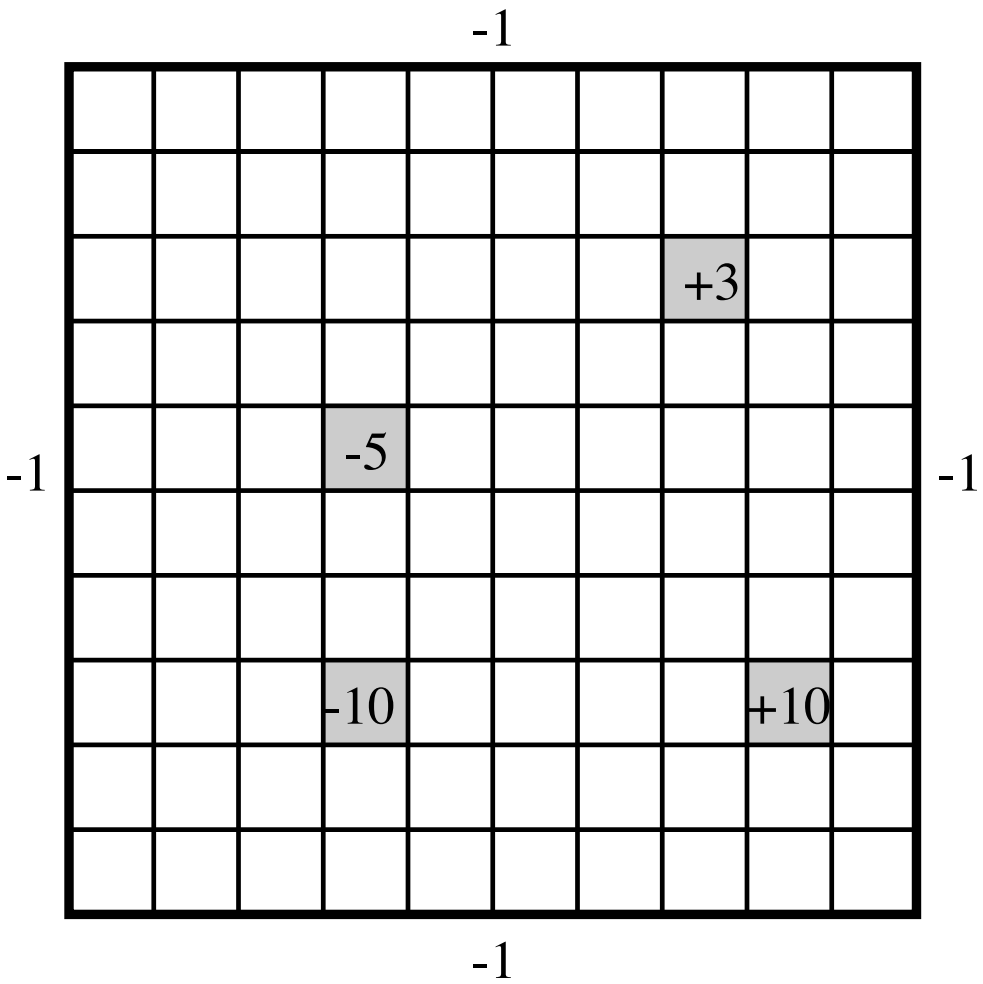
- States: $\{fit, unfit\}$
- Actions: $\{exercise, relax\}$
- Dynamics:

State	Action	$P(fit \mid State, Action)$
fit	exercise	0.99
fit	relax	0.7
unfit	exercise	0.2
unfit	relax	0.0

- Reward (does not depend on resulting state):

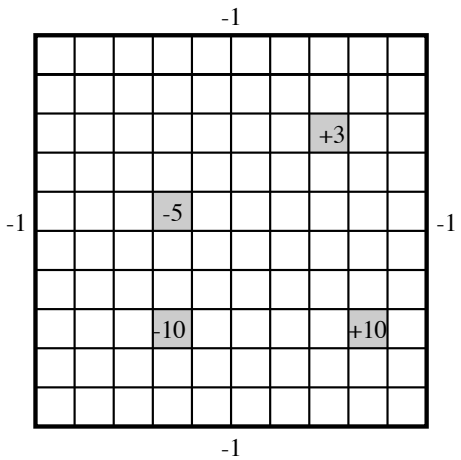
State	Action	Reward
fit	exercise	8
fit	relax	10
unfit	exercise	0
unfit	relax	5

Example: Simple Grid World



Grid World Model

- Actions: up, down, left, right.
- 100 states corresponding to the positions of the robot.
- Robot goes in the commanded direction with probability 0.7, and one of the other directions with probability 0.1.
- If it crashes into an outside wall, it remains in its current position and has a reward of -1.
- Four special rewarding states; the agent gets the reward when leaving.



Planning Horizons

The planning horizon is how far ahead the planner looks to make a decision.

- If the robot gets flung to one of the corners at random after leaving a positive (+10 or +3) reward state.
  - the process never halts
  - **infinite horizon**
- If the robot gets +10 or +3 in the state, then it stays there getting no reward. These are **absorbing states.**
  - The robot will eventually reach an absorbing state.
  - **indefinite horizon**

Information Availability

What information is available when the agent decides what to do?

- **fully-observable MDP** the agent gets to observe S_t when deciding on action A_t.
- **partially-observable MDP** (POMDP) the agent has some noisy sensor of the state. It needs to remember its sensing and acting history.

Policies

- A **stationary policy** is a function which assigns an action to each state:

  $$\pi : S \rightarrow A$$

  Given a state s, $\pi(s)$ specifies what action the agent who is following π will do.
- An **optimal policy** is one with maximum expected value
- For a fully-observable MDP with stationary dynamics and rewards with infinite or indefinite horizon, there is always an optimal stationary policy.

Value of a Policy

Given a policy π:

- $Q^\pi(s, a)$, where a is an action and s is a state, is the expected value of doing a in state s, then following policy π.
- $V^\pi(s)$, where s is a state, is the expected value of following policy π in state s.
- Q^π and V^π can be defined mutually recursively:

$$\begin{aligned} Q^\pi(s, a) &= \sum_{s'} P(s'|a, s) \left(R(s, a, s') + \gamma V^\pi(s')\right) \\ V^\pi(s) &= Q^\pi(s, \pi(s)) \end{aligned}$$

Value of the Optimal Policy

- $Q^*(s, a)$, where a is an action and s is a state, is the expected value of doing a in state s, then following the optimal policy.
- $V^*(s)$, where s is a state, is the expected value of following the optimal policy in state s.
- $\pi^*(s)$ is one of the a's that results in the maximum value of $Q^*(s, a)$
- Q^* and V^* can be defined mutually recursively:

$$\begin{aligned} Q^*(s,a) &= \sum_{s'} P(s'|a,s)\left(R(s,a,s') + \gamma V^*(s')\right) \\ V^*(s) &= \max_a Q^*(s,a) \\ \pi^*(s) &= \mathit{argmax}_a Q^*(s,a) \end{aligned}$$

Q, V, π, R

$$\begin{aligned} Q^*(s,a) &= \sum_{s'} P(s'|a,s)\left(R(s,a,s') + \gamma V^*(s')\right) \\ V^*(s) &= \max_a Q^*(s,a) \\ \pi^*(s) &= argmax_a Q^*(s,a) \end{aligned}$$

Let

$$R(s,a) = \sum_{s'} P(s'|a,s) R(s,a,s')$$

Then alternatively stated:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|a,s) V^*(s')$$

Value Iteration

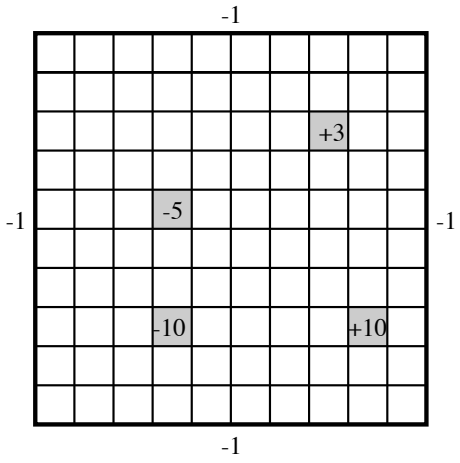
- Let V_k and Q_k be k-step lookahead value and Q functions.
- Idea: Given an estimate of the k-step lookahead value function, determine the $k+1$ step lookahead value function.
- Set V_0 arbitrarily.
- Compute Q_{i+1}, V_{i+1} from V_i.
- This converges exponentially fast (in k) to the optimal value function: rate is a contraction the factor γ

  The error reduces proportionally to $\frac{\gamma^k}{1-\gamma}$

Value Iteration

1: **Procedure** Value_Iteration(S,A,P,R,θ)
2: **Inputs**
3: S is the set of all states
4: A is the set of all actions
5: P is state transition function specifying $P(s'|s,a)$
6: R is a reward function $R(s,a,s')$
7: θ a threshold, $\theta>0$
8: **Output**
9: $\pi[S]$ approximately optimal policy
10: $V[S]$ value function
11: **Local**
12: real array $V_k[S]$ is a sequence of value functions
13: action array $\pi[S]$
14: assign $V_0[S]$ arbitrarily
15: $k \leftarrow 0$
16: **repeat**
17: $k \leftarrow k+1$
18: **for each** state s **do**
19: $V_k[s] = max_a \sum_{s'} P(s'|s,a) (R(s,a,s')+ \gamma V_{k-1}[s'])$
20: **until** $\forall s \ |V_k[s]-V_{k-1}[s]| < \theta$
21: **for each** state s **do**
22: $\pi[s] = argmax_a \sum_{s'} P(s'|s,a) (R(s,a,s')+ \gamma V_k[s'])$
23: **return** π, V_k

Value Iteration



For cells surrounding +10

V_1

0	0	-0.1
0	10	-0.1
0	0	-0.1

	Prob	Reward		Future Value	
	0.7×(	*0*	+	*0.9×10)*	*Agent goes left*
+	*0.1×(*	*0*	+	*0.9×-0.1)*	*Agent goes up*
+	*0.1×(*	*-1*	+	*0.9×-0.1)*	*Agent goes right*
+	*0.1×(*	*0*	+	*0.9×-0.1)*	*Agent goes down*

To calculate 6.2 in V_2:

V_2

0	6.3	-0.1
6.3	9.8	6.2
0	6.3	-0.1

V_3

4.5	6.2	4.4
6.2	9.7	6.6
4.5	6.1	4.4

Demo

Show demo

Asynchronous Value Iteration

- The agent doesn't need to sweep through all the states, but can update the value functions for each state individually.
- This converges to the optimal value functions, if each state and action is visited infinitely often in the limit.
- It can either store $V[s]$ or $Q[s, a]$.

Asynchronous VI: storing $Q[s, a]$

Repeat forever:

- Select state s, action a
- $Q[s, a] \leftarrow \sum_{s'} P(s'|s, a) \left(R(s, a, s') + \gamma \max_{a'} Q[s', a'] \right)$

Asynch Value Iteration


```
1: Procedure Asynchronous_Value_Iteration(S,A,P,R)
2:       Inputs
3:             S is the set of all states
4:             A is the set of all actions
5:             P is state transition function specifying P(s'|s,a)
6:             R is a reward function R(s,a,s')
7:       Output
8:             π[s] approximately optimal policy
9:             Q[S,A] value function
10:      Local
11:            real array Q[S,A]
12:            action array π[S]
13:      assign Q[S,A] arbitrarily
14:      repeat
15:            select a state s
16:            select an action a
17:            Q[s,a] = Σ_s' P(s'|s,a) (R(s,a,s')+ γmax_a' Q[s',a'])
18:      until termination
19:      for each state s do
20:            π[s] = argmax_a Q[s,a]
21:      return π,Q
```

To store $V[s]$ instead, update with:

$V[s] \leftarrow \max_a \sum_{s'} P(s'|s,a)\,(R(s,a,s') + \gamma V[s'])$

and store: $\pi[s] \leftarrow argmax_a \sum_{s'} P(s'|s,a)\,(R(s,a,s') + \gamma V[s'])$

Asynchronous VI: storing $V[s]$

In other words

Repeat forever:

- Select state s
- $V[s] \leftarrow \max_a \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V[s']\right)$

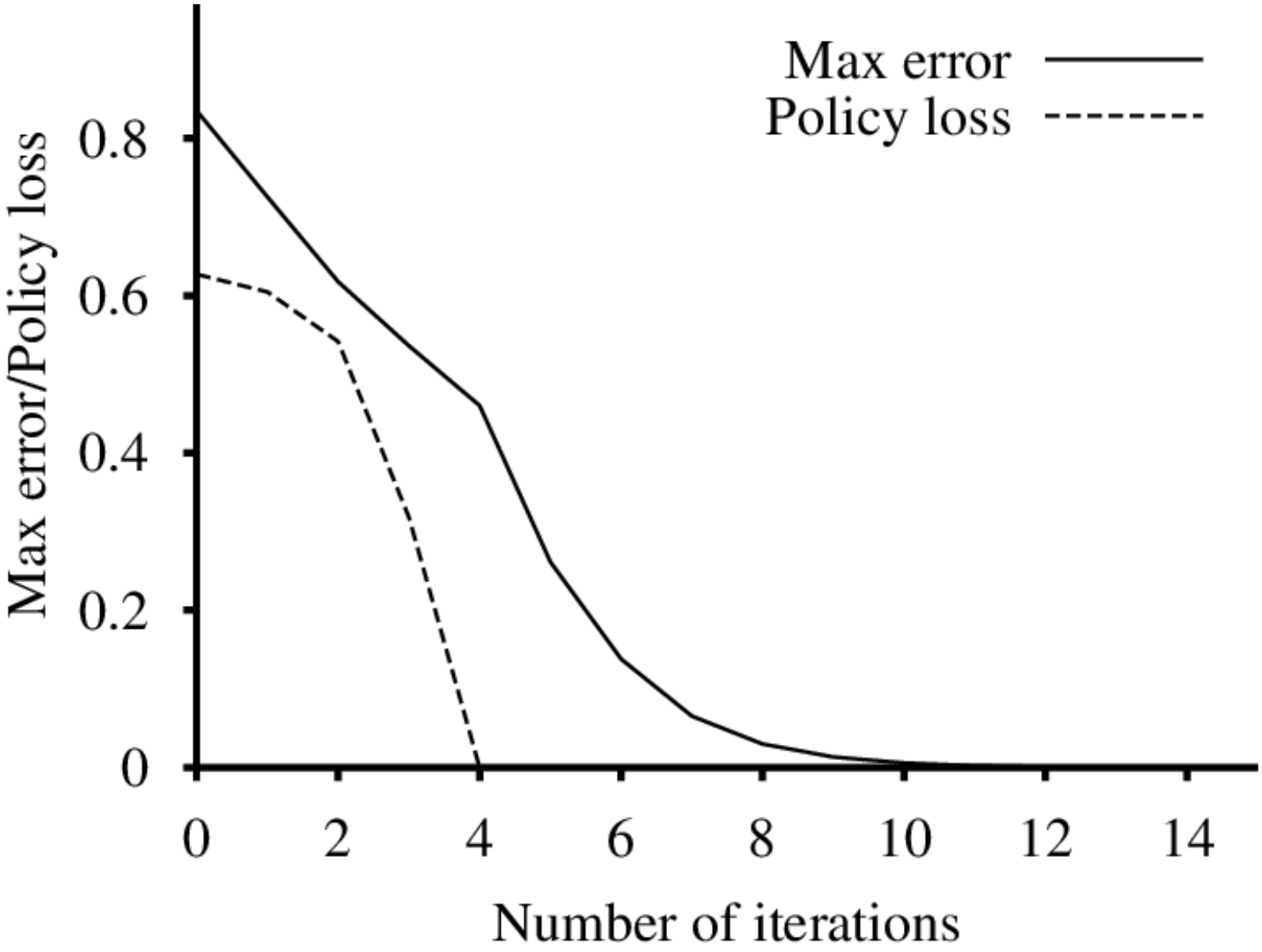
Policy Iteration

- Set π_0 arbitrarily, let $i = 0$
- Repeat:
  - evaluate $Q^{\pi_i}(s, a)$
  - let $\pi_{i+1}(s) = argmax_a Q^{\pi_i}(s, a)$
  - set $i = i + 1$
- until $\pi_i(s) = \pi_{i-1}(s)$

Evaluating $Q^{\pi_i}(s, a)$ means finding a solution to a set of $|S| \times |A|$ linear equations with $|S| \times |A|$ unknowns.

It can also be approximated iteratively by repeating this:
$V_{i+1}[s] \leftarrow \sum_{s'} P(s'|s, a) \left(R(s, a, s') + \gamma V_i[s']\right)$

Policy error versus value error



Which is a better stopping criterion?

Policy Iteration, high level (AIMA)

Extend the following principle to a lower-level:

1 **function** POLICY-ITERATION(mdp) **returns** a policy
2 **inputs** *mdp* with states S, actions $A(s)$, transition model $P(s'|s,a)$
3 **local** U vector of utilities for states in S, initially 0,
π policy vector indexed by state, initially 0
4 **repeat**
5 $U \leftarrow POLICY\_EVALUATION(\pi, U, mdp)$
6 $unchanged \leftarrow true$
7 **for** each state s in S **do**
8 **if** $\max_{a \in A(s)} \sum_{s'} P(s'|s,a)U[s']) > \sum_{s'} P(s'|s,\pi[s])U[s'])$
9 $\pi[s] = argmax_{a \in A(s)} \sum_{s'} P(s'|s,a)U[s'])$
10 $unchanged \leftarrow false$
11 **until** unchanged
12 **return** π

Note:

Value $\rightarrow$ improvement $\rightarrow$ Policy $\rightarrow$ evaluate $\rightarrow$ **Value...**

$\neq$ Value $\rightarrow$ improvement $\rightarrow$ **Value...**

Which is faster?

Policy Iteration (ArtInt)


```
1: Procedure Policy_Iteration(S,A,P,R)
2:       Inputs
3:             S is the set of all states
4:             A is the set of all actions
5:             P is state transition function specifying P(s'|s,a)
6:             R is a reward function R(s,a,s')
7:       Output
8:             optimal policy π
9:       Local
10:            action array π[S]
11:            Boolean variable noChange
12:            real array V[S]
13:      set π arbitrarily
14:      repeat
15:            noChange ←true
16:            Solve V[s] = ∑s'∈S P(s'|s,π[s])(R(s,a,s')+γV[s'])
17:            for each s∈S do
18:                  Let QBest=V[s]
19:                  for each a ∈A do
20:                        Let Qsa=∑s'∈S P(s'|s,a)(R(s,a,s')+γV[s'])
21:                        if (Qsa > QBest) then
22:                              π[s]←a
23:                              QBest ←Qsa
24:                              noChange ←false
25:      until noChange
26:      return π
```

Approximately solve $V[S]$ iteratively via: for i in k, for s in S, do

$$V_{i+1}[s] \leftarrow \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V_i[s']\right)$$

Modified Policy Iteration

Similar to last line on previous slide, but storing $Q[s, a]$

- Set $\pi[s]$ arbitrarily
- Set $Q[s, a]$ arbitrarily
- Repeat for a while:
  - Select state s, action a
  - $Q[s, a] \leftarrow \sum_{s'} P(s'|s, a) \left(R(s, a, s') + \gamma Q[s', \pi[s']]\right)$
  - $\pi[s] \leftarrow argmax_a Q[s, a]$
- until $\pi_i(s) = \pi_{i-1}(s)$

Partially Observable Markov Decision Process

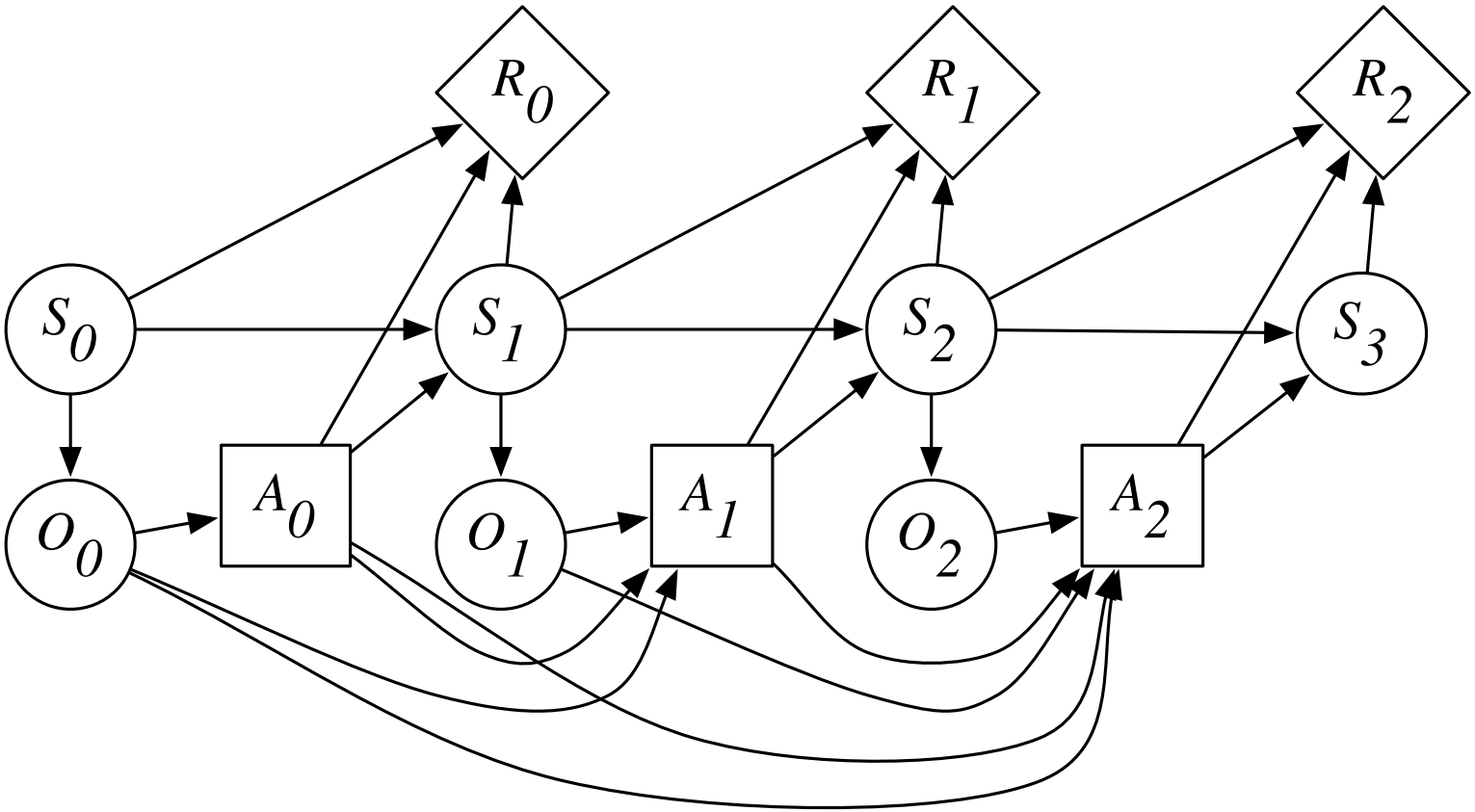
- A partially observable Markov decision process (POMDP) is a combination of an MDP and a hidden Markov model.
- Instead of assuming that the state is observable, we assume that there are some partial and/or noisy observations of the state that the agent gets to observe before it has to act.

POMDP

A POMDP consists of the following:

- S, a set of states of the world;
- A, a set of actions;
- O, a set of possible observations;
- $P(S_0)$, which gives the probability distribution of the starting state;
- $P(S'|S, A)$, which specifies the dynamics - the probability of getting to state S' by doing action A from state S
- $R(S, A, S')$, which gives the expected reward of starting in state S, doing action A, and transitioning to state S'
- $P(O|S)$, which gives the probability of observing O given the state is S

POMDP



Can be solved using:

- A version of value iteration after reducing to MDP
- Variable elimination
- other methods too messy to mention now