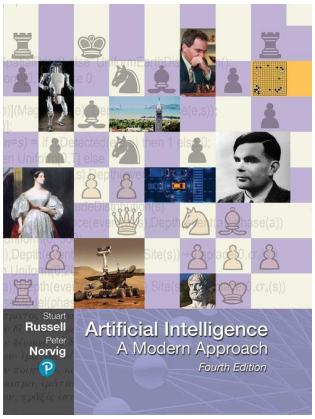
Artificial Intelligence: A Modern Approach

Fourth Edition



Chapter 17

Making Complex Decisions

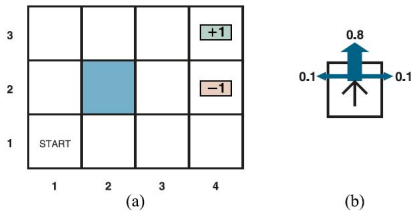
Outline

- Sequential Decision Problems
- Basic Probability Notation
- Bandit Problems
- Partially Observable MDPs
- Algorithms for Solving POMDPs

Sequential Decision Problems

- **Markov decision process (MDP):** a sequential decision problem for a fully observable, stochastic environment
- MDP consists of:
  - a set of states (with an initial state s_0);
  - a set ACTIONS(s) of actions in each state;
  - a transition model $P(s \mid s, a)$; and
  - a reward function $R(s, a, s)$. Methods for
- MDP solutions usually involve **dynamic programming** simplifying a problem by recursively breaking it into smaller pieces and remembering the optimal solutions to the pieces.
- A solution called **policy**.
  - specify what the agent should do for any state that the agent might reach
  - the quality of a policy is measured by the expected utility of possible environment histories generated
  - **optimal policy**: highest expected utility

Sequential Decision Problems

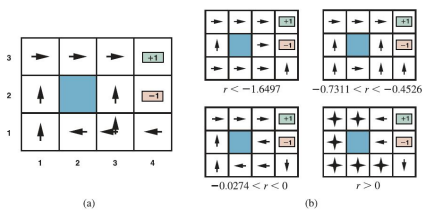


a) A simple, stochastic 43 environment that presents the agent with a sequential
decision problem.

(b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. Transitions into the two terminal states have reward +1 and –1, respectively, and all other transitions have a reward of –0.04.

Sequential Decision Problems

- Utilities over time
  - Not only possibility for the utility function on environment histories
  - Utilities: $U_h([s_0, a_0, s_1, a_1 \ldots, s_n])$.



(a) The optimal policies for the stochastic environment with $r = 0.04$ for transitions between nonterminal states. There are two policies because in state (3,1) both *Left* and *Up* are optimal. (b) Optimal policies for four different ranges of r.

Sequential Decision Problems

- **Finite horizon:** fixed time N after which nothing matters
  - $Uh([s_0, a_0, s_1, a_1, \ldots, s_{N+k}]) = Uh([s_0, a_0, s_1, a_1, \ldots, s_N])$
  - optimal action in a given state may depend on how much time is left
  - **Nonstationary:** A policy that depends on the time
- **infinite horizon:** no fixed time limit
  - there is no reason to behave differently in the same state at different times.
- Optimal action depends only on the current state, and the optimal policy is stationary.

Sequential Decision Problems

- **Additive discounted rewards:**
  $U_h([s_0, a_0, s_1, a_1, s_2, \ldots]) = R(s_0, a_0, s_1) + \gamma R(s_1, a_1, s_2) + \gamma^2 R(s_2, a_2, s_3) + \cdots$,
  where the **discount factor** γ is a number between 0 and 1.

  γ close to 0, not willing wait
  γ close to 1, willing wait long term reward

- **Additive** discounted rewards makes sense: empirical, economical, uncertainty about true rewards, preferences over histories.
- Reduces complexity of infinite sequence due to utility is finite.
- if $\gamma < 1$ and rewards are bounded by $\pm R_{\max}$, we have

$$U_h([s_0, a_0, s_1, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \leq \sum_{t=0}^{\infty} \gamma^t R_{\max} = \frac{R_{\max}}{1-\gamma},$$

- **Proper policy**: guaranteed to reach a terminal state
- Infinite sequences can be compared in terms of the **average reward** obtained per time

Sequential Decision Problems

- Utility of a state is the expected reward for the next transition plus the discounted utility of the next state, assuming that the agent chooses the optimal actio

$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)[R(s, a, s') + \gamma U(s')].$$

- This is called the Bellman equation, after Richard Bellman (1957).
- **Action-utility function**, or **Q-function**: $Q(s, a)$
  - the expected utility of taking a given action in a given state.
  - related to u way:

$$U(s) = \max_a Q(s, a).$$

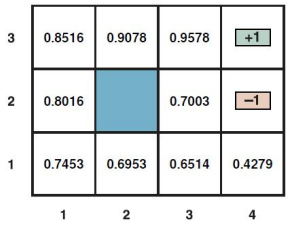
- The optimal pc om the Q-function

$$\pi^*(s) = \operatorname{argmax}_a Q(s, a)$$

- The Q-f

function Q-VALUE(mdp, s, a, U) **returns** a utility value
return $\sum_{s'} P(s' \mid s, a)[R(s, a, s') + \gamma U[s']]$

Sequential Decision Problems



The utilities of the states in the 4 3 world with $\gamma = 1$ and $r = 0.04$ for transitions to nonterminal states.

The expression for $U(1, 1)$ is

$\max\{ [0.8(-0.04 + \gamma U(1, 2)) + 0.1(-0.04 + \gamma U(2, 1)) + 0.1(-0.04 + \gamma U(1, 1))],$
$[0.9(-0.04 + \gamma U(1, 1)) + 0.1(-0.04 + \gamma U(1, 2))],$
$[0.9(-0.04 + \gamma U(1, 1)) + 0.1(-0.04 + \gamma U(2, 1))],$
$[0.8(-0.04 + \gamma U(2, 1)) + 0.1(-0.04 + \gamma U(1, 2)) + 0.1(-0.04 + \gamma U(1, 1))]\}$

Reward scales

- Transformation of rewards will leave the optimal policy unchanged in an MDP:

$$R(s, a, s) = mR(s, a, s) + b.$$

$$Q(s,a) = \sum_{s'} P(s' \mid s, a)[R(s, a, s') + \gamma \max_{a'} Q(s', a')].$$

$$\begin{aligned} Q'(s,a) &= \sum_{s'} P(s' \mid s, a)[R(s, a, s') + \gamma\Phi(s') - \Phi(s) + \gamma \max_{a'} Q'(s', a')] \\ &= \sum_{s'} P(s' \mid s, a)[R'(s, a, s') + \gamma \max_{a'} Q'(s', a')]. \end{aligned}$$

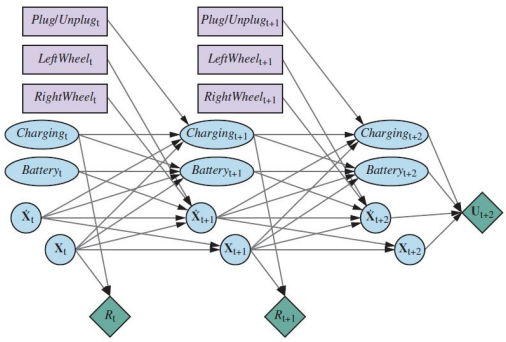
- Extract the optimal policy for M'

$$\pi^*_{M'}(s) = \operatorname{argmax}_a Q'(s,a) = \operatorname{argmax}_a Q(s,a) - \Phi(s) = \operatorname{argmax}_a Q(s,a) = \pi^*_M(s).$$

- *The* function $\Phi(s)$ is often called a **potential**,
- if $\Phi(s)$ has higher value in states that have higher utility, the addition of $\gamma\Phi(s') - \Phi(s)$ to the reward has the effect of leading the agent "uphill" in utility.

Representing MDPs

- **Dynamic decision networks**, or **DDNs** are factored representations

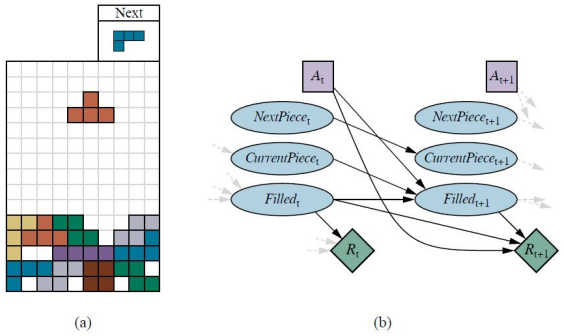


A dynamic decision network for a mobile robot with state variables for battery level, charging status, location, and velocity, and action variables for the left and right wheel motors and for charging.

Representing MDPs

- The state S_t decomposed into four state variables
  - $\mathbf{X}_t$ consists of the two-dimensional location on a grid plus the orientation;
  - $\dot{\mathbf{X}}_t$ is the rate of change of $\mathbf{X}_t$;
  - $Charging_t$ is true when the robot is plugged in to a power source;
  - $Battery_t$ is the battery level, which we model as an integer in the range 0, ..., 5.
- The state space for the MDP is the Cartesian product of the ranges of these four variables.
- The action is now a set $\mathbf{A}_t$ *Unplug*, which has three values (*plug*, *unplug*, and *no p*); *LeftWheel* for the power sent to the left wheel; and *RightWheel* for the power sent to the right wheel.
- The overall transition model is the conditional distribution $\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{X}_t, \mathbf{A}_t)$, computed as a product of conditional probabilities from the DDN

Representing MDPs



a) The game of Tetris. The T-shaped piece at the top center can be dropped in any orientation and in any horizontal position. If a row is completed, that row disappears and the rows above it move down, and the agent receives one point. The next piece (here, the L-shaped piece at top right) becomes the current piece, and a new next piece appears, chosen at random from the seven piece types. The game ends if the board fills up to the top.
b) The DDN for the Tetris MDP.

Algorithms for MDPs (Value Iteration)

Value Iteration

- **The Bellman equation** is the basis of the value iteration
- If there are n possible states, then there are n Bellman equations,
- The n equations contain n unknowns—the utilities of the states.
- The equations are nonlinear, because the "max" operator is not a linear operator.
- start with arbitrary initial values for the utilities, calculate the right-hand side of the equation, and plug it into the left-hand side—thereby updating the utility of each state from the utilities of its neighbors.
- Repeat this until reach an equilibrium.
- The iteration step, called a Bellman update, looks like this

$$U_{i+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} P(s' \mid s,a)[R(s,a,s') + \gamma U_i(s')],$$

- update is assumed to be applied simultaneously to all the states at each iteration

Algorithms for MDPs (Value Iteration)

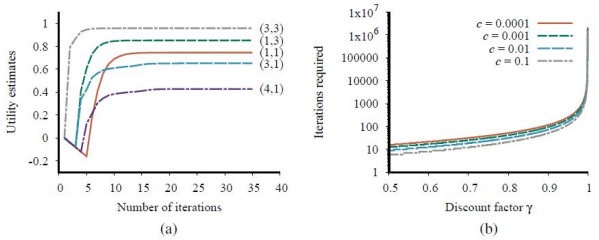
function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s,a)$, rewards $R(s,a,s')$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum relative change in the utility of any state

  repeat
    $U \leftarrow U'$; $\delta \leftarrow 0$
    for each state s **in** S **do**
      $U'[s] \leftarrow \max_{a \in A(s)}$ Q-VALUE(mdp, s, a, U)
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta \leq \epsilon(1-\gamma)/\gamma$
  return U

The value iteration algorithm for calculating utilities of states.

Algorithms for MDPs (Value Iteration)

Value Iteration applied to 4 x 3 world



(a) Graph showing the evolution of the utilities of selected states using value iteration.
(b) The number of value iterations required to guarantee an error of at most $E = c\,R_{max}$ for different values of c, as a function of the discount factor γ.

Algorithms for MDPs (Policy Iteration)

Policy Iteration

- Alternates the following two steps beginning from some initial policy π_0:
  - **Policy evaluation**: given a policy π_i, calculate $U_i = U^{\pi i}$ the utility of each state if π_i were to be executed.
  - **Policy improvement**: Calculate a new MEU policy π_{i+1}, using one-step look-ahead based on U_i
- The algorithm terminates when the policy improvement step yields no change in the utilities.'
- Action in each state is fixed by the policy. At the ith iteration, the policy π_i specifies the action $\pi_i(s)$ in state s.
- Simplified version of the Bellman equation relating the utility of s (under π_i) to the utilities of its neighbors:

$$U_i(s) = \sum_{s'} P(s' \mid s, \pi_i(s))[R(s, \pi_i(s), s') + \gamma U_i(s')].$$

- For large state spaces, time is prohibitive. Simplified Bellman update **modified policy iteration**

$$U_{i+1}(s) \leftarrow \sum_{s'} P(s' \mid s, \pi_i(s))[R(s, \pi_i(s), s') + \gamma U_i(s')],$$

Algorithms for MDPs (Policy Iteration)

function POLICY-ITERATION(mdp) **returns** a policy
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s,a)$
  local variables: U, a vector of utilities for states in S, initially zero
    π, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
    unchanged? $\leftarrow$ true
    for each state s **in** S **do**
      $a^* \leftarrow \underset{a \in A(s)}{\text{argmax}}$ Q-VALUE(mdp, s, a, U)
      if Q-VALUE(mdp, s, a^*, U) $>$ Q-VALUE($mdp, s, \pi[s], U$) **then**
        $\pi[s] \leftarrow a^*$; *unchanged*? $\leftarrow$ false
  until *unchanged*?
  return π

The policy iteration algorithm for calculating an optimal policy.

Algorithms for MDPs (Linear programming)

Linear programming (LP)

- General approach for formulating constrained optimization problems
- Minimize $U(s)$ for all s subject to the inequalities for every state s and every action a.

$$U(s) \geq \sum_{s'} P(s' \mid s,a)[R(s,a,s') + \gamma U(s')]$$

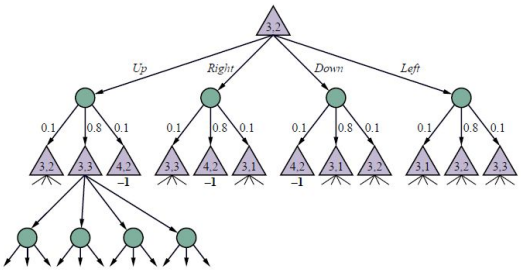
- In practice, it turns out that LP solvers are seldom as efficient as dynamic programming for solving MDPs.
- linear programming is solvable in polynomial time polynomial however the number of states is often very large.

Algorithms for MDPs (Online algorithms)

Online Algorithms

- EXPECTIMAX algorithm builds a tree of alternating max and chance nodes
- An evaluation function can be applied to the nonterminal leaves of the tree, or they can be given a default value.
- A decision can be extracted from the search tree by backing up the utility values from the leaves, taking an average at the chance nodes and taking the maximum at the decision nodes.
- The explored states actually constitute a sub-MDP of the original MDP, and this sub-MDP can be solved using any of the algorithms in this chapter.
- This approach is called real-time dynamic programming (RTDP)

Algorithms for MDPs (Online algorithms)



Part of an expectimax tree for the 43 MDP rooted at (3,2). The triangular nodes are max modes and the circular nodes are chance nodes.

Bandit Problems

- In Las Vegas, a **one-armed bandit** is a **slot machine**.
- An **n-armed bandit has n levers**.
- Behind each lever is a fixed but unknown probability distribution of winnings.
- The gambler must choose which lever to play on each successive coin
- Tradeoff between **exploitation** of the current best action to obtain rewards and **exploration** of previously unknown states and actions to gain information
- **Formal model for real problems** important areas, such as
  - deciding which of n possible new treatments to try to cure a disease,
  - which of n possible investments to put part of your savings into,
  - which of n possible research projects to fund

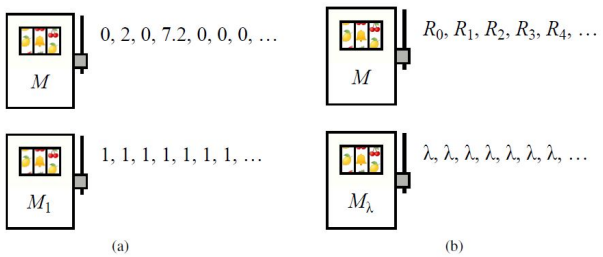
Bandit Problems

Bandit problems definitions

- "Each arm M_i is a **Markov reward process** or MRP, that is, an MDP with only one possible action a_i. It has states S_i, transition model $P_i(s' \mid s, a_i)$, and reward $R_i(s, a_i, s')$. The arm defines a distribution over sequences of rewards $R_{i,0}$, $R_{i,1}$, $R_{i,2}$, . . ., where each $R_{i,t}$ is a random variable."
- "The overall bandit problem is an MDP: the state space is given by the Cartesian product $S = S_1 \ S_n$; the actions are a_1, . . . , a_n; the transition model updates the state of whichever arm M_i is selected, according to its specific transition model, leaving the other arms unchanged; and the discount factor is γ."

- The key property is that the arms are independent, only one arm can work at a time.

Bandit Problems



(a) A simple deterministic bandit problem with two arms. The arms can be pulled in any order, and each yields the sequence of rewards shown.
(b) A more general case of the bandit in (a), where the first arm gives an arbitrary sequence of rewards and the second arm gives a fixed reward λ.

Bandit Problems

Computing the utility (total discounted reward) for each arm:

$$U(M) = (1.0\times 0)+(0.5\times 2)+(0.5^2\times 0)+(0.5^3\times 7.2) = 1.9$$
$$U(M_1) = \sum_{t=0}^{\infty} 0.5^t = 2.0.$$

Starting with M and then switching to M after the fourth reward gives the sequence S = 0, 2, 0, 7.2, 1, 1, 1, . . ., for which

$$U(S) = (1.0\times 0)+(0.5\times 2)+(0.5^2\times 0)+(0.5^3\times 7.2)+\sum_{t=4}^{\infty} 0.5^t = 2.025.$$

The strategy S that switches from M to M_1 at the right time is better than either arm individually

Bandit Problems

Optimal strategy is to run arm M up to time T and then switch to M_λ for the rest of time.

Gittins Index:

$$\lambda = \max_{T>0} \frac{E\left(\sum_{t=0}^{T-1}\gamma^t R_t\right)}{E\left(\sum_{t=0}^{T-1}\gamma^t\right)}.$$

value describes the maximum obtainable utility per unit of discounted time.

T	1	2	3	4	5	6
R_t	0	2	0	7.2	0	0
$\sum\gamma^t R_t$	0.0	1.0	1.0	1.9	1.9	1.9
$\sum\gamma^t$	1.0	1.5	1.75	1.875	1.9375	1.9687
ratio	0.0	0.6667	0.5714	1.0133	0.9806	0.9651

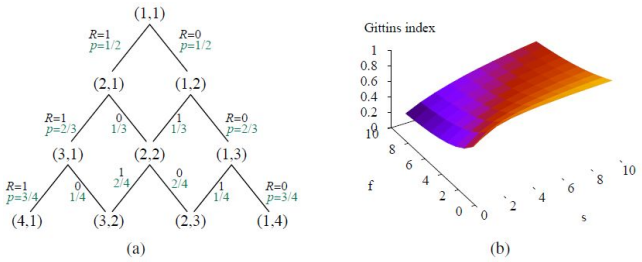
Bandit Problems

The Bernoulli bandit

- simplest and best-known instance of a bandit problem
- where each arm M_i produces a reward of 0 or 1 with a fixed but unknown probability μ_i.
- The state of arm M_i is defined by s_i and f_i, the counts of successes (1s) and failures (0s) so far for that arm;
- the transition probability predicts the next outcome to be 1 with probability $(s_i)/(s_i + f_i)$ and 0 with probability $(f_i)/(s_i + f_i)$.
- The counts are initialized to 1 so that the initial probabilities are 1/2 rather than 0 .

Bandit Problems

The Bernoulli bandit



(a) States, rewards, and transition probabilities for the Bernoulli bandit.
(b) Gittins indices for the states of the Bernoulli bandit process.

Partially Observable MDPs

- Partially observable MDPs (POMDPs)
- MDPs [the transition model $P(s' s, a)$, actions $A(s)$, and reward function $R(s, a, s')$]
- POMDPs are MDPs with ***sensor* model** $P(e|s)$.
- Obtain compact representations for large POMDPs by using dynamic decision networks
- We add sensor variables $\mathbf{E}_t$, assuming that the state variables $\mathbf{X}_t$ may not be directly observable.
- Thus the sensor model is $\boldsymbol{P(E_t|X_t)}$.

- Steps of decision cycle of a POMDP agent
  - Given the current belief state b, execute the action $a = \pi^*(b)$.
  - Observe the percept e.
  - Set the current belief state to FORWARD(b, a, e) and repeat.

Partially Observable MDPs

$$P(e|a,b) = \sum_{s'} P(e|a,s',b)P(s'|a,b)$$
$$= \sum_{s'} P(e|s')P(s'|a,b)$$
$$= \sum_{s'} P(e|s')\sum_{s} P(s'|s,a)b(s).$$

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

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

$P(b'|b, a)$ and $\rho(b, a)$ define an *observable* MDP on the space of belief states.

Algorithms for Solving POMDPs

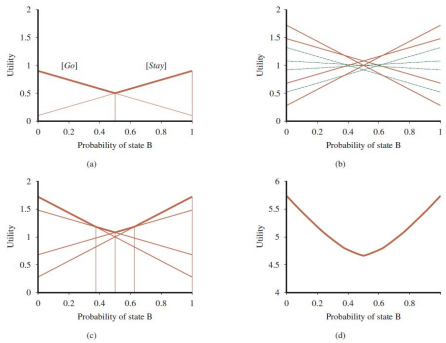
Value iteration for POMDPs

function POMDP-VALUE-ITERATION($pomdp, \epsilon$) **returns** a utility function
  inputs: $pomdp$, a POMDP with states S, actions $A(s)$, transition model $P(s' \mid s,a)$,
    sensor model $P(e \mid s)$, rewards $R(s,a,s')$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', sets of plans p with associated utility vectors α_p

  $U' \leftarrow$ a set containing all one-step plans $[a]$, with $\alpha_{[a]}(s) = \sum_{s'} P(s' \mid s,a)\, R(s,a,s')$
  repeat
    $U \leftarrow U'$
    $U' \leftarrow$ the set of all plans consisting of an action and, for each possible next percept,
      a plan in U with utility vectors computed according to Equation (16.18)
    $U' \leftarrow$ REMOVE-DOMINATED-PLANS(U')
  until MAX-DIFFERENCE(U, U') $\leq \epsilon(1-\gamma)/\gamma$
  return U

A high-level sketch of the value iteration algorithm for POMDPs. The REMOVE-DOMINATED-PLANS step and MAX-DIFFERENCE test are typically implemented as linear programs.

Algorithms for Solving POMDPs



(a) Utility of two one-step plans as a function of the initial belief state b(B) for the two-state world, with the corresponding utility function shown in bold. (b) Utilities for 8 distinct two-step plans. (c) Utilities for four undominated two-step plans. (d) Utility function for optimal eight-step plans.

Algorithms for Solving POMDPs

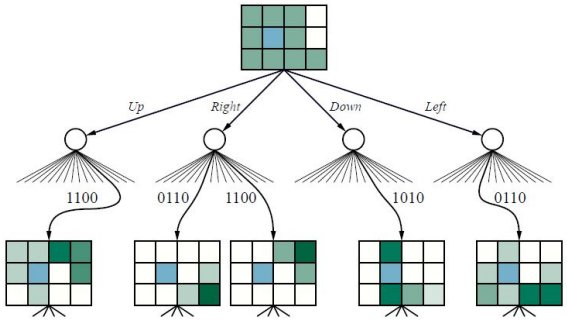
Online algorithms for POMDPs

- Starts with some prior belief state;
- It chooses an action based on some deliberation process centered on its current belief state;
- After acting, it receives an observation and updates its belief state using a filtering
- Algorithm; and the process repeats.

- Excpectimax algorithm (belief states rather than physical states as decision nodes)
- The chance nodes in the POMDP tree have branches labeled by possible observations and leading to the next belief state, with transition probabilities

- The combination of particle filtering and UCT applied to POMDPs goes under the name of partially observable Monte Carlo planning or **POMCP**.

Algorithms for Solving POMDPs



Part of an expectimax tree for the 43 POMDP with a uniform initial belief state.
The belief states are depicted with shading proportional to the probability of being in each location.

Summary

Sequential decision problems in stochastic environments, also called Markov decision processes, or MDPs, are defined by a transition model

The solution of an MDP is a policy that associates a decision with every state that the agent might reach.

The value iteration algorithm iteratively solves a set of equations relating the utility of each state to those of its neighbors

Policy iteration alternates between calculating the utilities of states under the current

Partially observable MDPs, or POMDPs, are much more difficult to solve than are
MDPs.