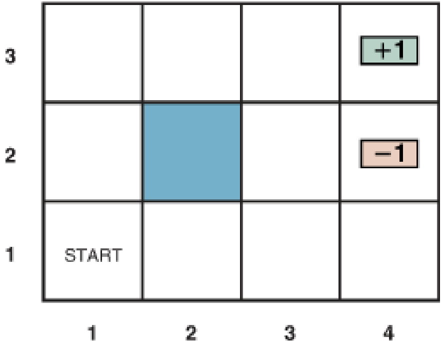
Markov Decision Processes

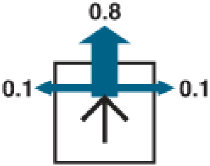
CSC 548, Artificial Intelligence II

Example: Grid World

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: actions to not always go as planned
  - 80% of the time the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the would have been takes, the agent does not move
- The agent receives rewards each time step
  - Small "living" reward each step (can be negative)
  - Big rewards come at the end (good or bad)
- Goal: maximize rewards

Grid World Actions





Markov Decision Processes

- A Markov Decision Process (MDP) is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function $T(s, a, s')$
    - Probability that a from s leads to s', that is, $P(s' \mid s, a)$
    - Also called the model or the dynamics
  - A reward function $R(s, a, s')$
    - Sometimes just $R(s)$ or $R(s')$
  - A start state
  - Maybe a terminal state
- MDPs are non-deterministic search problems
  - One way to solve them is with expectimax search

The Markov Assumption

- "Markov" generally means that given the present state, the future and past are independent
- For MDPs, "Markov" means action outcomes depend only on the current state

$$P(S_{t+1} = s' \mid S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, \ldots, S_0 = s_0)$$
$$=$$
$$P(S_{t+1} = s' \mid S_t = s_t, A_t = a_t)$$

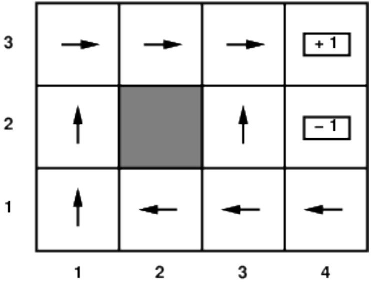
- This is just like search where the successor function could only depend on the current state (not the history)

Policies

- In deterministic single-agent search problems, we want an optimal plan, or sequence of actions from start to goal.
- For MDPs, we want an optimal policy $\pi^* : S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed
  - An explicit policy defines a reflex agent
- Expectimax did not compute entire policies
  - It computed the action for a single state only

Example: Grid World Policy

- Optimal policy when state penalty $R(s)$ is -0.04:



Utilities of Sequences

- What preferences should an agent have over reward sequences?
- More or less? [1,2,2] or [2,3,4]
- Now or later? [0,0,1] or [1,0,0]

Discounting

- It is reasonable to maximize the sum of rewards
- It is also reasonable to prefer rewards now to rewards later
- One solution: values of rewards decay exponentially
  - Discounted utility: $U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$

Discounting

- How to discount?
    - Each time we descend a level, we multiply in the discount once
- Why discount?
    - Sooner rewards probably have higher utility than later rewards
    - Also helps our algorithms converge
- Example: discount of 0.5
    - U([1,2,3]) = 1 * 1 + 0.5 * 2 + 0.25 * 3
    - U([1,2,3]) < U([3,2,1])

Stationary Preferences

- Theorem: if we assume stationary preferences:

$$[a_1, a_2, \ldots] \succ [b_1, b_2, \ldots] \Leftrightarrow [r, a_1, a_2, \ldots] \succ [r, b_1, b_2, \ldots]$$

- Then: there are only two ways to define utilities
  - Additive utility: $U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \ldots$
  - Discounted utility: $U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$

Infinite Utilities

- Problem: What if the game lasts forever? Do we get infinite rewards?
- Solutions:
  - Finite horizon: (similar to depth-limited search)
    - Terminate episodes after a fixed T steps (that is, life)
    - Gives nonstationary policies (π depends on time left)
  - Discounting: use $0 < \gamma < 1$

$$U([r_0, \ldots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq \frac{R_{max}}{1 - \gamma}$$

    - Smaller γ means smaller "horizon"
  - Absorbing state: guarantee that for every policy, a terminal state will evantually be reached

Recap: Defining MDPs

- A Markov Decision Processes:
  - Set of states S
  - Start state s_0
  - Set of actions A
  - Transitions $P(s' \mid s, a)$ (or $T(s, a, s')$)
  - Rewards $R(s, a, s')$ (and discount γ)
- MDP quantities so far:
  - Policy $=$ choice of action for each state
  - Utility $=$ sum of (discounted) rewards

Optimal Quantities

- The value (utility) of a state s:

  $V^*(s) =$ expected utility starting in s and acting optimally

- The value (utility) of a q-state (s, a):

  $Q^*(s, a) =$ expected utility starting out having taken action a from state s and (thereafter) acting optimally

- The optimal policy:

  $\pi^*(s) =$ optimal action from state s

Values of States

- Fundamental operation: compute the (expectimax) value of a state
  - Expected utility under optimal action
  - Average sum of (discounted) rewards
  - This is exactly what expectimax computed
- Recursive definition of value:

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

$$Q^*(s, a) = \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s')\right]$$

$$V^*(s) = \max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s')\right]$$

MDP Search Trees

- We do too much work with expectimax
- Potential Problem: States are repeated
  - Idea: only compute needed quantities once
- Potential Problem: Tree goes on forever
  - Idea: do a depth-limited computation, but with increasing depths until change is small
  - Note: deep parts of the tree eventually do not matter if $\gamma < 1$

Time-Limited Values

- Key idea: time-limited values
- Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps
  - Equivalently, it is what a depth-k expectimax would give from s

Value Iteration

- Start with $V_0(s) = 0$): no time steps left means an expected reward sum of zero
- Given a vector of $V_k(s)$ values, do one ply of expectimax from each state:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

- Repeat until convergence
- Complexity of each iteration $\mathcal{O}(S^2 A)$
- Theorem: will converge to unique optimal values
    - Basic idea: approximations get refined towards optimal values
    - Policy may converge long before the values do

Convergence

- How do we know the V_k vectors will converge?
- Case 1: If the tree has a maximum depth M, then V_M holds the actual untruncated values
- Case 2: If the discount is less than 1
  - Sketch: for any state V_k and V_{k+1} can be viewed as depth $k+1$ expectimax results in nearly identical search trees
  - The difference is that on the bottom layer, V_{k+1} has actual rewards while V_k has zeros
  - That last layer is at best all R_{MAX} and at worst R_{MIN}
  - But, everything is discounted by γ^k that far out
  - So, V_k and V_{k+1} are at most $\gamma^k max|R|$ different
  - So, as k increases, the values converge

Recap: Defining MDPs

- A Markov Decision Processes:
  - Set of states S
  - Start state s_0
  - Set of actions A
  - A set of actions $ainA$
  - Transitions $P(s' \mid s, a)$ (or $T(s, a, s')$)
  - Rewards $R(s, a, s')$ (and discount γ)$
- MDP quantities so far:
  - Policy = choice of action for each state
  - Utility = sum of (discounted) rewards
  - Values = expected future utility from a state (max node)
  - Q-Values = expected future utility from a q-state (chance node)

The Bellman Equations

- Definition of "optimal utility" via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values

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

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

- These are the Bellman equations and they characterize optimal values in a way that we will use repeatedly

Value Iteration

- The Bellman equations **characterize** the optimal values

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

- Value iteration **computes** the optimal values

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

- Value iteration is just a fixed point solution method
  - the V_k vectors are also interpretable as time-limited values

Fixed Policies

- Expectimax trees max over all actions to compute optimal values
- If we fixed some policy $\pi(s)$, then the tree would be simpler, that is, only one action per state
- But, the tree's value would depend on which policy we fixed

Utilities for a Fixed Policy

- Another basic operation: compute the utility of a state s under a fixed (generally non-optimal) policy
- Define the utility of a state s under a fixed policy π
- Recursive relation (one-step look-head / Bellman equations):

$$V^{\pi}(s) = \sum_{s'} T(s, a, s') \left[R(s, \pi(s), s') + \gamma V^{\pi}(s') \right]$$

Policy Evaluation

- How do we calculate the Vs for a fixed policy π?
- Idea 1: turn recursive Bellman equations into updates (like value iteration)

$$V_0^\pi(s) = V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,\pi(s),s') + \gamma V^\pi(s')\right]$$

- Efficiency: $\mathcal{O}(S^2)$ per iteration
- Idea 2: Without the maxes, the Bellman equations are just a linear system

Computing Actions from Values

- Let us assume we have the optimal values $V^*(s)$
- How should we act? (not obvious)
- We need to do a mini-expectimax (one step)

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s')\right]$$

- This is called policy extraction, since it gets the policy implied by the values

Computing Actions from Values

- Let us assume we have the optimal q-values
- How should we act? (trivial to decide)

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

- Important lesson: actions are easier to select from q-values than values

Problems with Value Iteration

- Value iteration repeats the Bellman updates

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

- Problem 1: it is slow, $\mathcal{O}(S^2A)$ per iteration
- Problem 2: the “max” at each state rarely changed
- Problem 3: the policy often converges long before the values

Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We do not track the policy, but taking the max over the actions implicitly recomputes it
- In policy iteration:
  - We do several passes that update the utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like value iteration)
  - The new policy will be better (or we are done)
- Both are dynamic programs for solving MDPs

Summary: MDP Algorithms

- So you want to…
  - Compute optimal values: use value iteration or policy iteration
  - Compute values for a particular policy: use policy evaluation
  - Turn your values into a policy: use policy extraction (one-step lookahead)
- These all look the same
  - They basically are – they are all variations of Bellman updates
  - They all use one-step lookahead expectimax fragments
  - They differ only in whether we plug in a fixed policy or max over actions