Outline

- Zero-sum deterministic two player games
  - Minimax
  - Evaluation functions for non-terminal states
  - Alpha-Beta pruning
- Stochastic games
  - Single player: expectimax
  - Two player: expectiminimax
- Non-zero sum
- Markov decision processes (MDPs)
Minimax Example

Speeding Up Game Tree Search

- Evaluation functions for non-terminal states

- Pruning: not search parts of the tree
  - Alpha-Beta pruning does so without losing accuracy, $O(b^d) \rightarrow O(b^{d/2})$
Pruning

Alpha-Beta Pruning

- General configuration
  - We’re computing the MIN-VALUE at $n$
  - We’re looping over $n$’s children
  - $n$’s value estimate is dropping
  - $a$ is the best value that MAX can get at any choice point along the current path
  - If $n$ becomes worse than $a$, MAX will avoid it, so can stop considering $n$’s other children
  - Define $b$ similarly for MIN
Alpha-Beta Pruning Example

a is MAX’s best alternative here or above
b is MIN’s best alternative here or above

Starting a/b

Raising a

Lowering b

Raising a

a is MAX’s best alternative here or above
b is MIN’s best alternative here or above
Alpha-Beta Pseudocode

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow -\infty$
    for $a, s$ in SUCCESSORS(state) do $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$
    return $v$

function MAX-VALUE(state, $\alpha$, $\beta$) returns a utility value
    inputs: state, current state in game
    $\alpha$, the value of the best alternative for MAX along the path to state
    $\beta$, the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow -\infty$
    for $a, s$ in SUCCESSORS(state) do
        $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$
        if $v \geq \beta$ then return $v$
        $\alpha \leftarrow \text{MAX}(\alpha, v)$
    return $v$

---

Alpha-Beta Pruning Properties

- This pruning has no effect on final result at the root
- Values of intermediate nodes might be wrong!
- Good child ordering improves effectiveness of pruning
  - Heuristic: order by evaluation function or based on previous search
- With “perfect ordering”:
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless...
- This is a simple example of metareasoning (computing about what to compute)
Outline

- Zero-sum deterministic two player games
  - Minimax
  - Evaluation functions for non-terminal states
  - Alpha-Beta pruning
- Stochastic games
  - Single player: expectimax
  - Two player: expectiminimax
- Non-zero sum
- Markov decision processes (MDPs)

Expectimax Search Trees

- What if we don’t know what the result of an action will be? E.g.,
  - In solitaire, next card is unknown
  - In minesweeper, mine locations
  - In pacman, the ghosts act randomly
- Can do expectimax search to maximize average score
  - Chance nodes, like min nodes, except the outcome is uncertain
  - Calculate expected utilities
  - Max nodes as in minimax search
  - Chance nodes take average (expectation) of value of children
- Later, we’ll learn how to formalize the underlying problem as a Markov Decision Process
**Expectimax Pseudocode**

```python
def value(s):
    if s is a max node return maxValue(s)
    if s is an exp node return expValue(s)
    if s is a terminal node return evaluation(s)

def maxValue(s):
    values = [value(s') for s' in successors(s)]
    return max(values)

def expValue(s):
    values = [value(s') for s' in successors(s)]
    weights = [probability(s, s') for s' in successors(s)]
    return expectation(values, weights)
```

**Expectimax Quantities**
Expectimax Pruning?

- Chance nodes
  - Chance nodes are like min nodes, except the outcome is uncertain
  - Calculate expected utilities
  - Chance nodes average successor values (weighted)
- Each chance node has a probability distribution over its outcomes (called a model)
  - For now, assume we’re given the model
- Utilities for terminal states
  - Static evaluation functions give us limited-depth search

Estimate of true expectimax value (which would require a lot of work to compute)
Expectimax for Pacman

- Notice that we’ve gotten away from thinking that the ghosts are trying to minimize pacman’s score.
- Instead, they are now a part of the environment.
- Pacman has a belief (distribution) over how they will act.
- Quiz: Can we see minimax as a special case of expectimax?
- Quiz: what would pacman’s computation look like if we assumed that the ghosts were doing 1-ply minimax and taking the result 80% of the time, otherwise moving randomly?
- If you take this further, you end up calculating belief distributions over your opponents’ belief distributions over your belief distributions, etc…
  - Can get unmanageable very quickly!
Expectimax Utilities

- For minimax, terminal function scale doesn’t matter
  - We just want better states to have higher evaluations (get the ordering right)
  - We call this insensitivity to monotonic transformations

- For expectimax, we need magnitudes to be meaningful

Stochastic Two-Player

- E.g. backgammon
- Expectiminimax (!)
  - Environment is an extra player that moves after each agent
  - Chance nodes take expectations, otherwise like minimax

```python
if state is a Max node then
  return the highest ExpectiMinimax-Value of Successors(state)
if state is a Min node then
  return the lowest ExpectiMinimax-Value of Successors(state)
if state is a chance node then
  return average of ExpectiMinimax-Value of Successors(state)
```
Stochastic Two-Player

- Dice rolls increase $b$: 21 possible rolls with 2 dice
  - Backgammon = 20 legal moves
  - Depth $2 = 20 \times (21 \times 20)^3 = 1.2 \times 10^9$
- As depth increases, probability of reaching a given search node shrinks
  - So usefulness of search is diminished
  - So limiting depth is less damaging
  - But pruning is trickier...
- TDGammon uses depth-2 search + very good evaluation function + reinforcement learning: world-champion level play
- 1st AI world champion in any game!

Non-Zero-Sum Utilities

- Similar to minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own utility and propagate (or back up) nodes from children
  - Can give rise to cooperation and competition dynamically...
Outline

- Zero-sum deterministic two player games
  - Minimax
  - Evaluation functions for non-terminal states
  - Alpha-Beta pruning
- Stochastic games
  - Single player: expectimax
  - Two player: expectiminimax
- Non-zero sum
- Markov decision processes (MDPs)

Reinforcement Learning

- Basic idea:
  - Receive feedback in the form of rewards
  - Agent’s utility is defined by the reward function
  - Must learn to act so as to maximize expected rewards
Grid World

- The agent lives in a grid
- Walls block the agent's path
- The agent's actions do 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 agent would have been taken, the agent stays put
- Small “living” reward each step
- Big rewards come at the end
- Goal: maximize sum of rewards

Grid Futures

<table>
<thead>
<tr>
<th>Deterministic Grid World</th>
<th>Stochastic Grid World</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Deterministic Grid" /></td>
<td><img src="image2" alt="Stochastic Grid" /></td>
</tr>
</tbody>
</table>

- ![Deterministic Grid Futures](image3)
- ![Stochastic Grid Futures](image4)
Markov Decision Processes

- An MDP is defined by:
  - A set of states \( s \in S \)
  - A set of actions \( a \in A \)
  - A transition function \( T(s, a, s') \)
    - Prob that \( a \) from \( s \) leads to \( s' \)
    - i.e., \( P(s' | s, a) \)
    - Also called the model
  - A reward function \( R(s, a, s') \)
    - Sometimes just \( R(s) \) or \( R(s') \)
  - A start state (or distribution)
  - Maybe a terminal state

- MDPs are a family of non-deterministic search problems
  - Reinforcement learning: MDPs where we don’t know the transition or reward functions

What is Markov about MDPs?

- Andrey Markov (1856-1922)

- “Markov” generally means that given the present state, the future and the past are independent

- For Markov decision processes, “Markov” means:
  \[
P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}, \ldots, S_0 = s_0) = P(S_{t+1} = s' | S_t = s_t, A_t = a_t)
  \]
In deterministic single-agent search problems, we want an optimal plan, or sequence of actions, from start to a goal. In an MDP, we want an optimal policy $\pi^*: S \rightarrow A$:

- A policy $\pi$ gives an action for each state.
- An optimal policy maximizes expected utility if followed.
- Defines a reflex agent.

Optimal policy when $R(s, a, s') = -0.03$ for all non-terminals $s$.

Example Optimal Policies:

- $R(s) = -0.01$
- $R(s) = -0.03$
- $R(s) = -0.4$
- $R(s) = -2.0$
Example: High-Low

- Three card types: 2, 3, 4
- Infinite deck, twice as many 2's
- Start with 3 showing
- After each card, you say “high” or “low”
- New card is flipped
- If you’re right, you win the points shown on the new card
- Ties are no-ops
- If you’re wrong, game ends

Differences from expectimax:
- #1: get rewards as you go
- #2: you might play forever!

High-Low as an MDP

- States: 2, 3, 4, done
- Actions: High, Low
- Model: $T(s, a, s')$:
  - $P(s'=4 \mid 4, \text{Low}) = 1/4$
  - $P(s'=3 \mid 4, \text{Low}) = 1/4$
  - $P(s'=2 \mid 4, \text{Low}) = 1/2$
  - $P(s'=\text{done} \mid 4, \text{Low}) = 0$
  - $P(s'=4 \mid 4, \text{High}) = 1/4$
  - $P(s'=3 \mid 4, \text{High}) = 0$
  - $P(s'=2 \mid 4, \text{High}) = 0$
  - $P(s'=\text{done} \mid 4, \text{High}) = 3/4$
  - ...
- Rewards: $R(s, a, s')$:
  - Number shown on $s'$ if $s \neq s'$ and $a$ is “correct”
  - 0 otherwise
- Start: 3
Example: High-Low

![Diagram of MDP Search Trees]

MDP Search Trees

- Each MDP state gives an expectimax-like search tree

\[
T(s, a, s') = P(s'|s, a) \\
R(s, a, s')
\]

(s, a) is a q-state

(s, a, s') called a transition

s is a state
Utilities of Sequences

- In order to formalize optimality of a policy, need to understand utilities of sequences of rewards
- Typically consider stationary preferences:

\[ [r, r_0, r_1, r_2, \ldots] \succ [r, r'_0, r'_1, r'_2, \ldots]\]

\[ [r_0, r_1, r_2, \ldots] \succ [r'_0, r'_1, r'_2, \ldots] \]

- Theorem: only two ways to define stationary utilities
  - Additive utility:
    \[ U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots \]
  - Discounted utility:
    \[ U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots \]

Infinite Utilities?!

- Problem: infinite state sequences have infinite rewards
- Solutions:
  - Finite horizon:
    - Terminate episodes after a fixed T steps (e.g. life)
    - Gives nonstationary policies (π depends on time left)
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like “done” for High-Low)
  - Discounting: for \( 0 < \gamma < 1 \)

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

- Smaller \( \gamma \) means smaller “horizon” – shorter term focus
Discounting

- Typically discount rewards by $\gamma < 1$ each time step
  - Sooner rewards have higher utility than later rewards
  - Also helps the algorithms converge

Recap: Defining MDPs

- Markov decision processes:
  - States $S$
  - Start state $s_0$
  - Actions $A$
  - Transitions $P(s'|s,a)$ (or $T(s,a,s')$)
  - Rewards $R(s,a,s')$ (and discount $\gamma$)

- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility (or return) = sum of discounted rewards
Optimal Utilities

- Fundamental operation: compute the values (optimal expectimax utilities) of states \( s \).
- Why? Optimal values define optimal policies!
- Define the value of a state \( s \):
  \( V^*(s) = \text{expected utility starting in } s \text{ and acting optimally} \)
- Define the value of a q-state \((s,a)\):
  \( Q^*(s,a) = \text{expected utility starting in } s, \text{ taking action } a, \text{ and thereafter acting optimally} \)
- Define the optimal policy:
  \( \pi^*(s) = \text{optimal action from state } s \)

The Bellman Equations

- Definition of “optimal utility” leads to a simple one-step lookahead relationship amongst optimal utility values:
  \( \text{Optimal rewards = maximize over first action and then follow optimal policy} \)
- Formally:
  \[
  V^*(s) = \max_a Q^*(s,a) \\
  Q^*(s,a) = \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V^*(s') \right] \\
  V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V^*(s') \right]
  \]
Value Estimates

- Calculate estimates $V_k^*(s)$
  - Not the optimal value of $s$!
  - The optimal value considering only next $k$ time steps (k rewards)
  - As $k \to \infty$, it approaches the optimal value

- Almost solution: recursion (i.e. expectimax)
- Correct solution: dynamic programming