Expectimax Search

Hanna Hajishirzi

Based on slides from Dan Klein, Luke Zettlemoyer
Many slides over the course adapted from either Stuart Russell or Andrew Moore
Overview:
Search
Search Problems

Pancake Example:
State space graph with costs as weights
function TREE-SEARCH(problem, strategy) returns a solution, or failure
initialize the search tree using the initial state of problem
loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the corresponding solution
    else expand the node and add the resulting nodes to the search tree
end

Path to reach goal: Flip four, flip three
Total cost: 7
Search Strategies

- **Uninformed Search algorithms:**
  - Depth First Search
  - Breath First Search
  - Uniform Cost Search: select smallest \( g(n) \)

- **Heuristic Search:**
  - Best First Search: select smallest \( h(n) \)
  - A* Search: select smallest \( f(n) = g(n) + h(n) \)

- **Graph Search**
Which Algorithm?
Which Algorithm?
Optimal A* Tree Search

- A* tree search is optimal if $h$ is admissible

- A heuristic $h$ is admissible (optimistic) if:
  \[
  h(n) \leq h^*(n)
  \]
  where $h^*(n)$ is the true cost to a nearest goal
Optimal A* Graph Search

- A* graph search is optimal if h is consistent

```
\[
g = 10
\]

Consistency for all edges (A,a,B):
- \( h(A) \leq c(A,a,B) + h(B) \)

Triangular inequality
Which Algorithm?
Overview: Adversarial Search
Value of a state: The best achievable outcome (utility) from that state

Non-Terminal States:
\[ V(s) = \max_{s' \in \text{children}(s)} V(s') \]

Terminal States:
\[ V(s) = \text{known} \]
Adversarial Game Tree

States Under Agent’s Control:
\[ V(s) = \max_{s' \in \text{successors}(s)} V(s') \]

States Under Opponent’s Control:
\[ V(s') = \min_{s \in \text{successors}(s')} V(s) \]

Terminal States:
\[ V(s) = \text{known} \]
Minimax Example
Minimax Properties

- **Optimal?**
  - Yes, against perfect player. Otherwise?

- **Time complexity?**
  - $O(b^m)$

- **Space complexity?**
  - $O(bm)$

- **For chess, $b \approx 35, m \approx 100$**
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?
Today

- Adversarial Search
  - Alpha-beta pruning
  - Evaluation functions
  - Expectimax

- Reminder:
  - Programming 1 due in one week!
  - Programming 2 will be on adversarial search
α is MAX’s best alternative here or above
β is MIN’s best alternative here or above
Alpha-Beta Pruning Example

\( \alpha \) is MAX’s best alternative here or above

\( \beta \) is MIN’s best alternative here or above

\( \leq 3 \)
Alpha-Beta Pruning Example

\[ \alpha \text{ is MAX's best alternative here or above} \]
\[ \beta \text{ is MIN's best alternative here or above} \]
α is MAX’s best alternative here or above
β is MIN’s best alternative here or above
Alpha-Beta Pruning Example

α is MAX’s best alternative here or above
β is MIN’s best alternative here or above
Alpha-Beta Pruning Properties

- This pruning has no effect on final result at the root

- Values of intermediate nodes might be wrong!
  - but, they are bounds

- Good child ordering improves effectiveness of pruning

- With “perfect ordering”:
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless…
Resource Limits

- Cannot search to leaves
- Depth-limited search
  - Instead, search a limited depth of tree
  - Replace terminal utilities with an eval function for non-terminal positions
    - e.g., $\alpha$-$\beta$ reaches about depth 8 – decent chess program
- Guarantee of optimal play is gone
- Evaluation function matters
  - It works better when we have a greater depth look ahead
Depth Matters

depth 2
Depth Matters

Score: 0

Depth 10
Evaluation Functions

- Function which scores non-terminals

\[
Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)
\]

- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features:
  - e.g. \( f_1(s) = \text{(num white queens} - \text{num black queens}) \), etc.
Bad Evaluation Function
Why Pacman Starves

- He knows his score will go up by eating the dot now
- He knows his score will go up just as much by eating the dot later on
- There are no point-scoring opportunities after eating the dot
- Therefore, waiting seems just as good as eating
What features would be good for Pacman?

$$Eval(s) = w_1f_1(s) + w_2f_2(s) + \ldots + w_n f_n(s)$$
Evaluation Function
Evaluation Function
Minimax Example

No point in trying
Expectimax

3 ply look ahead, ghosts move randomly

Wins some of the games
Worst-case vs. Average Case

Uncertain outcomes are controlled by chance not an adversary

- Chance nodes are new types of nodes (instead of Min nodes)
Stochastic Single-Player

- What if we don’t know what the result of an action will be? E.g.,
  - In solitaire, shuffle is unknown
  - In minesweeper, mine locations

- Can do *expectimax search*
  - Chance nodes, like actions except the environment controls the action chosen
  - Max nodes as before
  - Chance nodes take average (expectation) of value of children
Expectimax Pseudocode

def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v

v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10
Maximum Expected Utility

- Why should we average utilities? Why not minimax?

- Principle of maximum expected utility: an agent should choose the action which maximizes its expected utility, given its knowledge
  - General principle for decision making
  - Often taken as the definition of rationality
  - We’ll see this idea over and over in this course!

- Let’s decompress this definition…
A random variable represents an event whose outcome is unknown
A probability distribution is an assignment of weights to outcomes

Example: traffic on freeway?
- Random variable: $T = \text{whether there’s traffic}$
- Outcomes: $T \in \{\text{none, light, heavy}\}$
- Distribution: $P(T=\text{none}) = 0.25$, $P(T=\text{light}) = 0.55$, $P(T=\text{heavy}) = 0.20$

Some laws of probability (more later):
- Probabilities are always non-negative
- Probabilities over all possible outcomes sum to one

As we get more evidence, probabilities may change:
- $P(T=\text{heavy}) = 0.20$, $P(T=\text{heavy} \mid \text{Hour=8am}) = 0.60$
- We’ll talk about methods for reasoning and updating probabilities later
What are Probabilities?

- **Objectivist / frequentist answer:**
  - Averages over repeated experiments
  - E.g. empirically estimating $P(\text{rain})$ from historical observation
  - E.g. pacman’s estimate of what the ghost will do, given what it has done in the past
  - Assertion about how future experiments will go (in the limit)
  - Makes one think of *inherently random* events, like rolling dice

- **Subjectivist / Bayesian answer:**
  - Degrees of belief about unobserved variables
  - E.g. an agent’s belief that it’s raining, given the temperature
  - E.g. pacman’s belief that the ghost will turn left, given the state
  - Often *learn* probabilities from past experiences (more later)
  - New evidence *updates beliefs* (more later)
Uncertainty Everywhere

- Not just for games of chance!
  - I’m sick: will I sneeze this minute?
  - Email contains “FREE!”: is it spam?
  - Tooth hurts: have cavity?
  - 60 min enough to get to the airport?
  - Robot rotated wheel three times, how far did it advance?
  - Safe to cross street? (Look both ways!)

- Sources of uncertainty in random variables:
  - Inherently random process (dice, etc)
  - Insufficient or weak evidence
  - Ignorance of underlying processes
  - Unmodeled variables
  - The world’s just noisy – it doesn’t behave according to plan!
Reminder: Expectations

- We can define function $f(X)$ of a random variable $X$
- The expected value of a function is its average value, weighted by the probability distribution over inputs
- Example: How long to get to the airport?
  - Length of driving time as a function of traffic:
    - $L(\text{none}) = 20$, $L(\text{light}) = 30$, $L(\text{heavy}) = 60$
  - What is my expected driving time?
    - Notation: $E_{P(T)}[L(T)]$
    - Remember, $P(T) = \{\text{none}: 0.25, \text{light}: 0.5, \text{heavy}: 0.25\}$
    - $E[L(T)] = L(\text{none}) \times P(\text{none}) + L(\text{light}) \times P(\text{light}) + L(\text{heavy}) \times P(\text{heavy})$
    - $E[L(T)] = (20 \times 0.25) + (30 \times 0.5) + (60 \times 0.25) = 35$
Review: Expectations

- Real valued functions of random variables:
  \[ f : X \rightarrow R \]

- Expectation of a function of a random variable
  \[ E_{P(X)}[f(X)] = \sum_x f(x)P(x) \]

- Example: Expected value of a fair die roll

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P )</th>
<th>( f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1/6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1/6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1/6</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>1/6</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>1/6</td>
<td>6</td>
</tr>
</tbody>
</table>

\[
1 \times \frac{1}{6} + 2 \times \frac{1}{6} + 3 \times \frac{1}{6} + 4 \times \frac{1}{6} + 5 \times \frac{1}{6} + 6 \times \frac{1}{6} = 3.5
\]
Utilities

- Utilities are functions from outcomes (states of the world) to real numbers that describe an agent’s preferences.

- Where do utilities come from?
  - In a game, may be simple (+1/-1)
  - Utilities summarize the agent’s goals
  - Theorem: any set of preferences between outcomes can be summarized as a utility function (provided the preferences meet certain conditions)

- In general, we hard-wire utilities and let actions emerge (why don’t we let agents decide their own utilities?)

- More on utilities soon…
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**
  - 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 Search

- In expectimax search, we have a probabilistic model of how the opponent (or environment) will behave in any state
  - Model could be a simple uniform distribution (roll a die)
  - Model could be sophisticated and require a great deal of computation
  - We have a node for every outcome out of our control: opponent or environment
  - The model might say that adversarial actions are likely!
- For now, assume for any state we magically have a distribution to assign probabilities to opponent actions / environment outcomes
Expectimax Pruning
Expectimax Pruning

- **Not easy**
  - exact: need bounds on possible values
  - approximate: sample high-probability branches
Depth-limited Expectimax

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

- Evaluation functions quickly return an estimate for a node’s true value (which value, expectimax or minimax?)
- For minimax, evaluation 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
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 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?
Let’s say you know that your opponent is actually running a depth 2 minimax, using the result 80% of the time, and moving randomly otherwise

Question: What tree search should you use?

Answer: Expectimax!

To figure out EACH chance node’s probabilities, you have to run a simulation of your opponent

This kind of thing gets very slow very quickly

Even worse if you have to simulate your opponent simulating you...

... except for minimax, which has the nice property that it all collapses into one game tree
## Expectimax for Pacman

### Results from playing 5 games

<table>
<thead>
<tr>
<th></th>
<th>Minimizing Ghost</th>
<th>Random Ghost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Minimax Pacman</strong></td>
<td>Won 5/5</td>
<td>Won 5/5</td>
</tr>
<tr>
<td></td>
<td>Avg. Score: 493</td>
<td>Avg. Score: 483</td>
</tr>
<tr>
<td><strong>Expectimax Pacman</strong></td>
<td>Won 1/5</td>
<td>Won 5/5</td>
</tr>
<tr>
<td></td>
<td>Avg. Score: -303</td>
<td>Avg. Score: 503</td>
</tr>
</tbody>
</table>

Pacman does depth 4 search with an eval function that avoids trouble. Minimizing ghost does depth 2 search with an eval function that seeks Pacman.
Mixed Layer Types

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

```plaintext
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 \( \approx \) 20 legal moves
  - Depth 4 = 20 \( \times (21 \times 20)^3 \) \( \approx 1.2 \times 10^9 \)
- As depth increases, probability of reaching a given node shrinks
  - So value of lookahead is diminished
  - So limiting depth is less damaging
  - But pruning is less possible...
- TDGammon uses depth-2 search + very good eval function + reinforcement learning: world-champion level play
Multi-player Non-Zero-Sum Games

- Similar to minimax:
  - Utilities are now tuples
  - Each player maximizes their own entry at each node
  - Propagate (or back up) nodes from children
  - Can give rise to cooperation and competition dynamically…