CS 4700: Foundations of Artificial Intelligence

Spring 2020
Prof. Haym Hirsh

Lecture 7
February 5, 2020
Reminder: Technology Policy

No technology except for first four rows of left and right sides
“Primer”: Friday 5pm in Gates G01

Covers:
- Jupyter Notebooks and .ipynb files
- Google Colab (online Jupyter Notebook environment)
- Useful Python features

Bring your laptop
Recap

• Uninformed search
  • Depth-first search
  • Breadth-first search
  • Iterative deepening search

• Informed search
  • Best-first search
  • A*
  • Hill climbing
  • Simulated annealing

• Genetic algorithms/evolutionary computation
Other Search Topics (not covered - yet)

- Continuous spaces / gradient-based methods
- Non-deterministic actions
  - Oops, I dropped the cup
- Partially observable states
  - What’s behind me?
Other Search Topics (not covered - yet)

- Continuous spaces / gradient-based methods
- Non-deterministic actions
  - Oops, I dropped the cup
- Partially observable states
  - What’s behind me?

Will come up in machine learning
Other Search Topics (not covered - yet)

• Continuous spaces / gradient-based methods
  Will come up in machine learning
• Non-deterministic actions
  • Oops, I dropped the cup
  Will come up in reinforcement learning
• Partially observable states
  • What’s behind me?
Adversarial Search
(Playing Games)

Chapter 5
Two Player Zero-Sum Games

- Tic-Tac-Toe
- Chess
- Go
Tic Tac Toe
Tic Tac Toe

(Still using states and actions formulation)
Tic Tac Toe
Tic Tac Toe
Tic Tac Toe

My turn

Opponent’s turn

My turn

Opponent’s turn

TERMINAL

-1 0 1
Tic Tac Toe
Tic Tac Toe

\[ f(a) \quad f(b) \quad f(c) \]
Tic Tac Toe

\[
\min(f(a), f(b), f(c))
\]
Tic Tac Toe

\[
\begin{align*}
\text{MAX (x)} & \quad \text{MIN (o)} \\
\min(f(a), f(b), f(c)) & \quad \text{max(successors)} \\
\text{MAX (x)} & \quad \text{...} \\
\end{align*}
\]

f(a) \quad f(b) \quad f(c)
Adversarial Search

(Initial) assumptions:

• On my turn I consider all moves and pick the one that’s best for me
• My opponent will consider all moves and the pick the one that’s worst for me

\[ f_{\text{opponent}}(s) = -f_{\text{me}}(s) \]
Adversarial Search

(Initial) assumptions:
• On my turn I consider all moves and pick the one that’s best for me
• My opponent will consider all moves and the pick the one that’s worst for me

\[ f_{\text{opponent}}(s) = - f_{\text{me}}(s) \quad \text{“zero sum”} \]
Adversarial Search

(Initial) assumptions:

• On my turn I consider all moves and pick the one that’s best for me.
• My opponent will consider all moves and the pick the one that’s worst for me.

\[ f_{\text{opponent}}(s) = - f_{\text{me}}(s) \quad \text{“zero sum”} \]

We’ll use \( f(s) \) whenever we’re talking about \( f_{\text{me}}(s) \), and \(-f(s)\) whenever we’re talking about \( f_{\text{opponent}}(s) \).
Adversarial Search

(Initial) assumptions:
• On my turn I consider all moves and pick the one that’s best for me
• My opponent will consider all moves and the pick the one that’s worst for me

\[ f_{\text{opponent}}(s) = -f_{\text{me}}(s) \quad \text{“zero sum”} \]

We’ll use \( f(s) \) whenever we’re talking about \( f_{\text{me}}(s) \), and
\[-f(s) \text{ whenever we’re talking about } f_{\text{opponent}}(s) \]

My turn: \( \max f(s) \) for all successors \( s \) (pick my best move)
Opponent’s turn: \( \min f(s) \) for all successors \( s \) (pick my worst move)
Minimax Value of a Game

“Ground truth”

\( V(s): \) What the true value of a game state is

Given for terminal nodes

Computed from non-terminal nodes
Minimax Value of a Game

- I win = $+\infty$
- I lose = $-\infty$

My Turn

Terminal Nodes

+\infty  -\infty  -\infty
Minimax Value of a Game

- I win = $+\infty$
- I lose = $-\infty$

My Turn

Terminal Nodes
Minimax Value of a Game

- $V(s) = \text{value of win (to me)}$

My Turn

Terminal Nodes

- +2
- -3
- -4
Minimax Value of a Game

- $V(s) =$ value of win (to me)

My Turn

Terminal Nodes
Minimax Value of a Game

• \( V(s) = \text{value of win (to me)} \)

My Opponent’s Turn

Terminal Nodes

+2  -3  -4
Minimax Value of a Game

- \( V(s) = \text{value of win (to me)} \)

My Opponent’s Turn

Terminal Nodes

-4
+2
-3
-4
Minimax Value of a Game

My Turn
\[ V(s) = \text{Max of successors} \]

My Opponent’s Turn
\[ V(s) = \text{Min of successors} \]
Minimax Value of a Game

- My Turn
  \[ V(s) = \text{Max of successors} \]

- My Opponent’s Turn
  \[ V(s) = \text{Min of successors} \]
Minimax Value of a Game

My Turn
\[ V(s) = \text{Max of successors} \]

My Opponent’s Turn
\[ V(s) = \text{Min of successors} \]
Minimax Value of a Game

My Turn
\[ V(s) = \text{Max of successors} \]

My Opponent’s Turn
\[ V(s) = \text{Min of successors} \]
Minimax Value of a Game

My Turn
\[ V(s) = \text{Max of successors} \]

My Opponent’s Turn
\[ V(s) = \text{Min of successors} \]

The diagram illustrates the minimax values of a game tree. At each node, the values represent the outcomes (+ or -), with red circles indicating my turn and green circles indicating my opponent’s turn. The values at the root node represent the overall values for the game scenario.
Minimax Value of a Game

• Current state: s
• Available operators: ops
• Value of a terminal state: V(s)

• My turn:
  • Value of state \( s = V(s) = \max_{o \in \text{ops}} \{ V(\text{apply}(s, o)) \} \)
  • Best move = \( \arg\max_{o \in \text{ops}} \{ V(\text{apply}(s, o)) \} \)

• Opponent’s turn:
  • Value of state \( s = V(s) = \min_{o \in \text{ops}} \{ V(\text{apply}(s, o)) \} \)
  • Best move = \( \arg\min_{o \in \text{ops}} \{ V(\text{apply}(s, o)) \} \)
Minimax Value of a Game

My Turn
V(s) = Max of successors

My Opponent’s Turn
V(s) = Min of successors
Minimax Value of a Game

My Turn
\[ V(s) = \text{Max of successors} \]

My Opponent’s Turn
\[ V(s) = \text{Min of successors} \]

Continue until terminal nodes are reached
My Turn
\[ V(s) = \text{Max of successors} \]

My Opponent’s Turn
\[ V(s) = \text{Min of successors} \]

Continue until terminal nodes are reached
My Turn
\[ V(s) = \text{Max of successors} \]

My Opponent’s Turn
\[ V(s) = \text{Min of successors} \]

Continue until terminal nodes are reached
My Turn
\[ V(s) = \text{Max of successors} \]

My Opponent’s Turn
\[ V(s) = \text{Min of successors} \]

Continue until terminal nodes are reached
Minimax Value of a Game

- **My Turn**
  - $V(s) = \text{Max of successors}$

- **My Opponent’s Turn**
  - $V(s) = \text{Min of successors}$

Continue until terminal nodes are reached
Minimax Algorithm

Initial call:
• My turn: \( \text{maxturn(} \text{initial-state,ops)} \)
• Opponent’s turn: \( \text{minturn(} \text{initial-state,ops)} \)
Minimax Algorithm

maxturn(s,ops):
  if terminal(s) then return V(s)
  else
    val ← -∞;
    foreach o ∈ ops
      val’ ← minturn(apply(s,o),ops);
      if val’ > val then val ← val’; besto ← o;
    return val
Minimax Algorithm

maxturn(s,ops):

if terminal(s) then return V(s)
else
    val ← -∞;
    foreach o ∈ ops
        val’ ← minturn(apply(s,o),ops);
        if val’ > val then val ← val’; bestop ← o;
    return val

Shorthand for return a data structure with the value, the operator that took you here, possibly other info
Minimax Algorithm

$max\text{turn}(s,ops)$:

if terminal(s) then return $V(s)$
else
    $val \leftarrow -\infty$;
    foreach $o \in ops$
        $val' \leftarrow minturn(apply(s,o),ops)$;
        if $val' > val$ then
            $val \leftarrow val'$;
            bestop \leftarrow o;
    return $val$

$min\text{turn}(s,ops)$:

if terminal(s) then return $V(s)$
else
    $val \leftarrow +\infty$;
    foreach $o \in ops$
        $val' \leftarrow max\text{turn}(apply(s,o),ops)$;
        if $val' < val$ then
            $val \leftarrow val'$;
            bestop \leftarrow o;
    return $val$
Minimax Algorithm
(Complete Search)

maxturn(s,ops):
if terminal(s) then return V(s)
else
    val ← -∞;
    foreach o ∈ ops
        val’ ← minturn(apply(s,o),ops);
        if val’ > val then
            val ← val’;
            bestop ← o;
    return val

minturn(s,ops):
if terminal(s) then return V(s)
else
    val ← +∞;
    foreach o ∈ ops
        val’ ← maxturn(apply(s,o),ops);
        if val’ < val then
            val ← val’;
            bestop ← o;
    return val
Minimax Algorithm
(Complete Search)

maxturn(s,ops):
    if terminal(s) then return V(s)
    else
        val ← -∞;
        foreach o ∈ ops
            val’ ← minturn(apply(s,o),ops);
            if val’ > val then
                val ← val’;
                bestop ← o;
        return val

minturn(s,ops):
    if terminal(s) then return V(s)
    else
        val ← +∞;
        foreach o ∈ ops
            val’ ← maxturn(apply(s,o),ops);
            if val’ < val then
                val ← val’;
                bestop ← o;
        return val
Minimax Search

• Complete search:
  Generally intractable to go all the way to terminal nodes

• Key idea:
  Use a heuristic evaluation function $f(s)$ that applies to intermediate states and returns a number that estimates the value of $s$
Minimax Algorithm
(Complete Search)

maxturn(s,ops):

if terminal(s) then return V(s)
else
    val ← -∞;
    foreach o ∈ ops
        val’ ← minturn(apply(s,o),ops);
        if val’ > val then
            val ← val’;
            bestop ← o;
    return val

minturn(s,ops):

if terminal(s) then return V(s)
else
    val ← +∞;
    foreach o ∈ ops
        val’ ← maxturn(apply(s,o),ops);
        if val’ < val then
            val ← val’;
            bestop ← o;
    return val

Initial call:
• If I go first: maxturn(initial-state,ops)
• If opponent goes first: minturn(initial-state,ops)
Minimax Algorithm
(Heuristic Search)

maxturn(s,ops):
  if cutoff(s) then return f(s)
  else
    val ← -∞;
    foreach o ∈ ops
      val’ ← minturn(apply(s,o),ops);
      if val’ > val then
        val ← val’;
        bestop ← o;
    return val

minturn(s,ops):
  if cutoff(s) then return f(s)
  else
    val ← +∞;
    foreach o ∈ ops
      val’ ← maxturn(apply(s,o),ops);
      if val’ < val then
        val ← val’;
        bestop ← o;
    return val

Initial call:
• If I go first: maxturn(initial-state,ops)
• If opponent goes first: minturn(initial-state,ops)
Minimax Algorithm
(Heuristic Search)

Maxturn(s,ops):

if cutoff(s) then return f(s)
else
   val ← -∞;
   foreach o ∈ ops
      val’ ← minturn(apply(s,o),ops);
      if val’ > val then
         val ← val’;
         bestop ← o;
return val

Minturn(s,ops):

if cutoff(s) then return f(s)
else
   val ← +∞;
   foreach o ∈ ops
      val’ ← maxturn(apply(s,o),ops);
      if val’ < val then
         val ← val’;
         bestop ← o;
return val

Initial call:
• If I go first: maxturn(initial-state,ops)
• If opponent goes first: minturn(initial-state,ops)

cutoff(s) could be a depth-bound or something more sophisticated
Minimax Value of a Game

- My Turn: \( V(s) = \text{Max of successors} \)
- My Opponent’s Turn: \( V(s) = \text{Min of successors} \)

Continue until terminal nodes are reached
Minimax Value of a Game

- My Turn
  - $f(s) = \text{Max of successors}$

- My Opponent’s Turn
  - $f(s) = \text{Min of successors}$

Use heuristic evaluation function

Minimax Value of a Game

- My Turn
  - $f(s) = \text{Max of successors}$

- My Opponent’s Turn
  - $f(s) = \text{Min of successors}$

Use heuristic evaluation function
Minimax Value of a Game

My Turn
\[ f(s) = \text{Max of successors} \]

My Opponent’s Turn
\[ f(s) = \text{Min of successors} \]

Use heuristic evaluation function
Minimax Algorithm
(Heuristic Search)

\[
\text{maxturn}(s, \text{ops}):
\]
\[
\begin{align*}
\text{if } & \text{cutoff}(s) \text{ then return } f(s) \\
\text{else} & \\
& \text{val } \leftarrow -\infty; \\
& \text{foreach } o \in \text{ops} \\
& \quad \text{val’ } \leftarrow \text{minturn(apply}(s, o), \text{ops}); \\
& \quad \text{if val’ } > \text{val then} \\
& \quad \quad \text{val } \leftarrow \text{val’}; \\
& \quad \quad \text{bestop } \leftarrow o; \\
& \text{return val}
\end{align*}
\]

\[
\text{minturn}(s, \text{ops}):
\]
\[
\begin{align*}
\text{if } & \text{cutoff}(s) \text{ then return } f(s) \\
\text{else} & \\
& \text{val } \leftarrow +\infty; \\
& \text{foreach } o \in \text{ops} \\
& \quad \text{val’ } \leftarrow \text{maxturn(apply}(s, o), \text{ops}); \\
& \quad \text{if val’ } < \text{val then} \\
& \quad \quad \text{val } \leftarrow \text{val’}; \\
& \quad \quad \text{bestop } \leftarrow o; \\
& \text{return val}
\end{align*}
\]

Initial call:
• If I go first: \(\text{maxturn}(\text{initial-state}, \text{ops})\)
• If opponent goes first: \(\text{minturn}(\text{initial-state}, \text{ops})\)
Alpha-Beta Pruning

New idea to improve efficiency:
Can prune branches that are guaranteed never to be used
Analogy

Given AND(p,q,r)
If p is FALSE don’t need to evaluate q or r

Given OR(p,q,r)
If q is TRUE don’t need to evaluate q or r
Alpha-Beta Pruning

Don’t explore branches that will have no effect on the outcome
Alpha-Beta Pruning

My Turn
\[ f(s) = \text{Max of successors} \]

My Opponent’s Turn
\[ f(s) = \text{Min of successors} \]
Alpha-Beta Pruning

My Turn
\[ f(s) = \text{Max of successors} \]

My Opponent’s Turn
\[ f(s) = \text{Min of successors} \]
Alpha-Beta Pruning

My Turn
\( f(s) = \text{Max of successors} \)

My Opponent’s Turn
\( f(s) = \text{Min of successors} \)
Alpha-Beta Pruning

My Turn
\( f(s) = \text{Max of successors} \)

My Opponent’s Turn
\( f(s) = \text{Min of successors} \)
Alpha-Beta Pruning

My Turn
\( f(s) = \text{Max of successors} \)

My Opponent’s Turn
\( f(s) = \text{Min of successors} \)
This is guaranteed to be a better move than this regardless of what results from the other actions.
Alpha-Beta Pruning

There is no way to set a value to either of these two states that would change the outcome.

My Turn
\( f(s) = \text{Max of successors} \)

My Opponent’s Turn
\( f(s) = \text{Min of successors} \)
Alpha-Beta Pruning: General Case

Completed the search down these branches

In the middle of searching this branch
Alpha-Beta Pruning: General Case

Completed the search down these branches

In the middle of searching this branch
Alpha-Beta Pruning: General Case

\[ m \text{ is largest (best) value of all options available to opponent in response to my move thus far} \]

\[ n \text{ is lowest (worst) value of all options available to opponent in response to my move thus far} \]
Alpha-Beta Pruning: General Case

If m is better than n then there’s no need to search further for this node.
Alpha-Beta Pruning: General Case

The same is true for my opponent symmetrically.
Alpha-Beta Pruning: Implementation

• Key idea:
  • At each node keep track of two parameters, α and β (I’ll use a and b in the algorithm)
  • α is the best value I can guarantee achieving via some earlier move
  • β is the worst value my opponent can guarantee achieving via some earlier move

• Don’t explore further nodes at this level if
  • You’re at a min node and it’s already guaranteed that this will be worse than some other move I can force (worst so far means < α)
  • You’re at a max node and it’s already guaranteed that this will be better than some other move my opponent can force on me (best so far means < β)
Alpha-Beta Pruning: General Case

- $\alpha$ is the largest (best) value of all options available to the opponent in response to my move thus far.
- $\text{val}$ is the lowest (worst) value of all options available to the opponent in response to my move thus far.

If $\alpha \geq \text{val}$, then there's no need to search further for this node.
Alpha-Beta Pruning: General Case

\[ \begin{align*}
\beta & \text{ is lowest (worst) value of all options available to opponent in response to my move thus far} \\
\text{val} & \text{ is biggest (best) value of all options available to opponent in response to my move thus far}
\end{align*} \]

If \( \beta \leq \text{val} \) then there’s no need to search further for this node.
Minimax Algorithm

maxturn(s,ops):
    if cutoff(s) then return f(s)
    else
        val ← -∞;
        foreach o ∈ ops
            val’ ← minturn(apply(s,o),ops);
            if val’ > val then
                val ← val’;
                bestop ← o;
        return val

maxturn(s,ops):
    if cutoff(s) then return f(s)
    else
        val ← +∞;
        foreach o ∈ ops
            val’ ← minturn(apply(s,o),ops);
            if val’ < val then
                val ← val’;
                bestop ← o;
        return val

Initial call:
- If I go first: maxturn(initial-state,ops,0)
- If opponent goes first: minturn(initial-state,ops,0)
Minimax Algorithm with Alpha-Beta Pruning

maxturn(s, ops, a, b): {my turn}
  if cutoff(s, depth) then return f(s)
  else
    val ← -∞;
    foreach o ∈ ops
      val’ ← minturn(apply(s, o), ops, a, b);
      if val’ > val then
        val ← val’;
        bestop ← o;
    if val ≥ b then return val;
    a ← max(a, val)
  return val

minturn(s, ops, a, b): {opponent’s turn}
  if cutoff(s, depth) then return f(s)
  else
    val ← +∞;
    foreach o ∈ ops
      val’ ← minturn(apply(s, o), ops, a, b);
      if val’ < val then
        val ← val’;
        bestop ← o;
    if val ≤ a then return val;
    b ← min(b, val)
  return val

Initial call:
• If I go first: maxturn(initial-state, ops, -∞, +∞)
• If opponent goes first: minturn(initial-state, ops, -∞, +∞)