Adversarial Search

Robert Platt
Northeastern University

Some images and slides are used from:
1. CS188 UC Berkeley
2. RN, AIMA
What is adversarial search?

Adversarial search: planning used to play a game such as chess or checkers – algorithms are similar to graph search except that we plan under the assumption that our opponent will maximize his own advantage...
Examples of adversarial search

Chess

Checkers

Tic-tac-toe

Go
Examples of adversarial search

Chess | Solved/unsolved?
Checkers | Solved/unsolved?
Tic-tac-toe | Solved/unsolved?
Go | Solved/unsolved?

Outcome of game can be predicted from any initial state assuming both players play perfectly.
Examples of adversarial search

- Chess: Unsolved
- Checkers: Solved
- Tic-tac-toe: Solved
- Go: Unsolved

Outcome of game can be predicted from any initial state assuming both players play perfectly.
Examples of adversarial search

<table>
<thead>
<tr>
<th>Game</th>
<th>Status</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess</td>
<td>Unsolve</td>
<td>$\sim 10^{40}$ states</td>
</tr>
<tr>
<td>Checkers</td>
<td>Solved</td>
<td>$\sim 10^{20}$ states</td>
</tr>
<tr>
<td>Tic-tac-toe</td>
<td>Solved</td>
<td>Less than $9!=362k$ states</td>
</tr>
<tr>
<td>Go</td>
<td>Unsolve</td>
<td>?</td>
</tr>
</tbody>
</table>

Outcome of game can be predicted from any initial state assuming both players play perfectly
Different types of games

Deterministic / stochastic

Two player / multi player?

Zero-sum / non zero-sum

Fully observable / partially observable
What is a zero-sum game?

Zero-sum:
- Sum of utilities is zero
- In the case of a two player game: \( U_A = -U_B \)
- Pure competition

Not zero-sum:
- Agents have arbitrary utilities
- Might induce cooperation or competition
A formal definition of a deterministic game

Problem:
State set: $S$ (start at $s_0$)
Players: $P = \{1 \ldots N\}$ (usually take turns)
Action set: $A$
Transition Function: $S \times A \rightarrow S$
Terminal Test: $S \rightarrow \{t,f\}$
Terminal Utilities: $S \times P \rightarrow R$

Solution:
Policy, $S \rightarrow A$

Objective:
Find an optimal policy
– a policy that maximizes utility assuming that adversary acts optimally.
A formal definition of a deterministic game

Problem:
State set: \( S \) (start at \( s_0 \))
Players: \( P = \{1\ldots N\} \) (usually take turns)
Action set: \( A \)
Transition Function: \( S \times A \rightarrow S \)
Terminal Test: \( S \rightarrow \{t,f\} \)
Terminal Utilities: \( S \times P \rightarrow R \)

Solution:
Policy, \( S \rightarrow A \)

Objective:
Find an optimal policy
– a policy that maximizes utility assuming that adversary acts optimally.

How is this similar/different to the def'n of a standard search problem?
A formal definition of a deterministic game

**Problem:**
State set: $S$ (start at $s_0$)
Players: $P=\{1...N\}$ (usually take turns)
Action set: $A$
Transition Function: $S \times A \rightarrow S$
Terminal Test: $S \rightarrow \{t,f\}$
Terminal Utilities: $S \times P \rightarrow R$

**Solution:**
Policy, $S \rightarrow A$

**Objective:**
Find an optimal policy
– a policy that maximizes utility assuming that adversary acts optimally.

How do we solve this problem?
Adversarial search
This is a game tree for tic-tac-toe

Images: AIMA, Berkeley CS188 course notes (downloaded Summer 2015)
This is a game tree for tic-tac-toe

Images: AIMA, Berkeley CS188 course notes (downloaded Summer 2015)
This is a game tree for tic-tac-toe

Images: AIMA, Berkeley CS188 course notes (downloaded Summer 2015)
This is a game tree for tic-tac-toe

Images: AIMA, Berkeley CS188 course notes (downloaded Summer 2015)
This is a game tree for tic-tac-toe

MAX (x)

MIN (o)

MAX (x)

MIN (o)

TERMINAL

Utility

Images: AIMA, Berkeley CS188 course notes (downloaded Summer 2015)
This is a game tree for tic-tac-toe

Images: AIMA, Berkeley CS188 course notes (downloaded Summer 2015)
What is Minimax?

Consider a simple game:
1. you make a move
2. your opponent makes a move
3. game ends
What is Minimax?

Consider a simple game:
1. you make a move
2. your opponent makes a move
3. game ends

What does the minimax tree look like in this case?
What is Minimax?

Consider a simple game:
1. you make a move
2. your opponent makes a move
3. game ends

What does the minimax tree look like in this case?
What is Minimax?

Max (you)

Min (them)

Max (you)

These are terminal utilities – assume we know what these values are
What is Minimax?

Max (you)

Min (them)

Max (you)

$V(s) = \min_{s' \in \text{successors}(s)} V(s')$
What is Minimax?

\[ V(s) = \max_{s' \in \text{successors}(s)} V(s') \]
What is Minimax?

Max (you)

Min (them)

Max (you)

This is called “backing up” the values

\[ V(s) = \max_{s' \in \text{successors}(s)} V(s') \]
What is Minimax?

Okay – so we know how to back up values ...

... but, how do we construct the tree?

This tree is already built...
What is Minimax?

Notice that we only get utilities at the *bottom* of the tree ... – therefore, DFS makes sense.
What is Minimax?

Notice that we only get utilities at the *bottom* of the tree ... – therefore, DFS makes sense.
What is Minimax?

Notice that we only get utilities at the \textit{bottom} of the tree ... – therefore, DFS makes sense.
What is Minimax?

Notice that we only get utilities at the bottom of the tree … – therefore, DFS makes sense.
What is Minimax?

Notice that we only get utilities at the *bottom* of the tree … – therefore, DFS makes sense.
What is Minimax?

Notice that we only get utilities at the bottom of the tree ... – therefore, DFS makes sense.
What is Minimax?

Notice that we only get utilities at the *bottom* of the tree ... – therefore, DFS makes sense.
What is Minimax?

Notice that we only get utilities at the *bottom* of the tree ... – therefore, DFS makes sense.
What is Minimax?

Notice that we only get utilities at the *bottom* of the tree … – therefore, DFS makes sense.
What is Minimax?

Notice that we only get utilities at the bottom of the tree … therefore, DFS makes sense.
– since most games have forward progress, the distinction between tree search and graph search is less important
**What is Minimax?**

<table>
<thead>
<tr>
<th>function MINIMAX-DECISION(state) returns an action</th>
</tr>
</thead>
<tbody>
<tr>
<td>return ( \arg \max_a \in \text{ACTIONS}(s) \ \text{MIN-VALUE(RESULT}(\text{state}, a)) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>function MAX-VALUE(state) returns a utility value</th>
</tr>
</thead>
<tbody>
<tr>
<td>if TERMINAL-TEST(state) then return UTILITY(state)</td>
</tr>
<tr>
<td>( v \leftarrow -\infty )</td>
</tr>
<tr>
<td>for each a in ACTIONS(state) do</td>
</tr>
<tr>
<td>( v \leftarrow \text{MAX}(v, \text{MIN-VALUE(RESULT}(s, a))) )</td>
</tr>
<tr>
<td>return v</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>function MIN-VALUE(state) returns a utility value</th>
</tr>
</thead>
<tbody>
<tr>
<td>if TERMINAL-TEST(state) then return UTILITY(state)</td>
</tr>
<tr>
<td>( v \leftarrow \infty )</td>
</tr>
<tr>
<td>for each a in ACTIONS(state) do</td>
</tr>
<tr>
<td>( v \leftarrow \text{MIN}(v, \text{MAX-VALUE(RESULT}(s, a))) )</td>
</tr>
<tr>
<td>return v</td>
</tr>
</tbody>
</table>

**Figure 5.3** An algorithm for calculating minimax decisions. It returns the action corresponding to the best possible move, that is, the move that leads to the outcome with the best utility, under the assumption that the opponent plays to minimize utility. The functions MAX-VALUE and MIN-VALUE go through the whole game tree, all the way to the leaves, to determine the backed-up value of a state. The notation \( \arg \max_{a \in S} f(a) \) computes the element \( a \) of set \( S \) that has the maximum value of \( f(a) \).
Minimax properties

Is it always correct to assume your opponent plays optimally?
Minimax vs “expectimax”
Minimax vs “expectimax”
Minimax properties

Is minimax optimal? Is it complete?
Minimax properties

Is minimax optimal? Is it complete?

Time complexity = ?

Space complexity = ?
Minimax properties

Is minimax optimal? Is it complete?

Time complexity = $O(b^d)$

Space complexity = $O(bd)$
Minimax properties

Is minimax optimal? Is it complete?

Time complexity = $O(b^d)$

Space complexity = $O(bd)$

Is it practical? In chess, $b=35$, $d=100$
Minimax properties

Is minimax optimal? Is it complete?

Time complexity = $O(b^d)$

Space complexity = $O(bd)$

Is it practical? In chess, $b=35, d=100$.

$O(35^{100})$ is a big number...
Minimax properties

Is minimax optimal? Is it complete?

Time complexity $= O(b^d)$

Space complexity $= O(bd)$

Is it practical? In chess, $b=35$, $d=100$

$O(35^{100})$ is a big number...

So what can we do?
Evaluation functions

Key idea: cut off search at a certain depth and give the corresponding nodes an estimated value.

Image: Berkeley CS188 course notes (downloaded Summer 2015)
Key idea: cut off search at a certain depth and give the corresponding nodes an estimated value.

The evaluation function makes this estimate.

Cut it off here
Evaluation functions

How does the evaluation function make the estimate?
– depends upon domain

For example, in chess, the value of a state might equal the sum of piece values.
– a pawn counts for 1
– a rook counts for 5
– a knight counts for 3
...

A weighted linear evaluation function

\[ \text{eval}(s) = w_1 f_1(s) + \cdots + w_n f_n(s) \]

\[ f_1(s) \equiv \text{number of pawns on the board} \]
\[ f_2(s) \equiv \text{number of knights on the board} \]
\[ \vdots \]

\[ w_1 = 1 \quad \text{A pawn counts for 1} \]
\[ w_2 = 3 \quad \text{A knight counts for 3} \]
\[ \vdots \]
At what depth do you run the evaluation function?

Option 1: cut off search at a fixed depth

Option 2: cut off search at quiescent states deeper than a certain threshold

Option 3: ?

The deeper your threshold, the less the quality of the evaluation function matters...
At what depth do you run the evaluation function?

Search depth=2
At what depth do you run the evaluation function?

Search depth=10

Slide: Berkeley CS188 course notes (downloaded Summer 2015)
Alpha/Beta pruning

3

3 12 8
Alpha/Beta pruning

3

8

12

3
Alpha/Beta pruning
Alpha/Beta pruning
Alpha/Beta pruning

We don't need to expand this node!
We don't need to expand this node!

Why?
Alpha/Beta pruning

We don't need to expand this node!

Why?

Max

Min

3

12

8

2

4

We don't need to expand this node!

Why?
Alpha/Beta pruning

Max

Min

3

12

8

2

14

5

2
Alpha/Beta pruning

So, we don't need to expand these nodes in order to back up correct values!
So, we don't need to expand these nodes in order to back up correct values! That's alpha-beta pruning.
Alpha/Beta pruning: algorithm idea

- General configuration (MIN version)
  - We’re computing the MIN-VALUE at some node $n$
  - We’re looping over $n$’s children
  - $n$’s estimate of the childrens’ min is dropping
  - Who cares about $n$’s value? MAX
  - Let $a$ be the best value that MAX can get at any choice point along the current path from the root
  - If $n$ becomes worse than $a$, MAX will avoid it, so we can stop considering $n$’s other children (it’s already bad enough that it won’t be played)

- MAX version is symmetric
Alpha/Beta pruning: algorithm

α: best value so far for MAX along path to root
β: best value so far for MIN along path to root

def min-value(state, α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
    if v ≤ α return v
    β = min(β, v)
    return v

def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
    if v ≥ β return v
    α = max(α, v)
    return v

Slide: adapted from Berkeley CS188 course notes (downloaded Summer 2015)
Alpha/Beta pruning

(-\infty, +\infty)
Alpha/Beta pruning
Alpha/Beta pruning

Best value for far for MIN along path to root

(-inf,3) → 3 → 3 → (-inf,+inf)
Alpha/Beta pruning

Best value for far for MIN along path to root

(-inf,3)

3

3

12

(-inf,+inf)
Alpha/Beta pruning

Best value for far for MIN along path to root

(-inf, 3)

3

3  12  8

(-inf, +inf)
Alpha/Beta pruning

Best value for far for MAX along path to root

(3, +\infty)

(-\infty, 3)

Best value for far for MAX along path to root
Alpha/Beta pruning
Alpha/Beta pruning

(-inf, 3) → 3 → 3 → 3
(-inf, 3) → 3 → 12 → 8
(3, +inf) → 3 → 2 → 2
(3, +inf)
Alpha/Beta pruning

Prune because value (2) is out of alpha-beta range
Alpha/Beta pruning

(-\infty, 3) 

(3, +\infty)

(3, +\infty)

(3, +\infty)
Alpha/Beta pruning
Alpha/Beta pruning

(-\infty, 3)

(3, +\infty)

(3, 5)

(3, +\infty)
Alpha/Beta properties

Is it complete?
Alpha/Beta properties

Is it complete?

How much does alpha/beta help relative to minimax?

Minimax time complexity = $O(b^m)$

Alpha/beta time complexity $\geq O(b^{m/2})$

– the improvement w/ alpha/beta depends upon move ordering...
Is it complete?

How much does alpha/beta help relative to minimax?

Minimax time complexity = $O(b^m)$

Alpha/beta time complexity $\geq O\left(b^{\frac{m}{2}}\right)$

- the improvement w/ alpha/beta depends upon move ordering...

The order in which we expand a node.
Alpha/Beta properties

Is it complete?

How much does alpha/beta help relative to minimax?

Minimax time complexity = $O(b^m)$

Alpha/beta time complexity $\geq O(\left\lfloor b^{\frac{m}{2}} \right\rfloor)$

- the improvement w/ alpha/beta depends upon move ordering...

The order in which we expand a node.

How to choose move ordering? Use IDS.
- on each iteration of IDS, use prior run to inform ordering of next node expansions.