Game-Playing & Adversarial Search
Alpha-Beta Pruning, etc.

This lecture topic:
Game-Playing & Adversarial Search
(Alpha-Beta Pruning, etc.)
Read Chapter 5.3-5.5

Next lecture topic:
Constraint Satisfaction Problems (two lectures)
Read Chapter 6.1-6.4, except 6.3.3

(Please read lecture topic material before and after each lecture on that topic)
You Will Be Expected to Know

- Alpha-beta pruning (5.3)
- Expectimax (5.5)
Review of Previous Lecture

- Basic definitions (section 5.1)
- Minimax optimal game search (5.2)
- Evaluation functions (5.4.1)
- Cutting off search (5.4.2)
Alpha-Beta Pruning
Exploiting the Fact of an Adversary

• If a position is provably bad:
  – It is NO USE searching to find out exactly how bad

• If the adversary can force a bad position:
  – It is NO USE searching to find the good positions
    the adversary won’t let you achieve anyway

• Bad = not better than we can get elsewhere.
Tic-Tac-Toe Example with Alpha-Beta Pruning

Do these nodes matter?
If they = +1 million?
If they = −1 million?

Figure 4.17 Two-ply minimax applied to the opening move of tic-tac-toe.
Extended Alpha-Beta Example

“Possible Interval” View: \((\text{min}, \text{max})\)

Initially, interval is unknown \((-\infty, +\infty)\):

Range of possible values \(\rightarrow (\infty, +\infty)\)
Extended Alpha-Beta Example

“Possible Interval” View

Do DF-search until first leaf:

- Define the range of possible values as $(-\infty, +\infty)$.
- Child inherits the current interval.

Diagram:

- MAX
- MIN
- Range of possible values
- $(-\infty, +\infty)$
Extended Alpha-Beta Example

“Possible Interval” View

See first leaf, update possible interval:

Range of possible values

$(\infty, 3]$
Extended Alpha-Beta Example

“Possible Interval” View

See remaining leaves, and outcome is known:

\[ [3,3] \rightarrow (\infty, +\infty) \]
Extended Alpha-Beta Example

“Possible Interval” View

Pass outcome to caller, and update caller:

Range of possible values $\rightarrow [3, +\infty)$
Extended Alpha-Beta Example

“Possible Interval” View

Do DF-search until first leaf:

Child inherits current interval
Extended Alpha-Beta Example

“Possible Interval” View

See first leaf, update possible interval:

No remaining possibilities

[3,3]

[3,2]

[3, +∞)

This node is worse for MAX
Extended Alpha-Beta Example
“Possible Interval” View

Prune!! Play will never reach the other nodes:

Do these nodes matter?
If they = +1 million?
If they = −1 million?

We would prune if we had seen any value \( \leq 3 \)

Prune!!
Pass outcome to caller, and update caller:

MAX level, 3 ≥ 2, no change → [3, +∞)
General alpha-beta pruning

• Consider node $n$ in the tree --

• If player has a better choice at:
  – Parent node of $n$
  – Or any choice point further up

• Then $n$ is never reached in play.

• So:
  – When that much is known about $n$, it can be pruned.
Alpha-beta Algorithm

- Depth first search
  - only considers nodes along a single path from root at any time

\[ \alpha = \text{highest-value choice found at any choice point of path for MAX} \]
  (initially, \( \alpha = -\infty \))

\[ \beta = \text{lowest-value choice found at any choice point of path for MIN} \]
  (initially, \( \beta = +\infty \))

- Pass current values of \( \alpha \) and \( \beta \) down to child nodes during search

- Update values of \( \alpha \) and \( \beta \) during search:
  - MAX updates \( \alpha \) at MAX nodes
  - MIN updates \( \beta \) at MIN nodes

- Prune remaining branches at a node when \( \alpha \geq \beta \)
When to Prune?

- Prune whenever $\alpha \geq \beta$.
  - Prune below a Max node whose alpha value becomes $\geq$ to the beta value of its ancestors.
    - Max nodes update alpha based on children’s returned values.
  - Prune below a Min node whose beta value becomes $\leq$ the alpha value of its ancestors.
    - Min nodes update beta based on children’s returned values.
Pseudocode for Alpha-Beta Algorithm

function ALPHA-BETA-SEARCH(state) returns an action
inputs: state, current state in game
bestValue ← -\infty
bestAction ← NIL
for a in Actions(state) do
    v ← MAX-VALUE( Result(state, a), -\infty, +\infty )
    when ( v > bestValue ) do
        bestValue ← v
        bestAction ← a
return ( bestAction, bestValue )
Pseudocode for Alpha-Beta Algorithm

function ALPHA-BETA-SEARCH(state) returns an action

inputs: state, current state in game

bestValue ← - ∞
bestAction ← NIL

for a in Actions(state) do
    v ← MAX-VALUE(Result(state, a), - ∞, + ∞)
    when ( v > bestValue ) do
        bestValue ← v
        bestAction ← a

return ( bestAction, bestValue )

function MAX-VALUE(state, α, β) returns a utility value

if TERMINAL-TEST(state) then return UTILITY(state)

v ← - ∞

for a in ACTIONS(state) do
    v ← MAX( v, MIN-VALUE(Result(s,a), α, β) )
    if v ≥ β then return v
    α ← MAX(α, v)

return v

(MIN-VALUE is defined analogously)
Extended Alpha-Beta Example

“Alpha-Beta” View: \((\alpha, \beta)\)

Initially, possibilities are unknown \((\alpha=-\infty, \beta=+\infty)\):

\[ \alpha, \beta, \text{ initial values} \rightarrow \alpha=-\infty \]

\[ \beta = +\infty \]
Extended Alpha-Beta Example

“Alpha-Beta” View

Do DF-search until first leaf:

Child inherits current $\alpha$ and $\beta$
Extended Alpha-Beta Example

“Alpha-Beta” View

See first leaf, MIN level, MIN updates $\beta$:

$\alpha = -\infty$

$\beta = +\infty$

$MIN$ updates $\beta$

at $MIN$ nodes

$\alpha = -\infty$

$\beta = 3$

$\alpha < \beta$ so

no pruning
Extended Alpha-Beta Example

“Alpha-Beta” View

See remaining leaves, and outcome is known:

\[ \alpha = -\infty \]
\[ \beta = +\infty \]
Extended Alpha-Beta Example
“Alpha-Beta” View

Pass outcome to caller, MAX updates $\alpha$:

$\alpha = 3$

$\beta = +\infty$

MAX updates $\alpha$ at MAX nodes

Diagram:
- MAX
- MIN
- Nodes: 3, 12, 8
Extended Alpha-Beta Example

“Alpha-Beta” View

Do DF-search until first leaf:

Child inherits current \((\alpha, \beta)\)
Extended Alpha-Beta Example

“Alpha-Beta” View

See first leaf, MIN level, MIN updates $\beta$:

\[
\alpha \geq \beta \quad !!
\]

What does this mean?

This node is worse for MAX

$\alpha = 3$

$\beta = +\infty$

$\alpha = 3$

$\beta = 2$
**Extended Alpha-Beta Example**

**“Alpha-Beta” View**

Prune!! Play will never reach the other nodes:

- If they = +1 million?
- If they = −1 million?

\[ \alpha = 3 \]
\[ \beta = +\infty \]

\[ \alpha \geq \beta \]

Do these nodes matter?
- If they = +1 million?
- If they = −1 million?

Prune!!
Extended Alpha-Beta Example

“Alpha-Beta” View

Pass outcome to caller, and update caller:

MAX level,
3 ≥ 2,
no change

α = 3
β = +∞
Effectiveness of Alpha-Beta Search

- **Worst-Case**
  - branches are ordered so that no pruning takes place. In this case alpha-beta gives no improvement over exhaustive search

- **Best-Case**
  - each player’s best move is the left-most child (i.e., evaluated first)
  - in practice, performance is closer to best rather than worst-case
  - E.g., sort moves by the remembered move values found last time.
  - E.g., expand captures first, then threats, then forward moves, etc.
  - E.g., run Iterative Deepening search, sort by value last iteration.

- **In practice often get** $O(b^{m/2})$ **rather than** $O(b^m)$
  - this is the same as having a branching factor of $\sqrt{b}$,
    - $(\sqrt{b})^m = b^{m/2}$, i.e., we effectively go from $b$ to square root of $b$
    - e.g., in chess go from $b \approx 35$ to $b \approx 6$
    - this permits much deeper search in the same amount of time
Iterative (Progressive) Deepening

- In real games, there is usually a time limit $T$ on making a move

- How do we take this into account?
- In practice, Iterative Deepening Search (IDS) is used
  - IDS runs depth-first search with an increasing depth-limit
  - when the clock runs out we use the solution found at the previous depth limit

- Added benefit with Alpha-Beta Pruning:
  - Remember node values found at the previous depth limit
  - Sort current nodes so each player's best move is left-most child
  - Likely to yield good Alpha-Beta Pruning $\Rightarrow$ better, faster search
  - Only a heuristic: node values will change with the deeper search
  - Usually works well in practice
Final Comments about Alpha-Beta Pruning

- Pruning does not affect final results

- Entire subtrees can be pruned.

- Good move *ordering* improves pruning
  - Order nodes so player’s best moves are checked first

- Repeated states are still possible.
  - Store them in memory = transposition table
Example

-which nodes can be pruned?
Example

-which nodes can be pruned?

Answer: **NONE!** Because the most favorable nodes for both are explored last (i.e., in the diagram, are on the right-hand side).
Second Example
(the exact mirror image of the first example)

-which nodes can be pruned?
Second Example
(the exact mirror image of the first example)

-which nodes can be pruned?

Answer: **LOTS!** Because the most favorable nodes for both are explored first (i.e., in the diagram, are on the left-hand side).
Example With NO Pruning
Iterative Deepening Reordering

Assume static node score is the average of the leaf values below that node.

Answer: **NONE!** Because the most favorable nodes for both are explored last (i.e., in the diagram, are on the right-hand side).
Example With NO Pruning
Iterative Deepening Reordering

Assume static node score is the average of the leaf values below that node.
Example With NO Pruning
Iterative Deepening Reordering

Assume static node score is the average of the leaf values below that node

L=1
Max
6.5
Min
2.5
Max
6.5

For L=2 switch the order of these nodes!

3 4 1 2 7 8 5 6

Max

Assume static node score is the average of the leaf values below that node.
Example With NO Pruning
Iterative Deepening Reordering

Assume static node score is the average of the leaf values below that node.

Alpha-beta pruning would prune this node at L=2.
Example With NO Pruning
Iterative Deepening Reordering

Assume static node score is the average of the leaf values below that node.

For L=3 switch the order of these nodes!

L=2

7.5 5.5

3.5 5.5
Example With NO Pruning
Iterative Deepening Reordering

Assume static node score is the average of the leaf values below that node.

For L=3 switch the order of these nodes!
Example With NO Pruning
Iterative Deepening Reordering

Answer: **LOTS!** Because the most favorable nodes for both are explored earlier due to reordering based on iterative deepening.
Long Detailed Alpha-Beta Example

Branch nodes are labeled A-K for easy discussion

\(\alpha, \beta, \text{ initial values} \rightarrow \alpha = -\infty\)

\(\beta = +\infty\)
Long Detailed Alpha-Beta Example

Note that search cut-off occurs at different depths

current $\alpha$, $\beta$, passed to kids

$\alpha = -\infty$
$\beta = +\infty$

$\alpha = -\infty$
$\beta = +\infty$

$\alpha = -\infty$
$\beta = +\infty$

$\alpha = -\infty$
$\beta = +\infty$

$\alpha = -\infty$
$\beta = +\infty$

$\alpha = -\infty$
$\beta = +\infty$
Long Detailed Alpha-Beta Example

see first leaf, MAX updates $\alpha$

$\alpha = -\infty$
$\beta = +\infty$

$\alpha = -\infty$
$\beta = +\infty$
kid = A

$\alpha = 4$
$\beta = +\infty$
kid = E

We also are running MiniMax search and recording node values within the triangles, without explicit comment.
Long Detailed Alpha-Beta Example

see next leaf, MAX updates $\alpha$

$\alpha = -\infty$
$\beta = +\infty$

$\alpha = 5$
$\beta = +\infty$
kid = E

4 5 6 6 1 9 5 4 1 3 2 6 1 3 6 8 6 3
Long Detailed Alpha-Beta Example

see next leaf, 
MAX updates $\alpha$

$\alpha = -\infty$
$\beta = +\infty$
kid=A

$\alpha = 6$
$\beta = +\infty$
kid=E
Long Detailed Alpha-Beta Example

return node value, MIN updates $\beta$

$\alpha = -\infty$

$\beta = +\infty$

$\alpha = -\infty$

$\beta = 6$

kid=A

6
Long Detailed Alpha-Beta Example

current $\alpha$, $\beta$, passed to kid F

$\alpha = -\infty$
$\beta = +\infty$

kid = A

$\alpha = -\infty$
$\beta = 6$

kid = F

$\alpha = -\infty$
$\beta = 6$
Long Detailed Alpha-Beta Example

see first leaf,
MAX updates $\alpha$

$\alpha=-\infty$
$\beta=+\infty$

$\alpha=6$
$\beta=6$
kid=A

$\alpha=6$
kid=F
Long Detailed Alpha-Beta Example

\[ \alpha \geq \beta \text{ !!} \]

Prune!!

\[ \alpha = -\infty \]
\[ \beta = +\infty \]

\[ \alpha = 6 \]
\[ \beta = 6 \]

kid = A

\[ \alpha = 6 \]
\[ \beta = 6 \]

kid = F

\[ \alpha \geq \beta \text{ !!} \]

Prune!!
Long Detailed Alpha-Beta Example

α = −∞
β = +∞

return node value, MIN updates β, no change to β

If we had continued searching at node F, we would see the 9 from its third leaf. Our returned value would be 9 instead of 6. But at A, MIN would choose E(=6) instead of F(=9). Internal values may change; root values do not.
Long Detailed Alpha-Beta Example

See next leaf, MIN updates $\beta$, no change to $\beta$

$\alpha = -\infty$

$\beta = +\infty$

$\alpha = -\infty$

$\beta = 6$

Kid = A
Long Detailed Alpha-Beta Example

return node value, $\alpha = 6$

$\beta = +\infty$

MAX updates $\alpha$

$\alpha = 6$
Long Detailed Alpha-Beta Example

current $\alpha$, $\beta$, passed to kids

$\alpha=6$  
$\beta=+\infty$  
kid=B

$\alpha=6$  
$\beta=+\infty$  
kid=G

$\alpha=6$  
$\beta=+\infty$
Long Detailed Alpha-Beta Example

see first leaf, MAX updates $\alpha$, no change to $\alpha$

$\alpha=6$
$\beta=+\infty$
kid=B

$\alpha=6$
$\beta=+\infty$
kid=G

$\alpha=6$
$\beta=+\infty$
kid=B

$\alpha=6$
$\beta=+\infty$
kid=G
Long Detailed Alpha-Beta Example

\[ \alpha = 6 \]
\[ \beta = +\infty \]

Kid = B
\[ \alpha = 6 \]
\[ \beta = +\infty \]

Kid = G
\[ \alpha = 6 \]
\[ \beta = +\infty \]

See next leaf, MAX updates \( \alpha \), no change to \( \alpha \)
Long Detailed Alpha-Beta Example

return node value, MIN updates β

α=6
β = +∞

α=6
β=5
kid=B

MAX

MIN
Note that we never find out, what is the node value of H? But we have proven it doesn’t matter, so we don’t care.
Long Detailed Alpha-Beta Example

return node value, $\alpha=6$
MAX updates $\alpha$, no change to $\alpha$

$\beta = +\infty$
Long Detailed Alpha-Beta Example

current $\alpha$, $\beta$, passed to kid=C

$\alpha=6$
$\beta=+\infty$
kid=C

MAX

MIN

current $\alpha$, $\beta$, passed to kid=C

$\alpha=6$
$\beta=+\infty$
kid=C

MAX

MIN

MAX
Long Detailed Alpha-Beta Example

\[ \alpha = 6 \]
\[ \beta = +\infty \]

see first leaf, MIN updates \( \beta \)

\[ \alpha = 6 \]
\[ \beta = 9 \]
\( \text{kid} = C \)
Long Detailed Alpha-Beta Example

current $\alpha$, $\beta$, passed to kid I

$\alpha = 6$
$\beta = +\infty$

$\alpha = 6$
$\beta = 9$
kid = C

$\alpha = 6$
$\beta = 9$
kid = I

current $\alpha$, $\beta$, passed to kid I

$\alpha = 6$
$\beta = 9$
kid = I
Long Detailed Alpha-Beta Example

see first leaf, 
MAX updates $\alpha$, 
no change to $\alpha$

$\alpha=6$  
$\beta=+\infty$

$\alpha=6$  
$\beta=9$  
kid=C

$\alpha=6$  
$\beta=9$  
kid=I

see first leaf, 
MAX updates $\alpha$, 
no change to $\alpha$
Long Detailed Alpha-Beta Example

see next leaf, MAX updates $\alpha$, no change to $\alpha$

$\alpha = 6$
$\beta = +\infty$

MAX

$\alpha = 6$
$\beta = 9$
kid = C

$\alpha = 6$
$\beta = 9$
kid = I

see next leaf, MAX updates $\alpha$, no change to $\alpha$
return node value, MIN updates $\beta$

$\alpha=6$
$\beta = +\infty$

A 6

B 5

C 6

D

E 6

F 6

G 5

H

I 6

J

K

MAX

MIN

MAX

MAX

return node value, MIN updates $\beta$
Long Detailed Alpha-Beta Example

\[ \alpha = 6 \]
\[ \beta = 6 \]

\[ \text{kid} = C \]

\[ \alpha \geq \beta \text{ !!} \]

\[ \text{Prune!!} \]

\[ \alpha = 6 \]
\[ \beta = +\infty \]

MAX

MIN

\[ X \times X \times X \times X \]
Long Detailed Alpha-Beta Example

return node value, \( \alpha = 6 \)

MAX updates \( \alpha \),
no change to \( \alpha \)

\( \alpha = 6 \)
Long Detailed Alpha-Beta Example

\[ \alpha = 6 \]
\[ \beta = +\infty \]

current \( \alpha, \beta, \) passed to kid=D

\[ \alpha = 6 \]
\[ \beta = +\infty \]

kid=D
Long Detailed Alpha-Beta Example

See first leaf, MIN updates β

α = 6
β = +∞
Long Detailed Alpha-Beta Example

\[ \alpha \geq \beta \text{ !! Prune!!} \]

\[ \alpha = 6 \]
\[ \beta = +\infty \]

\[ \alpha = 6 \]
\[ \beta = 6 \]

kid = D
return node value, $\alpha=6$

MAX updates $\alpha$, $\beta=+\infty$

no change to $\alpha$
Long Detailed Alpha-Beta Example

**MAX moves to A, and expects to get 6**

Although we may have changed some internal branch node return values, the final root action and expected outcome are identical to if we had not done alpha-beta pruning. Internal values may change; root values do not.
Nondeterministic games: backgammon
Nondeterministic games in general

In nondeterministic games, chance introduced by dice, card-shuffling

Simplified example with coin-flipping:

```
MAX

CHANCE

MIN

2 4 7 4 6 0 5 -2
```
Algorithm for nondeterministic games

EXPECTIMINIMAX gives perfect play

Just like MINIMAX, except we must also handle chance nodes:

\[\ldots\]
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)
\[\ldots\]
Partially Observable Games
(R&N, Chapter 5.6 --- “The fog of war.”)

• Background: R&N, Chapter 4.3-4
  – Searching with Nondeterministic Actions/Partial Observations

• Search through Belief States (see Fig. 4.14)
  – Agent’s current belief about which states it might be in,
    given the sequence of actions & percepts to that point

• Actions(b) = ?? Union? Intersection?
  – Tricky: an action legal in one state may be illegal in another
  – Is an illegal action a NO-OP? or the end of the world?

• Transition Model:
  – Result(b,a) = { s’ : s’ = Result(s, a) and s is a state in b }

• Goaltest(b) = every state in b is a goal state
Belief States for Unobservable Vacuum World
Partially Observable Games  
(R&N, Chapter 5.6)

• Player’s current node is a belief state
• Player’s move (action) generates child belief state
• Opponent’s move is replaced by Percepts(s)
  – Each possible percept leads to the belief state that is consistent with that percept
• Strategy=a move for every possible percept sequence
• Minimax returns the worst state in the belief state

• Many more complications and possibilities!!
  – Opponent may select a move that is not optimal, but instead minimizes the information transmitted, or confuses the opponent
  – May not be reasonable to consider ALL moves; open P-QR3??

• See R&N, Chapter 5.6, for more info
The State of Play

• **Checkers:**

• **Chess:**

• **Othello:**
  - human champions refuse to compete against computers: they are too good.

• **Go:**
  - AlphaGo recently beat the human European Go Champion.
  - $b > 300$ (!); full game tree has $> 10^{760}$ leaf nodes (!!!)

• See (e.g.) [http://www.cs.ualberta.ca/~games/](http://www.cs.ualberta.ca/~games/) for more information
PHILADELPHIA (Reuters) - IBM chess computer Deep Blue made chess history Saturday when it defeated world champion Garry Kasparov, the first time a computer program has beaten a grandmaster under strict tournament conditions.

IBM Deep Blue - Kasparov, G [B22]
Philadelphia (1), 1996

13.Nb5 Qe7 14.Ne5 Bxe2 15.Qxe2 0-0 16.Rac1 Rac8 17.Bg5 Bb6
23.d5 Rxd5 24.Rxd5 exd5 25.b3 Kh8 26.Qxb6 Rg8 27.Qc5 d4
Re1+ 34.Kh2 Nxf2 35.Nxf2+ Kg7 36.Ng5+ Kh6 37.Rxh7+ 1-0
Deep Blue

• 1957: Herbert Simon
  – “within 10 years a computer will beat the world chess champion”

• 1997: Deep Blue beats Kasparov

• Parallel machine with 30 processors for “software” and 480 VLSI processors for “hardware search”

• Searched 126 million nodes per second on average
  – Generated up to 30 billion positions per move
  – Reached depth 14 routinely

• Uses iterative-deepening alpha-beta search with transpositioning
  – Can explore beyond depth-limit for interesting moves
Moore’s Law in Action?

Figure 5.12  Ratings of human and machine chess champions.
Summary

• Game playing is best modeled as a search problem
• Game trees represent alternate computer/opponent moves
• Evaluation functions estimate the quality of a given board configuration for the Max player.
• Minimax is a procedure which chooses moves by assuming that the opponent will always choose the move which is best for them
• Alpha-Beta is a procedure which can prune large parts of the search tree and allow search to go deeper
• For many well-known games, computer algorithms based on heuristic search match or out-perform human world experts.