Adversarial Search

Professor Marie Roch
Chapter 5, Russell & Norvig

Two player games

Primary focuses
• Zero-sum games: What’s good for you is bad for me
• turn-based (alternating)
• perfect information
• pruning: Removing portions of search tree
Game search problem

- initial state
- player – turn indicator
- actions – Set of legal actions
- result(state, action) – Result of player taking action
- terminal-test(state) – game over predicate
- utility(state, player) – utility of state for specified player

Game tree

- Game tree – exploration of the game search space
- Can be large
  - Chess – average branch factor 35
    (about $10^{40}$ distinct nodes, and $10^{154}$ states in a 50 move game)
  - Checkers, about $10^{40}$
- Turn consists of two plys
- Frequently use the same utility function, maximizing for one player and minimizing for the other
Tic-tac-toe

Fig. 5.1, R&N

min(o): Player O is likely to pick the move that minimizes the utility for player X, so we consider the parent’s score to be the descendent with the lowest utility.

Minimax algorithm

\[
\text{minimax(state)} = \arg\max_{a \in \text{actions(state)}} \text{minval(result(state, a))}
\]

\[
\text{maxval(state)}
\]

- if terminal(state), \(v = \text{utility(state)}\)
- else
  \[
  v = -\infty
  \]
  for a in actions(state)
  \[
  v = \max(v, \text{minval(result(state, a))})
  \]
  return v

\[
\text{minval(state)}
\]

- if terminal(state), \(v = \text{utility(state)}\)
- else
  \[
  v = \infty
  \]
  for a in actions(state)
  \[
  v = \min(v, \text{maxval(result(state, a)))}
  \]
  return v
minimax(A)
    return arg\( \max_{a \in \text{actions} = \{a_1, a_2, a_3\}(A)} \) \( \minval(\text{result}(A,a)) \)

minval(B)
    if terminal(B), v = utility(B)
    else
        v = \( \infty \)
        for a in \{b_1, b_2, b_3\}
            v = min(v, maxval(\text{result}(B, a)))
    return v
maxval(state)
  if terminal(state), v = utility(state)
  else
    v = -\infty
    for a in actions(state)
      v = max(v, minval(result(state, a)))
  return v

minval(B)
  if terminal(B), v = utility(B)
  else
    v = \infty
    for a in \{b_1, b_2, b_3\}
      v = min(v, maxval(result(B, a)))
  return v
minval(C)
   if terminal(C), v = utility(C)
   else
     v = ∞
     for a in \{c_1, c_2, c_3\}
       v = min(v, maxval(result(C, a)))
     return v
Trivial game tree

minimax(A)
return arg(max_{a \in \{a_1,a_2,a_3\}}(A) minval(result(A,a)))

best move a_1!

Multiplayer games

- Replace utility value with utility vector: \([u_1,u_2,\ldots,u_N]\)
- When evaluating nodes, utility is interpreted as a function of the utility vector and the agent that produced the node.
  - Every player for themselves: \(u(i) = u_i\)
  - Alliances \(u(i) = f([u_1,u_2,\ldots,u_N])\)
Multiplayer games \((u_A, u_B, u_C)\)

![Game tree diagram]

**Game tree size**

- Non-trivial trees are large

- Strategies to reduce search size:
  - alpha-beta pruning – Remove subtrees that can’t influence the decision
  - move-ordering –order affects pruning performance
  - cutoff – Don’t evaluate all the way to terminal nodes
Recall our trivial game tree

Once we know 2, we don’t need to look at siblings...
alpha-beta pruning

- Prune partial computations
  - max(\(a_{\text{min}}\), \(\min(\ldots < a_{\text{min}} \ldots)\))
    - example from earlier slide
    - max(min(3, 12, 8), min(2, 4, 6), min(14, 5, 2))
  - min(\(a_{\text{max}}\), \(\max(\ldots > a_{\text{max}} \ldots)\))

- Can be applied at any depth in tree
- Does not change minimax decision

Bounding a node

- Possible values for a node
  - \(\alpha\) - lower bound
  - \(\beta\) - upper bound
- \(\alpha\)-\(\beta\) pruning looks for situations where \(\alpha > \beta\)
  - as these are not viable solutions
Bounding a node

- Try to increase lower bound of max nodes

- Try to decrease upper bound of min nodes

Bounding a node

- Max nodes – loop children
  - If min node child value ≥ β
    - return child value
  - else see if we can increase lower bound α

- Min nodes – loop children
  - if max node child value ≤ α
    - return child value
  - else see if we can decrease upper bound β

Formal algorithm later, this is just to build your intuition
Bounding example:
\[ \max(\min(3,12,8), \min(2,4,6), \min(14,5,2)) \]

Processing B provides increases the lower bound \( \alpha \) to 3 for C

Processing C decreases the upper bound \( \beta \) to 2

Game tree

lower & upper bounds will be denoted \([\alpha, \beta]\)
bounding a node

See Dr. José Manuel Torres's alpha-beta animation (homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html)

tree structure:
2 2 2 2 2 1 2 2 1 2 2 1 2
terminal values:
3 17 2 12 15 25 0 2 5 3 2 14

bounding a node

Each node has $[\alpha, \beta]$:
- $\alpha$ - lower bound
- $\beta$ - upper bound
bounding a node

This case is tricky, let's look at the algorithm in detail before we proceed.

alpha-beta algorithm

alpha-beta search(state)
  v = maxvalue(state, α=−∞, β=∞)
  return action in actions(state) with value v

maxvalue(state, α, β)
  if terminal(state) then v = utility(state)
  else
    v = −∞
    for a ∈ actions(state)
      v = max(v, minvalue(result(state, a), α, β))
      if v ≥ β then break else α = max(α, v)
    return v

minvalue(state, α, β)
  if terminal(state) then v = utility(state)
  else
    v = ∞
    for a ∈ actions(state)
      v = min(v, maxvalue(result(state, a), α, β))
      if v ≤ α then break else β = min(β, v)
  return v

each node has [α, β]
α - lower bound
β - upper bound
minvalue(state, α, β)
if terminal(state) then \( v = utility(state) \) else
  \( v = \infty \)
  for \( a \in actions(state) \)
    \( v = \min(v, maxvalue(result(state, a), α, β)) \)
    if \( v \leq α \) then break else \( β = \min(β, v) \)
return \( v \)

maxvalue(state, α, β)
if terminal(state) then \( v = utility(state) \) else
  \( v = -\infty \)
  for \( a \in actions(state) \)
    \( v = \max(v, minvalue(result(state, a), α, β)) \)
    if \( v \geq β \) then break else \( α = \max(α, v) \)
return \( v \)

15>3 but minvalue only checks α
returns a 15 back to parent

Proceed along right subtree at home
final tree

• $\alpha - \beta$ pruning sensitive to order value
Move ordering

• killer move heuristic
  • iterative deepening search to ply above
  • use heuristic value to order nodes

• games frequently have repeated states
  • transposition table stores heuristic of visited states
  • only store good states (heuristic required)
  • favor states in transposition table

Okay, so you’re not Brad Pitt

Imperfect real-time decisions

• Replace utility function with an evaluation function
  • examines non-terminals
  • heuristic giving strength of current state

• How deep should we search?
  • cutoff test provides decision of whether to explore or apply evaluation function
Evaluation function

• Estimate of the expected utility
• Performance strongly linked to evaluation function choice.

• Guidelines
  • Order terminal states in the same order as the utility function, e.g. $u(a) \leq u(b) \leq u(c) \Rightarrow e(a) \leq e(b) \leq e(c)$
  • Estimator should be
    • correlated with odds of winning
    • fast

Features...

help us identify something based on state or percept.
Who has the better position?

Possible features for checkers?

Most features are relative to the opponent. Count player & opponent and subtract one from other

• Number of pieces
• Number of kings
• Offense
  • Advance of pawns (# moves away from being kinged... sum, mean, max)
  • Number of possible: moves, captures (by pawn/king?)
Possible features for checkers?

• Defense, are pieces
  • On edges of board? (better defended)
  • protected by neighbors of the same color (good) or menaced
    by neighbors of the opposite color (bad)?

• These features need to be weighted and combined,
  typically as a weighted linear combination

\[
f_{\text{eval}}(\text{board}) = \sum_{i=1}^{\text{#features}} w_i f_i(\text{board})
\]

• Determine weights?
  • genetic algorithms?
  • other learning techniques? (beyond our reach for now)

Pruning search

• Game-tree search → $$$

• Replace terminal-test with

  \[
  \text{if cutoff-test(state, depth), return } f_{\text{eval}}(\text{board})
  \]

• How do we define this?
cutoff-test

• Fixed depth – simple

• Timed iterative deepening
  • Run consecutively deeper searches
  • When time runs out, use deepest completed search

cutoff-test – Prefer quiescent states

• Sometimes, a state can change value radically in a few moves.

• Quiescence search
  • Search a few plies from a candidate cutoff node.
  • Look for relative stability of evaluation function → quiet state
Opening moves

- Ruy Lopez – 1400s
- Sicilian defense – 1600s
- Stonewall attack
- Queens gambit ca. 1490
- Modern defense
- Dutch defense

Lookup

- Reasonable to search billion game-tree nodes to move a pawn at beginning of game?
- Common to rely on table lookup for openings. Rely on human experience
Lookup and retrograde search

- Endgames
  - Have a reduced search space
  - retrograde search: Backwards minimax search backwards from terminal states

Stochastic games

- Consist of
  - probability distributions
  - strategy contingent on probabilities

- Game trees need to somehow incorporate chance

- We will review a few concepts about probability before diving in
Probability Distributions

- Show the probability of an event happening.
  - Distributions are nonnegative
  - Must sum to 1
- Example with a die
  - \( P(X=3) \) denotes the probability of rolling a 3
  - Fair die \( \rightarrow P(X=3) = \frac{1}{6} \)
- Distributions of continuous variables are sometimes described by parameterized formulae

Probability distributions

- Some games depend on independent events

- Example – Roll two dice

- Probabilities that are independent can be multiplied.

\[
P(R = 1)P(G = 6) = \frac{1}{6} \cdot \frac{1}{6} = \frac{1}{36}
\]

- Many times, we only care about the probability of 1 and 6, not which die produced it. How do we do this?
Expectation operator

\[ E[u(X)] = \sum_{x} u(X = x) P(X = x) \]

When \( u(X) = X \), we call this the mean, or average value:

\[ \mu = E[X] \]

Notes

- Distinguish between a random variable \( X \) and an instance of a random variable \( x \).
- The \( X=x \) notation is frequently dropped and just \( x \) is used.
- \( P(x) \) is frequently denoted as \( f(x) \)

Backgammon

![Backgammon board with black and white goals]
Backgammon

Rules in a nutshell (not everything)

• Move into goal by exact count
• Can capture opponent and send to bar by landing on a single opponent.
• Roll can be divided amongst pieces or combined

Backgammon

moves
blue + orange
blue + blue
blue + green
orange + orange
orange + red
green + red
Stochastic game search

• Incorporate chance nodes into min/max search tree

• Each edge denotes an outcome with a probability

• Minimax search is still the goal but how do we pick extrema in the face of uncertainty?

• For each chance node, compute the expected outcome
Stochastic minimax

\[
E_{\text{minimax}}(\text{searchnode}): \\
\text{switch type of searchnode:} \\
\quad \text{case terminal:} \\
\quad \quad \text{return utility(} \text{searchnode.} \text{state)} \\
\quad \text{case maxnode:} \\
\quad \quad \text{return arg max}_{a \in \text{actions}}(E_{\text{minimax}}(\text{result(} \text{searchnode,} \text{a)}))) \\
\quad \text{case minnode:} \\
\quad \quad \text{return arg min}_{a \in \text{actions}}(E_{\text{minimax}}(\text{result(} \text{searchnode,} \text{a)}))) \\
\quad \text{case chance:} \\
\quad \quad E_{\text{minimax}} = \sum_{r \in \text{roll}} P(r) \ E_{\text{minimax}}(\text{result(} \text{searchnode,} \text{r)}))
\]

note: multiple actions may be associated with a roll, so last case is a little more complicated.

Stochastic games and evaluation

• Not as clear as with deterministic games
• Consider possible values for three game states

<table>
<thead>
<tr>
<th>State</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>eval fn A</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>eval fn B</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>400</td>
</tr>
</tbody>
</table>

In a standard minimax routine, we would come up with the same solution, it is the relative ordering that is important.

What about here?
**Stochastic games and evaluation**

![Stochastic game tree diagram](image)

<table>
<thead>
<tr>
<th>State</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>eval fn A</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>eval fn B</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>400</td>
</tr>
</tbody>
</table>

**Stochastic minimax search**

- Chance dramatically increases the branching factor
- Common not to search for more than a few plies
- It is possible to modify alpha-beta pruning to work on bounds of expected values [beyond our scope]
Stochastic search

Monte-Carlo simulations are an alternate strategy

- Play millions of games using some search algorithm
- Assign state values based on results of wins/losses of the millions of simulated games

Partial information

- Not all games fully observable
- Basic ideas
  - maintain a belief state
  - use and-or trees to represent possible states
    - problematic: lots of states

you will not be held responsible for this material
additional example
alpha-beta pruning
maxvalue(state, α, β)
  if terminal(state) then v = utility(state)
  else
    v = -∞
    for a ∈ actions(state)
      v = max(v, minvalue(result(state, a), α, β))
    if v ≥ β then break else α = max(α, v)
  return v

minvalue(state, α, β)
  if terminal(state) then v = utility(state)
  else
    v = ∞
    for a ∈ actions(state)
      v = min(v, maxvalue(result(state, a), α, β))
    if v ≤ α then break else β = min(β, v)
  return v

alpha-beta pruning example

Note: When minvalue/maxvalue break their loop, they do not actually set α and β but for emphasis, we will show the values at nodes as if they did.
alpha-beta pruning example

```
maxvalue(state, α, β)
    if terminal(state) then v = utility(state)
    else
        v = -∞
        for a ∈ actions(state)
            v = max(v, minvalue(result(state, a), α, β))
        if v ≥ β then break else α = max(α, v)
    return v
```

```
minvalue(state, α, β)
    if terminal(state) then v = utility(state)
    else
        v = ∞
        for a ∈ actions(state)
            v = min(v, maxvalue(result(state, a), α, β))
        if v ≤ α then break else β = min(β, v)
    return v
```

node c ≤ 2, node a already has α=3 no need to look further...
alpha-beta pruning example

minvaluelresult(A,actionAtoD),3,
children of D have utilities [14, 5, 2]
\nu = \min(-\infty, 14) = 14
\nu = \min(14, 5) = 5
\nu = \min(5, 2) = 2