Adversarial Search

- Basic minimax search
- Evaluation Functions (resource limited)
- Alpha beta pruning
- Techniques used in board playing
  - World champion of Checkers, Othello is a machine (http://www.cs.ualberta.ca/~chinook/)
  - Machines challenging in Chess, Backgammon

Games and Search

“Unpredictable” opponent ⇒ solution is a contingency plan
Time limits ⇒ unlikely to find goal, must approximate
Plan of attack:

- algorithm for perfect play (Von Neumann, 1944)
- finite horizon, approximate evaluation (Zuse, 1945; Shannon, 1950; Samuel, 1952–57)
- pruning to reduce costs (McCarty, 1956)
Game Trees and Minimax

Perfect play for deterministic, perfect-information games
Idea: choose move to position with highest minimax value
= best achievable payoff against best play
E.g., 2-ply game:

```
function Minimax-Decision(game) returns an operator
  for each op in OPERATORS(game) do
    VALUE[op] ← Minimax-Value(Apply(op, game), game)
  end
  return the op with the highest VALUE[op]

function Minimax-Value(state, game) returns a utility value
  if TERMINAL-Test(game)(state) then
    return UTILITY(game)(state)
  else if MAX is to move in state then
    return the highest Minimax-Value of SUCCESSORS(state)
  else
    return the lowest Minimax-Value of SUCCESSORS(state)
```

Minimax Example
Properties of Minimax Search

Complete?? Yes, if tree is finite (chess has specific rules for this)
Optimal?? Yes, against an optimal opponent. Otherwise??
Time complexity?? $O(b^n)$
Space complexity?? $O(bm)$ (depth-first exploration)
For chess, $b \approx 35$, $m \approx 100$ for “reasonable” games
⇒ exact solution completely infeasible
Three Player Games

How do we create game trees that incorporate chance?

Chance as a Bipartisan Third Player

In nondeterministic games, chance introduced by dice, card-shuffling.
Simple example with coin-flipping.

\[
\begin{align*}
\text{MAX} & \quad \text{CHANCE} & \quad \text{MIN} \\
& \quad 2 \quad 0.5 \quad 3 \quad 0.5 \quad -1 \\
2 \quad 4 \quad 7 \quad 6 \quad 0 \quad 5 \quad -2 \\
\end{align*}
\]

EXPECTIMININAX gives perfect play

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

\[
\begin{align*}
\text{if } \text{state} \text{ is a MAX node then} & \quad \text{return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)} \\
\text{if } \text{state} \text{ is a MIN node then} & \quad \text{return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)} \\
\text{if } \text{state} \text{ is a chance node then} & \quad \text{return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(state)} \\
\end{align*}
\]

...
Depth Limited Search

\textbf{MINIMAXCUTOFF} is identical to \textbf{MINIMAXVALUE} except
1. \texttt{TERMINAL?} is replaced by \texttt{CUTOFF}?
2. \texttt{UTILITY} is replaced by \texttt{EVAL}

Evaluation function is just a heuristic (not an admissible heuristic)
Break game state into m components
Eval\( (n) = w_1c_1(n) + w_2c_2(n) + \ldots + w_mc_m(n) \)
Note multiple states have the exact same component values
Eval function some estimate of the the expectation of
\#wins/\#games from this point (as described by the components)

Evaluation Functions

For chess, typically \textit{linear} weighted sum of features
\[ \text{Eval}(s) = w_1f_1(s) + w_2f_2(s) + \ldots + w_nf_n(s) \]
e.g., \( w_1 = 9 \) with
\( f_1(s) = (\text{number of white queens}) - (\text{number of black queens}) \)
etc.
Example – 3D Tic-Tac-Toe

Depth of tree?
Number of nodes?
Evaluation function?
What are the components?
How should we score each component?

Close Enough is Good Enough

MAX

MIN

Behaviour is preserved under any monotonic transformation of Eval.
Only the order matters:
    payoff in deterministic games acts as an ordinal utility function
Cutting of Search

MINIMAXCUTOFF is identical to MINIMAXVALUE except
1. TERMINAL? is replaced by CUTOFF?
2. UTILITY is replaced by EVAL

Does it work in practice?

\[ b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4 \]

4-ply lookahead is a hopeless chess player!

4-ply \( \approx \) human novice
8-ply \( \approx \) typical PC, human master
12-ply \( \approx \) Deep Blue, Kasparov

---

Alpha Beta Pruning

- Some branches will never be played by rational players since they include sub-optimal decisions (for either player)

What is vital to the success of this approach?
**Result**

![Game Tree Diagram]

- **nodes that were never explored !!!**

---

**Alpha-Beta pruning**

```plaintext
function Max-Value(state, game, α, β) returns the minimax value of state
  input: state, current state in game
         game, game description
         α, the best score for MAX along the path to state
         β, the best score for MIN along the path to state
  if GOAL-TEST(state) then return EVAL(state)
  for each s in SUCCESSORS(state) do
    α' ← Max(Max-Value(s, game, α, β))
    if α' ≥ β then return β
  end
  return α

function Min-Value(state, game, α, β) returns the minimax value of state
  input: state, current state in game
         game, game description
         α, the best score for MAX along the path to state
         β, the best score for MIN along the path to state
  if GOAL-TEST(state) then return EVAL(state)
  for each s in SUCCESSORS(state) do
    β' ← Min(Max-Value(s, game, α, β))
    if β' ≤ α then return α
  end
  return β
```
In Practice

Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.


Othello: human champions refuse to compete against computers, who are too good.

Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

Minimax Example Zero Sum Game

Effectively depth first search, time complexity $O(b^d)$
Properties of Minimax Search

Complete?? Yes, if tree is finite (chess has specific rules for this)
Optimal?? Yes, against an optimal opponent. Otherwise??
Time complexity?? $O(b^m)$
Space complexity?? $O(bm)$ (depth-first exploration)
For chess, $b \approx 35$, $m \approx 100$ for “reasonable” games
⇒ exact solution completely infeasible

Three Player and Non-Zero Sum Games

How do we create game trees that incorporate chance?
**Chance as a Bipartisan Third Player**

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

Simplified example with coin-flipping:

Before each link implicitly
Had Pr=1 on it.

**Depth Limited Search**

**MINIMAXCUTOFF** is identical to **MINIMAXVALUE** except
1. **TERMINAL?** is replaced by **CUTOFF**?
2. **UTILITY** is replaced by **EVAL**

Evaluation function is just a heuristic (not an admissible heuristic)
Break game state into m components
Eval(n) = w_1c_1(n)+w_2c_2(n)...w_mc_m(n)
Note multiple states have the exact same component values
Eval function some estimate of the the expectation of
#wins/#games from this point (as described by the components)

State space, evaluation function for a single player
checkers program?
Checkers

Evaluation function features

Samuel experimented with Over 50. Recent work looks at learning the feature weights. Simplest evaluation function

\[ F(n) = 3.c_3 + 2.c_2 + 1.c_1 + 30.a_3 + 20.a_2 + 10.a_1 \]

Alpha Beta Pruning

- Produces exact same results as minimax algorithm
- Reduces search by not expanding sub-optimal paths
- Alpha and Beta terms are the floor and ceiling of the range of values, the player is interested in.
Alpha-beta pruning example: Step 1
\( \alpha \) is the maximum lower bound of possible solutions (MAX plays)
\( \beta \) is the minimum upper bound of possible solutions (MIN plays)

Max fills left hand alpha value
Min fills in right beta value
Parent can set current best
Estimates for alpha and beta

For this path, worst min can do is 5

Alpha-beta pruning example: Step 2
\( \alpha \) is the maximum lower bound of possible solutions (MAX plays)
\( \beta \) is the minimum upper bound of possible solutions (MIN plays)

For this path, worst max can do is 5
Alpha-beta pruning example: Step 4
- $\alpha$ is the maximum lower bound of possible solutions (MAX plays)
- $\beta$ is the minimum upper bound of possible solutions (MIN plays)

Alpha-beta pruning example: Step 5
- $\alpha$ is the maximum lower bound of possible solutions (MAX plays)
- $\beta$ is the minimum upper bound of possible solutions (MIN plays)

Alpha-beta pruning example: Step 6
- $\alpha$ is the maximum lower bound of possible solutions (MAX plays)
- $\beta$ is the minimum upper bound of possible solutions (MIN plays)

Alpha-beta pruning example: Step 7
- $\alpha$ is the maximum lower bound of possible solutions (MAX plays)
- $\beta$ is the minimum upper bound of possible solutions (MIN plays)
Alpha-beta pruning example: Step 8
α is the maximum lower bound of possible solutions (MAX plays)
β is the minimum upper bound of possible solutions (MIN plays)

All branches “complete”

Alpha-beta pruning example: Step 9
α is the maximum lower bound of possible solutions (MAX plays)
β is the minimum upper bound of possible solutions (MIN plays)

Alpha-beta pruning example: Step 10
α is the maximum lower bound of possible solutions (MAX plays)
β is the minimum upper bound of possible solutions (MIN plays)

Alpha-beta pruning example: Step 11
α is the maximum lower bound of possible solutions (MAX plays)
β is the minimum upper bound of possible solutions (MIN plays)
Alpha-beta pruning example: Step 12
\( \alpha \) is the maximum lower bound of possible solutions (MAX plays)
\( \beta \) is the minimum upper bound of possible solutions (MIN plays)

Alpha-beta pruning example: Step 13
\( \alpha \) is the maximum lower bound of possible solutions (MAX plays)
\( \beta \) is the minimum upper bound of possible solutions (MIN plays)

Alpha-beta pruning example: Step 14
\( \alpha \) is the maximum lower bound of possible solutions (MAX plays)
\( \beta \) is the minimum upper bound of possible solutions (MIN plays)

Alpha-beta pruning example: Step 15
\( \alpha \) is the maximum lower bound of possible solutions (MAX plays)
\( \beta \) is the minimum upper bound of possible solutions (MIN plays)
Initial call is to MAX_VALUE(root_node, -∞, ∞)

Alpha <= Beta, otherwise cut-off search

**Alpha-Beta pruning**

**Max of min values**

```
function MAX_VALUE(state, game, α, β) returns the minimax value of state
inputs: state, current state in game
        game, game description
        α, the best score for MAX along the path to state
        β, the best score for MIN along the path to state
if GOAL-TEST(state) then return EVAL(state)
for each s in SUCCESSORS(state) do
    β ← MIN(β, MIN_VALUE(s, game, α, β))
if α ≥ β then return β
end
return β
```

**Min of max values**

```
function MIN_VALUE(state, game, α, β) returns the minimax value of state
inputs: state, current state in game
        game, game description
        α, the best score for MAX along the path to state
        β, the best score for MIN along the path to state
if GOAL-TEST(state) then return EVAL(state)
for each s in SUCCESSORS(state) do
    β ← MAX(β, MAX_VALUE(s, game, α, β))
if β ≤ α then return β
end
return β
```
Analysis of Alpha Beta Pruning

  - Most results for perfectly ordered trees
  - Some for randomly ordered trees

Performance analysis of Alpha-Beta Pruning

- Since alpha-beta pruning performs a minimax search while pruning much of the tree, its effect is to allow a deeper search with the same amount of computation.

- **The question:** how much does alpha-beta improve performance?
Example of alpha-beta worst case

- Evaluation from left to right

Minimax value of game trees

- The most natural definition for the average case is that the leaf nodes are randomly ordered.
- Heuristic node ordering would violate this assumption.
- Average case performance is not a prediction of its performance in practice