Heuristic Alpha-Beta Tree Search

- Cut off the search early by applying a heuristic evaluation function.
- Replace UTILITY with EVAL, which estimates a state's utility.

Heuristic Alpha-Beta Tree Search

- Cut off the search early by applying a heuristic evaluation function.
- Replace UTILITY with EVAL, which estimates a state's utility.

H-MINIMAX(s, d) = // s: state // d: search depth

$$\begin{cases} \text{EVAL}(s, \text{MAX}) \text{ or } \text{EVAL}(s, \text{MIN}) & \text{if IS-CUTOFF}(s, d) \\ \max\limits_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s, a), d+1) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min\limits_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s, a), d+1) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

Evaluation Functions

EVAL(s, p) returns an estimate of the expected utility s to player p.

- EVAL(s, p) = UTILITY(s, p) if s is terminal;
- UTILITY$(loss, p) \leq$ EVAL$(s, p) \leq$ UTILITY(win, p) if s is nonterminal.

Evaluation Functions

EVAL(s, p) returns an estimate of the expected utility s to player p.

- EVAL(s, p) = UTILITY(s, p) if s is terminal;
- UTILITY$(loss, p) \leq$ EVAL$(s, p) \leq$ UTILITY(win, p) if s is nonterminal.

Criteria:

- No excessive computation time.
- Strong correlation with actual chances of winning.

Eval Function: State Categorization

- Calculate various features of the state (e.g., #pawns, #queens in chess).

Eval Function: State Categorization

- Calculate various features of the state (e.g., #pawns, #queens in chess).
- Define categories (equivalence classes) of states (e.g., all two-pawn vs one-pawn endgames).
  - Each category may contain states leading to wins, draws, and losses.
  - Nevertheless, all such states have the same feature values.

Eval Function: State Categorization

- Calculate various features of the state (e.g., #pawns, #queens in chess).
- Define categories (equivalence classes) of states (e.g., all two-pawn vs one-pawn endgames).
  - Each category may contain states leading to wins, draws, and losses.
  - Nevertheless, all such states have the same feature values.
- Determine an expected value for each category.

Eval Function: State Categorization

- Calculate various features of the state (e.g., #pawns, #queens in chess).
- Define categories (equivalence classes) of states (e.g., all two-pawn vs one-pawn endgames).
  - Each category may contain states leading to wins, draws, and losses.
  - Nevertheless, all such states have the same feature values.
- Determine an expected value for each category.

e.g., two-pawns vs. one-pawn

$1 \mapsto$	wins	82%
$0 \mapsto$	losses	2%
$0.5 \mapsto$	draws	16%

$$(0.82 \times 1) + (0.02 \times 0) + (0.16 \times 0.5) = 0.9$$

Eval Function: State Categorization

- Calculate various features of the state (e.g., #pawns, #queens in chess).
- Define categories (equivalence classes) of states (e.g., all two-pawn vs one-pawn endgames).
  - Each category may contain states leading to wins, draws, and losses.
  - Nevertheless, all such states have the same feature values.
- Determine an expected value for each category.

e.g., two-pawns vs. one-pawn

$1 \mapsto$	wins	82%
$0 \mapsto$	losses	2%
$0.5 \mapsto$	draws	16%

$$(0.82 \times 1) + (0.02 \times 0) + (0.16 \times 0.5) = 0.9$$

- Too many categories and too much dependence on experience.

Eval Function: Feature Combination

Compute separate numerical contributions from each feature and combine them.

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$$

weight e.g., #pawns in chess

Eval Function: Feature Combination

Compute separate numerical contributions from each feature and combine them.

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$$

weight e.g., #pawns in chess

- Strongly correlated with the chance of winning.

Eval Function: Feature Combination

Compute separate numerical contributions from each feature and combine them.

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$$

weight e.g., #pawns in chess

- Strongly correlated with the chance of winning.
- But not necessarily linearly correlated.

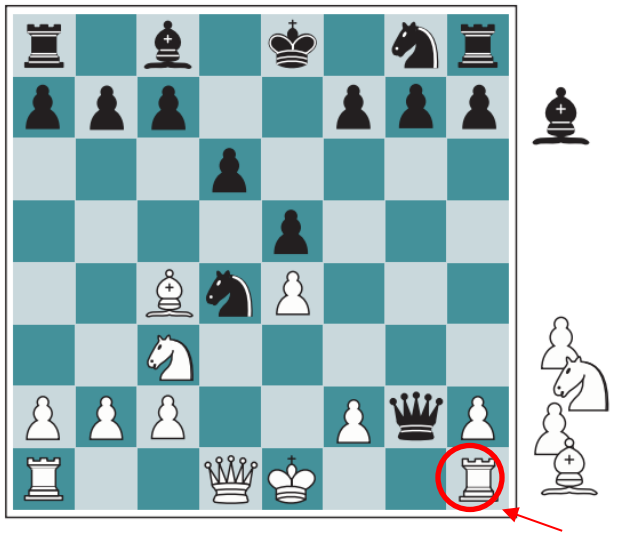
Eval Function: Feature Combination

Compute separate numerical contributions from each feature and combine them.

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$$

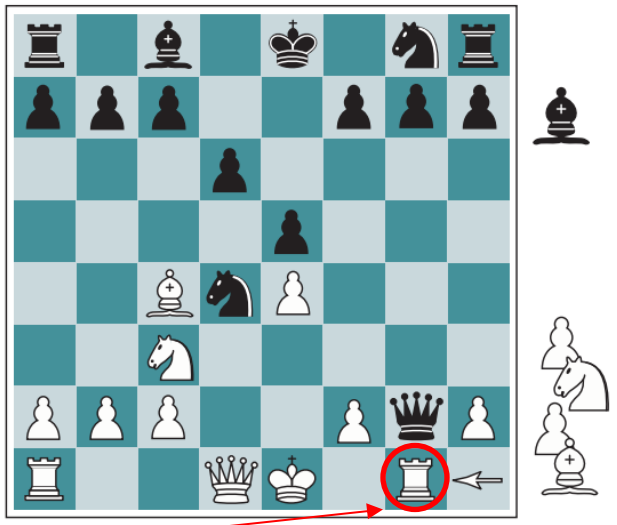
weight e.g., #pawns in chess

- Strongly correlated with the chance of winning.
- But not necessarily linearly correlated.



(a) White to move

Only difference



(b) White to move

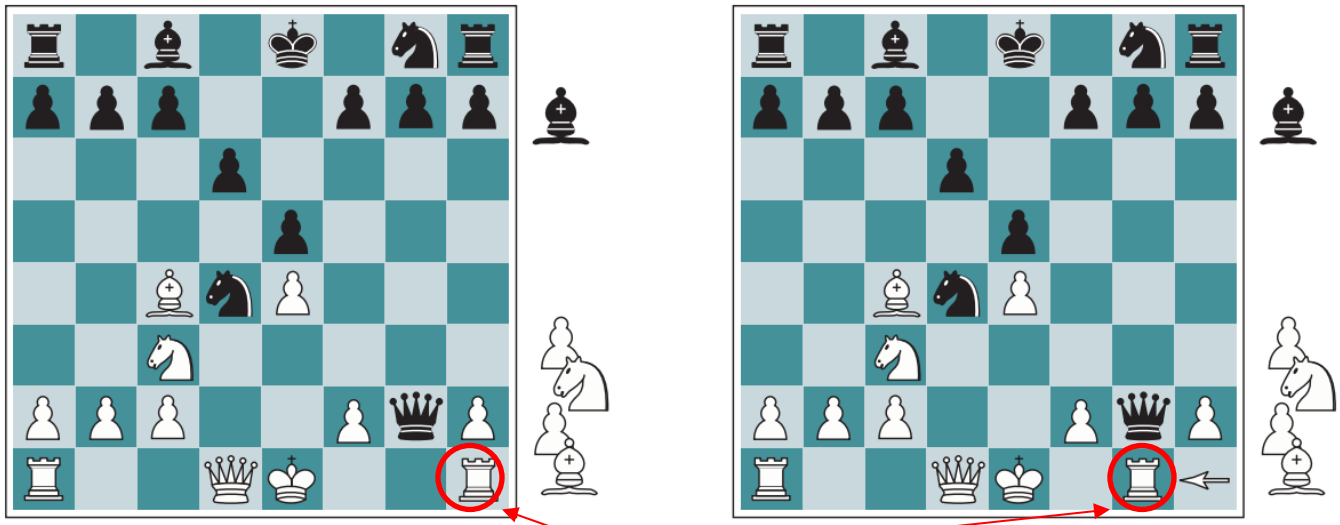
Eval Function: Feature Combination

Compute separate numerical contributions from each feature and combine them.

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$$

weight e.g., #pawns in chess

- Strongly correlated with the chance of winning.
- But not necessarily linearly correlated.



(a) White to move Only difference (b) White to move

Black should win because of an advantage (1 knight & 2 pawns)

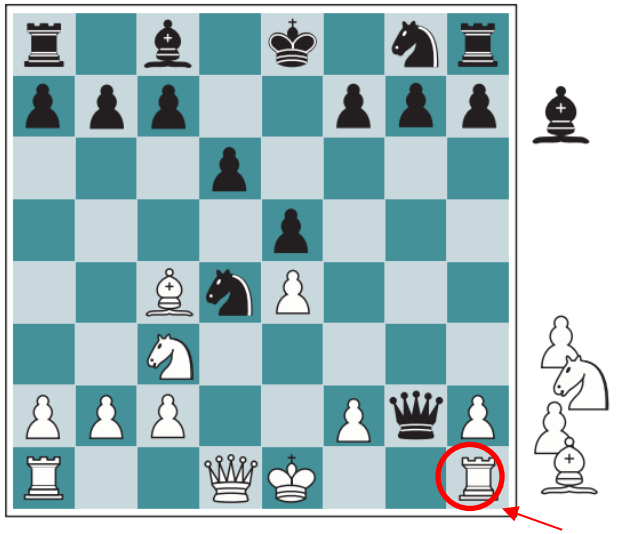
Eval Function: Feature Combination

Compute separate numerical contributions from each feature and combine them.

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$$

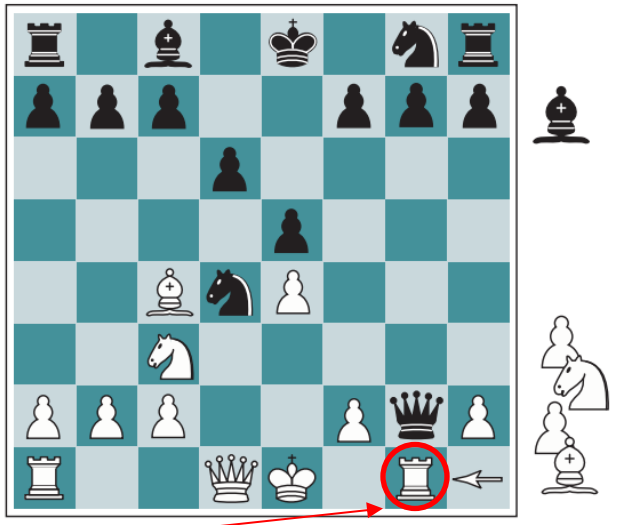
weight e.g., #pawns in chess

- Strongly correlated with the chance of winning.
- But not necessarily linearly correlated.



(a) White to move

Only difference



(b) White to move

Black should win because of an advantage (1 knight & 2 pawns)

White should win because its rook will capture the queen.

Eval Function (cont'd)

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$$

♠ Assumes independent feature contributions.

♦ Use a nonlinear feature combination.

e.g., two bishops might be worth more than twice the value of a single bishop.

Cutting off Search

if $game.$~~IS-TERMINAL~~$(state)$ **then return** $game.$~~UTILITY~~$(state, player), null$

IS-CUTOFF

EVAL

Cutting off Search

if $game.$~~IS-TERMINAL~~$(state)$ **then return** $game.$~~UTILITY~~$(state, player), null$

IS-CUTOFF EVAL

Some strategies:

- Set a fixed depth limit d to control the amount of search.

  IS-CUTOFF returns true if depth $> d$.

Cutting off Search

if $game.$~~IS-TERMINAL~~$(state)$ **then return** $game.$~~UTILITY~~$(state, player), null$

IS-CUTOFF EVAL

Some strategies:

- Set a fixed depth limit d to control the amount of search.

  IS-CUTOFF returns true if depth $> d$.

- Apply iterative deepening:

  When time runs out, returns the move selected by the deepest completed search.

Real-Time Decisions

- Minimax with alpha-beta pruning.
- Extensively tuned evaluation function.
- Pruning heuristics.
- A transposition table of repeated states and evaluations.
  - To avoid re-searching the game tree below that state.
- A large database of optimal opening and endgame moves.
  - Table lookup instead of search.
  - Chess endgames with up to 7 pieces solved.
- Minimax unsuccessful in Go.