Artificial Intelligence

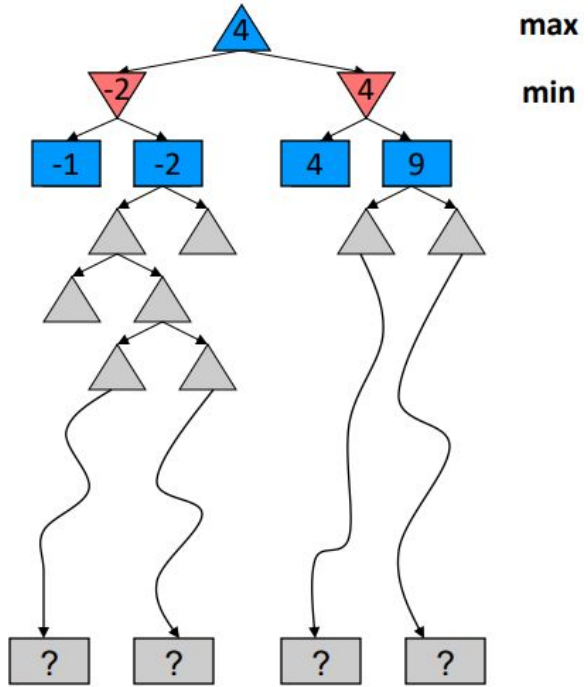
Lec 11: Adversarial Search (contd.)

Pratik Mazumder

Content taken from various sources including
CS188 Intro to AI at UC Berkeley

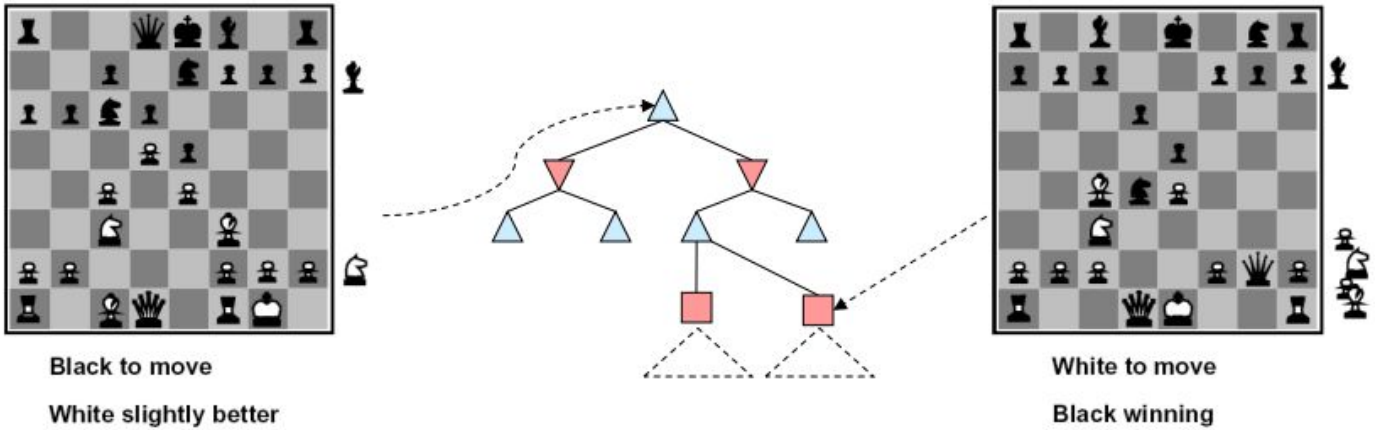
Overcoming Resource Limits: Limiting Depth

- Problem: In realistic games, you cannot search upto leaves!
- Solution: Depth-limited search
  - Instead, search only to a limited depth in the tree
  - Use an evaluation function for non-terminal positions
- Example:
  - Suppose we have 100 seconds for a move, and can explore 10K nodes per second.
  - So can check 1M nodes per move
- Guarantee of optimal play is gone.
- More plies/moves makes a BIG difference.
- Use iterative deepening.



Evaluation Function

- Evaluation functions score non-terminals in depth-limited search



- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

- e.g. f1(s) = (num white queens – num black queens), etc.

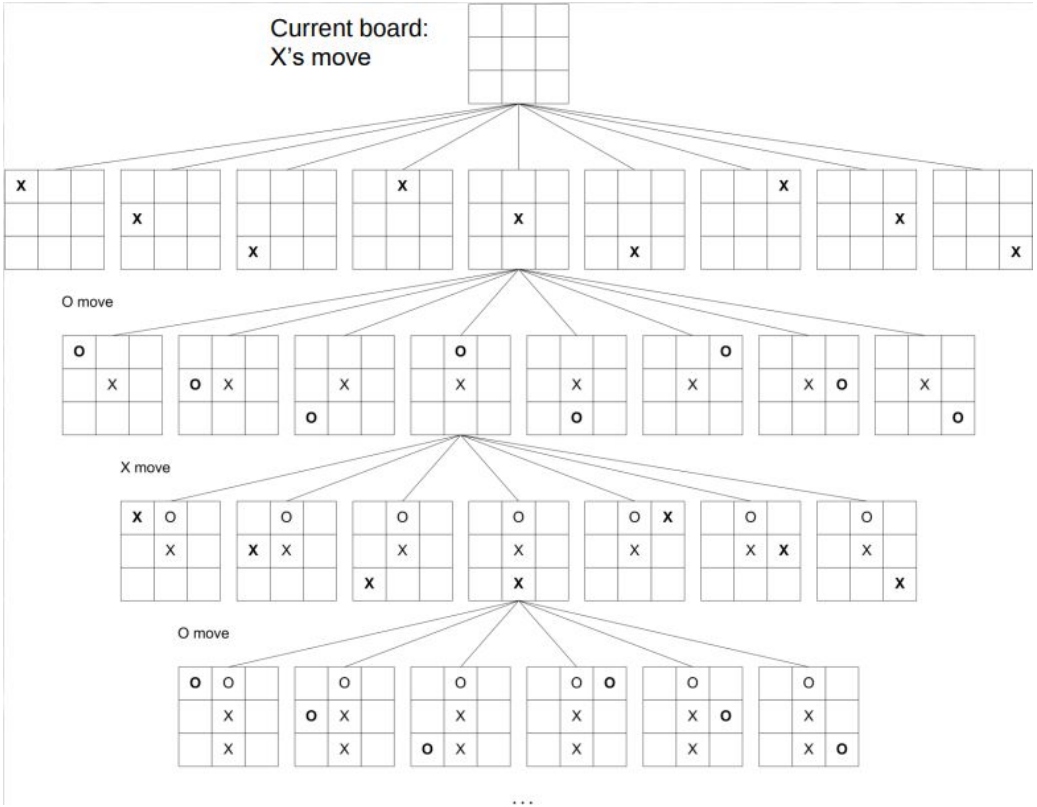
Evaluation Functions

- The performance of a game-playing program depends strongly on the quality of its evaluation function
  - but how to design a good evaluation function?
  - The evaluation function should order the terminal states in the same way as the true utility function
    - states that are wins must evaluate better than draws, etc.
  - The computation must not take too long.
  - For nonterminal states, the evaluation function should be strongly correlated with the actual “chances of winning.”
- Note that if the search must be cut off at nonterminal states, then the algorithm will necessarily be uncertain about the final outcomes of those states.

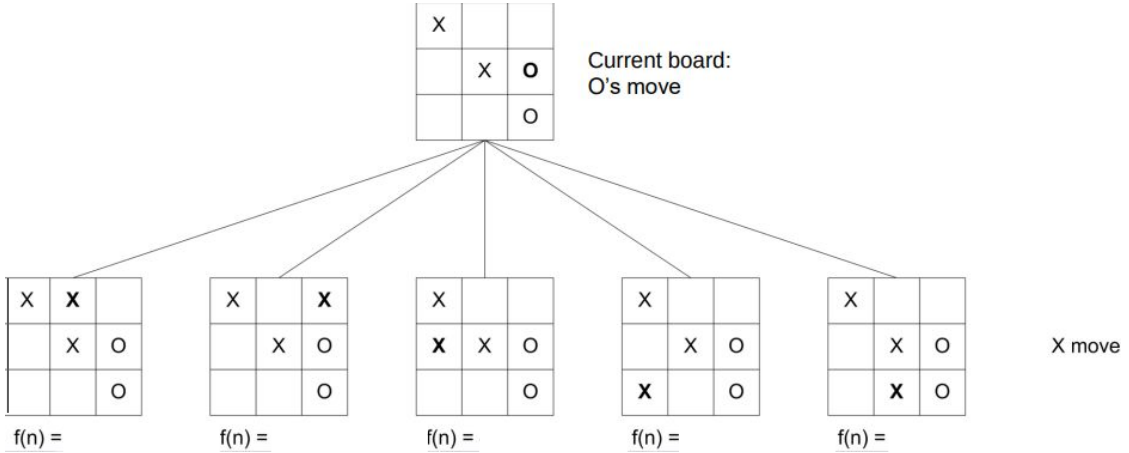
Evaluation Functions

- Usually, an evaluation function **calculates various features of a state** (e.g., number of pawns, etc.).
- The features define various categories or classes of states:
  - the states in each category have the same values for all features.
- A **better feature value** generally corresponds to a **better evaluation function value** for that state.
  - Indicates a **better chance of a higher final utility value** if we follow down the path from this state to an actual terminal state.

Evaluation Functions: Example



Evaluation Functions: Example

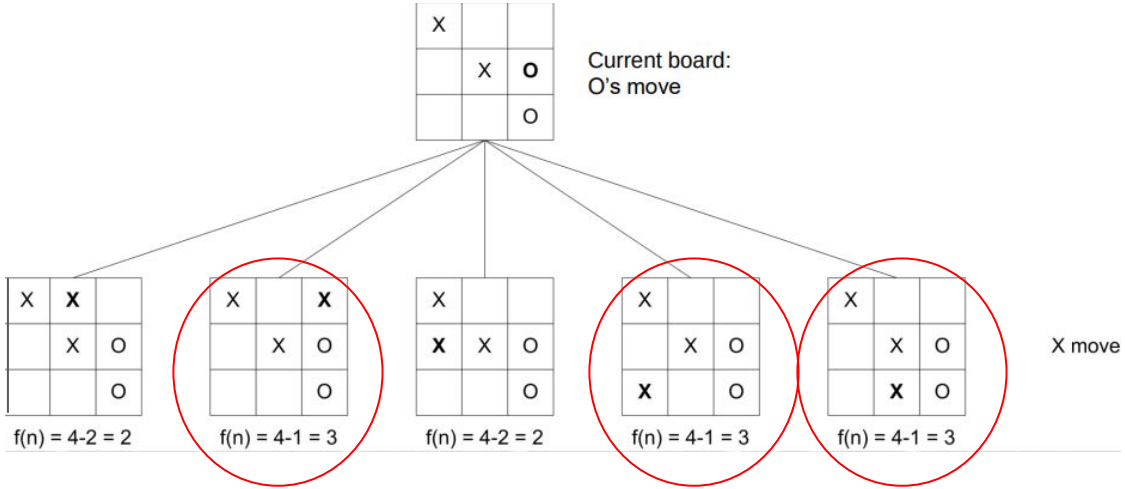


Evaluation function 1:

- Say Nx is the number of rows, columns, diagonals (not blocked by opponent) in which player X can still win,
- Say No is the number of rows, columns, diagonals (not blocked by opponent) in which player O can still win,
- Eval(n) = f(n) = Nx - No
- Positive number if state n has more chance of X winning and Negative number if it has more chance of O winning
- Evaluation function f(n) measures “goodness” of board configuration n.

Assumed to be better estimate as search is deepened (i.e., at lower levels of game tree).

Evaluation Functions: Example

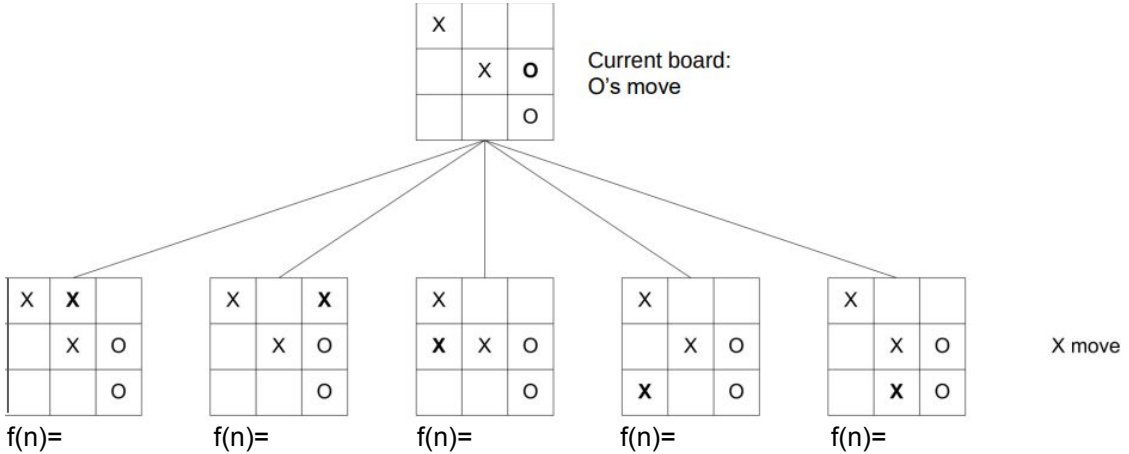


Evaluation function 1:

- Say Nx is the number of rows, columns, diagonals (not blocked by opponent) in which player X can still win,
- Say No is the number of rows, columns, diagonals (not blocked by opponent) in which player O can still win,
- Eval(n) = f(n) = Nx - No
- Positive number if state n has more chance of X winning and Negative number if it has more chance of O winning
- Evaluation function f(n) measures “goodness” of board configuration n.

Assumed to be better estimate as search is deepened (i.e., at lower levels of game tree).

Evaluation Functions: Example

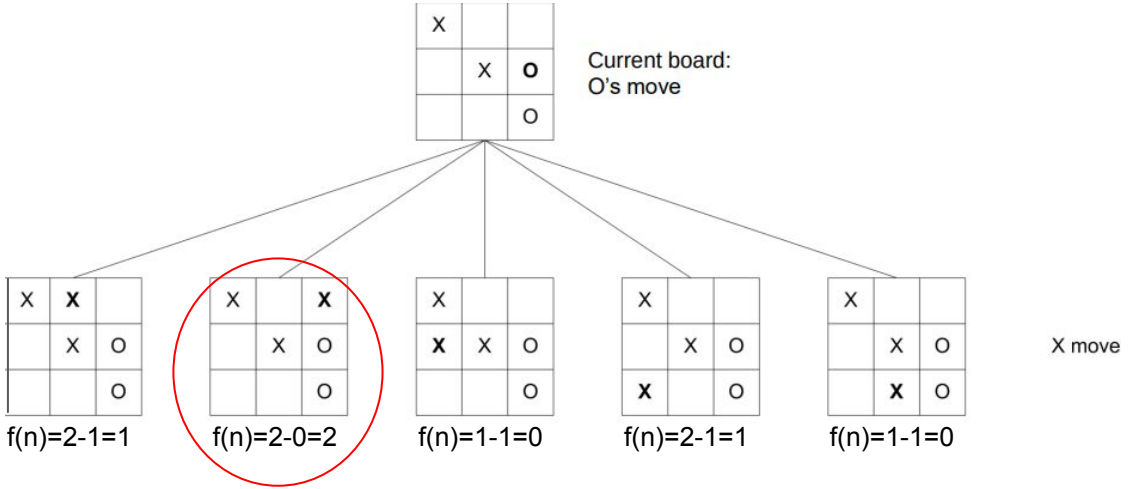


Another Evaluation function:

- Say Nx is the number of rows, columns, diagonals containing 2 X's and no O's
- Say No is the number of rows, columns, diagonals containing 2 O's and no X's
- Eval(n) = f(n) = Nx - No
- Limitation
  - Cannot apply at initial states
- Advantage
  - Identifies better states

Which state will you normally take?

Evaluation Functions: Example



Another Evaluation function:

- Say Nx is the number of rows, columns, diagonals containing 2 X's and no O's
- Say No is the number of rows, columns, diagonals containing 2 O's and no X's
- Eval(n) = f(n) = Nx - No
- Limitation
  - Cannot apply at initial states
- Advantage
  - Identifies better states

Which state will this evaluation function give highest value?

Evaluation Function

- The previous evaluation functions are based on **one property or feature** of the current state of the game.
  - Very **difficult** to come up with **one best reliable property or feature**.

- Most evaluation functions compute **separate numerical contributions from each feature** and then combine them to find the total value.

  - Typical evaluation function is a linear sum of features.

  - Eval(s) = $w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$

    - $w_1 = 9$

    - $f_1(s)$ = number of white queens – number of black queens

    - $f_2(s)$ = number of white bishops – number of black bishops

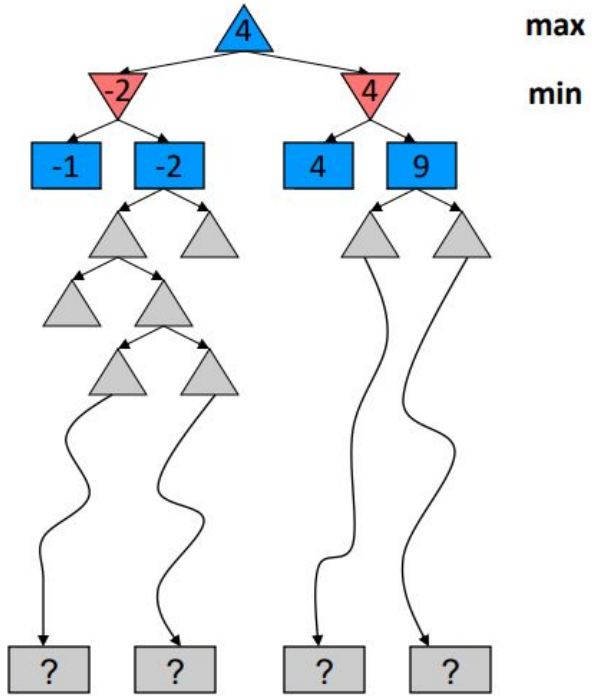
    - and so on

Evaluation Function

- Weighted combination of the output from different features, require the **weights to be appropriate**.
- At an intermediate state s, say $Eval(s) = w_1f_1(s) + w_2f_2(s) + \ldots + w_nf_n(s)$, with some initial $w_1, w_2, .., w_n$ say 1
- Assume that players A (you) and B have the same values of the weights.
- Fix the weights of B and update the weights of A to win against B
  - How to update
    - $\delta(n)$ = backed-up-value(n) - computed-value(n)
    - [Better than assumption] If $\delta > 0$, then the features that had contributed a positive value are given more weight (increased by a fixed amount) and the features that had contributed a negative value are given less weight (decreased by a fixed amount)
    - [Worse than assumption] If $\delta > 0$, then the features that had contributed a negative value are given more weight (increased by a fixed amount) and the features that had contributed a positive value are given less weight (decreased by a fixed amount)
- Give the learned weights to B and repeat the process to learn better weights for A

Cutting Off Search

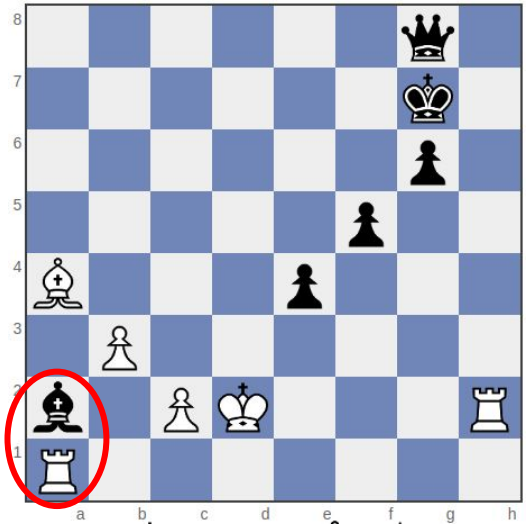
- Expand the game tree by m ply (levels in the game tree).
- Cannot get utility function at states at lowest level after cutoff if not terminal states, so assume them to be pseudo terminal states.
- Then apply the evaluation function at the lowest level and propagate the results back up the tree.
- Replace terminal test (end of game) by cutoff test (don't search deeper)
- Replace utility function (win/lose/draw) by heuristic evaluation function that estimates results on the best path below this board
    - Like A* search, good evaluation functions mean good results (and vice versa)
- Replace the move generator with a plausible move generator (don't consider "dumb" moves).



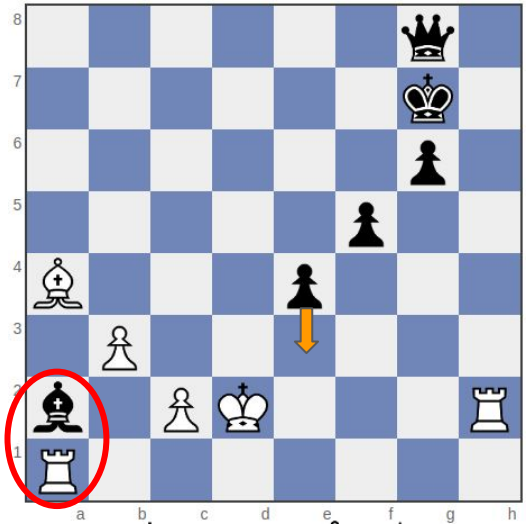
Cutting Off Search

- A more sophisticated type of cut off search applies evaluation functions only to positions that are quiescent (meaning: unlikely to exhibit wild swings in value in the near future).
- Non-quiescent positions can be expanded further until quiescent positions are reached.
- This extra search is called a quiescent search; sometimes it is restricted to consider only certain types of moves (e.g., capture moves).
- To mitigate the horizon effect, one can use singular extension (apply a move that is clearly better than all others at a given position).

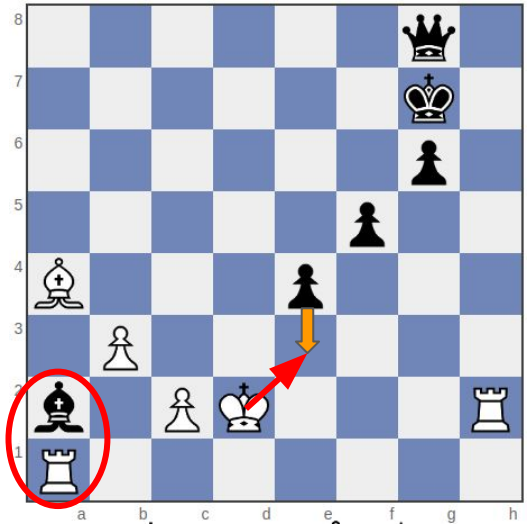
Problem with fixed depth Searches: Horizon Effect



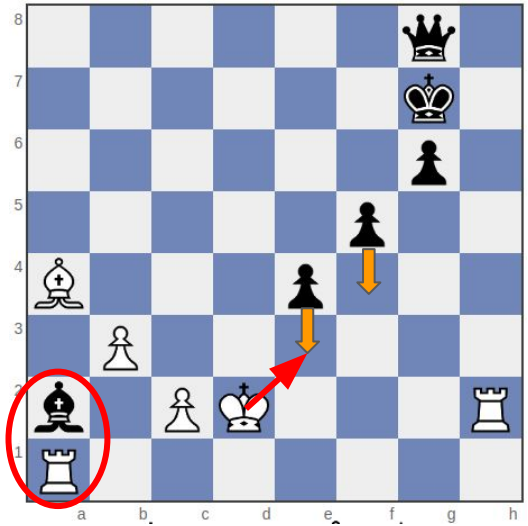
Problem with fixed depth Searches: Horizon Effect



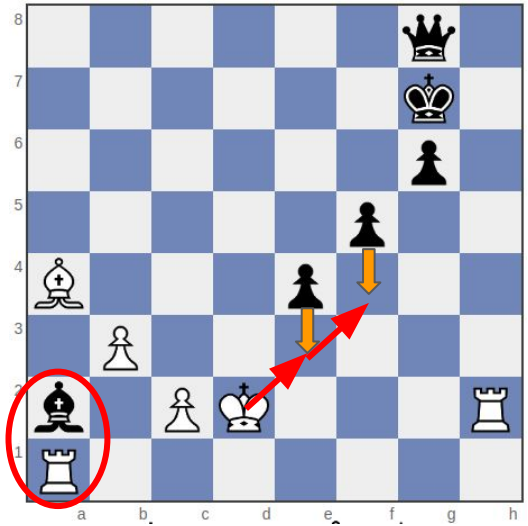
Problem with fixed depth Searches: Horizon Effect



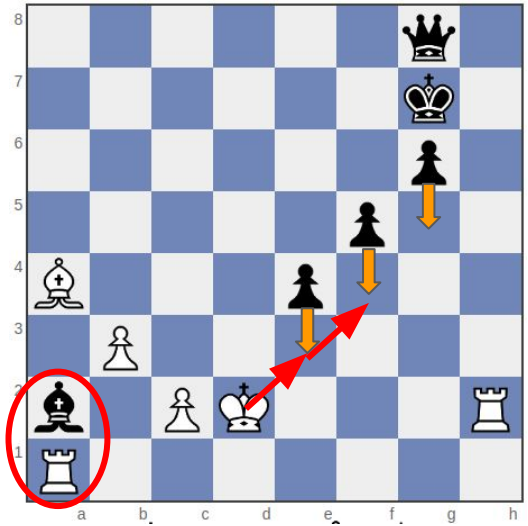
Problem with fixed depth Searches: Horizon Effect



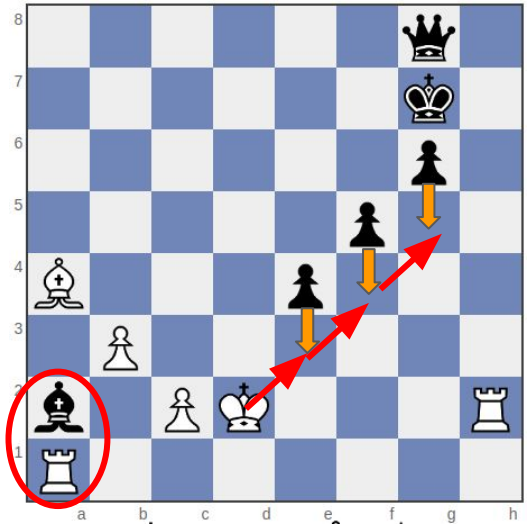
Problem with fixed depth Searches: Horizon Effect



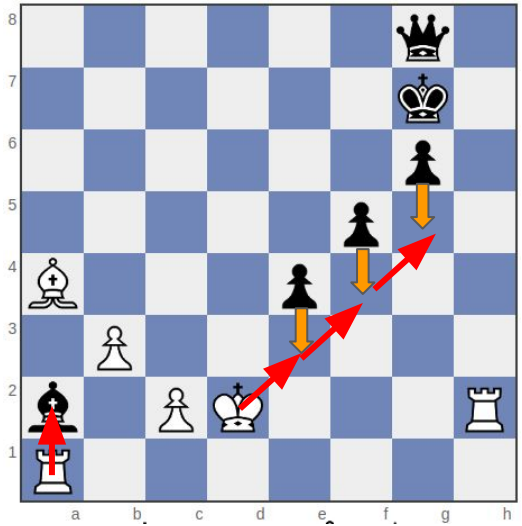
Problem with fixed depth Searches: Horizon Effect



Problem with fixed depth Searches: Horizon Effect



Problem with fixed depth Searches: Horizon Effect



Problem with fixed depth Searches: Horizon Effect

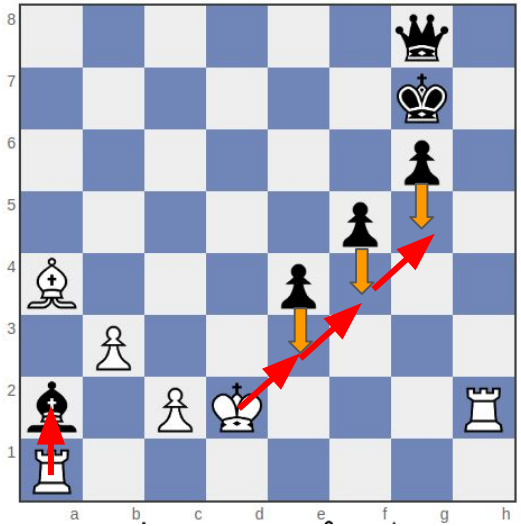
- If we **only search n moves ahead** (depth limited/cutoff), we **may not see** the **catastrophy** that might occur only **one or few moves later**.

- Horizon Effect: Arises when the program is facing an opponent's move that causes serious damage and is ultimately **unavoidable**, but can be t**emporarily avoided by delaying tactic**.

- With Black to move, the black bishop is surely doomed.
- But Black can forestall that event by checking the white king with its pawns, forcing the king to capture the pawns.

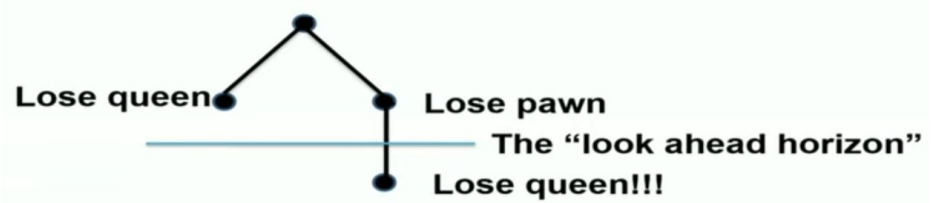
- This pushes the inevitable loss of the bishop over the horizon (depth limit), and thus the pawn sacrifices are seen by the search algorithm as good moves rather than bad ones.

- Black thinks that the line of play has saved the bishop at the price of two pawns, when actually all it has done is push the inevitable capture of the bishop beyond the horizon that Black can see.



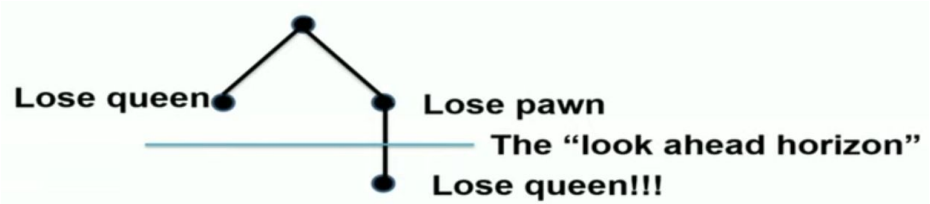
Problem with fixed depth Searches: Horizon Effect

- Inevitable losses are postponed
- Unachievable goals appear achievable
- Short-term gains mask unavoidable consequences.



- Countering the Horizon Effect
  - Feedover
    - Do not cut off search at non-quiescent board positions (dynamic positions)
    - Example, king in danger (check), some other piece in immediate danger
    - Keep searching down the path until you reach quiescent (stable) states
  - Secondary Search
    - Search further down a selected path to ensure it is the best move
  - Openings/Endgames
    - For some part of the game (especially initial and end moves), keep a catalog of the best moves to make

Problem with fixed depth Searches: Horizon Effect



- Singular Extensions: One strategy to mitigate the horizon effect is the singular extension, a move that “clearly better” than all other moves in a given position.
  - Idea: Find obviously good moves and try them at cutoff.
  - Once discovered anywhere in the tree in the course of a search, this singular move is remembered.
  - When the search reaches the normal depth limit, the algorithm checks to see if the singular extension is a legal move
    - if it is, the algorithm allows the move to be considered.
  - This makes the tree deeper, but because there will be few singular extensions, it does not add many total nodes to the tree.