Algorithms for Data Structures: Search for Games

Phillip Smith

27/11/13

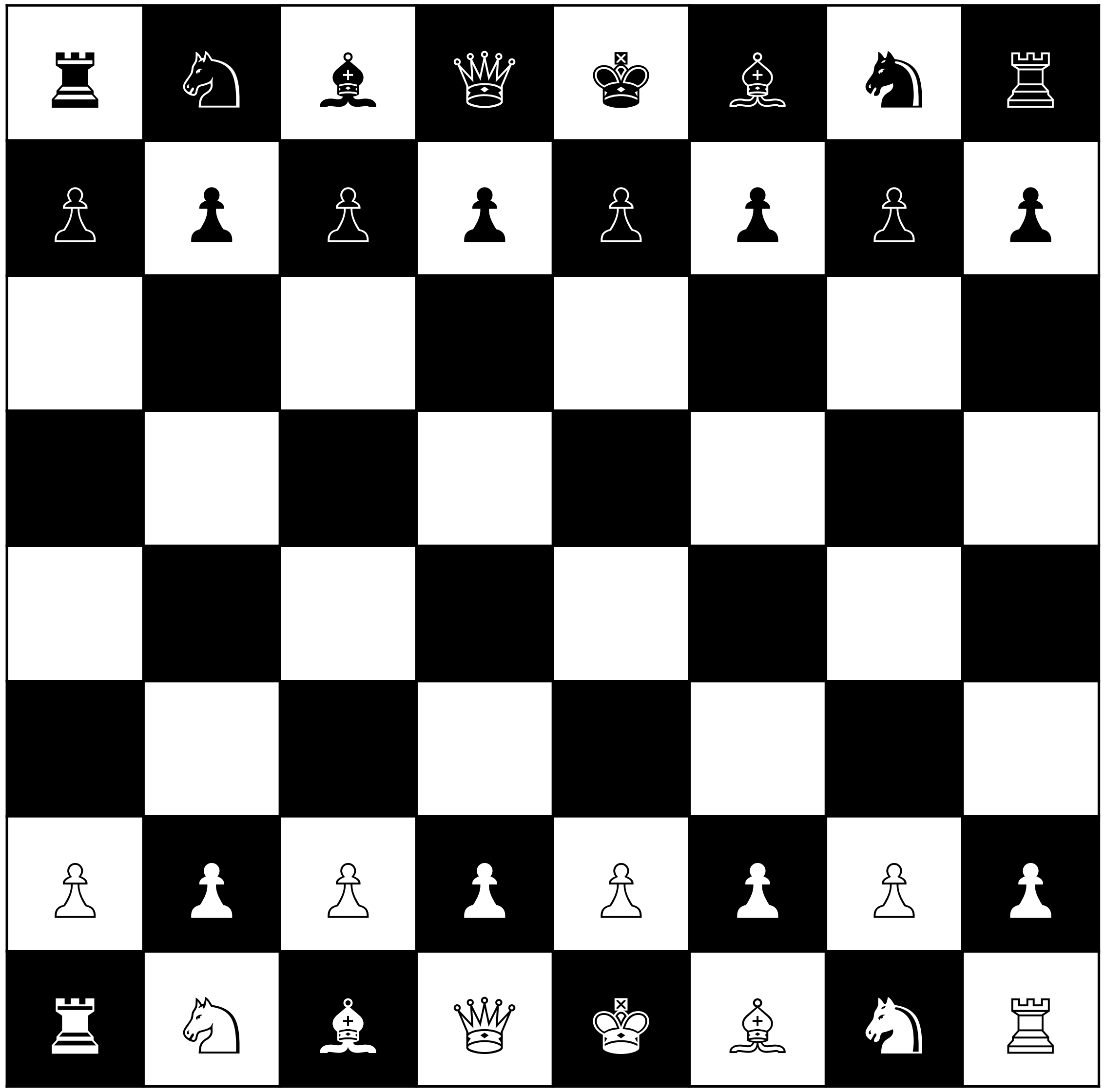
Search for Games

- Following this lecture you should be able to:
  - Understand the search process in games
  - How an AI decides on the best moves to make, given a search space
  - Implement various evaluation functions
  - How to prune a search tree to make sure irrelevant paths aren't followed

Playing Games with AI

- Games such as chess offer pure, abstract competition, which makes them popular for AI research
- The states of a game such as chess are easy to represent
- Actions on the states are restricted to a small number of fairly well-defined actions
- Game playing is an idealisation of worlds in which hostile agents act to diminish one's well being, within this world

State Representation: Chess



Shannon(1950), Programming a Computer for Playing Chess, Philosophical Magazine 41(314)

Why is chess important to AI?

- A chess-playing machine would be an existence of the proof of a machine doing something thought to require intelligence

- Simplicity of rules combined with the programmable states of the world means that this can be represented as a search problem through a space of <u>all possible game positions</u>

- The representation of the game can be correct in every possible way, unlike other similar problems e.g fighting a war

Why is chess important to AI?

- The introduction of an opponent brings uncertainty to the search space
  - Opponent will attempt to make the best move possible
- Search for games is hard to solve
- Chess has an average branching factor of 35
- Games go on to roughly 50 moves per player
  - Search tree has around 35^{100} nodes
- The uncertainty in this is choosing the best move given all combinations
  - We can't search through all possible solutions in a reasonable time
- We must guess based upon past experiences

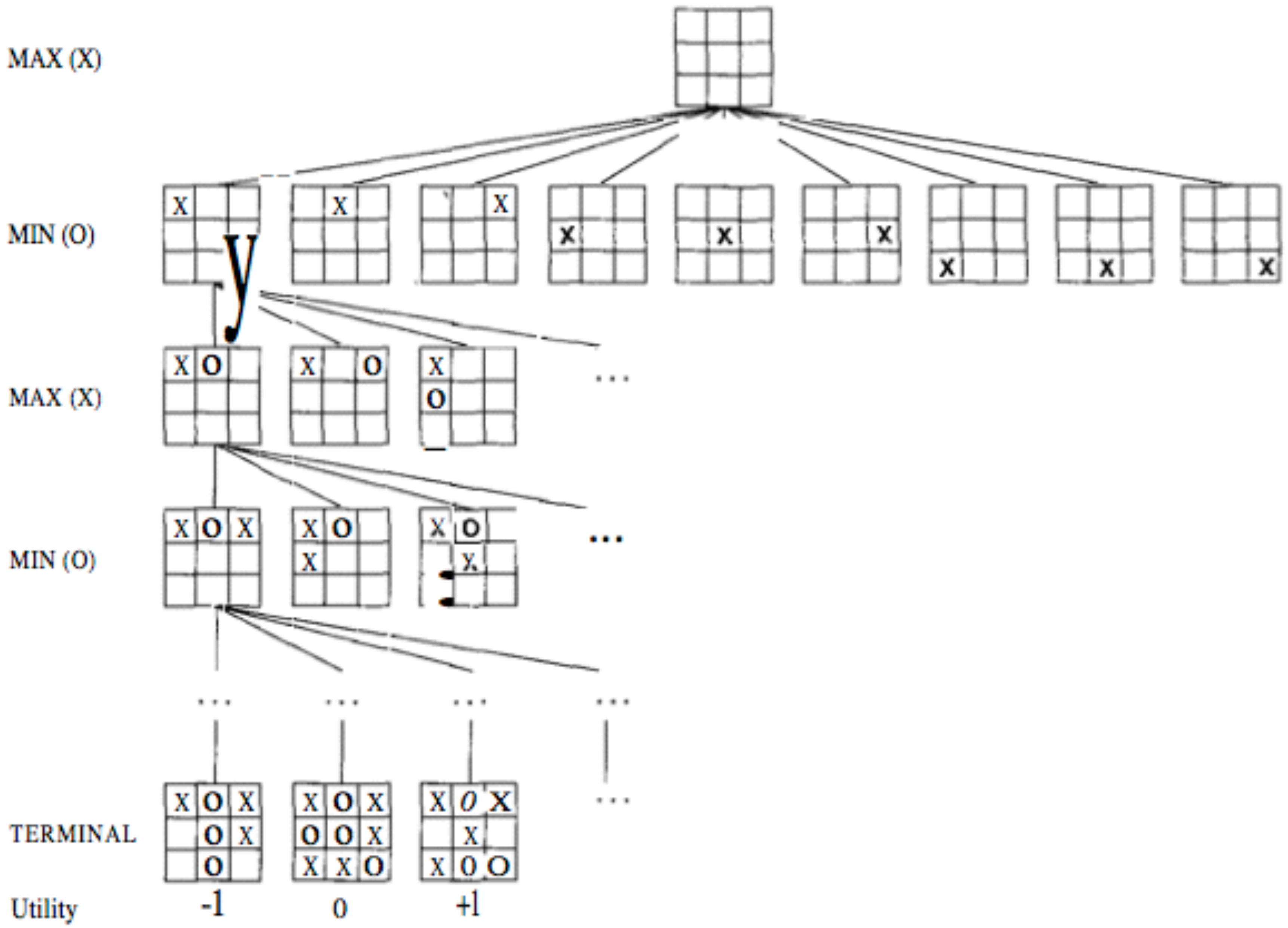
Decision Making in Two-Player Games

- Consider a two player game, with players named MAX & MIN
- MAX moves first, and take turns moving until the game is over
- At the end of the game, point are awarded to the winning player
- We can define a game as a search problem as follows:
  - Initial state: Board position and whose move it is
  - Set of actions: Legal moves a player can make
  - Terminal test: Check if the game is over
  - Set of utility functions: Numeric value of the outcome of the game: +1, 0, -1

Strategy

- In normal search, MAX would search for moves that lead to a winning state
- MIN however will interfere with this
- MAX requires a strategy to win, regardless of MIN's actions
- MAX will attempt to make the correct move corresponding to the actions of MIN

Noughts and Crosses

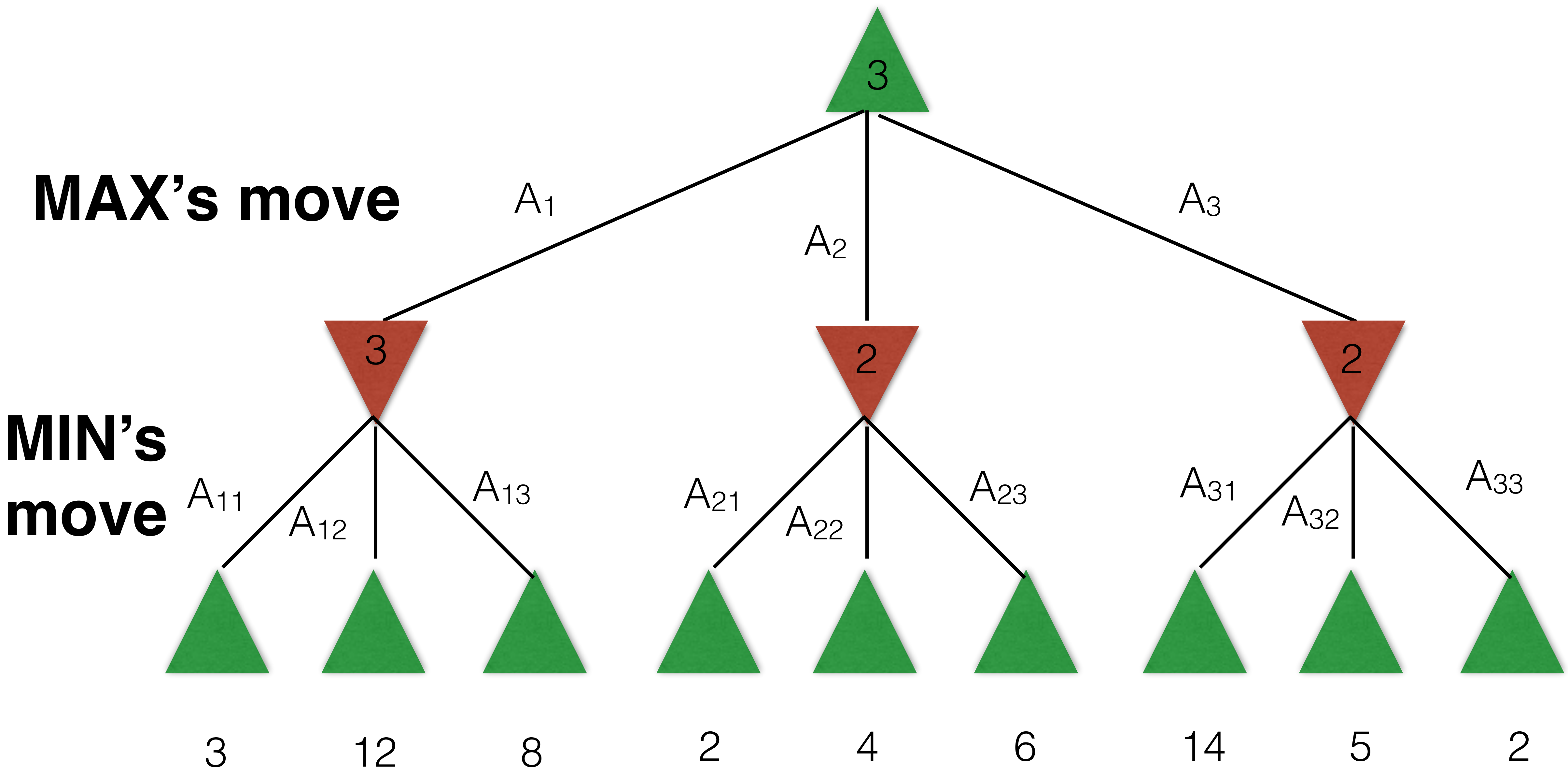


Minimax Algorithm

- Five steps:

  1. Generate the whole game tree, down to terminal states
  2. Apply the utility function to each terminal state to get its value
  3. Use this utility value of each terminal to determine the utility value of its parent nodes
  4. Continue backing up the values from the leaf nodes towards the root, one layer at a time
  5. Eventually we will reach the root of the tree. Now MAX must choose the root that maximises the utility value. This is called the minimax decision.

Trivial Game Tree



Minimax Algorithm: Pseudocode

```
MiniMax-Decision( game ):
    for each action in actions( game ):
        state = apply( action, game )
        value[ action ] = MiniMax-Value( state, game)
    return action with the highest value[ action ]

MiniMax-Value( state, game ):
    if state == gameOver: // is a leaf node
        return utility-value( state )

    else if state == maxNode: // MAX's move
        return highest MiniMax-Value of successors( state, game )

    else: // MIN's move
        return lowest MiniMax-Value of successors( state, game )
```

Minimax Algorithm: Properties

- If the maximum depth of the tree is *m*, and there are *b* legal moves at each point, then the time complexity is $O(b^m)$
  - Impractical for real games
- Depth-first search algorithm
- Implementation uses <u>recursion</u> instead of a queue
- Space requirements are linear in *m* and *b*
- Algorithm is basis for more realistic methods
- Basis for mathematical analysis of games

Overcoming Impracticalities

- Minimax assumes that a program must search all the way to terminal states
- This isn't practical
- Shannon proposed that program should cut off the search earlier and apply a <u>heuristic evaluation function</u>
- Minimax is altered in two ways:
    1. Utility function replaced by evaluation function EVAL
    2. Terminal test is replaced by cutoff test CUTOFF-TEST

Evaluation Function

- Estimates expected utility of the game from a given position
  - Not new, take chess for example with its material values of pieces
- Quality of game-playing program dependent upon evaluation function
  - How do we measure quality?

Measuring Quality of Evaluation Function

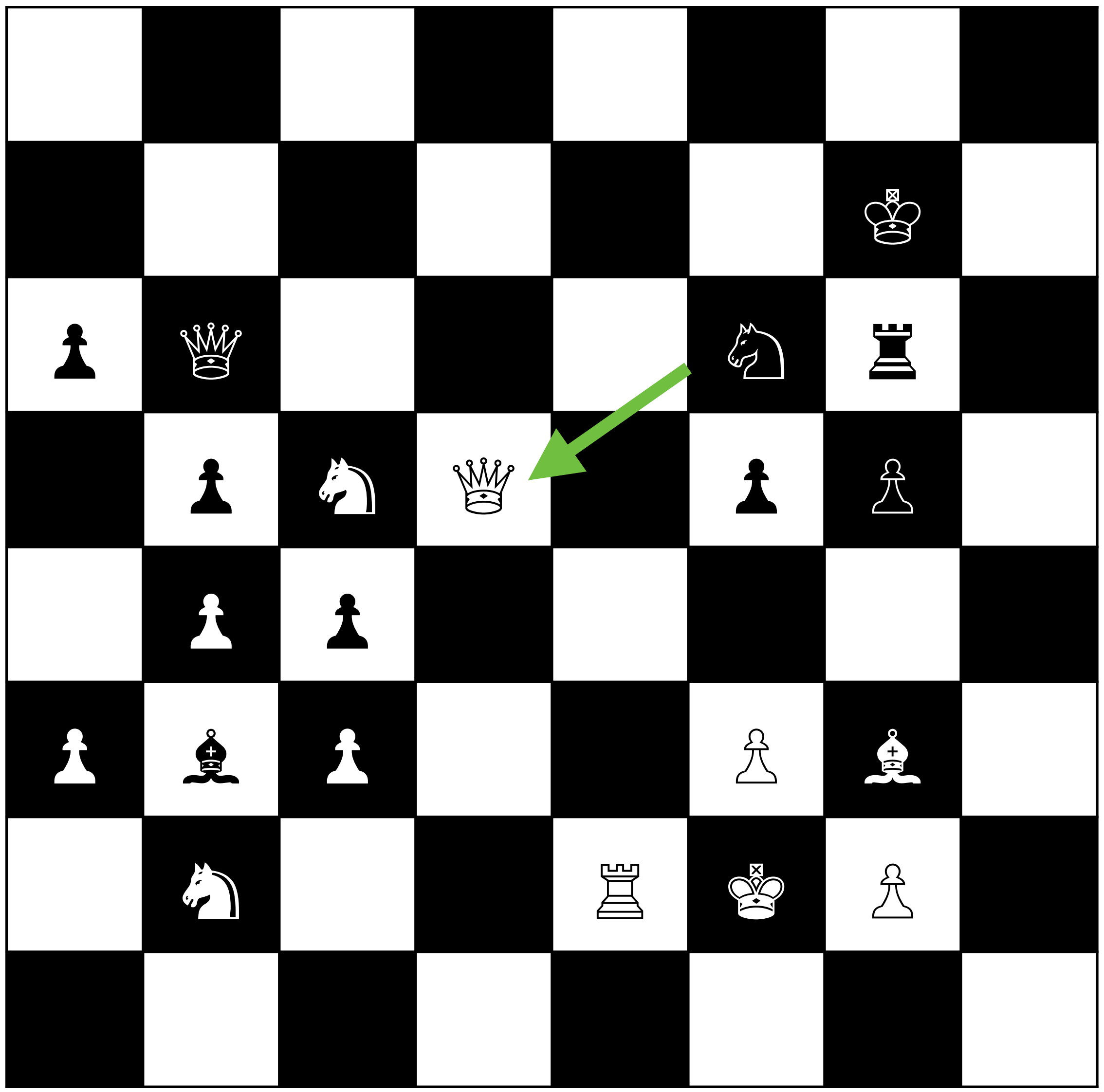
1. EVAL must agree with utility on terminal states
2. EVAL must not take too long
3. EVAL must reflect chances of winning

Cutting Off Search

- Set a depth limit - depth-limited search
- Depth is chose to ensure game does not take to long
- Iterative deepening is a more robust approach
- When time runs out, program returns move selected by the deepest completed search

Negative Consequence of Cutting Off Search

- Consider material advantage of chess:



Suppose search reaches limit at following position

EVAL(White) = 28
EVAL(Black) = 25

- White ahead by a knight
- Black to move
- Black captures white queen
- No loss in return
- In reality, black gains advantage
- Evaluation function does not reflect this

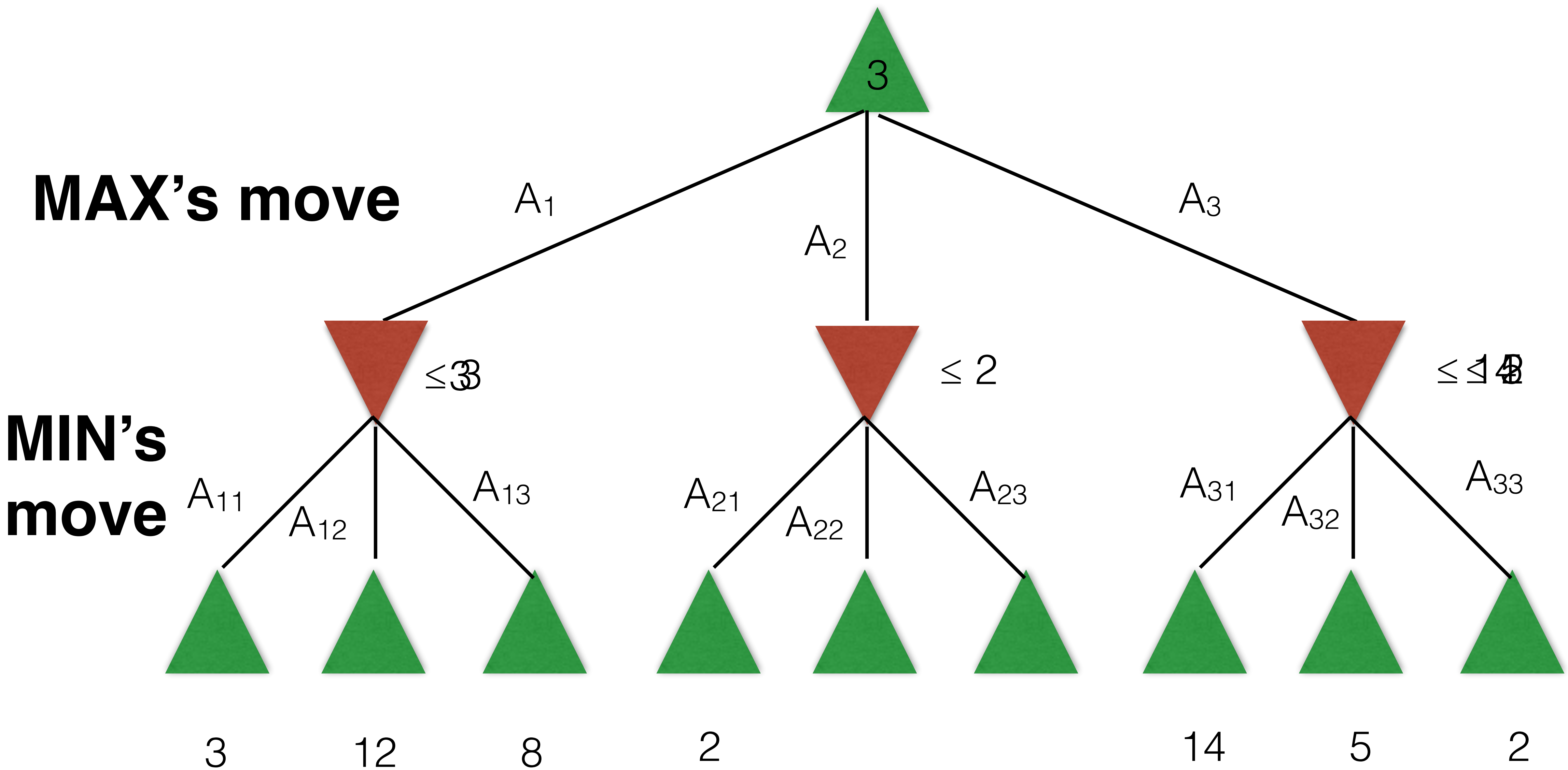
Pruning the Search Tree

- Take our chess example:
  - One can search ~1000 moves per second
  - If we take 150 seconds per move we can look at 150,000 positions
  - Chess has a branching factor of ~35
    - We will only be able to look ahead 3 or 4 ply
    - Average humans plan ahead 6 to 8 ply

Alpha-beta Pruning

- Returns the same move as minimax,
  - Prunes branches that cannot possibly influence the final decision

Alpha-beta Pruning



Alpha-beta Pruning: General Principle

- Consider a node *n* in the tree
- If a player has a better choice *m* either at the parent node of *n* or at any point further up, then *n* will never be reached in actual play
- So once we know enough about *n* to reach this conclusion, we prune it.

Summary

- Search process in games
- Evaluation function for moves in a game
- Pruning branches of a search tree