CS 4100/5100: Foundations of AI

Adversarial Search (i.e. game playing)

Instructor: Rob Platt
r.platt@neu.edu

College of Computer and information Science
Northeastern University

September 5, 2013

Games

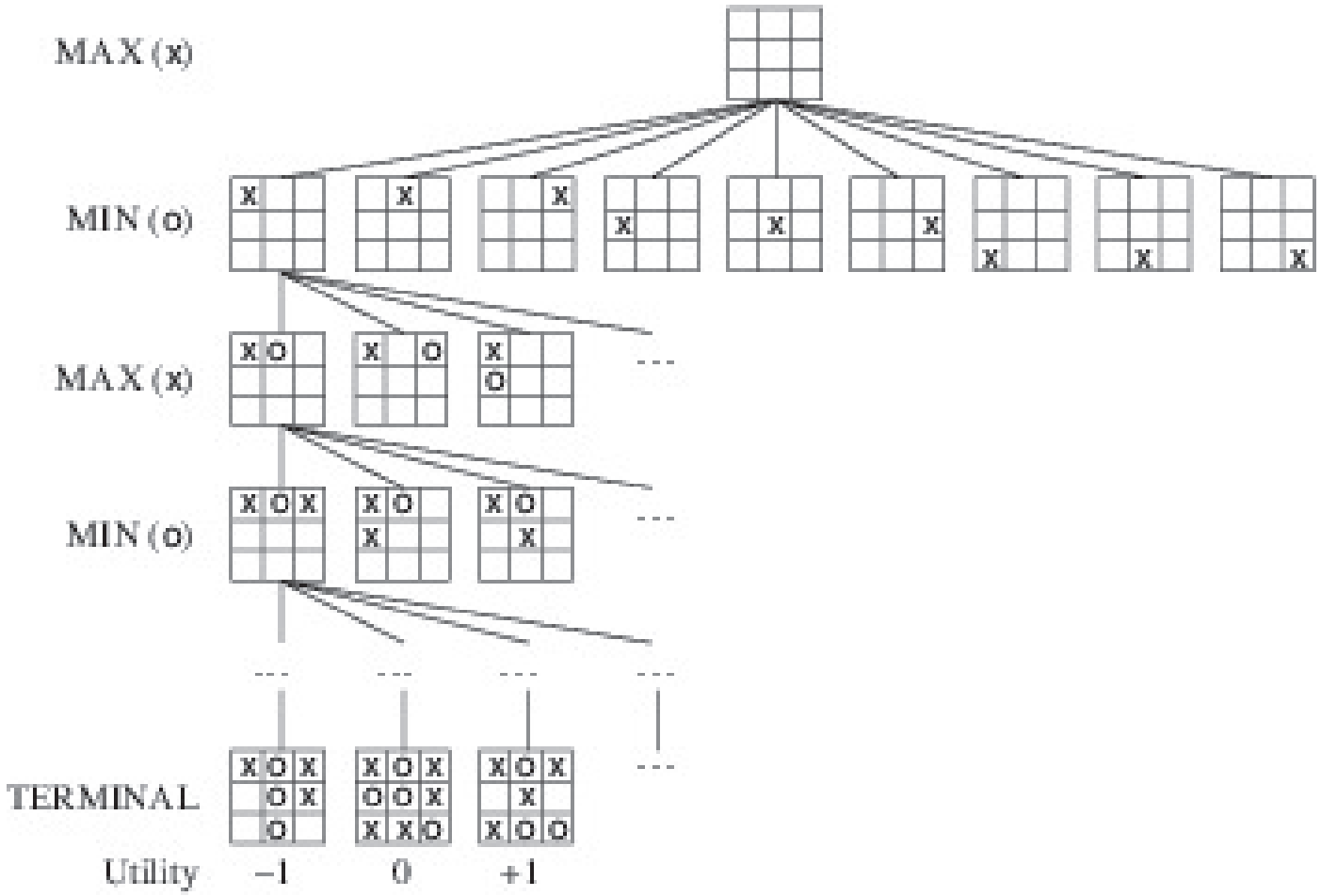
Kinds of games:

- stochastic vs perfect information
- fully observable vs partially observable
- most of the games we consider are zero-sum

Formally, a game is defined by:

- initial state
- transition function and description of feasible actions
- terminal test and utility function
- description of how players take turns
  - this is the only thing specific to games

Game tree

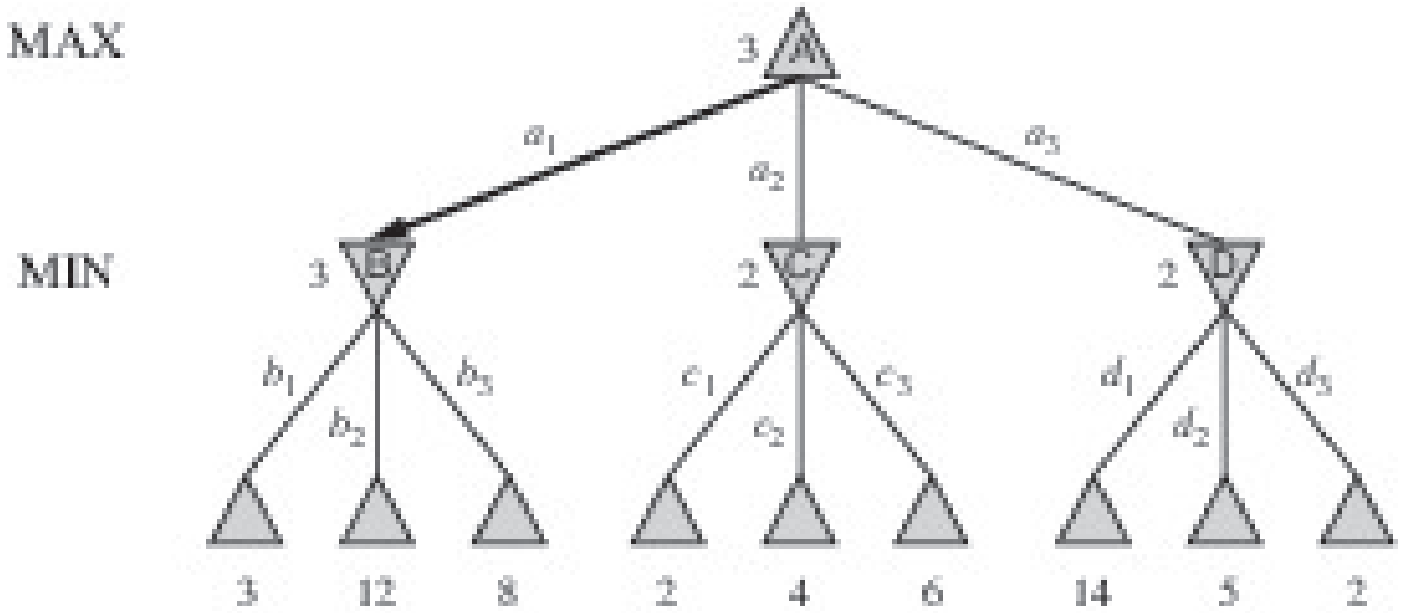


Game tree

The only difference between game search and regular search is that you need to take into account the actions of an opponent. How to do that?

- *assume in advance that the opponent always makes the best possible move available to him/her.*
- Both players are assumed to play optimally. An optimal strategy is as good as you can do assuming you are playing an infallible opponent.
- This is basically the same as AND-OR search. Why?

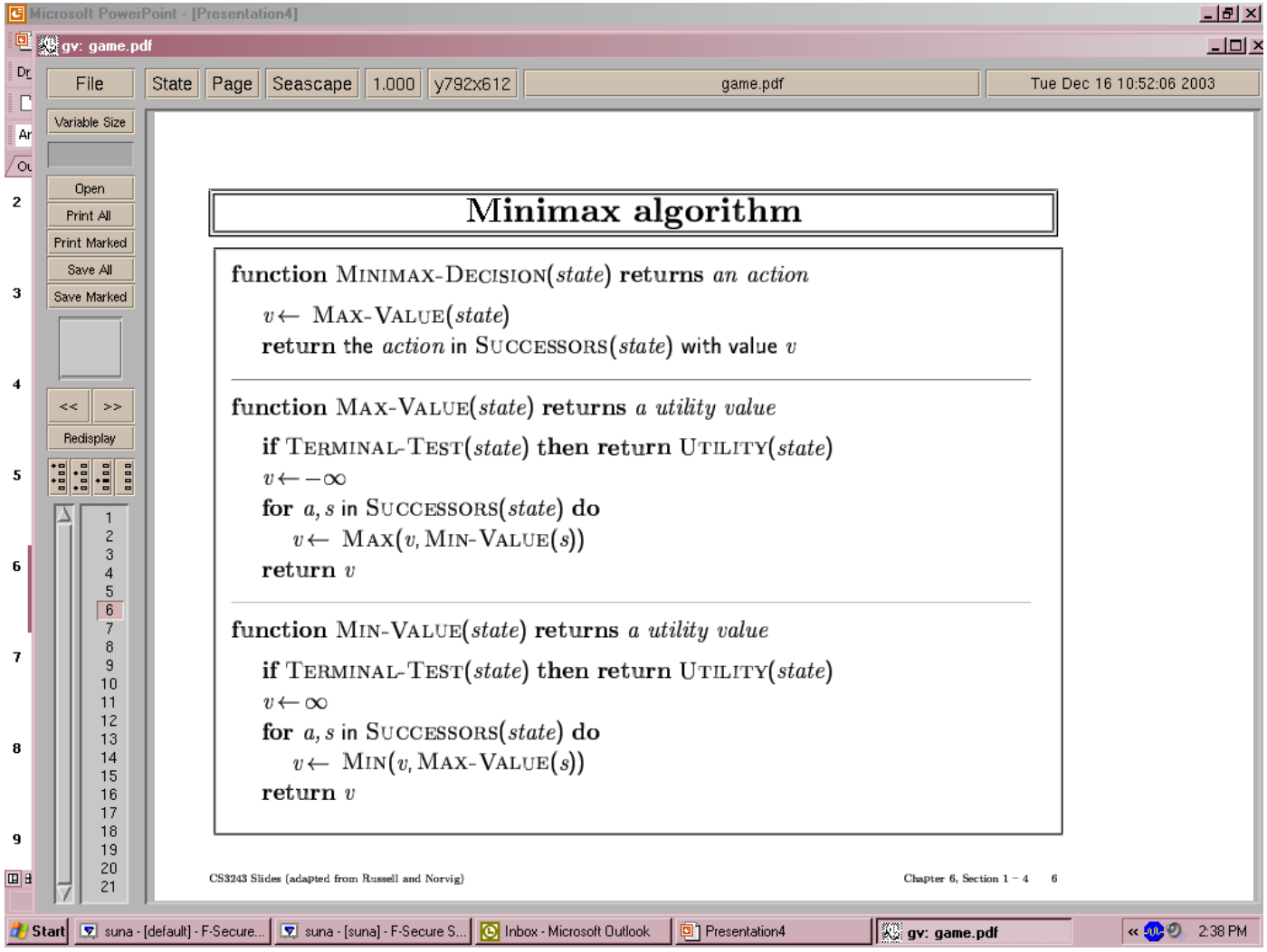
Minimax search



Minimax search is and-or search for game trees.

- recursively explore the tree (DFS)

Minimax search



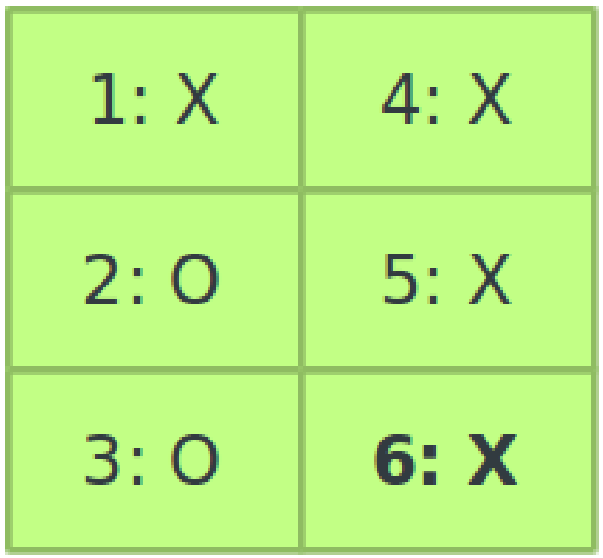
Minimax search

Properties of minimax

- optimal
- complete
- time complexity?
- space complexity?

Chess: $b \approx 35$, $m \approx 100$.

Minimax example



There are six spaces on a board that must be filled in order. Player 1 (MAX) uses x, and Player 2 (MIN) uses o. Each player can place at most 3 symbols per turn. Whoever places their symbol in the last slot on the board wins. Draw the minimax search tree.

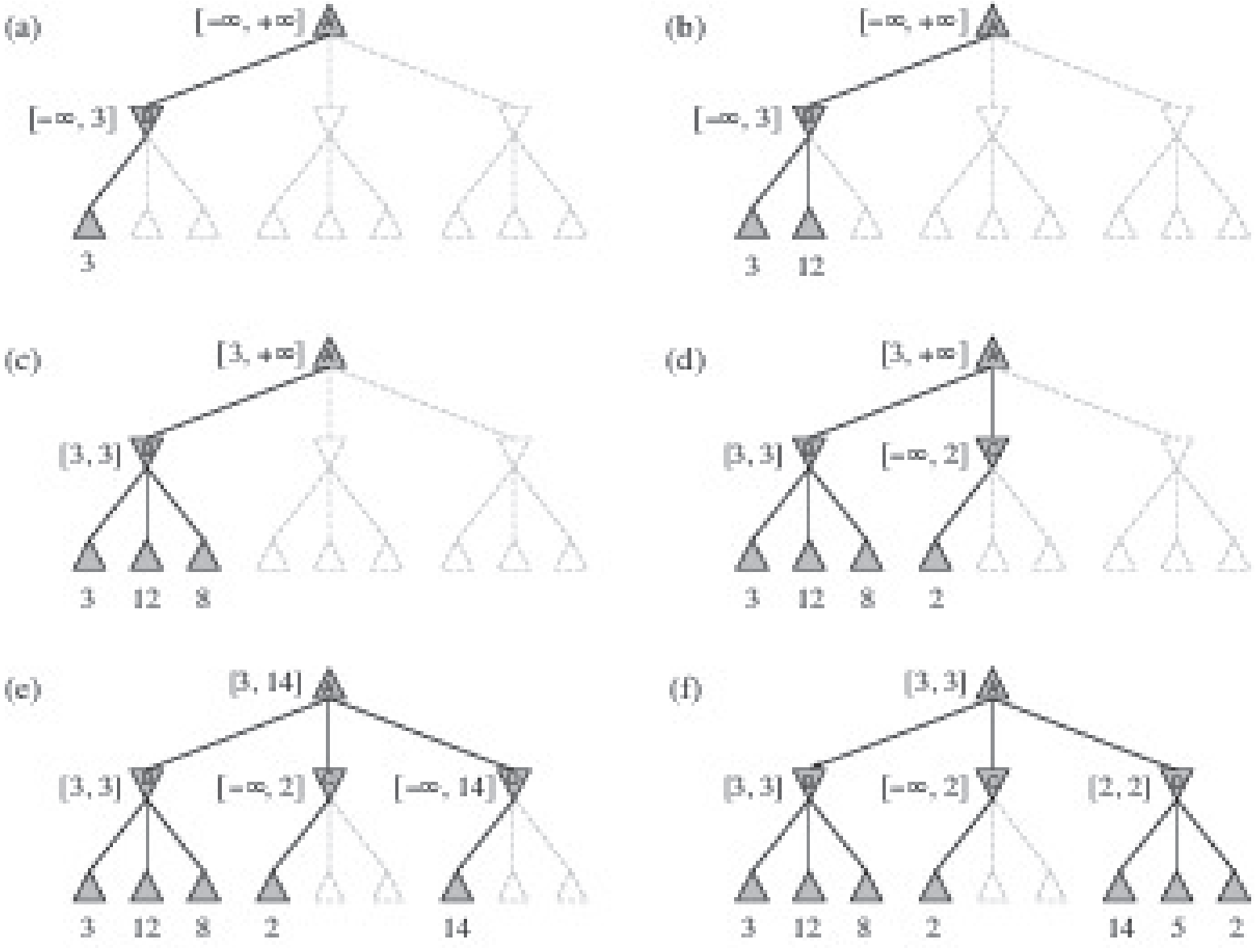
Alpha-beta search

Idea: prune parts of the tree that you *know* won't change your strategy.

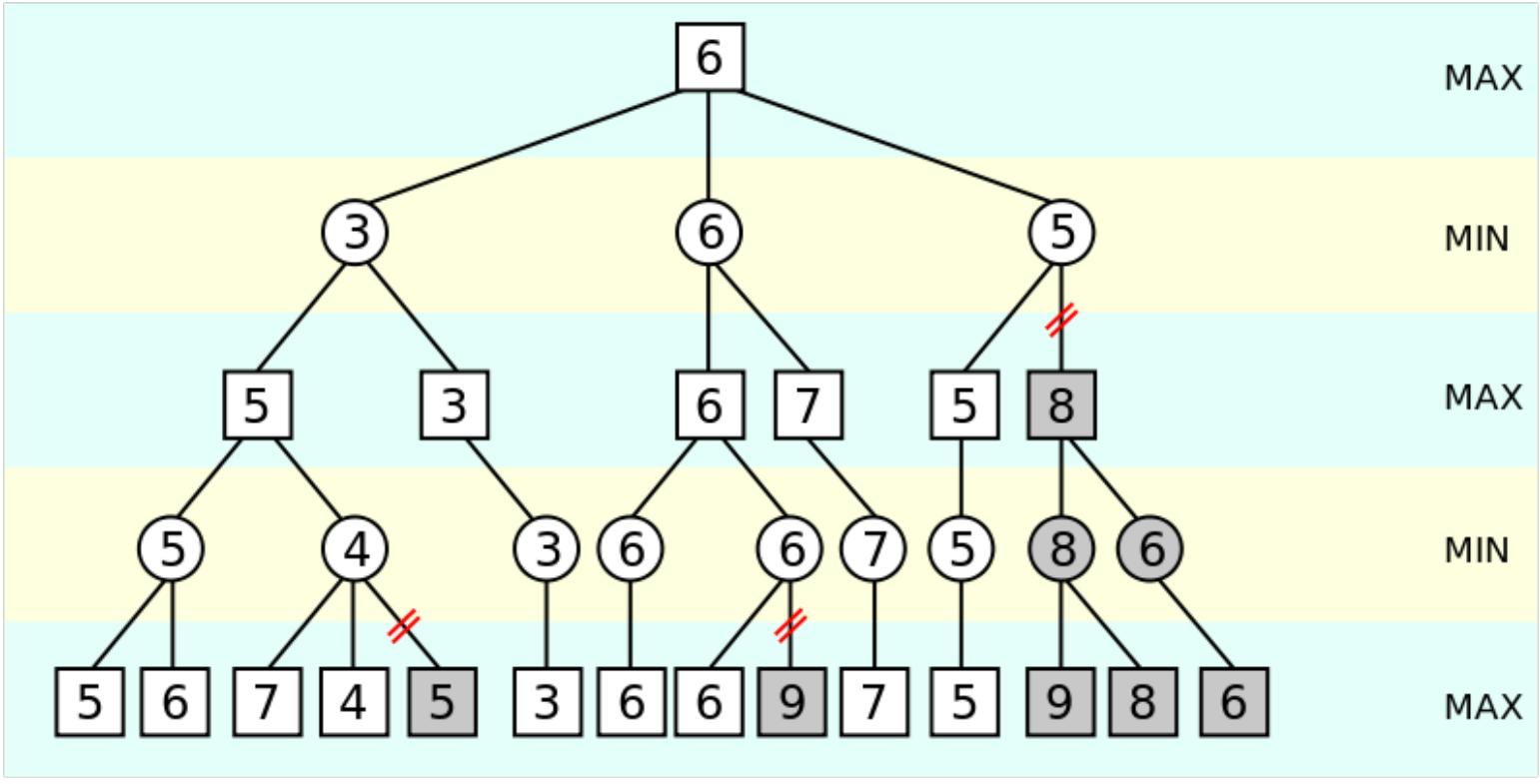
A player is in the process of unrolling the search tree in order to decide what to do...

- suppose that the best move that the player has found so far has value v
- suppose that you already know that the node you are currently expanding has a *maximum* value of $w < v$ (how can this happen?)
- then, you don't need to explore that node any more and can move on to the next one...

Alpha-beta search



Alpha-beta search



Move ordering in Alpha-beta search

Notice that the effectiveness of AB depends on the order in which states are examined.

- minimax search complexity: b^m
- AB search w/ random move ordering: $b^{0,75*m}$
- AB search w/ "optimal" move ordering: $b^{0.5m}$ (killer move heuristic)

But, how do you get a good move ordering?

- a heuristic
- iterative deepening...

Avoiding repeated search:

- *tranposition table*: hash table that stores the values for previously explored states.

Doing better

Usually, AB isn't enough. AB is complete and sometimes that just isn't possible.

- alternative: cut off search early w/ an approximated value for that state.

Evaluation function: an estimate of the expected utility of a given state.

- for example, in Chess: assign each board position an approximate value in terms of the number of pawns, bishops, knights, *etc.*
- cut off search after a certain depth and just use the evaluation function

Doing better (specifically)

But how do you decide how to set the evaluation function?

Think of it as dividing up the state space into equivalence classes:

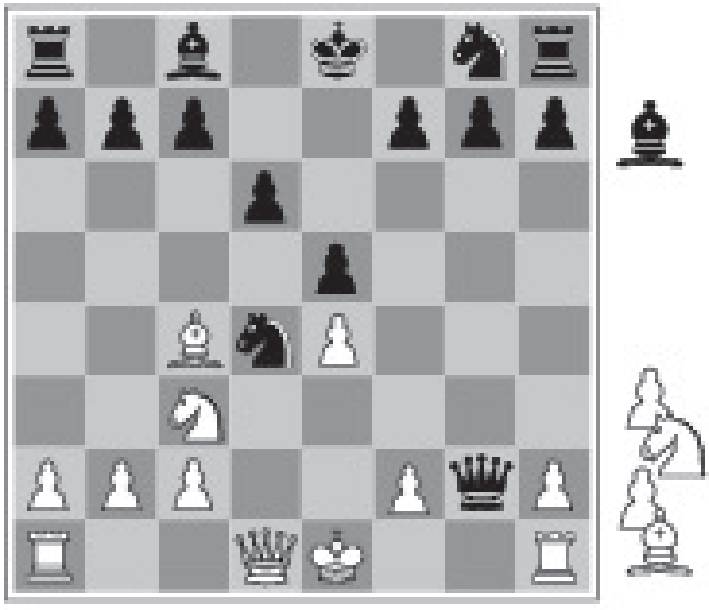
- all board states w/ 5 pawns, 2 bishops, etc. belong in a single equiv class
- the value of each equiv class is equal to the *probability* of winning from that state (i.e. the expected value in general).
- but, how do you calculate that? learn from experience?

Cutting off search

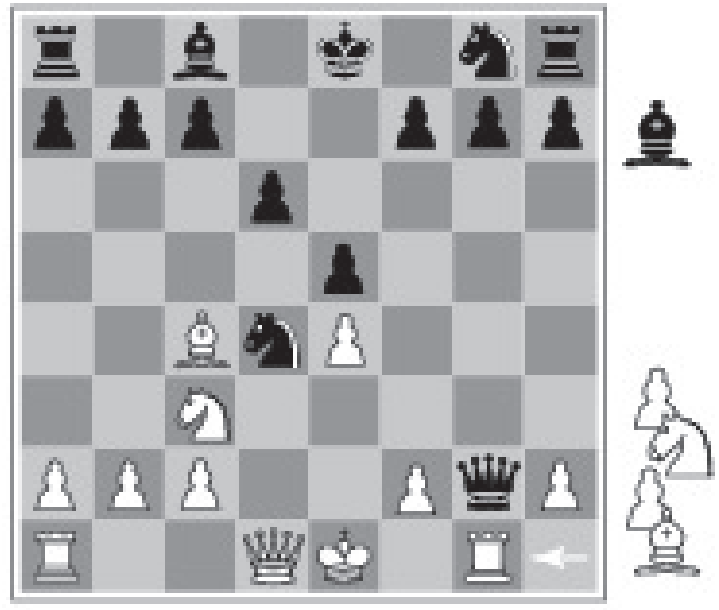
A simple way to cut off search is to use iterative deepening and just return the best current answer when the time's up.

- ID also helps w/ move ordering

Cutting off search



(a) White to move



(b) White to move

Don't want to cut off search pre-maturely. Sometimes a given state is likely to suffer a large change in value in the near future...

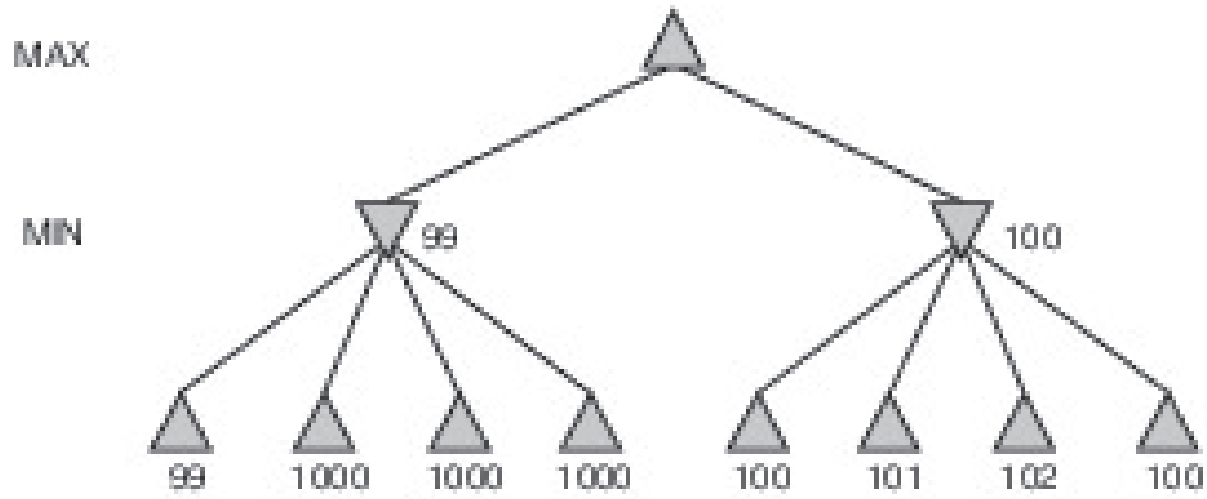
- only apply cutoff to states that are *quiescent*

Forward pruning

Look at evaluation function *before* expanding nodes. Just don't expand low-scoring nodes:

- for example: on each move, only do search on the top n moves.
- alternative (probcut): estimate how likely a given node is to be outside the (α, β) threshold based on past experience and without searching the entire tree.

Minimax variations



Sometimes, strict minimax seems to be the wrong thing to do:

- alternative is to associate the evaluation function w/ some measure of variance. Then, calculate the probability that the node is actually better.