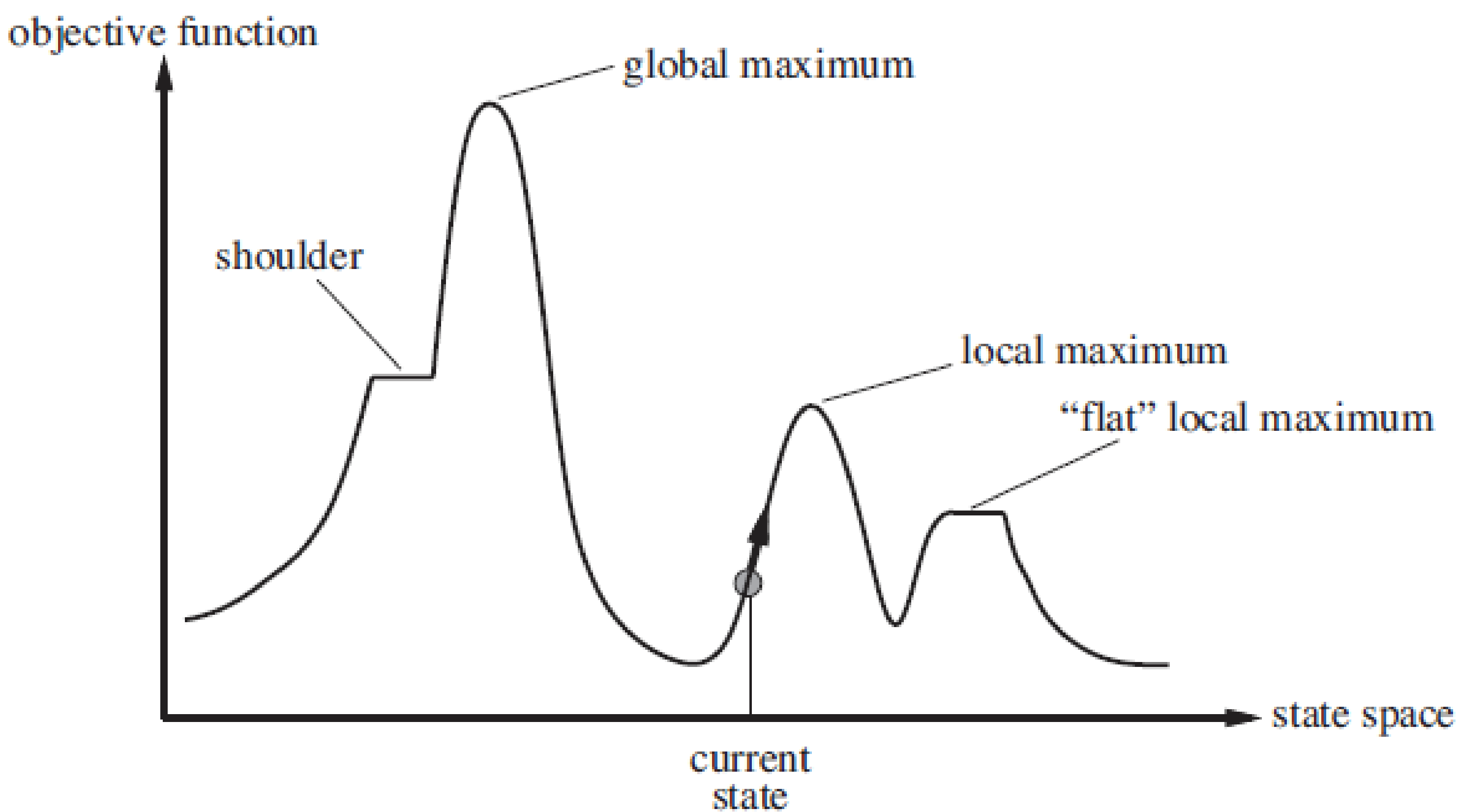
Artificial Intelligence

CS482, CS682, MW 1 – 2:15, SEM 201, MS 227

Prerequisites: 302, 365

Instructor: Sushil Louis, sushil@cse.unr.edu, http://www.cse.unr.edu/~sushil

Non-classical search



- Path does not matter, just the final state
- Maximize objective function

How does it work

String decoded f(x^2) fi/Sum(fi) Expected Actual

01101	13	169	0.14	0.58	1	
11000	24	576	0.49	1.97	2	
01000	8	64	0.06	0.22	0	
10011	19	361	0.31	1.23	1	
Sum		1170	1.0	4.00	4.00	
Avg		293	.25	1.00	1.00	
Max		576	.49	1.97	2.00	

How does it work cont'd

String mate offspring decoded f(x^2)

0110|1	2	01100	12	144		
1100|0	1	11001	25	625		
11|000	4	11011	27	729		
10|011	3	10000	16	256		
Sum				1754		
Avg				439		
Max				729		

GA Theory

- Why fitness proportional selection?
  - Fitness proportional selection optimizes the tradeoff between exploration and exploitation. Minimizes the expected loss from choosing unwisely among competing schema
- Why binary representations?
  - Binary representations maximize the ratio of the number of schemas to number of strings
- Excuse me, but what is a **schema**?
- Mutation can be thought of as beam hill-climbing. Why have crossover?
  - Crossover allows information exchange that can lead to better performance in some spaces

Schemas and Schema Theorem

- How do we analyze GAs?
  - Individuals do not survive
  - Bits and pieces of individuals survive
- Three questions:
  - What do these bits and pieces signify?
  - How do we describe bits and pieces?
  - What happens to these bits and pieces over time?

Schemas

- What does part of a string that encodes a candidate solution signify?

1	1	1	0	0	0

A point in the search space

1	1	1			

An area of the search space

Different kind of crossover lead to different kinds of areas that need to be described

1			0	1	

A different kind of area

1	*	*	0	1	*

A schema denotes a portion of the search space

Schema notation

- Schema H = 01*0* denotes the set of strings:
  - 01000
  - 01001
  - 01100
  - 01101

Schema properties

- Order of a schema H → O(H)
  - Number of fixed positions
  - O(10**0) = 3
- Defining length of a schema
  - Distance between first and last fixed position
  - d(10**0) = 4
  - d(*1*00) = 3

What does GA do to schemas?

- What does selection do to schemas?
  - If m (h, t) is the number of schemas h at time t then
  - m(h, t+1) = $\frac{f_i}{\bar{f}}$ m (h, t) → above average schemas increase exponentially!
- What does crossover do to schemas?
  - Probability that schema gets disrupted
  - Probability of disruption = $P_c \frac{\partial(h)}{l-1}$
    - This is a conservative probability of disruption. Consider what happens when you crossover identical strings
- What does mutation do to schemas?
  - Probability that mutation does not destroy a schema
  - Probability of conservation = $(1 - P_m)^{o(h)}$ = (1 - $o(h)\ P_m$ - (higher order terms))

The Schema theorem

- Schema Theorem:
  - $M(h, t+1) \geq \frac{f_i}{\bar{f}} m(h, t) \left[1 - P_c \frac{\partial(h)}{l-1} - o(h) P_m\right]$ … ignoring higher order terms

- The schema theorem leads to the **building block hypothesis** that says:
  - *GAs work by juxtaposing, short (in defining length), low-order, above average fitness schema or building blocks into more complete solutions*

GA Theory

- Why fitness proportional selection?
- Why crossover?
- Why mutation?
- Why binary representations?

Continuous spaces

Suppose we want to site three airports in Romania:
- 6-D state space defined by (x_1, y_2), (x_2, y_2), (x_3, y_3)
- objective function $f(x_1, y_2, x_2, y_2, x_3, y_3) =$
  sum of squared distances from each city to nearest airport

Discretization methods turn continuous space into discrete space, e.g., empirical gradient considers $\pm\delta$ change in each coordinate

Gradient methods compute

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3}\right)$$

to increase/reduce f, e.g., by $\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$

- What is a good value for α ?
  - Too small, it takes too long
  - Too large, may miss the optimum

Newton Raphson Method

Sometimes can solve for $\nabla f(\mathbf{x}) = 0$ exactly (e.g., with one city).
Newton–Raphson (1664, 1690) iterates $\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x})\nabla f(\mathbf{x})$
to solve $\nabla f(\mathbf{x}) = 0$, where $\mathbf{H}_{ij} = \partial^2 f / \partial x_i \partial x_j$

Linear and quadratic programming

- Constrained optimization
  - Optimize f(**x**) subject to
    - Linear convex constraints – polynomial time in number of variables
      - Linear programming – scales to thousands of variables
    - Convex non-linear constraints – special cases → polynomial time
      - In special cases non-linear convex optimization can scale to thousands of variables

Games and game trees

- Multi-agent systems + competitive environment → games and adversarial search
- In game theory any multiagent environment is a game as long as each agent has "significant" impact on others
- In AI many games were
  - Game theoretically: Deterministic, Turn taking, Two-player, Zero-sum, Perfect information
  - AI: deterministic, fully observable environments in which two agents act alternately and utility values at the end are equal but opposite. One wins the other loses
- Chess, Checkers
- Not Poker, backgammon,

Game types

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

Starcraft? Counterstrike? Halo? WoW?

Search in Games

"Unpredictable" opponent ⇒ solution is a strategy
specifying a move for every possible opponent reply

Time limits ⇒ unlikely to find goal, must approximate

Plan of attack:

- Computer considers possible lines of play (Babbage, 1846)
- Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)
- Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948; Shannon, 1950)
- First chess program (Turing, 1951)
- Machine learning to improve evaluation accuracy (Samuel, 1952–57)
- Pruning to allow deeper search (McCarthy, 1956)

Tic-Tac-Toe

- Two player, deterministic, small tree
- Two players: Max versus Min
- Approximately: 9! tree nodes

Tic-Tac-Toe

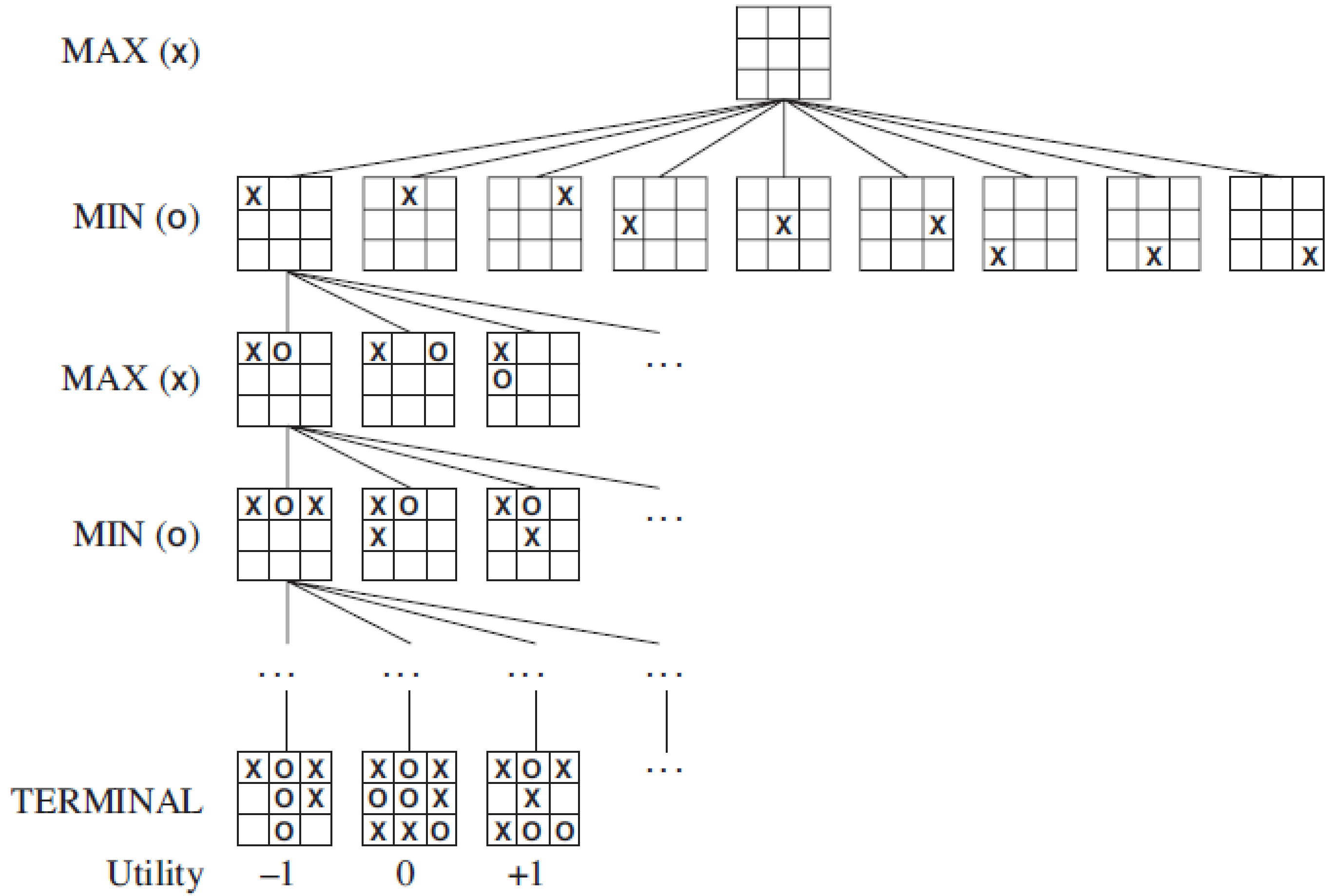


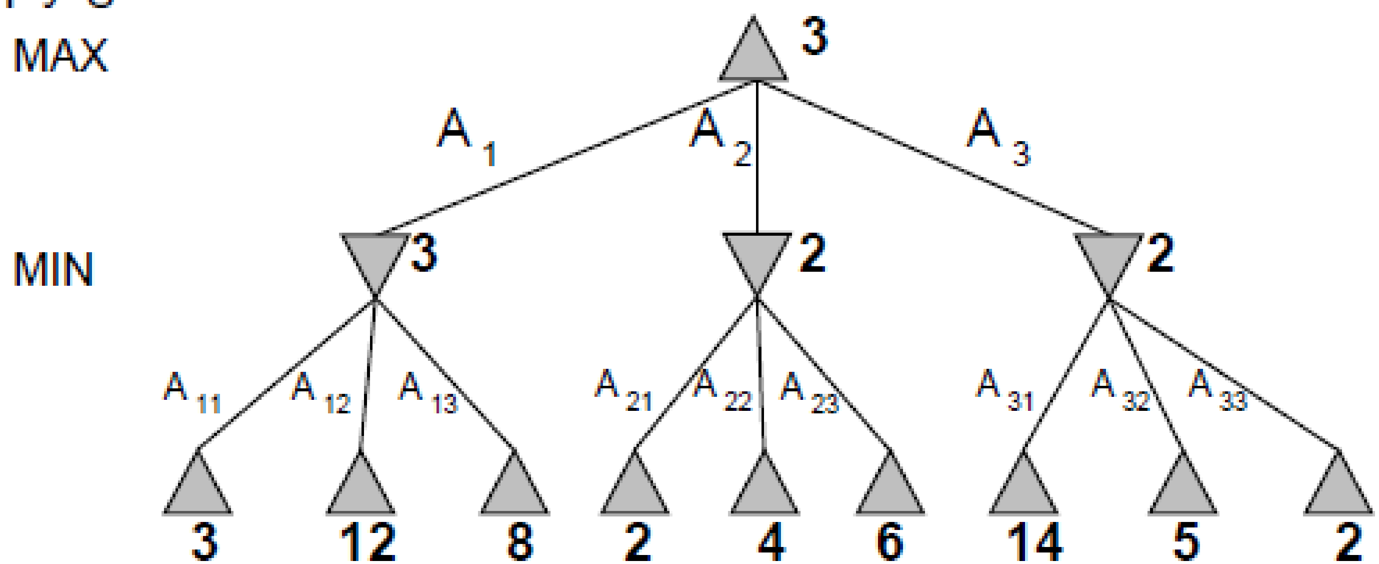
Figure 5.1 **FILES: figures/tictactoe.eps (Tue Nov 3 16:23:55 2009).** A (partial) game tree for the game of tic-tac-toe. The top node is the initial state, and MAX moves first, placing an X in an empty square. We show part of the tree, giving alternating moves by MIN (O) and MAX (X), until we eventually reach terminal states, which can be assigned utilities according to the rules of the game.

Minimax search

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:



Minimax algorithm

function MINIMAX-DECISION(*state*) **returns** *an action*
  inputs: *state*, current state in game

  return the a in ACTIONS(*state*) maximizing MIN-VALUE(RESULT(a, *state*))

---

function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS(*state*) **do** $v \leftarrow$ MAX(v, MIN-VALUE(s))
  return v

---

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for a, s in SUCCESSORS(*state*) **do** $v \leftarrow$ MIN(v, MAX-VALUE(s))
  return v

3 player Minimax

- Two player minimax reduces to one number because utilities are opposite – knowing one is enough
- But there should actually be a vector of two utilities with player choosing to maximize their utility at their turn
- So with three players → you have a 3 vector
- Alliances?

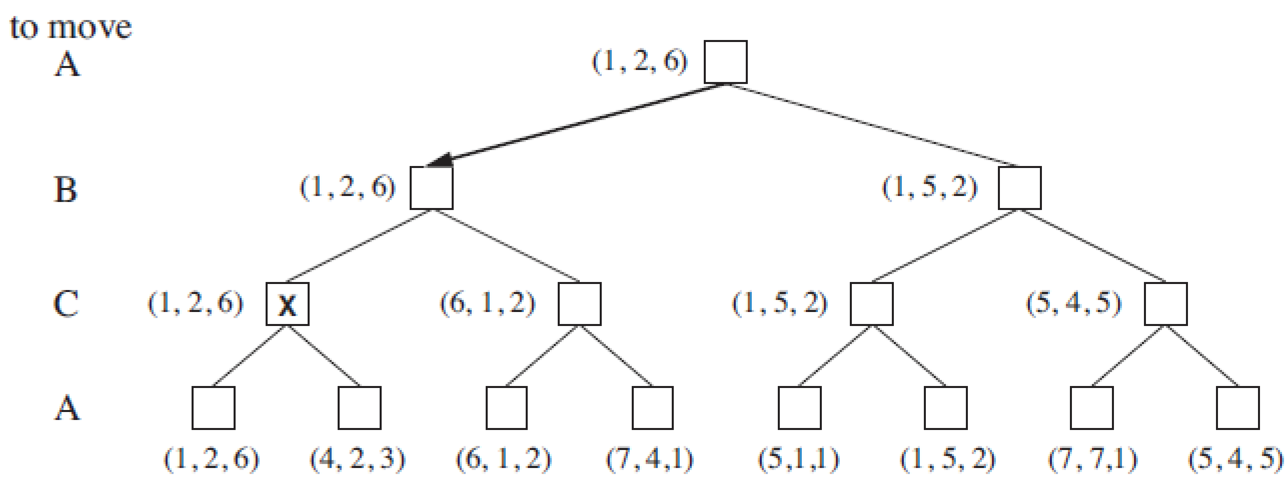
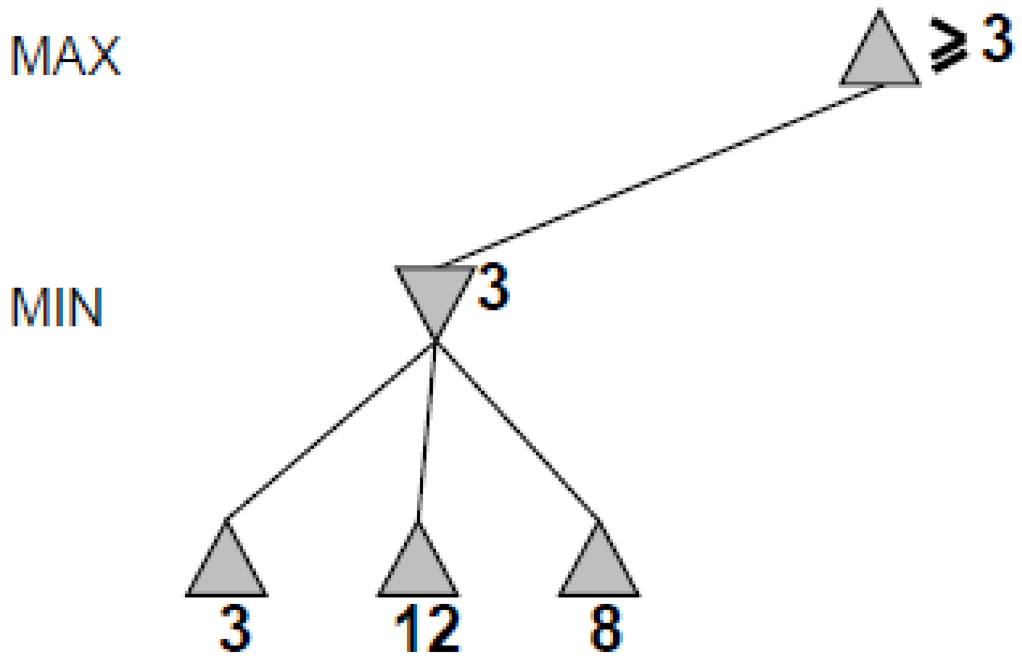


Figure 5.4 **FILES: figures/minimax3.eps (Tue Nov 3 16:23:11 2009).** The first three plies of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

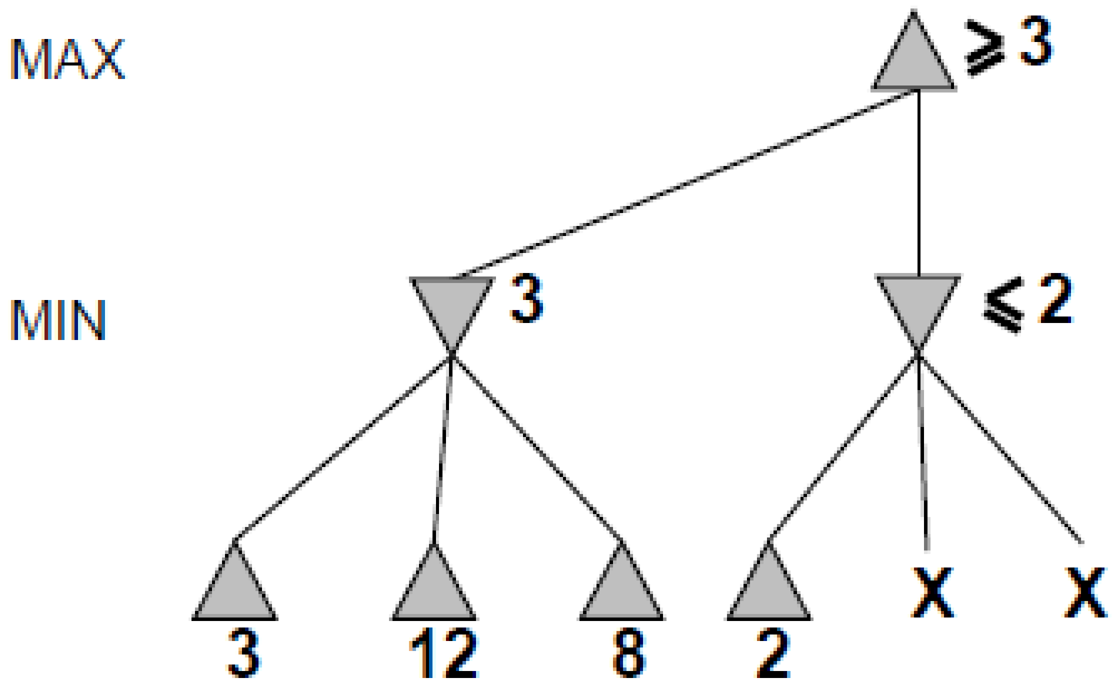
Minimax properties

- Complete?
  - Only if tree is finite
    - Note: A finite strategy can exist for an infinite tree!
- Optimal?
  - Yes, against an optimal opponent! Otherwise, hmmmm
- Time Complexity?
  - O(b^m)
- Space Complexity?
  - O(bm)
- Chess:
  - b ~= 35, m ~= 100 for reasonable games
  - Exact solution still completely infeasible

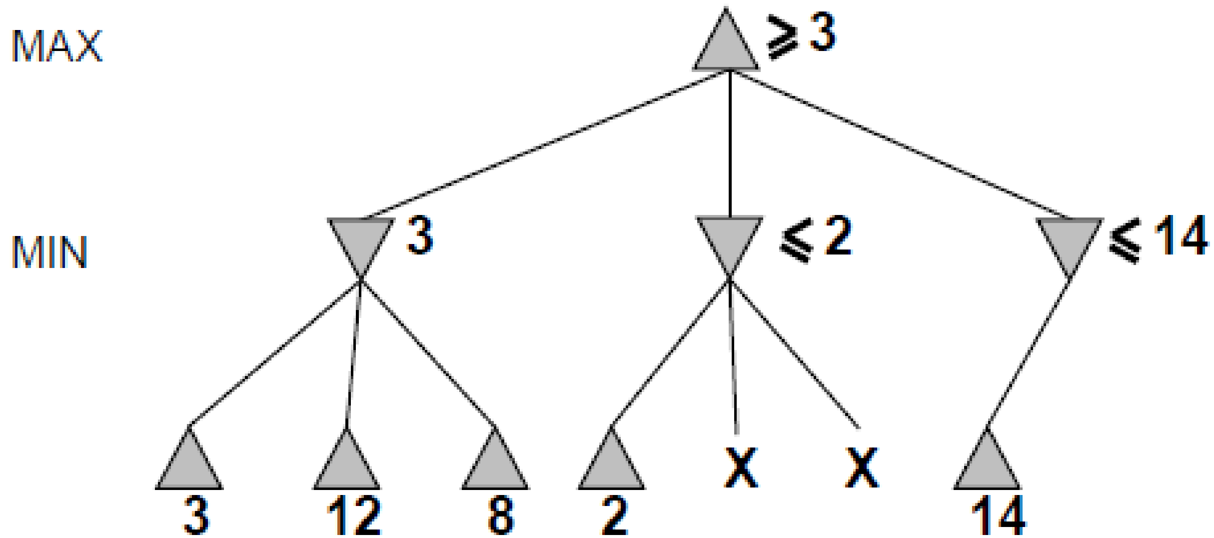
Alpha-beta pruning



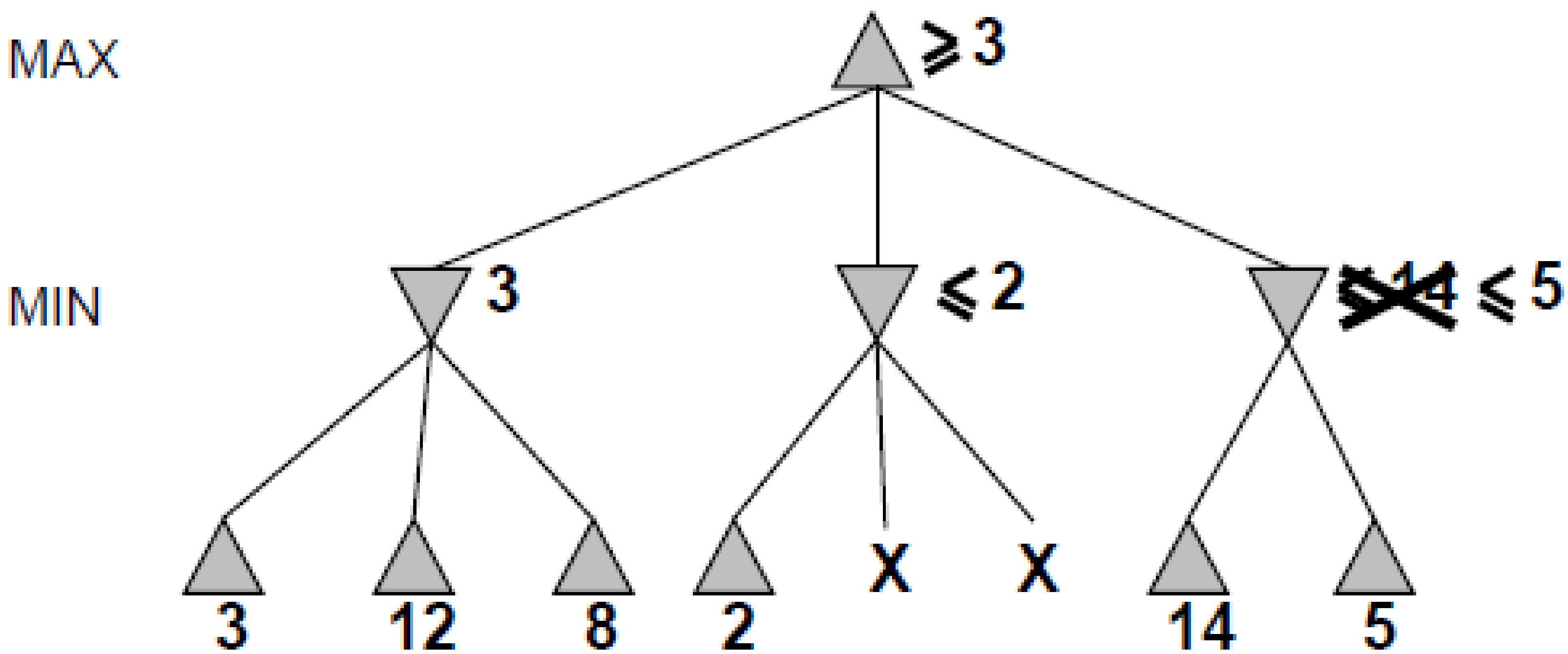
Alpha-beta



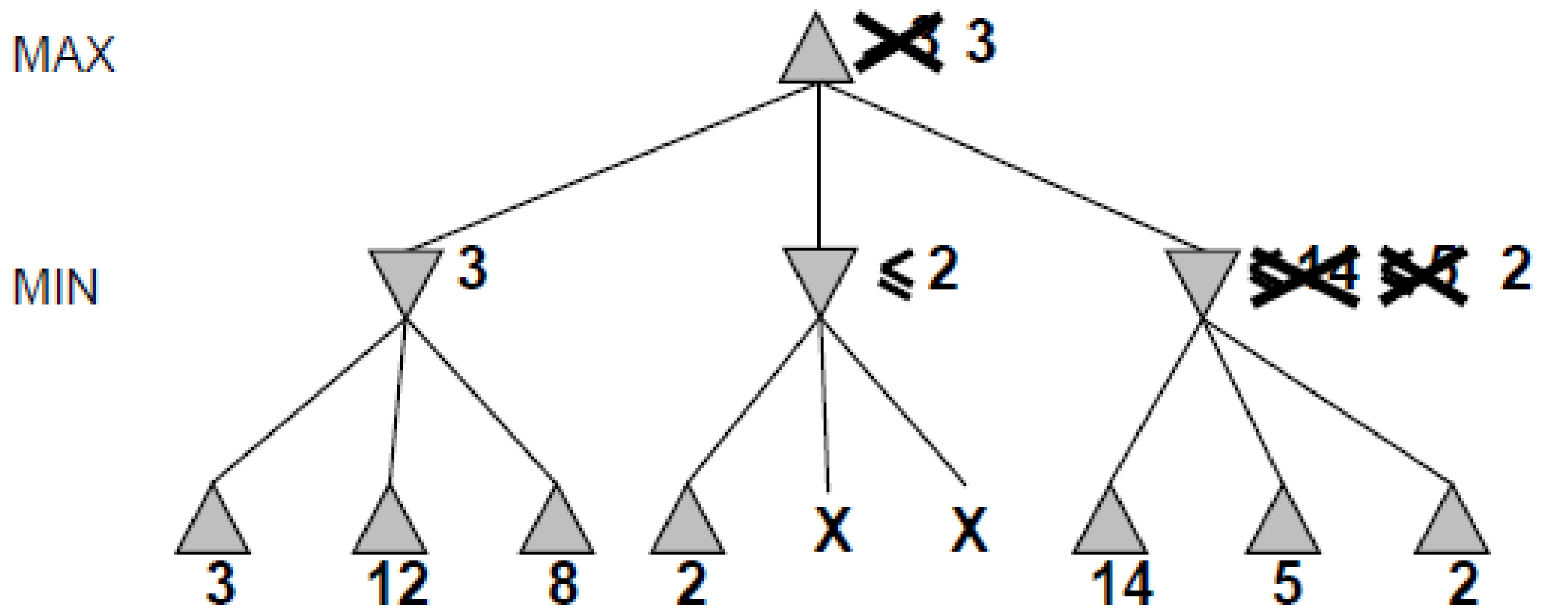
Alpha-beta



Alpha-beta



Alpha-beta



Alpha-beta

- Alpha is the best value (for Max) found so far at any choice point along the path for Max
  - Best means highest
  - If utility v is worse than alpha, max will avoid it
- Beta is the best value (for Min) found so far at any choice point along the path for Min
  - Best means lowest
  - If utility v is larger than beta, min will avoid it

Alpha-beta algorithm

function ALPHA-BETA-DECISION($state$) **returns** an action
  return the a in ACTIONS($state$) maximizing MIN-VALUE(RESULT($a, state$))

---

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  inputs: $state$, current state in game
    α, the value of the best alternative for MAX along the path to $state$
    β, the value of the best alternative for MIN along the path to $state$

  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

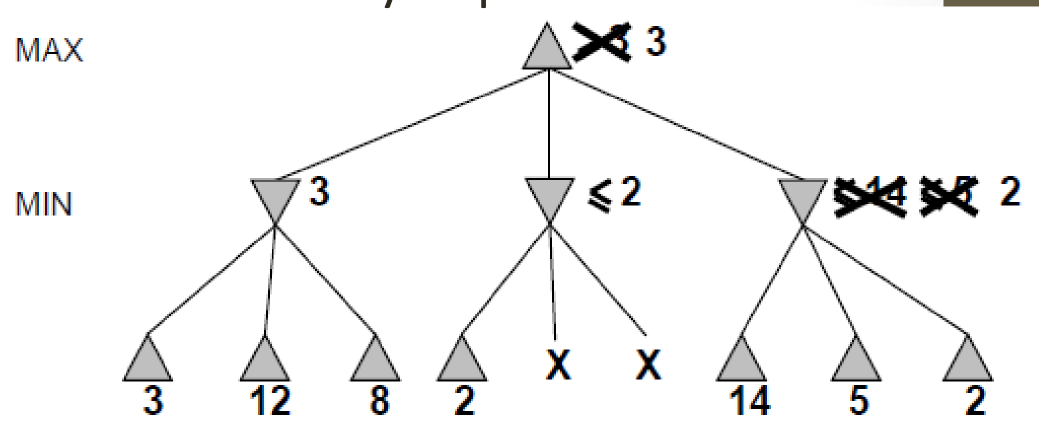
function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  same as MAX-VALUE but with roles of α, β reversed

Alpha beta example

- Minimax(root)
  - = max (min (3, 12, 8), min(2, x, y), min (14, 5, 2))
  - = max(3, min(2, x, y), 2)
  - = max(3, aValue <= 2, 2)
  - = 3

Alpha-beta pruning analysis

- Alpha-beta pruning can reduce the effective branching factor
- Alpha-beta pruning's effectiveness is heavily dependent on **MOVE ORDERING**
- **14, 5, 2 versus 2, 5, 14**
- **If** we can order moves well
- O($b^{\frac{m}{2}}$)
- Which is O($(b^{1/2}).^{m}$
- Effective branching factor then become square root of b
- For chess this is huge → from 35 to 6
- Alpha-beta can solve a tree twice as deep as minimax in the same amount of time!
    - Chess: Try captures first, then threats, then forward moves, then backward moves comes close to b = 12



Imperfect information

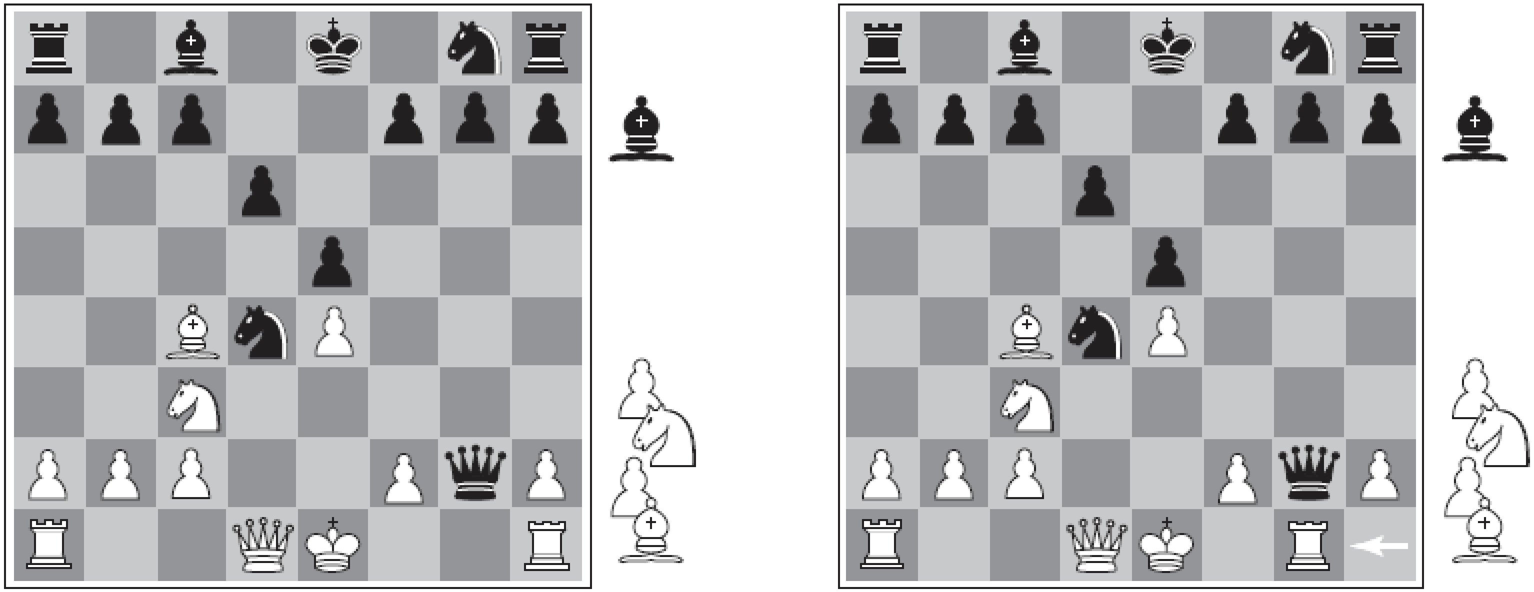
- You still cannot reach all leaves of the chess search tree!
- What can we do?
  - Go as deep as you can, then
  - Utility Value = Evaluate(Current Board)
  - Proposed in 1950 by Claude Shannon

- Apply an **evaluation function** to non-terminal nodes
- Use a **cutoff test** to decide when to stop expanding nodes and apply the evaluation function

Evaluation function

- Must order nodes in the same way as the utility function
  - Wins > Draws > Losses
- Fast
  - Otherwise it is better to search deeper and get more information
- For non-terminal states, high evaluations should mean higher probability of winning
  - Chess is not a chancy game
  - But computational limitations make eval function chancy!

Which is better?



(a) White to move

(b) White to move

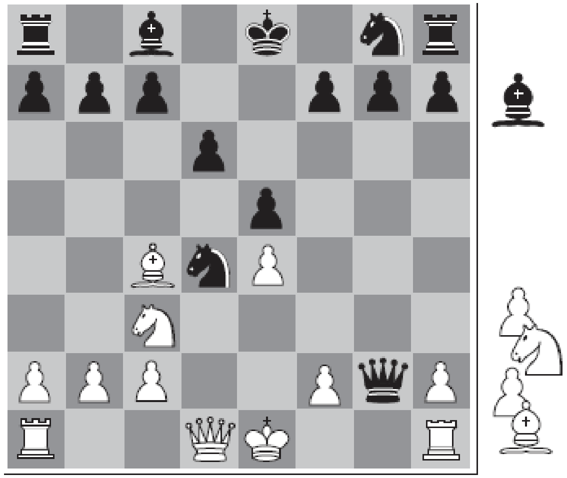
Figure 5.8 **FILES: figures/chess-evaluation3.eps (Tue Nov 3 16:22:33 2009).** Two chess positions that differ only in the position of the rook at lower right. In (a), Black has an advantage of a knight and two pawns, which should be enough to win the game. In (b), White will capture the queen, giving it an advantage that should be strong enough to win.

Evaluation functions

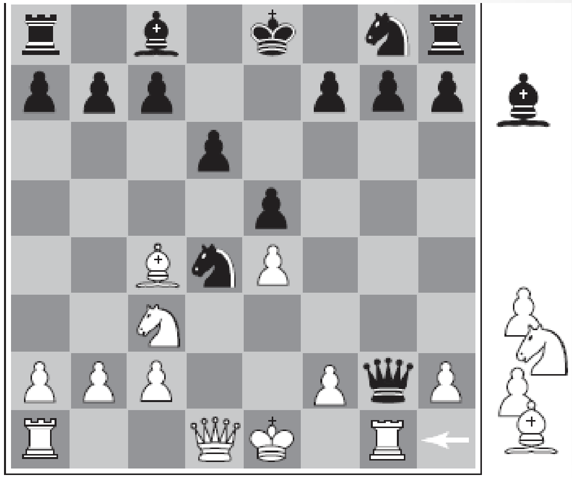
- A function of board features
  - Use proportions of board-states with winning, losing, and drawing states to compute probabilities.
    - 72% winning (1.0)
    - 20% draws (0.0)
    - 8% losses (0.5)
    - Then: evalFunction(board state) = (0.72 * 1) + (0.2 * 0) + (0.08 * 0.5)
  - Use a **weighted linear sum** of board features (Can also use non-linear f)
    - Chess book: pawn = 1, bishop/knight = 3, rook = 5, queen = 9
    - Good pawn structure = A, king safety = B
    - evalFunction(board state) = w_1* pawns + w_2 * bishops + w_3 * knight + w_4 * rook + … + w_n * good pawn structure + ….
  - All this information for chess comes from centuries of human expertise
  - For new games?

When do we cutoff search

- Quiescence

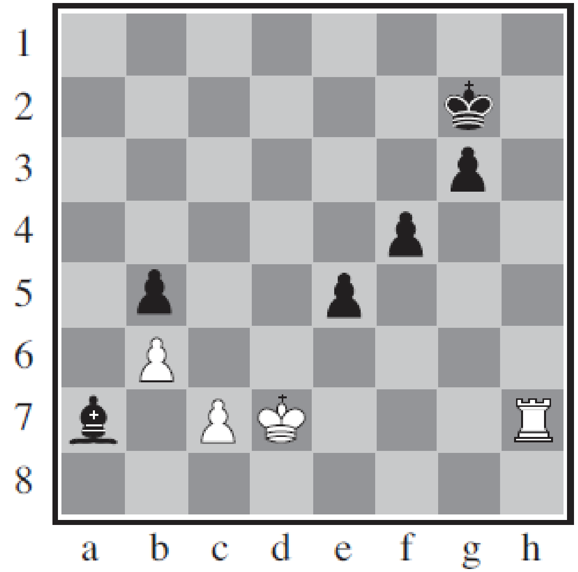


(a) White to move



(b) White to move

- Horizon effect and singular extension



Forward pruning

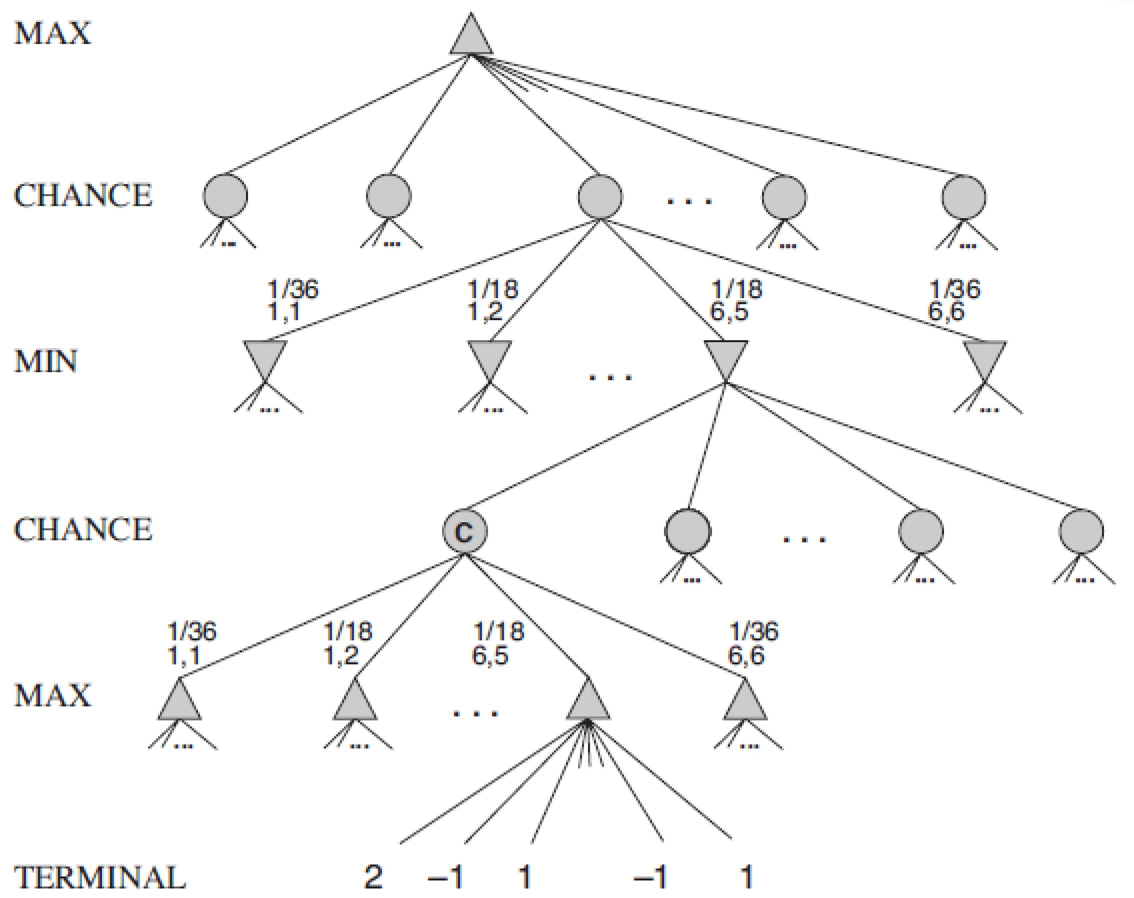
- Beam search
- ProbCut – learn from experience to reduce the chance that good moves will be pruned
  - Like alpha-beta but prunes nodes that are probably outside the current alpha-beta window
  - Othello
- Combine all these techniques plus

Table lookups

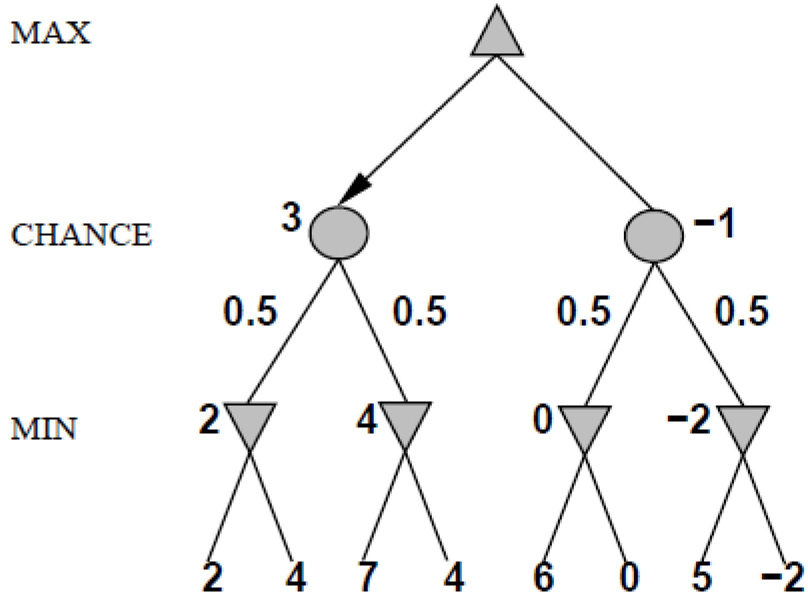
- Chess
  - Openings (perhaps upto 10 moves)
  - Endings (5, 6 pieces left)
    - King-Rook versus King (KRK)
    - King-Bishop-Knight versus King (KBNK)
- Checkers
  - Is solved!

Stochastic Games

- Chance is involved (Backgammon, Dominoes, …)
- Increases depth if modeled like:



Simple example (coin flipping)



Expected value minimax

if *state* is a MAX node **then**
    return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
if *state* is a MIN node **then**
    return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
if *state* is a chance node **then**
    return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

Backgammon

Dice rolls increase b: 21 possible rolls with 2 dice
Backgammon $\approx$ 20 legal moves (can be 6,000 with 1-1 roll)

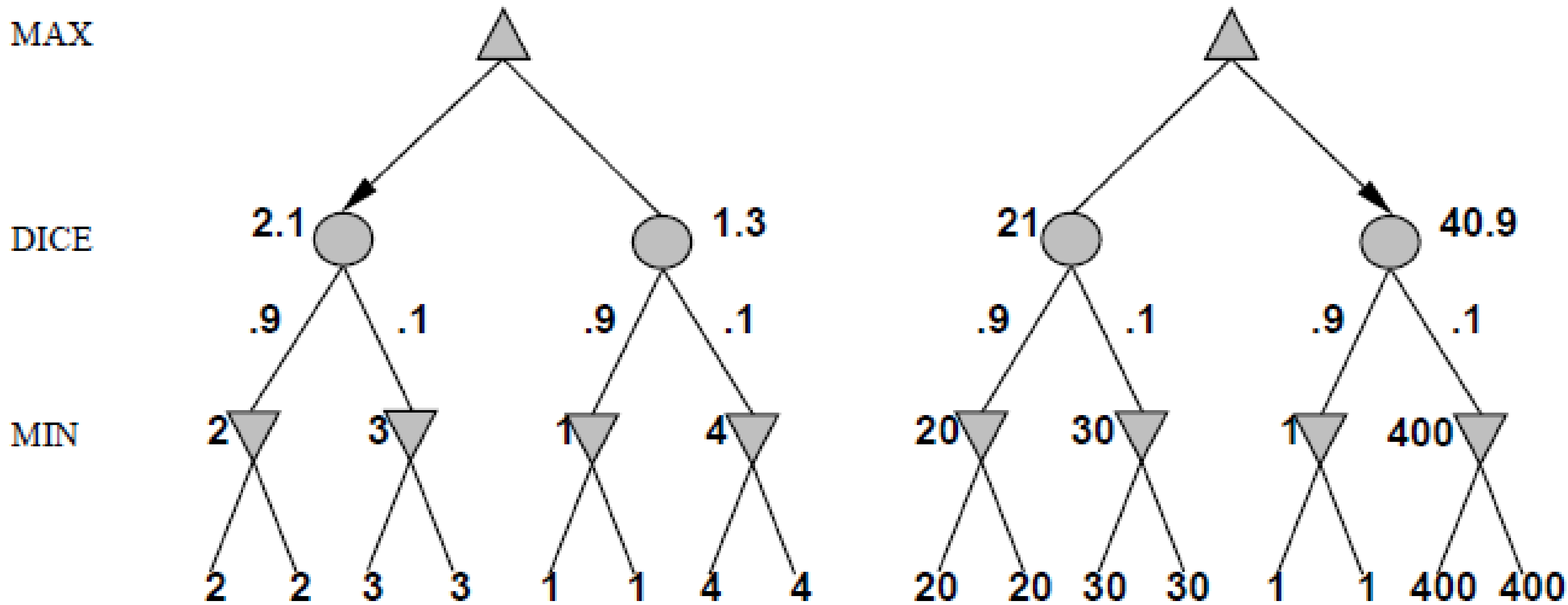
$$\text{depth } 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$$

As depth increases, probability of reaching a given node shrinks
$\Rightarrow$ value of lookahead is diminished

α–β pruning is much less effective

TDGammon uses depth-2 search + very good Eval
$\approx$ world-champion level

With chance, exact values matter

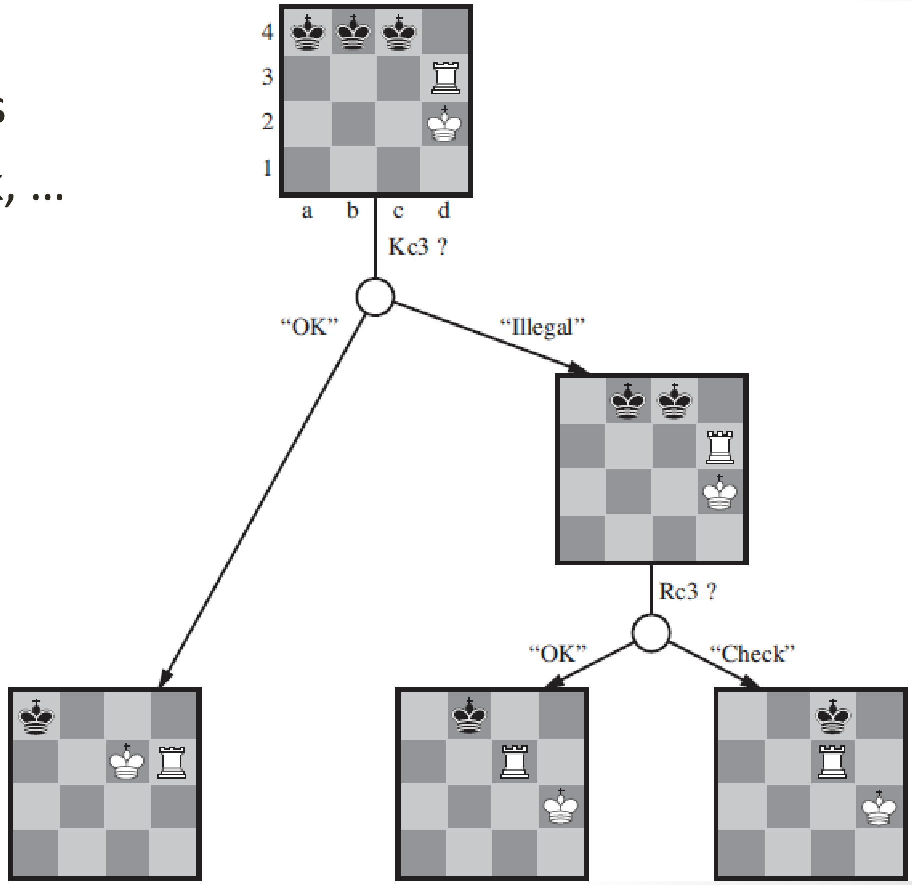


Behaviour is preserved only by positive linear transformation of EVAL

Hence EVAL should be proportional to the expected payoff

Fog of War

- Use belief states to represent the set of states you could be in given all the percepts so far
- Kriegspiel
  - You can only see your pieces
  - Judge says: Ok, illegal, check, …

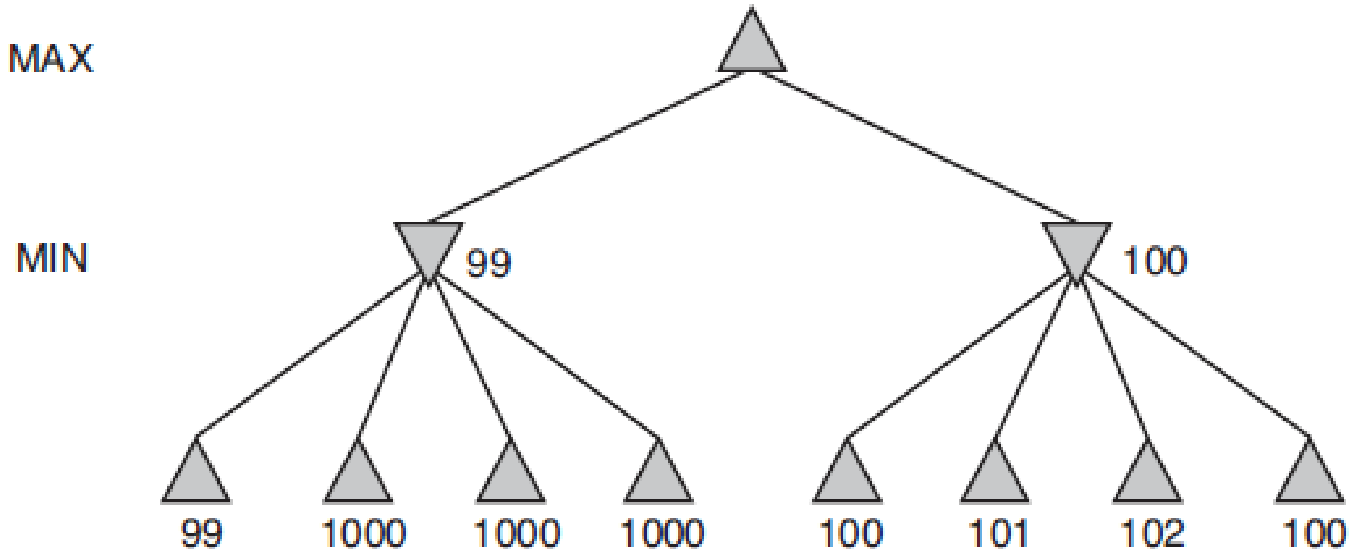


Card Games

- Consider all possible deals of a deck of cards, solve each deal as a fully observable game, then choose best move averaged over all deals
- Computationally infeasible but:
  - Let us do Monte Carlo approximation
  - Deal a 100 deals, a 1000 deals, … whatever is computational feasible
  - Choose best outcome move

- Read section 5.7 – state of the art game programs

Errors in evaluation functions!



Summary

- Games are fun to work on
- They give insight on several important issues in AI
  - Perfection is unattainable → approximate
  - Think about what to think about
  - Uncertainty constrains assignment of values to states
  - Optimal decisions depend on information state, not real state

- Games are to AI as grand prix racing is to automobile design

Searching with Nondeterministic actions

Search

- Problem solving by searching for a solution in a space of possible solutions
- Uninformed versus Informed search
- Atomic representation of state
- Solutions are fixed sequences of actions