EBERHARD KARLS UNIVERSITÄT TÜBINGEN

MATHEMATISCH-NATURWISSENSCHAFTLICHE FAKULTÄT

Artificial Intelligence
Chapter 5: Adversarial Search

Andreas Zell

After the Textbook: Artificial Intelligence,
A Modern Approach
by Stuart Russel and Peter Norvig (3rd Edition)

5.1 Games

- Chapter 2 introduced multiagent environments.
- Each agent must consider the actions of the other agents.
- Unpredictability of these actions can introduce contingencies into the problem-solving process.
- Games are competitive environments, in which agents' goals are in conflict, giving rise to adversarial search problems.

5.1 Games

- Mathematical game theory views any multiagent environment as a game, in which the impact of each agent on the others is significant, be it cooperative or competitive.
- In AI, the most common games are:
  - deterministic
  - turn-taking
  - two-player games
  - zero-sum games (result is draw, or a win and a loss)
  - have perfect information (i.e., fully observable)

5.1 Games

- Games have many advantages for AI research
- The state is easy to represent, there is usually a small number of possible actions, whose outcomes are defined by precise rules.
- Games are often too hard to solve, making them more interesting
- “easy to learn, but hard to master”

5.1 Games

- Example: Chess
- Average branching factor of 35
- Games often go to 50 moves by each player
- Search tree has about 35^{100} or 10^{154} nodes
- Search graph has about 10^{40} distinct nodes
- Calculating the optimal decision is infeasible
- Games often have time constraints, so efficiency plays an important role

5.1 Games

- Techniques for finding an optimal move when time is limited:
- Pruning: Ignore portions of the search tree that make no difference to the final choice
- Evaluation functions: Heuristics to approximate the true utility of a state without doing a complete search
- Imperfect information: How to find an optimal move if not all information are available?

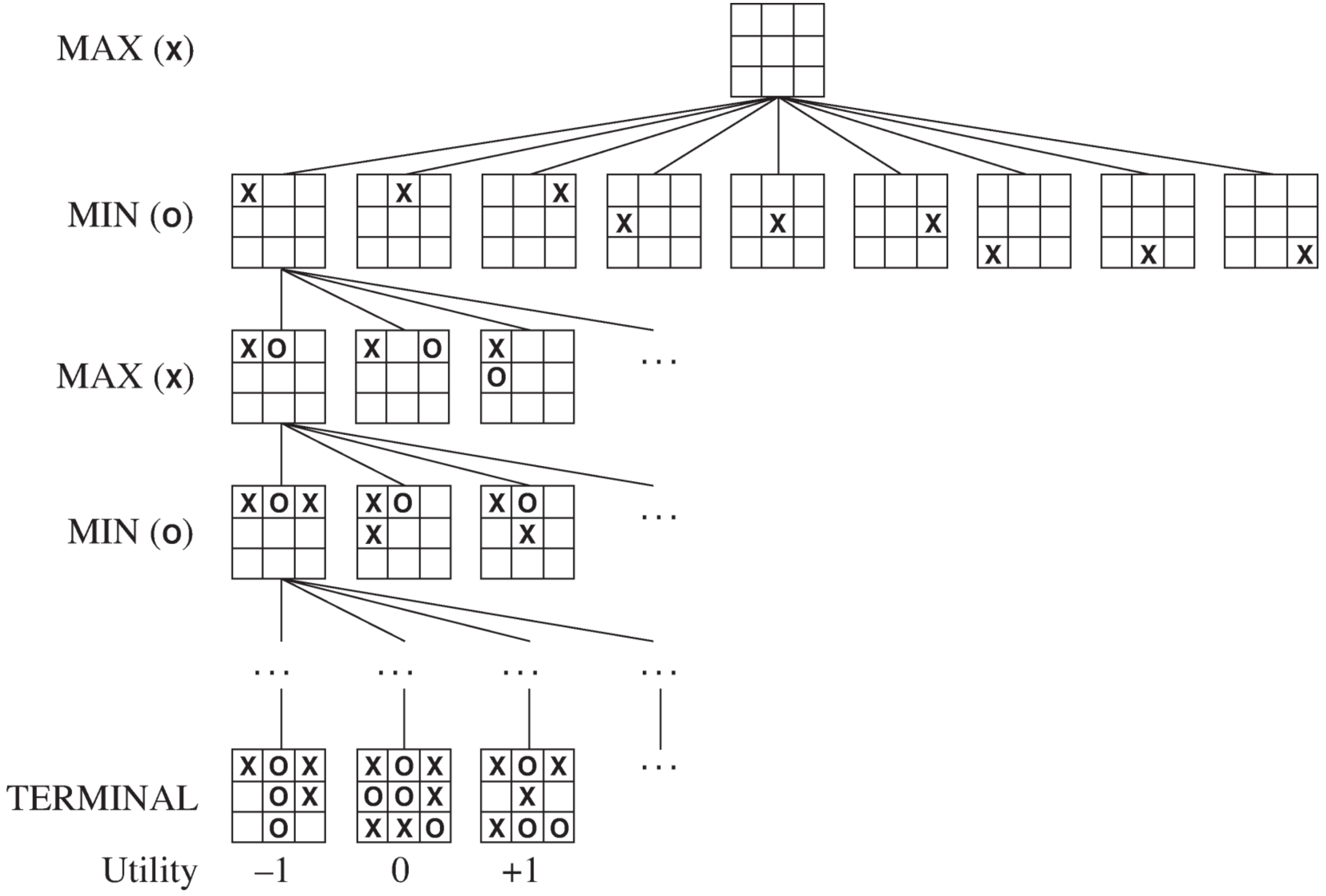
5.1 Games

- Definition of a game as a search problem
  - S_0: initial state, game setup
  - PLAYER(*s*): player that has the move in state *s*
  - ACTIONS(*s*): legal moves in state *s*
  - RESULT(*s*,*a*): transition model, result of move *a*
  - TERMINAL-TEST(*s*): terminal test or goal test. States where the game has ended are called terminal states
  - UTILITY(*s*,*p*): utility function, defines the numeric value in a terminal state s for player *p*
    - Example: Loss is -1, draw is 0, and win is +1
    - zero-sum: Sum of utility of both players is zero

5.1 Games

- Initial state S_0, ACTIONS(s) and RESULT(s,a) define the game tree
- Nodes are game states
- Edges are moves
- Two players, let’s name them MIN and MAX
- Alternating turns, MAX begins
- Example: TicTacToe
  - only 9! = 362,880 terminal nodes in search tree
  - remember, chess has over 10^{40} nodes

5.1 Games



5.2 Optimal Decisions in Games

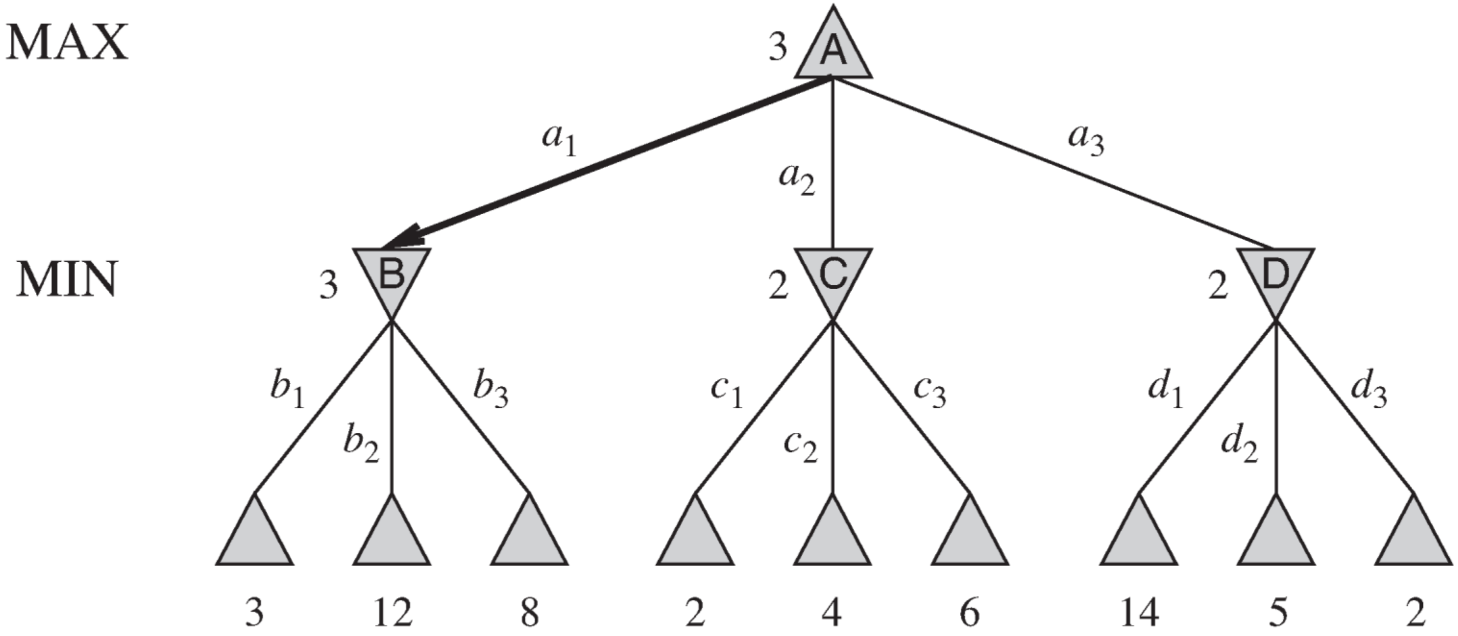
- In a normal search problem, the optimal solution is a sequence of actions leading to a goal state.
- In adversarial search, every other action is chosen by MIN.
- MAX must find a strategy that specifies his move after every possible response to his move by MIN, then his moves after every possible response by MIN to those moves, and so on.
- An optimal strategy must lead to outcomes at least as good as any other strategy when playing against an infallible opponent.

5.2 Optimal Decisions in Games

- Similar to AND-OR search algorithm
- MAX is OR, MIN is AND
- Given a game tree, the optimal strategy can be determined from the minmax value of each node
- It is assumed that both players play optimally

$$MINIMAX(s) = \begin{cases} UTILITY(s) & \text{if } TERMINAL\text{-}TEST(s) \\ \max_{a \in Actions(s)} MINIMAX(RESULT(s,a)) & \text{if } PLAYER(s) = MAX \\ \min_{a \in Actions(s)} MINIMAX(RESULT(s,a)) & \text{if } PLAYER(s) = MIN \end{cases}$$

- Example game tree: MAX always chooses the move that maximizes the minimax value (a_1), while MIN selects the one that minimizes it (b_1)



5.2 Optimal Decisions in Games

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- In game parlance, the previous game tree is considered one move deep
- a move consists of two half-moves
- a half-move is also called a ply

5.2 Optimal Decisions in Games

- The optimal strategy of MAX assumes that MIN also plays optimally
- If MIN does not play optimally, MAX will do better, at least equally good
- There might be other strategies that are better if MIN plays suboptimally, but those will always be worse against optimal opponents
- The minimax algorithm computes the optimal decision using the minimax values by traversing the game tree all the way down to the leaves

5.2 Optimal Decisions in Games

function MINIMAX-DECISION(*state*) **returns** *an action*
  return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(*state*, *a*))

---

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

---

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

5.2 Optimal Decisions in Games

- The minimax algorithm performs a depth-first search
- For a tree with maximum depth *m* and *b* possible actions at each state
  - time complexity is $O(b^m)$
  - space complexity is $O(bm)$ if all actions are generated at once, $O(m)$ if actions are generated one at a time
- For most real games, this time complexity is totally impractical

5.2 Optimal Decisions in Games

- Minimax can be extended for games that have more than two players
- Instead of a utility value, there is now a utility vector, for example $<v_a, v_b, v_c>$ for three players
- Instead of MAX choosing the action maximizing the minimax value and MIN minimizing it, each player chooses the action maximizing his utility vector component
- Instead of a value being backed up the tree, a vector is backed up

5.3 Alpha-Beta Pruning

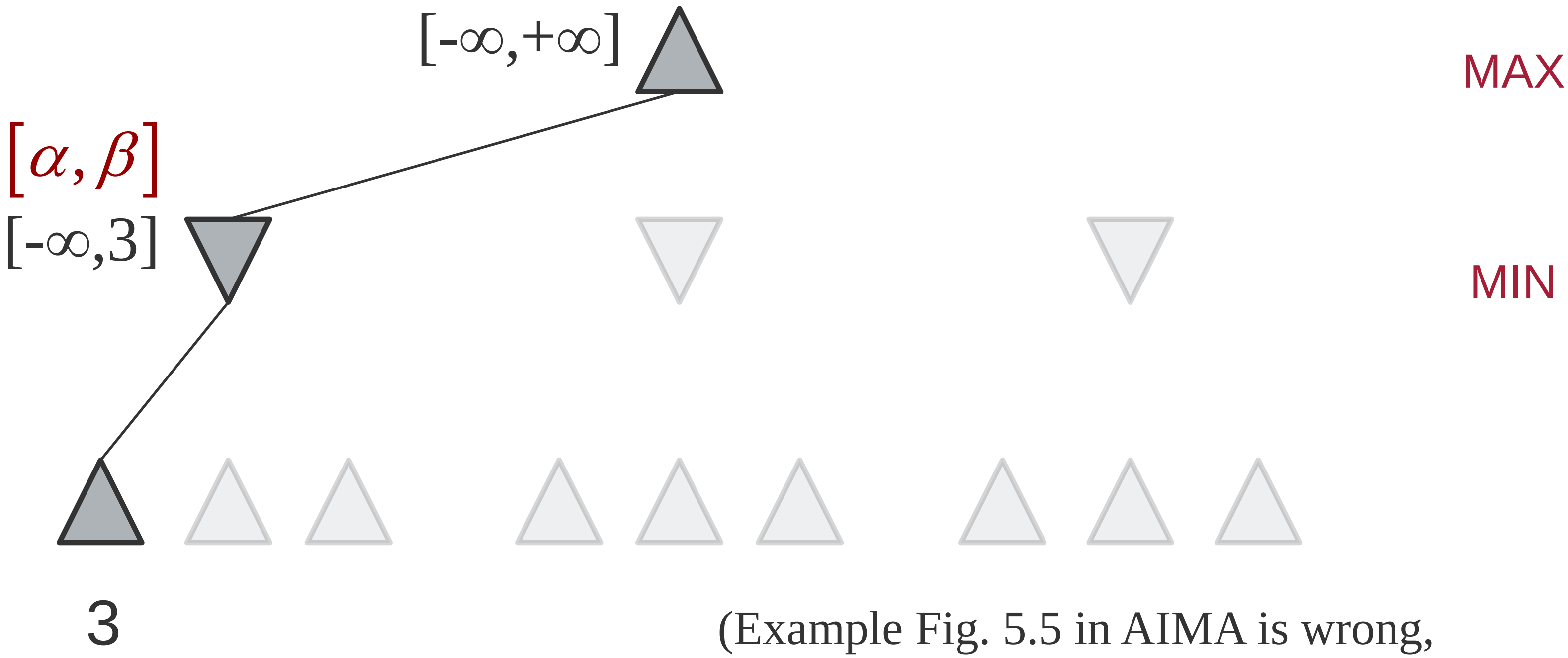
- Obviously, the exponential time complexity of the minimax algorithm is a problem
- Approach: Prune away those parts of the tree that we don’t need to examine, because they wouldn’t change the optimal decision
- Examine the previous game tree again, this time applying alpha-beta pruning

5.3 Alpha-Beta Pruning

- α: value of the best (highest-value) choice found so far any choice point along the path for MAX
- β: value of the best (lowest-value) choice found so far any choice point along the path for MIN
- If an action of MAX leads to a state where MIN has at least one action that leads to a worse result than an already evaluated action of MAX, we don't need to examine it further
- Same holds vice versa

5.3 Alpha-Beta Pruning

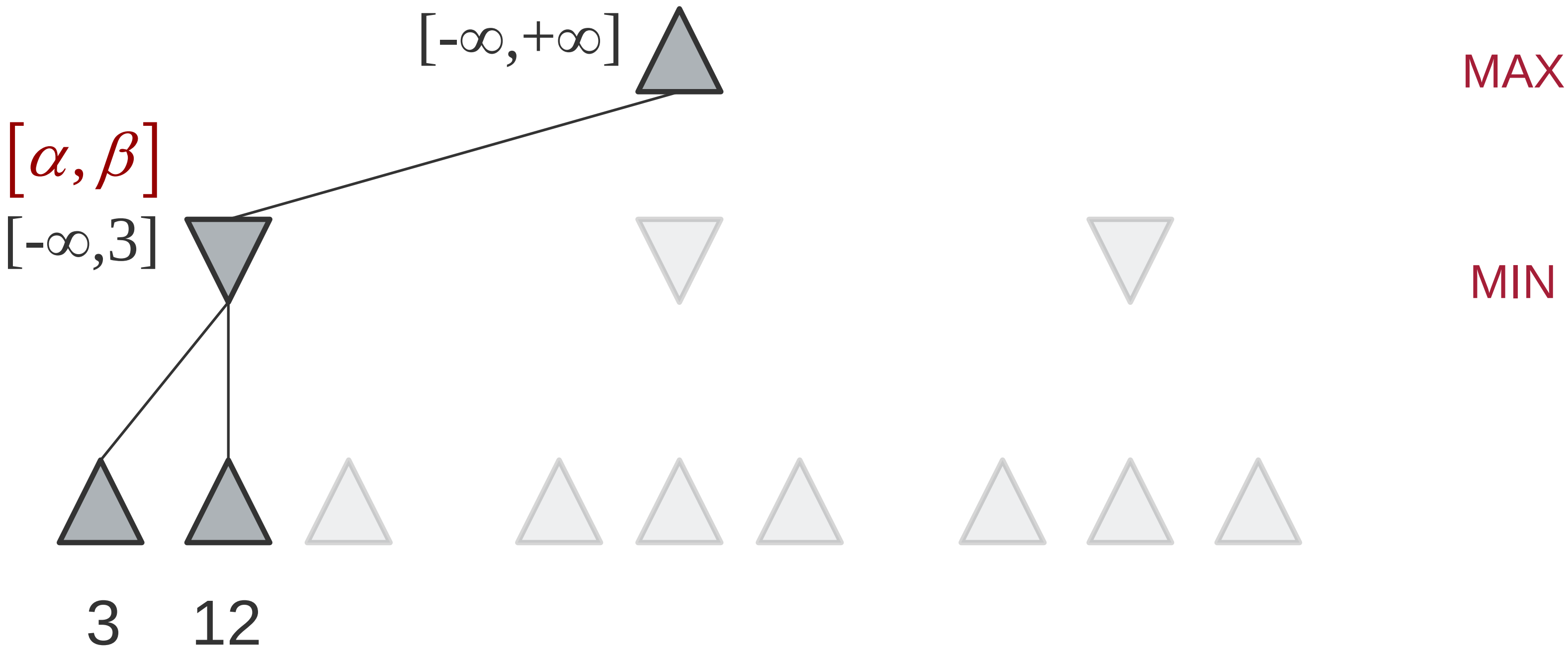
EBERHARD KARLS UNIVERSITÄT TÜBINGEN



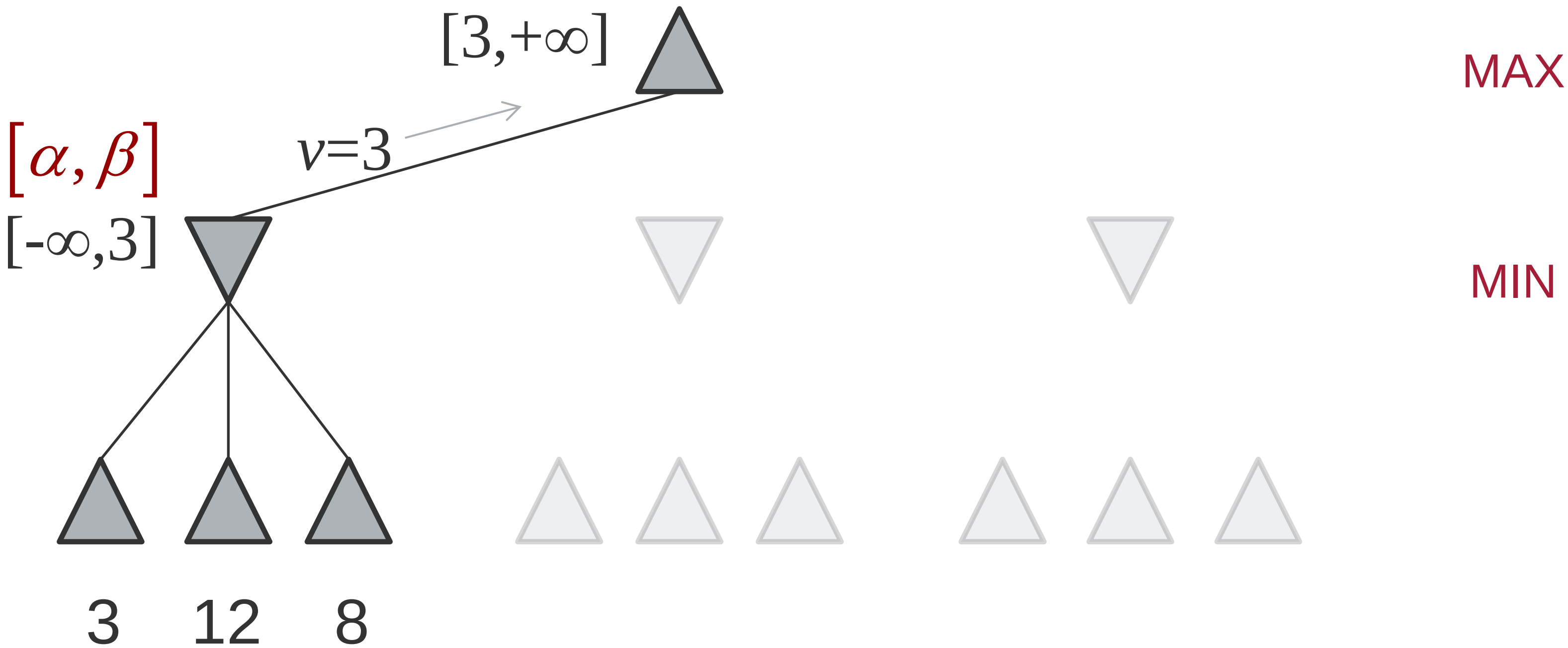
(Example Fig. 5.5 in AIMA is wrong, is not the algorithm in Fig. 5.7)

- Best choice for MIN so far, set β = 3
- Best choice for MAX can't be known yet

5.3 Alpha-Beta Pruning

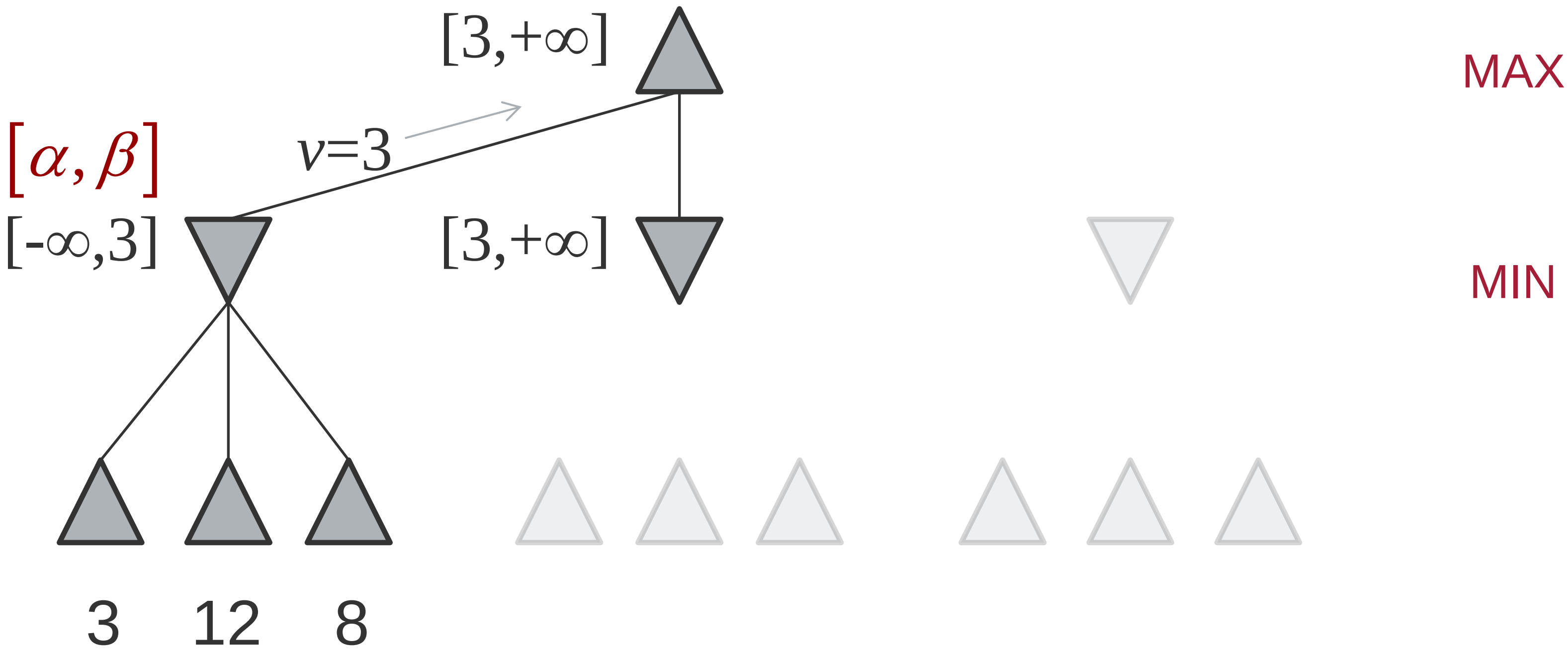


- Next action has higher value than 3, so MIN won’t choose it



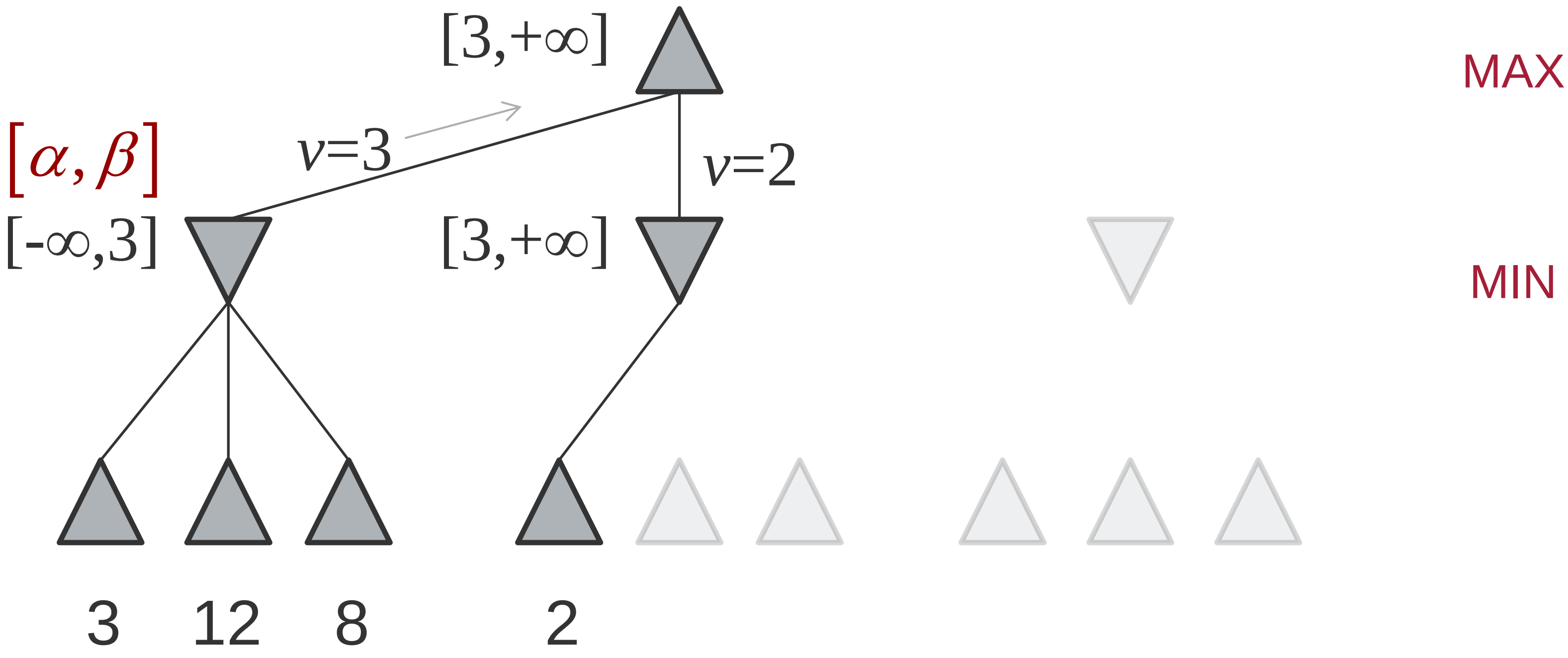
- Still no better choice for MIN, so back up minimal value and set as α = 3 on MAX

5.3 Alpha-Beta Pruning



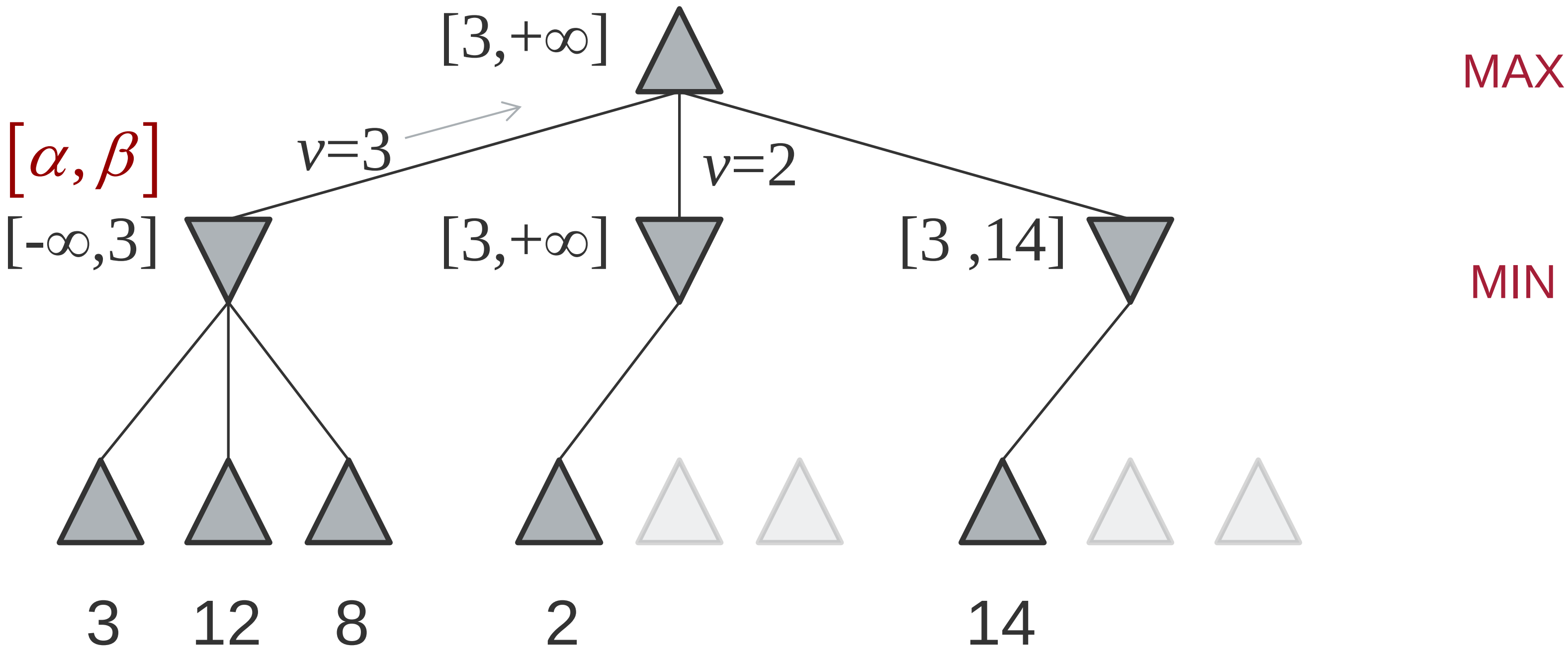
- Backed up α value, best value for MAX so far
- Go down to next node, pass α and β along

5.3 Alpha-Beta Pruning

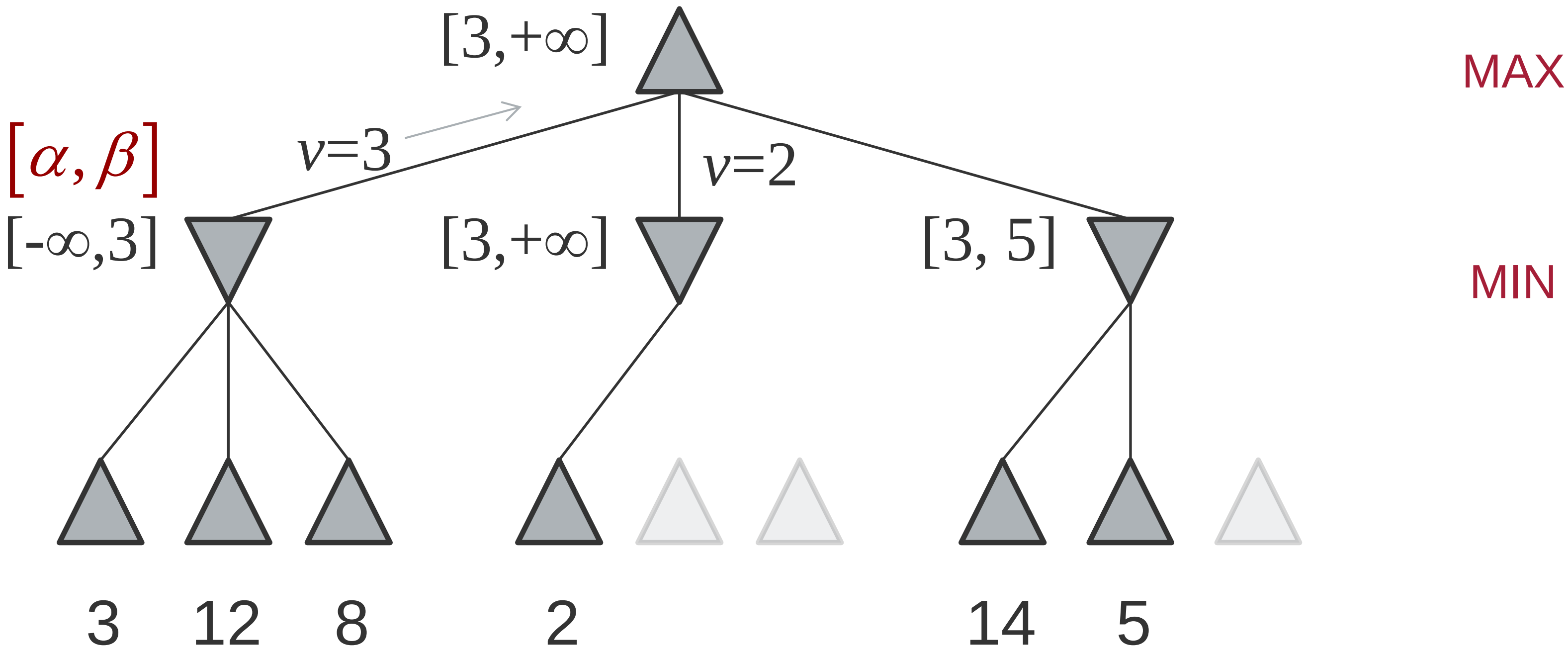


- MIN will choose at most a value of 2, but MAX already knows a better move (2 < α). Prune here

5.3 Alpha-Beta Pruning

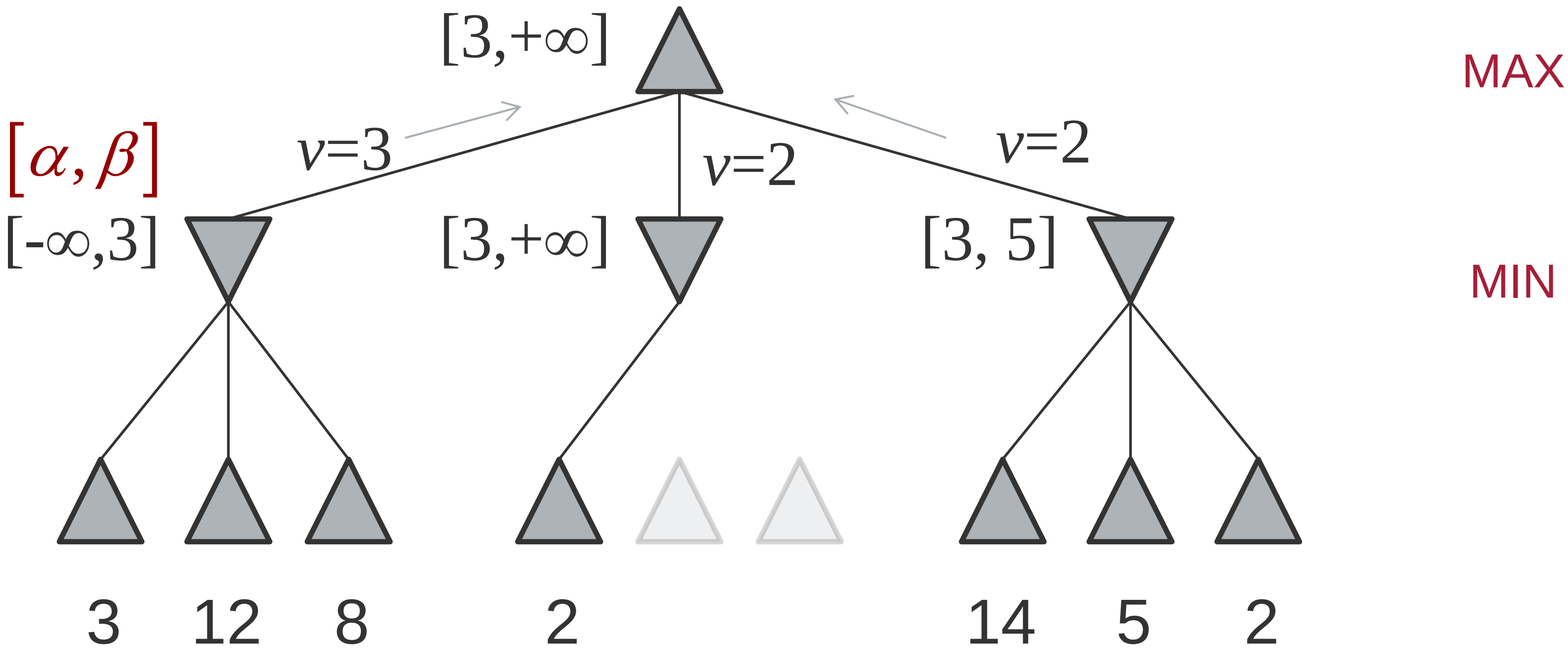


- Go down to next node, pass α and β along
- Leaf 14 >= α, update β



- Leaf 5 >= α, update β

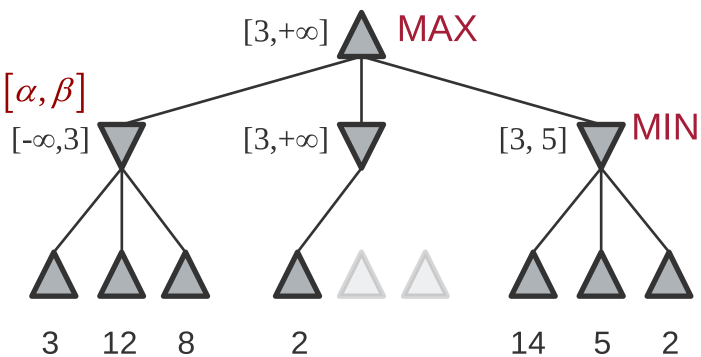
5.3 Alpha-Beta Pruning



- MIN would choose action leading to 2
- 2 < α, so no β update, pass up 2 to MAX
- Best move for MAX in root known now

5.3 Alpha-Beta Pruning

- Algorithm for minimax search with alpha-beta pruning



function ALPHA-BETA-SEARCH($state$) **returns** an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the $action$ in ACTIONS($state$) with value v

---

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), 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*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

5.3 Alpha-Beta Pruning

- Last subtree in example could have been skipped earlier, if action with result 2 would have been the first examined action
- Is it possible to order the examined moves in a way to allow for earlier alpha-beta pruning?
- If best moves are examined first, alpha-beta only needs to examine $O(b^{m/2})$ nodes, instead of $O(b^m)$ for minimax
- Branching factor b becomes $\sqrt{b}$
- If successors are examined in random order, alpha-beta needs to examine $O(b^{3m/4})$ nodes

5.3 Alpha-Beta Pruning

- If perfect ordering was possible, we would already know the best move without searching
- But heuristics can be used to select moves first that are probably better (e.g. attacking moves)
- Best moves are called killer moves and are selected by the killer move heuristic
- States may occur multiple times, so it is possible to create a transposition table, storing game states that already have been evaluated
- But: Storing all states is not practical, because there are too many

5.4 Imperfect Real-Time Decisions

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Alpha-beta prunes large portions of game tree away, but still has to go down to terminal nodes
- Approach: Apply a heuristic evaluation function on nonterminal nodes to limit search depth
- A cutoff-test decides when to apply heuristic

$$H\text{-}MINIMAX(s,d) =$$

$$\begin{cases} EVAL(s) & \text{if } CUTOFF\text{-}TEST(s,d) \\ \max_{a \in Actions(s)} H\text{-}MINIMAX(RESULT(s,a), d+1) & \text{if } PLAYER(s) = MAX \\ \min_{a \in Actions(s)} H\text{-}MINIMAX(RESULT(s,a), d+1) & \text{if } PLAYER(s) = MIN \end{cases}$$

5.4 Imperfect Real-Time Decisions

- Heuristic evaluation functions in Chapter 3 return the estimed distance to the goal
- Heuristic evaluation functions here return the estimated utility of a game state
- Performance of a game-playing program highly depends on the quality of its evaluation function
- Incorrect evaluation of a game state might direct the search into a wrong direction

Requirements

1. The evaluation function should order terminal nodes in the same way as the utility function
   - Win > Draw > Loss
2. The computation of this function must be fast
3. The result of the evaluation function should strongly correlate with the chances of winning for nonterminal states
   - Here, “chance” means the uncertainty due to computational limitations, even if the game itself is deterministic

5.4 Imperfect Real-Time Decisions

- One way to define an evaluation function is to group states into categories by their features
- Features can be for example (in chess):
  - number of pawns, queens, knights for black and white
  - number of threatened pieces for black and white
- A large knowledge-base could be used to calculate expected values, e.g.
  - In 80% of previous games where white has three pawns and black has two pawns, white won the game, in 10% it was a draw and in 10% white lost

$$\text{expected value} = (0.8 \times 1) + (0.1 \times 0.5) + (0.1 \times 0)$$

5.4 Imperfect Real-Time Decisions

- In practice, this requires the analysis of too many categories and too much experience
- The evaluation function doesn't need to return expected values, as long as it orders the states correctly
- Chess often uses approximate material values
  - a pawn is worth 1, a knight or bishop is worth 3, etc.
  - features like "king safety" might be worth 0.5
- Values are added for each player

- Here, the evaluation function becomes a weighted linear function

$$\mathrm{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i$$

where w_i is a weight and f_i is a feature.

- Strong assumption here: The value of each piece is independent of the others
- In chess, bishops are more powerful in the endgame (move number is high or number of remaining pieces is low)
- So, in an endgame, bishops are worth more

5.4 Imperfect Real-Time Decisions

- Deciding when to cut off the search is important
- Simple approach: Stop at a fixed depth
- Better: Use iterative deepening
- The deepest completed search at the end of the time limit specifies the selected move
- Bonus advantage: Previous depth searches can give additional information for move ordering
- But, there still might be situations, where an important move is just one move deeper than calculated

5.4 Imperfect Real-Time Decisions

- Quiescence search: The evaluation function is only applied to quiescent positions, i.e. that are unlikely to dramatically change in the near future
- Positions with favorable captures are expanded until quiescent positions are reached
- There still is the horizon effect that some important moves are “pushed over the horizon” and are not included in the evaluation
- There are approaches to mitigate this effect, but we won’t discuss them here

5.4 Imperfect Real-Time Decisions

- A more human-like approach is forward pruning
- In chess, humans usually consider only a few moves in each position, pruning the others without further consideration
- Beam search only considers the *n* best moves in each position (according to the evaluation)
- This might prune away the subtree with the best overall move

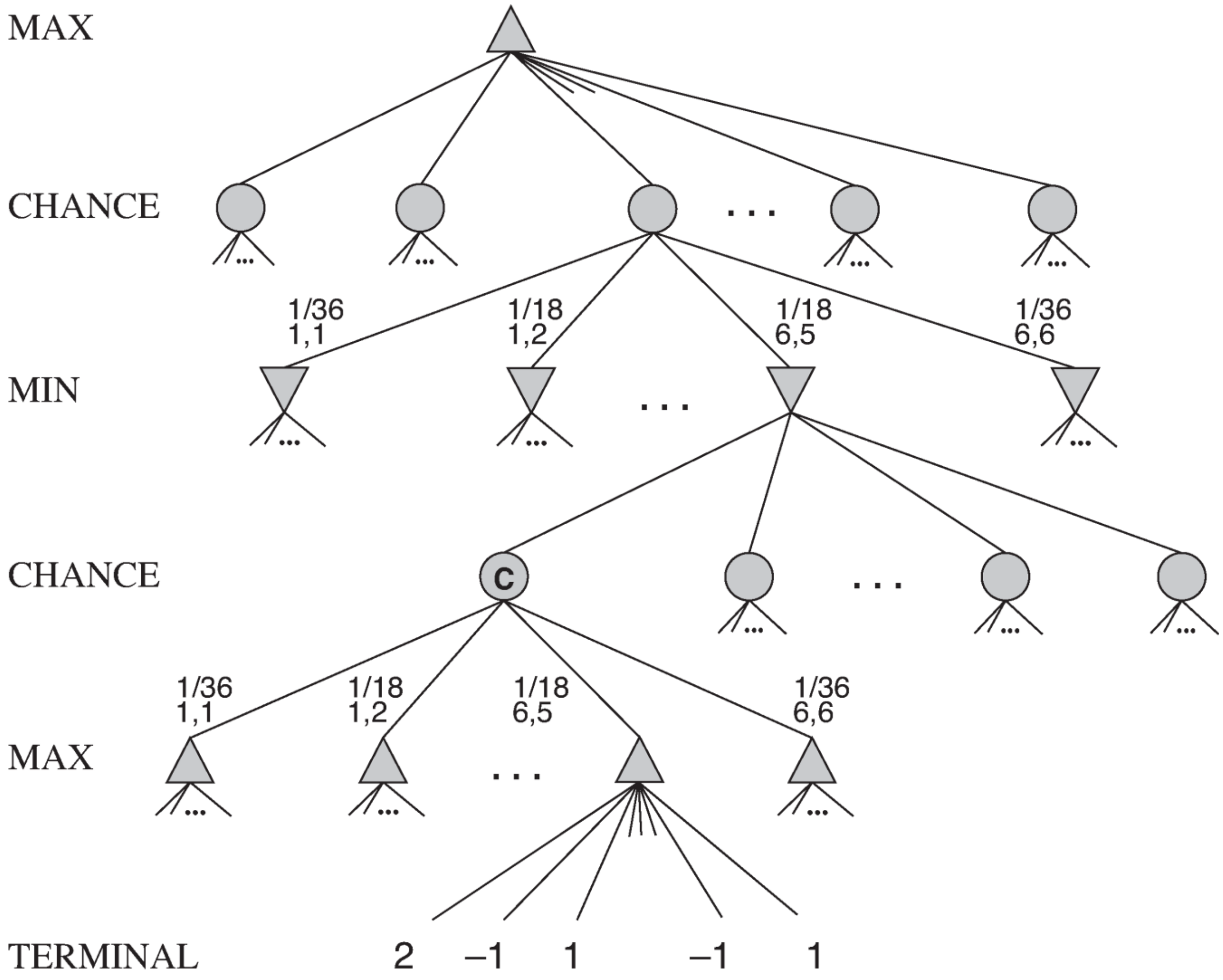
EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- For (phases of) games with only a limited number of states, a lookup table can be feasible
- Opening tables in chess are very successful (every game has the same starting position, so there is a large data base for statistics)
- In endgames, the lookup table can be made into a graph, where for each position the best move is already known
- E.g. King-bishop-and-knight-versus-king (KBNK) has 462 x 62 x 61 x 2 = 3,494,568 positions

5.5 Stochastic Games

- There are also many stochastic games with random elements, like throwing a dice
- Example: Backgammon
- Each player knows his own legal moves after throwing the dice, but he doesn't know what will be the legal moves of the opponent
- Standard game tree can't be constructed
- Add chance nodes to the game tree

5.5 Stochastic Games



5.5 Stochastic Games

- How to compute the best move here?
- Calculate the expected value of each move
- That is, the weighted average over all possible outcomes of chance nodes
- Generalization of the minimax value to the expected minimax value

5.5 Stochastic Games

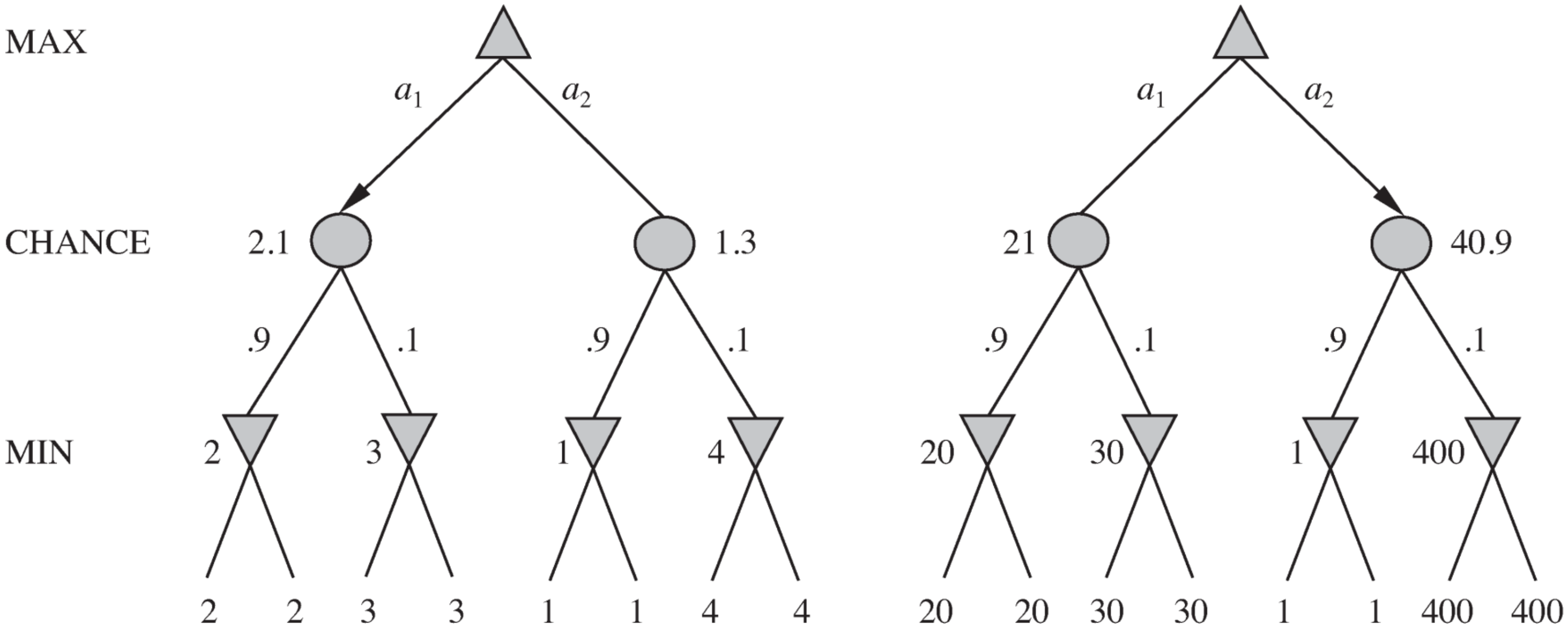
- additional player CHANCE
- *r* represents a chance event (e.g. dice roll)
- *RESULT(s,r)* is the same state as *s*, with the additional fact that the chance event was *r*

$$
EXPECTIMINIMAX(s) =
\begin{cases}
UTILITY(s) & \text{if } TERMINAL\text{-}TEST(s) \\
\max_a EXPECTIMINIMAX(RESULT(s,a)) & \text{if } PLAYER(s) = MAX \\
\min_a EXPECTIMINIMAX(RESULT(s,a)) & \text{if } PLAYER(s) = MIN \\
\sum_r P(r) EXPECTIMINIMAX(RESULT(s,r)) & \text{if } PLAYER(s) = CHANCE
\end{cases}
$$

5.5 Stochastic Games

- Obvious approximation is to cut the search off at a certain point and apply an evaluation function
- For deterministic games, the evaluation function only needed to order the leaves correctly
- For stochastic games, this can lead to problems
- Solution is a stronger requirement:
- Evaluation function must be a positive linear transformation of the probability of winning

- both evaluation functions order leaves the same
- but they lead to different expected values at the chance nodes



5.5 Stochastic Games

- Time complexity for normal minimax is $O(b^m)$
  - b is the branching factor
  - m is the maximum depth of the game tree
- With addition of chance events, it is $O(b^m n^m)$
  - n is the number of distinct chance events
- In backgammon, the maximal feasible search depth was three plies
- Alpha-beta pruning ignores future developments that won't happen given best play
- With chance events, this becomes harder

5.5 Stochastic Games

- For alpha-beta pruning, we need find upper and lower bounds for nodes
- Is it possible to find these bounds for chance nodes without examining all children?
- Yes, if there are bounds on the possible values of the utility function, the expected values are also bounded
- Alternate approach: Monte Carlo simulation
- Simulate thousands of games from a position with random dice rolls and use the resulting win percentage as an evaluation function

5.6 Partially Observable Games

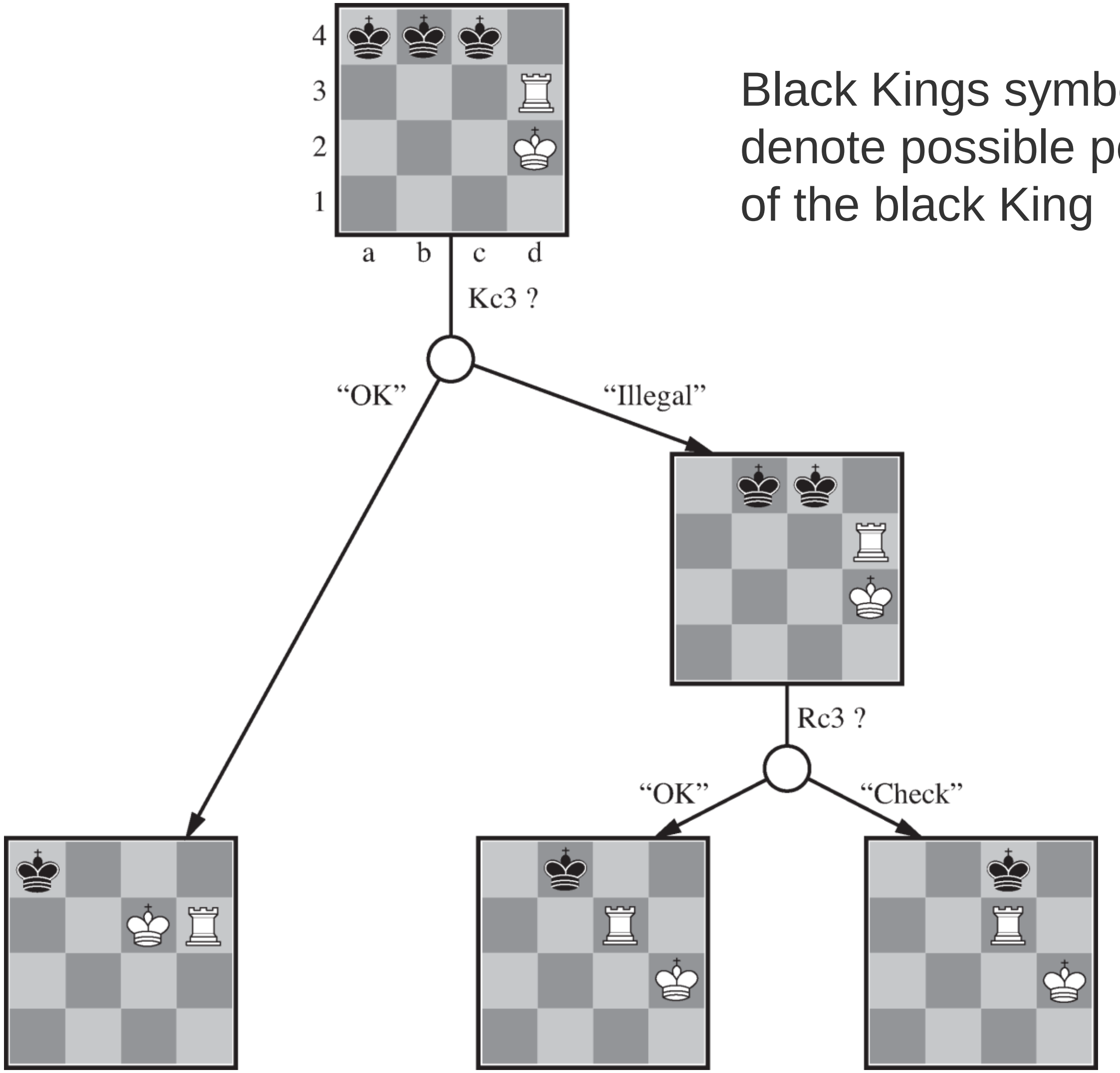
- So far, all games were fully observable
- But some games have the “fog of war”
- Existence and positions of enemy units is unknown until revealed by direct contact
- This adds new aspects to games
- The purpose of some actions can be to gather information only
- Successful strategies may involve bluffing

5.6 Partially Observable Games

- “Kriegspiel”:
- A partially observable variant of chess
- Moves and pieces of the opponent are hidden
- Players propose an action to a referee
  - If the action is illegal, the referee announces “illegal”
  - The player keeps proposing moves until a legal one is found, which is then executed
  - The referee then announces things like
    - “Capture on square X”
    - “Check by [direction]”
  - It is the other player’s turn to move

5.6 Partially Observable Games

- Players have to act upon belief states (see 4.4)
- The initial belief state is a singleton, because the starting positions are known
- With each move, the belief state contains more positions
- A winning strategy, or guaranteed checkmate, is one that, for each possible percept sequence, leads to an actual checkmate for every possible board state in the current belief state, regardless of how the opponent moves



Black Kings symbols here denote possible positions of the black King

5.6 Partially Observable Games

- With this definition of a strategy, the belief state of the opponent is irrelevant, which greatly simplifies computation
- There is also a probabilistic checkmate:
  A strategy that contains randomized moves and will still lead to a checkmate, eventually
- A strategy that will only work for some of the positions in a belief state is called an accidental checkmate
- For that, it can be helpful to consider whether some positions are more likely than others

5.6 Partially Observable Games

- Many card games also provide examples of stochastic partial observability
- E.g., cards are dealt randomly at the beginning of the game and a player's hand is not visible to the others (like in Poker, Bridge, Skat, …)
- All random effects occurred at the beginning
- Solve each possible state as if it were a fully observable game, then choose the move with the best outcome averaged over all deals

5.6 Partially Observable Games

- With an exact minimax

$$\arg\max_a \sum_a P(s)MINIMAX(RESULT(s,a))$$

- If computationally infeasible, run *H-MINIMAX*
- If the number of possible deals is too large, one can use a Monte Carlo approximation

$$\arg\max_a \frac{1}{N}\sum_{i=1}^{N} P(s)MINIMAX(RESULT(s_i,a))$$

- But both approaches assume a fully observable game after the first move

5.6 Partially Observable Games

- Those strategies will never select actions that gather information
- They never choose moves that hide information, or that provide information to partners
- They will also never bluff, because they assume that the opponent can see all cards

5.7 State-of-the-Art Game Programs

- Chess: Deep Blue (IBM, 1997)
  - 30 IBM RS/6000 processors for alpha-beta search
  - 480 custom VLSI chess processors for move generation, move ordering and leaf node evaluation
  - up to 30 billion positions per move, search depth 14
  - evaluation function with 8000 features
  - opening book of 4000 positions
  - database of 700,000 grandmaster games
  - large endgame database of solved positions containing all positions with 5 pieces and many with 6

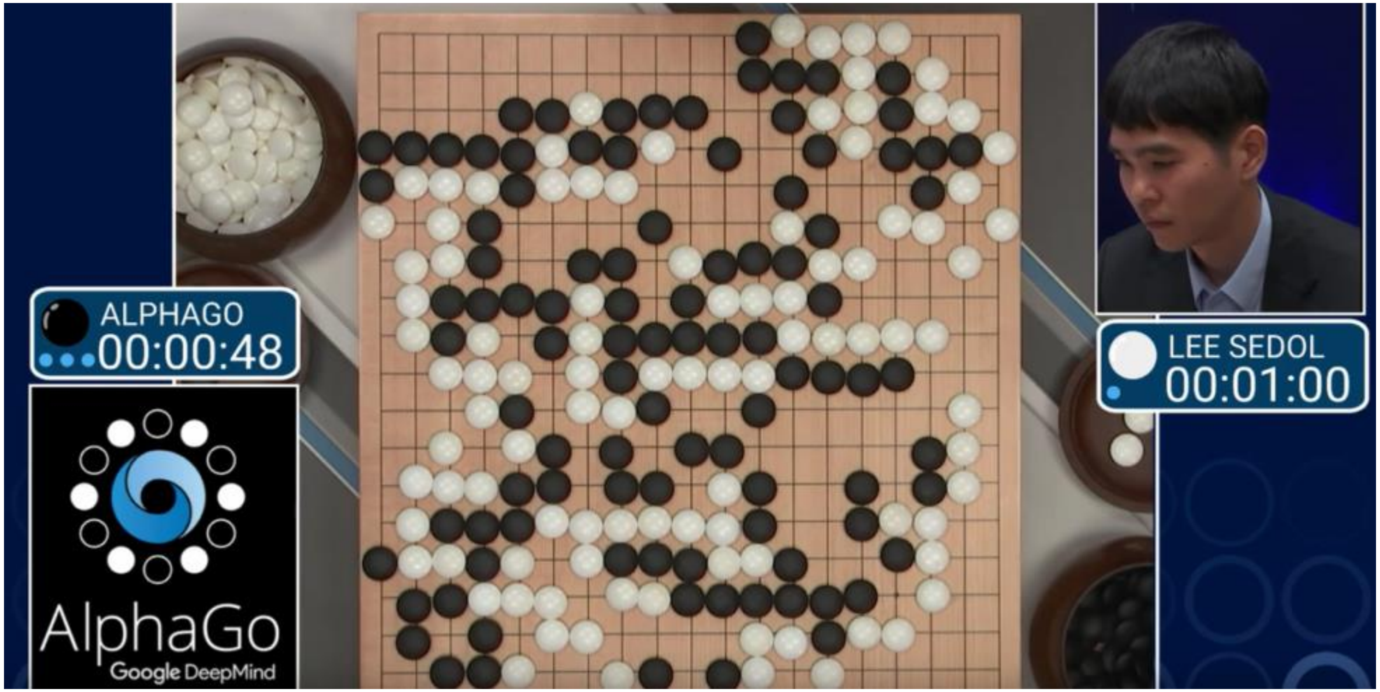
- Chess on standard PCs
  - Algorithmic improvements allow standard PCs to win World Computer Chess Championships
  - Effective pruning heuristics reduce the effective branching factor to less than 3
  - Most important one is the null move heuristic, where the opponent moves twice in the search, generating a good lower bound on the current position
  - There is also futility pruning, helping to decide in advance which moves will cause a beta cutoff in successor nodes
- Top chess programs beat all human contenders

5.7 State-of-the-Art Game Programs

- Checkers: Chinook
  - It defeated the human world champion in 1990
  - Since 2007, it can play perfectly by using alpha-beta search and an endgame database with 39 trillion positions
- Backgammon: TD-Gammon
  - The evaluation function was improved using reinforcement learning and neural networks
  - After playing more than 1 trillion training games against itself, it is competitive with top human players
  - Some of its opening moves radically altered the previous general game play knowledge

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Go: AlphaGo
  - AlphaGo, developed by Google DeepMind (London) defeated the best professional Go player Lee Sedol (9-dan, Korea) 4:1 in 5 matches, March 2016



5.7 State-of-the-Art Game Programs

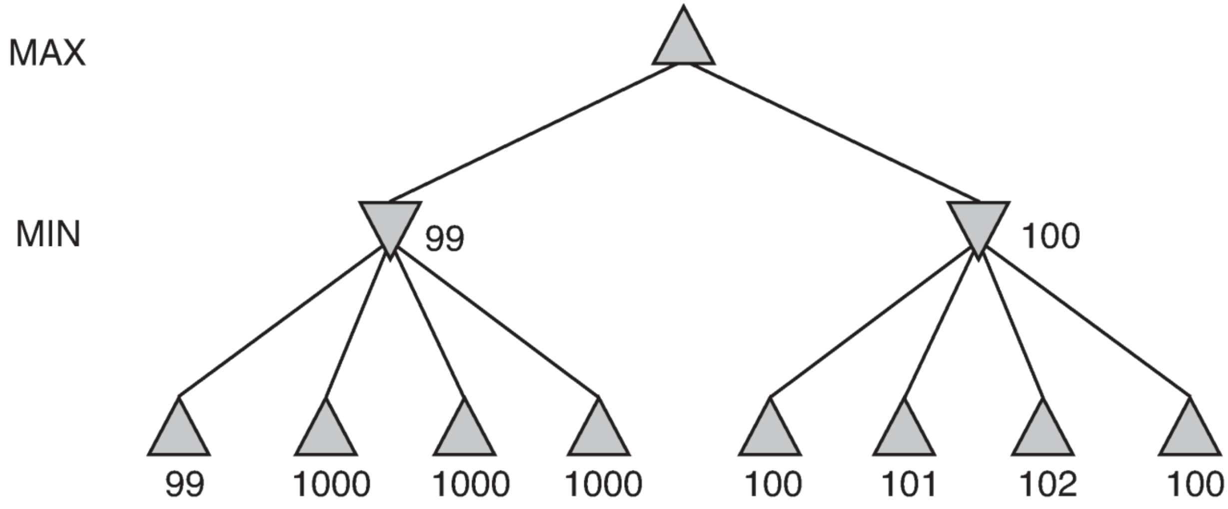
- AlphaGo used 1,920 CPUs and 280 GPUs in the match against Lee Sedol
- It used Monte Carlo tree search, guided by a value network and a policy network, both implemented using deep neural network technology
- AlphaGo was initially trained on a database of around 30 million moves of expert human players
- Then it was trained further by playing large numbers of games against other instances of itself, using reinforcement learning to improve its play
- https://de.wikipedia.org/wiki/AlphaGo
- https://en.wikipedia.org/wiki/AlphaGo

5.8 Alternative Approaches

- Minimax, evaluation functions and alpha-beta are just one way of approximating optimal decisions in games
- Optimality heavily depends on the correctness of the evaluation function
- Suppose that the evaluation of each node can have an error of mean zero and a standard deviation of σ and is independent of the evaluation of the other nodes

5.8 Alternative Approaches

- With σ = 5, the left branch is better 71% of the time, whereas for σ = 2 it is only better 58% of the time
- In reality, a possible error of the evaluation function is not independent of the other nodes



5.8 Alternative Approaches

- Utility of a node expansion
- Don’t waste (computation) time evaluating nodes that are unlikely to lead to a good move
- A good search algorithm should select node expansions of high utility, i.e. nodes that are likely to lead to the discovery of a significantly better move
- This also works for symmetrical moves, for which no amount of search will show that one move is better than another

5.8 Alternative Approaches

- This kind of reasoning about what computations to do is called metareasoning
- Alpha-beta uses the simplest kind of metareasoning, a theorem to the effect that certain branches of the tree can be ignored without loss.

5.8 Alternative Approaches

- Humans usually don’t play games the way described in this chapter
- Often one has a particular goal in mind and uses it to selectively generate plausible plans for achieving it
- This goal-directed reasoning or planning can eliminate combinatorial search altogether
- How to combine these approaches is an unsolved question
- A fully integrated system would be a significant achievement for AI research in general

5.9 Summary

- A game can be defined by
  - the initial state,
  - the legal actions in each state,
  - a terminal test,
  - and a utility function that applies to terminal states.
- In two-player zero-sum games with perfect information, minimax can select the optimal move via a depth-first search of the game tree
- Alpha-beta search computes the same optimal moves, but achieves much greater efficiency

5.9 Summary

- Usually, it is not feasible to consider the whole game tree (even with alpha-beta pruning)
- We need to cut off evaluation and apply a heuristic evaluation function to states
- Precomputed opening and endgame tables are often used in game programs
- Games of chance can be handled by extending the minimax algorithm to include chance nodes
- Optimal play in games of imperfect information requires reasoning about current and future belief states

5.9 Summary

- Computer programs have bested champion human players at games such as **chess**, **checkers** and **Othello,** and in **Go (2016).**
- Humans retain the edge in several games of imperfect information, such as **poker** and **bridge,** and in modern computer games with many move options and very strategic play.
- But even this might change soon, as DeepMind is trying to learn to play StarCraft II (Blizzard) with reinforcement learning and deep neural networks.