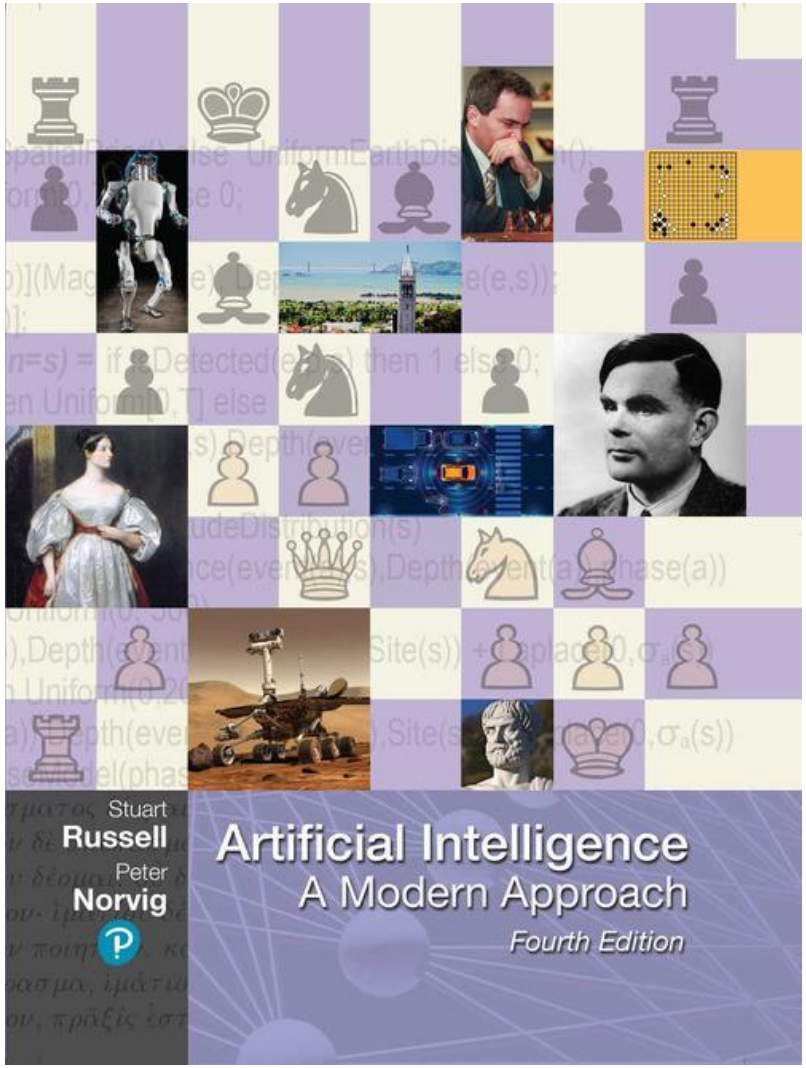
Artificial Intelligence: A Modern Approach

Fourth Edition



Chapter 5

Adversarial Search And Games

Outline

- Game Theory
- Optimal Decisions in Games
  - minimax decisions
  - α–β pruning
  - Monte Carlo Tree Search (MCTS)
- Resource limits and approximate evaluation
- Games of chance
- Games of imperfect information
- Limitations of Game Search Algorithms

Games Theory

Two players
- Max-min
- Taking turns, fully observable

Moves: Action

Position: state

Zero sum:
- good for one player, bad for another
- No win-win outcome.

Games Theory

S_0: The initial state of the game

TO-MOVE(s): player to move in state s.

ACTIONS(s): The set of legal moves in state s.

RESULT(s, a): The transition model, resulting state

IS-TERMINAL(s): A terminal test to detect when the game is over

UTILITY(s; p): A utility function (objective/payoff)

Games vs. search problems

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

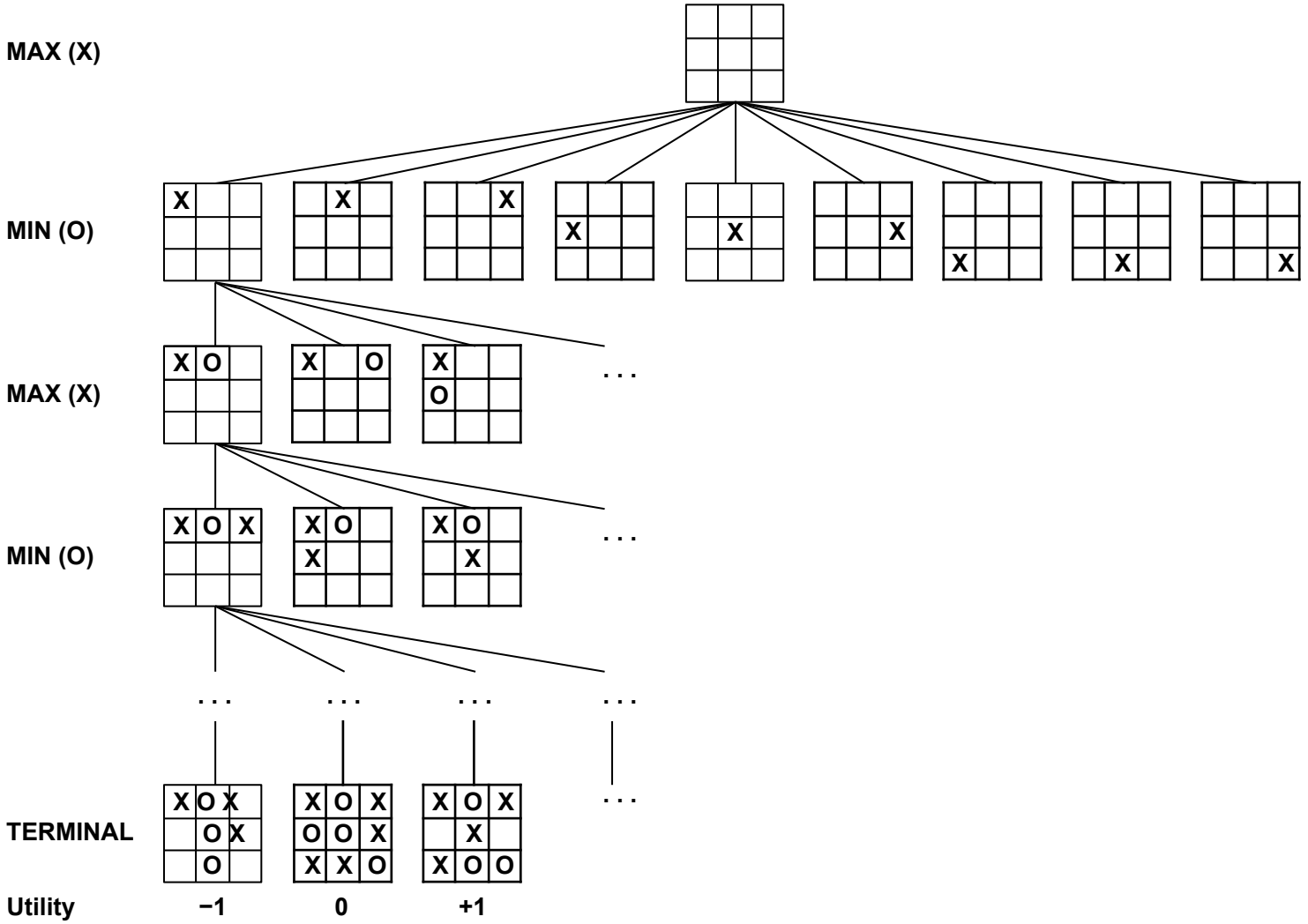
Time limits ⇒ unlikely to find goal, must approximate Plan of attack:

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

Types of games

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

Game tree (2-player, deterministic, turns)

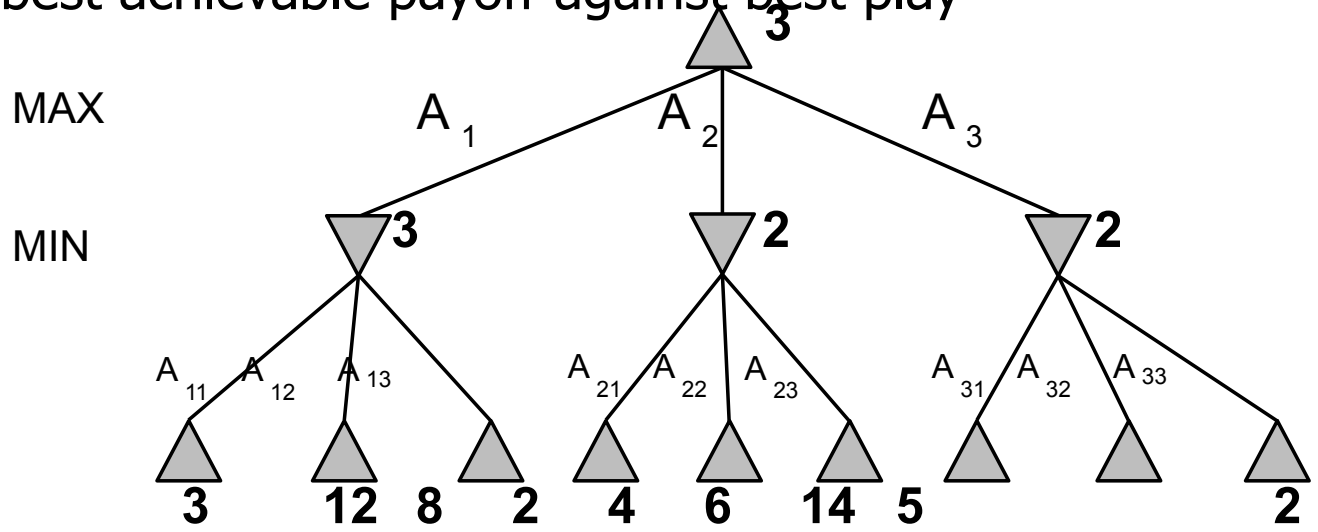


Minimax

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 ← −∞
    for a, s in Successors(state) do v ← Max(v, Min-Value(s))
    return v

---

function Min-Value(state) returns a utility value
    if Terminal-Test(state) then return Utility(state)
    v ← ∞
    for a, s in Successors(state) do v ← Min(v, Max-Value(s))
    return v
```

Properties of minimax

Complete?
?

Properties of minimax

Complete?? Only if tree is finite (chess has specific rules for this).

NB a finite strategy can exist even in an infinite tree!

Optimal??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)
Optimal?? Yes, against an optimal opponent.

Otherwise?? Time complexity??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)
Optimal?? Yes, against an optimal opponent.

Otherwise?? Time complexity?? $O(b^m)$

Space complexity??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)
Optimal?? Yes, against an optimal opponent.

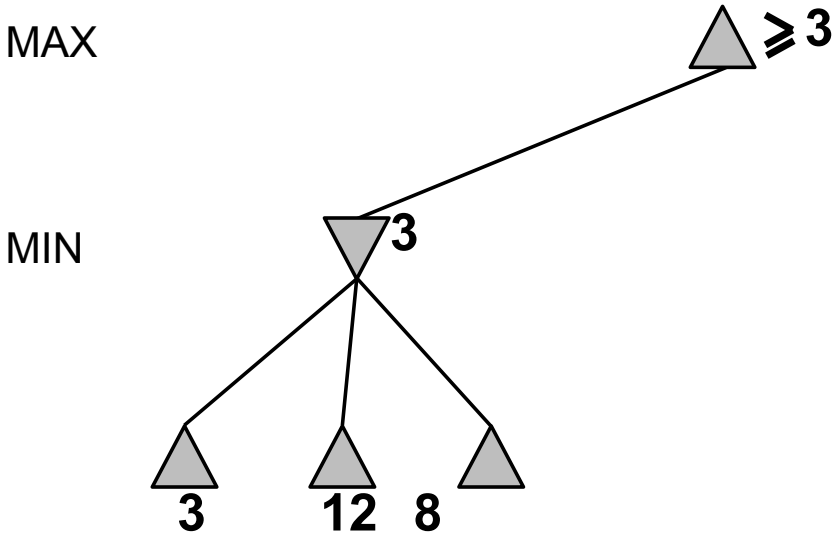
Otherwise?? Time complexity?? $O(b^m)$

Space complexity?? $O(bm)$ (depth-first exploration) For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games

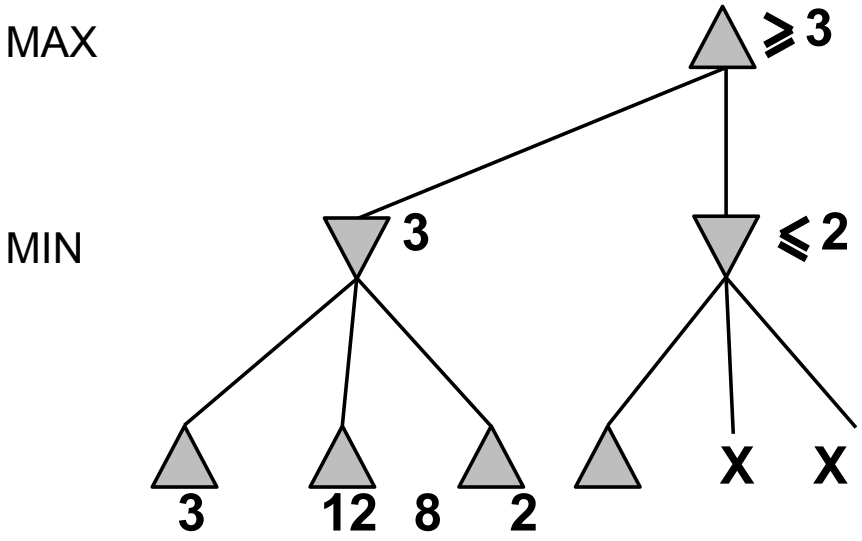
⇒ exact solution completely infeasible

But do we need to explore every path?

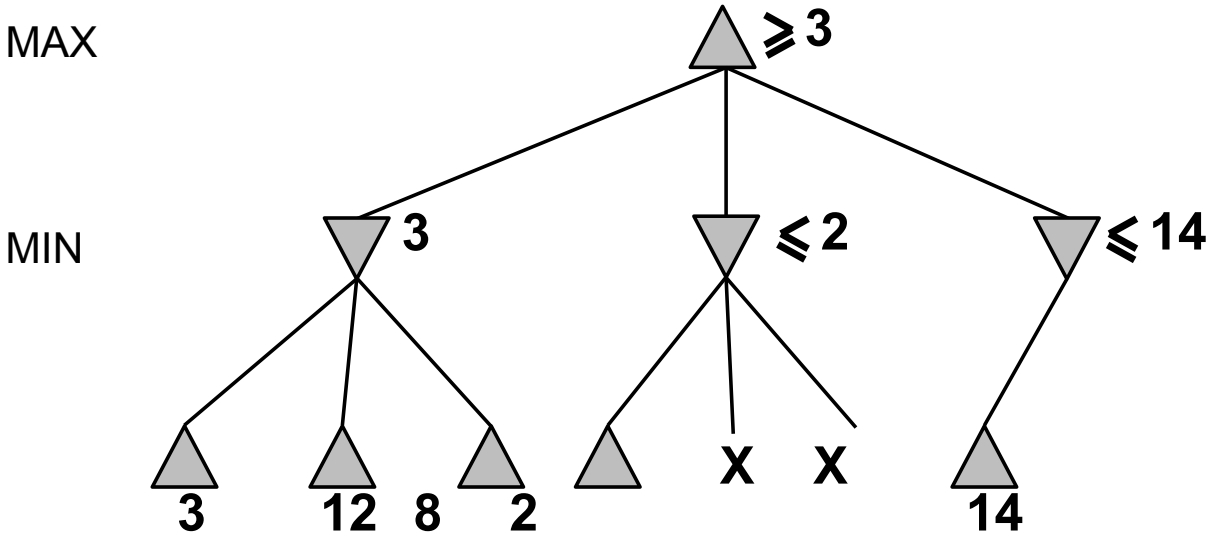
α–β pruning example



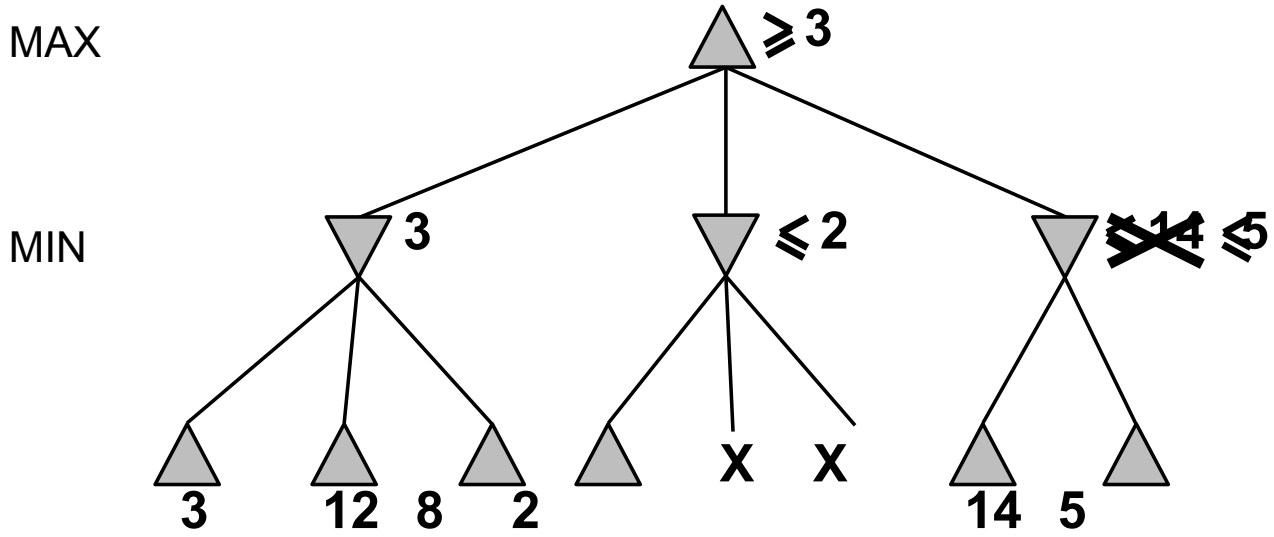
α–β pruning example



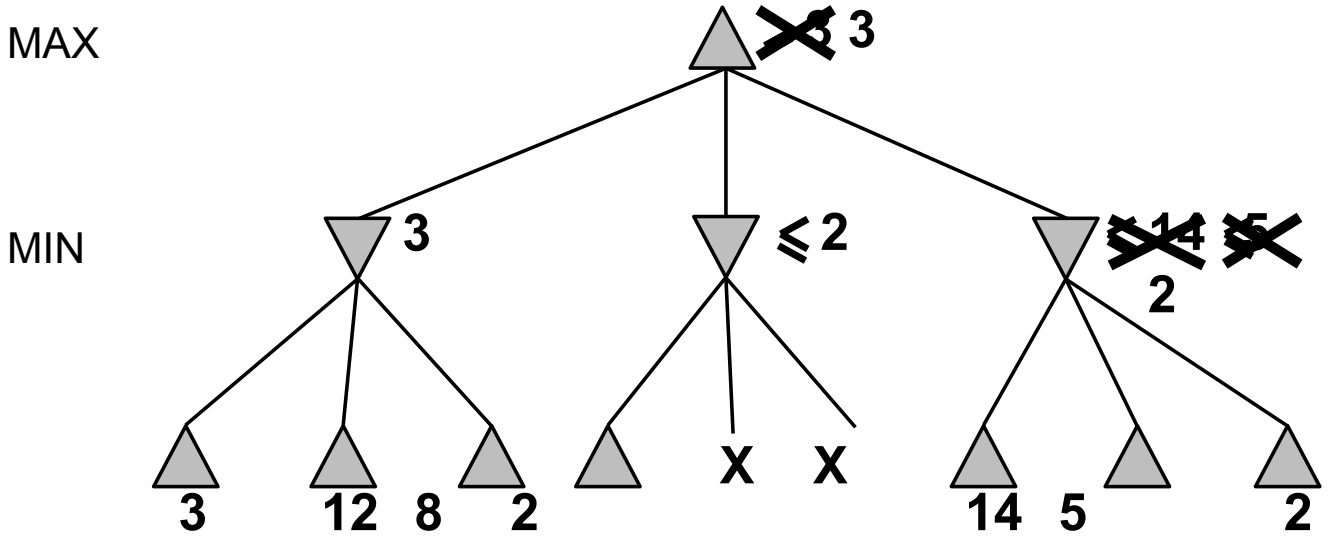
α–β pruning example



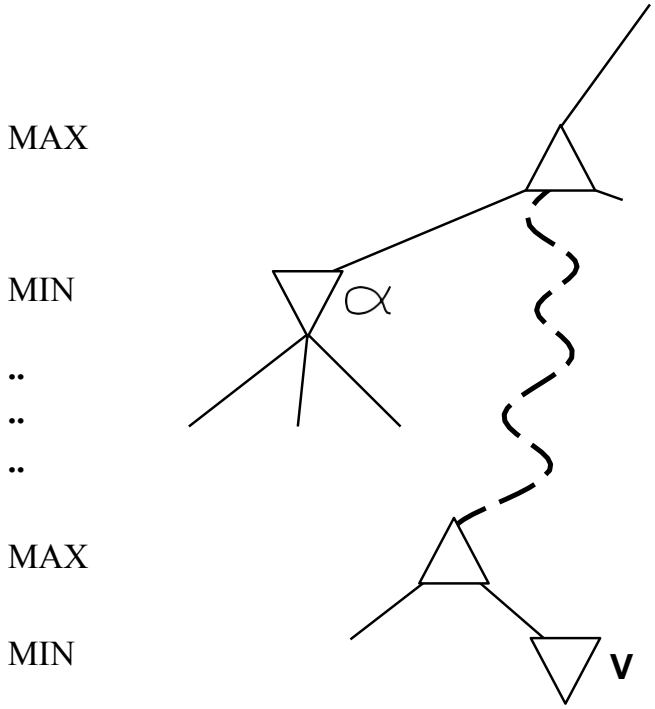
α–β pruning example



α–β pruning example



Why is it called α–β?



α is the best value (to $\max$) found so far off the current path

If V is worse than α, $\max$ will avoid it ⇒ prune that branch Define β similarly for $\min$

The α–β 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, α, β) 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 ← −∞
    for a, s in Successors(state) do
        v ← Max(v, Min-Value(s, α, β))
        if v ≥ β then return v
        α ← Max(α, v)
    return v

---

function Min-Value(state, α, β) returns a utility value
    same as Max-Value but with roles of α, β reversed
```

Properties of α–β

Pruning does not affect final result
Good move ordering improves effectiveness of

pruning With "perfect ordering," time

complexity = $O(b^{m/2})$

⇒ doubles solvable depth

A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

Unfortunately, 35^{50} is still impossible!

Monte Carlo Tree Search

The basic Monte Carlo Tree Search (MCTS) strategy does not use a heuristic evaluation function. Value of a state is estimated as the average utility over number of simulations

- **Playout**: simulation that chooses moves until terminal position reached.
- **Selection**: Start of root, choose move (selection policy) repeated moving down tree
- **Expansion**: Search tree grows by generating a new child of selected node
- **Simulation**: playout from generated child node
- **Back-propagation**: use the result of the simulation to update all the search tree nodes going up to the root

Monte Carlo Tree Search

UCT: Effective selection policy is called “upper confidence bounds applied to trees”

UCT ranks each possible move based on an upper confidence bound formula UCT called UCB1

$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(\text{PARENT}(n))}{N(n)}}$$

where U(n) is the total utility of all playouts that went through node n, N(n) is the number of playouts through node n, and PARENT(n) is the parent node of n in the tree.

Monte Carlo Tree Search

function MONTE-CARLO-TREE-SEARCH(state) **returns** an action
tree ← NODE(state)
while IS-TIME-REMAINING() **do**
leaf ← SELECT(tree)
child ← EXPAND(leaf)
result ← SIMULATE(child)
BACK-PROPAGATE(result, child)
return the move in ACTIONS(state) whose node has highest number of playouts

Monte Carlo Tree Search

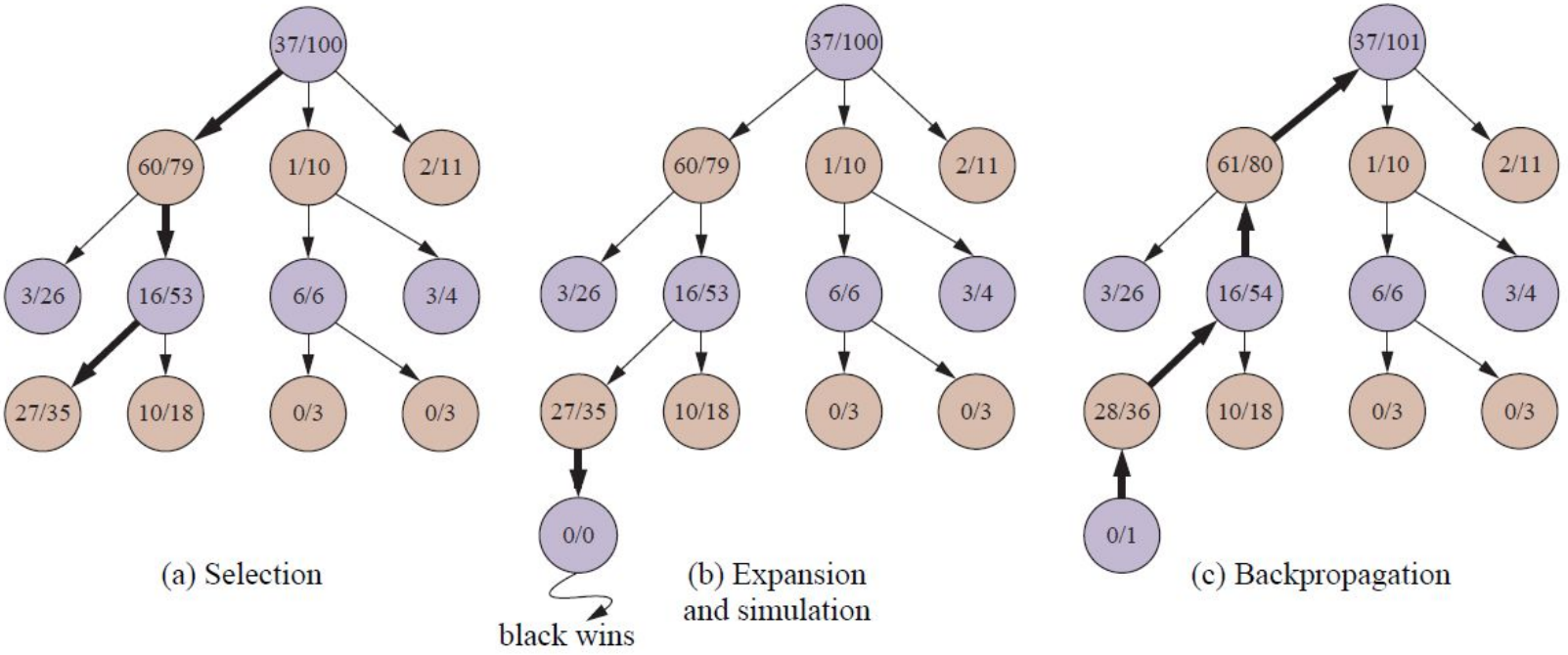


Figure 6.10 One iteration of the process of choosing a move with Monte Carlo tree search (MCTS) using the upper confidence bounds applied to trees (UCT) selection metric, shown after 100 iterations have already been done. In (a) we select moves, all the way down the tree, ending at the leaf node marked 27/35 (for 27 wins for black out of 35 playouts). In (b) we expand the selected node and do a simulation (playout), which ends in a win for black. In (c), the results of the simulation are back-propagated up the tree.

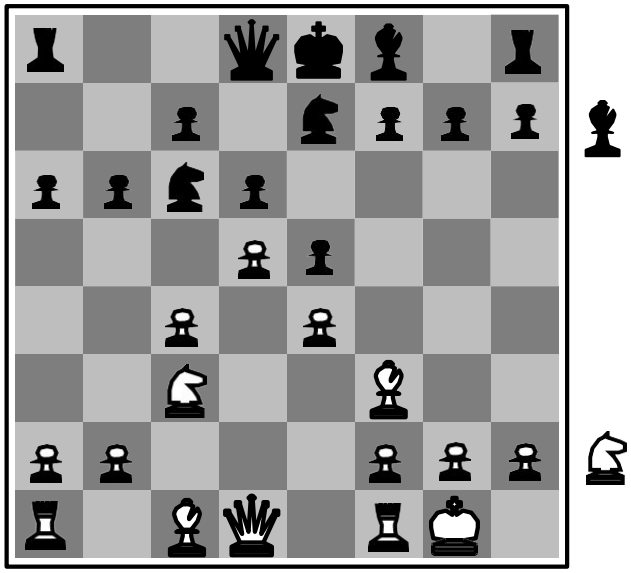
Resource limits

Standard approach:

- Use Cutoff-Test instead of Terminal-Test
  e.g., depth limit (perhaps add quiescence search)
- Use Eval instead of Utility
  i.e., evaluation function that estimates desirability of position

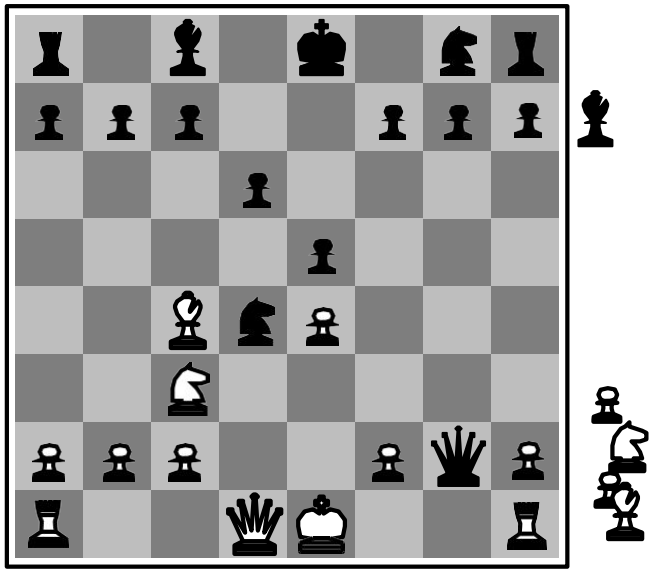
Suppose we have 100 seconds, explore 10^4 nodes/second
⇒ 10^6 nodes per move ≈ $35^{8/2}$
⇒ α–β reaches depth 8 ⇒ pretty good chess program

Evaluation functions



Black to move

White slightly better



White to move

Black winning

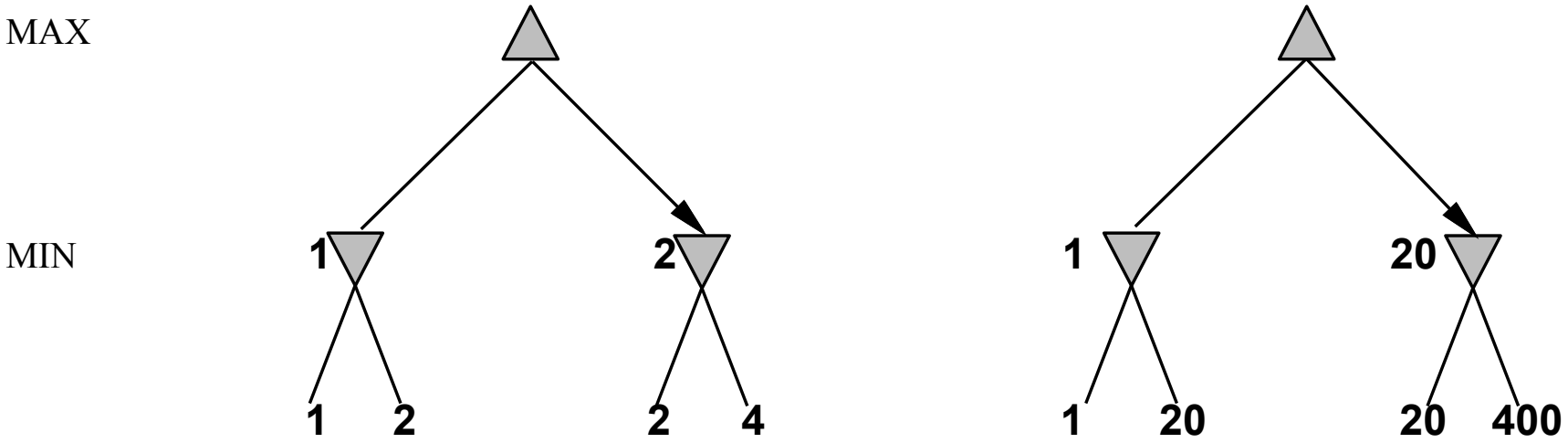
For chess, typically linear weighted sum of features

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

e.g., $w_1 = 9$ with

$f_1(s)$ = (number of white queens) – (number of black queens), etc.

Digression: Exact values don't matter



Behaviour is preserved under any monotonic transformation of Eval

Only the order matters:

payoff in deterministic games acts as an ordinal utility function

Deterministic games in practice

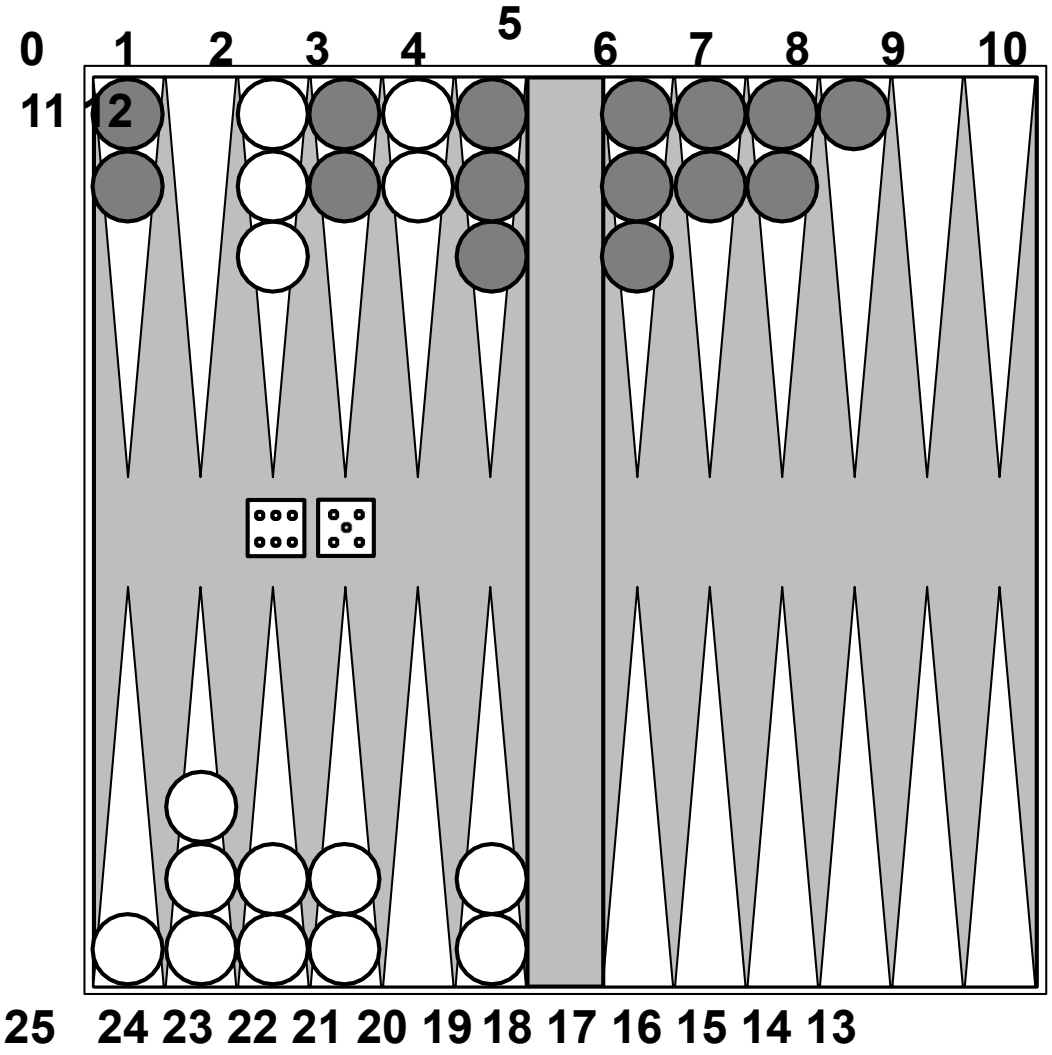
Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.

Chess: Deep Blue defeated human world champion Gary Kasparov in a six- game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

Othello: human champions refuse to compete against computers, who are too good.

Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

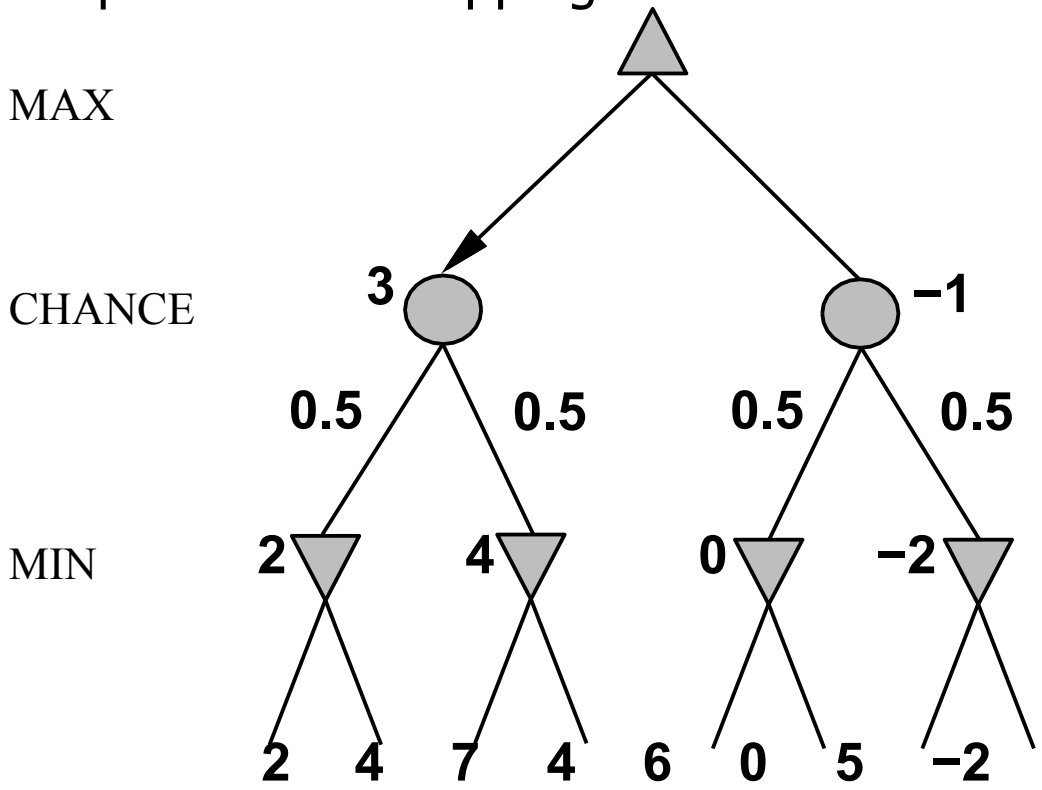
Nondeterministic games: backgammon



Nondeterministic games in general

In nondeterministic games, chance introduced by dice, card-shuffling

Simplified example with coin-flipping:



Algorithm for nondeterministic games

Expectiminimax gives perfect play

Just like Minimax, except we must also handle chance nodes:

```
. . .
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)
. . .
```

Nondeterministic games in practice

Dice rolls increase b: 21 possible rolls with 2 dice
Backgammon ≈ 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

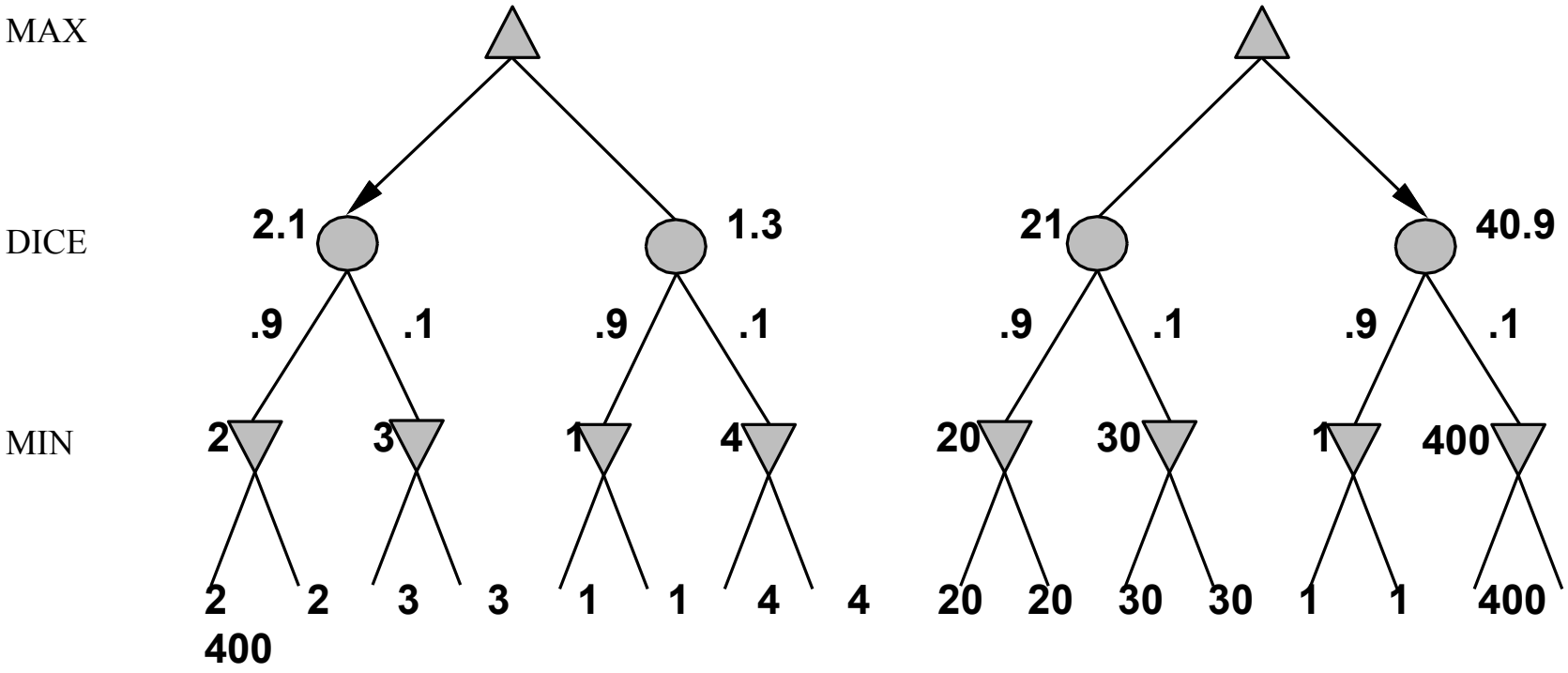
⇒ value of lookahead is diminished

α–β pruning is much less effective

TDGammon uses depth-2 search + very good Eval

≈ world-champion level

Digression: Exact values DO matter



Behaviour is preserved only by positive linear transformation of Eval

Hence Eval should be proportional to the expected payoff

Games of imperfect information

E.g., card games, where opponent's initial cards are unknown

Typically we can calculate a probability for each possible deal
Seems just like having one big dice roll at the beginning of

the game* Idea: compute the minimax value of each action in

each deal,

then choose the action with highest expected value over all deals*

Special case: if an action is optimal for all deals, it's

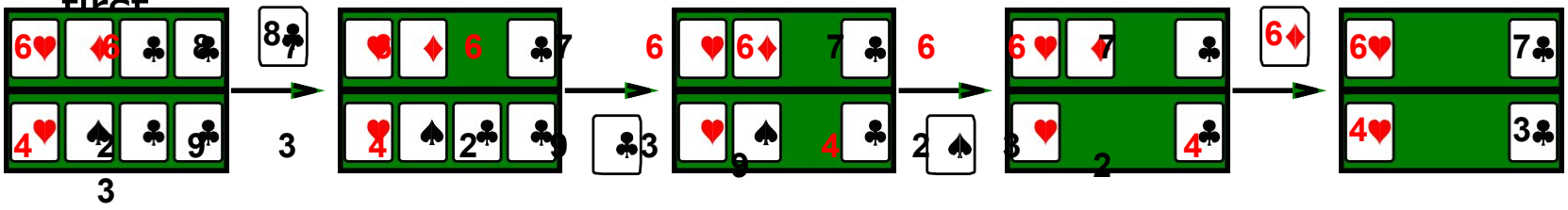
optimal.* GIB, current best bridge program,

approximates this idea by

1) generating 100 deals consistent with bidding information
2) picking the action that wins most tricks on average

Example

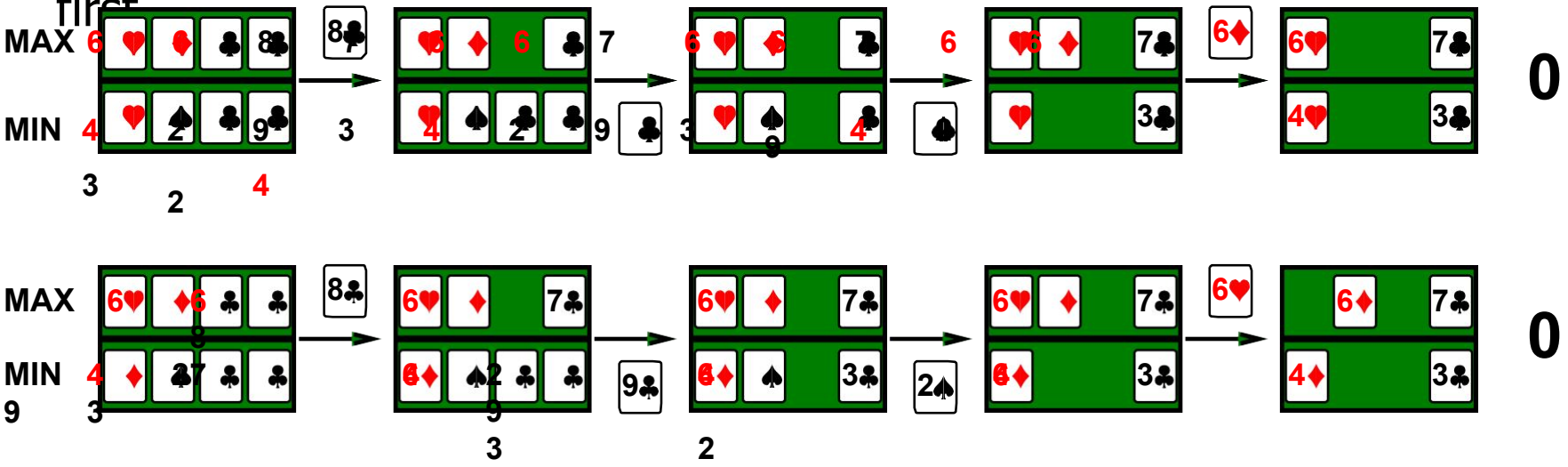
Four-card bridge/whist/hearts hand, Max to play first



0

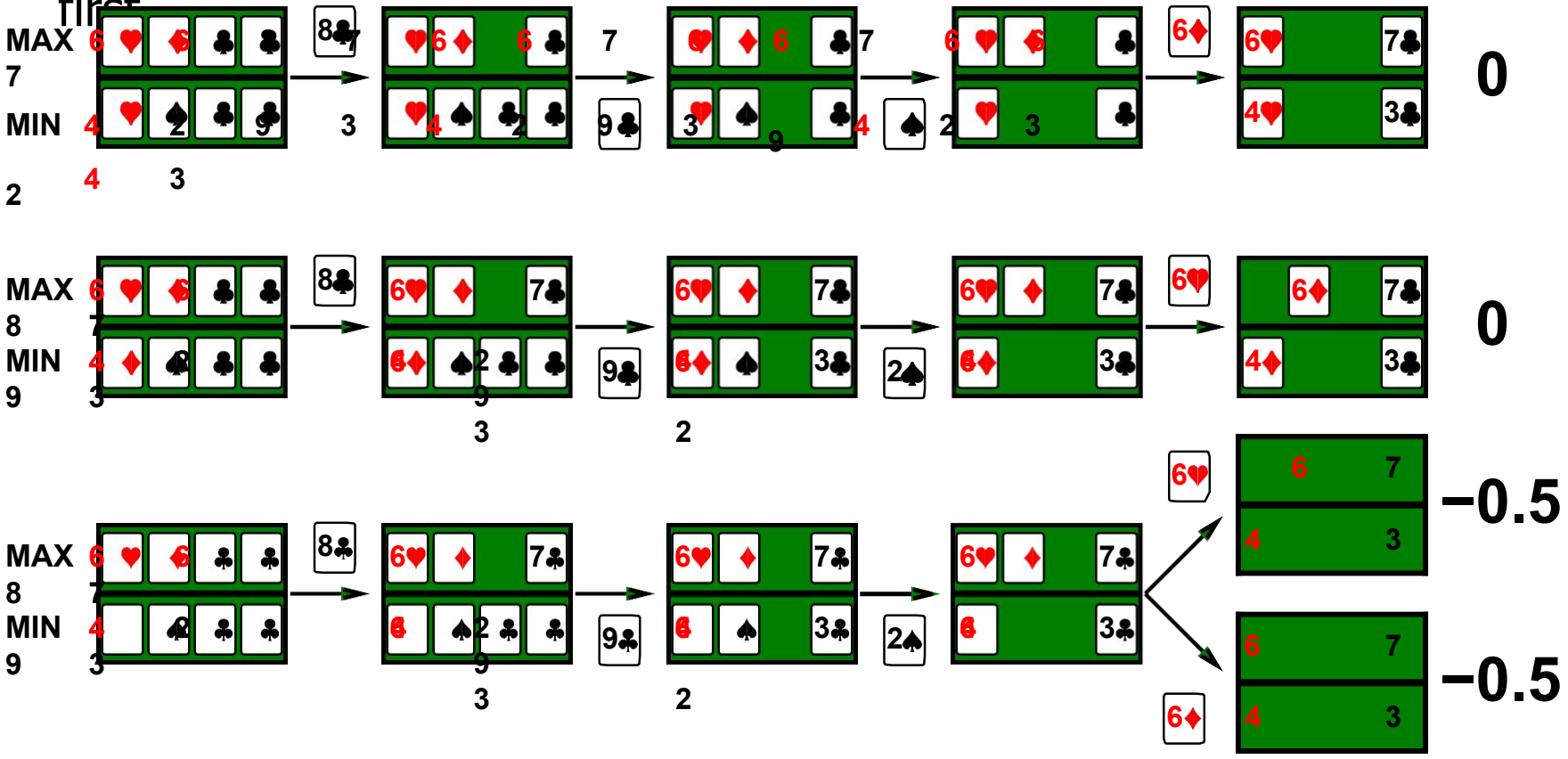
Example

Four-card bridge/whist/hearts hand, Max to play first



Example

Four-card bridge/whist/hearts hand, Max to play first



Common sense example

Road A leads to a small heap of gold pieces Road B leads to a fork:

take the left fork and you'll find a mound of jewels; take the right fork and you'll be run over by a bus.

Common sense example

Road A leads to a small heap of gold pieces Road B leads to a fork:

take the left fork and you'll find a mound of jewels; take the right fork and you'll be run over by a bus.

Road A leads to a small heap of gold pieces Road B leads to a fork:

take the left fork and you'll be run over by a bus; take the right fork and you'll find a mound of jewels.

Common sense example

Road A leads to a small heap of gold pieces Road B leads to a fork:

take the left fork and you'll find a mound of jewels; take the right fork and you'll be run over by a bus.

Road A leads to a small heap of gold pieces Road B leads to a fork:

take the left fork and you'll be run over by a bus; take the right fork and you'll find a mound of jewels.

Road A leads to a small heap of gold pieces Road B leads to a fork:

guess correctly and you'll find a mound of jewels; guess incorrectly and you'll be run over by a bus.

Proper analysis

*Intuition that the value of an action is the average of its values in all actual states is WRONG

With partial observability, value of an action depends on the information state or belief state the agent is in

Can generate and search a tree of information states Leads to rational behaviors such as

- Acting to obtain information
- Signalling to one's partner
- Acting randomly to minimize information disclosure

Limitations of Game Search Algorithms

- Alpha–beta search vulnerable to errors in the heuristic function.
- Waste of computational time for deciding best move where it is obvious (meta-reasoning).
- Reasoning done on individual moves. Humans reason on abstract levels.
- Possibility to incorporate Machine Learning into game search process.

Summary

Minimax algorithm: selects optimal moves by a depth-first enumeration of the game tree.

Alpha–beta algorithm: greater efficiency by eliminating subtrees

Evaluation function: a heuristic that estimates utility of state.

Monte Carlo tree search (MCTS): no heuristic, play game to the end with rules and repeated multiple times to determine optimal moves during playout.