

# GAME PLAYING

## Chapter 6

# Game Playing

- Game playing was thought to be a good problem for AI research:
  - game playing is non-trivial
    - players need “human-like” intelligence
    - games can be very complex (e.g., Chess, Go)
    - requires decision making within limited time
  - games usually are:
    - well-defined and repeatable
    - fully observable and limited environments
  - can directly compare humans and computers

# Computers Playing Chess



**NEW**

# Computers Playing Go



# Outline

- ◆ Games
- ◆ Perfect play (最优策略)
  - minimax decisions
  - $\alpha$ - $\beta$  pruning (剪枝)
- ◆ Resource limits and approximate evaluation
- ◆ Games of chance (包含几率因素的游戏)
- ◆ Games of imperfect information

# Games vs. search problems

“Unpredictable” opponent (不可预测的对手)  $\Rightarrow$  solution is a **strategy** specifying a move for every possible opponent reply

Time limits  $\Rightarrow$  unlikely to find goal, must approximate  
游戏对于低效率有严厉的惩罚

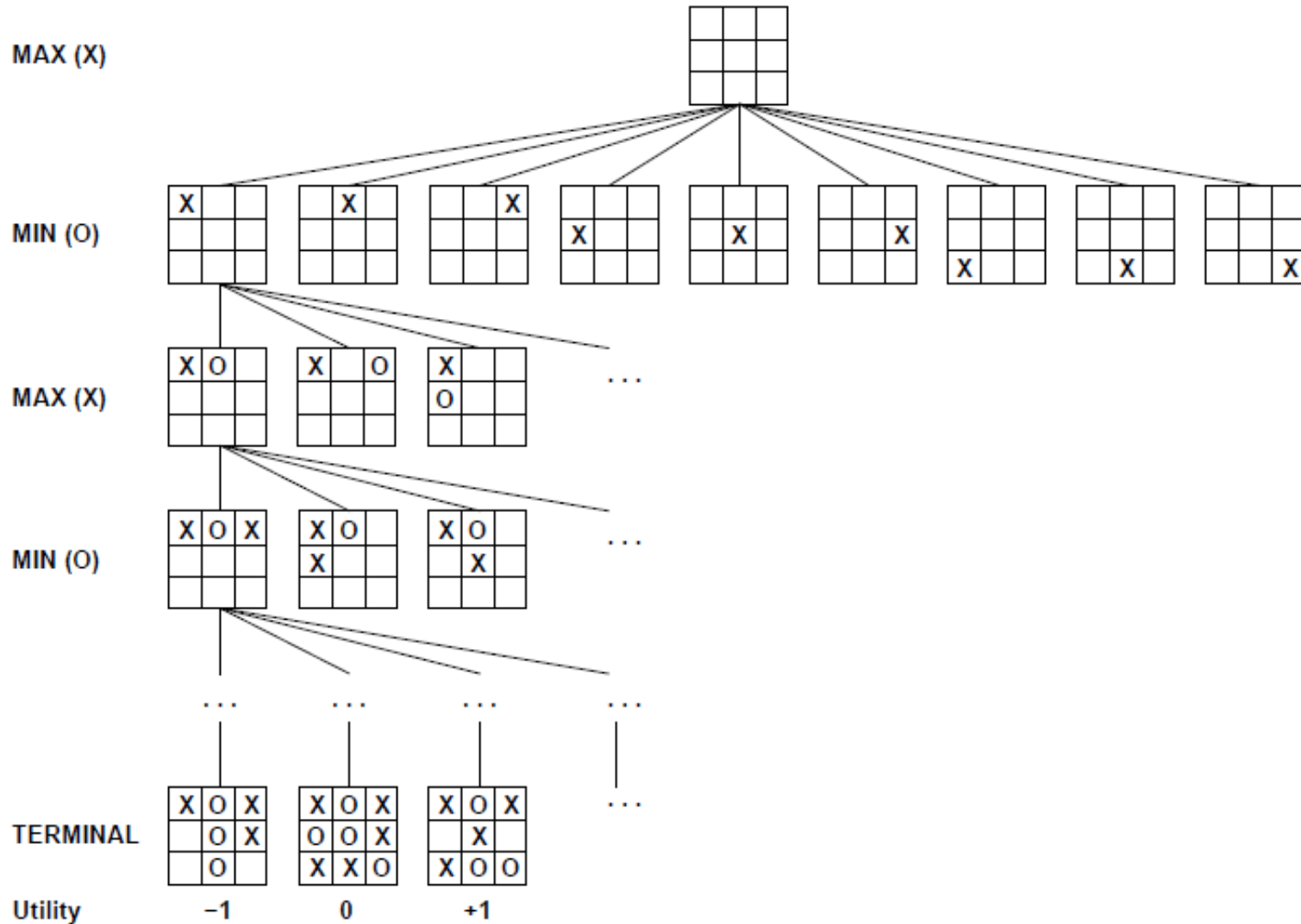
Plan of attack:

- 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 (剪枝) to allow deeper search (McCarthy, 1956)

# Types of games

|                       | Deterministic                           | Stochastic (chance)                           |
|-----------------------|---|---|
| perfect information   | chess, checkers,<br>Go (围棋),<br>othello | Backgammon (西洋双陆棋)<br>monopoly                |
| imperfect information | battleships,<br>blind tictactoe         | bridge, poker, scrabble (拼字游戏)<br>nuclear war |

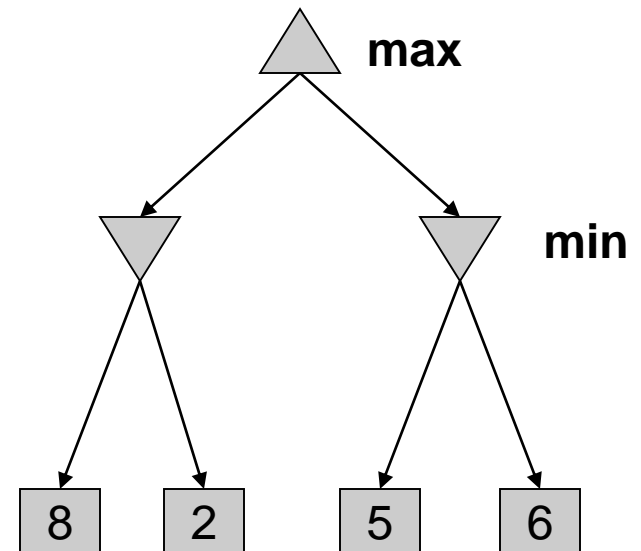
# Game tree (2-player, deterministic, turns)





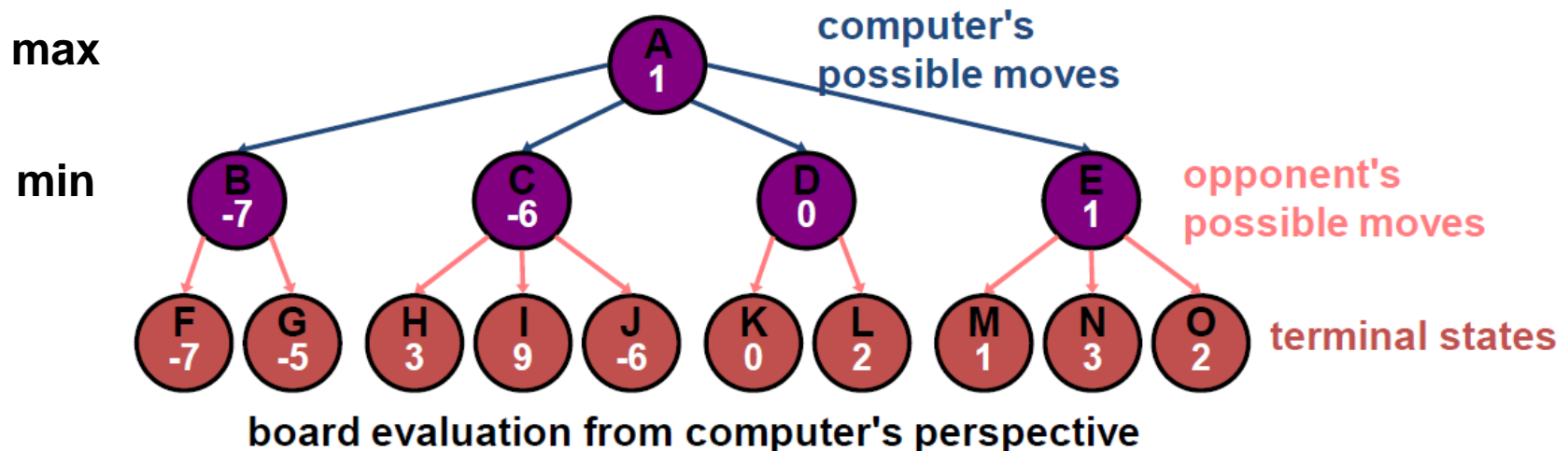
# Deterministic Two-Player

- E.g. tic-tac-toe, chess, checkers
- **Game search**
  - A state-space search tree
  - Players alternate
  - Each layer, or ply, consists of a round of moves
  - Choose move to position with highest achievable utility
- Zero-sum games
  - One player maximizes result
  - The other minimizes result



# Minimax Principle

- Assume *both* players play optimally
  - The computer assumes after it moves the opponent will choose the minimizing move
  - The computer chooses the best move considering both its move and the opponent's optimal move



# Minimax

Perfect play (最优策略) for deterministic, perfect-information games

Idea: choose move to position with highest **minimax value**

= best achievable payoff against best play

在对手 *也使用最优策略* 的条件下, 能导致至少不比其它策略差的结果

假设两个游戏者都按照最优策略进行, 那么节点的极小极大值就是对应状态的效用值 (对于MAX)

- MAX优先选择有极大值的状态
- MIN优先选择有极小值的状态

MINMAX - VALUE( $n$ ) =

$$\left\{ \begin{array}{ll} \text{UTILITY}(n) & \text{当 } n \text{ 为终止状态} \\ \max_{s \in \text{Successors}(n)} \text{MINMAX - VALUE}(s) & \text{当 } n \text{ 为MAX节点} \\ \min_{s \in \text{Successors}(n)} \text{MINMAX - VALUE}(s) & \text{当 } n \text{ 为MIN节点} \end{array} \right.$$

# Minimax

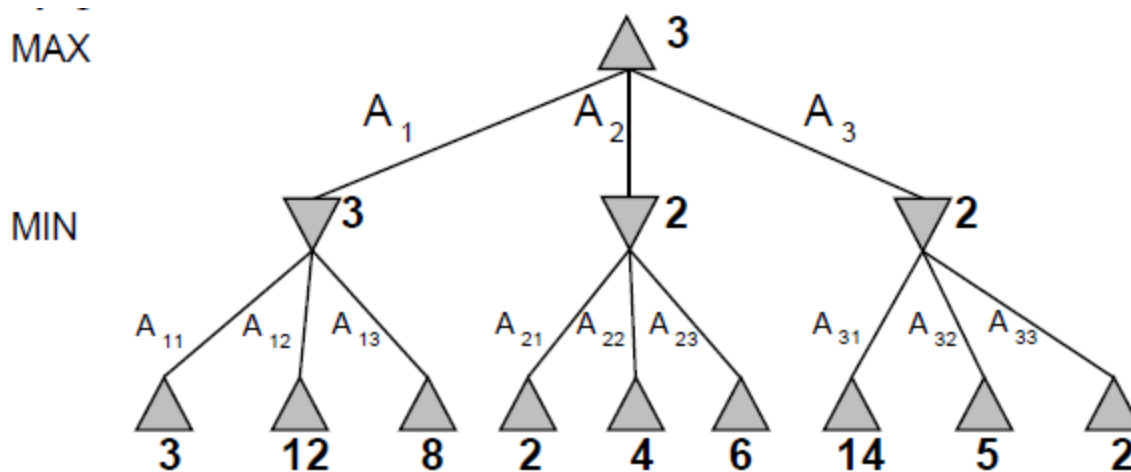
Perfect play (最优策略) for deterministic, perfect-information games

Idea: choose move to position with highest **minimax value**

= best achievable payoff against best play

在对手也使用最优策略的条件下，能导致至少不比其它策略差的结果

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  $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$

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  $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$

return *v*

# Properties of minimax

Complete??

# Properties of minimax

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

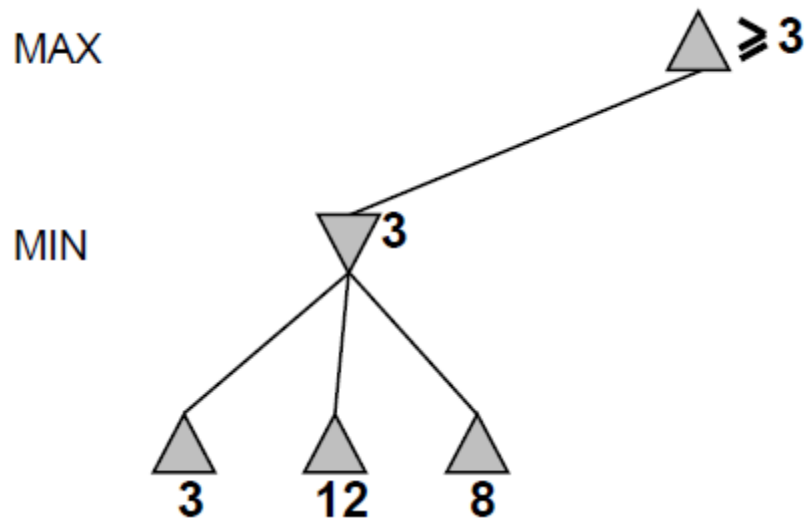
# $\alpha - \beta$ Pruning

- Some of the branches of the game tree won't be taken if playing against an intelligent opponent
- “If you have an idea that is surely bad, don't take the time to see how truly awful it is.”

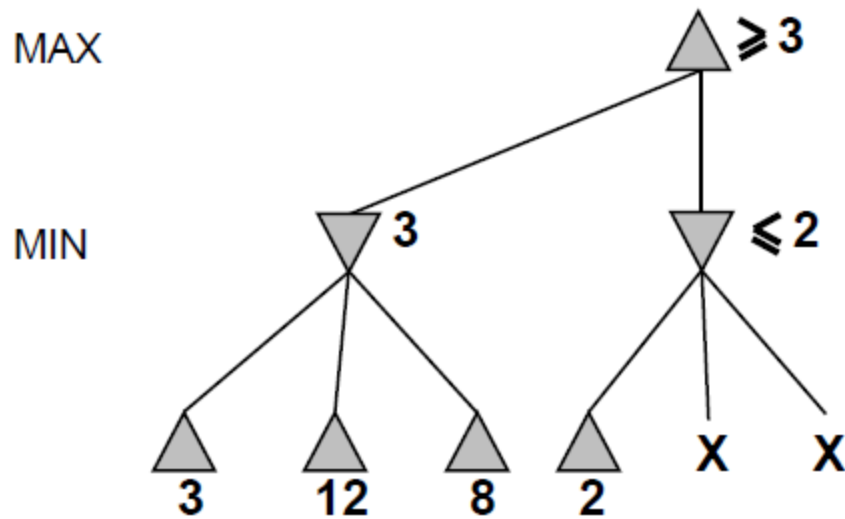
-- Pat Winston

- Pruning can be used to ignore some branches

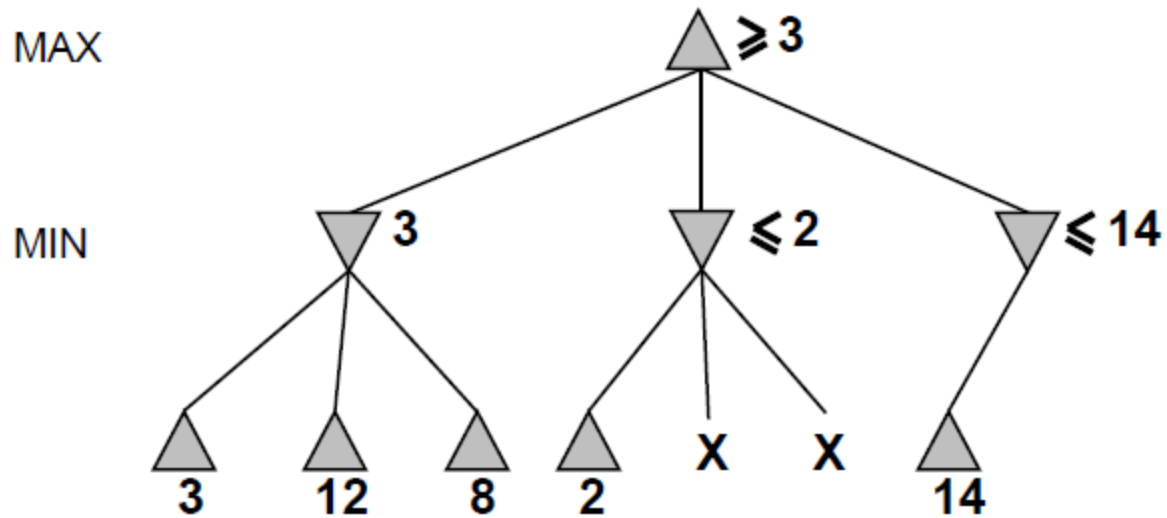
# $\alpha - \beta$ pruning example



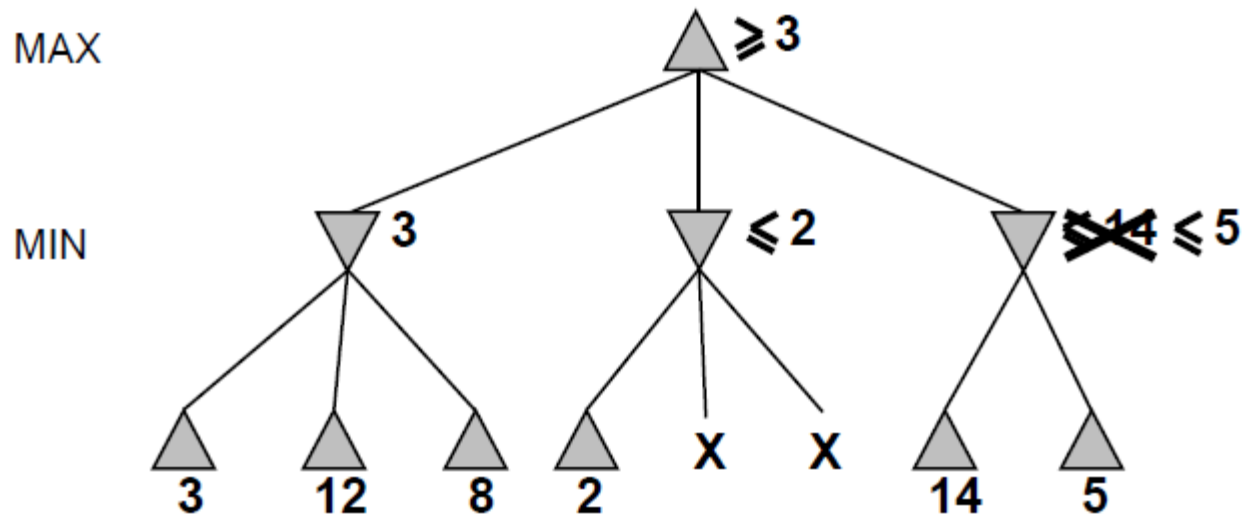
# $\alpha - \beta$ pruning example



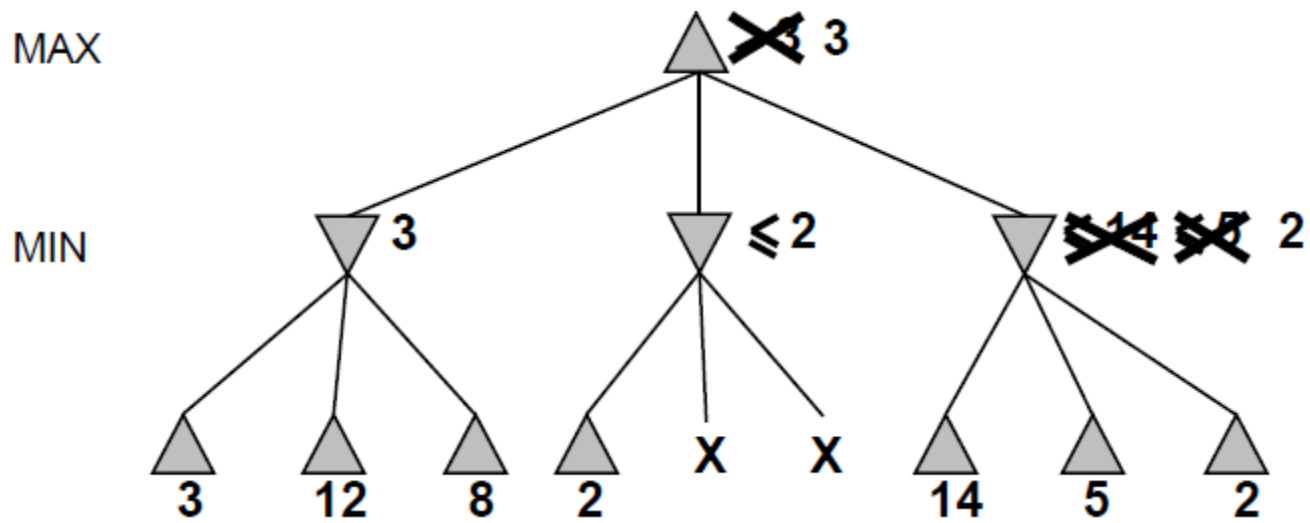
# $\alpha - \beta$ pruning example



# $\alpha - \beta$ pruning example

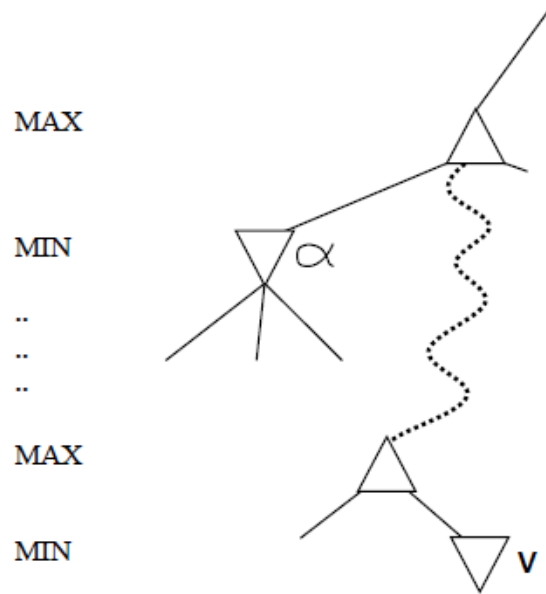


# $\alpha - \beta$ pruning example





# Why is it called $\alpha - \beta$



- $\alpha$  is the best value (to MAX) found so far on the current path  
到目前为止在路径上的任意选择点发现的MAX的最佳（即最大值）选择
- If  $v$  is worse than  $\alpha$ , MAX will avoid it, so can stop considering  $v$ 's other children  $\rightarrow$  prune that branch
- Define  $\beta$  similarly for MIN

# The $\alpha - \beta$ 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
          $\alpha$ , the value of the best alternative for MAX along the path to state
          $\beta$ , 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
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$ 
    if  $v \geq \beta$  then return v
     $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
  return v
```

---

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

# Effectiveness of $\alpha - \beta$ Search

- Effectiveness depends on the order in which successors are examined; more effective if best successors are examined first
- Worst Case:
  - ordered so that no pruning takes place
  - no improvement over exhaustive search
- Best Case:
  - each player's best move is evaluated first
- In practice, performance is closer to best, rather than worst, case

# Properties of $\alpha - \beta$

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!

# Outline

- ◆ Games
- ◆ Perfect play (最优策略)
  - minimax decisions
  - $\alpha$ - $\beta$  pruning (剪枝)
- ◆ Resource limits and approximate evaluation
- ◆ Games of chance (包含几率因素的游戏)
- ◆ Games of imperfect information

# Resource limits

Standard approach: Depth-limited search

- Use CUTOFF-TEST (截断测试) instead of TERMINAL-TEST (终止测试)  
e.g., depth limit (perhaps add quiescence search 静态搜索)
- Use EVAL instead of UTILITY  
用可以估计棋局效用值的启发式评价函数EVAL取代效用函数  
i.e., evaluation function that estimates desirability of position

Suppose we have 100 seconds, explore  $10^4$  nodes/second

$$\Rightarrow 10^6 \text{ nodes per move} \approx 35^{8/2}$$

$$\Rightarrow \alpha - \beta \text{ reaches depth } 8 \Rightarrow \text{pretty good chess program}$$

4-ply lookahead is a hopeless chess player!

- 4-ply  $\approx$  human novice
- 8-ply  $\approx$  typical PC, human master
- 12-ply  $\approx$  Deep Blue, Kasparov

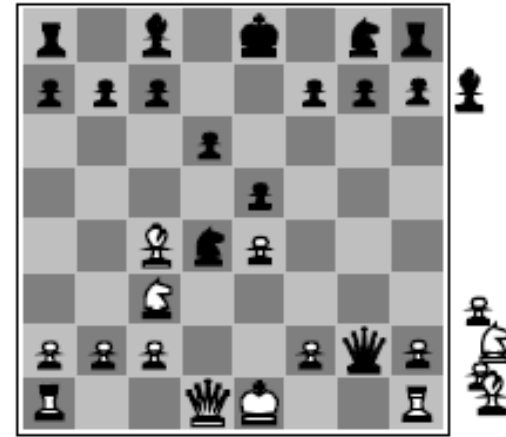
# Evaluation functions

- Function which scores non-terminals



Black to move

White slightly better



White to move

Black winning

- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of **features** (特征) :

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

e.g., for chess,  $w_1 = 9$  with

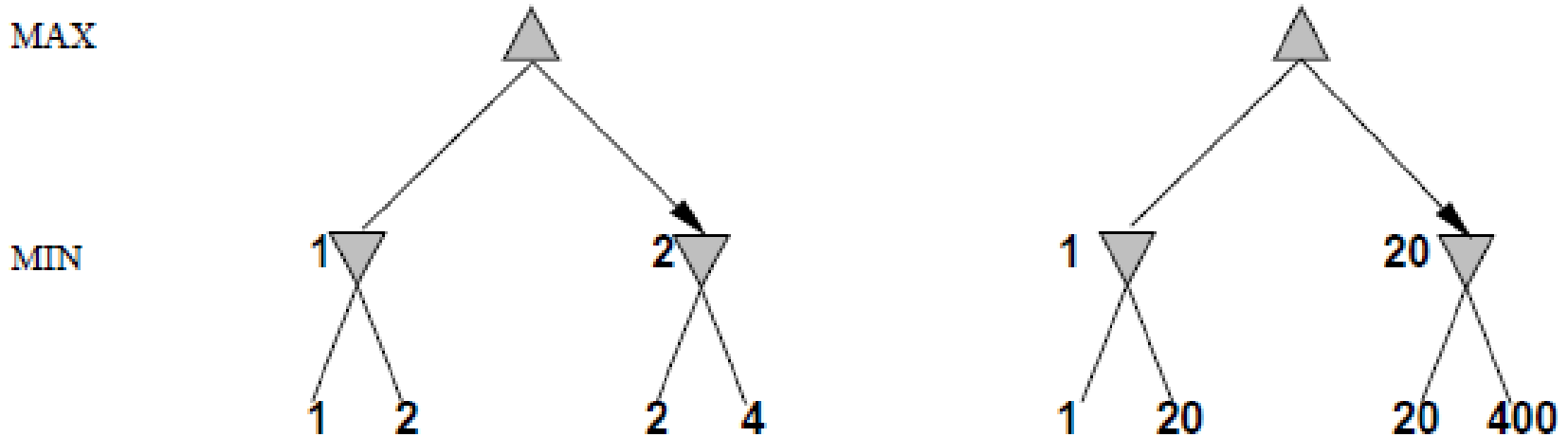
$f_1(s) = (\text{number of white queens}) - (\text{number of black queens}), \text{ etc.}$

# More on Evaluation Functions

- The board evaluation function estimates how good the current board configuration is
- A linear evaluation function of the features is a weighted sum of  $f_1$ ,  $f_2$ ,  $f_3$ ,  
...
  - More important features get more weight
- The quality of play depends directly on the quality of the evaluation function
- To build an evaluation function we have to:
  - construct good features using expert domain knowledge
  - pick or learn good weights



# Digression: Exact values don't matter



Behavior is preserved under any **monotonic** (单调的) transformation of EVAL

Only the order matters:

payoff (结果) in deterministic games acts as an **ordinal utility** (序数效用) function

# Dealing with Limited Time

- In real games, there is usually a time limit  $T$  on making a move
- How do we take this into account?
  - cannot stop alpha-beta midway and expect to use results with any confidence
  - so, we could set a conservative depth-limit that guarantees we will find a move in time  $< T$
  - but then, the search may finish early and the opportunity is wasted to do more search

# Dealing with Limited Time

- In practice, **iterative deepening search (IDS)** is used
  - run alpha-beta search with an *increasing depth limit*
  - when the clock runs out, use the solution found for the last completed alpha-beta search (i.e., the deepest search that was completed)

# Deterministic games in practice

Chess (国际象棋) : 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 (层、厚度) .

- 计算机能够预见它的决策中的长期棋局序列。机器拒绝走一步有决定性短期优势的棋—显示了非常类似于人类的对危险的感觉。 ——Kasparov
- Kasparov lost the match 2 wins to 3 wins and 1 tie
- Deep Blue played by “brute force” (i.e., raw power from computer speed and memory); it used relatively little that is similar to human intuition and cleverness
- Used minimax, alpha-beta, sophisticated heuristics

# Deterministic games in practice

Checkers (西洋跳棋) : **Chinook**, the World Man-Machine Checkers Champion.

- Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994.
- In 2007, **checkers was solved**: perfect play leads to a draw.

*Chinook cannot ever lose*

使用了一个提前计算好的存有443,748,401,247个不多于8个棋子的棋局数据库，使它的残局(endgame)走棋没有缺陷

50 machines working in parallel on the problem



# Deterministic games in practice

Othello (奥赛罗) : human champions refuse to compete against computers, who are too good.

Go (围棋) : human champions refuse to compete against computers, who are too bad. In go,  $b > 300$  (棋盘为19x19), so most programs use pattern knowledge bases to suggest plausible moves.

**A new benchmark for Artificial Intelligence (人工智能新的试金石)**

# AlphaGo: First to beat human pro in 19x19 Go

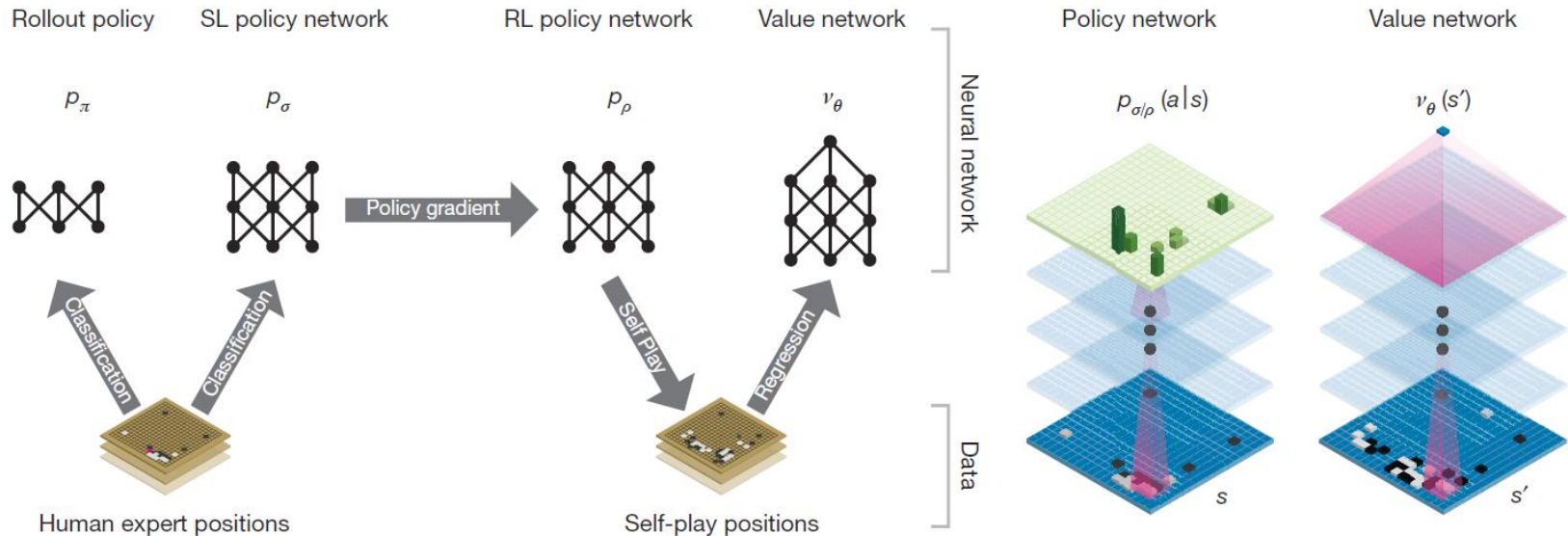
- Google DeepMind computer go player
  - deep neural networks:
    - value networks: to evaluate board positions
    - policy networks: to select moves
  - trained by
    - supervised learning
    - reinforcement learning by self-play
  - search algorithm
    - Monte-Carlo simulation + value/policy networks

# AlphaGo: Background

- reduction of search space:
  - reduced depth
    - position evaluation
  - reduced branching
    - move sampling based on policy
    - policy = probability distribution  $p(a|s)$



# Deep Neural Networks in AlphaGo

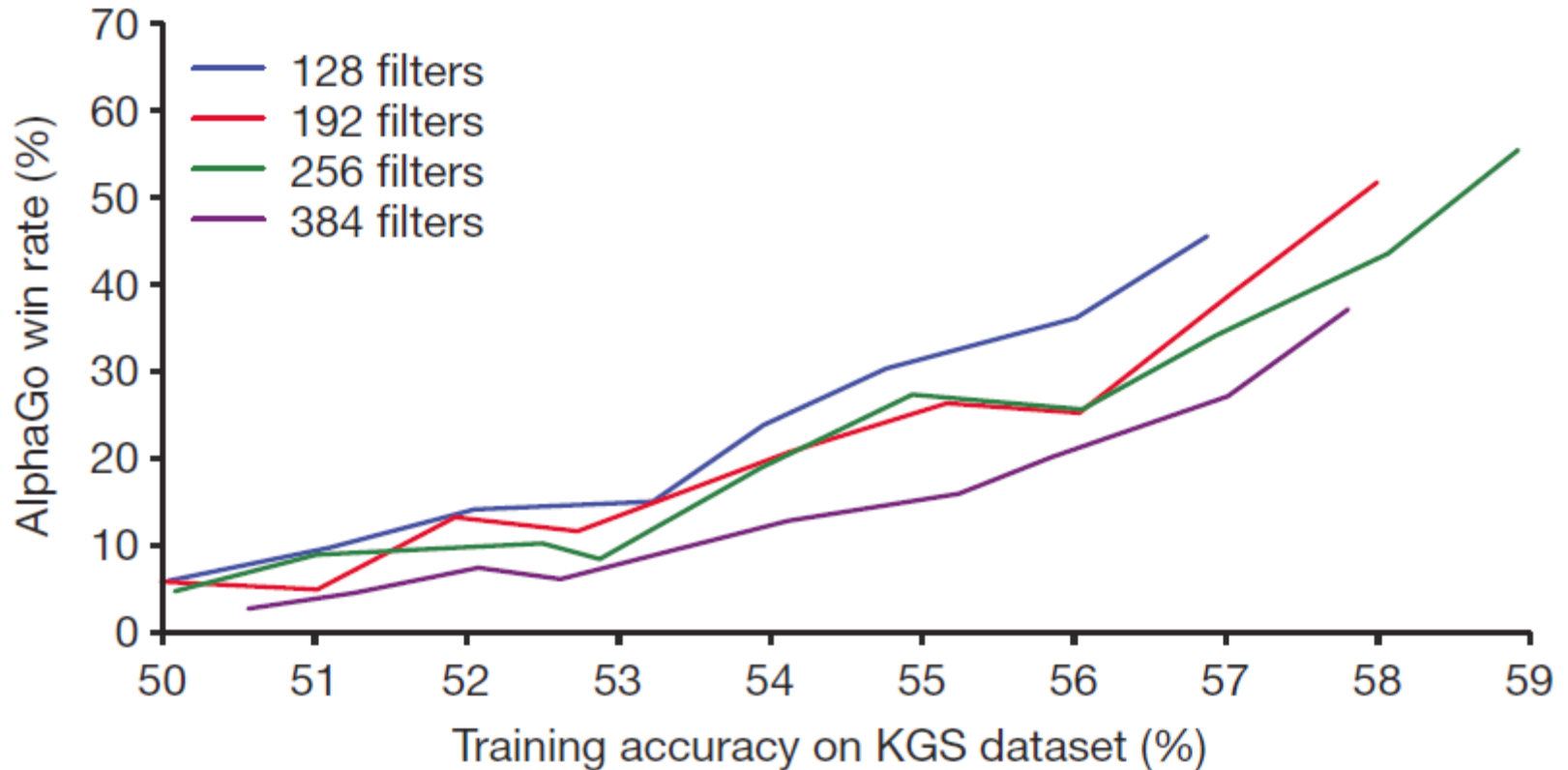


AlphaGo uses two types of neural networks:

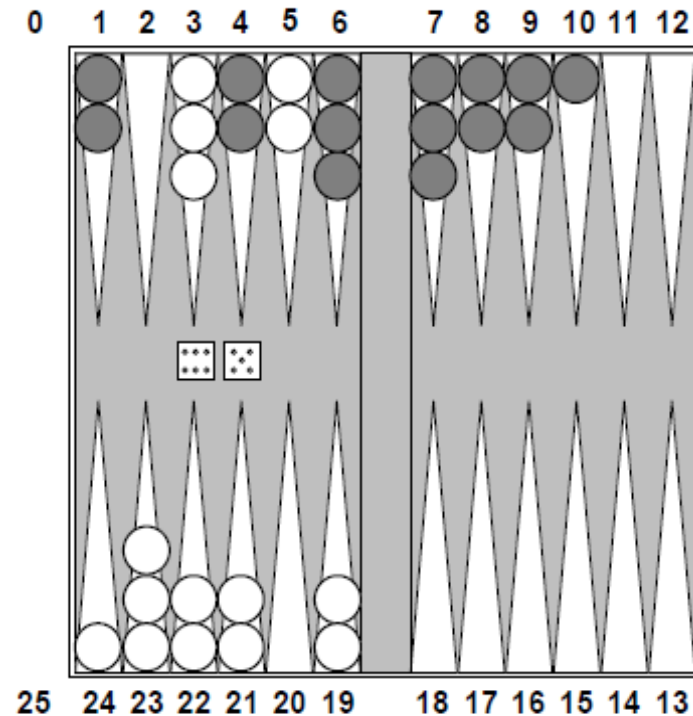
- policy network: what is the next move?
  - learned from human expert moves
- value network: what is the value of a state?
  - learned from self-play using a policy network

SL = supervised learning, RL = reinforcement learning 41

# Deep Neural Networks in AlphaGo

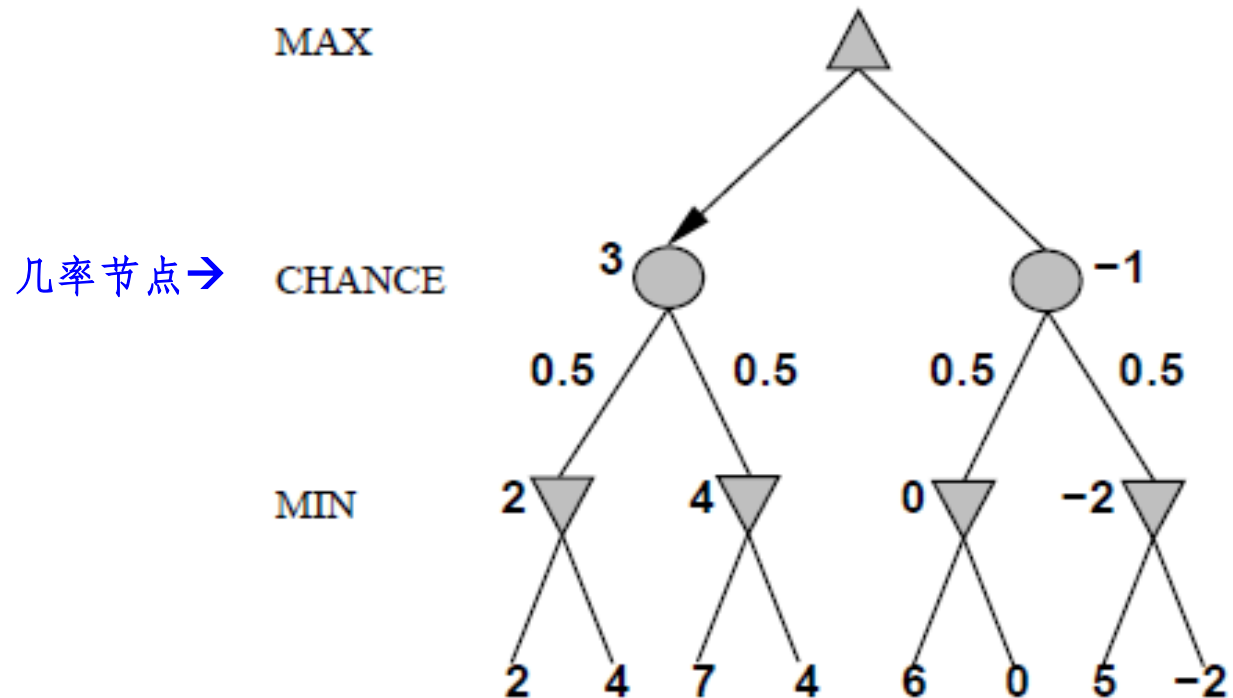


# Nondeterministic games: backgammon(西洋双陆棋)



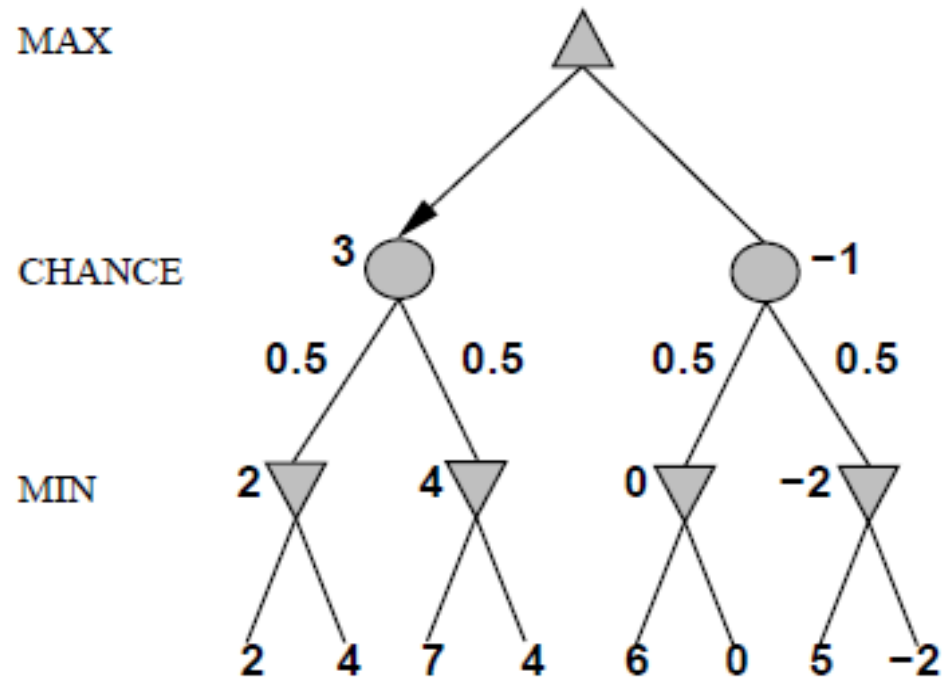
# Nondeterministic games in general

In nondeterministic games, chance introduced by dice, card-shuffling  
Simplified example with coin-flipping:



# Nondeterministic games in general

- Weight score by the *probability* that move occurs
- Use expected value for move: instead of using max or min, compute the average, weighted by the probabilities of each child
- Choose move with *highest expected value*



# Maximum Expected Utility

- Why should we average utilities? Why not minimax?
- Principle of maximum expected utility: an agent should choose the action which **maximizes its expected utility, given its knowledge**
- General principle for decision making
- Often taken as the definition of rationality
- We'll see this idea over and over in this course!

# 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*)

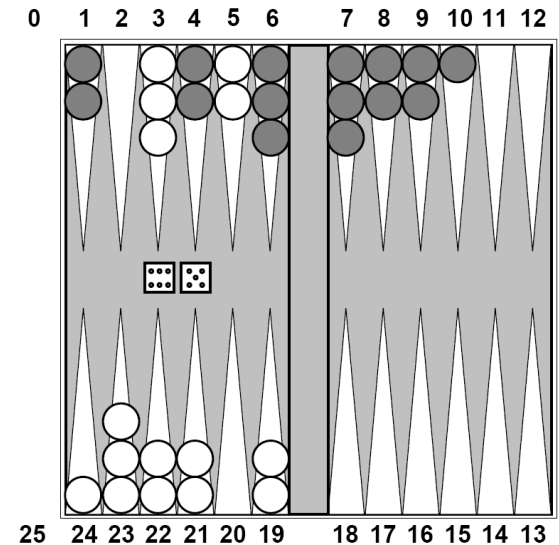
if *state* is a chance node then

    return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

...

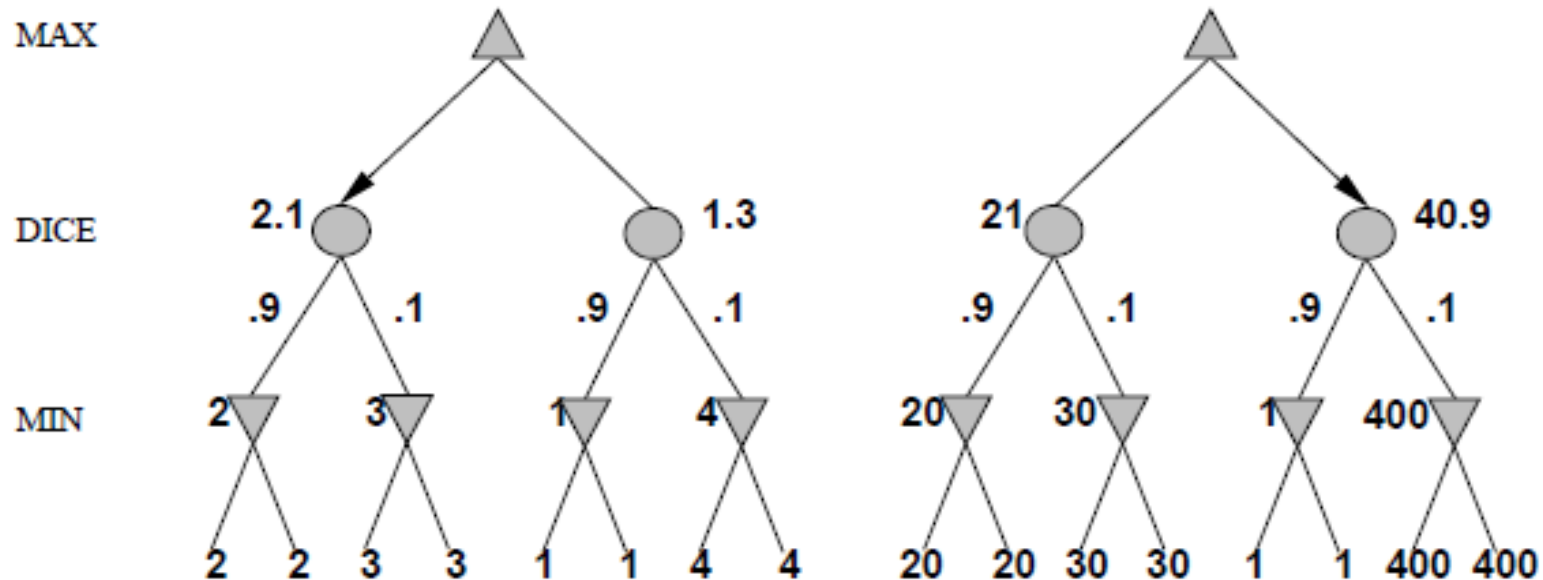
# Stochastic Two-Player

- Dice rolls increase  $b$ : 21 possible rolls with 2 dice
  - Backgammon  $\approx 20$  legal moves
  - Depth 4 =  $20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$
- As depth increases, probability of reaching a given node shrinks
  - So value of lookahead is diminished
  - So limiting depth is less damaging
  - But pruning is less possible...
- TDGammon uses depth-2 search + very good eval function + reinforcement learning: world-champion level play





# Digression: Exact values DO matter



Behaviour is preserved only by **positive linear** transformation of EVAL

Hence EVAL should be proportional to the expected payoff

评价函数应该是棋局的期望效用值的**正线性**变换

# Outline

- ◆ Games
- ◆ Perfect play (最优策略)
  - minimax decisions
  - $\alpha$ - $\beta$  pruning (剪枝)
- ◆ Resource limits and approximate evaluation
- ◆ Games of chance (包含几率因素的游戏)
- ◆ Games of imperfect information

# 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

在评价一个有未知牌的给定行动过程时，首先计算出每副可能牌的出牌行动的极小极大值，然后再用每副牌的概率来计算得到对所有发牌情况的期望值。

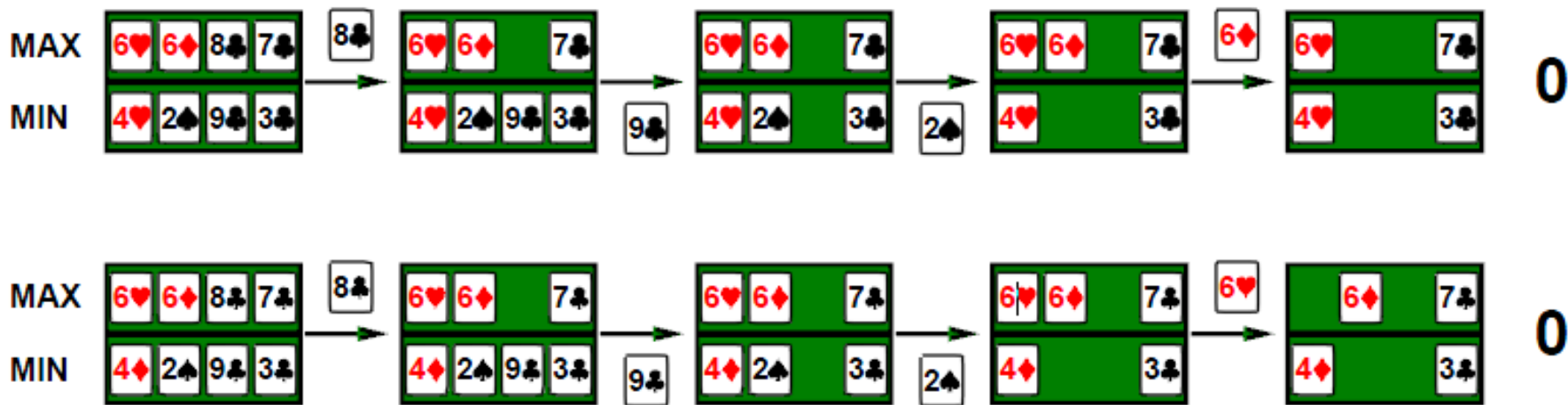
# Example

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



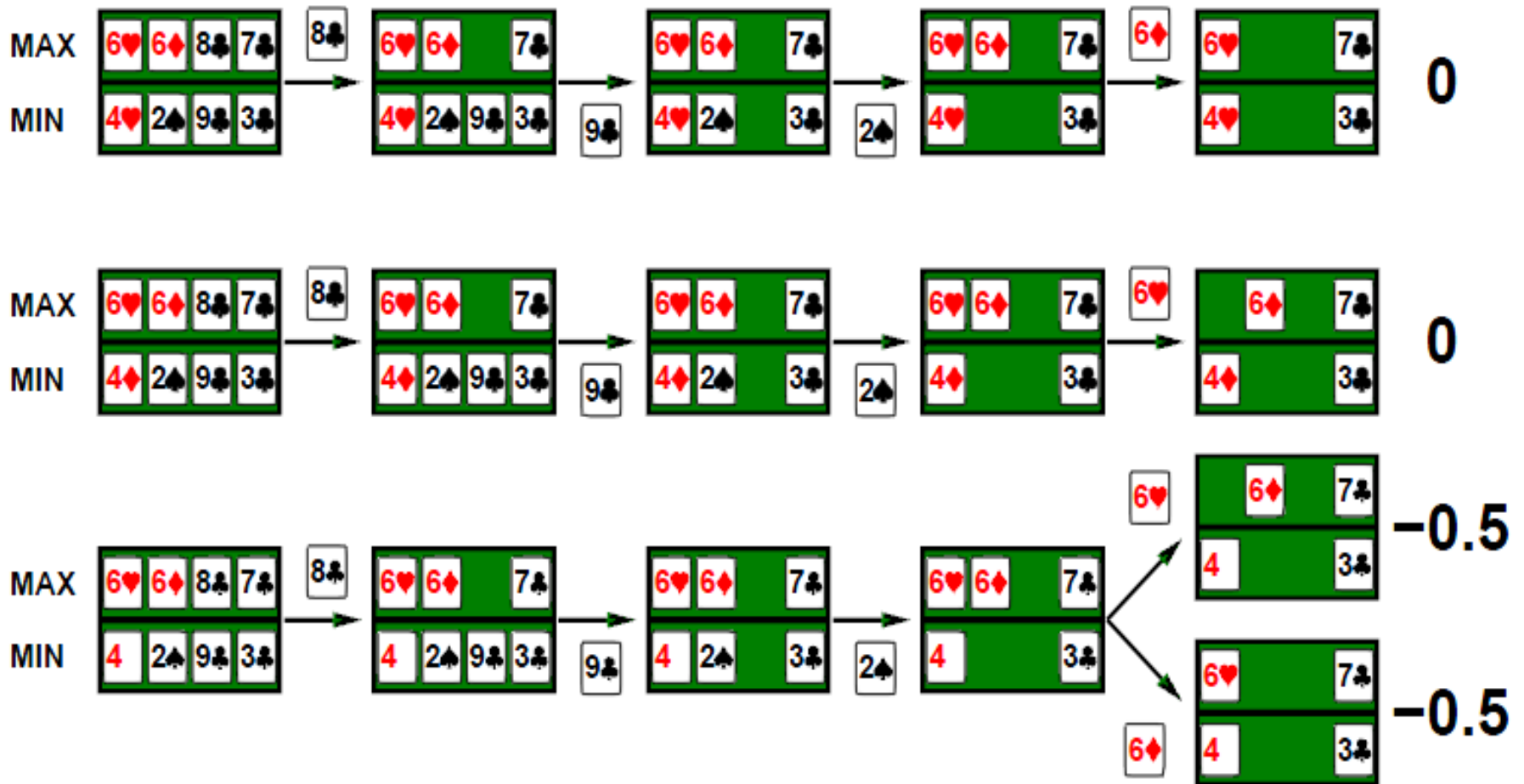
# Example

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



# Example

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



# 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** (信度状态) the agent is in

Can generate and search a tree of information states

Leads to rational behaviors such as

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

# Computers Playing Texas Holder

- 2015, Heads-up limit hold'em poker is solved (有限下注德州扑克)
- 2017, DeepStack became the first AI capable of beating professional poker players at heads-up no-limit Texas hold'em poker (无限下注德州扑克)
- 2019, AI now masters six-player poker, Science



# Computers Playing Texas Holder

## A MYSTERY AI JUST CRUSHED THE BEST HUMAN PLAYERS AT POKER



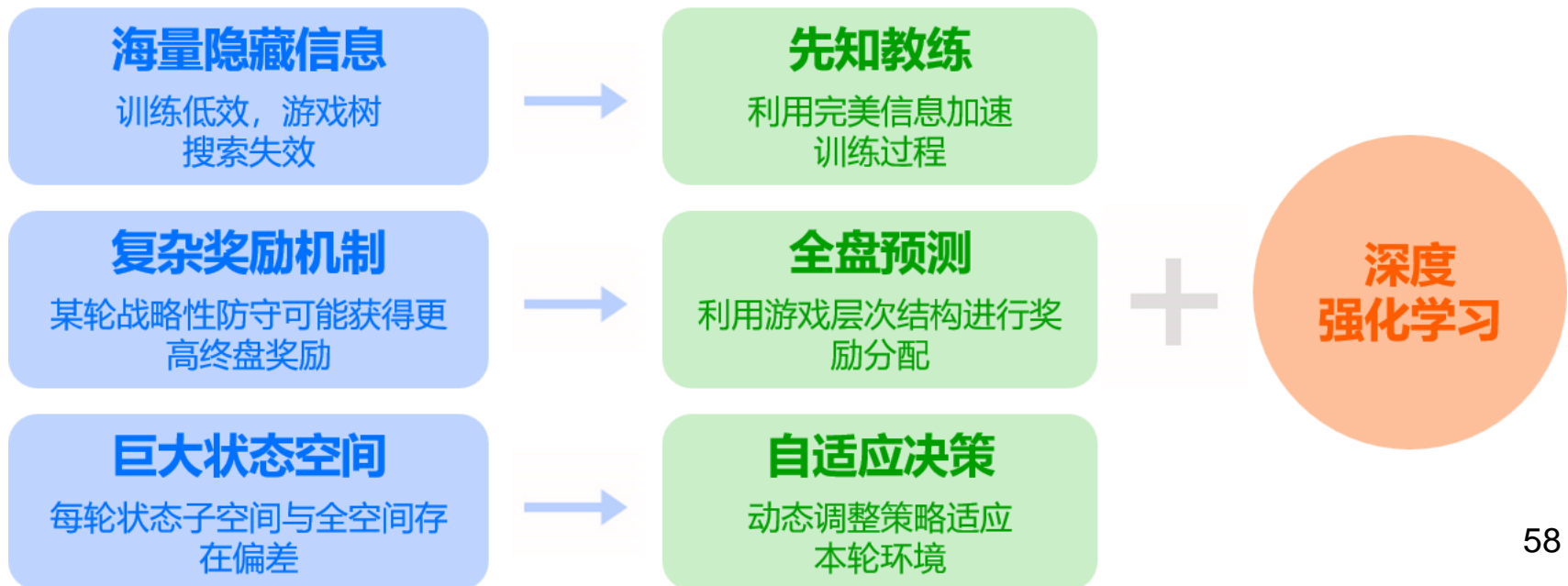
Professional poker player Jason Les plays against "Libratus," at Rivers Casino in Pittsburgh, on January 11, 2017.  ANDREW RUSH/PITTSBURGH POST-GAZETTE/AP

According to the human players that lost out to the machine, Libratus is aptly named. It does a little bit of everything well:

- knowing when to bluff
- and when to bet low with very good cards,
- as well as when to change its bets just to thrown off the competition.

# Computers Playing Mahjong

- 2017 年，竞技麻将将成为继围棋、象棋、国际象棋、桥牌和国际跳棋之后的第六项国际正式智力运动项目
- 2019年6月，由微软亚洲研究院开发的麻将 AI 系统 Suphx 成为首个在国际知名专业麻将平台“天凤”上荣升十段的 AI 系统



# Summary

- Games are fun to work on!
  - perfection is unattainable → must approximate
  - Games are to AI as grand prix racing is to automobile design
- Game playing is best modeled as a search problem
  - Search trees for games represent alternate computer/opponent moves
- Evaluation functions estimate the quality of a given board configuration for each player
- **Minimax** is an algorithm that chooses “optimal” moves by assuming that the opponent always chooses their best move
- **Alpha-beta** is an algorithm that can avoid large parts of the search tree, thus enabling the search to go deeper — 消除无关的子树以提高效率

# Summary of Search

- Uninformed search strategies
  - Breadth-first search (BFS), Uniform cost search, Depth-first search (DFS), Depth-limited search, Iterative deepening search
- Informed search strategies
  - Best-first search: greedy, A\*
  - Local search: hill climbing, simulated annealing etc.
- Constraint satisfaction problems
  - Backtracking = depth-first search with one variable assigned per node
  - Enhanced with: Variable ordering and value selection heuristics, forward checking, constraint propagation

# 作业

- 6.1, 6.3, 6.5 (第二版) = 5.9, 5.8, 5.13 (第三版)