Game Theory Meets Computer Science

Part Two：Game Playing Algorithms

南开大学软件学院
COLLEGE OF SOFTWARE
NANKAI UNIVERSITY

姜桂飞
G.Jiang@nankai.edu.cn

Exercise: Pick up a number

- Without showing your neighbor what you‘re doing, put in the box below a whole number between 1 and 100.
- We will calculate the average number chosen in the class.
- The winner in this game is the person whose number is closest to one-fifth times the average in the class.
- The winner will win the prize.

Winners

- 参与人数：111
- 有效人数：103
- 平均数：25.728
- 平均数的1/5：5.1456

5

黄渲雯，王智，张靖超，王瑞琦

Recap

- Three Concepts
  - Dominant-strategy equilibrium
  - Nash equilibrium
  - Mixed-strategy Nash equilibrium
- Four Lessons
  - *Put yourself in your opponents' shoes*

Outline

- Background
- Game Tree
- Adversarial Search
  - Minimax Search
  - Alpha-Bata Pruning
- Monte-Carlo Tree Search
- Generalized Reinforcement Learning

Background

Adversarial Search often Known as Games

Definitions of Game theory

- Study of strategic decision making . Specifically , study of mathematical models of conflict and cooperation between intelligent rational decision makers

Applications of Game theory

- Economics, political science, psychology, logic, computer science, and biology
- Behavioral relations and decision science, including both humans and non-humans (e.g. computers).

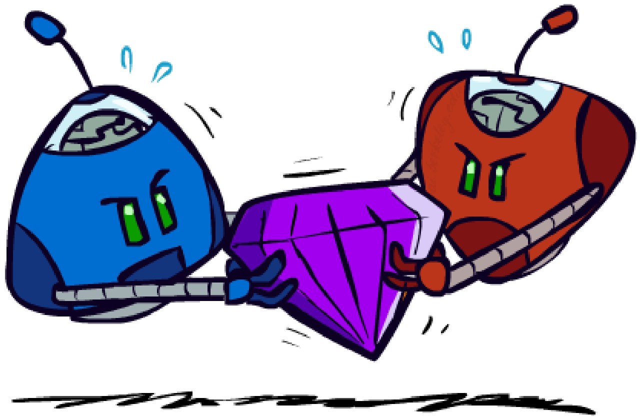
Features of games

- Two or more players (agents)
- Turn-taking vs. simultaneous moves
- Perfect information vs. imperfect information
- Deterministic vs. stochastic
- Cooperative vs. competitive
- Zero-sum vs. non zero-sum

Zero Sum vs. Non-zero Sum

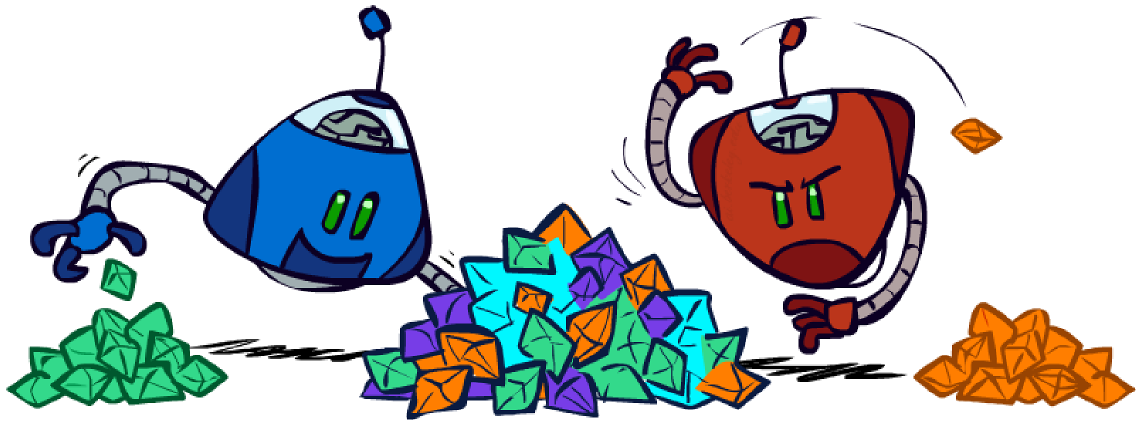
Zero sum games

- Agents have *opposite utilities*.
- Pure competition: win-lose, its sum is “zero”.



Non-zero sum games

- Agents have *independent* utilities.
- Cooperation, indifference, competition, ...
- Win-win, win-lose or lose-lose, its sum is not “zero”.

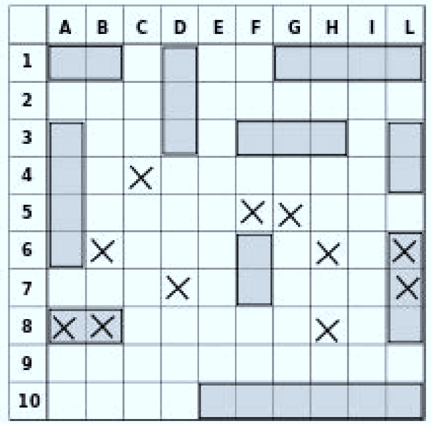


Types of Games

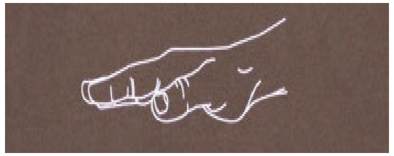
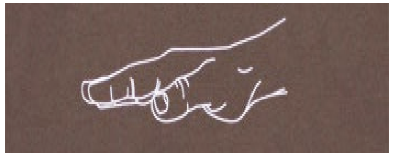
	Deterministic	**Stochastic**
Perfect information (fully observable)	Chess 国际象棋 Checkers 西洋跳棋 Go 围棋 Othello 黑白棋	Backgammon西洋双陆棋 Monopoly 大富翁

Types of Games

	Deterministic	Stochastic
Imperfect information (partially observable)	Stratego 西洋陆军棋 Battleships 海战棋	Bridge 桥牌 Poker 扑克 Scrabble拼字游戏



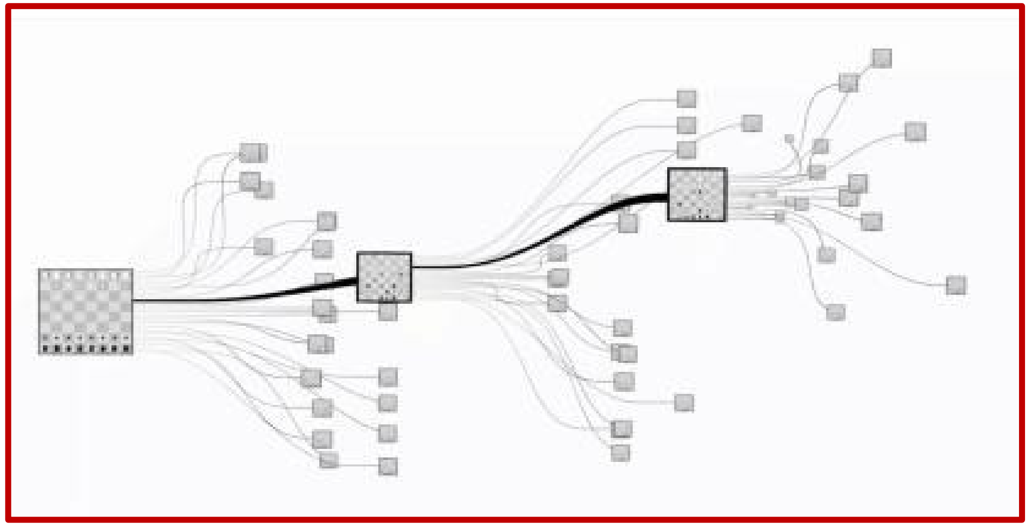
Ex： Rock-Paper-Scissor

P1 \ P2			
	0,0	-1,1	1,-1
	1,-1	0,0	-1 , 1
	-1,1	1 , -1	0,0

Which type?

Games are Interesting but Too Hard to Solve

- E.g., Chess: average branching factor ≈35, each player often go to 50 moves, so search tree has about 35^{100} or 10^{154} nodes!



- Games, like the real world, therefore require the ability to make *some* decision even when calculating the *optimal* decision is infeasible.
- Game playing research has spawned a number of interesting ideas on how to make the best possible use of time

AI Algorithms

"Games are the perfect platform for developing and testing AI algorithms."

--------Demis Hassabis

Origins of Game Playing Algorithms

1912	Ernst Zermelo 恩斯特·策梅洛	Minimax algorithm 最小最大算法
1949	Claude Shannon 克劳德·香农	Chess playing with evaluation function, selective search 用评价函数和选择性搜索下国际象棋
1956	John McCarthy 约翰·麦卡锡	Alpha-beta search Alpha-beta搜索
1956	Arthur Samuel 亚瑟·塞缪尔	Checkers program that learns its own evaluation function 学习自身的评价函数的西洋跳棋程序

- Ernst Zermelo(1871–1953), a German logician and mathematician.
- Claude Shannon (1916–2001), an American mathematician, and cryptographer known as “the father of information theory”.
- John McCarthy (1927-2011), an American computer scientist and cognitive scientist, and one of the founders of AI.
- Arthur Samuel (1901-1990), an American pioneer of computer gaming, AI, and ML.

Game Playing Algorithms Nowadays

Computers are better than humans

Checkers 西洋跳棋	Solved in 2007 2007 年已解决
Chess 国际象棋	IBM Deep Blue defeated Kasparov in 1997 IBM 深蓝于 1997 年战胜了卡斯帕罗夫
Go 围棋	Google AlphaGo beat Ke Jie in 2017 谷歌 AlphaGo 于 2017 年 3 月战胜了 柯洁

Computers are competitive with top human players

Backgammon 西洋双陆棋	TD Gammon used reinforcement learning to learn evaluation function TD Gammon 使用了强化学习方法来得到评价函数
Bridge 桥牌	Top systems use Monte Carlo simulation & alpha-beta search 顶级的系统使用 蒙特卡罗仿真和 alpha-beta 搜索

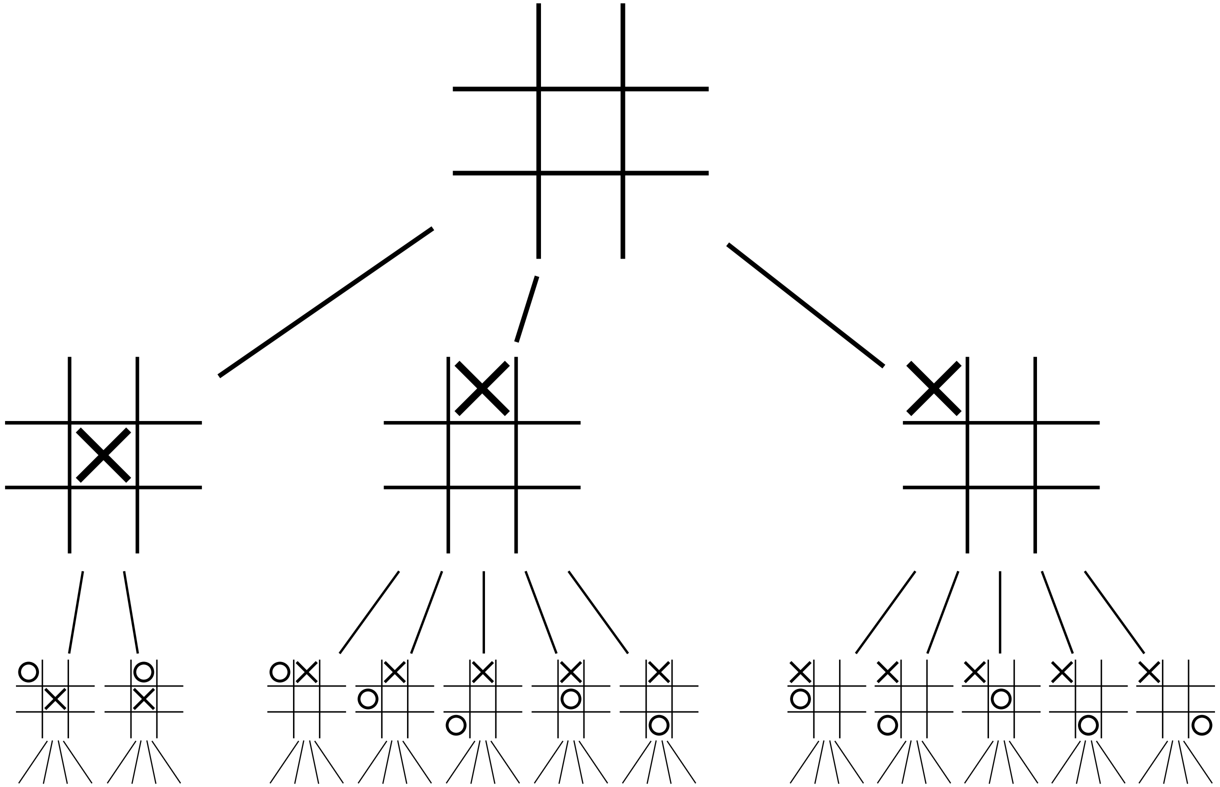
Game Tree

Two Players Games

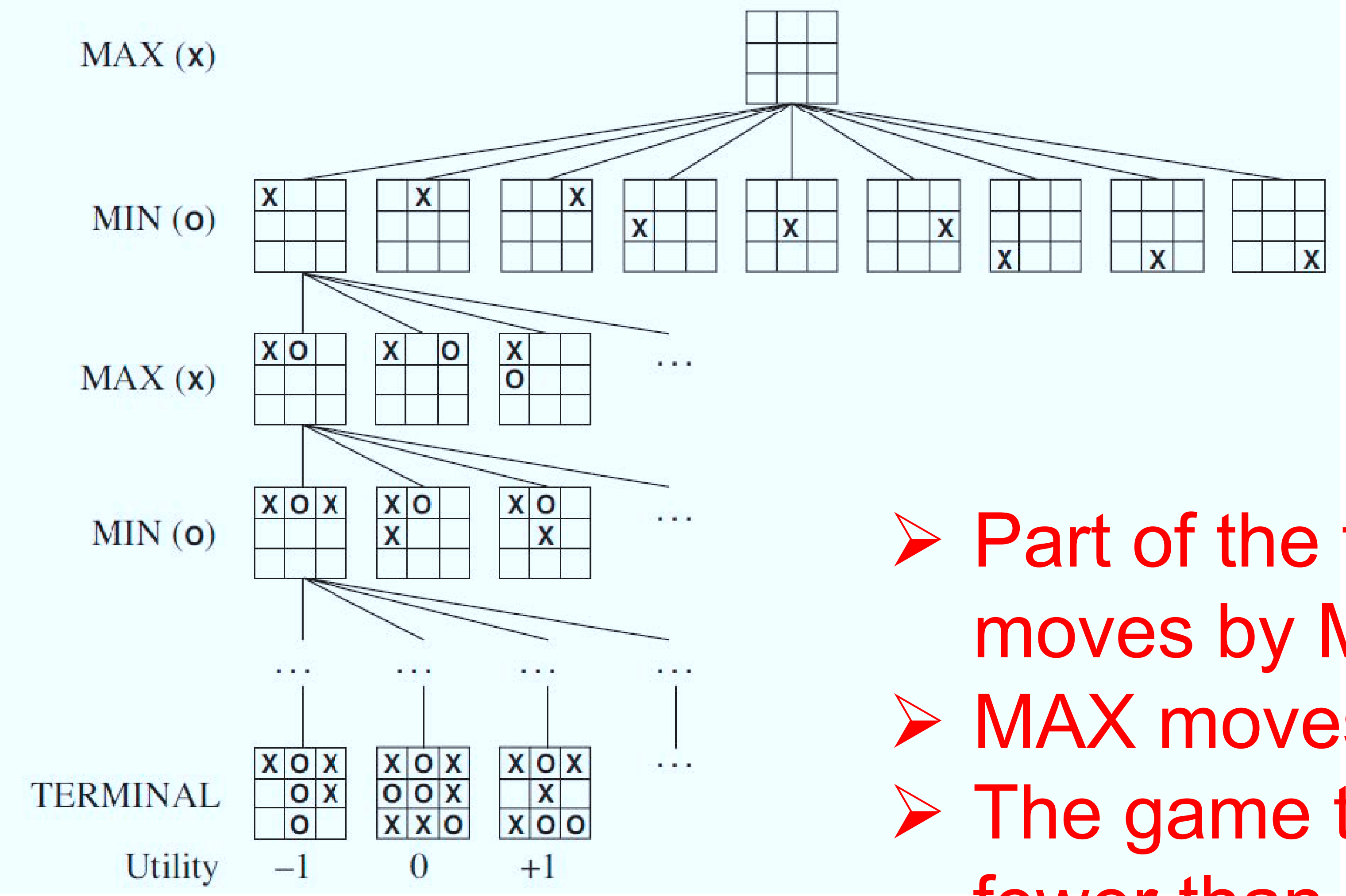
- Feature
  - deterministic, perfect information, turn-taking, two players, zero-sum.
- Calling the two players
  - MAX, MIN.
  - MAX moves first, and then they take turns moving, until the game is over.
- At game end
  - winner: award points
  - loser: give penalties.

Game Tree

- A directed graph whose nodes are positions in a game and whose edges are moves.

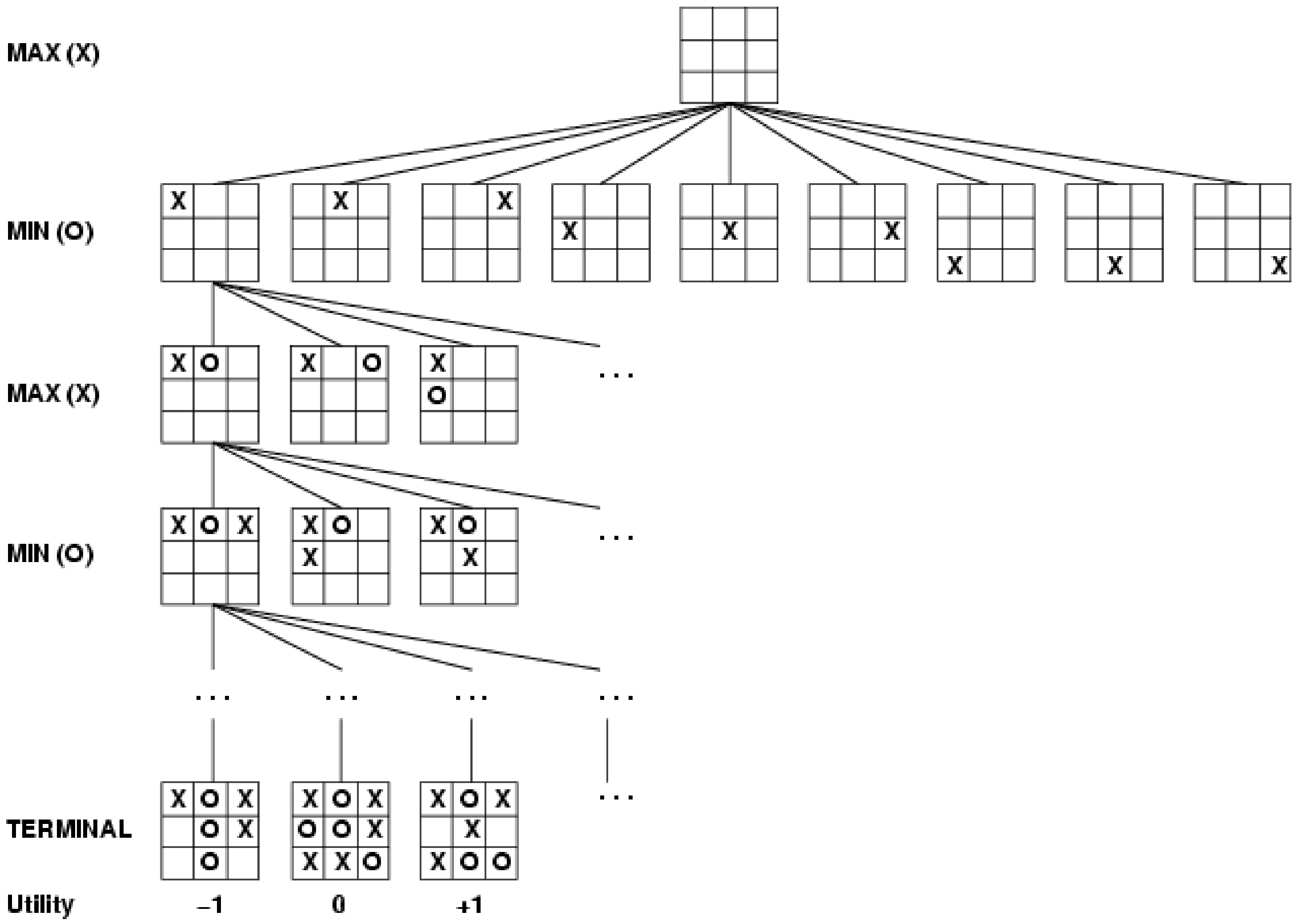


Example: Game Tree of Tic-tac-toe



- ➢ Part of the tree, giving alternating moves by MIN(O) and MAX(X).
- ➢ MAX moves first.
- ➢ The game tree is relatively small, fewer than 9! = 362,880 nodes.

Example: Game Tree of Tic-tac-toe



How do we search this tree to find the optimal move?

Search versus Games

- Search – no adversary
  - Solution is (heuristic) method for finding goal
  - Heuristics and CSP techniques can find *optimal* solution
  - Evaluation function: estimate of cost from start to goal through given node
  - Examples: path planning, scheduling activities

- Games – adversary
  - Solution is strategy
    - strategy specifies move for every possible opponent reply.
  - Time limits force an *approximate* solution
  - Evaluation function: evaluate “goodness” of game position
  - Examples: chess, checkers, Othello, backgammon

Games as Search

- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over
  - Winner gets reward, loser gets penalty.
  - "Zero sum" means the sum of the reward and the penalty is a constant.
- MAX uses search tree to determine next move.

Formal definition as a search problem

- **Initial state:** Set-up specified by the rules
- **Player(s):** Defines which player has the move in a state.
- **Actions(s):** Returns the set of legal moves in a state.
- **Result(s,a):** Transition model defines the result of a move.
- **Terminal-Test(s):** Is the game finished? True if finished, false otherwise.
- **Utility function(s,p):** Gives numerical value of terminal state s for player p.
  - E.g., win (+1), lose (-1), and draw (0) in tic-tac-toe.
  - E.g., win (+1), lose (0), and draw (1/2) in chess.

Minimax Search

Optimal Solution

In normal search

- The optimal solution would be a sequence of actions leading to a goal state(terminal state) that is a win.

In adversarial search

- Both of MAX and MIN could have an optimal strategy.
- In initial state, MAX must find a strategy to specify MAX's move
- then MAX's moves in the states resulting from every possible response by MIN, and so on.

Minimax Theorem

For every two-player, zero-sum game with finitely many strategies, there exists a value V and a mixed strategy for each player, such that

- (a) Given player 2’s strategy, the best payoff possible for player 1 is V,
- (b) Given player 1's strategy, the best payoff possible for player 2 is −V.

For a zero sum game, the name minimax arises because each player *minimizes the maximum payoff* possible for the other, he also *minimizes his own maximum loss*.

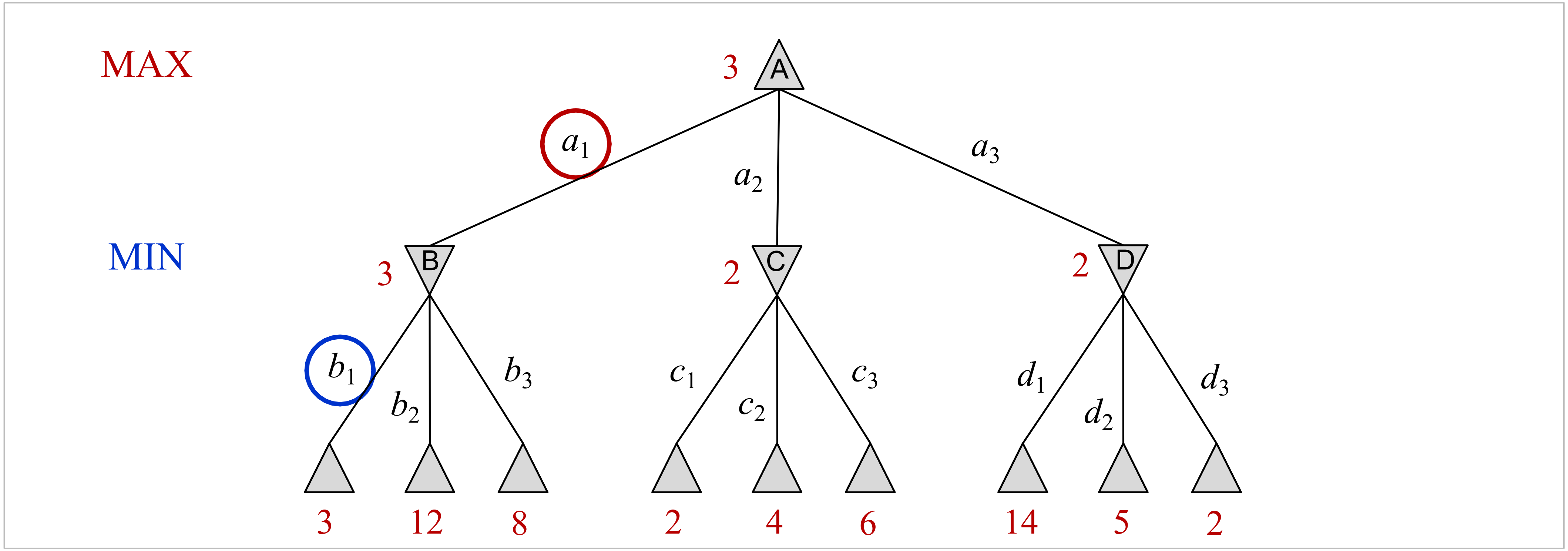
Optimal Solution in Adversarial Search

- Given a game tree, the optimal strategy can be determined from the minimax value of each node, write as MINIMAX(n).
- Assume that both players play optimally from there to the end of the game.

Function MINIMAX(s) **returns** an action
  if TERMINAL-TEST(s) **then return** UTILITY(s)
  if PLAYER(s) = MAX **then return** $\max_{a \in \text{ACTIONS}(s)}$ MINIMAX(RESULT(s, a))
  if PLAYER(s) = MIN **then return** $\min_{a \in \text{ACTIONS}(s)}$ MINIMAX(RESULT(s, a))

The minimax value of a terminal state is just its utility.
MAX prefers to move to a state of maximum value, MIN prefers a state of minimum value

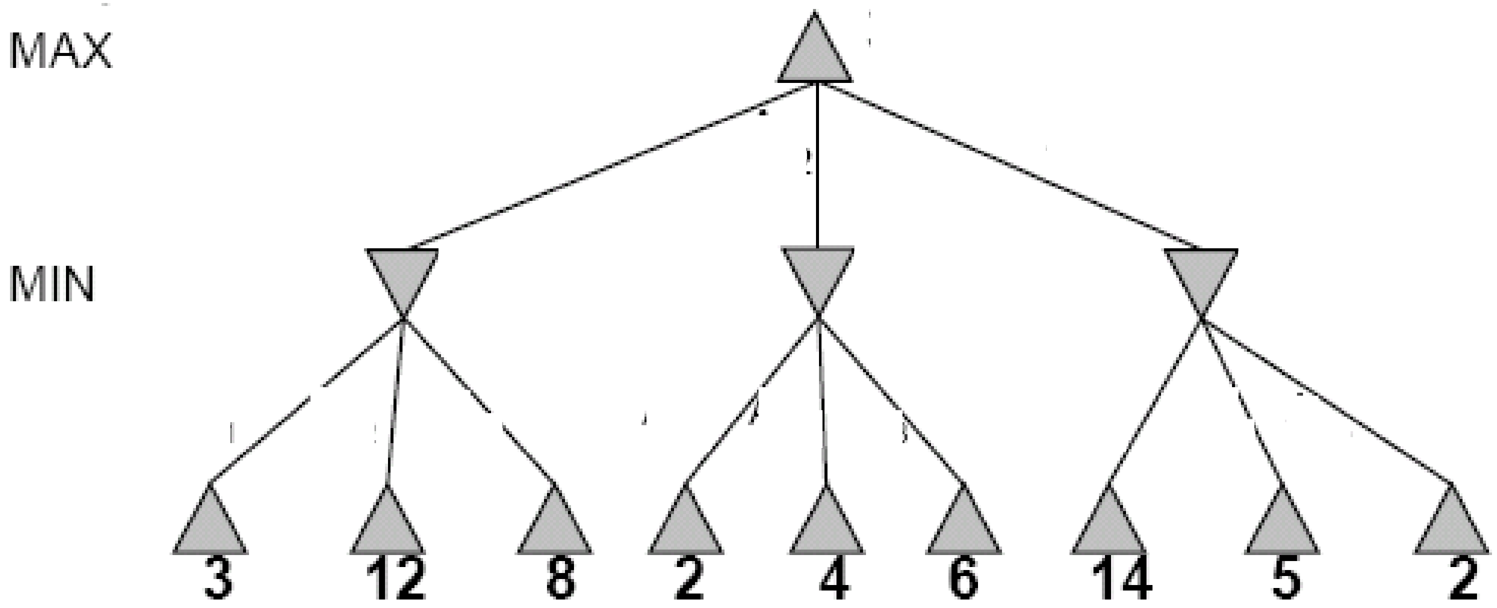
Minimax Decision -- A Two-player Game Tree



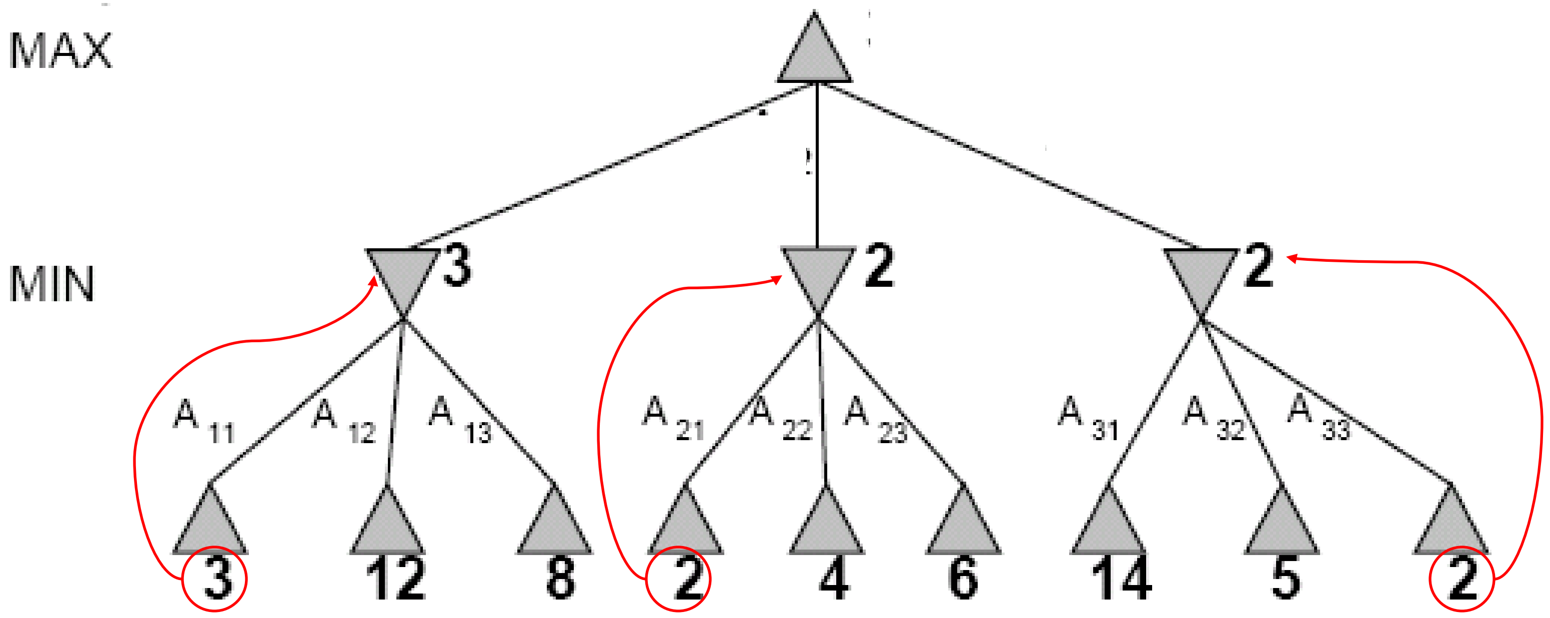
MAX's best move at root is a_1 (with the highest minimax value)

MIN's best reply at B is b_1 (with the lowest minimax value)

A Two-player Game Tree

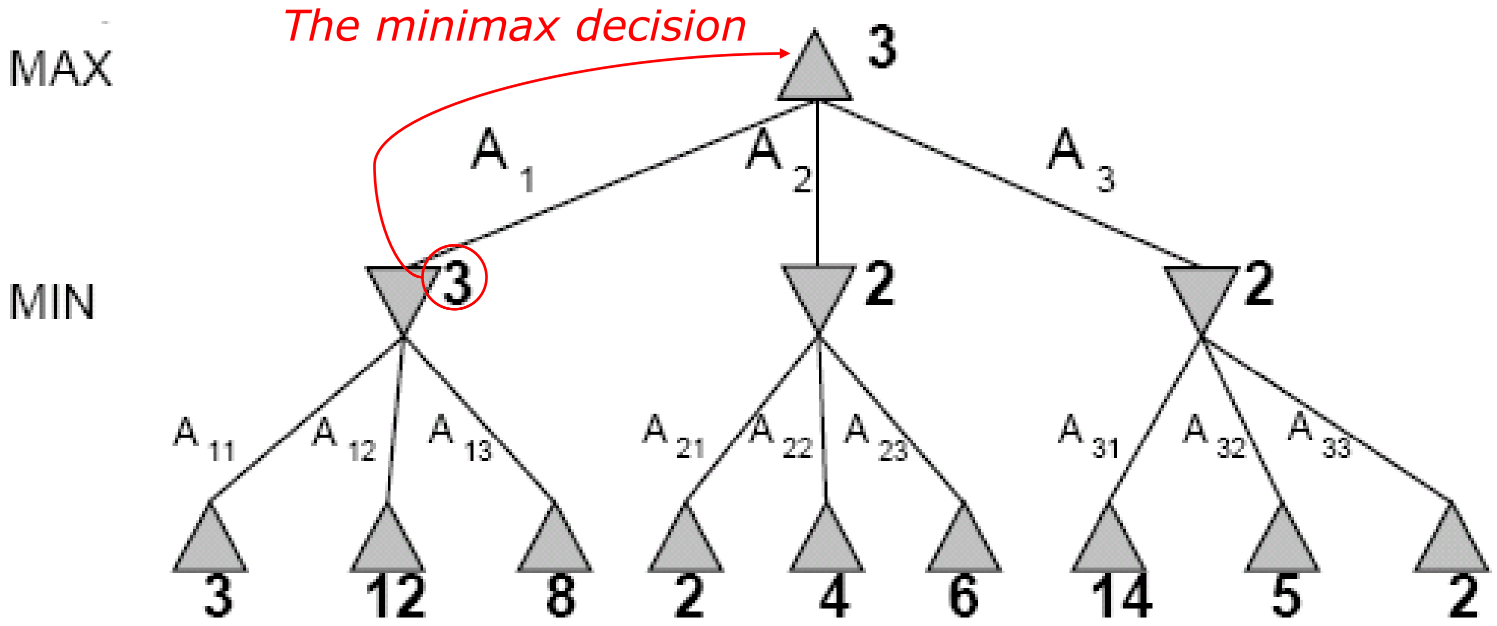


A Two-player Game Tree



A Two-player Game Tree

Minimax maximizes the utility for the worst-case outcome for max



Minimax Algorithm

Designed to find the optimal strategy for Max and find best move:

- 1. Generate the whole game tree, down to the leaves.
- 2. Apply utility (payoff) function to each leaf.
- 3. Back-up values from leaves through branch nodes:
  - a Max node computes the Max of its child values
  - a Min node computes the Min of its child values
- 4. At root: choose the move leading to the child of highest value.

Pseudocode for Minimax Algorithm

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

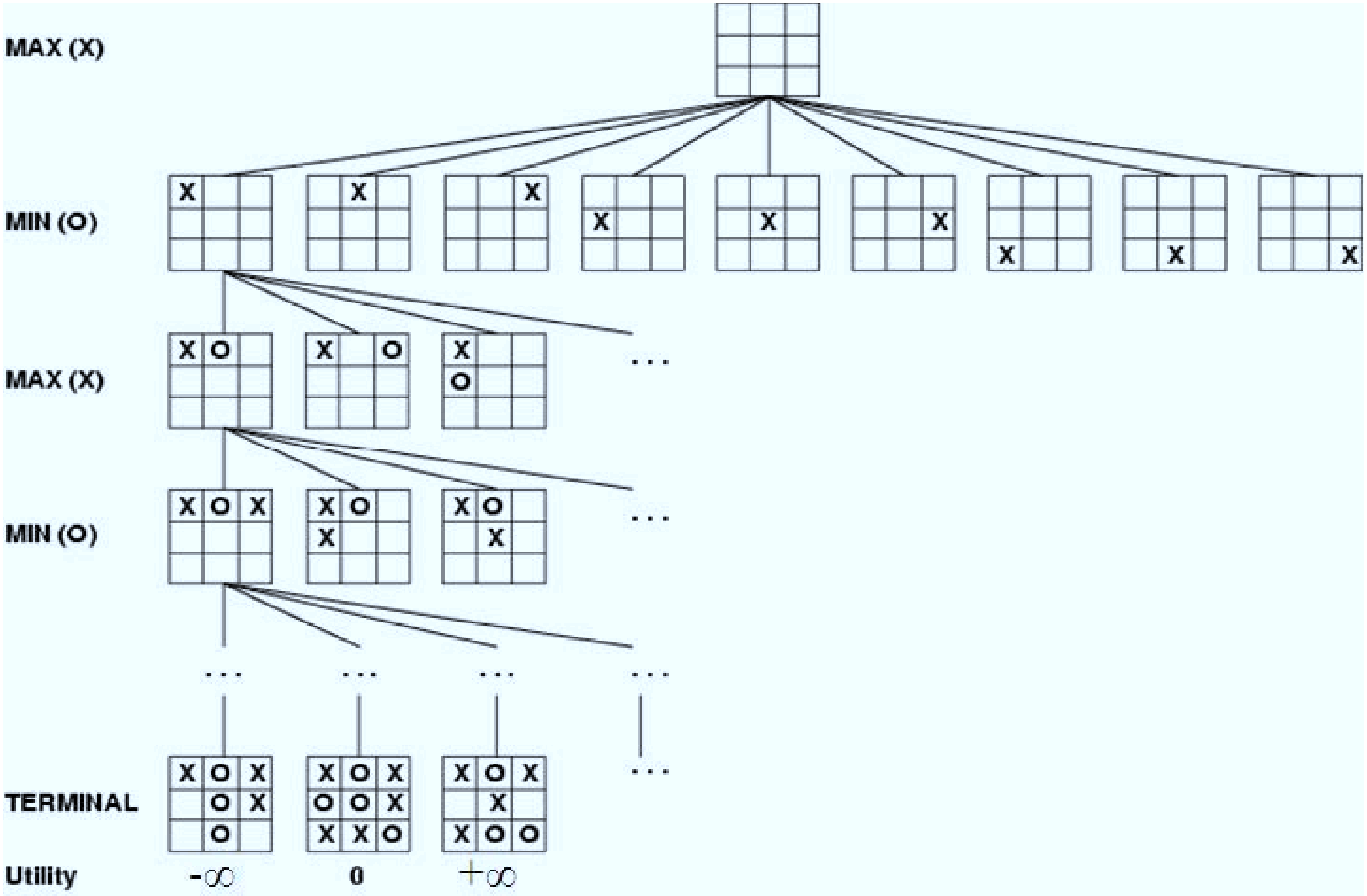
---

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(*state*, *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(*state*, *a*)))
    return v

Ex: Tic-Tac-Toe



Applying MiniMax to tic-tac-toe

Problem with minimax search

Problem with minimax search

- Minimax search needs to generate the whole game tree
- Number of game states is exponential in depth of the tree.

E.g. Chess

$b \approx 35$ (approximate average branching factor)

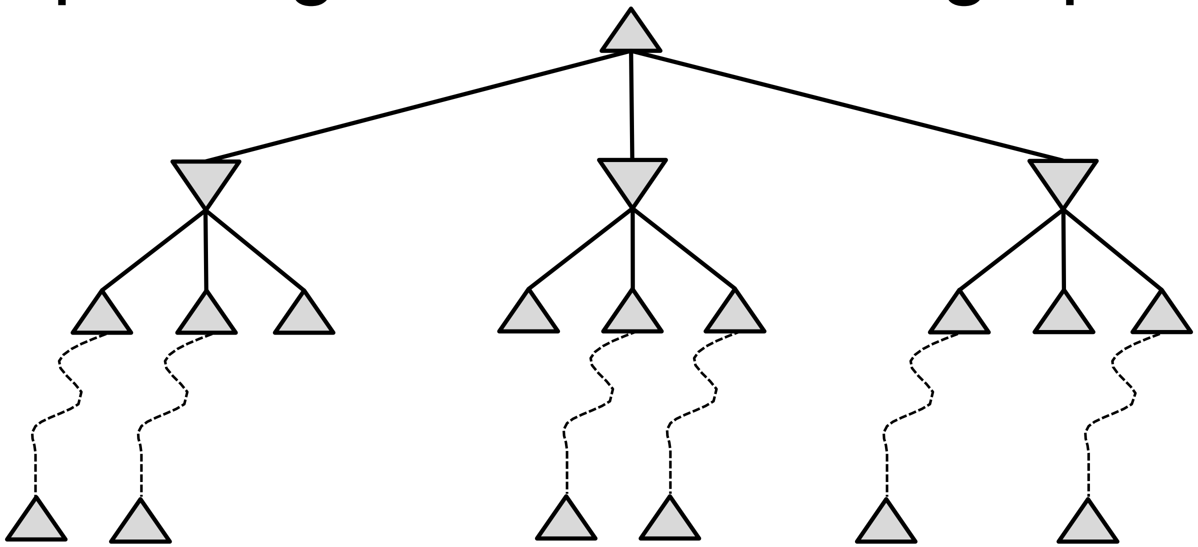
$d \approx 100$ (depth of game tree for "typical" game)

$b^d \approx 35^{100} \approx 10^{154}$ nodes!!

It is usually impossible to develop the whole search tree

One Solution

- Compute correct minimax decision without looking at every node in game tree.
- That is, use “pruning” to eliminate large parts of the tree.



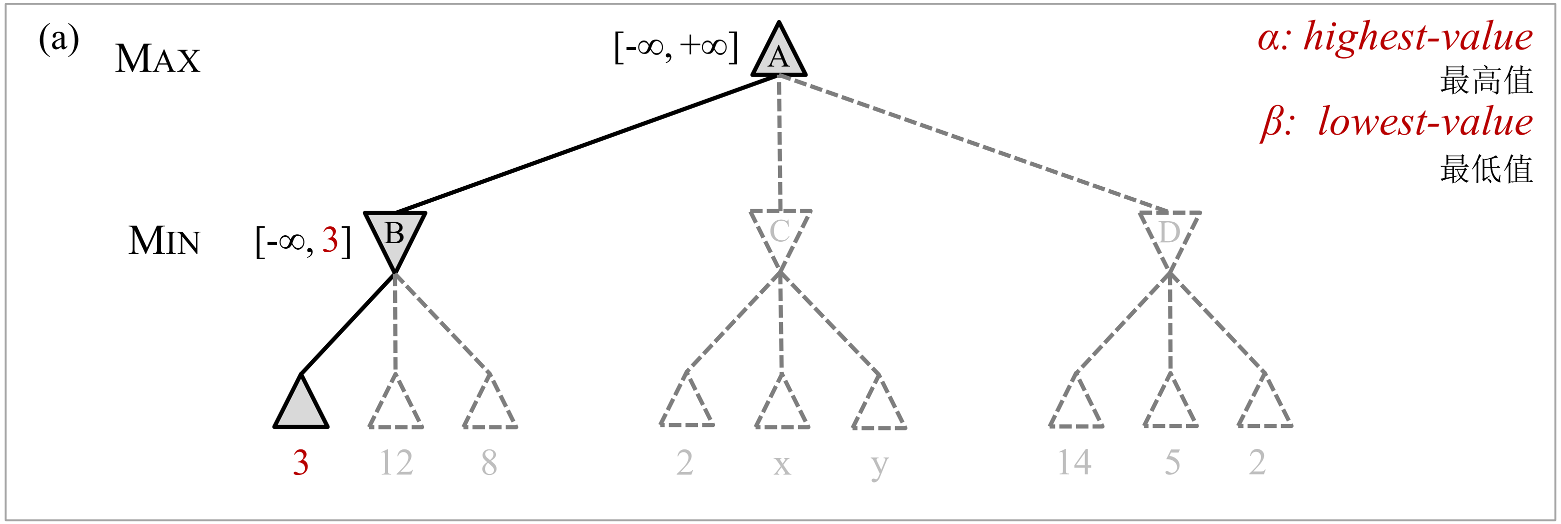
If you have an idea that is surely bad, don’t take the time to see how truly awful it is.
-- Pat Winston (Director, MIT AI Lab, 1972-1997)

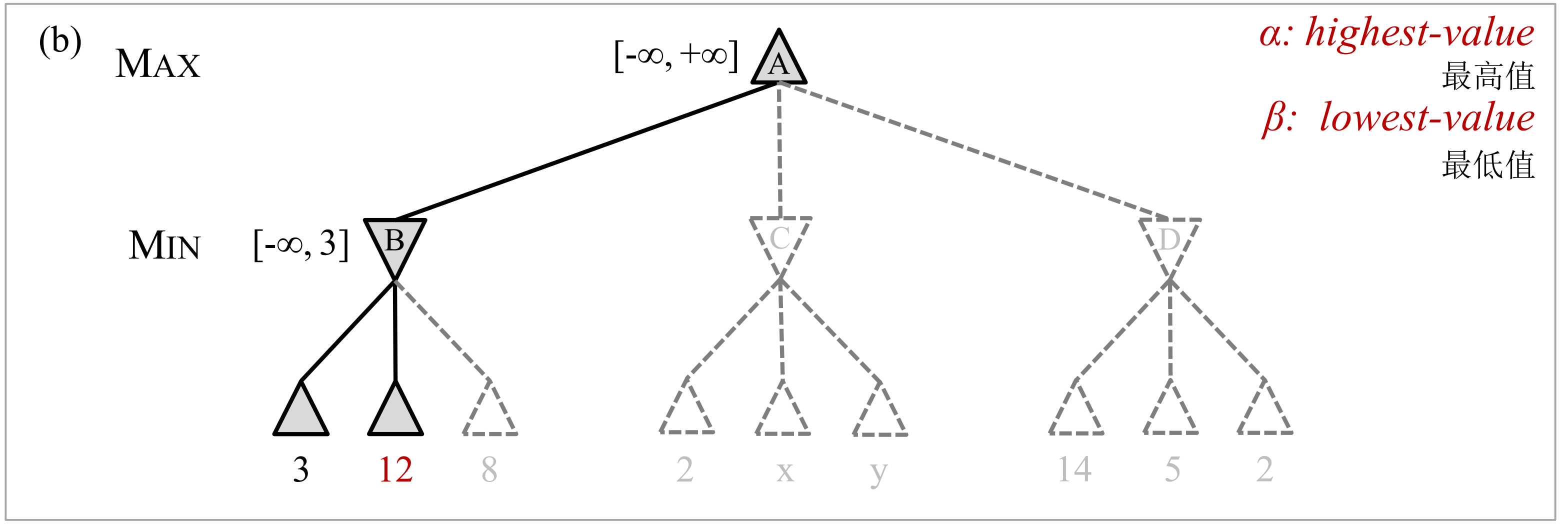
Alpha-Beta Pruning

Alpha-Beta Pruning

- A search algorithm to decrease the number of nodes that are evaluated by the minimax algorithm.
  - α: highest-value we have found so far at any point along the path for MAX.
  - β: lowest-value we have found so far at any point along the path for MIN.
- Alpha–beta search respectively:
  - updates the values of α and β as it goes along, and
  - prunes the remaining branches at a node as soon as the value of the current node is known to be worse than the current α or β value for MAX or MIN.

Example: Game Tree Using Alpha-Beta Pruning





Initial value: 初始值：

$A[\alpha=-\infty, \beta=+\infty]$

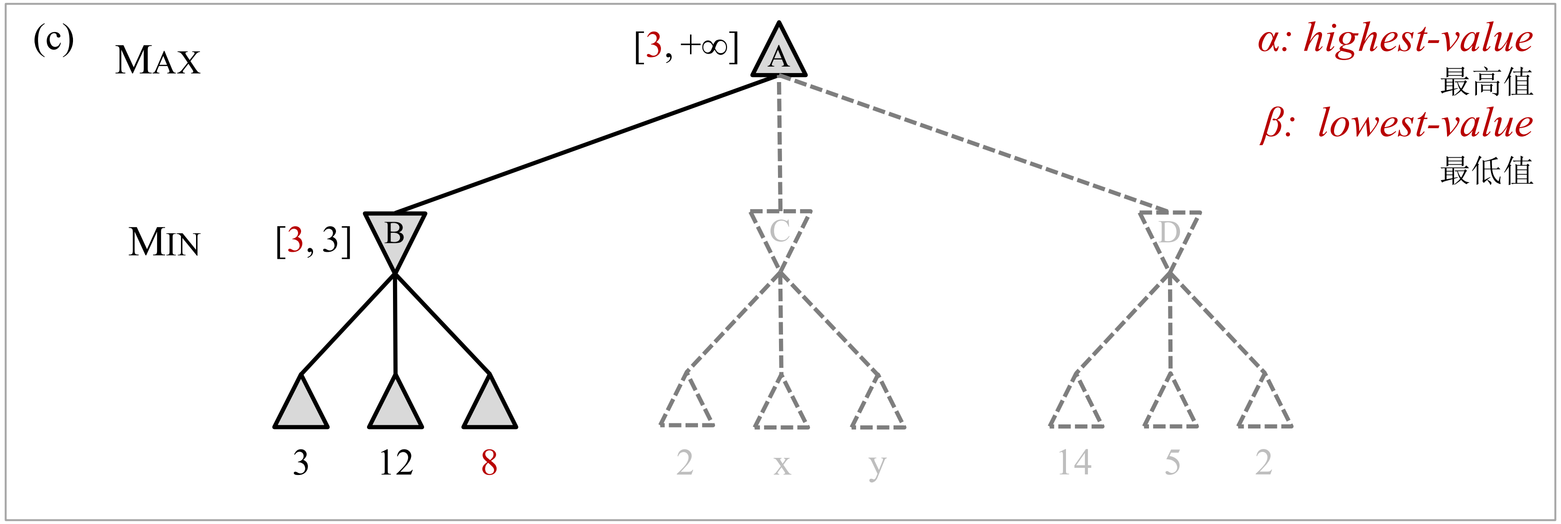
(a) The 1st leaf below B has the value 3. Hence, B, as a MIN node, $B[\beta=3]$.

B下面第一个叶节点的值为3。因此，B作为MIN节点，$B[\beta=3]$。

(b) The 2nd leaf below B has a value of 12; MIN would avoid this move, still $B[\beta=3]$.

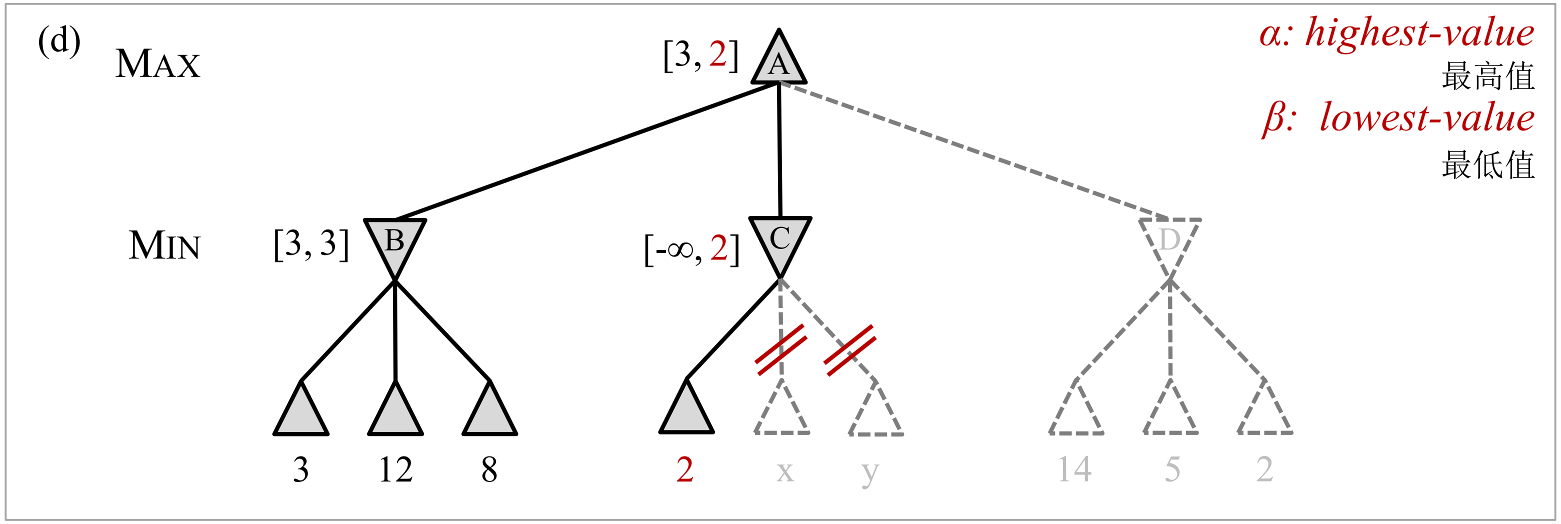
B下面第二个叶节点的值为12； MIN将回避这个移动，仍然是$B[\beta=3]$。

Example: Game Tree Using Alpha-Beta Pruning



(c) The 3rd leaf below B has a value of 8; so exactly MIN node $B[\beta=3]$. Now, we can infer $B[\alpha=3]$, because MAX has $A[\alpha\geq 3]$.

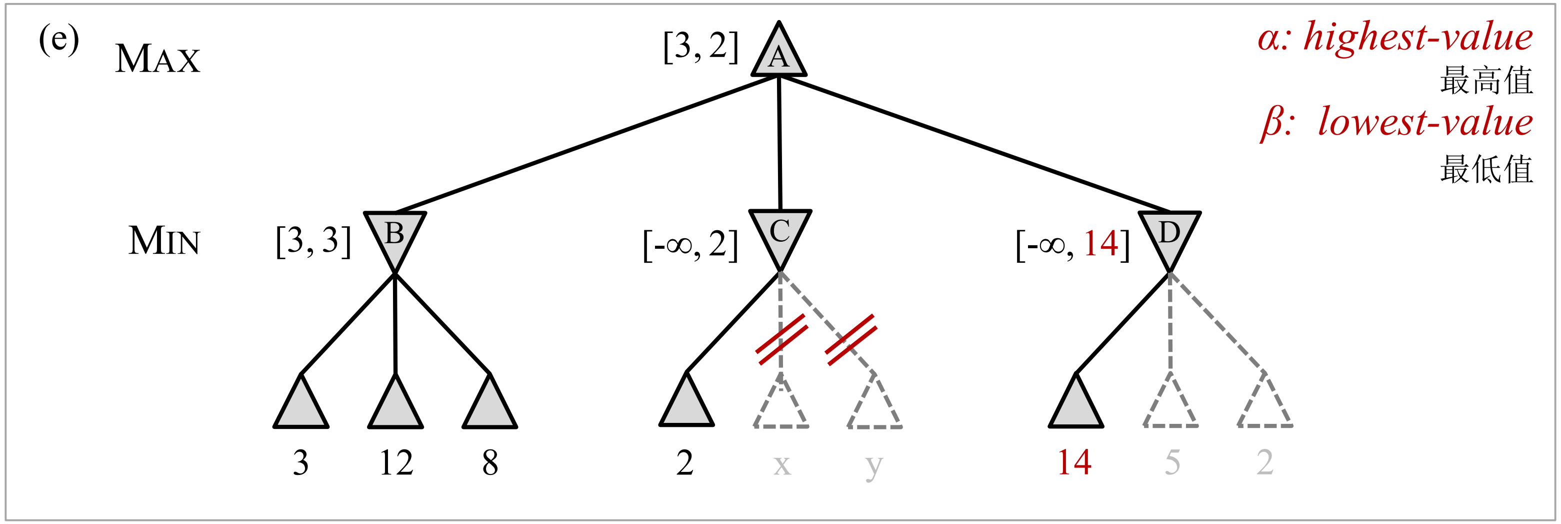
B下面第三个叶节点的值为8；故MIN节点正是$B[\beta=3]$。现在，因为MAX为$A[\alpha\geq 3]$，我们能够推出$B[\alpha=3]$。



(d) The 1st leaf below C has the value 2, hence, as a MIN node $C[\beta=2]$, and $B[\beta=3]>C[\beta=2]$, so MAX would never choose C. Therefore just prune all successor of C (α–β pruning).

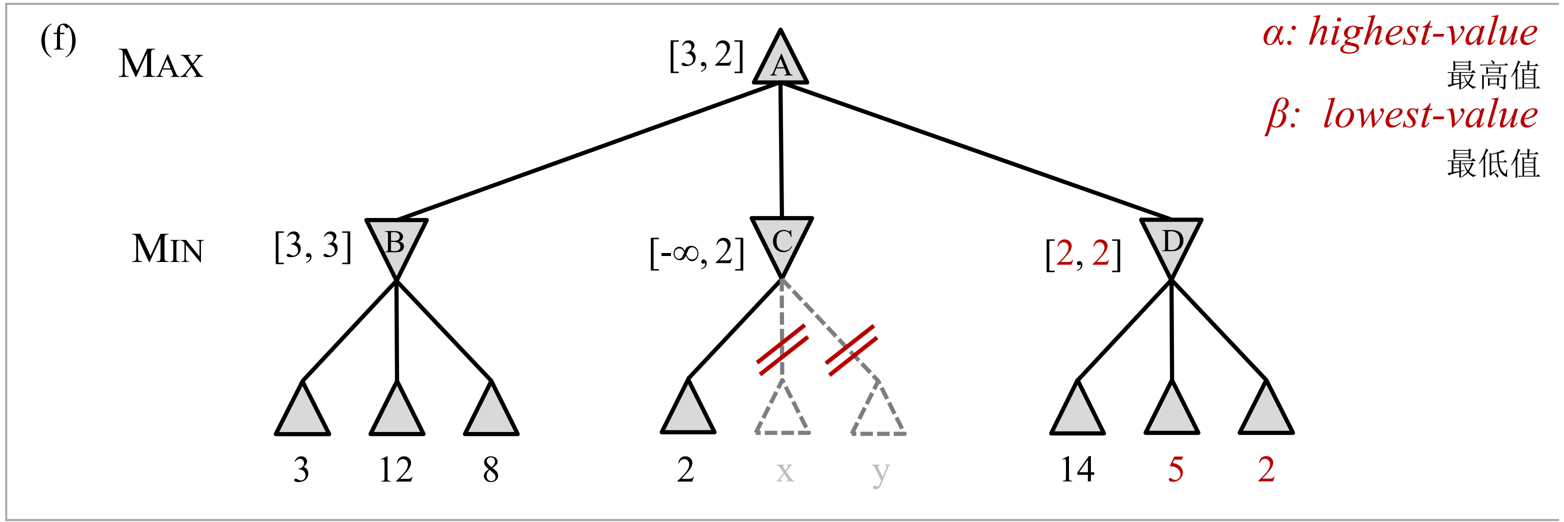
C下面第一个叶节点的值为2，因此，由于MIN节点$C[\beta=2]$, 且$B[\beta=3]>C[\beta=2]$，故MAX将不会选择C，所以只需剪掉C的所有后继节点 (α–β pruning)。

Example: Game Tree Using Alpha-Beta Pruning



(e) The 1st leaf below D is 14, $D[\beta\leq14]$, so we need to keep exploring D's successor states. We now have bounds on all of root's successors, so $A[\beta\leq2]$.

D下面第一个叶节点为14，$D[\beta\leq14]$，故我们需要不断搜索D节点的后继状态。到此我们已经遍布了根节点的所有后继节点，故$A[\beta\leq2]$。

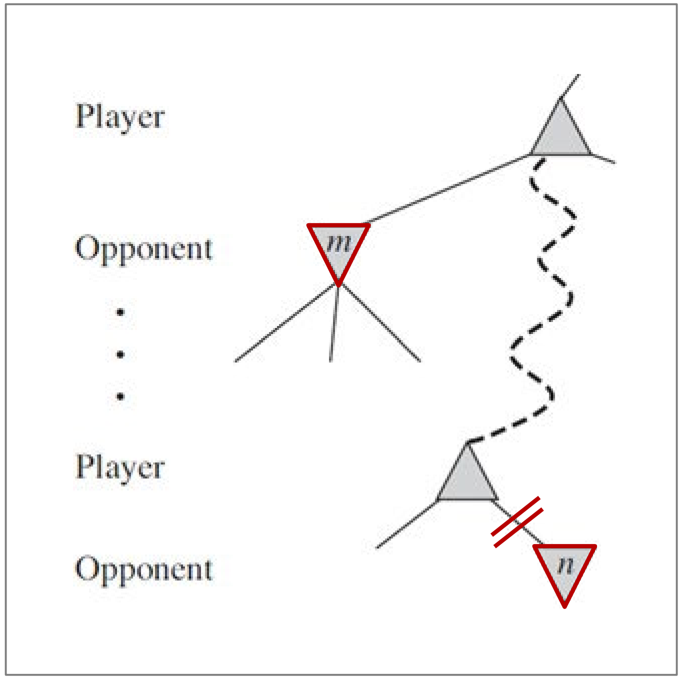


(f) The 2nd successor of D is worth 5, so keep exploring. The 3rd successor is worth 2, so $D[\beta=2]$. MAX's decision at the root keeps $A[\beta=2]$.

D的第2个后继节点的值等于5，故不断搜索，第3个后继节点等于2，故$D[\beta=2]$。根节点MAX的抉择保持$A[\beta=2]$。

General Principle of Alpha-Beta Pruning

- Alpha–beta pruning can be applied to trees of any depth, and often possible to prune entire subtrees rather than just leaves.
- The general principle:
  - Consider a node n somewhere in the tree, such that Player has a choice of moving to that node.
  - If Player has a better choice m at parent node of n, or at any choice point further up, then n *will never be reached* in actual play.



When to Prune

- Prune whenever $\alpha \geq \beta$.
  - Prune below a Max node whose alpha value becomes greater than or equal to the beta value of its ancestors.
    - **Max nodes update alpha** based on children's returned values.
  - Prune below a Min node whose beta value becomes less than or equal to the alpha value of its ancestors.
    - **Min nodes update beta** based on children's returned values.

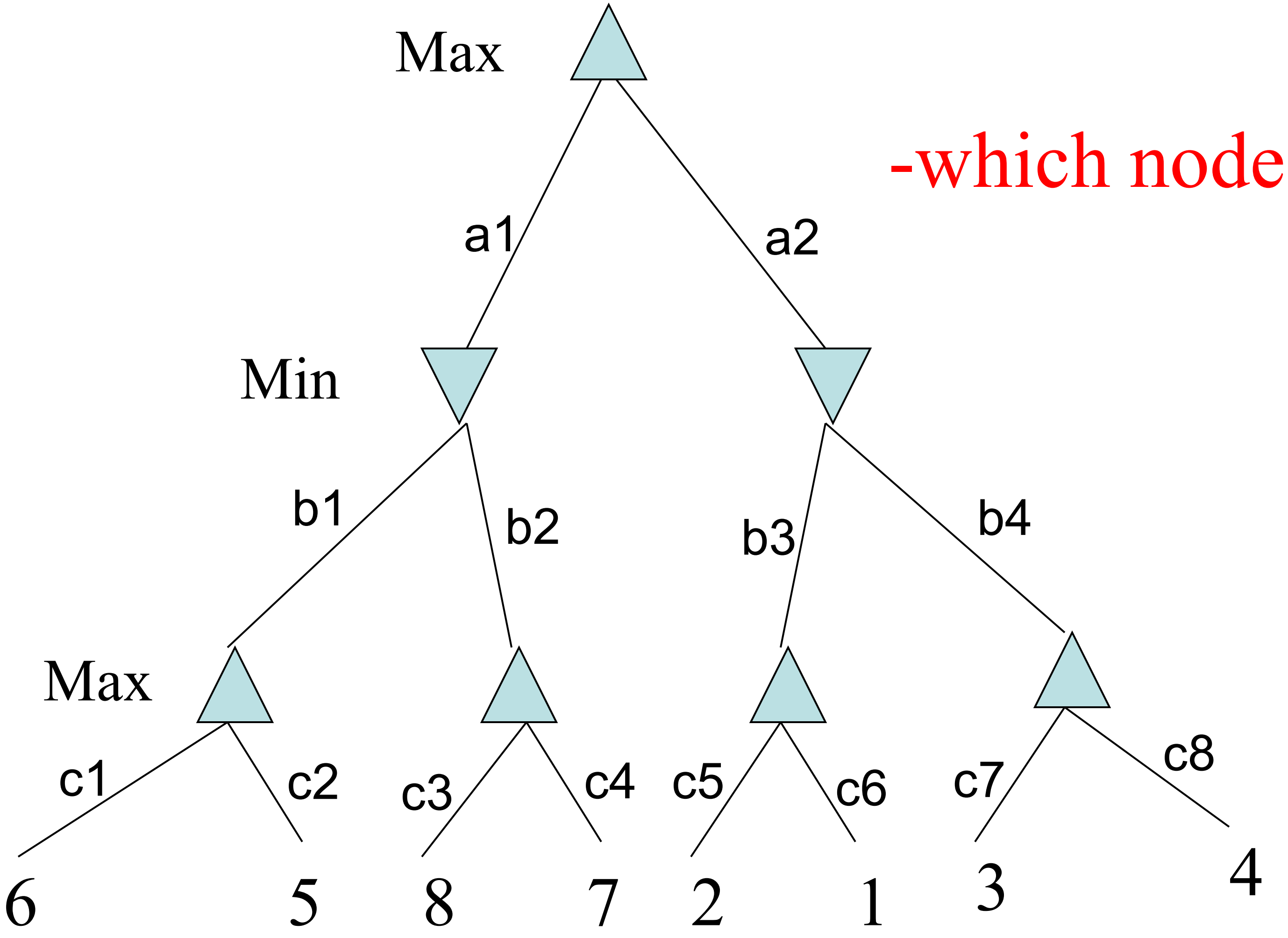
Alpha-Beta Search Algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
   v ← MAX-VALUE(state, −∞, +∞)
   return the action in ACTIONS(state) with value v
```

```
function MAX-VALUE(state, α, β) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for each a in ACTIONS(state) do
     v ← MAX(v, MIN-VALUE(RESULT(state, a), α, β))
     if v ≥ β then return v else α ← MAX(α, v)
   return v
```

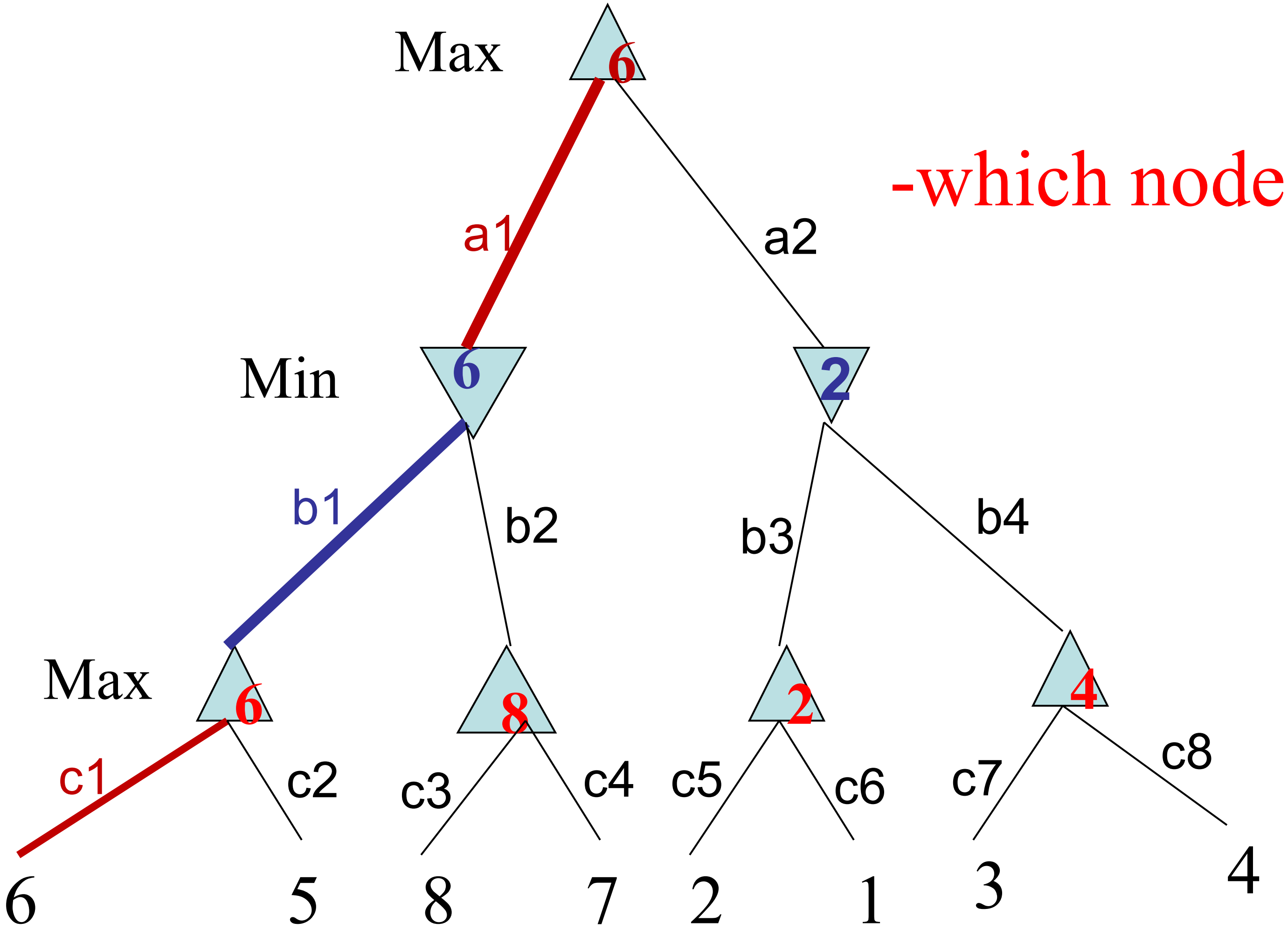
```
function MIN-VALUE(state, α, β) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← +∞
   for each a in ACTIONS(state) do
     v ← MIN(v, MAX-VALUE(RESULT(state, a), α, β))
     if v ≤ α then return v else β ← MIN(β, v)
   return v
```

Example



-which nodes can be pruned?

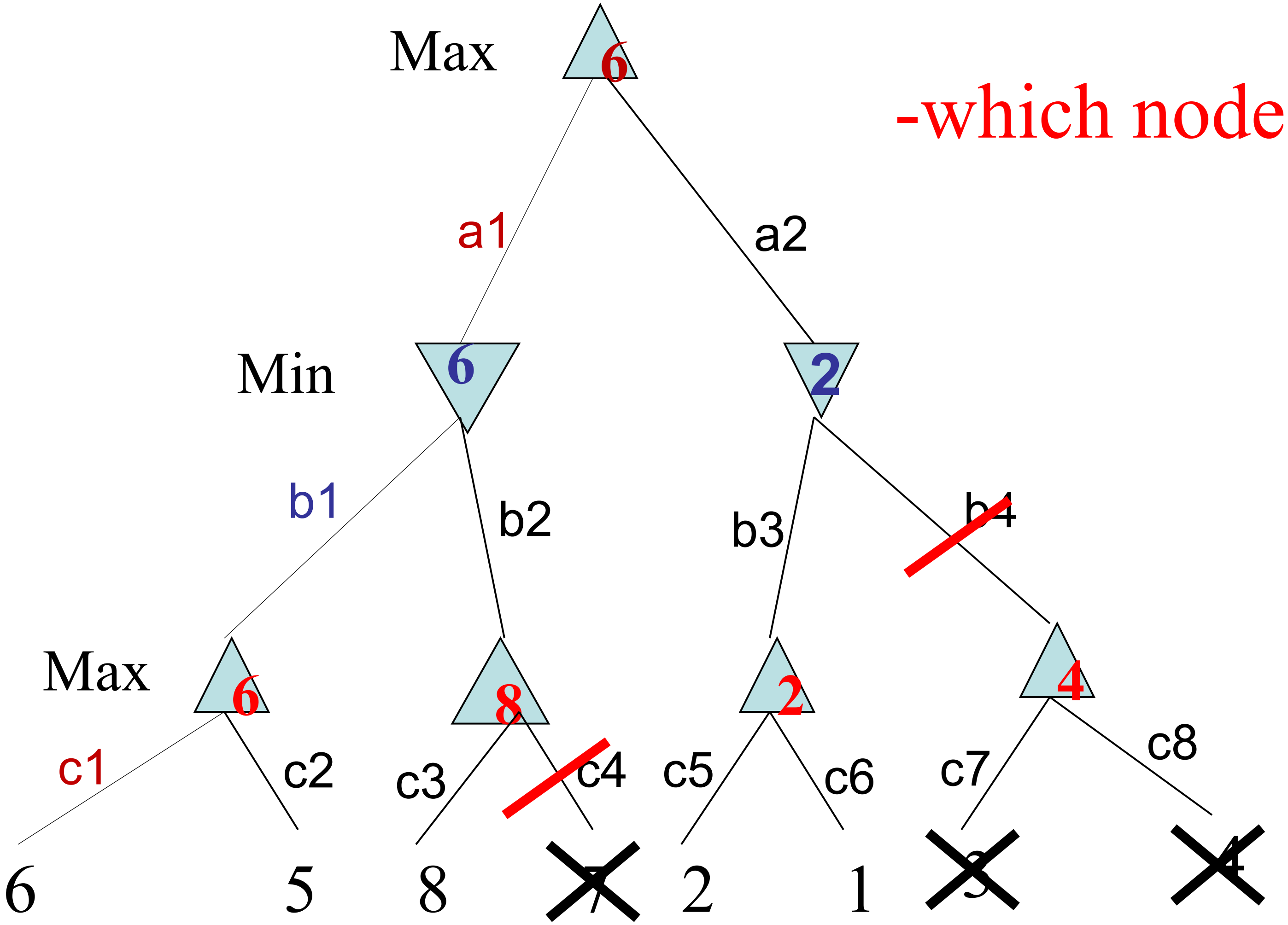
Example



-which nodes can be pruned?

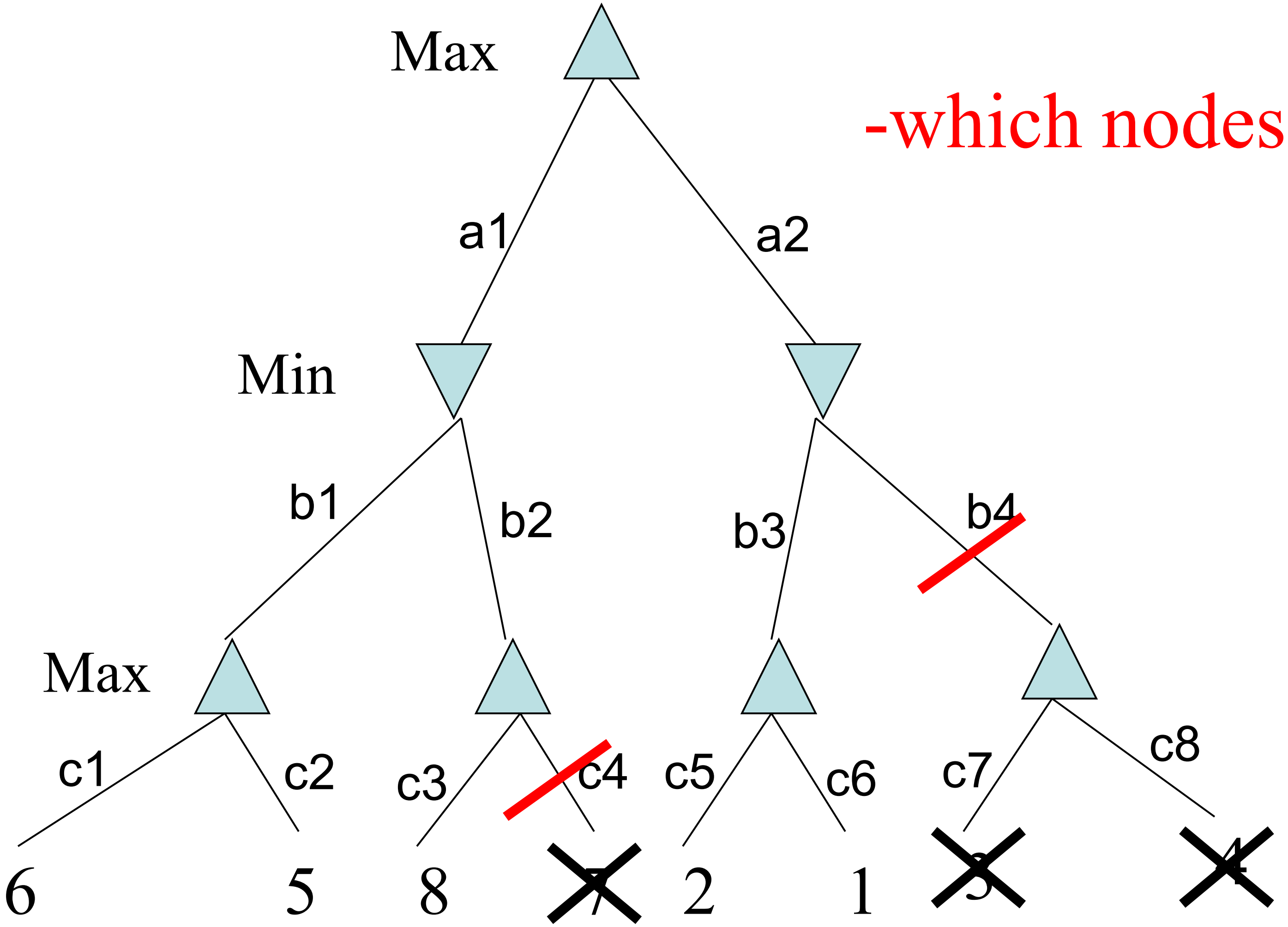
Example

-which nodes can be pruned?

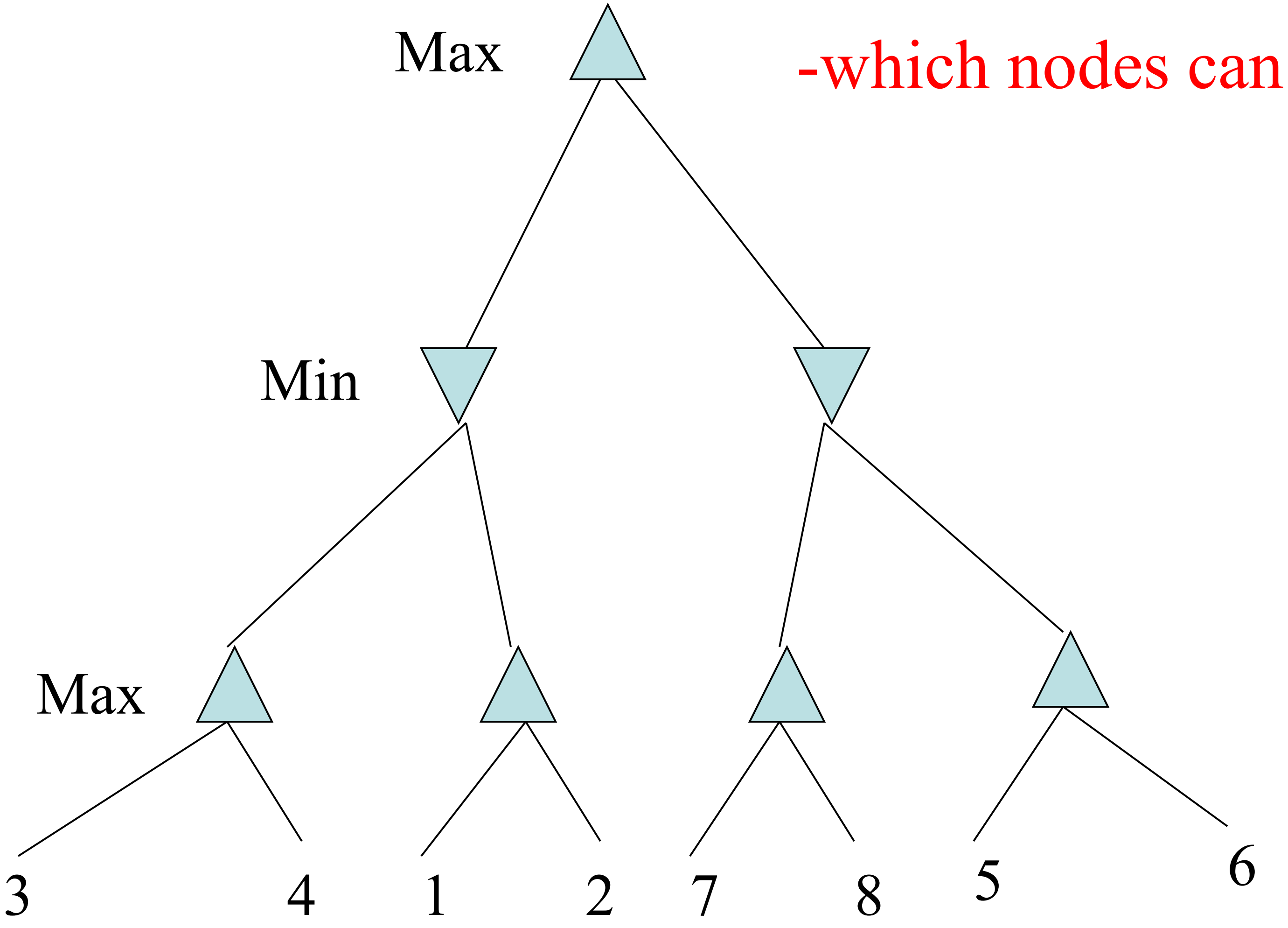


Example

-which nodes can be pruned?



Answer to Example

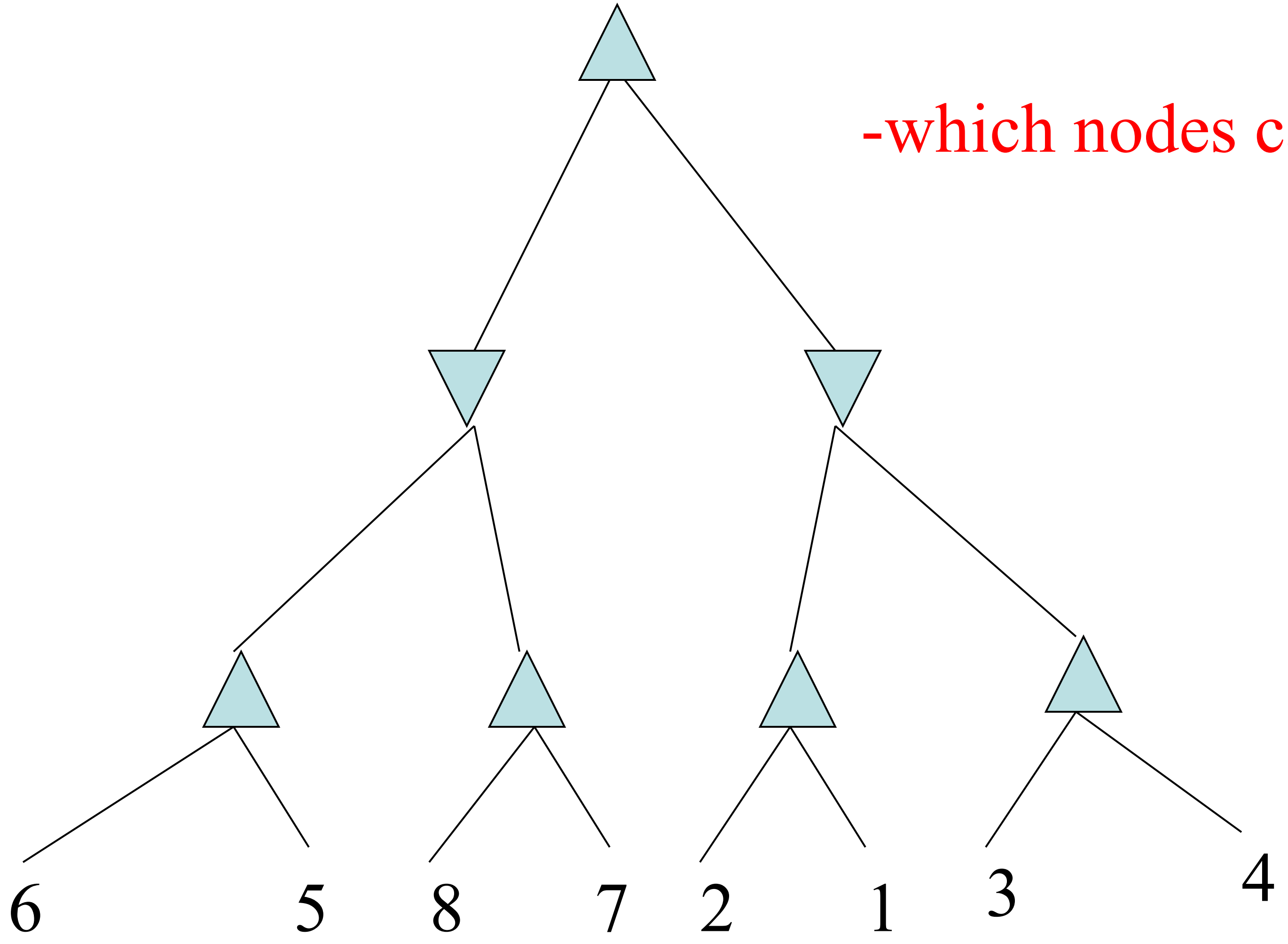


-which nodes can be pruned?

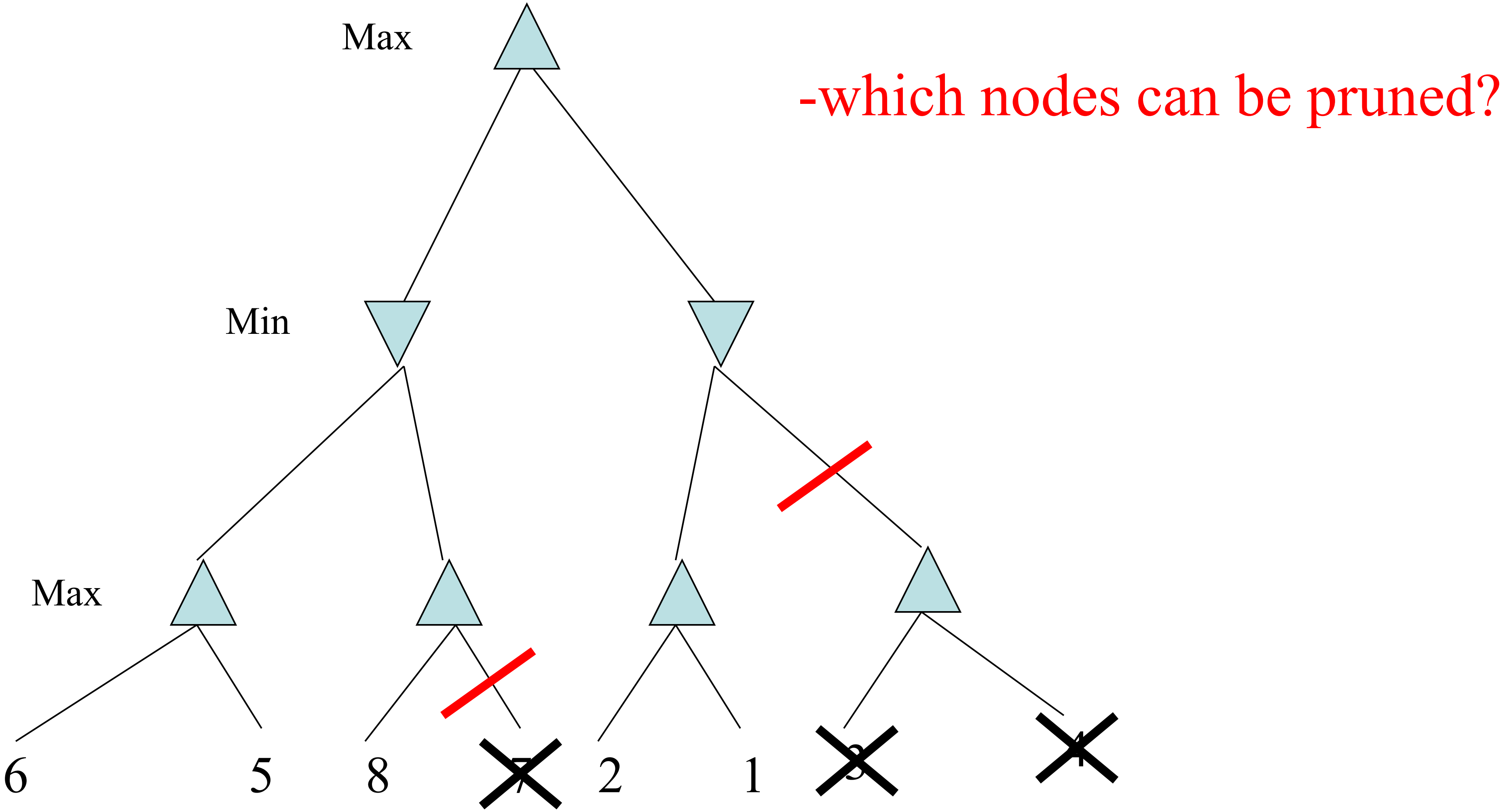
Answer: NONE! Because the most favorable nodes for both are explored last (i.e., in the diagram, are on the right-hand side).

Second Example

-which nodes can be pruned?



Answer to Second Example



Answer: LOTS! Because the most favorable nodes for both are explored first (i.e., in the diagram, are on the left-hand side).

Deep Blue Algorithm

- AlphaBeta Pruning
- **Opening** Database of opening moves
- **Endgame** Database of all positions with five or fewer pieces

AlphaBeta in Go

- **GNUGo**
- **In CGOS，GNUGo benchmark**
- **业余5～10级左右**

- http://www.gnu.org/software/gnugo/

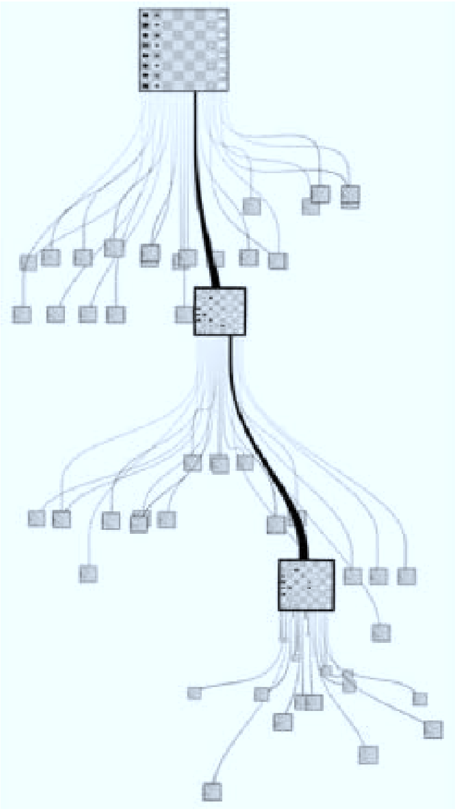
```
    White (O) has captured 0 pieces
    Black (X) has captured 0 pieces

    A B C D E F G H J K L M N O P Q R S T
 19 . . . . . . . . . . . . . . . . . . . 19
 18 . . . . . . . . . . . . . . . . . . . 18
 17 . . . . . . . . . . . . . . . . . . . 17
 16 . . . + . . . . . + . . . . . + . . . 16
 15 . . . . . . . . . . . . . . . . . . . 15
 14 . . . . . . . . . . . . . . . . . . . 14
 13 . . . . . . . . . . . . . . . . . . . 13
 12 . . . . . . . . . . . . . . . . . . . 12
 11 . . . . . . . . . . . . . . . . . . . 11
 10 . . . + . . . . . + . . . . . + . . . 10
  9 . . . . . . . . . . . . . . . . . . . 9
  8 . . . . . . . . . . . . . . . . . . . 8
  7 . . . . . . . . . . . . . . . . . . . 7
  6 . . . . . . . . . . . . . . . . . . . 6
  5 . . . . . . . . . . . . . . . . . . . 5
  4 . . . + . . . . . + . . . . . + . . . 4
  3 . . . . . . . . . . . . . . . . . . . 3
  2 . . . . . . . . . . . . . . . . . . . 2
  1 . . . . . . . . . . . . . . . . . . . 1
    A B C D E F G H J K L M N O P Q R S T

black(1):
```

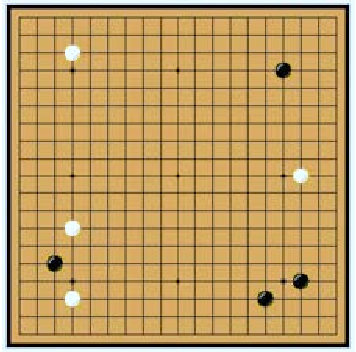
Go vs. Chess

- Go has long been viewed as one of most complex game and most challenging of classic games for AI.



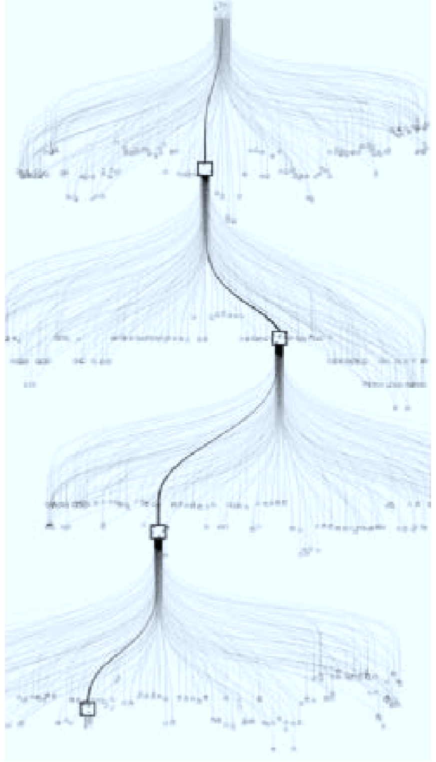
Chess ($b \approx 35$, $d \approx 80$)

$8 \times 8 = 64$, possible games$\approx 10^{120}$



Go ($b \approx 250$, $d \approx 150$)

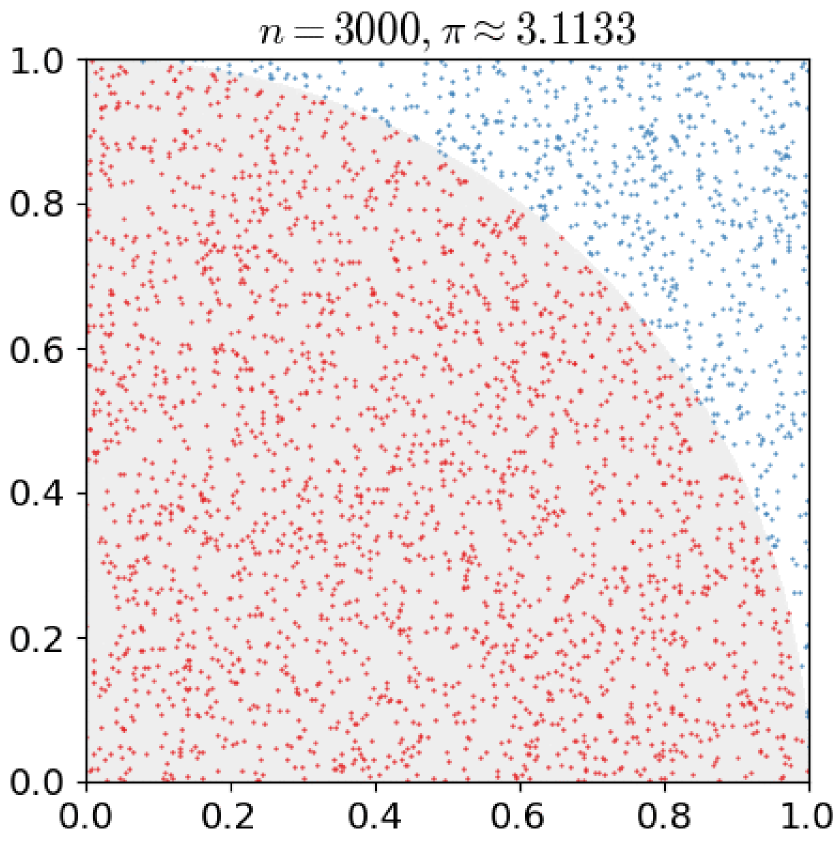
$19 \times 19 = 361$, possible games$\approx 10^{170}$



Monte Carlo Tree Search

MCTS

- Rely on repeated random sampling to obtain numerical results.
- *Example*: Approximating π by Monte Carlo Method



Given that circle and square have a ratio of areas that is $\pi/4$, the value of π can be approximated using a Monte-Carlo method:

1) Draw a square on the ground, then inscribe a circle within it.
2) Uniformly scatter some objects of uniform size over the square.
3) Count the number of objects inside the circle and the square.
4) The ratio of the two counts is an estimate of the ratio of the two areas, which is $\pi/4$. Multiply the result by 4 to estimate π.

```python
import random

N = 50000
count = 0   # 将count当作Nr（即落入四分之一圆内的点）
for i in range(0, N):
    x = random.uniform(0, 1)
    y = random.uniform(0, 1)
    if (x*x + y*y)<1:
        count = count+1

pi = 4*count/N
print("当模拟落点", N, "次时,pi的值为：", pi)
```

Run: trypi

```
D:\anaconda3\python.exe E:/browser_down/expert-system-master/expert-system-master/trypi.py
当模拟落点 50000 次时,pi的值为： 3.1464

进程已结束，退出代码 0
```

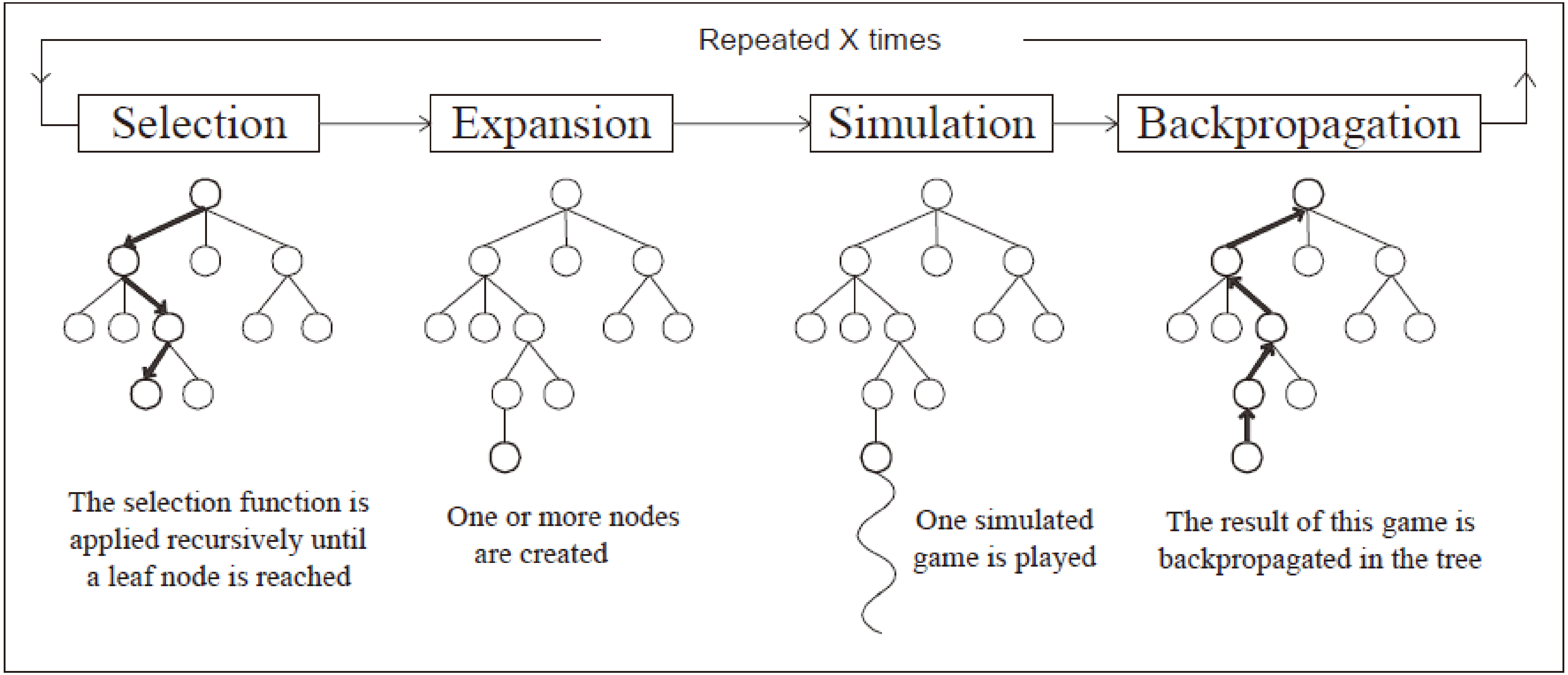
```
D:\anaconda3\python.exe E:/browser_down/expert-s
当模拟落点 100000 次时,pi的值为： 3.14608
```

```
D:\anaconda3\python.exe E:/browser_down/expert-sy
当模拟落点 1000000 次时,pi的值为： 3.14346
```

```
D:\anaconda3\python.exe E:/browser_down/expert-sys
当模拟落点 10000000 次时,pi的值为： 3.1418616
```

MCTS

- Four Steps to develop the game tree



MCTS 的具体步骤

1. Selection（选择）
   – 从根节点开始，根据某种策略依次选择最佳的子节点，直到到达叶子节点。选择节点的好坏直接影响搜索的好坏，目前广泛采用的策略是UCT算法（Upper Confidence Bound Apply to Tree）
2. Expansion（扩展）
   – 扩展叶子节点，将一个或多个可行的落子添加为该叶子节点的子节点。
3. Simulation（模拟）
   – 根据某种策略（比如围棋中的完全随机落子）从扩展的位置进行到游戏结束。模拟总是会产生一个结果，对于围棋类游戏来说就是获胜、失败或平局，但是广义上来说模拟的合法结果可以是任意值。
4. Backpropagation（反向传播）
   – 将模拟的结果沿着传递路径反向传递回根节点。

UCT算法（Upper Confidence Bound applied to Trees）

“Selection is the strategic task that selects one of the children of a given node. It controls the balance between exploitation and exploration.”

Exploitation(利用)：给定过去的经验选择能期望产生好的回报的动作。

Exploration(探索)：尝试可能能够使得在未来做出更好决策的新事物。

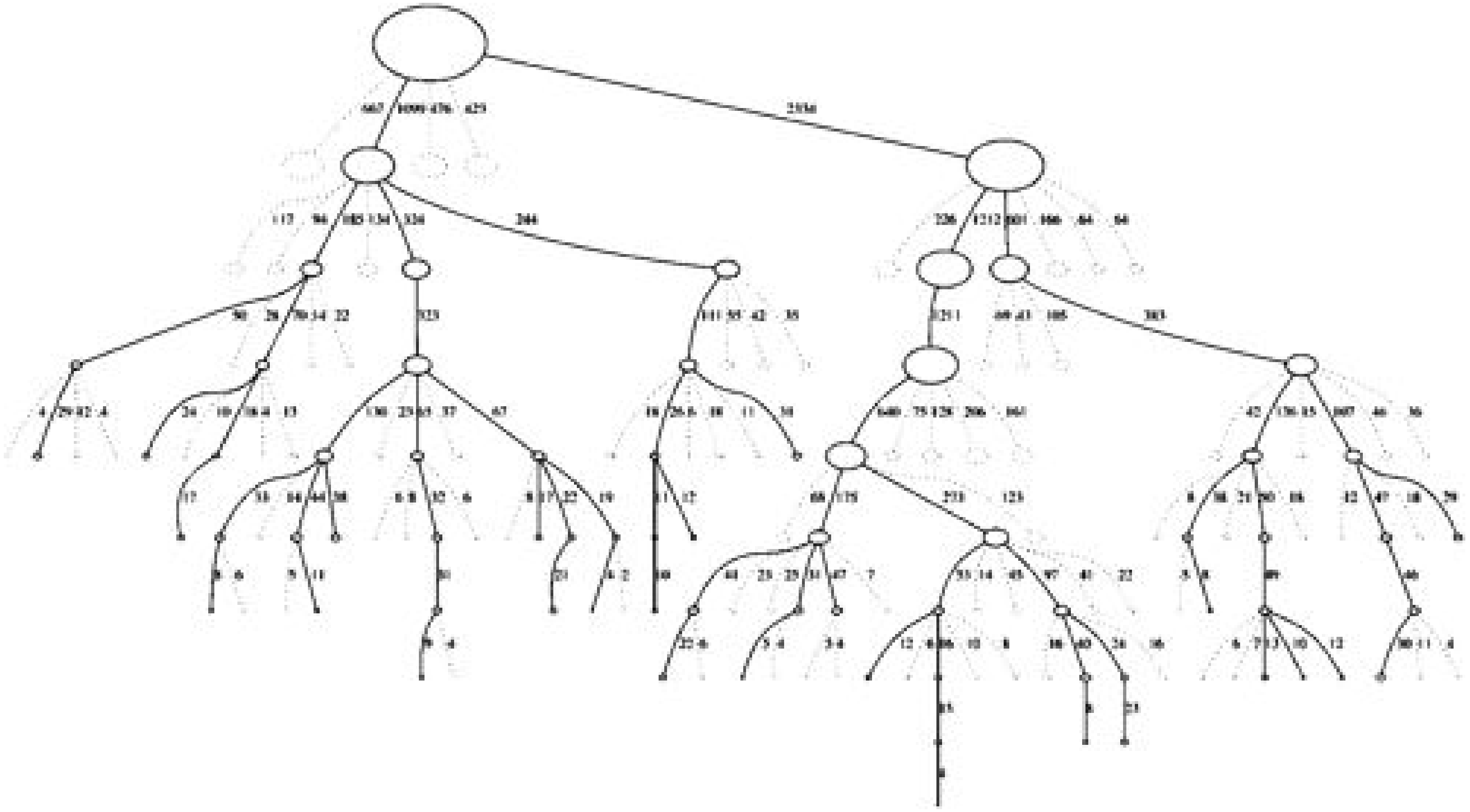
在选择节点的时候，我们目前来讲当然应该考虑收益较高的节点；但同时也要考虑那些由于被探测数量少，暂时收益不高，但在未来很有希望的节点.

$$\mathrm{UCT}(v_i, \mathrm{v}) = \frac{Q(v_i)}{N(v_i)} + \mathrm{c}\sqrt{\frac{\ln(N(v))}{N(v_i)}}$$

公式的第一部分表示截至目前v_i节点平均每次的收益

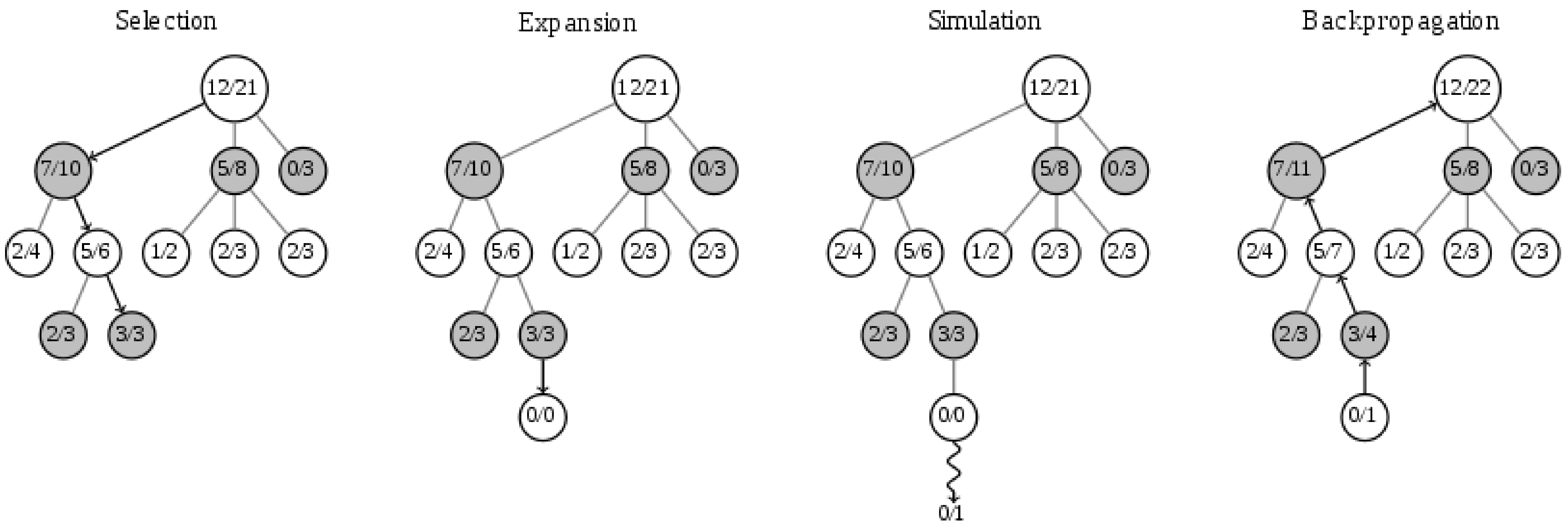
公式的第二部分则倾向于那些相对较少被探索的节点

Asymmetric（非对称的建树过程）



将更多算力用于探索未来发展更加优秀的分支上

wiki上的一个MCTS例子



每个节点（代表不同的局面）都有两个值，代表这个节点以及它的子节点模拟的次数和赢的次数，比如模拟了21次，赢了12次，记为 12/21。

这两个值也分别对应着最原始MCTS中的Q(v)以及N(v)——进行一局比赛N(v)+1；赢一局Q(v)+1，否则不变

- Score(7/10)=uct(7/10，12/21)= 7/10 + C$\sqrt{\frac{\ln(21)}{10}}$ = 0.7 + 0.55C
- Score(5/8)=uct(5/8，12/21)= 5/8 + C$\sqrt{\frac{\ln(21)}{5}}$ = 0.625 + 0.62C
- Score(0/3)=uct(0/3，12/21)= 0/3 + C$\sqrt{\frac{\ln(21)}{3}}$ = 0 + 1.00C

…..

c越大越倾向于广度搜索，也就是探索有潜力的节点；c越小越倾向于深度搜索，也就是多访问在当前已知信息下，平均奖励最高的节点。

- Score(7/10)=uct(7/10，12/21)= 7/10 + C$\sqrt{\frac{\ln(21)}{10}}$ = 0.7 + 0.55C
- Score(5/8)=uct(5/8，12/21)= 5/8 + C$\sqrt{\frac{\ln(21)}{5}}$ = 0.625 + 0.62C
- Score(0/3)=uct(0/3，12/21)= 0/3 + C$\sqrt{\frac{\ln(21)}{3}}$ = 0 + 1.00C

…..

c越大越倾向于广度搜索，也就是探索有潜力的节点；c越小越倾向于深度搜索，也就是多访问在当前已知信息下，平均奖励最高的节点。

MCTS in Go

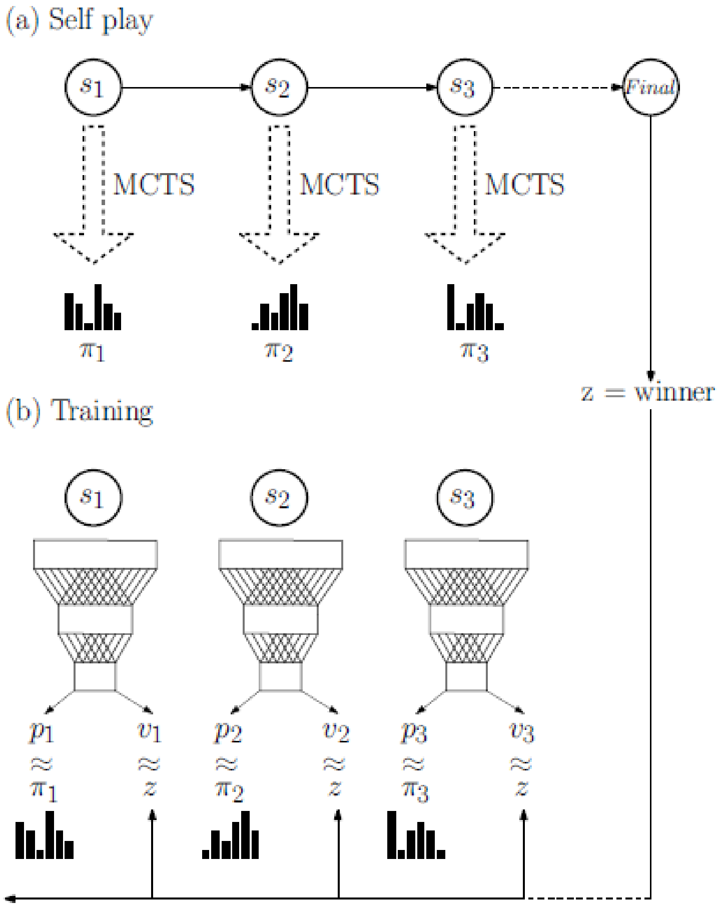
- **MoGo第一个使用蒙特卡洛树搜索的围棋程序(2006年)，在9×9的棋盘上击败了职业选手**
- **DeepZenGo是AlphaGo之前最强的围棋程序之一，可以达到与职业棋士差距3~4子的水平**

Algorithm of AlphaGo

- Deep neural networks
  - value networks: used to evaluate board positions
  - policy networks: used to select moves.
- Monte-Carlo tree search (MCTS)
  - Combines Monte-Carlo simulation with value networks and policy networks.
- Reinforcement learning
  - used to improve its play

*Source:*Mastering Go with deep networks and tree search
Nature, Jan. 28, 2016

Algorithm of AlphaGo Zero



- MCTS to generate the training set through self-play
- Neural Network
- State, Move Distribution and Winner

Compared

软件或人类	BayesElo
AlphaGo Zero（40 blocks版）	5422?
AlphaGo（Master版）	5231?
AlphaGo Zero（20 blocks版）	5022?
AlphaGo（Lee版）	4672?
朴廷桓	4592?
柯洁	4590?
井山裕太	4546?
李世乭	4514?
DeepZenGo	4269
AlphaGo（Fan版，176 GPU）	4122?
AlphaGo（Fan版，48 CPU与8 GPU）	3862?
GNU Go	1800

From AlphaGo to AlphaGo Zero and AlphaZero

Nature (2017)

Mastering the game of Go without human knowledge

David Silver[1]*, Julian Schrittwieser[1]*, Karen Simonyan[1]*, Ioannis Antonoglou[1], Aja Huang[1], Arthur Guez[1], Thomas Hubert[1], Lucas Baker[1], Matthew Lai[1], Adrian Bolton[1], Yutian Chen[1], Timothy Lillicrap[1], Fan Hui[1], Laurent Sifre[1], George van den Driessche[1], Thore Graepel[1] & Demis Hassabis[1]

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here we introduce an algorithm based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo's own move selections and also the winner of AlphaGo's games. This neural network improves the strength of the tree search, resulting in higher quality move selection and stronger self-play in the next iteration. Starting *tabula rasa*, our new program AlphaGo Zero achieved superhuman performance, winning 100–0 against the previously published, champion-defeating AlphaGo.

AlphaGo Zero

Science

Current Issue First release papers Archive About Submit manusc

HOME > SCIENCE > VOL. 362, NO. 6419 > A GENERAL REINFORCEMENT LEARNING ALGORITHM THAT MASTERS CHESS, SHOGI, AND GO THROUGH SELF-PLAY

REPORT

A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play

DAVID SILVER , THOMAS HUBERT, JULIAN SCHRITTWIESER, IOANNIS ANTONOGLOU, MATTHEW LAI, ARTHUR GUEZ, MARC LANCTOT, LAURENT SIFRE, DHARSHAN KUMARAN , [...] DEMIS HASSABIS +4 authors Authors Info & Affiliations

SCIENCE • 7 Dec 2018 • Vol 362, Issue 6419 • pp. 1140-1144 • DOI: 10.1126/science.aar6404

One program to rule them all

AlphaZero

"AlphaZero's creative insights coupled with the encouraging results we see in other projects such as AlphaFold, give us confidence in our mission to create general purpose learning systems that will one day help us find novel solutions to some of the most important and complex scientific problems."

https://deepmind.com/blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go

AlphaFold

nature

Explore content ⌄ About the journal ⌄ Publish with us ⌄

nature › articles › article

Article | Published: 15 January 2020

Improved protein structure prediction using potentials from deep learning

Andrew W. Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander W. R. Nelson, Alex Bridgland, Hugo Penedones, Stig Petersen, Karen Simonyan, Steve Crossan, Pushmeet Kohli, David T. Jones, David Silver, Koray Kavukcuoglu & Demis Hassabis

Nature **577**, 706–710 (2020) | Cite this article

DeepMind

nature

Explore content ⌄ About the journal ⌄ Publish with us ⌄

nature › articles › article

Article | Open Access | Published: 01 December 2021

Advancing mathematics by guiding human intuition with AI

Alex Davies, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, Marc Lackenby, Geordie Williamson, Demis Hassabis & Pushmeet Kohli

Nature **600**, 70–74 (2021) | Cite this article

计算机科学家和数学家们首次使用AI来帮助证明或提出新的数学定理，包括复杂理论中的纽结理论和表象理论。

AlphaTensor

Discovering faster matrix multiplication algorithms with reinforcement learning

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis & Pushmeet Kohli

AlphaTensor 建立在 AlphaZero 的基础上，它是第一个可用于为矩阵乘法等基本任务发现新颖、高效且可证明正确的算法的人工智能系统。

Reward is enough?

ABSTRACT

In this article we hypothesise that intelligence, and its associated abilities, can be understood as subserving the maximisation of reward. Accordingly, reward is enough to drive behaviour that exhibits abilities studied in natural and artificial intelligence, including knowledge, learning, perception, social intelligence, language, generalisation and imitation. This is in contrast to the view that specialised problem formulations are needed for each ability, based on other signals or objectives. Furthermore, we suggest that agents that learn through trial and error experience to maximise reward could learn behaviour that exhibits most if not all of these abilities, and therefore that powerful reinforcement learning agents could constitute a solution to artificial general intelligence.

Artificial Intelligence 299 (2021) 103535

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Reward is enough

David Silver*, Satinder Singh, Doina Precup, Richard S. Sutton

ARTICLE INFO

Article history:
Received 12 November 2020
Received in revised form 28 April 2021
Accepted 12 May 2021
Available online 24 May 2021

Keywords:
Artificial intelligence
Artificial general intelligence
Reinforcement learning
Reward

ABSTRACT

In this article we hypothesise that intelligence, and its associated abilities, can be understood as subserving the maximisation of reward. Accordingly, reward is enough to drive behaviour that exhibits abilities studied in natural and artificial intelligence, including knowledge, learning, perception, social intelligence, language, generalisation and imitation. This is in contrast to the view that specialised problem formulations are needed for each ability, based on other signals or objectives. Furthermore, we suggest that agents that learn through trial and error experience to maximise reward could learn behaviour that exhibits most if not all of these abilities, and therefore that powerful reinforcement learning agents could constitute a solution to artificial general intelligence.

1. Introduction

Expressions of intelligence in animal and human behaviour are so bountiful and so varied that there is an ontology of associated abilities to name and study them, e.g. social intelligence, language, perception, knowledge representation, planning, imagination, memory, and motor control. What could drive agents (natural or artificial) to behave intelligently in such a diverse variety of ways?

One possible answer is that each ability arises from the pursuit of a goal that is designed specifically to elicit that ability. For example, the ability of social intelligence has often been framed as the Nash equilibrium of a multi-agent system; the ability of language by a combination of goals such as parsing, part-of-speech tagging, lexical analysis, and sentiment analysis; and the ability of perception by object segmentation and recognition. In this paper, we consider an alternative hypothesis: that the generic objective of maximising reward is enough to drive behaviour that exhibits most if not all abilities that are studied in natural and artificial intelligence.

This hypothesis may startle because the sheer diversity of abilities associated with intelligence seems to be at odds with any generic objective. However, the natural world faced by animals and humans, and presumably also the environments faced in the future by artificial agents, are inherently so complex that they require sophisticated abilities in order to succeed (for example, to survive) within those environments. Thus, success, as measured by maximising reward, demands a variety of abilities associated with intelligence. In such environments, any behaviour that maximises reward must necessarily exhibit those abilities. In this sense, the generic objective of reward maximisation contains within it many or possibly even all the goals of intelligence.

Reward thus provides two levels of explanation for the bountiful expressions of intelligence found in nature. First, different forms of intelligence may arise from the maximisation of different reward signals in different environments, resulting for example in abilities as distinct as echolocation in bats, communication by whale-song, or tool use in chimpanzees. Sim-

* Corresponding author.
E-mail address: davidsilver@google.com (D. Silver).

https://doi.org/10.1016/j.artint.2021.103535

General Game Playing

- **General game playing (GGP)** is the design of artificial intelligence programs to be able to play more than one game successfully.

- Unlike specialised game players (e.g. Deep Blue, AlphaGo), they do not use algorithms designed in advance for specific games.

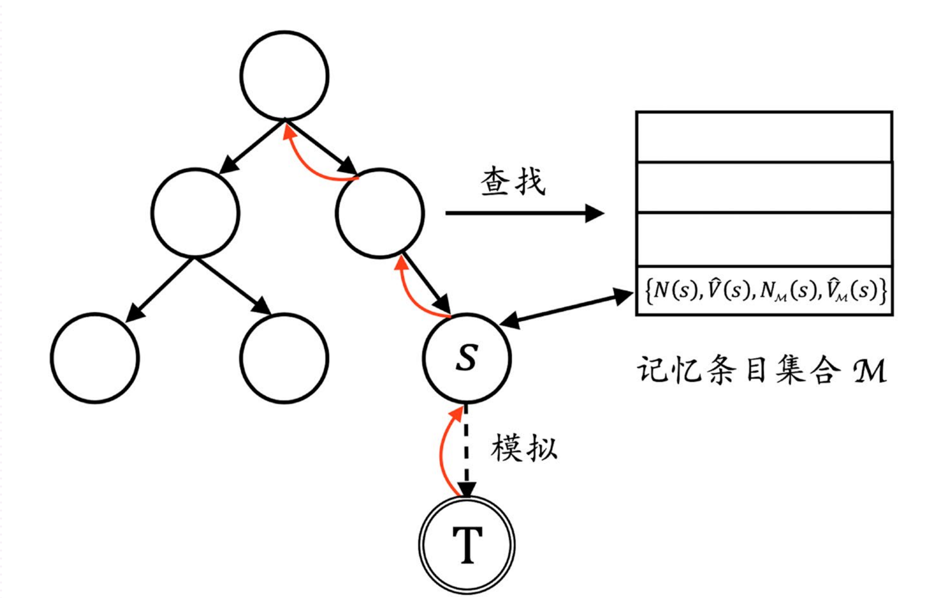
General Game Playing Contest
@AAAI since 2005

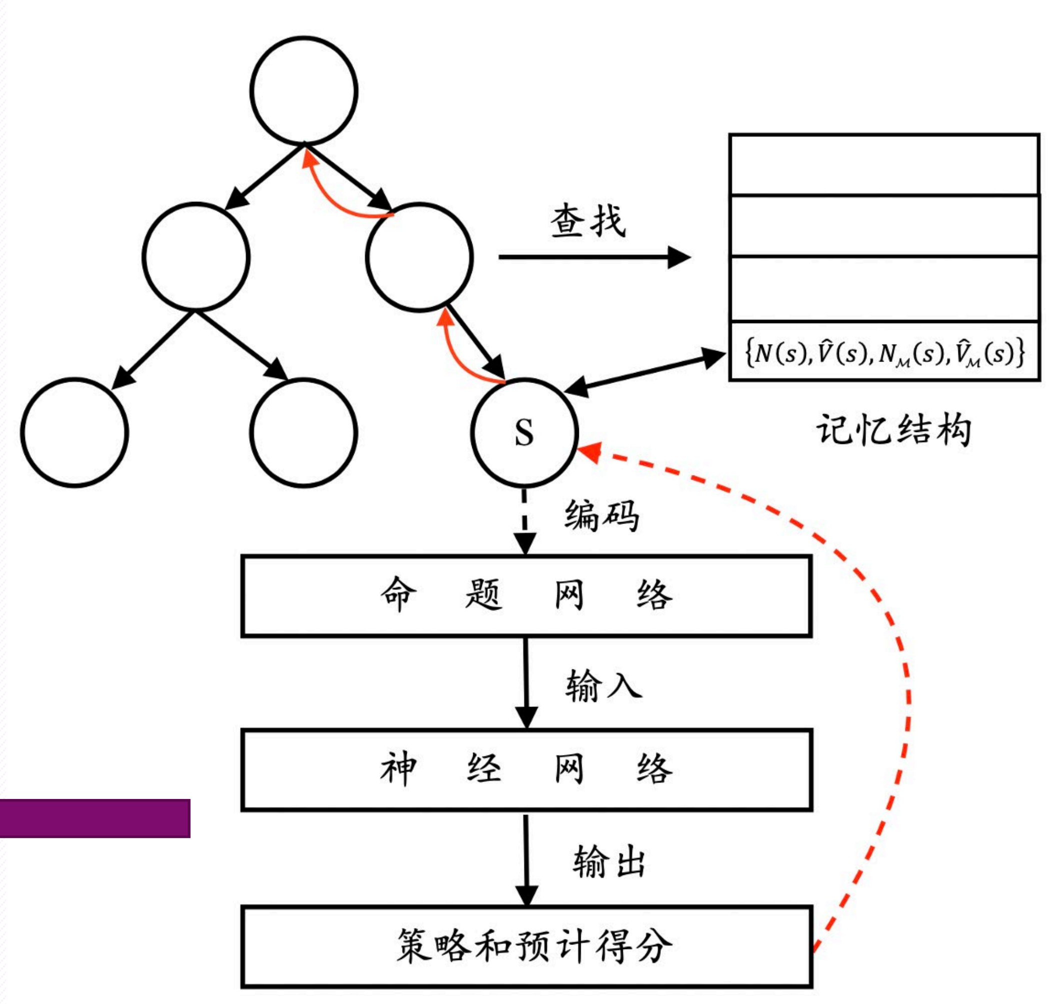
General Game Playing

- Rather than being concerned with a specialized solution to a narrow problem, General Game Playing encompasses a variety of AI areas:
  - Game Playing
  - Knowledge Representation
  - Planning and Search
  - Learning
  - ……

General Game Playing is considered a grand AI Challenge

研究成果





游戏名称	玩家数目	合作与否	信息	行动顺序
Connect 4 (6×7 棋盘)	2	零和	信息对称	回合制
Breakthrough (6×6 棋盘)	2	零和	信息对称	回合制
Babel	3	合作	信息对称	同时
Pacman3p (6×6 棋盘)	3	合作/零和	非信息对称	回合制/同时混合

Liang, S., **Jiang, G.,** Zhang, Y., Combining M-MCTS and Deep Reinforcement Learning for General Game Playing, DAI-2021.

Research Topics

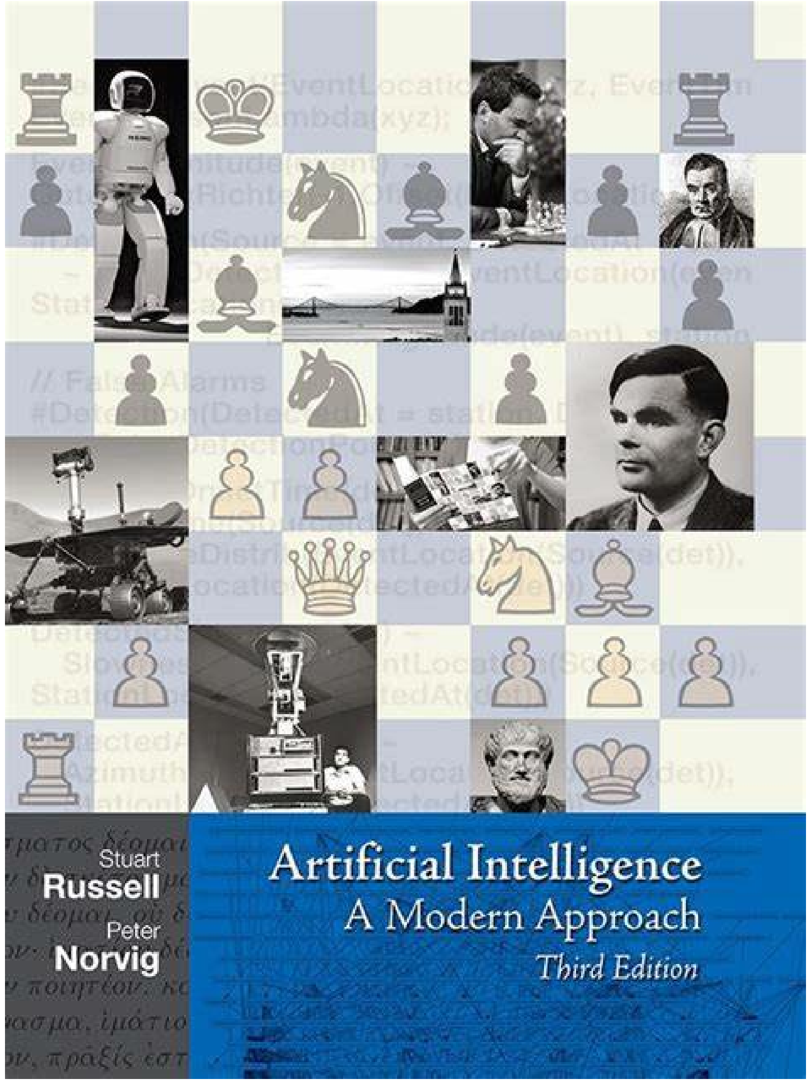
1. Modelling Strategic Reasoning
   - Game Description, strategy representation and reasoning
2. Strategy Generation for General Game Player
   - Adversarial Search
   - Monte Carlo Tree Search
   - Deep Reinforcement Learning
3. Building General Auction Player
   - ANR project University of Toulouse

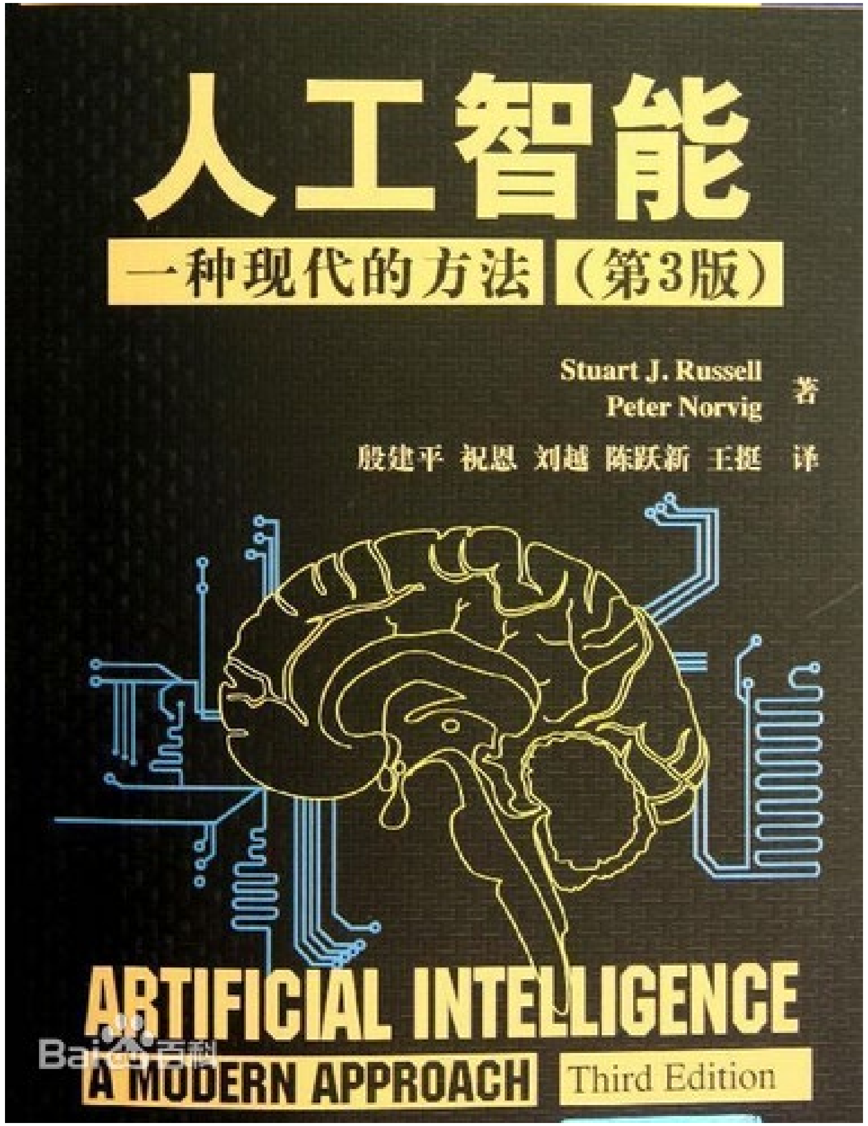
PRICAI-2014, IJCAI-2016, LORI-2017, PRICAI-2019, Artificial Intelligence 2021， ECAI-2023

Summary

- Adversarial Search Methods
  - Minimax Search
  - Alpha-Bata Pruning
- Monte-Carlo Tree Search
- Generalized Reinforcement Learning

Reference


Stuart
Russell
Peter
Norvig
Artificial Intelligence
A Modern Approach
Third Edition


人工智能
一种现代的方法（第3版）
Stuart J. Russell
Peter Norvig
著
殷建平 祝恩 刘越 陈跃新 王挺 译
ARTIFICIAL INTELLIGENCE
A MODERN APPROACH
Third Edition