Artificial Intelligence

School of Electronic and Computer Engineering
Peking University

Wang Wenmin

Contents:

Contents:

- ☐ 3. Solving Problems by Search
- ☐ 4. Local Search and Swarm Intelligence
- ☐ 5. Adversarial Search
- ☐ 6. Constraint Satisfaction Problems

Objectives 教学目的

- To examine the problems that arise when we try to plan ahead in a world where other agents are planning against us.

  去考察这样一些会发生的问题，当我们试图在某个环境预先规划时，其他智能体也正在针对我们做规划。

5. Adversarial Search

Contents:

Search vs. Adversarial Search 搜索与对抗搜索

Search 搜索	Adversarial Search 对抗搜索
Single agent 单智能体	Multiple agents 多智能体
Solution is (heuristic) method for finding goal. 解是寻找目标的（启发式）方法	Solution is strategy (strategy specifies move for every possible opponent reply). 解是策略（指定对每个可能对手回应的行动策略）
Heuristics can find *optimal* solution. 启发式法可以找到最优解	Time limits force an *approximate* solution. 时间受限被迫执行一个近似解
Evaluation function: estimate of cost from start to goal through given node. 评价函数：穿过给定节点从起始到目标的代价估计	Evaluation function: evaluate “goodness” of game position. 评价函数：评估博弈局势的“好坏”

Adversarial Search often Known as Games 对抗搜索通常称为博弈

- Definitions of Game theory 博弈论的定义
  - Study of strategic decision making. Specifically, study of mathematical models of conflict and cooperation between intelligent rational decision-makers.
    研究战略决策制定。具体来说，研究智能理性决策者之间的冲突与合作的数学模型。
  - An alternative term is interactive decision theory.
    一个可替代的术语是交互式决策理论。
- Applications of Game theory 博弈论的应用
  - Economics, political science, psychology, logic, computer science, and biology.
    经济学、政治学、心理学、逻辑、计算机科学、以及生物学。
  - Behavioral relations and decision science, including both humans and non-humans (e.g. computers).
    行为关系与决策科学，包括人类与非人类（如计算机等）。

Games are Good Problems for AI 博弈是AI研究的好材料

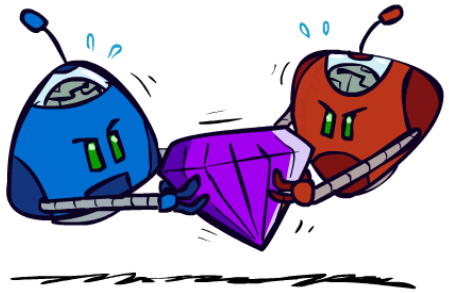
- Machines (players) need “human-like” intelligence.
  机器（玩家）需要“类人”的智能。
- Requiring to make decision within limited time.
  要求在有限的时间内进行决策。
- Features of games: 博弈的特征：

Two, or more players (agents)	■	两个、或多个玩家（智能体）
Turn-taking vs. simultaneous moves	■	轮流、与同步行动
Perfect information vs. imperfect information	■	完全信息、与不完全信息
Deterministic vs. stochastic	■	确定性、与随机
Cooperative vs. competitive	■	合作式、与对抗式
Zero-sum vs. non zero-sum	■	零和、与非零和

Zero Sum vs. Non-zero Sum 零和与非零和博弈

Zero sum games 零和博弈

- Agents have *opposite utilities*.
  智能体之间是对立的方式。
- Pure competition: win-lose, its sum is zero.
  纯竞争：输赢、其和为零。



Non-zero sum games 非零和博弈

- Agents have *independent* utilities.
  智能体之间是自主的方式。
- Cooperation, indifference, competition, ...
  合作、中立、竞争、…
- Win-win, win-lose or lose-lose, its sum is not zero.
  双赢、输赢、或双输，其和不为零。

Example: Prisoner's Dilemma 囚徒困境

- Two members of a criminal gang are arrested and imprisoned. Each prisoner is given the opportunity either to: betray the other by testifying that the other committed the crime, or to cooperate with the other by remaining silent. Here is the offer:

  有两个犯罪集团的成员被逮捕和监禁。每个囚徒只有二选一的机会：揭发对方并证明其犯罪，或者与对方合作保持沉默。惩罚方式如下：

  - If A and B each betray the other, each of them serves 2 years in prison.

    若A和B彼此揭发对方，则每个囚徒监禁2年。

  - If A betrays B but B remains silent, A will be set free and B will serve 3 years in prison (and vice versa).

    若A揭发B而B保持沉默，则A被释放而B监禁3年（反之亦然）。

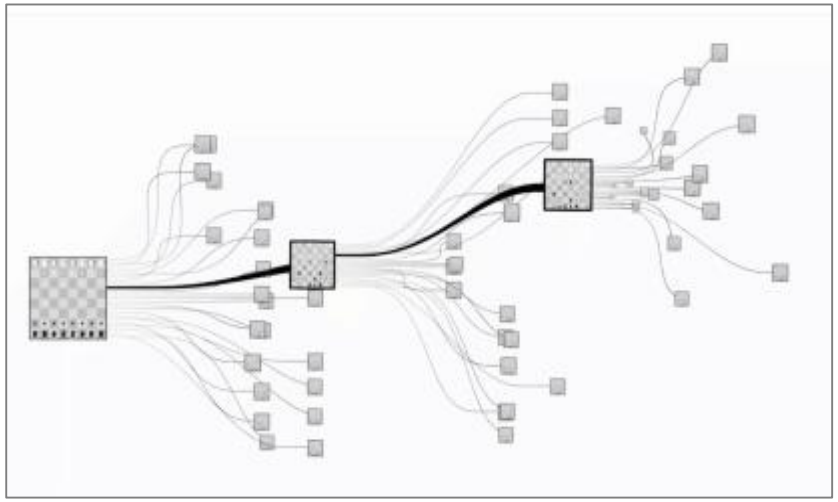
  - If A and B both remain silent, both of them will only serve 1 year in prison.

    若A和B都保持沉默，则他们仅被监禁1年。

Games are Interesting but Too Hard to Solve 博弈有趣但难以求解

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!

例如，国际象棋：平均分支数约等于35，每个对弈者常常走50多步，故该搜索树约有35^{100}或10^{154}个节点！



- Games, like the real world, therefore require the ability to make *some* decision even when calculating the *optimal* decision is infeasible.

  博弈，与现实世界相似，因而当无法算计出最优决策时，需要某种决策的能力。

- Game-playing research has spawned a number of interesting ideas on how to make the best possible use of time.

  博弈的研究已经产生了大量的有趣思想，即如何尽可能的利用时间。

Types of Games 博弈的类型

	Deterministic 确定性	Stochastic 随机性
Perfect information (fully observable) 完全信息 （可完全观测）	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

(a) Checkers
西洋跳棋

(b) Othello
黑白棋

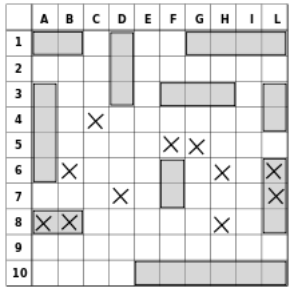
(c) Backgammon
西洋双陆棋

(d) Monopoly
大富翁

Types of Games 博弈的类型

	Deterministic 确定性	Stochastic 随机性
Imperfect information (partially observable) 不完全信息 （不可完全观测）	Stratego 西洋陆军棋 Battleships 海战棋	Bridge 桥牌 Poker 扑克 Scrabble 拼字游戏

(e) Stratego
西洋陆军棋



(f) Battleships
海战棋

(g) Scrabble
拼字游戏

Origins of Game Playing Algorithms 博弈算法的起源

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 Today 博弈算法的进展

Computers are better than humans 计算机优于人类

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 Lee Sedol, a 9 dan professional in Mar. 2016 谷歌AlphaGo于2016年3月战胜了9段职业棋手李世乭

Computers are competitive with top human players 计算机与顶级人类玩家媲美

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 and alpha-beta search 顶级的系统使用蒙特卡罗仿真和alpha-beta搜索

Two Players Games 两个玩家博弈

- Feature 特点

  deterministic, perfect information, turn-taking, two players, zero-sum.

  确定性、完整信息、轮流、两个玩家、零和。

- Calling the two players: 将两个玩家称为：

  MAX, MIN.

- MAX moves first, and then they take turns moving, until the game is over.

  MAX先走棋，然后轮流走棋，直到博弈结束。

- At game end 博弈结束时
  - winner: award points

    胜者：奖励点数
  - loser: give penalties.

    败者：给予处罚

Formally Defined as a Search Problem 形式化定义为搜索问题

S_0 Initial state, specifies how the game is set up at the start.

初始状态，指定博弈开始时的设定。

PLAYER(s) Defines which player has the move in a state.

定义哪个玩家在某状态下动作。

ACTIONS(s) Returns the set of legal moves in a state.

返回某个状态下的合法动作。

RESULT(s, a) Transition model, defines the result of a move.

转换模型，定义一步动作的结果。

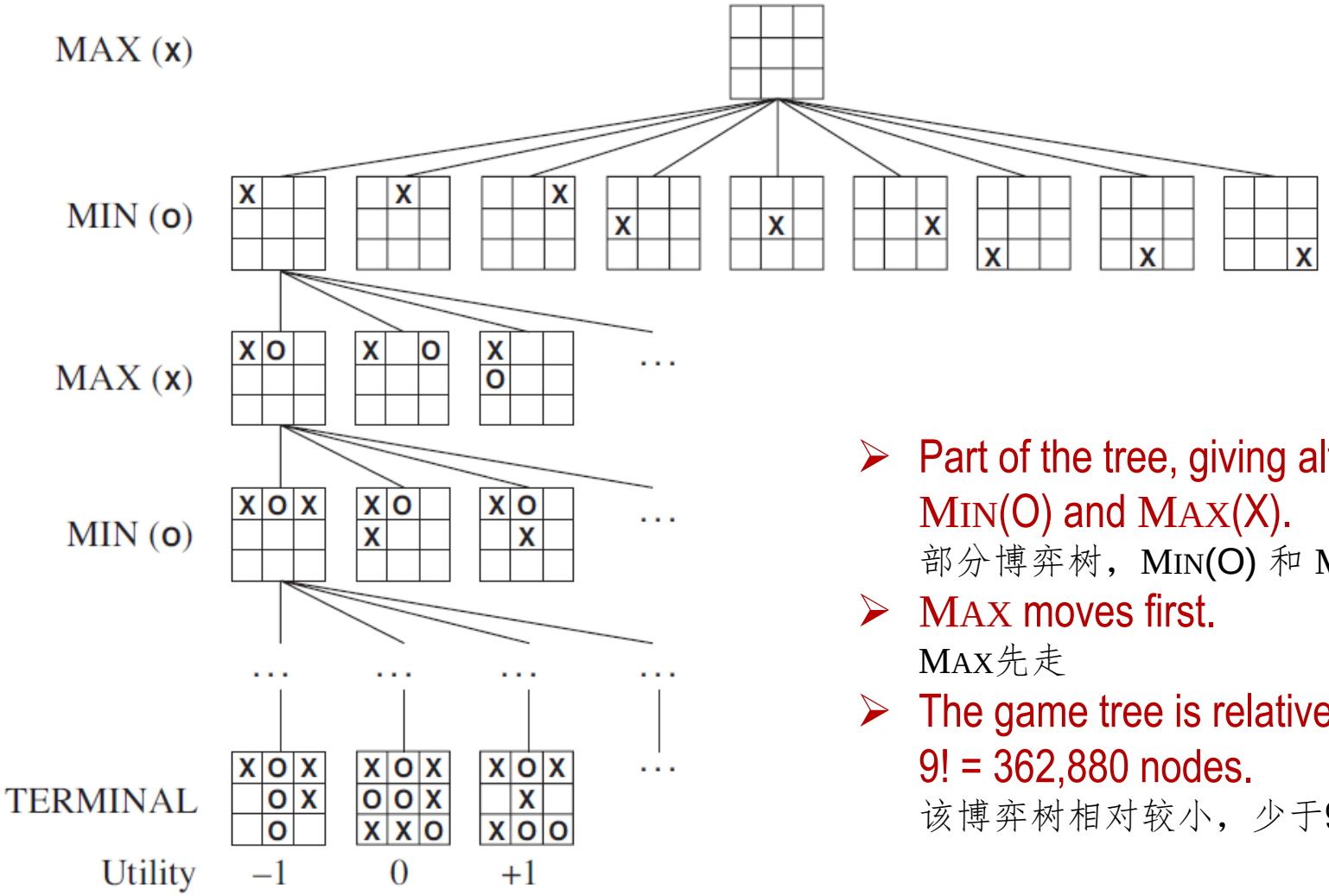
TERMINAL-TEST(s) Terminal test, *true* when the game is over and *false* otherwise.

终止检测，博弈结束时为*true*，否则为*false*。

UTILITY(s, p) Utility function, defines the value in state s for a player p.

效用函数，定义在状态s、玩家为p的值。

Example: Game Tree of Tic-tac-toe 井字棋的博弈树



- Part of the tree, giving alternating moves by MIN(O) and MAX(X).
  部分博弈树，MIN(O) 和 MAX(X)之间交替走子
- MAX moves first.
  MAX先走
- The game tree is relatively small, fewer than 9! = 362,880 nodes.
  该博弈树相对较小，少于9! = 362,880个节点

5. Adversarial Search

Contents:

Optimal Solution 最优解

- In normal search 普通搜索
  - The optimal solution would be a sequence of actions leading to a goal state (terminal state) that is a win.
    最优解将是导致获胜的目标状态（终端状态）的一系列动作。
- In adversarial search 对抗搜索
  - Both of MAX and MIN could have an optimal strategy.
    MAX和MIN都会有一个最优策略。
    - In initial state, MAX must find a strategy to specify MAX's move,
      在初始状态，MAX必须找到一个策略来确定MAX的动作，
    - then MAX's moves in the states resulting from every possible response by MIN, and so on.
      然后MAX针对MIN的每个合理的对应采取相应的动作，以此类推。

Minimax Theorem 最小最大定理

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

对于两个玩家、具有有限多个策略的零和博弈，每个玩家存在一个值V和一个混合策略，使得：

(a) Given player 2's strategy, the best payoff possible for player 1 is V,

给定玩家2的策略，则玩家1可能的最好收益是V，

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

给定玩家1的策略，则玩家2可能的最好收益是-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*.

对于零和博弈来说，其名称minimax的由来是因为每个玩家会使对手可能的最大收益变得最小，还会使自己的最大损失变得最小。

Optimal Solution in Adversarial Search 对抗搜索的最优解

- Given a game tree, the optimal strategy can be determined from the minimax value of each node, write as MINIMAX(n).

  给定一棵博弈树，则最优策略可以由每个节点的minimax值来确定，记作MINIMAX(n)。

- Assume that both players play optimally from there to the end of the game.

  假设两个玩家博弈自始至终都发挥得很好。

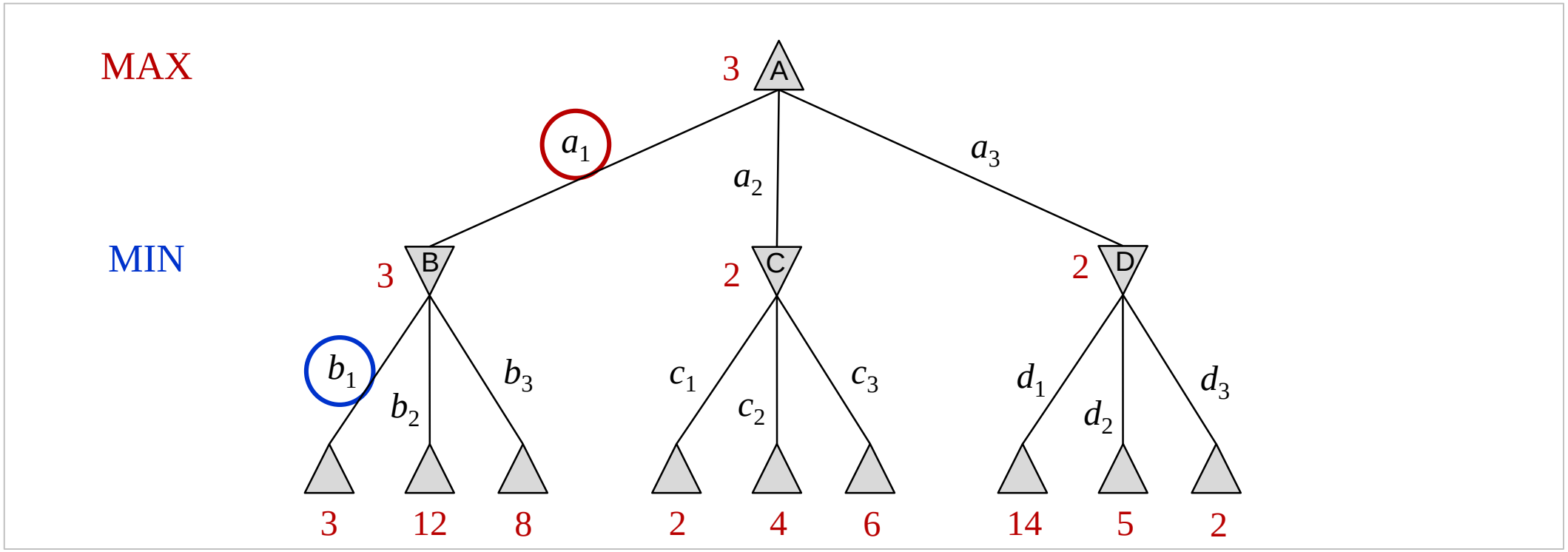
function MINIMAX(s) **returns** an action
  if TERMINAL-TEST(s) **then return** UTILITY(s)
  if PLAYER(s) = MAX **then return** $\max_{a \in \text{ACTIONS}(s)}$ MINIMAX(RESULT(s, a))
  if PLAYER(s) = MIN **then return** $\min_{a \in \text{ACTIONS}(s)}$ MINIMAX(RESULT(s, a))

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.

终端状态的minimax值只是其效用。MAX倾向于移动到一个最大值状态，MIN则倾向于一个最小值状态。

Minimax Decision -- A Two-player Game Tree 一个双人玩家的博弈树



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

根节点处MAX的最佳移动是a_1（具有最高的minimax值）

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

B节点处MIN的最佳应对是b_1（具有最低的minimax值）

Minimax Algorithm 最小最大算法

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*

Properties of Minimax Decision 最小最大决策的性质

The minimax algorithm performs a depth-first exploration of the game tree.

最小最大算法表现为博弈树的深度优先探索。

- Time complexity 时间复杂性

$$O(b^m)$$

- Space complexity 空间复杂性

$O(bm)$ -- The algorithm generates all actions at once

算法同时生成所有动作

$O(m)$ -- The algorithm generates actions one at a time

算法一次生成一个动作

where

- b -- The branching factor (legal moves at each point)

  分支因子（每个点的合法走子）

- m -- The maximum depth of any node

  任一节点的最大深度

Optimal Decisions in Multi-player Games 多玩家博弈中的最优决策

- Let us examine how to extend the minimax idea to multiplayer games.

  让我们考察一下如何将minimax思想扩展到多玩家博弈中。

- We need to replace single value for each node with a *vector* of values.

  我们需要将每个节点的单一值替换为一个值的向量。

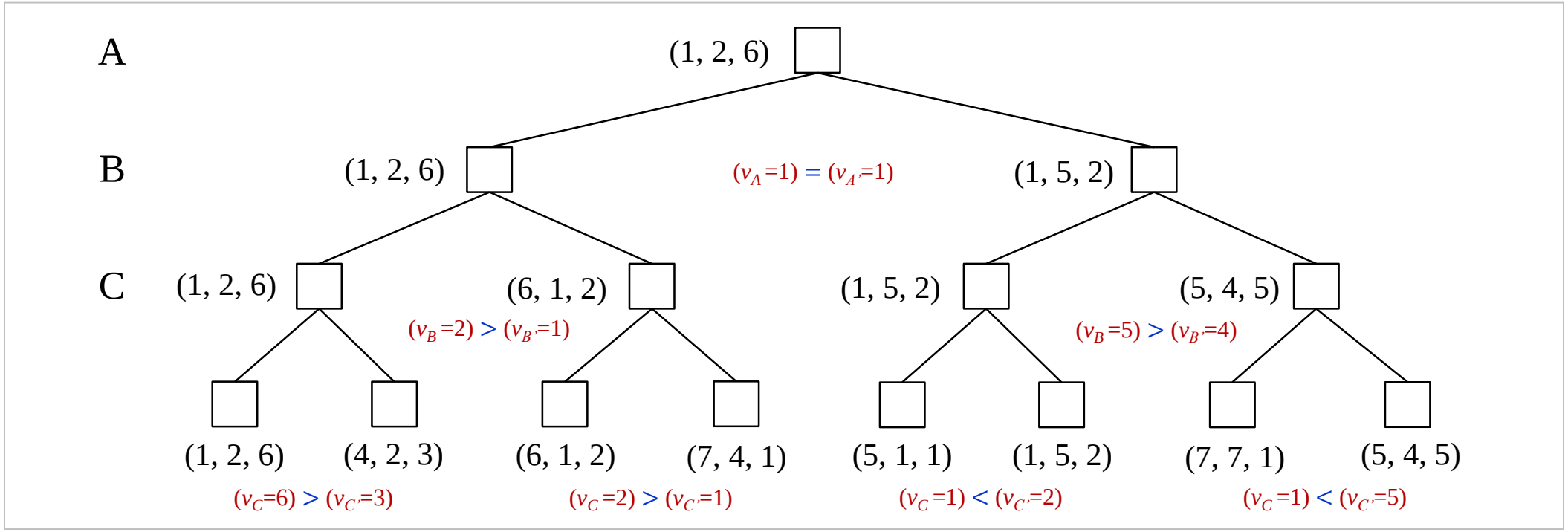
  - E.g., in a three-player game with players A, B, and C, a vector (v_A, v_B, v_C) is associated with each node.

    例如，对于具有A、B、C三个玩家的博弈，将向量(v_A, v_B, v_C)与每个节点相关联。

- For terminal states, this vector gives the utility of the state from each player's viewpoint. The simplest way to implement this is to have the UTILITY function return a vector of utilities.

  对于终端状态，从每个玩家的角度来看，这个向量给定该状态的效用。实现的最简单方法是拥有一个返回效用向量的UTILITY函数。

Optimal Decisions in Multi-player Games 多玩家博弈中的最优决策



Multiplayer games usually involve formal or informal alliances among the players.

多玩家博弈通常在玩家之间涉及正式的或非正式的联盟。

Alliances are made and broken as the game proceeds.

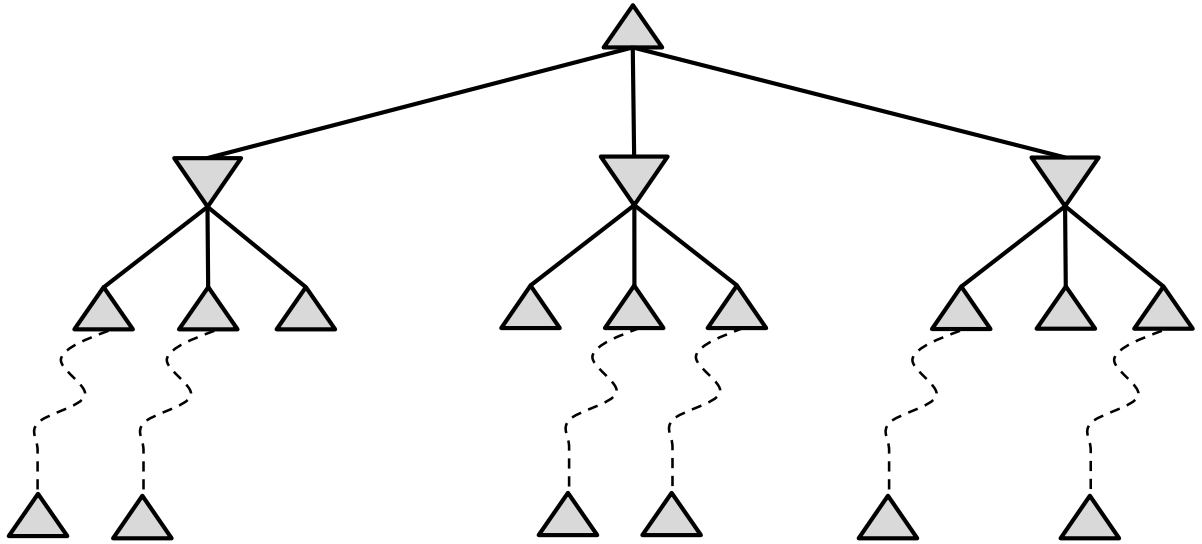
联盟随着博弈收益的变化建立或者破裂。

5. Adversarial Search

Contents:

Overview 概述

- The problem with minimax search: 采用minimax搜索的问题：
  - Number of game states is exponential in depth of the tree.
    博弈状态的量随着树的深度呈现指数式增长。
  - We can't eliminate the exponent, but we can effectively cut it in half.
    我们无法消除这种指数，但实际上我们可以将它剪掉一半。



Overview 概述

- The trick to solve the problem: 解决该问题的技巧
  - Compute correct minimax decision without looking at every node in game tree.
    计算正确的minimax决策而不考虑博弈树的每个节点。
  - That is, use “pruning” to eliminate large parts of the tree.
    就是说，采用“剪枝”方法来消除该树的大部分。
- What is alpha–beta pruning 什么是alpha–beta剪枝
  - It is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm.
    是一种搜索算法，旨在削减由minimax算法评价的节点数量。

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)

Why Called Alpha-Beta 为何称其为Alpha-Beta

- Alpha–beta pruning gets its name from the following two parameters:

  Alpha-Beta剪枝从如下两个参数得到其名称：

  - α: highest-value we have found so far at any point along the path for MAX.

    α：沿着MAX路径上的任意选择点，迄今为止我们已经发现的最高值。

  - β: lowest-value we have found so far at any point along the path for MIN.

    β：沿着MIN路径上的任意选择点，迄今为止我们已经发现的最低值。

- Alpha–beta search respectively: Alpha-Beta搜索依次完成如下动作：

  - updates the values of α and β as it goes along, and

    边搜索边更新α和β的值，并且

  - 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.

    一旦得知当前节点的值比当前MAX或MIN的α或β值更差，则在该节点剪去其余的分枝。

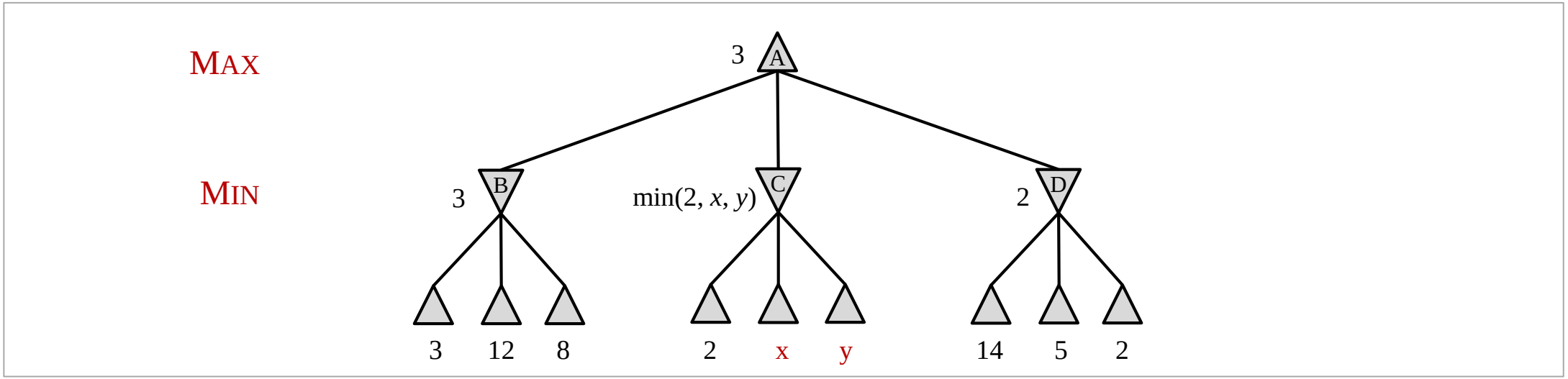
Alpha-Beta Search Algorithm Alpha-Beta搜索算法

```
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: Game Tree Using Minimax 采用Minimax的博弈树

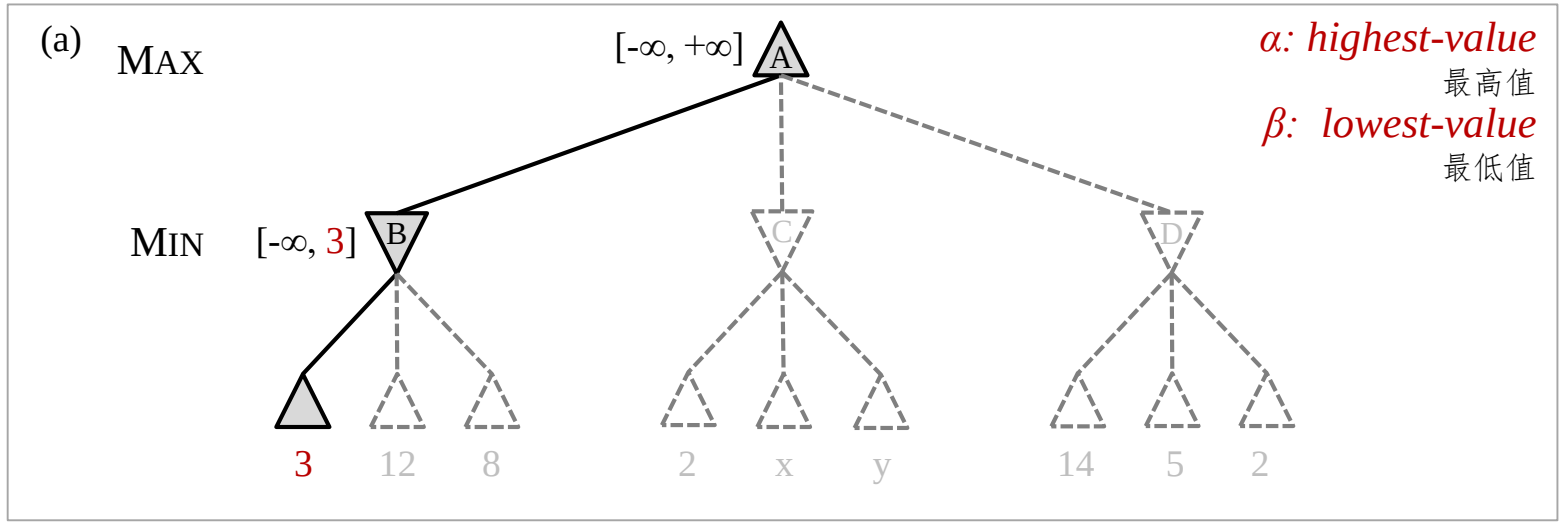


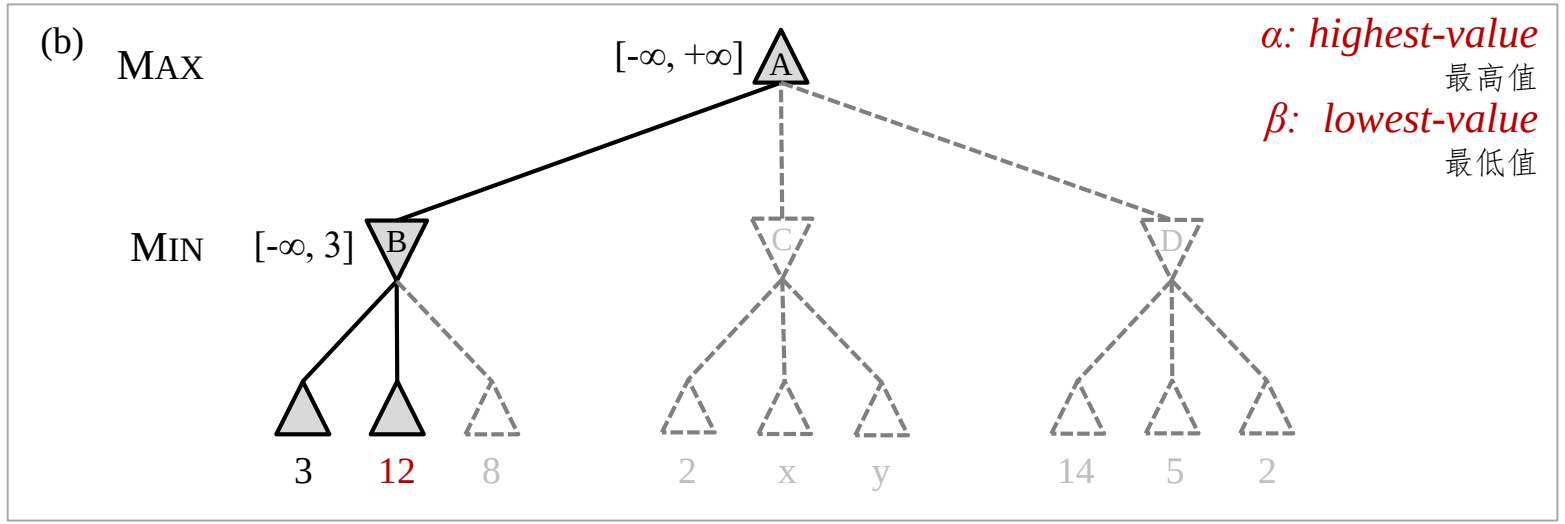
- The value of root node is given by: 根节点的值由如下方法得出：

$$\begin{aligned} \text{MINIMAX}(root) &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\ &= \max(3, \min(2, x, y), 2) \\ &= \max(3, z, 2) \quad \text{where } z = \min(2, x, y) \le 2 \\ &= 3. \end{aligned}$$

no pruning!
无剪枝！

Example: Game Tree Using Alpha-Beta Pruning 采用Alpha-Beta剪枝的博弈树





Initial value: 初始值：

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

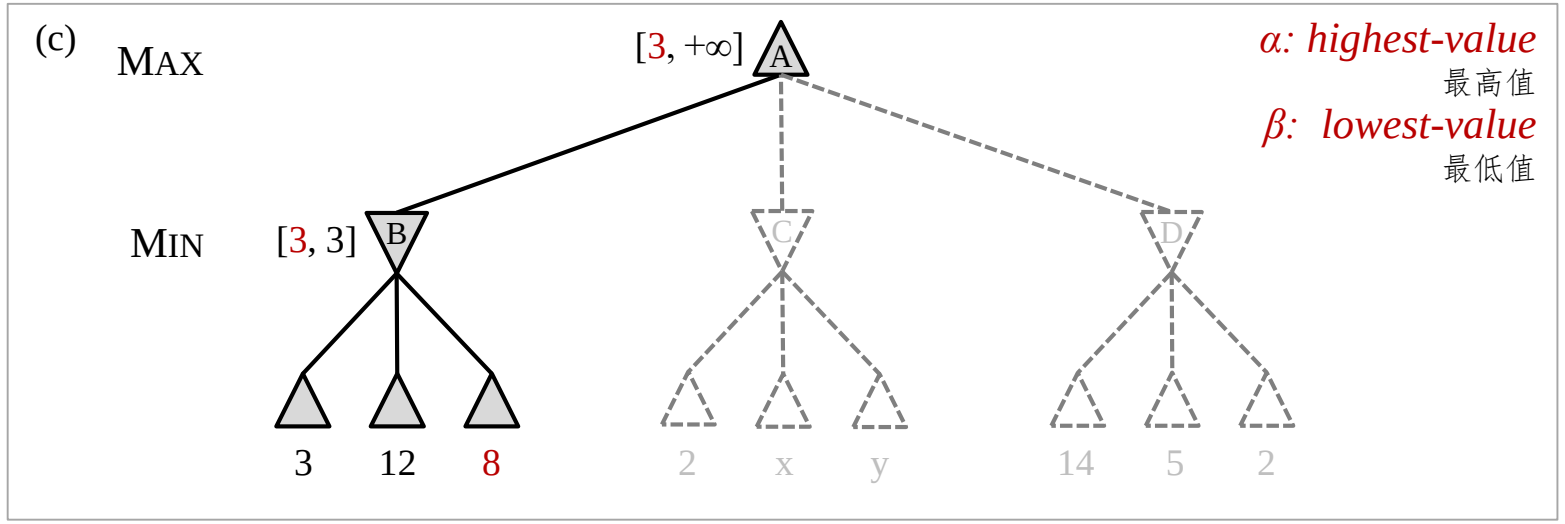
(a) The 1st leaf below *B* has the value 3. Hence, *B*, as a MIN node, $B[\beta=3]$.

*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 $B[\beta=3]$.

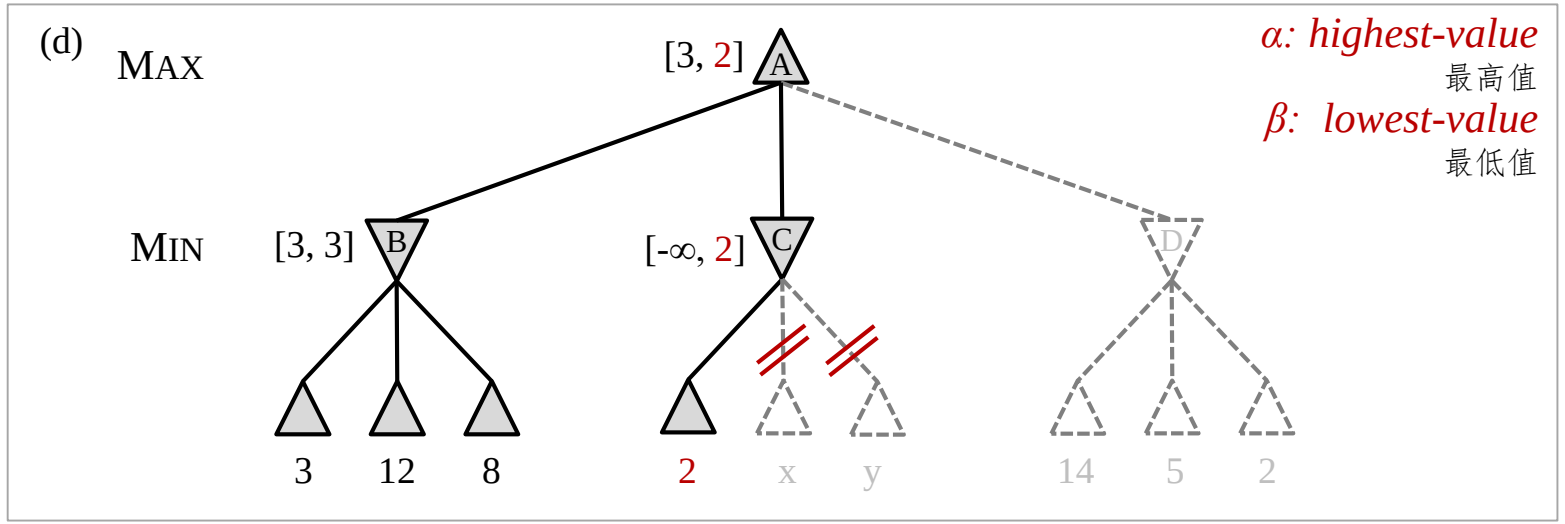
*B*下面第二个叶节点的值为12； MIN将回避这个移动，仍然是$B[\beta=3]$。

Example: Game Tree Using Alpha-Beta Pruning 采用Alpha-Beta剪枝的博弈树



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

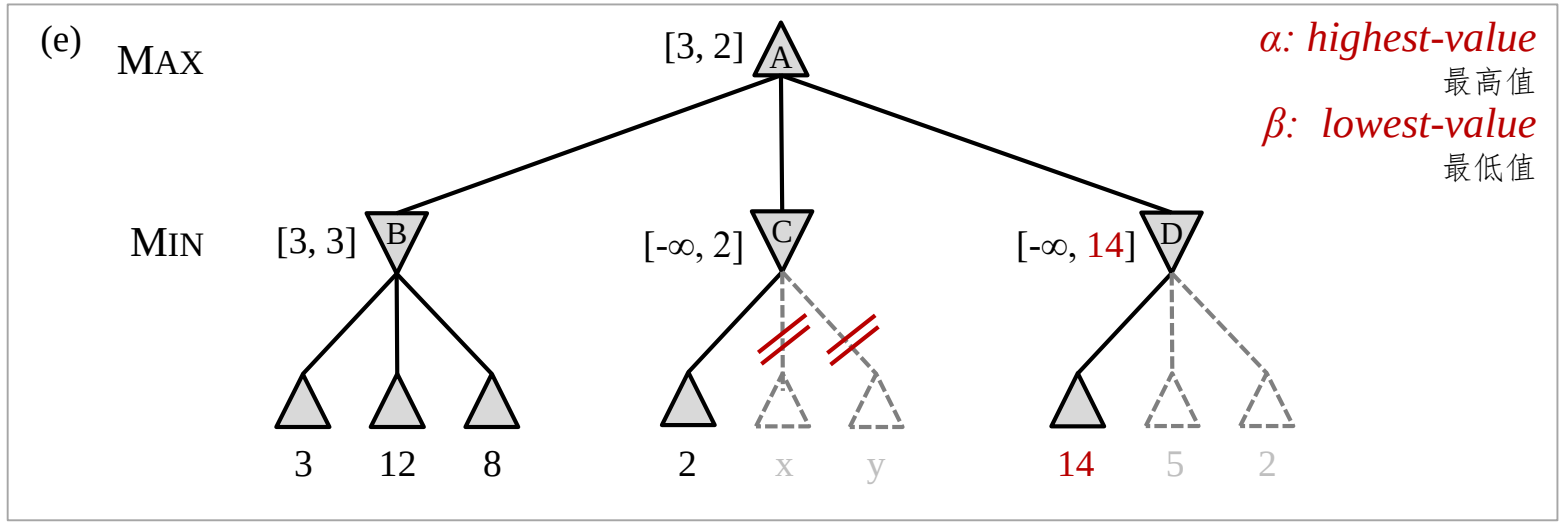
*B*下面第三个叶节点的值为8；故MIN节点正是*B*[β=3]。现在，因为MAX为*A*[$\alpha \geq 3$]，我们能够推出*B*[α=3]。



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

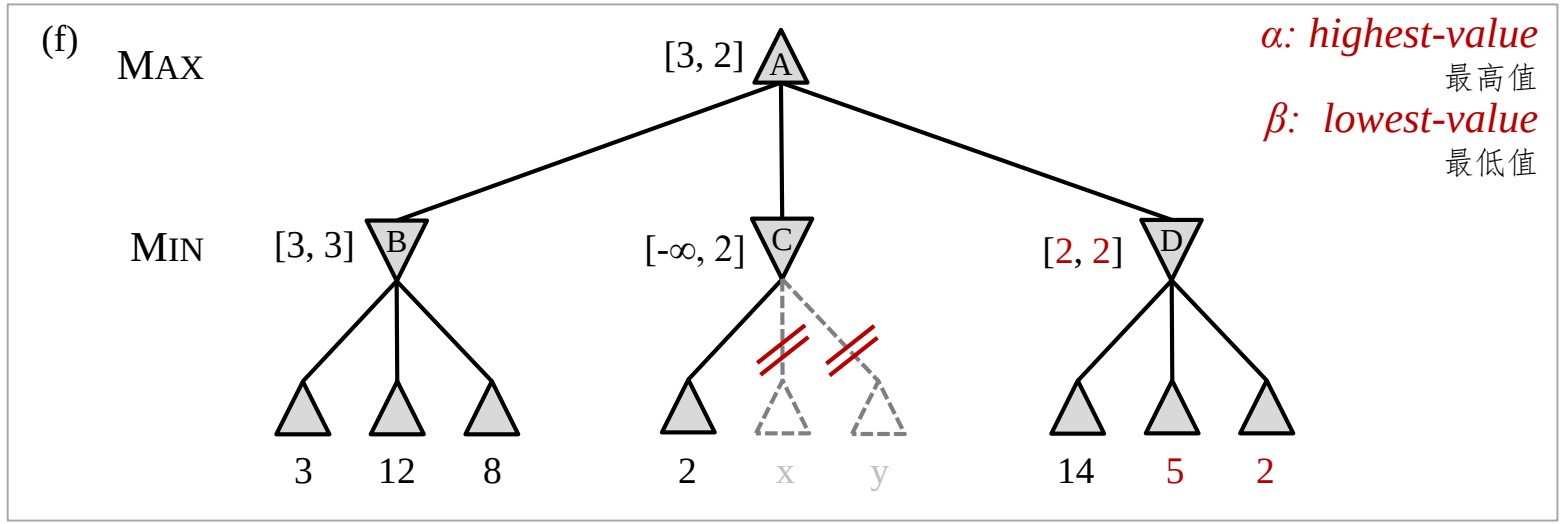
C下面第一个叶节点的值为2，因此，由于MIN节点*C*[β=2], 且*B*[β=3]>*C*[β=2]，故MAX将不会选择C，所以只需剪掉C的所有后继节点 (*α–β* pruning)。

Example: Game Tree Using Alpha-Beta Pruning 采用Alpha-Beta剪枝的博弈树



(e) The 1st leaf below *D* is 14, $D[\beta \leq 14]$, so we need to keep exploring *D*'s successor states. We now have bounds on all of root's successors, so $A[\beta \leq 2]$.

*D*下面第一个叶节点为14，$D[\beta \leq 14]$，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$A[\beta \leq 2]$。



(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剪枝的一般原则

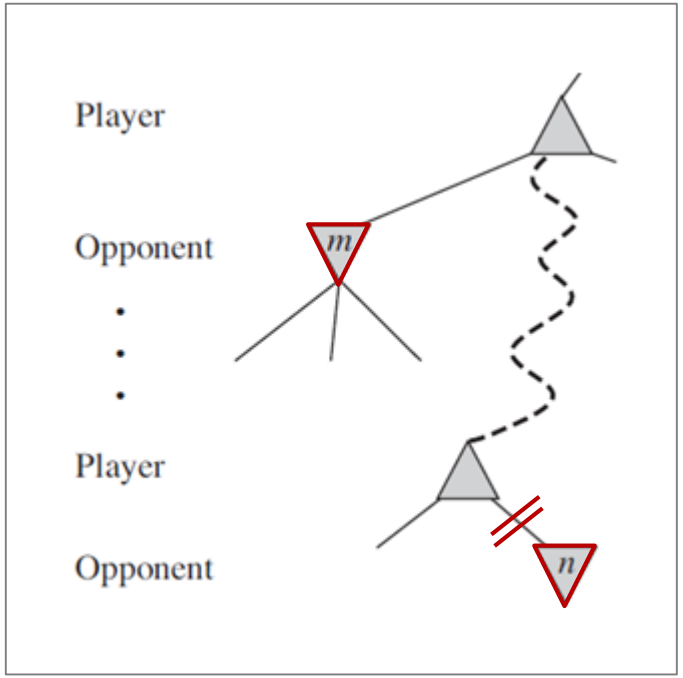
- Alpha–beta pruning can be applied to trees of any depth, and often possible to prune entire subtrees rather than just leaves.

  Alpha-Beta剪枝可被用于任意深度的树，并且常常可以剪去整个子树而不仅仅是叶节点。

- The general principle: 一般原则：
  - Consider a node *n* somewhere in the tree, such that Player has a choice of moving to that node.

    设某个节点*n*位于树的某处，于是玩家选择移向那个节点。
  - 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.

    若玩家在位于*n*的父节点或更上层处有更好的选择*m*，则在实战中完全没必要抵达*n*。



5. Adversarial Search

Contents:

Overview 概述

- The minimax algorithm
  generates the entire game search space.
  minimax算法生成整个博弈搜索空间。
- The alpha–beta pruning algorithm
  allows us to prune large parts of it.
  alpha–beta剪枝算法允许我们将其剪去大部分。
- However, alpha–beta still has to search all the way to terminal states for at least a portion of the search space.
  然而，alpha–beta仍然需要搜索抵达终端状态的所有途径、至少是搜索空间的一部分。
- This depth is usually not practical, because moves must be made in a reasonable amount of time.
  这个深度通常是不实际的，因为移动必须在合理的时间内完成。

Apply a Heuristic Evaluation Function 应用启发式评价函数

- Claude Shannon proposed instead that programs should cut off the search earlier and apply a heuristic evaluation function to states in the search, effectively turning nonterminal nodes into terminal leaves. The suggestion is in two ways:

  克劳德·香农提出：程序应该早一些剪断搜索，并在搜索中对状态应用启发式评估函数，有效地将非终端节点转换为终端叶节点。该建议是用如下两种方法：

  - Use EVAL instead of UTILITY

    用EVAL来代替UTILITY

    EVAL: a heuristic evaluation function, which estimates the position's utility.

    EVAL：一个启发式评价函数，用于估计位置的效用。

  - Use CUTOFF-TEST instead of TERMINAL-TEST

    用CUTOFF-TEST来代替TERMINAL-TEST

    CUTOFF-TEST: decides when to apply EVAL.

    CUTOFF-TEST：确定何时应用EVAL函数。

Heuristic H-Minimax vs. Minimax 启发式H-Minimax与Minimax

function H-MINIMAX(s, d) **returns** an action
  if CUTOFF-TEST(s, d) **then return** EVAL(s)
  if PLAYER(s) = MAX **then return** $\max_{a \in \text{ACTIONS}(s)}$ H-MINIMAX(RESULT(s, a), d +1)
  if PLAYER(s) = MIN **then return** $\min_{a \in \text{ACTIONS}(s)}$ H-MINIMAX(RESULT(s, a), d +1)

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))

Evaluation Functions 评价函数

- An evaluation function returns an *estimate* of the expected utility of the game from a given position. How to design good evaluation functions?

  一个评价函数返回从一个给定位置该博弈的期望效用估计。如何设计好的评价函数?

  - It should order the *terminal* states in the same way as the true utility function:

    a) the states that are wins must evaluate better than draws,

    b) the states that are draws must be better than losses.

    它应该与真实效用函数相同的方式对终端状态进行整理：

    a) 获胜状态必须评价为优于平局，

    b) 平局状态必须评价为优于失败。

  - The computation must not take too long.

    计算的时间一定不能太长。

  - Nonterminal states should be strongly correlated with actual chances of winning.

    非终端状态应该与实际获胜的机会密切相关。

Weighted Linear Function 加权线性函数

-- A kind of evaluation function 一种评价函数

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

where 其中

- w_i -- a weight 权重
- f_i -- a feature of the position 位置的特征

For Chess 对于国际象棋

- f_i could be the numbers of each kind of piece on the board
  应该为棋盘上每种棋子的数量
- w_i could be the values of the pieces: 1 for pawn, 3 for bishop, etc.
  应该为棋子的数值：1表示兵、3表示象、等等。

Chess 国际象棋

(a) White to move 白棋要走子

(b) White moved 白棋走子后

Two chess positions that differ only in the position of the rook at lower right:
(a) Black has an advantage of a knight and two pawns, which should be enough to win the game.
(b) White will capture the queen, giving it an advantage that should be strong enough to win.

两盘国际象棋布局仅在右下角车的位置不同：
(a) 黑棋多一个马和两个兵，应该赢得这盘棋。(b) 白棋将捕获皇后，使得它处于足以获胜的态势。

5. Adversarial Search

Contents:

Stochastic Game 随机博弈

- What is stochastic game 什么是随机博弈
  - It is a dynamic game with probabilistic transitions played by one or more players, introduced in the early 1950s.
    是一种具有概率转换的动态博弈，有一个或多个玩家，于1950年代初提出的。
  - In real life, many unpredictable events can put us into unforeseen situations.
    在现实生活中，许多无法预测的事件可以使我们陷入始料不及的处境。
  - Many games mirror this unpredictability by including a random element, such as the throwing of dice.
    许多博弈通过引入一种随机元素来仿照这种不可预测性，例如掷骰子。
- Applications 应用
  - economics, evolutionary biology, computer networks.
    经济学、进化生物学、计算机网络。

Backgammon 西洋双陆棋

- Be a typical game that combines luck and skill, and one of the oldest classes of board games for two players.
  是一种典型的运气与技巧并存的博弈，并且是一个最老的两个玩家的棋盘博弈。
- The playing pieces are moved according to the roll of dice, and a player wins by removing all of their pieces from the board before their opponent.
  走棋是根据掷骰子来决定的，在对手之前将所有的棋子移到棋盘外的玩家则获胜。
- With each roll of the dice, players must choose from numerous options for moving their checkers and anticipate possible counter-moves by the opponent.
  随着每次掷骰子，玩家们必须从许多选项中选择如何移动棋子，并且要预见对手可能的对攻棋。

A Typical Backgammon Position 一盘典型的西洋双陆棋棋局

- White moves clockwise toward 25, and Black moves counterclockwise toward 0. The goal is to move all pieces off the board.

  白棋顺时针移到25，然后黑棋逆时针移到0。目标是将所有的棋子移到棋盘外。

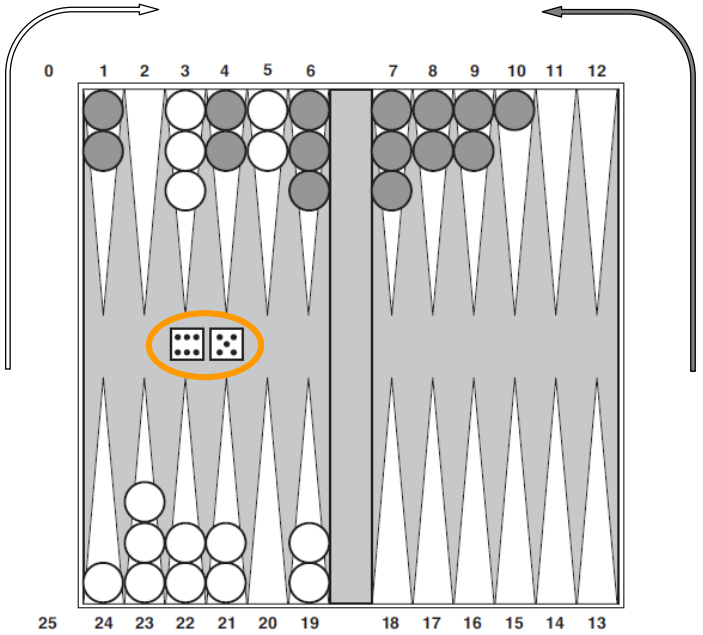
- A piece can move to any position unless multiple opponent pieces are there; if there is one opponent, it is captured and must start over.

  一个棋子可以移到任意位置，除非在那里有多个对手的棋子；如果有一个对手的棋子，它就被抓住、然后必须重新开始。

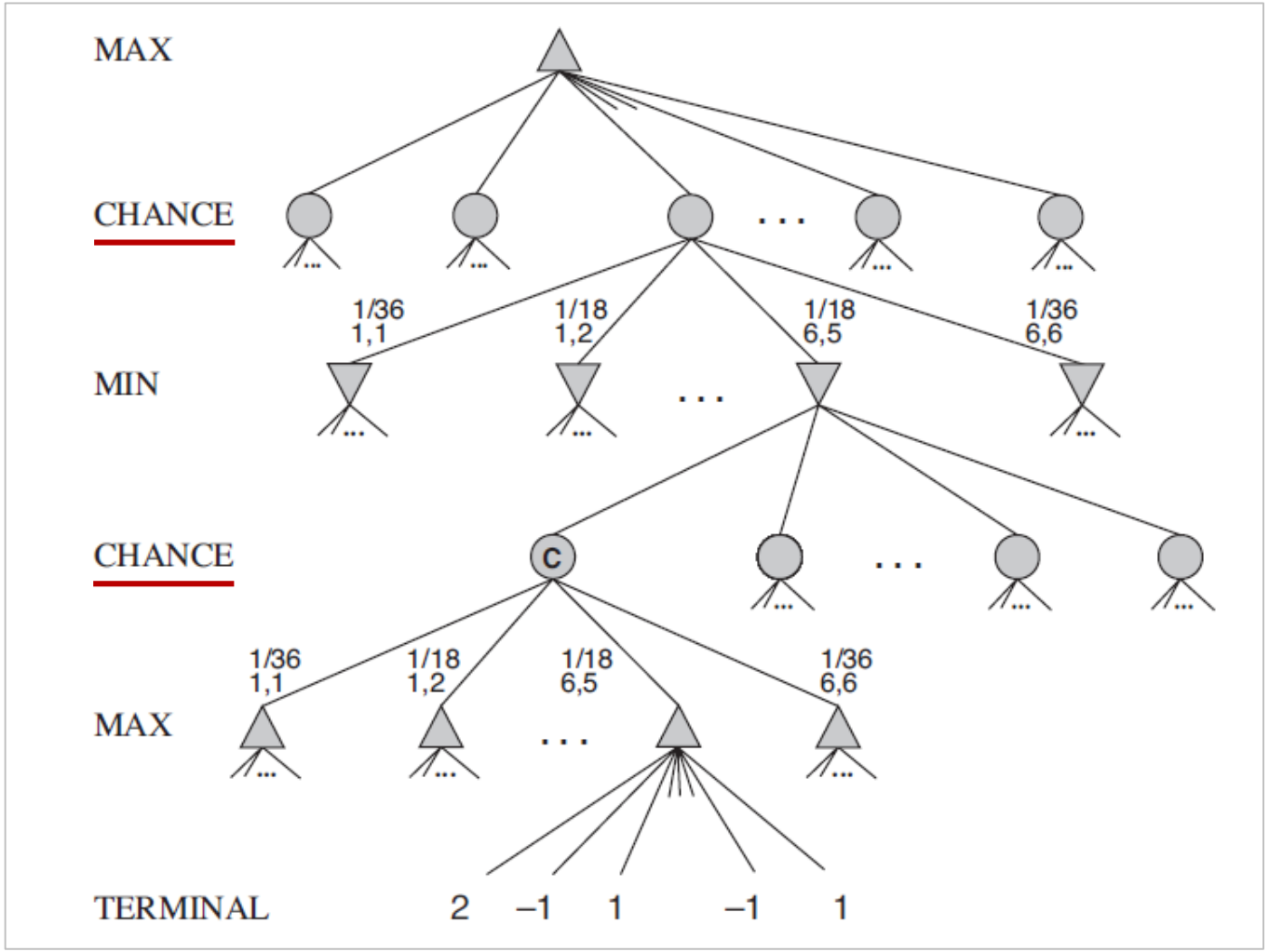
- In the position shown, White has rolled 6-5 and must choose among four legal moves:

  如该棋局所示，白棋已经掷了6-5，因而必须从四种合法的走棋中选择：

  (5-10, 5-11), (5-11, 19-24), (5-10, 10-16), (5-11, 11-16).



Game Tree for a Backgammon Position 西洋双陆棋棋局的博弈树



- There are 36 ways to roll two dice, each equally likely; but because a (6,5) is the same as a (5,6), there are only 21 distinct rolls.

  投掷两个骰子有36种方式，每种都有同样可能；但是由于(6, 5)与(5, 6)相同，故仅有21种不同的投掷。

- The six doubles, (1,1) through (6,6), each have a probability of 1/36, so $P(1,1) = 1/36$.

  六对儿双数，即 (1, 1) 到 (6, 6)，每对儿的概率为1/36，故：P(1, 1) = 1/36。

- The other 15 distinct rolls each have a 1/18 probability.

  其他15种不同的掷骰子，每种有1/18的概率。

Expected Minimax Value 期望Minimax值

- Expected minimax value for games with chance nodes.

  博弈的期望Minimax值具有机率节点。

- For chance nodes we compute the expected value, which is the sum of the value over all outcomes, weighted by the probability of each chance action:

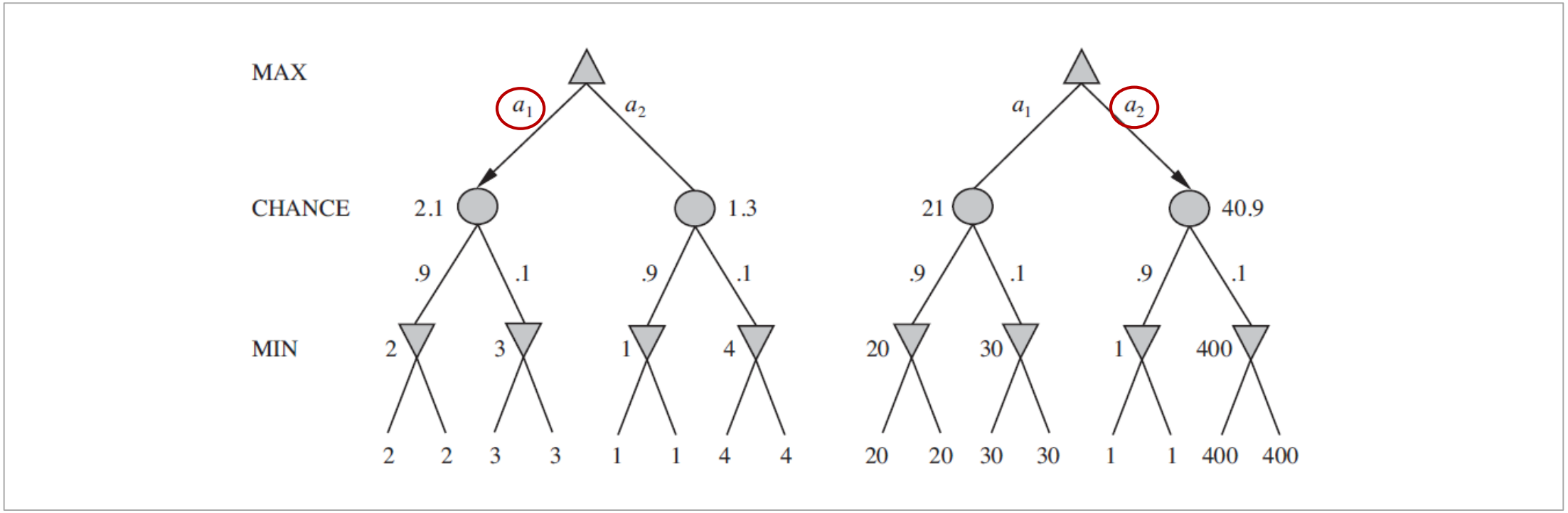
  对于机率节点，我们计算该期望值，它是涵盖所有结果值之和，由每个机率动作的概率来加权。

```
function EXPECTI-MINIMAX(s) returns an action
  if TERMINAL-TEST(s) then return UTILITY(s)
  if PLAYER(s) = MAX then return max_a EXPECTI-MINIMAX(RESULT(s, a))
  if PLAYER(s) = MIN then return min_a EXPECTI-MINIMAX(RESULT(s, a))
  if PLAYER(s) = CHANCE then return ∑_r P(r)EXPECTI-MINIMAX(RESULT(s, r))
```

Where r represents a possible dice roll (or other chance event), and
RESULT(s, r) is the same state as s, with the additional fact that the result of the dice roll is r.

其中，r表示一种可能的骰子投掷（或其他机率事件），而RESULT(s, r)与s的状态相同、且具有附加的骰子投掷结果为r的事实。

An Order-preserving Transformation 一种保序变换

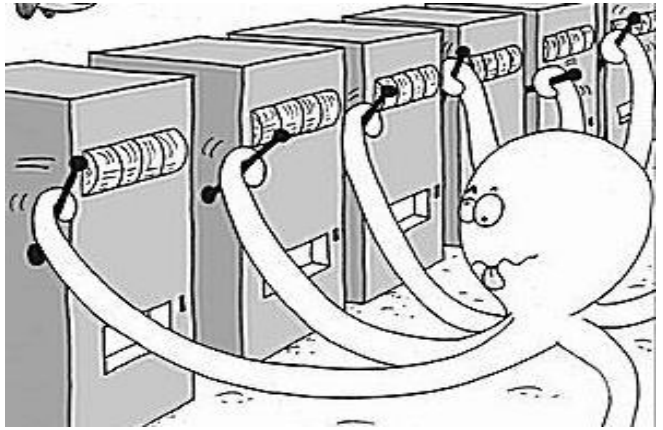


An order-preserving transformation on leaf values changes the best move:
assigns the values [1, 2, 3, 4] to the leaves, move a_1 is best; with values [1, 20, 30, 400], move a_2 is best.

一种叶节点值的保序变换改变最佳移动：
分配叶节点值为[1, 2, 3, 4]时，移动a_1最佳；而值为[1, 20, 30, 400]时，则移动a_2最佳。

Multi-armed Bandit 多臂老虎机

- A gambler faces several slot machines (one-armed bandits), that look identical but produce different expected winnings.
  一个赌徒面对着几台角子机（单臂老虎机），它们看起来完全相同，但具有不同的期望赢率。
- The issue is the trade-off between acquiring new information and capitalizing on the information available so far.
  问题是如何在获取新的信息和利用以前的可用信息之间权衡。
- One aspect that we are particularly interested in concerns modeling and efficiently using various types of side information that may be available to the algorithm.
  一个我们特别感兴趣的方面是关注建模以及有效地使用对算法或许有用的各种辅助信息。



5. Adversarial Search

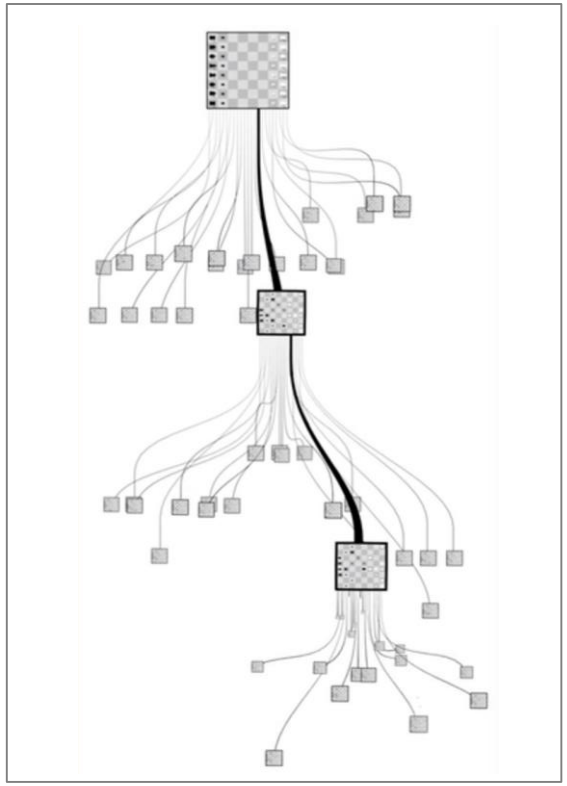
Contents:

- 5.1. Games
- 5.2. Optimal Decisions in Games
- 5.3. Alpha-Beta Pruning
- 5.4. Imperfect Real-time Decisions
- 5.5. Stochastic Games
- 5.6. Monte-Carlo Methods

Go vs. Chess 围棋与国际象棋

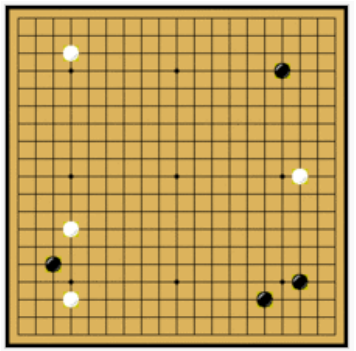
- Go has long been viewed as one of most complex game and most challenging of classic games for AI.

  围棋一直被视为最复杂的博弈之一、而且是最具挑战性的AI经典博弈。



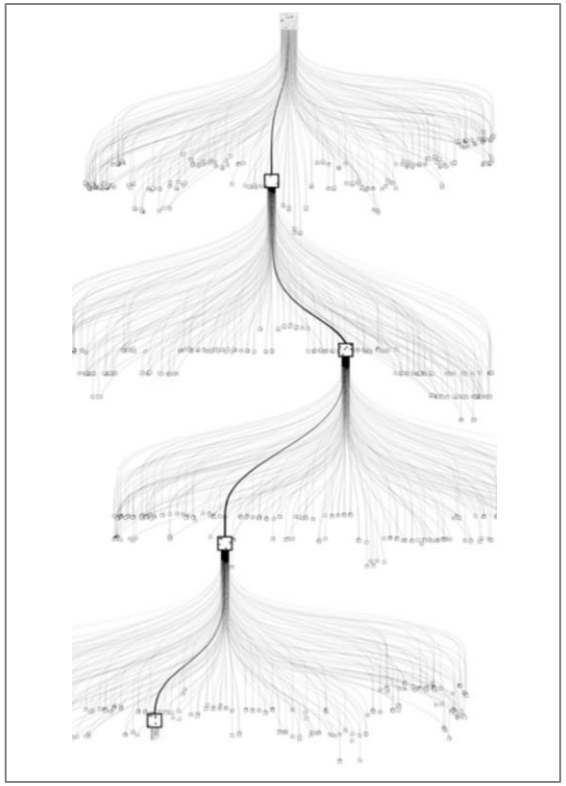
Chess (b ≈ 35, d ≈ 80)

8 x 8 = 64, possible games ≈ 10^{120}



Go (b ≈ 250, d ≈ 150)

19 x 19 = 361, possible games ≈ 10^{170}



Algorithm of AlphaGo AlphaGo的算法

- Deep neural networks 深度神经元网络
  - value networks: used to evaluate board positions
    价值网络：用于评估棋局
  - policy networks: used to select moves.
    策略网络：用于选择走子
- Monte-Carlo tree search (MCTS) 蒙特卡罗树搜索 (MCTS)
  - Combines Monte-Carlo simulation with value networks and policy networks.
    将蒙特卡罗仿真与价值和策略网络相结合
- Reinforcement learning 强化学习
  - used to improve its play.
    用于改进它的博弈水平。

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

About Monte-Carlo Methods 关于蒙特卡罗方法

- Monte-Carlo methods are a broad class of computational algorithms that rely on *repeated random sampling* to obtain numerical results.

  蒙特卡罗方法是一大类计算算法，它凭借重复随机采样来获得数值结果。

- They tend to follow a particular pattern:

  它们往往遵循如下特定模式：

  - define a domain of possible inputs;

    定义一个可能的输入域；

  - generate inputs randomly from a probability distribution over the domain;

    从该域的一个概率分布随机地生成输入；

  - perform a deterministic computation on the inputs;

    对该输入进行确定性计算；

  - aggregate the results.

    将结果聚合。

Example: Approximating π by Monte-Carlo Method 用蒙特卡罗方法估计π

- 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:

  鉴于圆形与正方形面积之比为$\pi/4$，则π的值可采用蒙特卡罗方法近似得出：

  a) Draw a square on the ground, then inscribe a circle within it.

  先画出一个正方形，然后在其中画一个圆弧。

  b) Uniformly scatter some objects of uniform size over the square.

  将尺寸大小一致的小颗粒散落在正方形上。

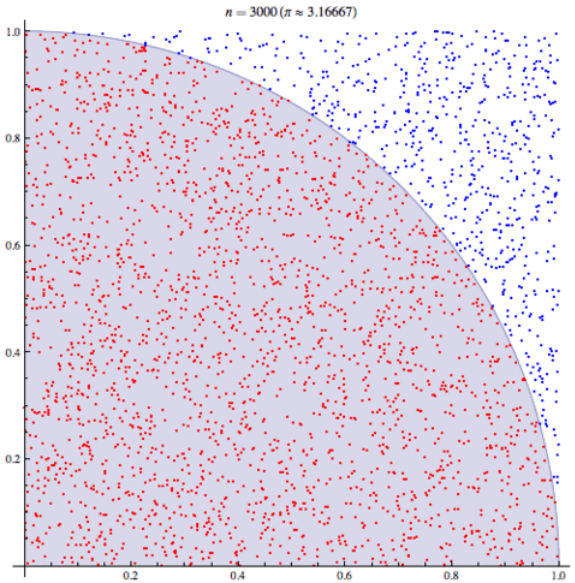
  c) Count the number of objects inside the circle and the square.

  计算圆形和正方形中小颗粒的数量和总的数量。

  d) 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.

  两个数量之比为两个面积的估算，即$\pi/4$。结果乘以4得出π。

Source: Wikipedia



Family of Monte-Carlo Methods 蒙特卡罗方法的家族

- **Classical** Monte-Carlo: 经典蒙特卡罗

  samples are drawn from a probability distribution, often the classical Boltzmann distribution;

  样本来自于概率分布，往往是经典的玻兹曼分布；

- **Quantum** Monte-Carlo: 量子蒙特卡罗

  random walks are used to compute quantum-mechanical energies and wave functions;

  采用随机走查方法来计算量子力学的能量和波函数；

- **Volumetric** Monte-Carlo: 容积式蒙特卡罗

  random number generators are used to generate volumes per atom or to perform other types of geometrical analysis;

  采用随机数生成的方法产生每个原子的容量、或进行其它类型的几何分析。

- **Kinetic** Monte-Carlo: 动力学蒙特卡罗

  simulate processes using scaling arguments to establish timescales or by introducing stochastic effects into molecular dynamics.

  仿真过程采用尺度分析来建立时间尺度、或者将随机效应引入分子动力学。

Monte-Carlo Tree Search (MCTS) 蒙特卡罗树搜索 (MCTS)

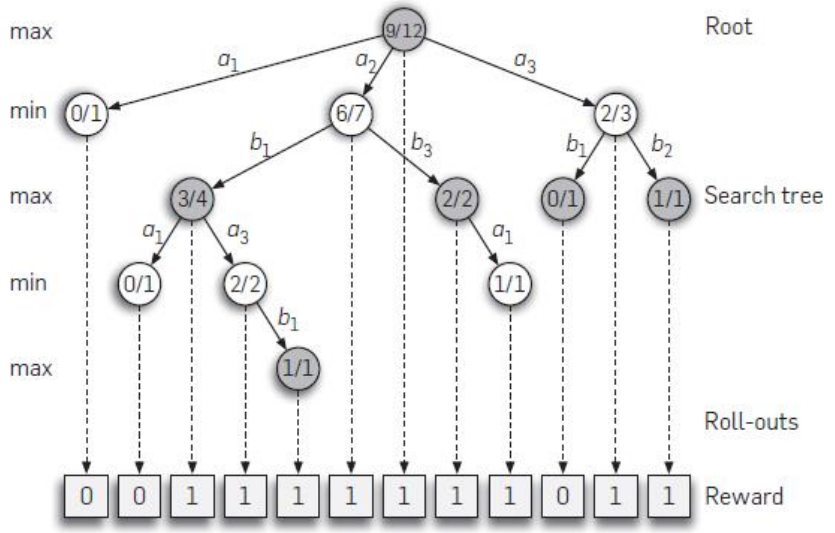
- MCTS combines Monte-Carlo simulation with game tree search.
  MCTS将蒙特卡罗仿真与博弈树搜索相结合。
- Like minimax, each node corresponds to a single state of game.
  和minimax一样，每个节点对应于一个的博弈状态。
- Unlike minimax, the values of nodes are estimated by Monte-Carlo simulation.
  不同于minimax，节点的值通过蒙特卡罗仿真来估值。

$W(a)/N(a)$ = the value of action a 动作a的值

where: 其中

$W(a)$ = the total reward 总的奖励

$N(a)$ = the number of simulations 仿真的数量



Source: Communications of the ACM, Mar. 2012, 55(3), pp. 106-113

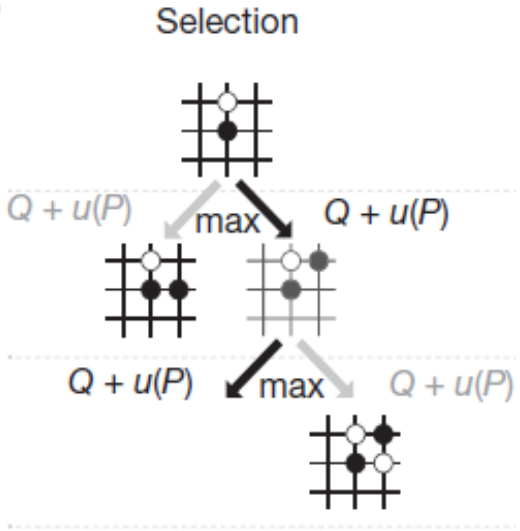
Algorithm of AlphaGo AlphaGo的算法

- It uses a combination of machine learning and tree search techniques, combined with extensive training, both from human and computer play.
  采用机器学习和树搜索技术相结合的方式，并且用人类和计算机走棋的棋局进行大量的训练。
- Two deep neural networks 两个深度神经网络
  - value networks to evaluate board positions and policy networks to select moves.
    价值网络来评价棋盘位置、策略网络来选择走棋。
- Tree search 树搜索
  - Monte Carlo tree search (MCTS).
    蒙特卡罗树搜索 (MCTS)
- Reinforcement learning 强化学习
  - be used to improve its play.
    用于改善其走棋。

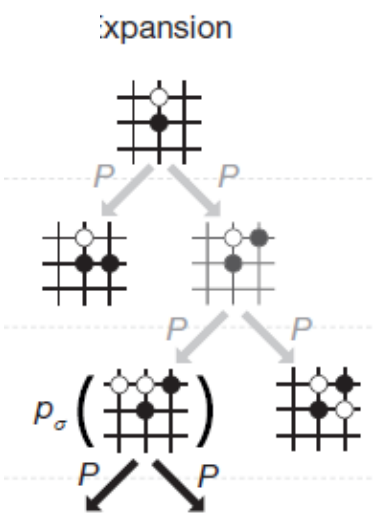
Monte-Carlo Tree Search in AlphaGo AlphaGo的蒙特卡罗树搜索

Source: Nature, Jan. 28, 2016

(a)



(b)



(a) Selection: Each simulation traverses the tree by selecting edge with maximum action value Q + bonus $u(P)$ that depends on a stored prior probability P for that edge.

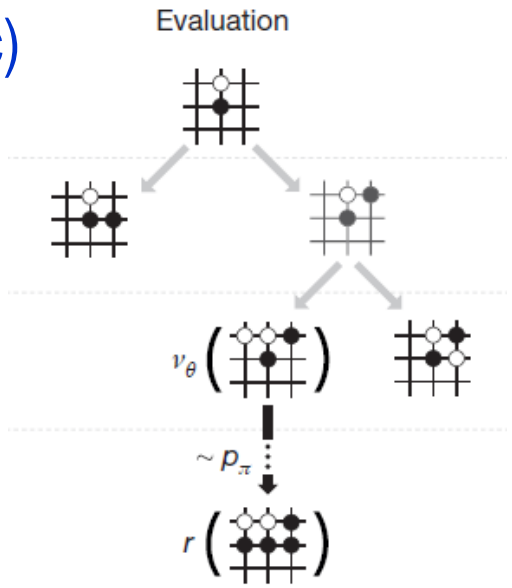
选择：每次仿真通过选择边与最大动作值Q+奖励$u(P)$对搜索树进行遍历，依赖于该条边存储的先验概率P。

(b) Expansion: The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action.

扩展：叶节点可以扩展；新的节点先由策略网络p_σ处理，然后其输出概率存储为每个动作的先验概率P。

Monte-Carlo Tree Search in AlphaGo

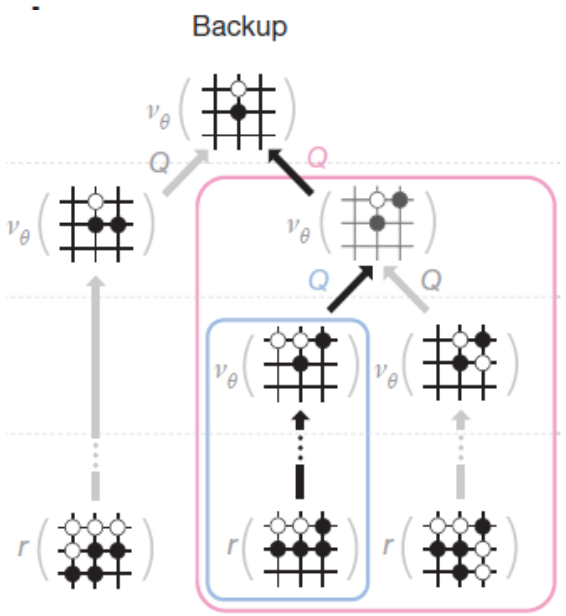
(c)



(c) Evaluation (*Simulation*): The leaf node is evaluated in two ways: 1) using the value network v_θ; 2) by running a rollout to the end of the game with the fast rollout policy p_π, then computing the winner with function r.

评价（仿真）:叶节点用两种方法评价：1) 使用价值网络v_θ；2) 使用快速走子策略p_π运行到博弈结束，然后用函数r计算出胜者。

(d)



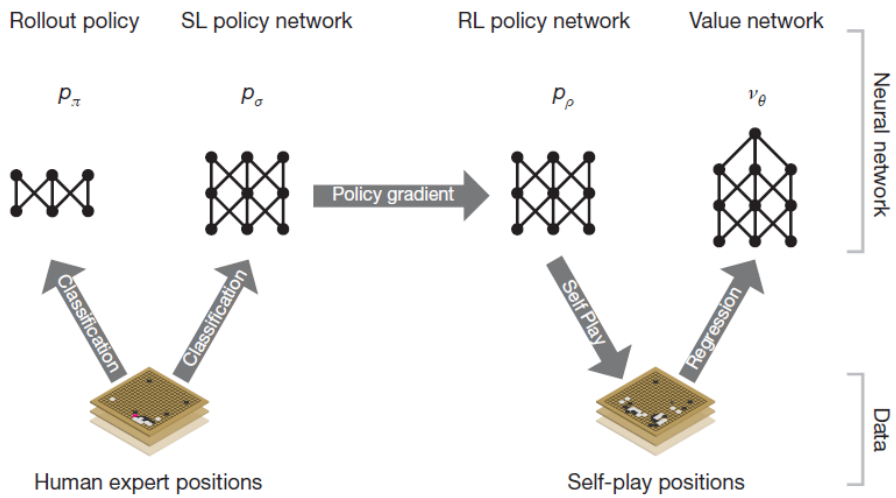
(d) Backup (Back propagation): Action values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.

后援（反向传播）：更新动作值Q来跟踪在该动作下面子树的所有评价函数$r(\cdot)$和$v_\theta(\cdot)$的平均值。

Neural Network Training Pipeline in AlphaGo AlphaGo的神经网络训练管线

- A fast rollout policy p_π and supervised learning (SL) policy network p_σ are trained to predict human expert moves in a data set of positions.

  快速走子策略 p_π与有监督学习 (SL) 策略网络 p_σ用棋局数据集进行训练，来预测人类的走棋。



- A reinforcement learning (RL) policy network p_ρ is initialized to the SL policy network, and then improved by policy gradient learning to maximize the outcome.

  强化学习 (RL) 策略网络p_ρ 被初始化为SL 策略网络，然后通过策略梯度学习使输出最大化。

- A new data set is generated by playing games of self-play with the RL policy network.

  经过自我对弈和PL策略网络生成一个新的数据集。

- A value network v_θ is trained by regression to predict the expected outcome.

  通过回归训练价值网络 v_θ来预测所期望的输出。

Neural Network Architecture in AlphaGo AlphaGo的神经网络架构

- The policy network 策略网络
  - input: the board position s
    输入：棋盘位置s
  - passes s through convolutional layers with parameters σ or ρ
    将s穿过具有参数σ或ρ的卷积层
  - outputs: a probability distribution over legal moves a.
    输出：一个合法走子a的概率分布。
- The value network 价值网络
  - input: the board position s'
    输入：棋盘位置s'
  - similarly uses many convolutional layers with parameters θ
    同样采用具有参数θ的卷积层
  - output: a scalar value $v_\theta(s')$ that predicts the expected outcome.
    输出：一个预测期望输出的标量值$v_\theta(s')$