5.4 启发式图搜索策略

- **5.4.1 启发信息和估价函数**
- **5.4.2 A搜索算法**
- **5.4.3 A*搜索算法及其特性分析**
- 补充：博弈搜索策略

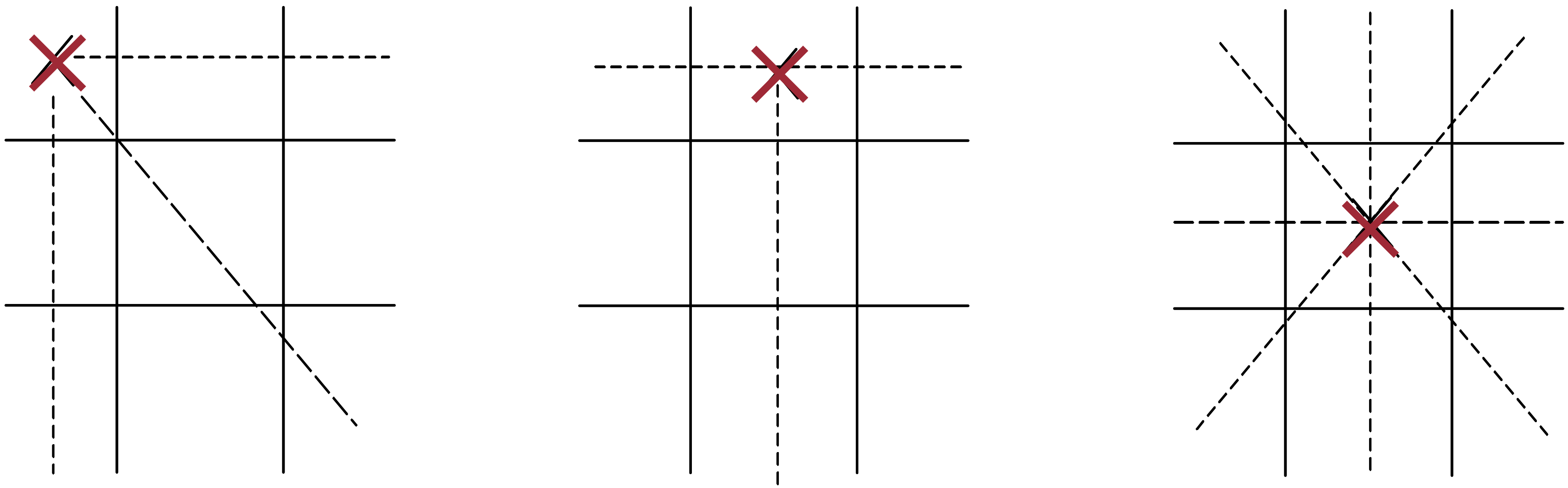
启发式策略例子

- 例5.6， 一字棋或井字棋（Tic-Tac-Toe游戏）（http://www.4399.com/flash/50489_1.htm）。在九宫棋盘上，从空棋盘开始，双方轮流在棋盘上摆各自的棋子 × 或 ○（每次一枚），谁先取得三子一线（一行、一列或一条对角线）的结果就取胜。

×	×	×
○	×	
○		○

- × 和 ○ 能够在棋盘中摆成的各种不同的棋局就是问题空间中的不同状态。
- 可能的走法： $9\times8\times7\times\cdots\times1$ ， 有 9！=362,880种棋局。

启发式策略例子



赢的几率③　　赢的几率②　　赢的几率④

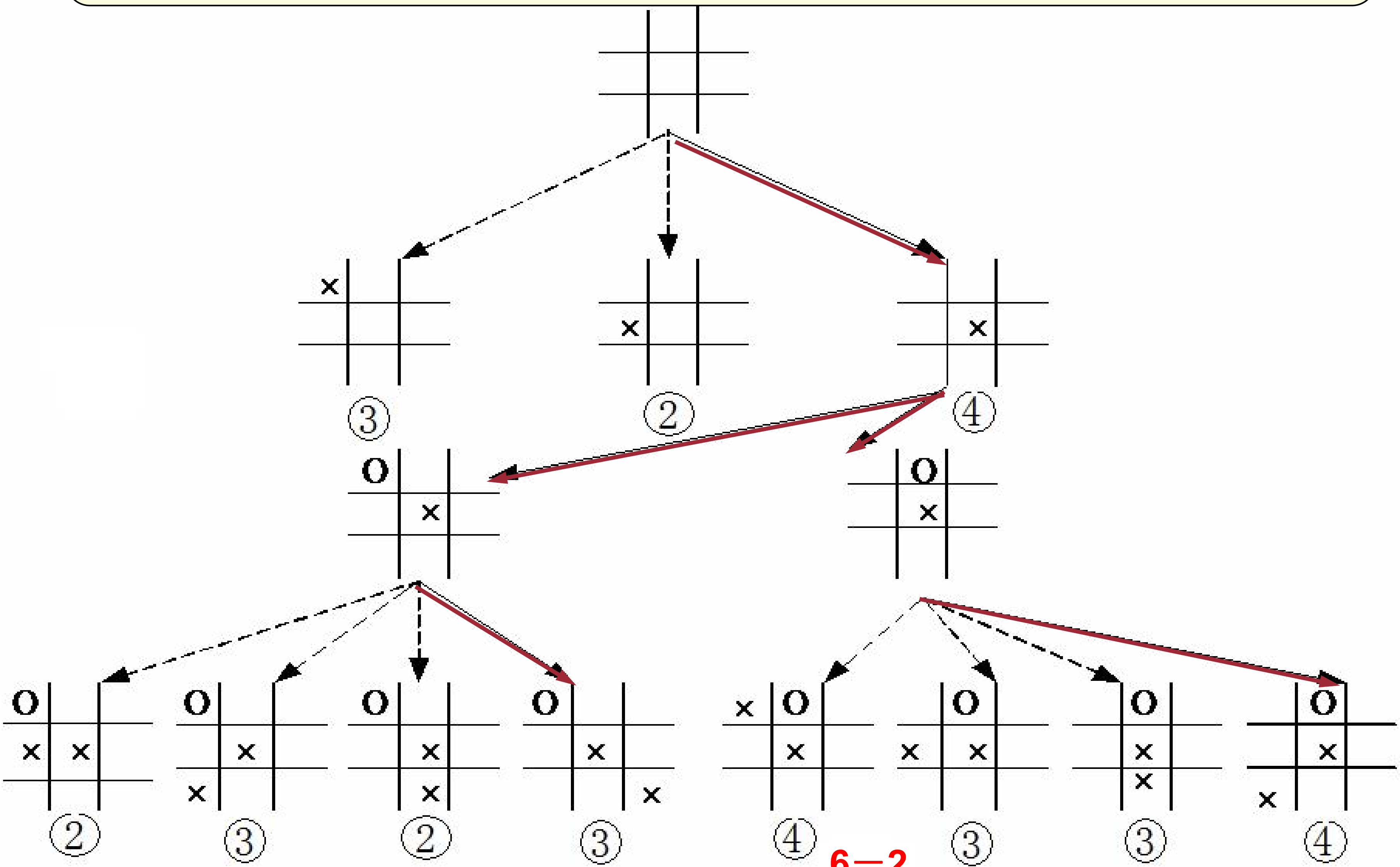
棋局的启发值：所有空格都放上X后三子一线的总数 - 所有空格都放上O后三子一线的总数。

8－5　　**8－6**　　**8 － 4**

图5.12 启发式策略的运用

启

棋局的启发值：所有空格都放上X后三子一线的总数 - 所有空格都放上O后三子一线的总数。



5—3 5—2 6—2

图5.13启发式搜索下缩减的状态空间

启发式策略例子

×	×	×
○	×	
○		○

- 采用启发式策略的一字棋搜索的上限大约为**4.5×9**，近40种状态，比原来的**9！**大小的状态空间缩小了很多。

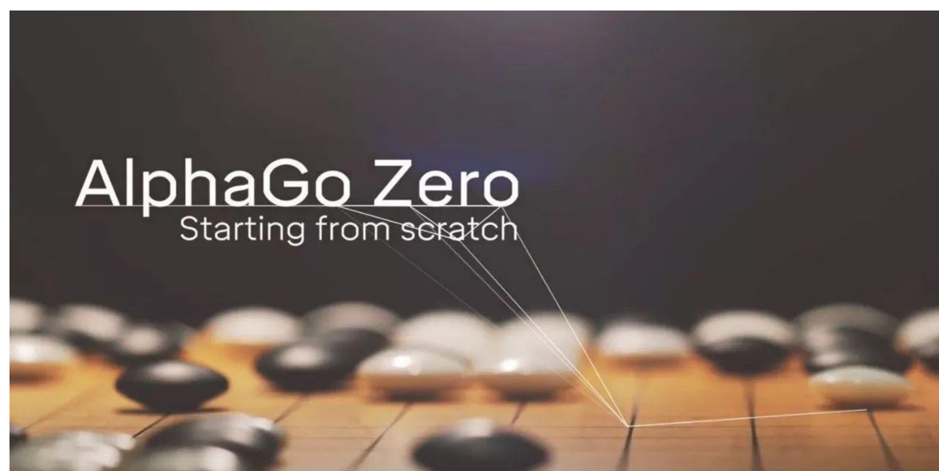
- 现实中有许多复杂问题是阶乘或指数级别的，其规模很大，例如:

• 西洋跳棋是10^{78}，国际象棋是10^{120}，围棋是10^{761}。

• 假设计算机每步可以搜索一个棋局，用极限并行速度（10^{-104}年/步）来处理，搜索一遍国际象棋的全部棋局就要10^{16}年即1亿亿年，而已知的宇宙寿命才100亿年。

博弈搜索(Game Search)

- **极小极大搜索(Minimax Search)**或最小最大搜索：博弈搜索中最为基本的一种让玩家来计算最优策略的方法。
- **Alpha-Beta剪枝搜索(α−β剪枝搜索)**: 一种对最小最大搜索进行改进的算法，即在搜索过程中可剪除无需搜索的分支节点，且不影响搜索结果。
- **蒙特卡洛树搜索(Monte-Carlo Tree Search, MCTS)**: 通过采样而非穷举方法来实现搜索。

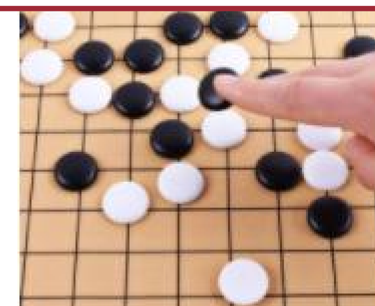


博弈搜索(Game Search)

- 整个博弈过程属于零和博弈，即一方的收益必然意味着另一方的损失，博弈双方的收益和损失相加总和永远是零，双方不存在任何合作的可能。
- 博弈双方足够聪明，即每一方在决策时总会选择使自己利益最大化的决策。
- **极小极大搜索(Minimax Search):**

正方（MAX节点）从所有子节点中，选取具有最大评估值的节点。

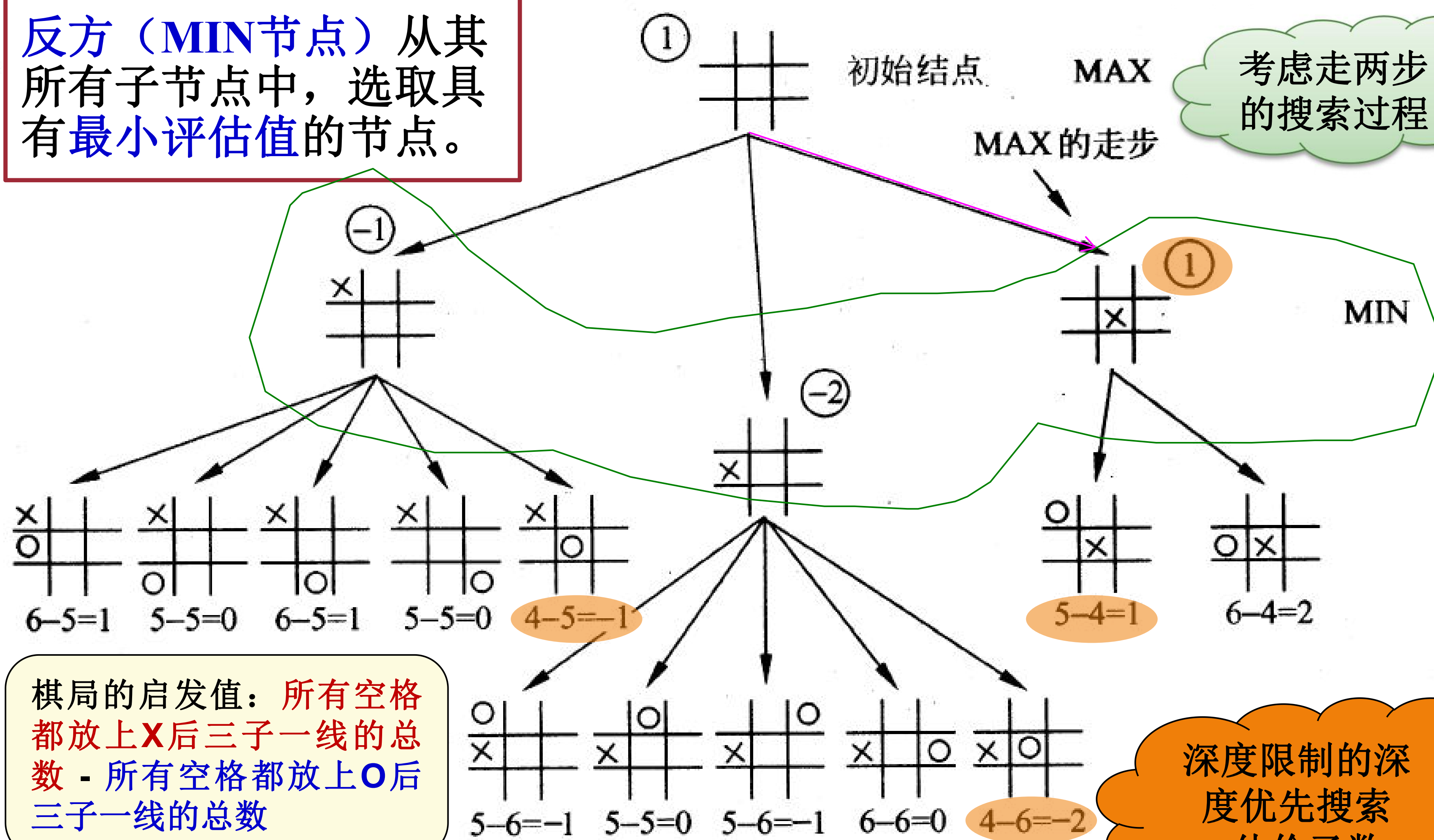
反方（MIN节点）从其所有子节点中，选取具有最小评估值的节点。



极小极大搜索过程

正方（MAX节点）从所有子节点中，选取具有最大评估值的节点。

反方（MIN节点）从其所有子节点中，选取具有最小评估值的节点。



考虑走两步的搜索过程

棋局的启发值：所有空格都放上X后三子一线的总数 - 所有空格都放上O后三子一线的总数

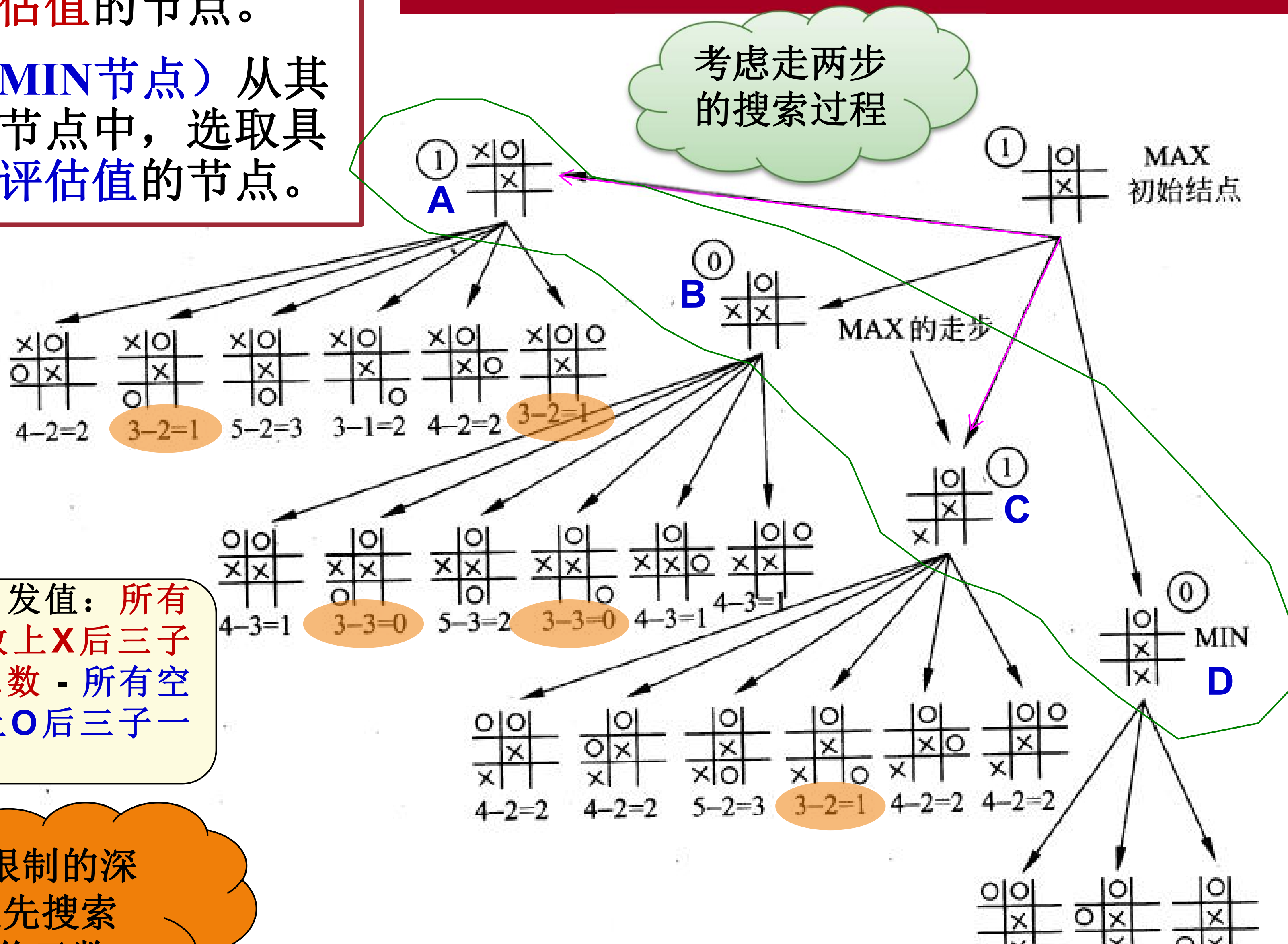
深度限制的深度优先搜索+估价函数

一字棋第一阶段搜索树

极小极大搜索过程

正方（MAX节点）从所有子节点中，选取具有最大评估值的节点。

反方（MIN节点）从其所有子节点中，选取具有最小评估值的节点。



棋局的启发值：所有空格都放上X后三子一线的总数 - 所有空格都放上O后三子一线的总数

深度限制的深度优先搜索+估价函数

一字棋第二阶段搜索树

极小极大搜索(Minimax Search)

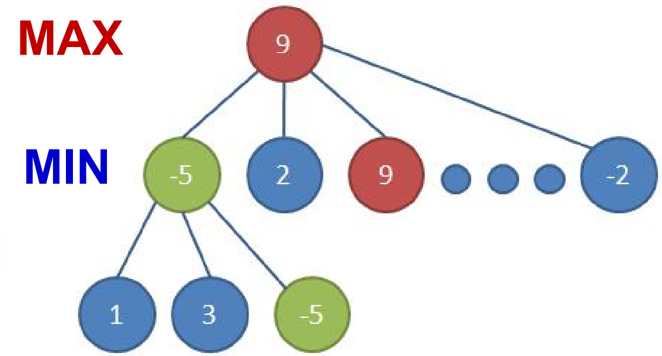


图-极大极小搜索

- **Complete** ? Yes (if tree is finite)
- **Optimal** ? Yes (against an optimal opponent)
- **Time complexity** ? $O(b^m)$
- **Space complexity** ? $O(b \times m)$ (depth-first exploration)

m 是游戏树的最大深度，在每个节点存在b个有效走法

- For chess, b ≈ 35, m ≈100 for "reasonable" games
  → exact solution completely infeasible

极小极大搜索(Minimax Search)

- **优点:**
  - 算法是一种简单有效的博弈搜索手段
  - 在对手也“尽力而为”前提下，算法可返回最优结果
- **缺点:**
  - 如果搜索树极大，则无法在有效时间内返回结果
- **改善:**
  - 使用**Alpha-Beta剪枝**算法来减少搜索节点，降低搜索的宽度
  - 对节点进行采样、而非逐一搜索 (例如蒙特卡洛树搜索)，降低搜索的深度

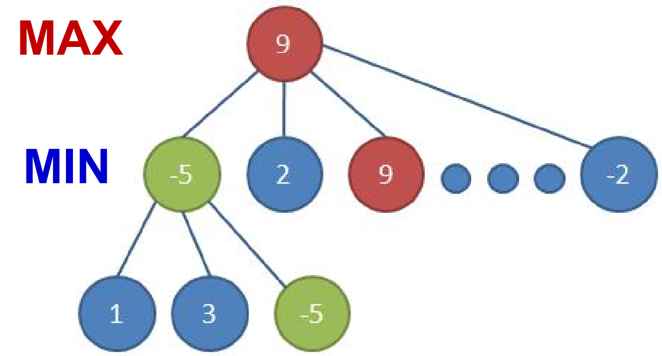
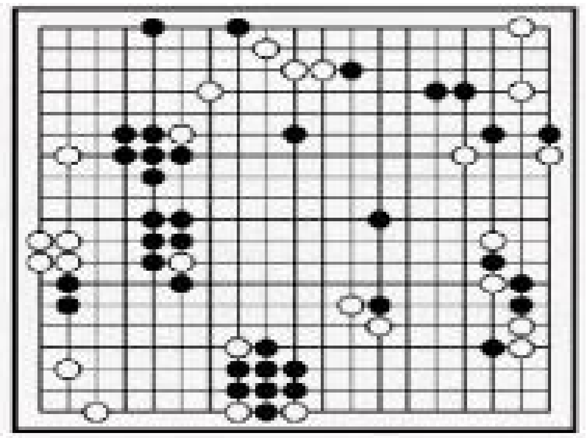


图-极大极小搜索



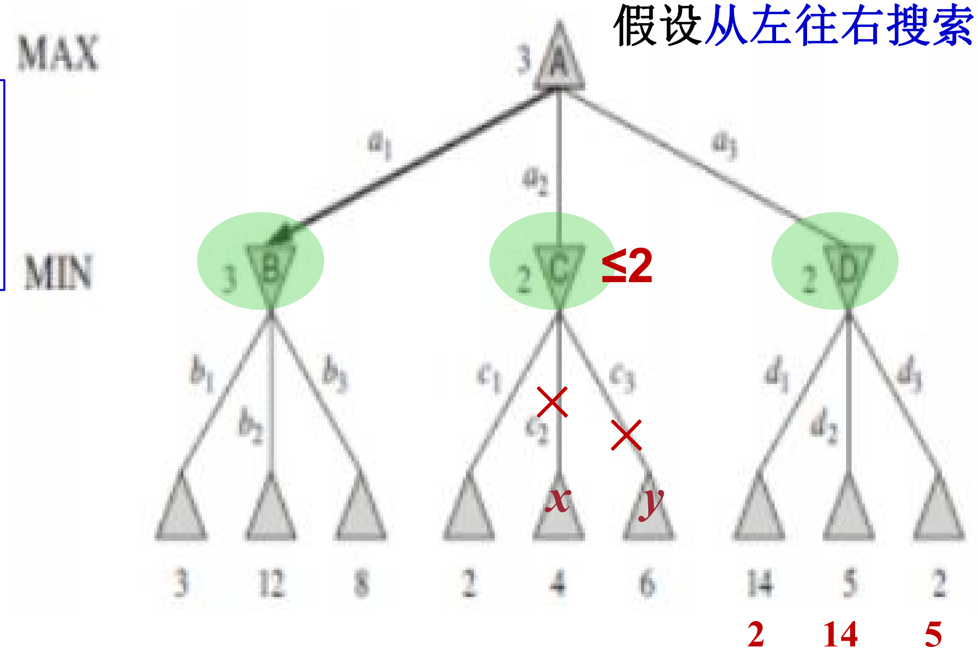
枚举当前局面之后每一种下法，然后计算每个后续局面的赢棋概率，选择概率最高的后续局面

α-β剪枝搜索(Alpha-Beta 剪枝搜索)

- **α-β剪枝法（20世纪50年代，约翰.麦卡锡）**

在极小化极大算法（**minimax**算法）中减少所搜索的搜索树节点数。该算法和极小化极大算法所得结论相同，但剪去了不影响最终结果的搜索分枝。

假设从左往右搜索



$$
\begin{aligned}
&MINIMAX(root)\\
&= \max(\min(3.12,8), \min(2,x,y), \min(14,5,2))\\
&= \max(3, \min(2,x,y), 2)\\
&= \max(3, z, 2)=3
\end{aligned}
$$

$where\ z = \min(2,x,y) \le 2$

可以看出：根节点（即MAX选手）的选择与x和y 两个值无关（因此，x和y可以被剪枝去除）

深度限制的深度优先搜索 +估价函数+剪枝

需要注意的是，剪枝的效果与树节点的访问顺序有关。

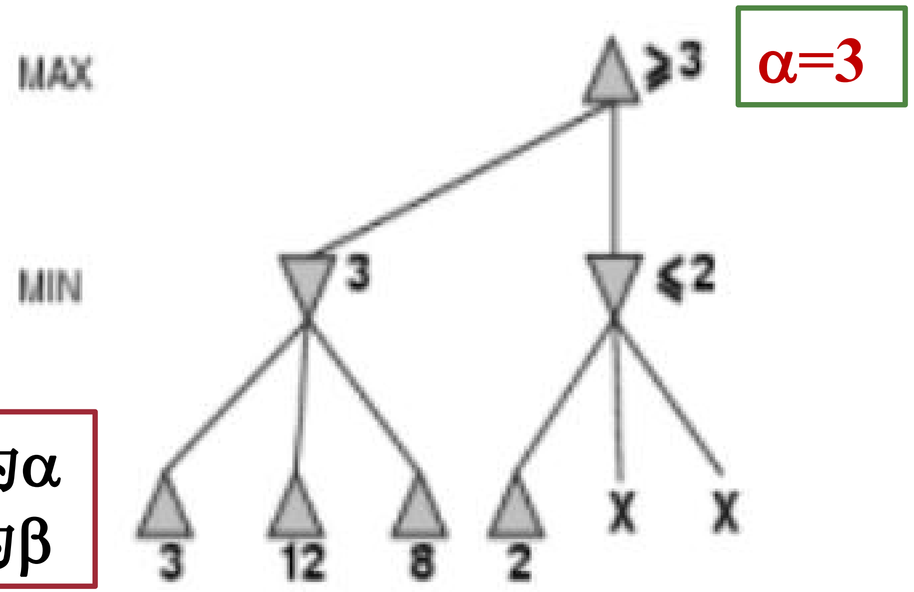
图中**MIN**选手所在的节点**C**下属分支**4**和**6**与根节点最终优化决策的取值无关，可不被访问。

α-β剪枝搜索(Alpha-Beta 剪枝搜索)

- **MAX节点的评估下限值α**

作为正方出现的MAX节点，假设它的MIN子节点有N个，那么当它的第一个MIN子节点的评估值为α时，则对于其它的子节点，如果有高过α的，就取那最高的值作为该MAX节点的评估值；如果没有，则该MAX节点的评估值为α。

总之，该MAX节点的评估值不会低于α，这个α就称为该MAX节点的评估下限值。



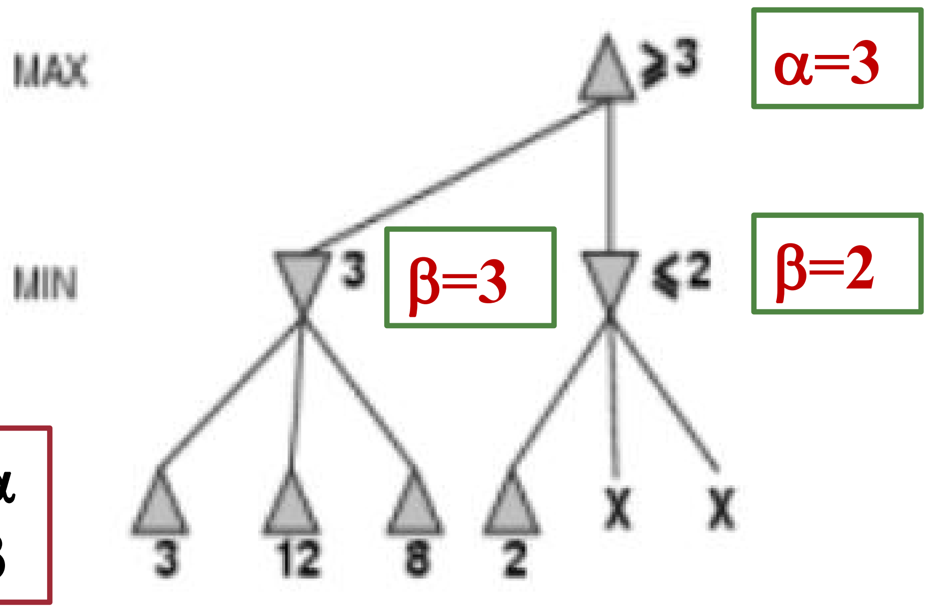
◆极大（**MAX**）节点的下界为α
◆极小（**MIN**）节点的上界为β

α-β剪枝搜索(Alpha-Beta 剪枝搜索)

MIN节点的评估上限值β

作为反方出现的MIN节点，假设它的MAX子节点有N个，那么当它的第一个MAX子节点的评估值为β时，则对于其它子节点，如果有低于β的，就取那个低于β的值作为该MIN节点的评估值；如果没有，则该MIN节点的评估值取β。

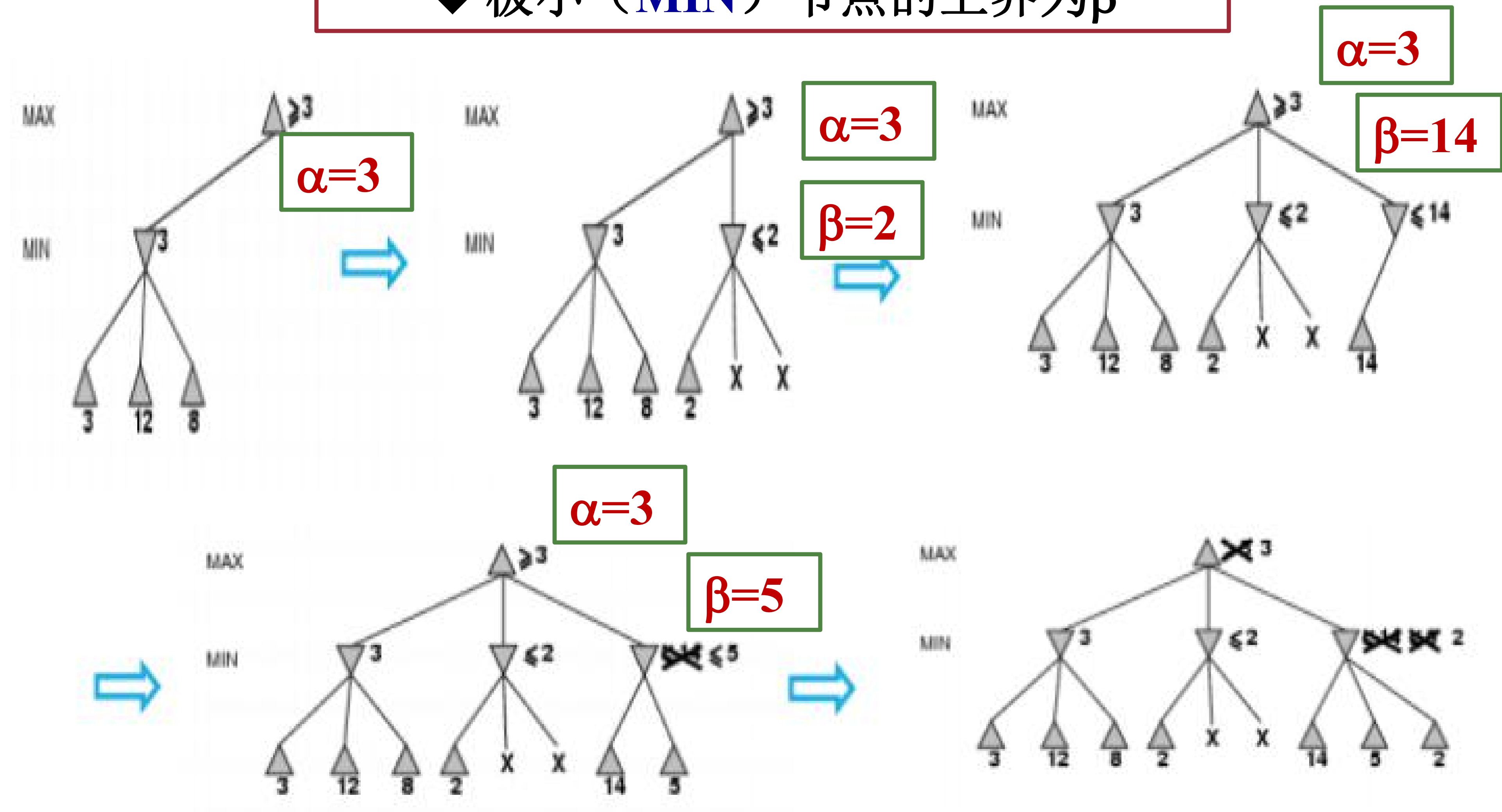
总之，该MIN节点的评估值不会高过β，这个β就称为该MIN节点的评估上限值。



◆极大（**MAX**）节点的下界为α
◆极小（**MIN**）节点的上界为β

α-β剪枝搜索(Alpha-Beta 剪枝搜索)

◆ 极大（**MAX**）节点的下界为α
◆ 极小（**MIN**）节点的上界为β

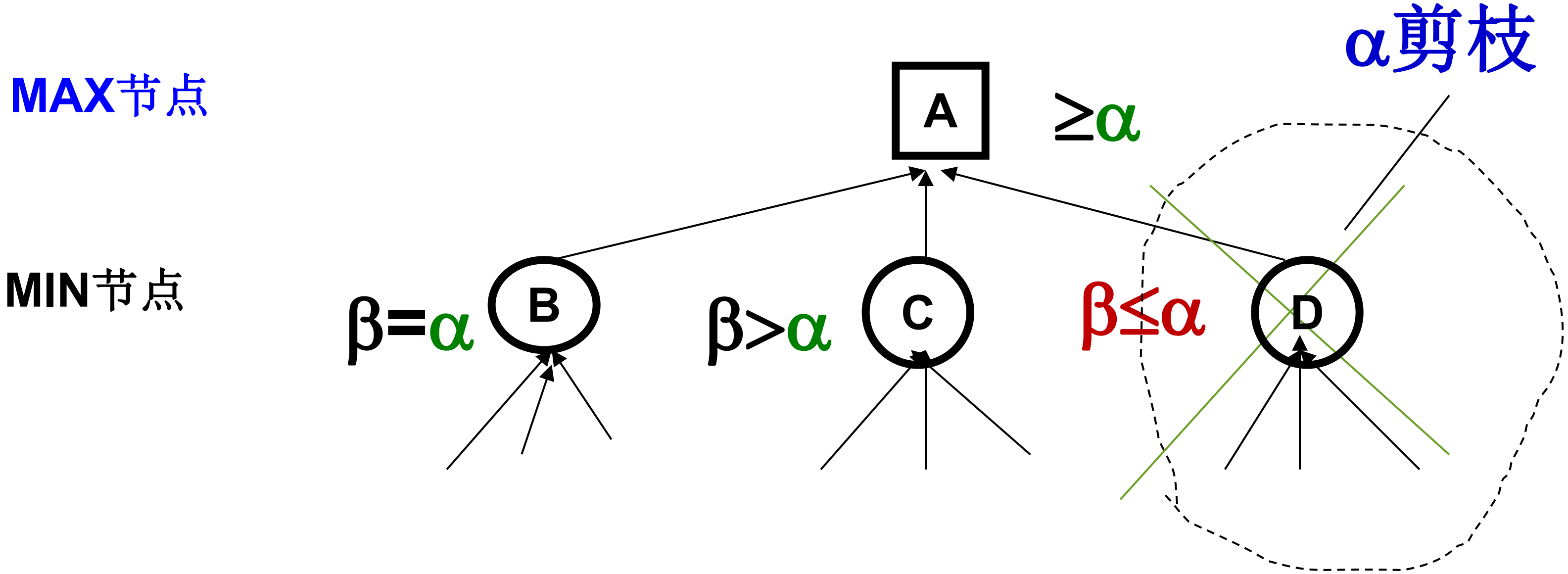


α-β剪枝搜索(Alpha-Beta 剪枝搜索)

◆极大（**MAX**）节点的下界为α
◆极小（**MIN**）节点的上界为β

- α剪枝

设**MAX**节点的下限为α，则其所有的**MIN**子节点中，其评估值的β上限小于等于α的节点，其以下部分的搜索都可以停止了，即对这部分节点进行了α剪枝。



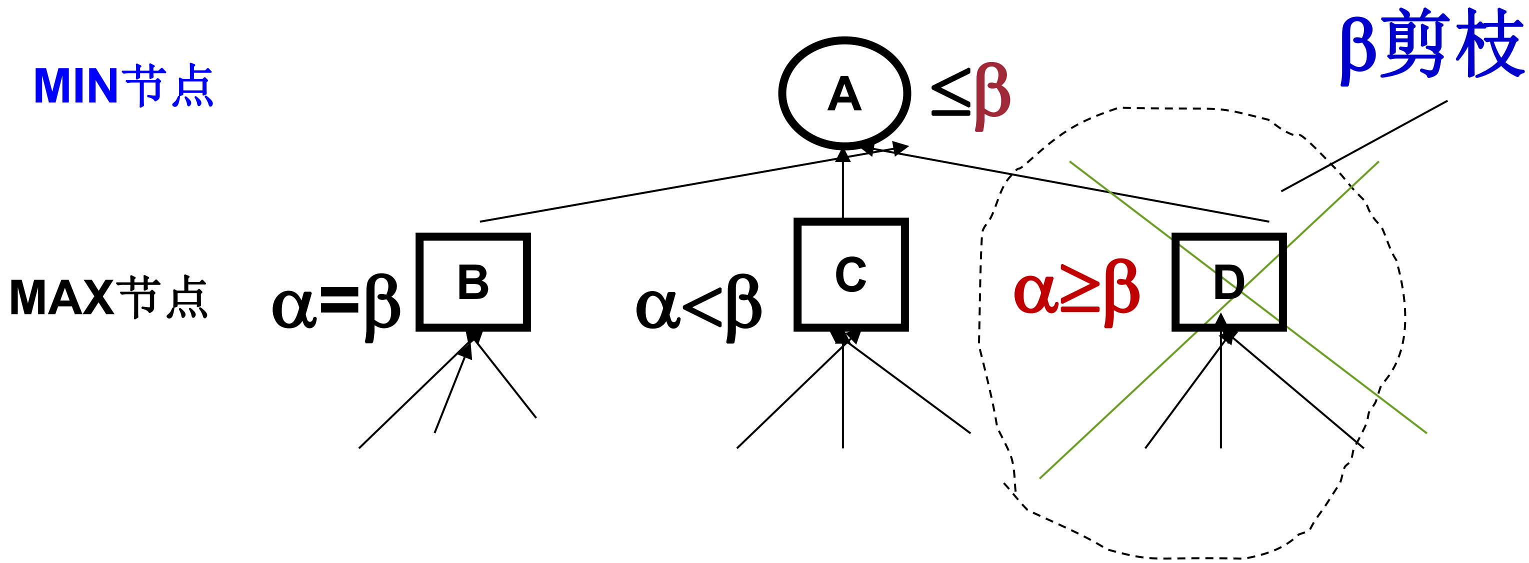
当一个 Min 节点的 β值 ≤ 任何一个父节点的α值时，剪掉该节点的所有子节点

α-β剪枝搜索(Alpha-Beta 剪枝搜索)

- ◆极大（**MAX**）节点的下界为α
- ◆极小（**MIN**）节点的上界为β

β剪枝

设MIN节点的上限为β，则其所有的MAX子节点中，其评估值的α下限大于等于β的节点，其以下部分的搜索都可以停止了，即对这部分节点进行了β剪枝。



当一个 Max 节点的 α值 ≥ 任何一个父节点的β值时，剪掉该节点的所有子节点

α-β剪枝搜索(Alpha-Beta 剪枝搜索)

- MAX节点的下界为α
- MIN节点的上界为β

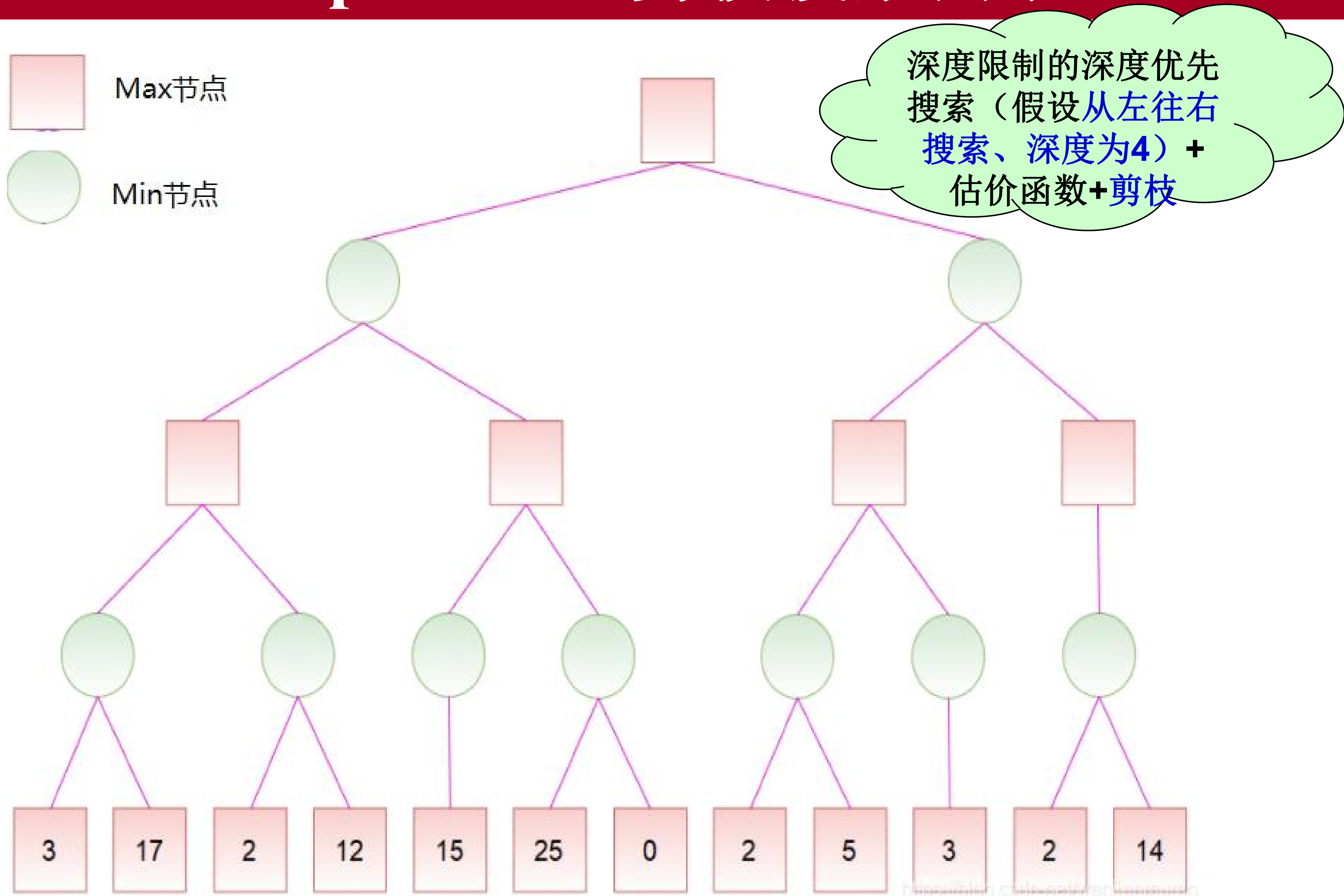
假设节点N的收益为 reward(N)，
对于α ≤ reward(N) ≤ β:
若 α ≤ β 则reward(N)有解。
若 α > β 则reward(N)无解，可剪枝

- 剪枝的条件：
  - MAX节点：子节点的β值 ≤ 父节点的α值时， α剪枝
  - MIN节点：子节点的α 值 ≥ 父节点的β值时， β剪枝

Alpha值(α)	MAX节点目前得到的最高收益
Beta值(β)	MIN节点目前可给对手的最小收益
α和β的值初始化分别设置为$-\infty$和∞	

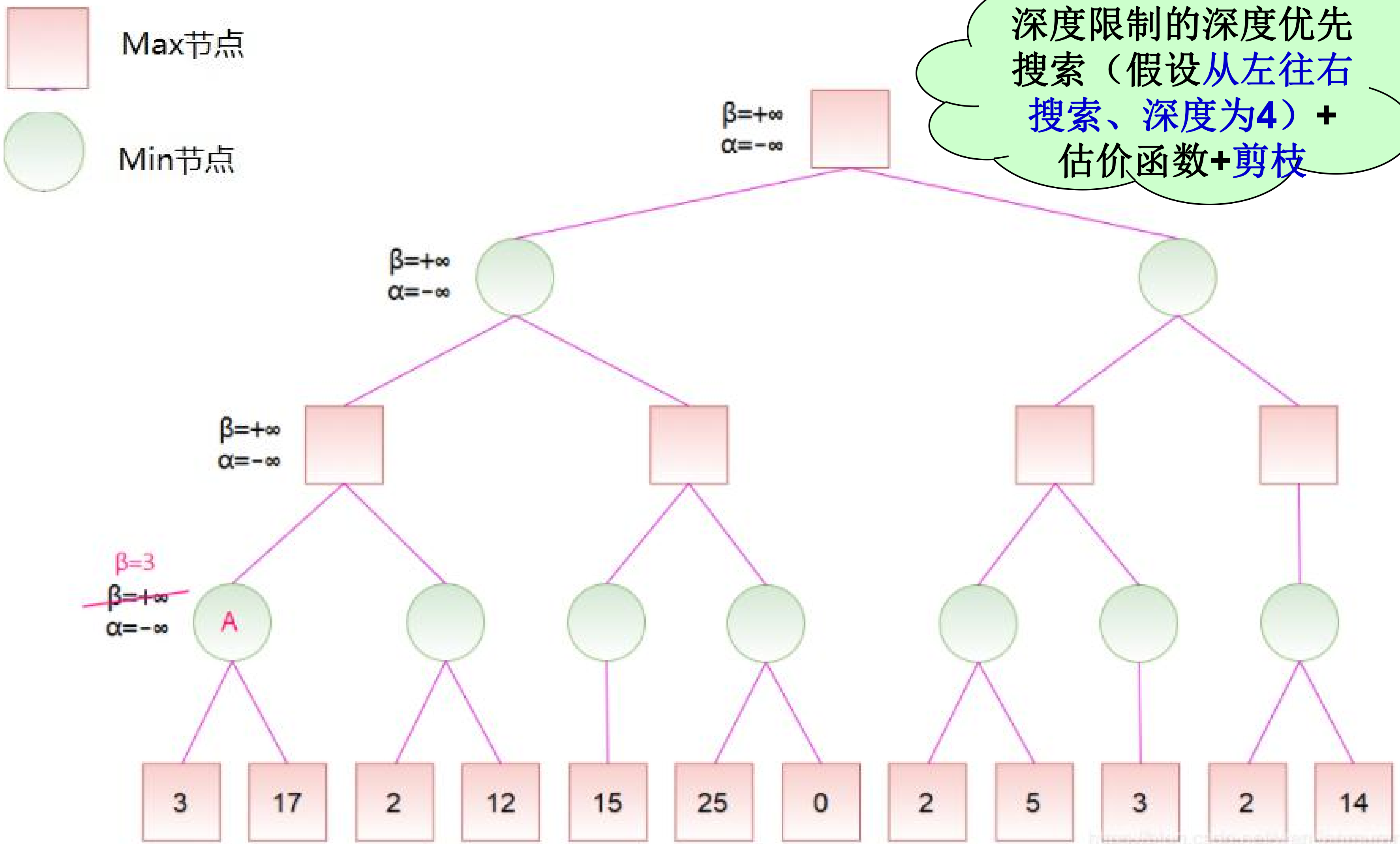
每个节点有两个值，分别是α和β。节点的α和β值在搜索过程中不断变化。其中，α从负无穷大($-\infty$)逐渐增加、β从正无穷大(∞)逐渐减少。如果一个节点中$\alpha > \beta$，则该节点的后续节点可剪枝。

Alpha-Beta 剪枝搜索例子

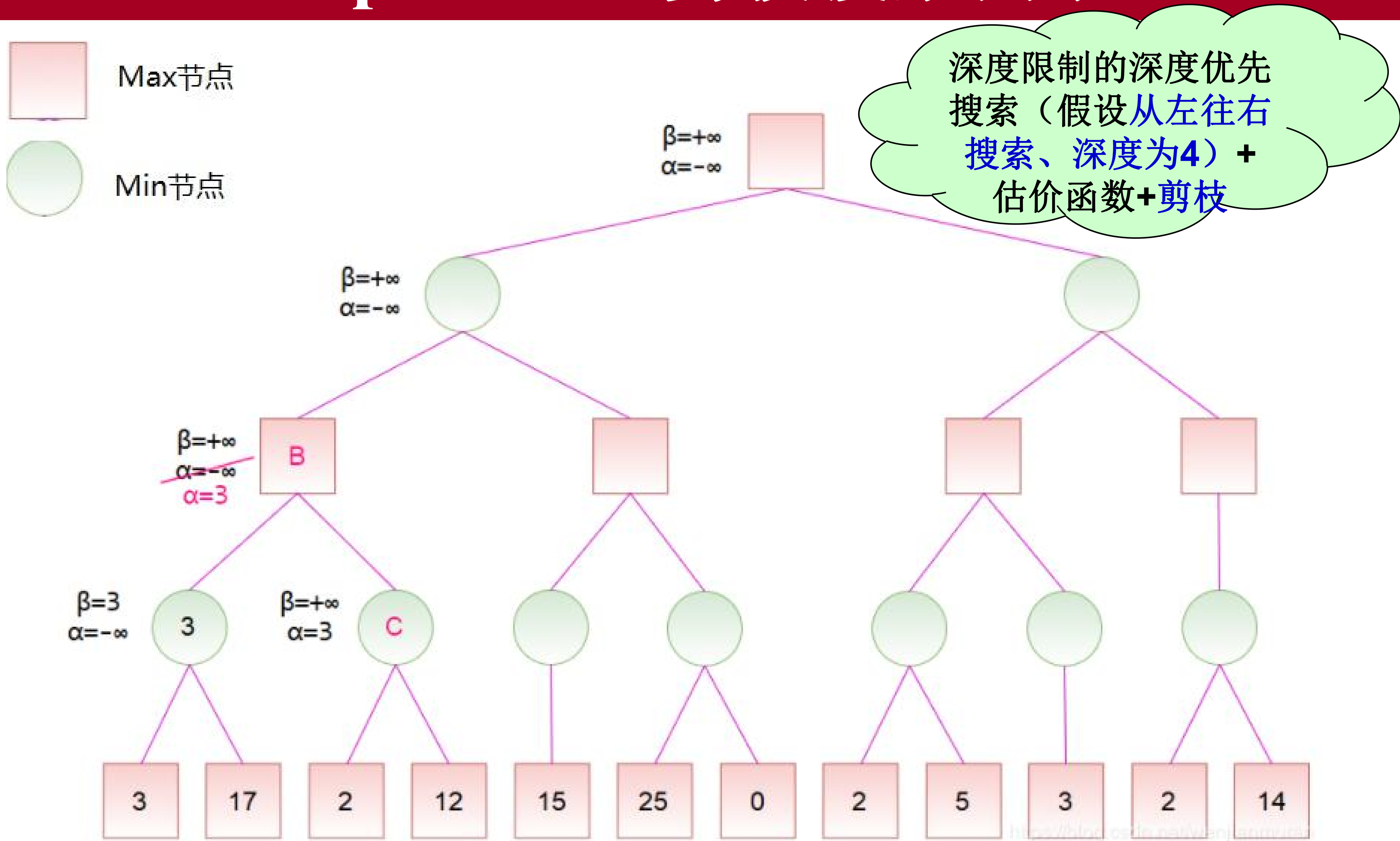


Alpha-Beta 剪枝搜索例子

深度限制的深度优先搜索（假设从左往右搜索、深度为4）+估价函数+剪枝

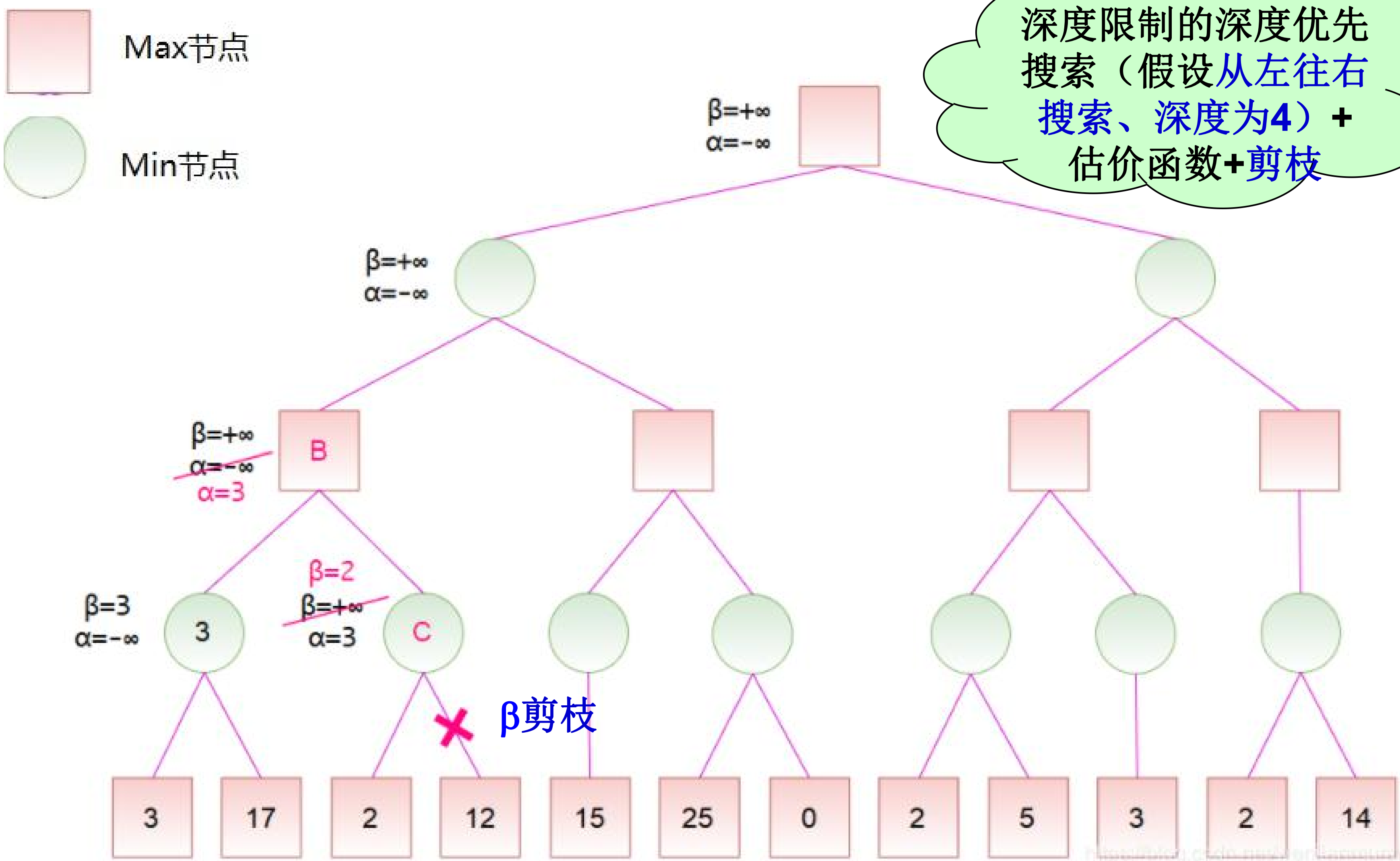


Alpha-Beta 剪枝搜索例子



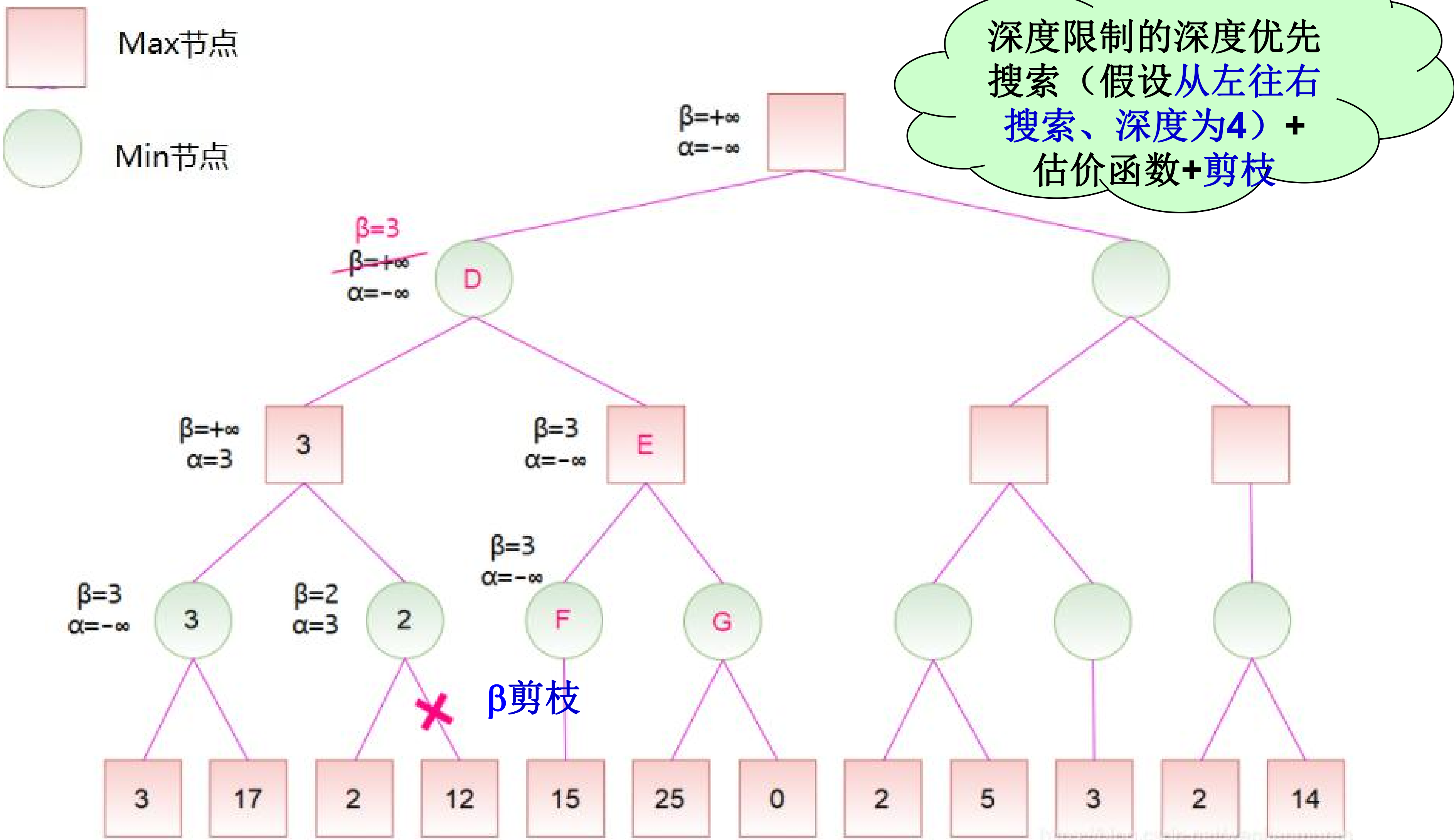
Alpha-Beta 剪枝搜索例子

深度限制的深度优先搜索（假设从左往右搜索、深度为4）+估价函数+剪枝

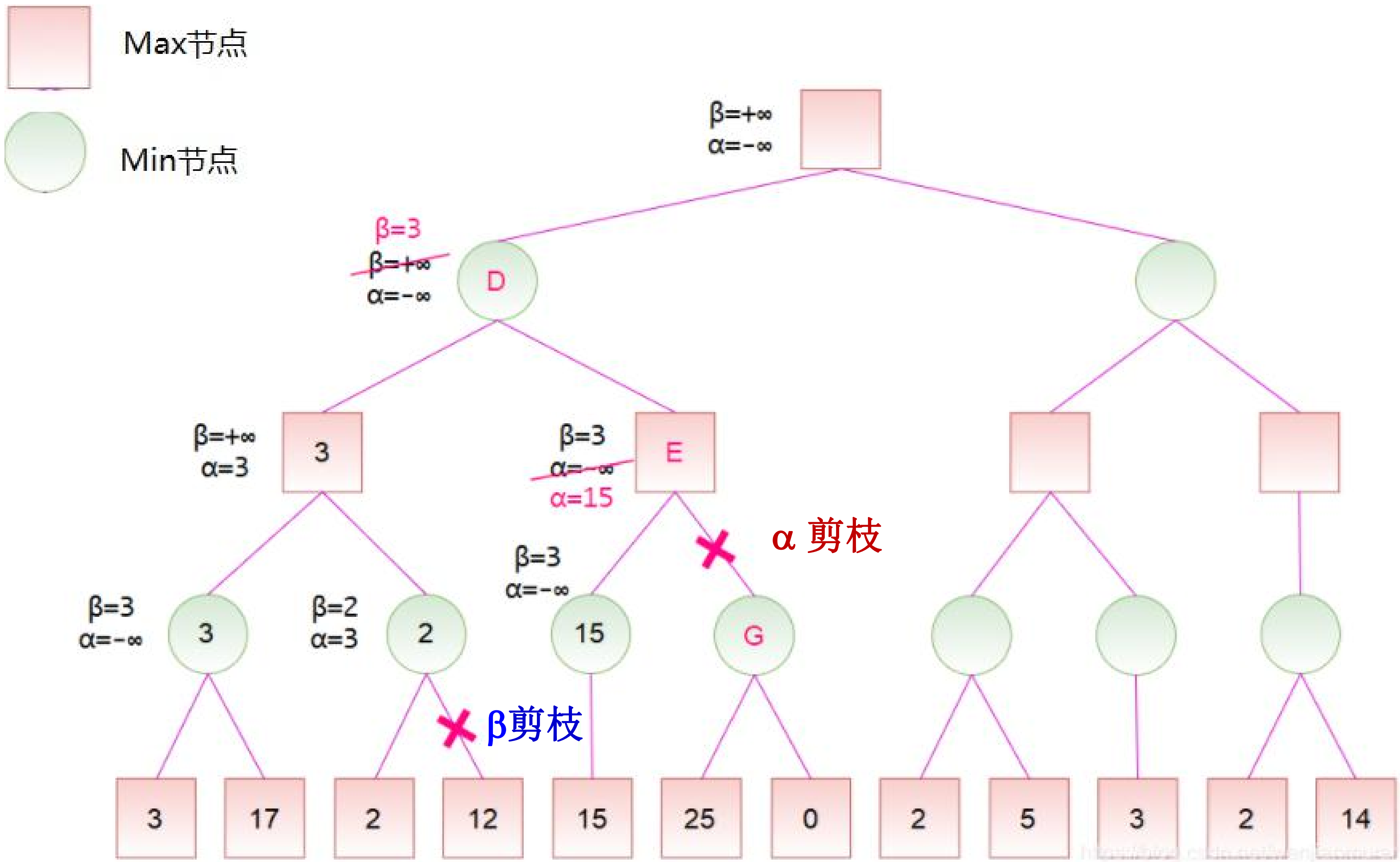


Alpha-Beta 剪枝搜索例子

深度限制的深度优先搜索（假设从左往右搜索、深度为4）+估价函数+剪枝

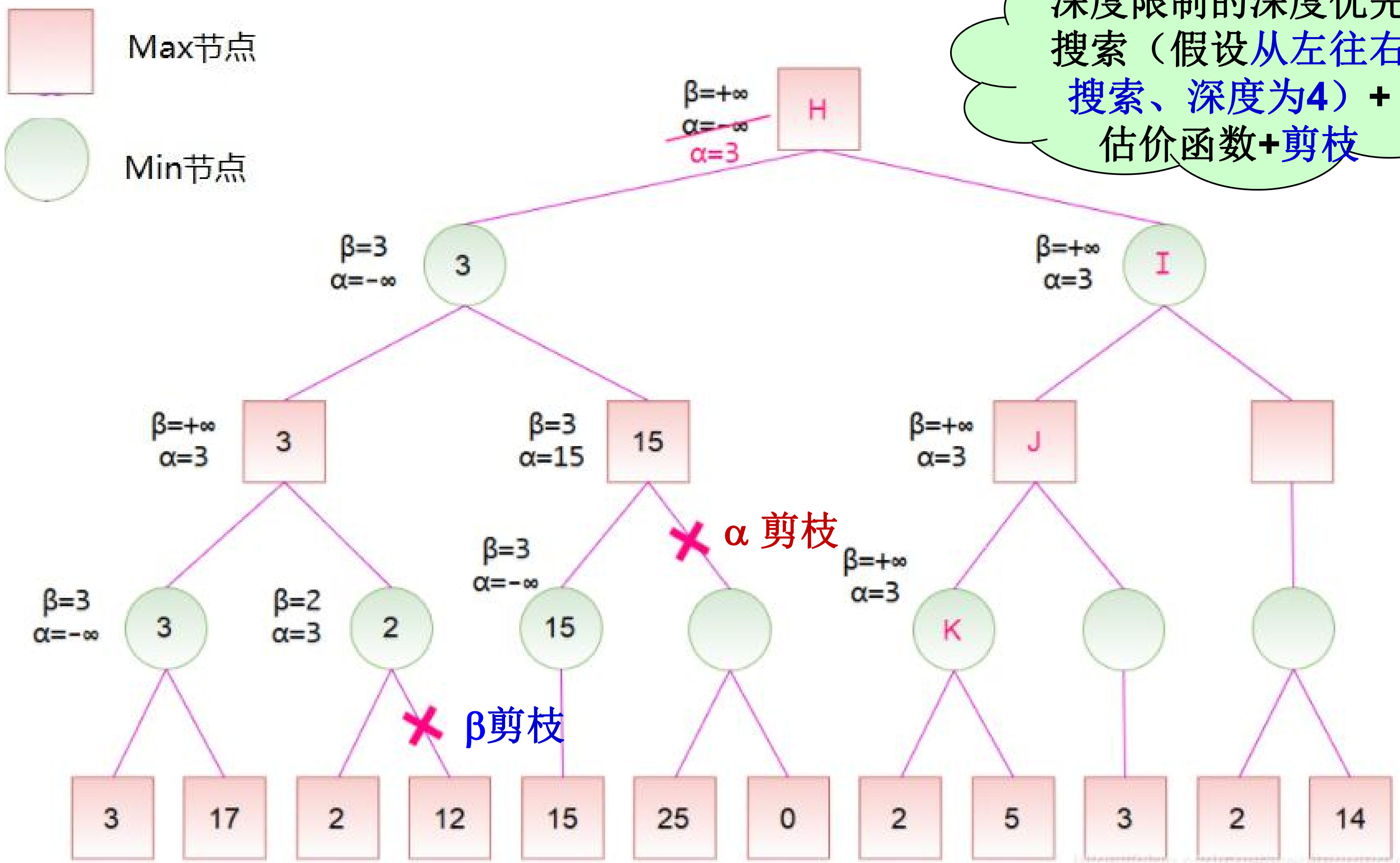


Alpha-Beta 剪枝搜索例子-课后作业



Alpha-Beta 剪枝搜索例子-课后作业

深度限制的深度优先搜索（假设从左往右搜索、深度为4）+估价函数+剪枝



α-β剪枝搜索(Alpha-Beta 剪枝搜索)

- 对棋局如何打分是α-β剪枝算法中非常关键的内容。
- 深蓝采用规则的方法对棋局打分：对不同的棋子按照重要程度给予不同的分数，还考虑棋子的位置赋予不同的权重、棋子之间的联系等。
- 对于国际象棋，进入残局后，棋子的多少可能就决定了胜负。
- 如果不采用α-β剪枝算法，要达到深蓝的下棋水平，每步棋需搜索17年——许峰雄博士

- 剪枝本身不影响算法输出结果
- 节点先后次序会影响剪枝效率
- 如果节点次序“恰到好处”，Alpha-Beta剪枝的时间复杂度为$O(b^{m/2})$
  极小极大搜索的时间复杂度为$O(b^m)$
  m 是游戏树的最大深度，在每个节点存在b个有效走法

蒙特卡洛树搜索

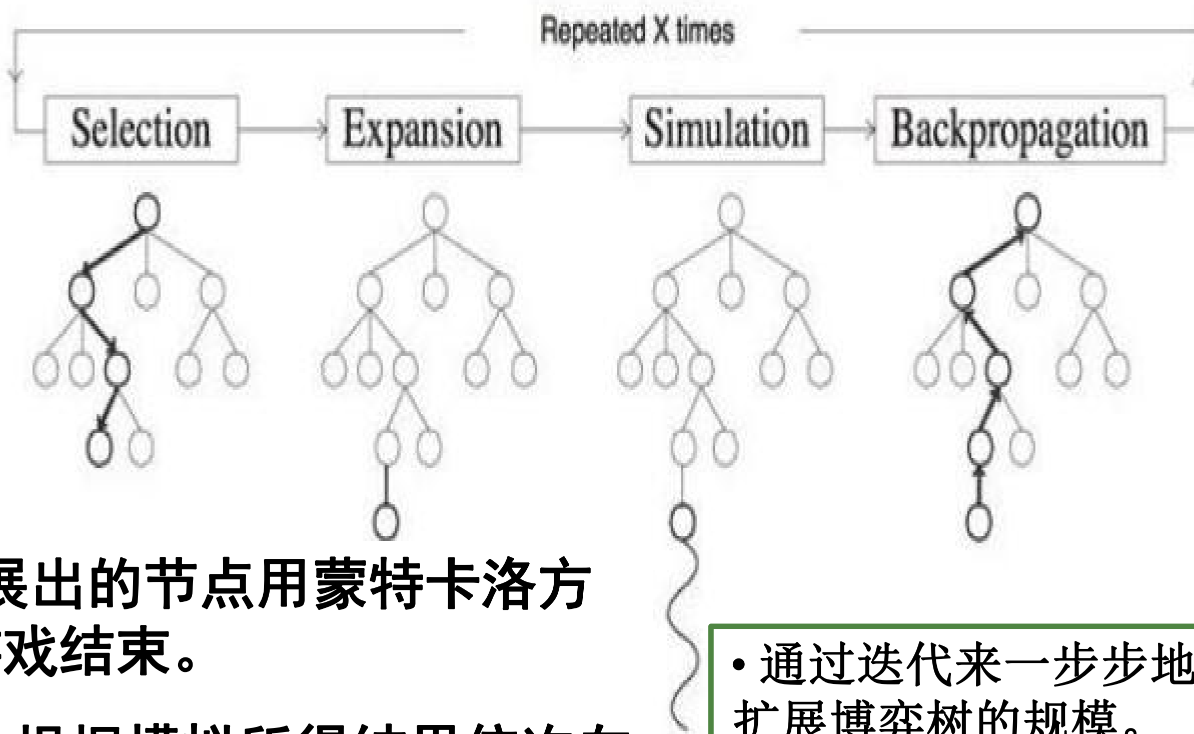

ALPHAGO
01:55:46
Google DeepMind
Challenge Match
LEE SEDOL
01:55:41
AlphaGO
Lee Se-dol
1202 CPUs, 176 GPUs,
100+ Scientists.
1 Human Brain,
1 Coff

基本的蒙特卡洛树状搜索算法

蒙特卡洛树搜索（MCTS）：在UCB1（Upper Confidence Bound，上限置信区间）基础上发展出来的一种解决多轮序贯博弈问题的策略。

1. **选择**(selection)：从根节点开始，向下递归选择子节点，直至选择一个叶子节点。

2. **扩展**(expansion)：向选定的点添加一个或多个子节点。



3. **模拟**(simulation)：对扩展出的节点用蒙特卡洛方法进行模拟，直到博弈游戏结束。

4. **反传**(BackPropagation)：根据模拟所得结果依次向上更新祖先节点估计值（获胜次数和访问次数）

- 通过迭代来一步步地扩展博弈树的规模。
- 每一个节点包含代表的局面，被访问的次数，累计评分。

L. Kocsis and C. Szepesvari, Bandit based Monte-Carlo Planning, *ECML*, 2006:282–293

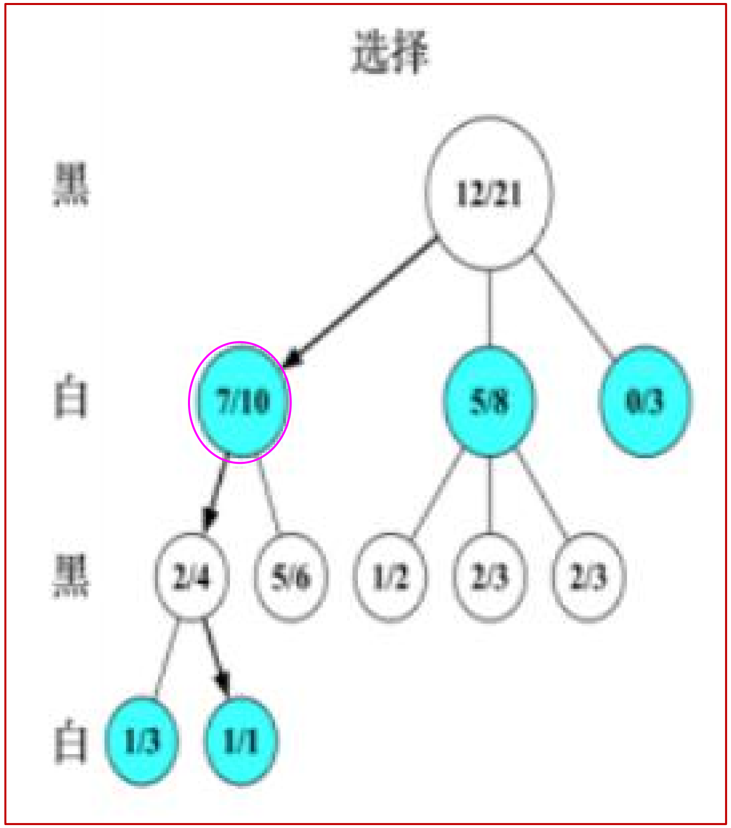
蒙特卡洛树搜索：例子

以围棋为例，假设**根结点**由**黑棋行棋**，由**UCB1（Upper Confidence Bound，上限置信区间）**公式来计算根节点的后续节点如下值，取一个值最大的节点作为后续节点：

左1：7/10对应的局面奖赏值为

$$\frac{7}{10}+\sqrt{\frac{\log(21)}{10}} = 1.252$$

UCB1策略：在选择已知回报最多的矿区（exploit）和尝试尽可能多的矿区（explore）之间平衡



图中每一个节点都代表一个局面，每一个局面记录两个值A/B：

A：该局面被访问中黑棋胜利次数。对于黑棋表示己方胜利次数，对于白棋表示己方失败次数（对方胜利次数）；

B：该局面被访问的总次数。

蒙特卡洛树搜索：例子

以围棋为例，假设根结点由黑棋行棋，由UCB1（Upper Confidence Bound，上限置信区间）公式来计算根节点的后续节点如下值，取一个值最大的节点作为后续节点：

左1：7/10对应的局面奖赏值为

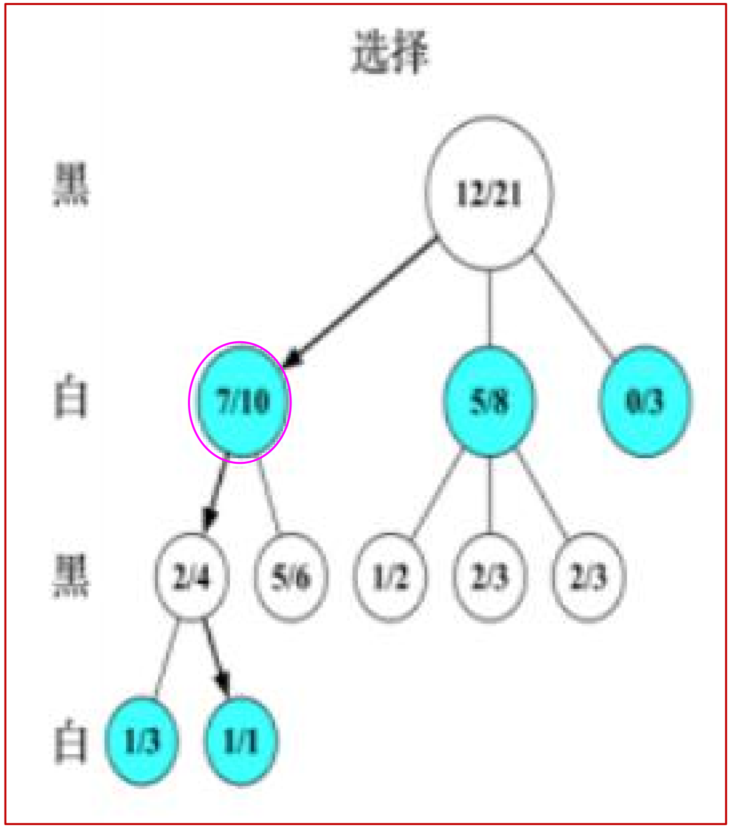
$$\frac{7}{10}+\sqrt{\frac{\log(21)}{10}}=1.252$$

左2，5/8对应的局面奖赏值为$\frac{5}{8}+\sqrt{\frac{\log(21)}{8}}=1.243$

左3，0/3对应的局面评估分数为

$$\frac{0}{3}+\sqrt{\frac{\log(21)}{3}}=1.007$$

由此可见，黑棋会选择局面7/10进行行棋。



图中每一个节点都代表一个局面，每一个局面记录两个值A/B：

A：该局面被访问中黑棋胜利次数。对于黑棋表示己方胜利次数，对于白棋表示己方失败次数（对方胜利次数）；

B：该局面被访问的总次数。

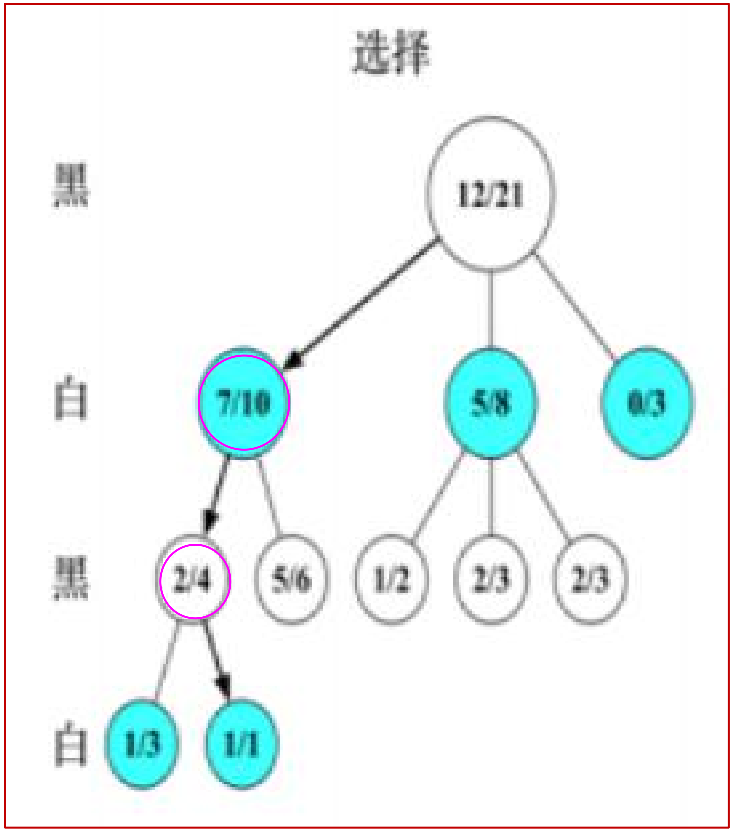
蒙特卡洛树搜索：例子

在节点7/10，由白棋行棋，评估该节点下面的两个局面，由UCB1公式可得（注意：此时A记录的的是白棋失败的次数，所以第一项为1-A/B）：

左1，2/4对应的局面奖赏为 $1-\frac{2}{4}+\sqrt{\frac{\log(10)}{4}}=1.26$

左2，5/6对应的局面奖赏为 $1-\frac{5}{6}+\sqrt{\frac{\log(10)}{6}}=0.786$

由此可见，白棋会选择局面2/4进行行棋。



图中每一个节点都代表一个局面，每一个局面记录两个值A/B：

A：该局面被访问中黑棋胜利次数。对于黑棋表示己方胜利次数，对于白棋表示己方失败次数（对方胜利次数）；

B：该局面被访问的总次数。

蒙特卡洛树搜索：例子

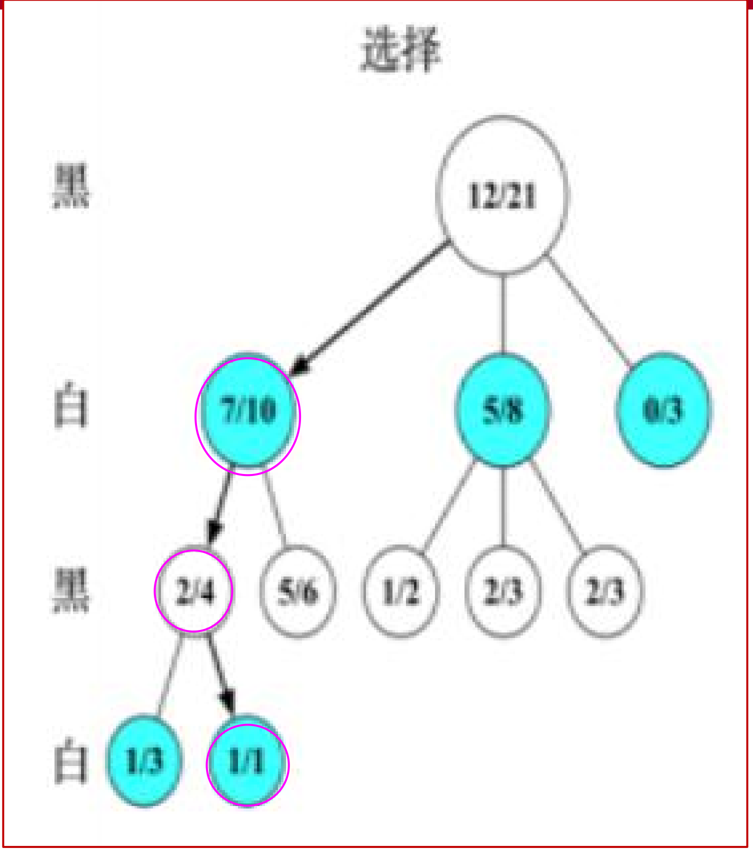
在节点2/4，黑棋评估下面的两个局面，由UCB1公式可得：：

左1，1/3对应的局面奖赏为$\frac{1}{3}+\sqrt{\frac{\log(4)}{3}}=$ 1.01

左2，1/1对应的局面奖赏$\frac{1}{1}+\sqrt{\frac{\log(4)}{1}}=$ 2.18

由此可见，黑棋会选择局面**1/1**进行行棋。此时已经到达叶子结点，需要进行扩展。

随机生成一个新节点。由于该新节点未被访问，所以初始化为**0/0**，接着在该节点下进行模拟。



图中每一个节点都代表一个局面，每一个局面记录两个值**A**/**B**：

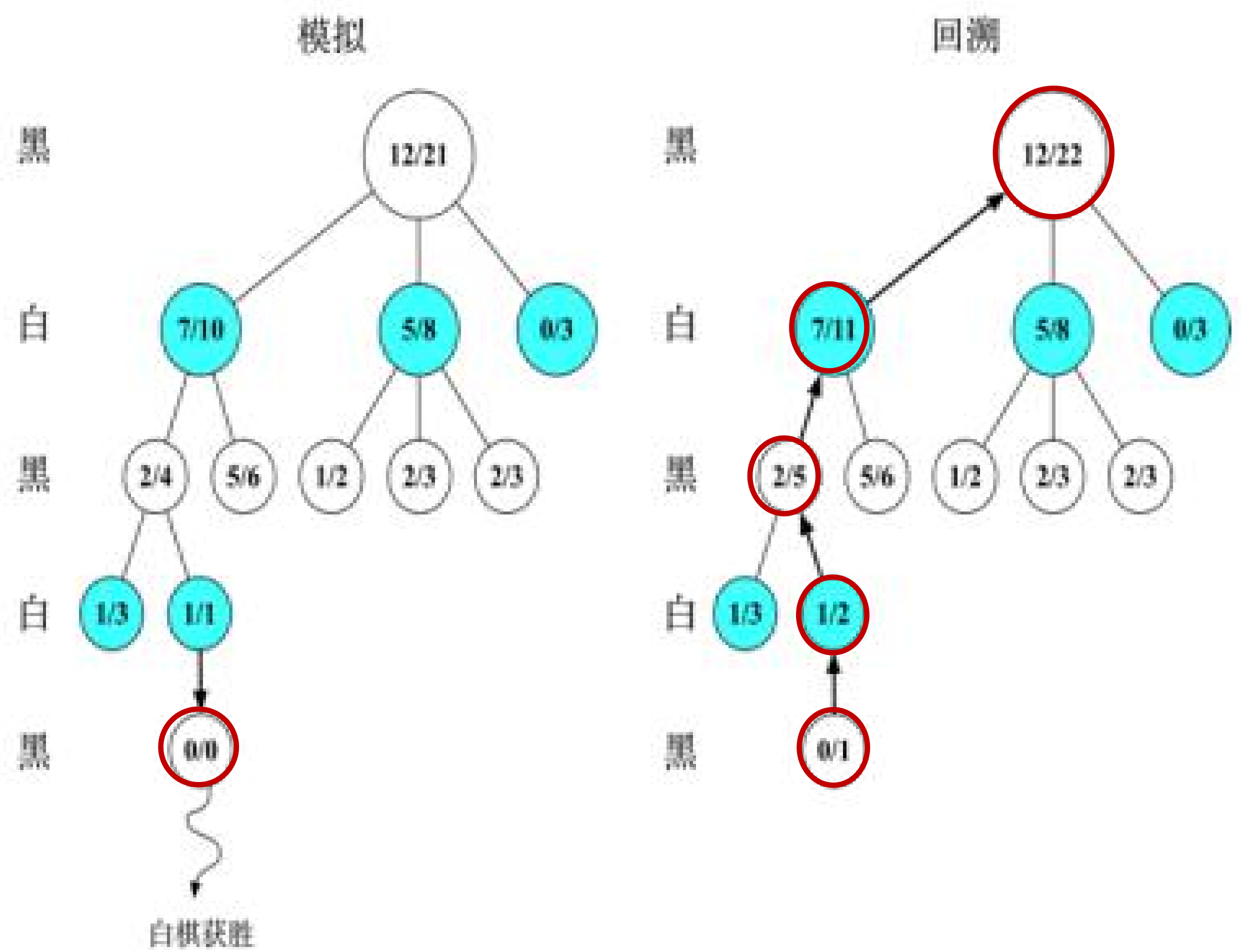
A：该局面被访问中黑棋胜利次数。对于黑棋表示己方胜利次数，对于白棋表示己方失败次数（对方胜利次数）；

B：该局面被访问的总次数。

蒙特卡洛树搜索：例子

假设经过一系列仿真行棋后，最终白棋获胜。

根据仿真结果来更新该仿真路径上每个节点的A/B值，该新节点的A/B值被更新为0/1，并向上回溯到该仿真路径上新节点的所有父辈节点，即所有父辈节点的A不变，B值加1。

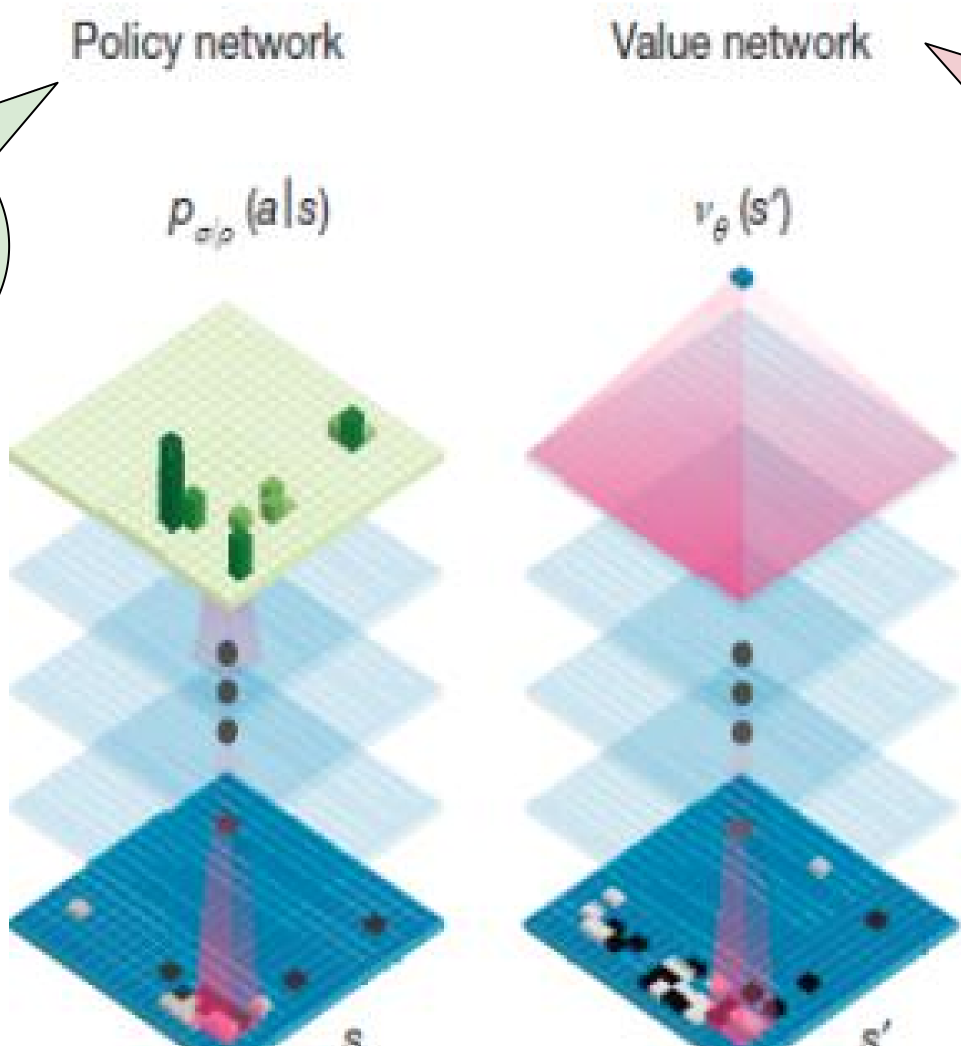


蒙特卡洛树搜索基于采样来得到结果、而非穷尽式枚举（虽然在枚举过程中也可剪掉若干不影响结果的分支）

图中每一个节点都代表一个局面，每一个局面记录两个值A/B。A：该局面被访问中黑棋胜利次数。对于黑棋表示己方胜利次数，对于白棋表示己方失败次数（对方胜利次数）；B：该局面被访问的总次数。

AlphaGo算法解读

用于决策，输出对应着棋盘上的一个概率分布



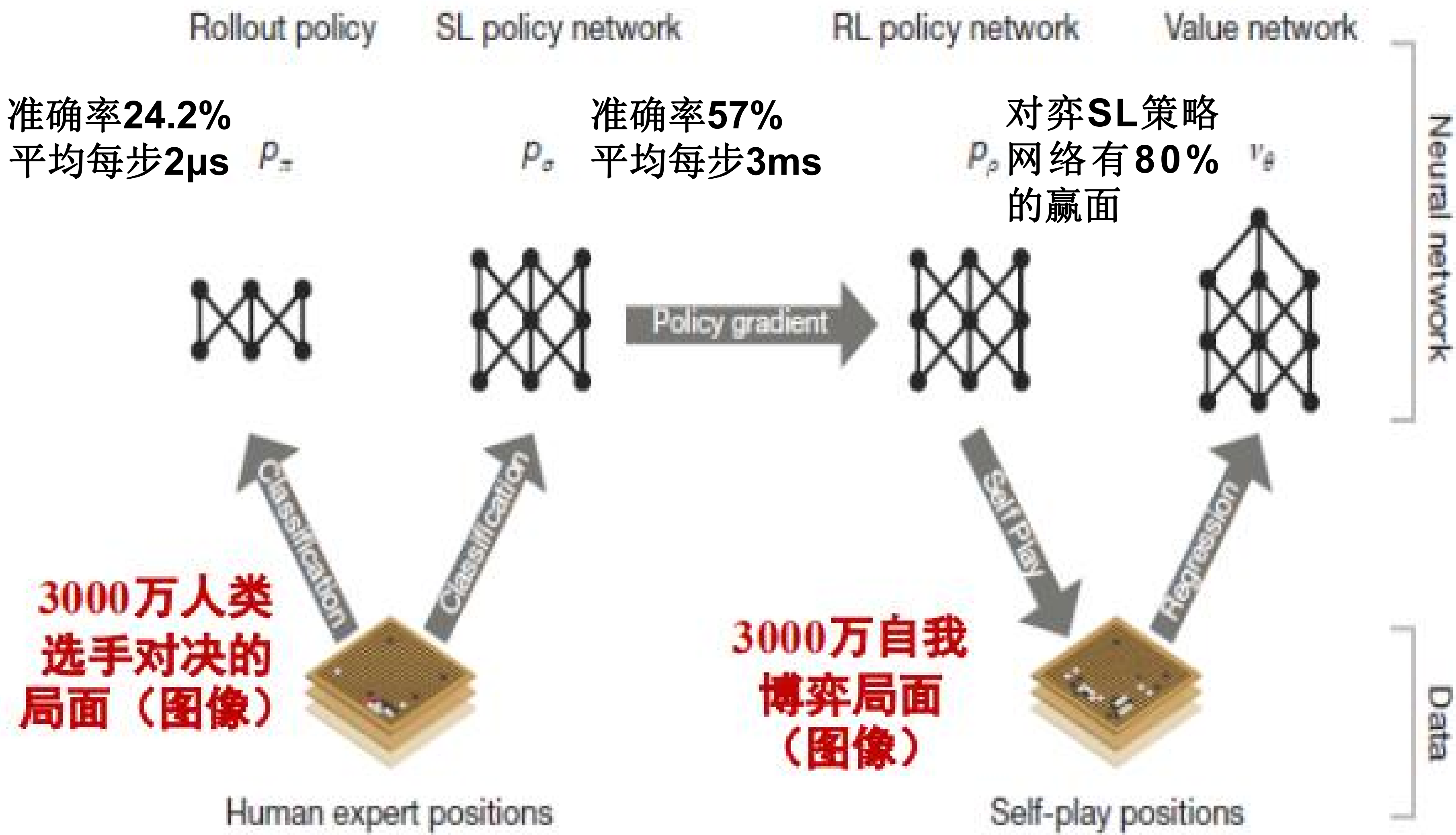
用于评估局面，输出一个标量，即当前局面落子后整盘游戏的胜算

$p_{\sigma,p}(a|s)$表示当前状态为s（局面）时，采取行动a后所得到的概率；$v_\theta(s')$表示当前状态为s'时，整盘棋获胜的概率。

AlphaGo利用了蒙特卡洛树状搜索与两个深度神经网络相结合的方法，以估值网络value network来评估大量落子间的优劣，而以策略网络policy network来选择落子。

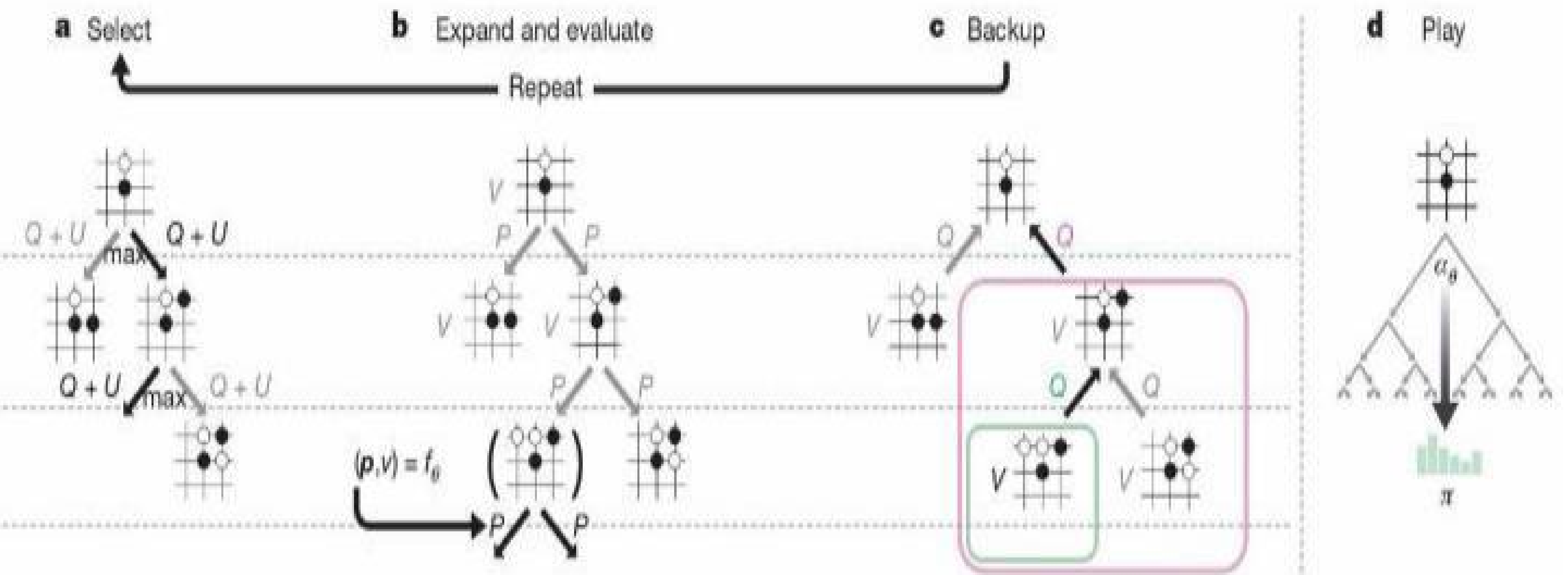
AlphaGo算法解读

能快速预估棋面价值（棋势



在MCTS的框架下引入两个卷积神经网络policy network和value network以改进纯随机的Monte Carlo模拟，并借助监督学习supervised learning和强化学习reinforcement learning训练这两个网络

MCTS 使用神经网络模拟落子选择的过程



蒙特卡洛树搜索中融入了策略网络和价值网络，给定节点v_0，将具有如下最大值的节点v选择作为v_0的后续节点：

$$\frac{Q(v)}{N(v)} + \frac{P(v|v_0)}{1 + N(v)}$$

- 这里$P(v|v_0)$的值由策略网络计算得到。
- 在模拟策略阶段(default policy)，AlphaGo不仅考虑仿真结果，而且考虑价值网络计算结果。
- 策略网络和价值网络是离线训练得到的。

AlphaGo的实现原理

围棋是完全信息博弈，从理论上来说可以通过暴力搜索所有可能的对弈过程来确定最优的走法

基本算法

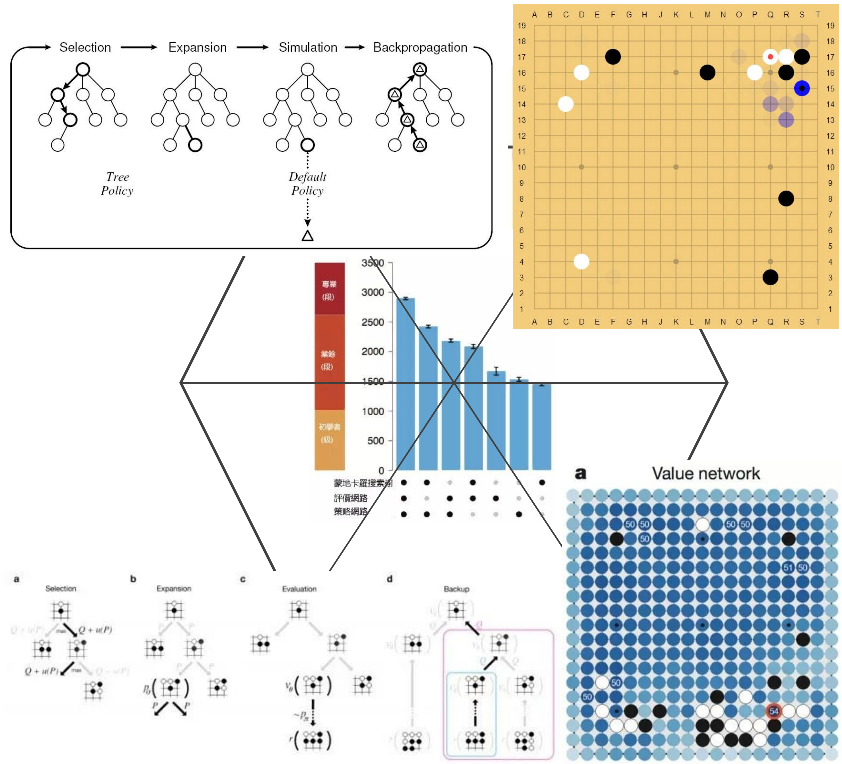
MCTS（蒙特卡洛树搜索）

给胜率高的点分配更多的计算力
任意时间算法，计算越多越精确
1、选择　2、扩展　3、模拟　4反传

快速模拟

Rollout（随机模拟走子）

通过随机模拟走子胜率来判定形势
速度很快（1ms/盘）
随机性与合理性的平衡



控制宽度（250）

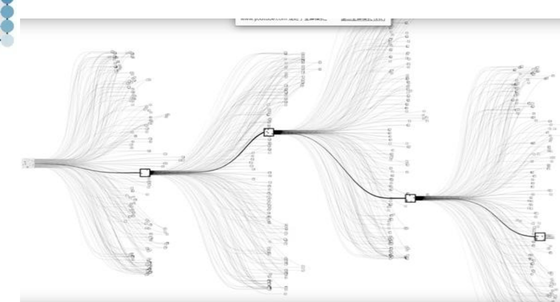
Policy Network（策略网络）

13个卷积层，每层192个卷积核，
每个卷积核3*3，参数个数800万+
GPU　3ms/步
预测准确率 57%

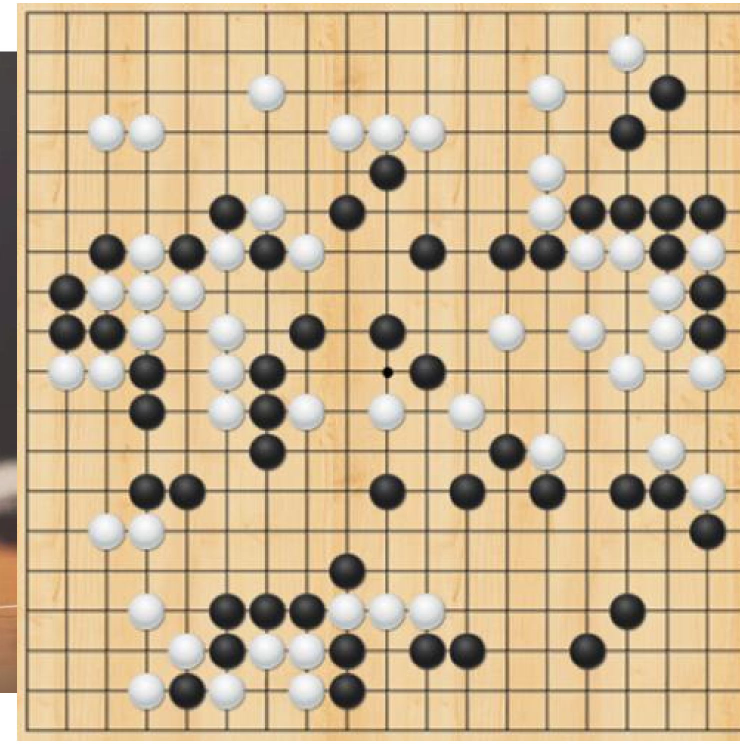
控制深度（150）

Value Network（价值网络）

在每个分支节点直接判断形势
与Rollout随机模拟相结合，互为补充

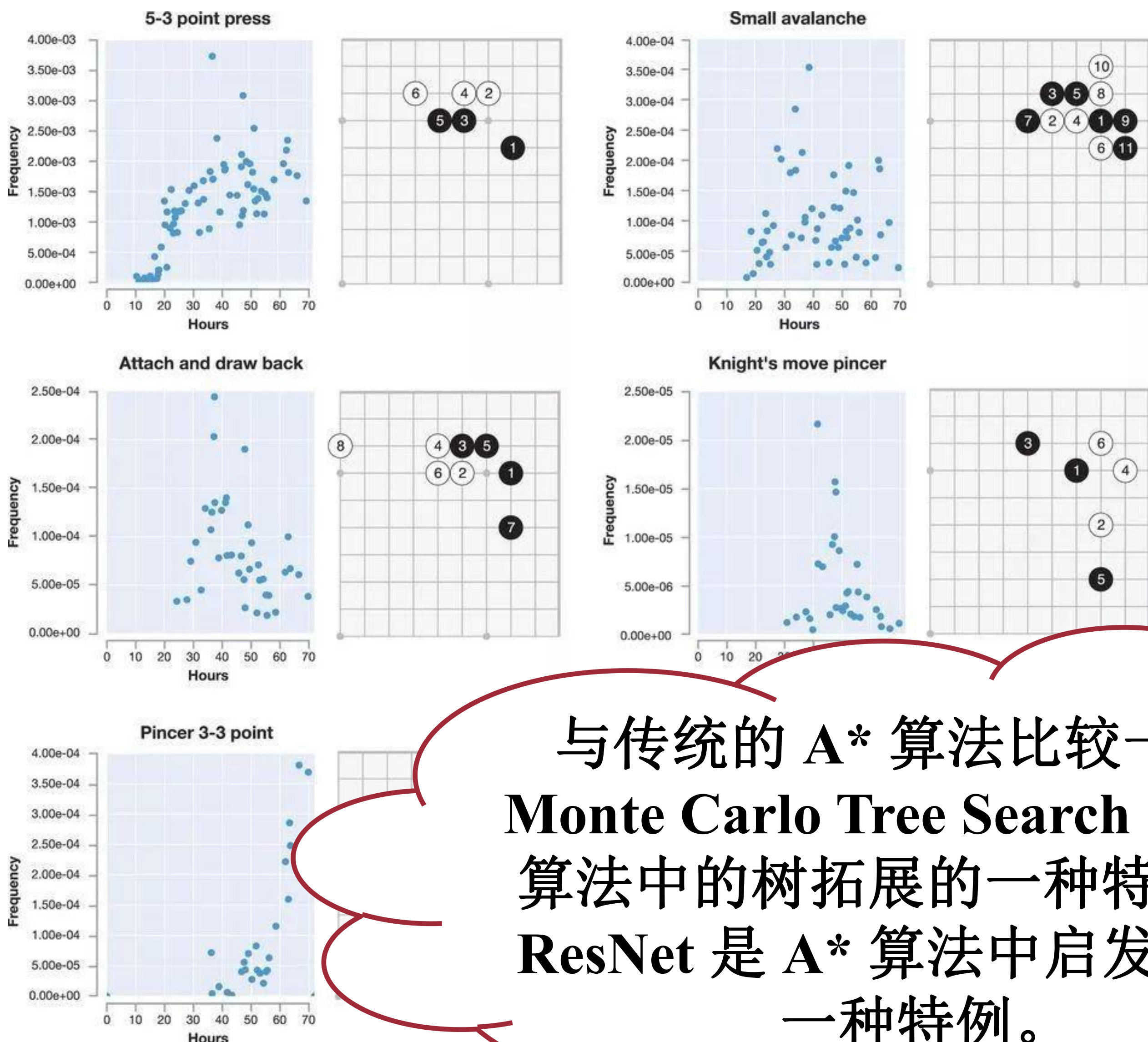


David Silver, et.al., Mastering the game of Go with Deep Neural Networks and Tree Search, Nature, 529:484-490,2016



◆ **Monte Carlo Tree Search**：不穷举所有组合，找到最优或次优位置。
◆ 深度学习启发函数：用 深度残差网络 （ **ResNet** ）来改进围棋投子策略，输入是当前的棋面，输出是当前棋面的赢率、下一手投子的位置及其概率。
◆ 置信上限： 鼓励探索新的投子位置，越是以往很少投子的位置， 得分越高。

定式（Joseki）与投子位置热力图



其中某些位置被投子的概率，比其它位置显著地高，这些位置加上前面几手的落子位置和相应的棋面，就是围棋定式。

与传统的 A* 算法比较一下，**Monte Carlo Tree Search** 只是 A* 算法中的树拓展的一种特例，而 **ResNet** 是 A* 算法中启发函数的一种特例。

- **MOOC上：第7讲的课堂讨论、单元测试、在线作业**
- **网络教学平台作业W5-2**
- **准备下周二的A*算法实验**

THE END