华东师范大学计算机科学与技术学院
School of Computer Science and Technology

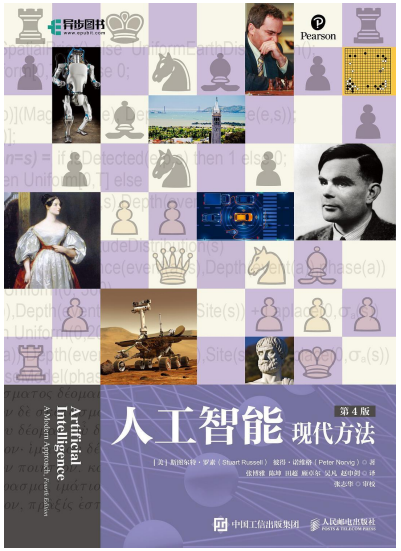
2024年春季学期

《人工智能》

第5讲：博弈搜索

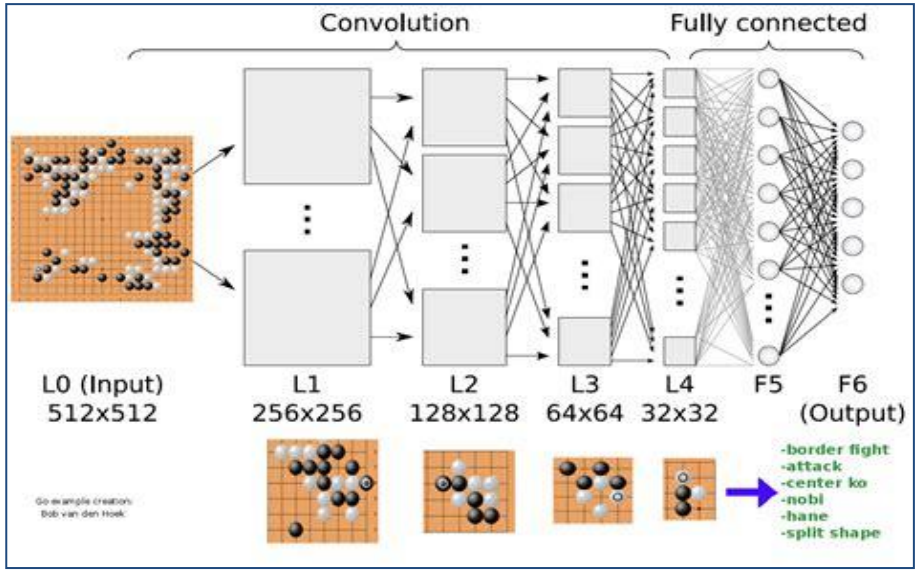
第5章 博弈搜索

- **主要内容**
  - 5.1 博弈背景
  - 5.2 极大极小搜索
  - 5.3 α-β剪枝
  - 5.4 本章小结
  - 5.5 蒙特卡洛树搜索
- **参考书目**
  - 《人工智能：现代方法（第4版）》（美）罗素,（美）诺维格，人民邮电出版社，2022。
    Ch5: pp.118-141.
  - 《人工智能导论：模型与算法 》，吴飞编著，高等教育出版社，2020年。
    Ch3: pp.63-105.



5.1 博弈背景

- 围棋人机大战



5.1 博弈背景

- **博弈问题的基本假设**
  - 两个棋手交替地走棋
  - 比赛的最终结果，是赢、输和平局中的一种
  - 可用图搜索技术进行，但效率很低
  - 博弈的过程，是寻找置对手于必败态的过程
  - 双方都无法干预对方的选择

5.2 极大极小搜索

- **基本思想**
  - 下棋的双方是对立的
  - 一方为“正方”，这类节点称为“MAX”节点
  - 另一方为“反方”，这类节点称为“MIN”节点
  - 正方从所有子节点中，选取具有最大评估值的节点
  - 反方从其所有子节点中，选取具有最小评估值的节点
  - 反复进行这种选取，就可以得到双方各个节点的评估值。这种确定棋步的方法，称为极大极小搜索法。

5.2 极大极小搜索

- **对各个局面进行评估**
  - 评估的目的：对后面的状态提前进行考虑，并且以各种状态的评估值为基础作出最好的走棋选择。
  - 评估的方法：用评价函数对棋局进行评估。赢的评估值设为$+\infty$，输的评估值设为$-\infty$，平局的评估值设为0。
  - 评估的标准：由于下棋的双方是对立的，只能选择其中一方为评估的标准方。

所有评估站在正方的立场！

极大极小搜索算法

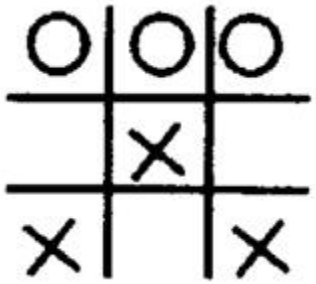
1. T:=(s, max), Open:=(s), Closed:=();
2. Loop1: // 扩展深度至k
3. If Open=( ) Then Goto Loop2
3. n:=First(Open), Remove(n,Open), Add(n,Closed)
4. If f(n)=-∞, +∞, 0, Then Goto Loop1 // 可以被判定
   Else $\{n_i\}$:=Expand(n) , Add($\{n_i\}$,T)
   If d(n_i)<k Then Add(n_i,Open), Goto Loop1
   Else 计算f(n_i), Goto Loop1
5. Loop2: // 赋值
6. If Closed=() Then Goto Loop3
   Else n_p:=First(Closed)
7. If (n_p Is Max) And (f(n_{pi})有值) // n_{pi}是n_p的子结点，且都有值
   Then f(n_p):=max f(n_{pi}), Remove(n_p,Closed)
   If (n_p Is MIN) And (f(n_{pi})有值) // n_{pi}是n_p的子结点，且都有值
   Then f(n_p):=min f(n_{pi}), Remove(n_p,Closed)
   If n_p 未被赋值, Then MoveTail(n_p, Closed)
8. Goto Loop2
9. Loop3:
10. If f(s)有值 Then Exit(End or M(Move, T)) // s被赋值，结束该步

向下扩展

向上赋值

例：一字棋

- **在九宫格棋盘上，两位选手轮流在棋盘上摆各自的棋子(每次一枚)，谁先取得三线的结果就取胜。**
  - 设程序方MAX的棋子用(×)表示， MAX先走。
  - 对手MIN的棋子用(o)表示。
  - 例如：

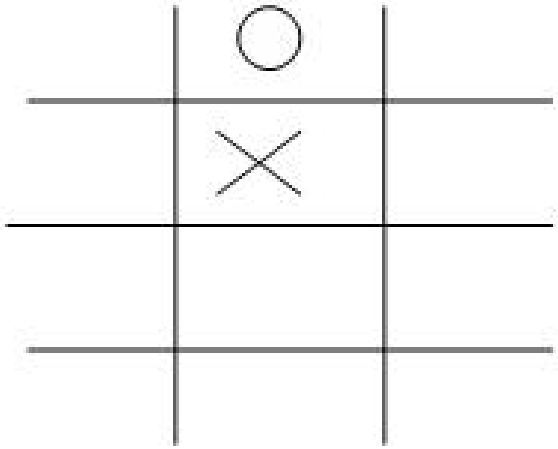


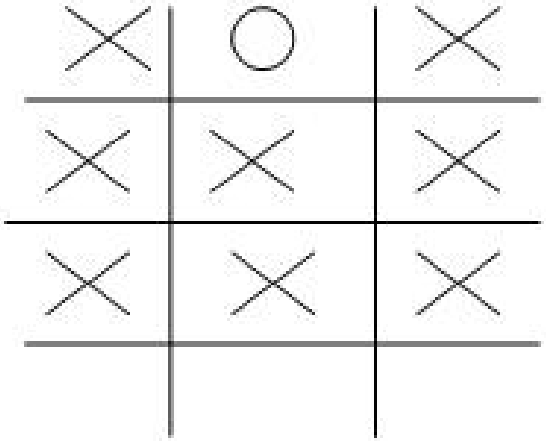
MIN取胜

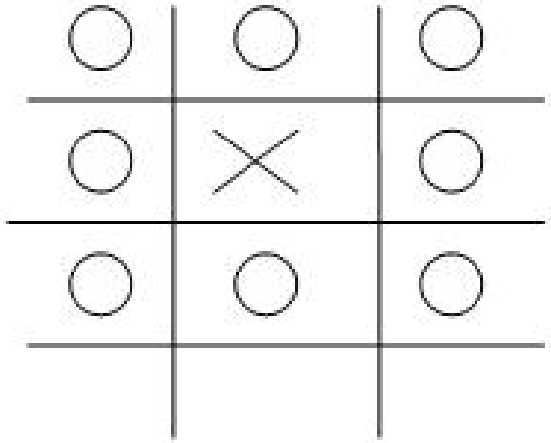
- **估计函数**
  - f(p)=(所有空格都放上MAX的棋子之后，MAX三子成线数)-(所有空格都放上MIN的棋子之后，MIN三子成线数);
  - 若P是MAX获胜的格局，则f(p)=+∞；
  - 若P是MIN获胜的格局，则f(p)＝-∞ 。

例：一字棋

- 评估函数

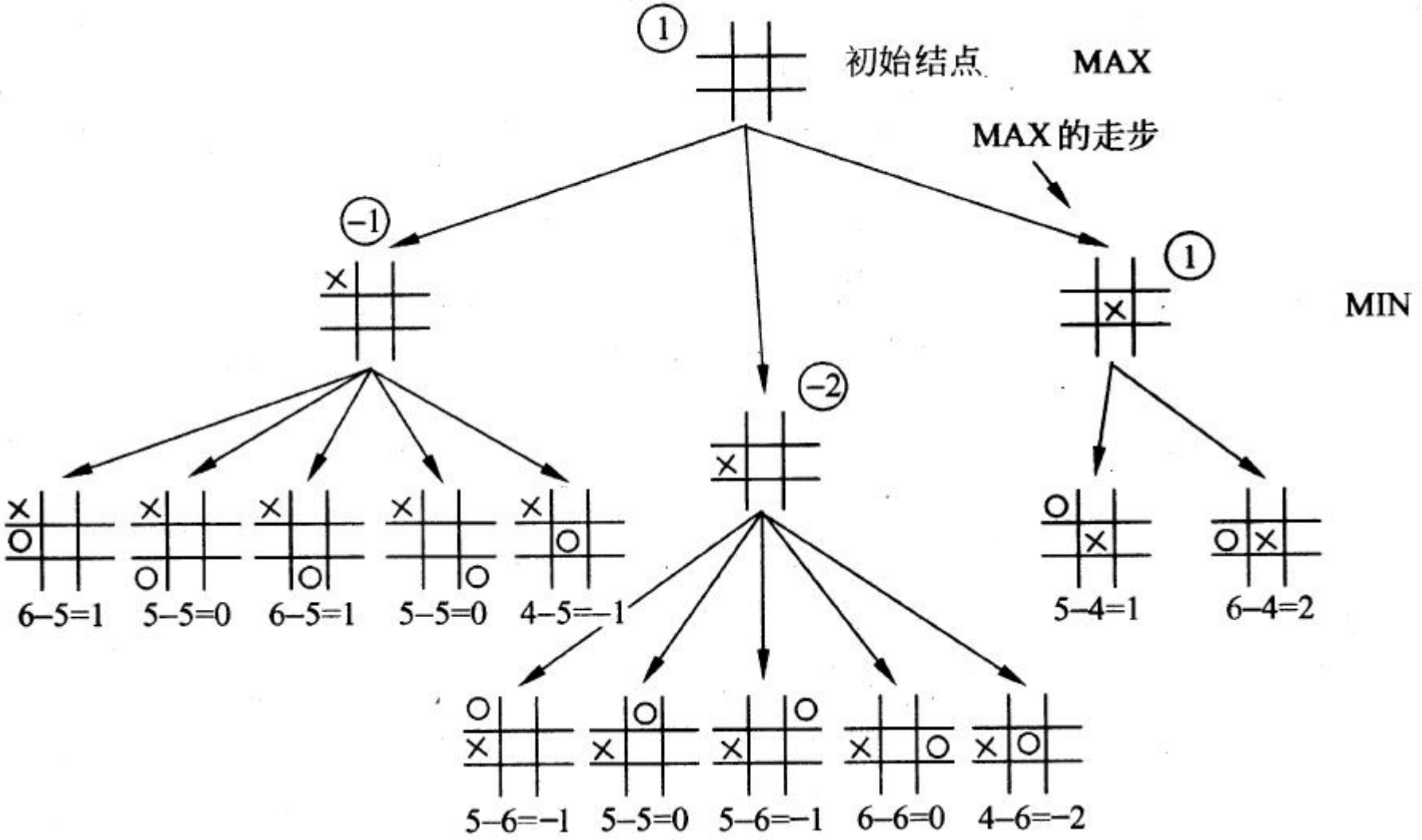






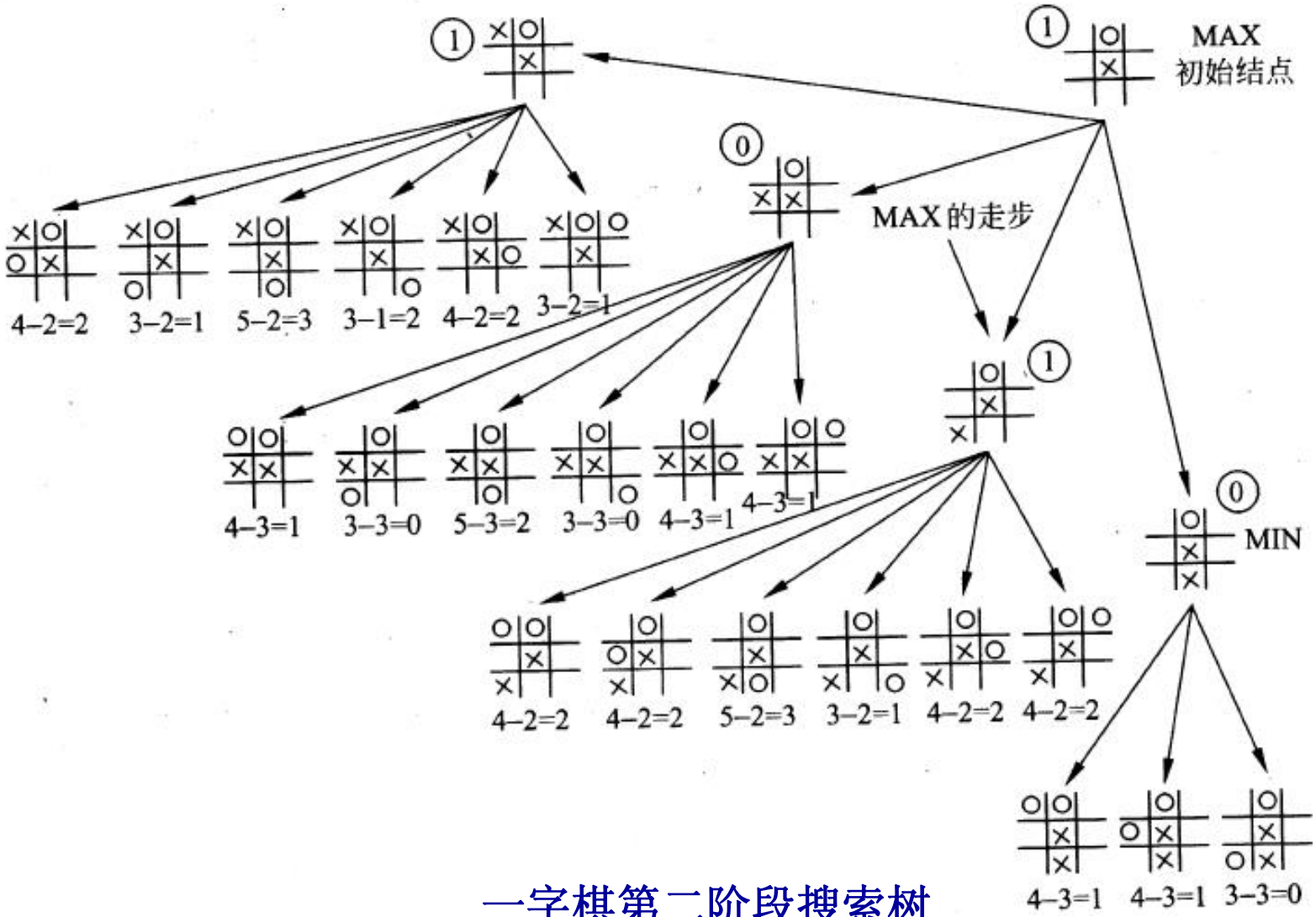
$f(p)=6-4=2$

例：一字棋



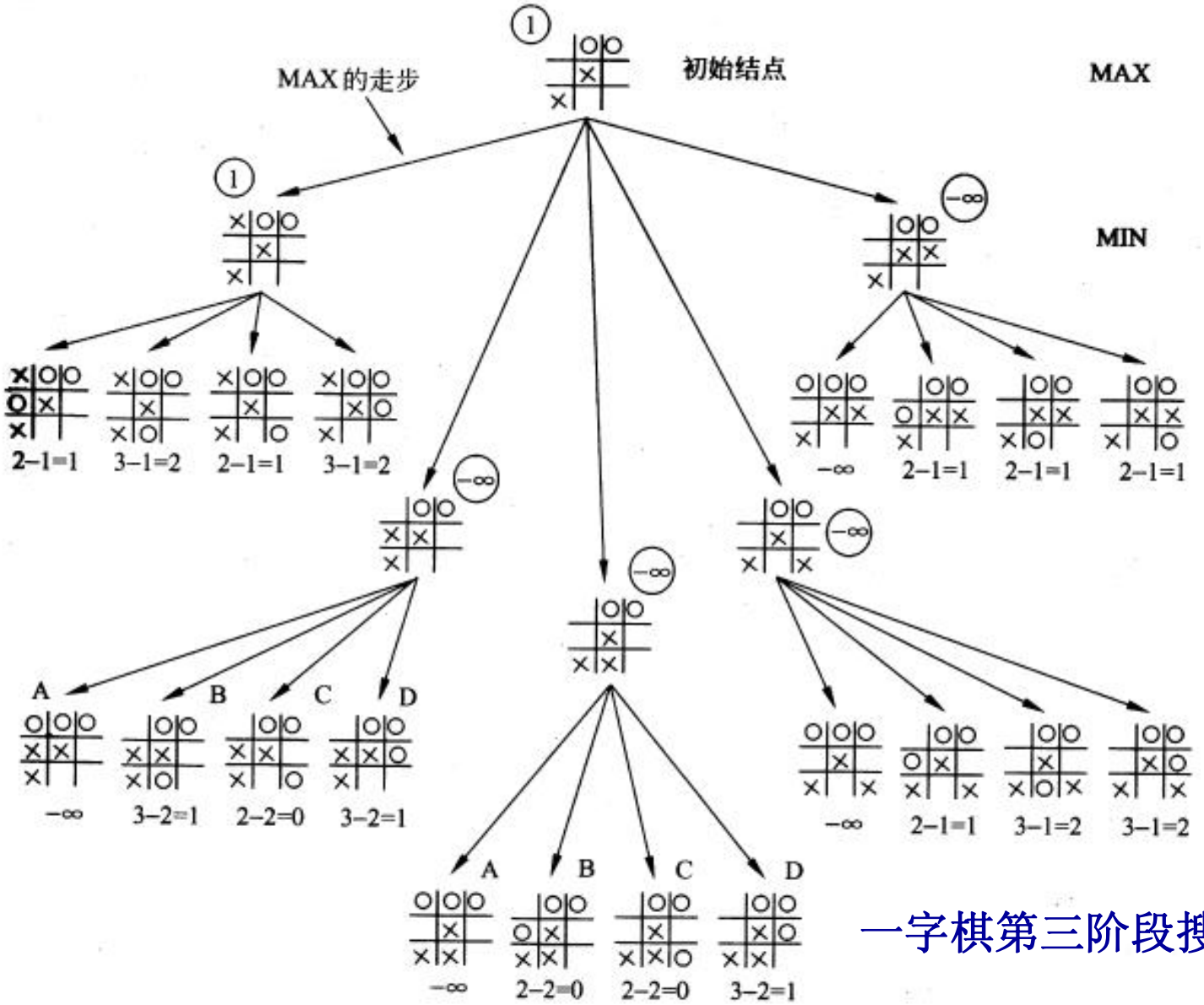
一字棋第一阶段搜索树

例：一字棋



一字棋第二阶段搜索树

例：一字棋

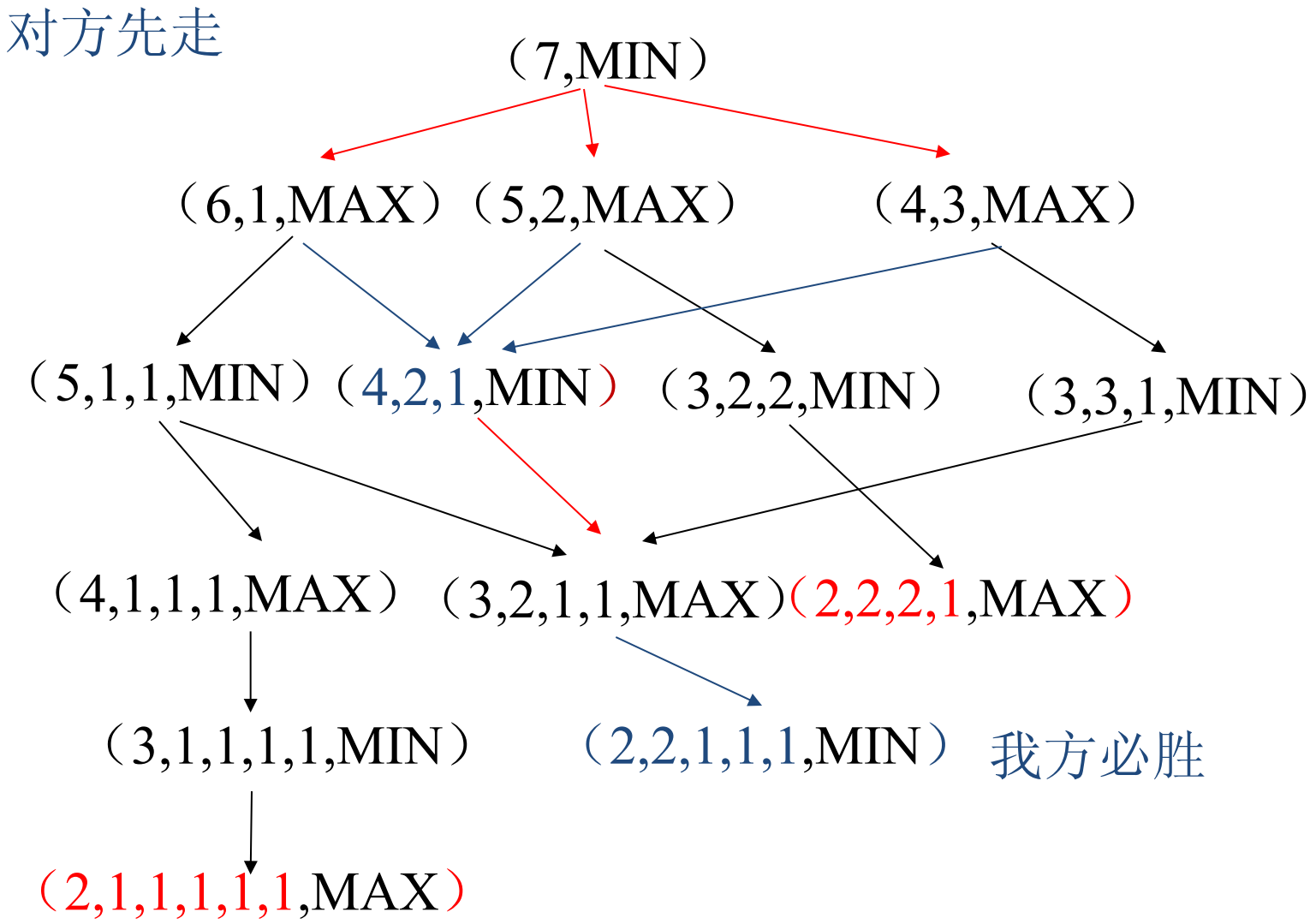


一字棋第三阶段搜索树

例：分钱币

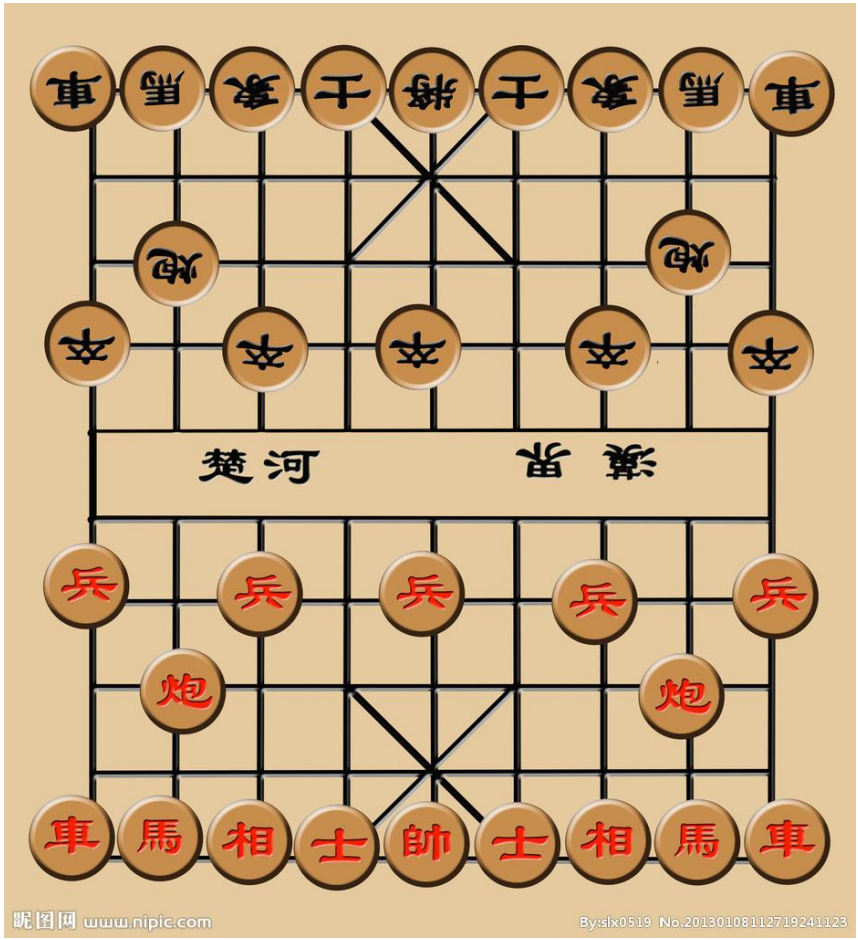
- **问题描述**
  - 有一堆数目为N的钱币，由两位选手轮流进行分堆，要求每个选手每次只把其中某一堆分成数目不等的两小堆。例如，选手甲把N分成两堆后，轮到选手乙就可以挑其中一堆来分，如此进行下去，直到有一位选手先无法把钱币再分成不相等的两堆时就得认输。

例：分钱币


对方先走
（7,MIN）
（6,1,MAX）
（5,2,MAX）
（4,3,MAX）
（5,1,1,MIN）
（4,2,1,MIN）
（3,2,2,MIN）
（3,3,1,MIN）
（4,1,1,1,MAX）
（3,2,1,1,MAX）
（2,2,2,1,MAX）
（3,1,1,1,1,MIN）
（2,2,1,1,1,MIN）
我方必胜
（2,1,1,1,1,1,MAX）

例：中国象棋

- 一盘棋平均走50步，总状态数约为10的161次方。
- 假设1毫微秒走一步，约需10的145次方年。
- 结论
  - 不可能穷举
  - 对小规模问题有效



5.3 α-β剪枝

- α-β剪枝法的引入
  - 极大极小搜索算法中，必须求出所有终端节点的评估值，当预先考虑的棋步比较多时，计算量会大大增加。
  - 为了提高搜索的效率，引入了通过对评估值的上下限进行估计、从而减少需进行评估的节点范围的α-β剪枝法。

5.3 α-β剪枝

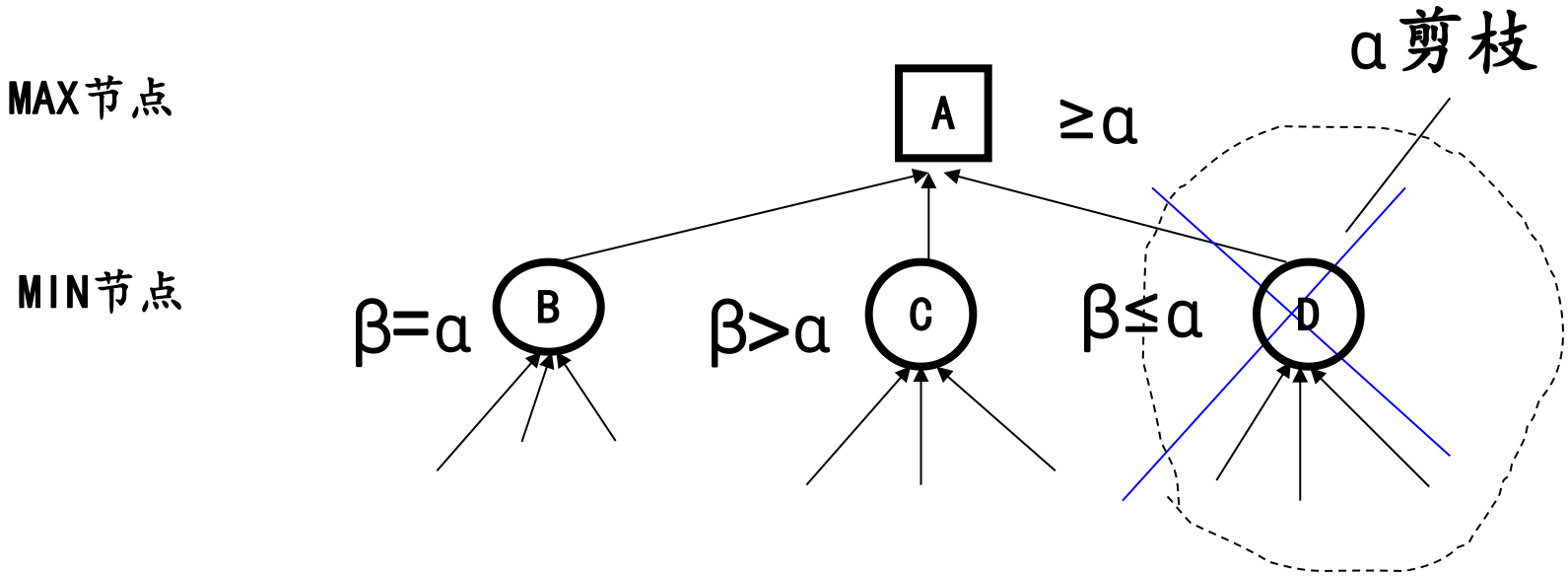
- **MAX节点的评估下限值α**
  - 作为正方出现的MAX节点，假设它的MIN子节点有N个，那么当它的第一个MIN子节点的评估值为α时，则对于其它的子节点，如果有高过α的，就取那最高的值作为该MAX节点的评估值；如果没有，则该MAX节点的评估值为α。总之，该MAX节点的评估值不会低于α，这个α就称为该MAX节点的评估下限值。
- **MIN节点的评估上限值β**
  - 作为反方出现的MIN节点，假设它的MAX子节点有N个，那么当它的第一个MAX子节点的评估值为β时，则对于其它子节点，如果有低于β的，就取那个低于β的值作为该MIN节点的评估值；如果没有，则该MIN节点的评估值取β。总之，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。

5.3 α-β剪枝

- 极大节点的下界为α
- 极小节点的上界为β
- 剪枝的条件：
  - 后辈节点的β值≤祖先节点的α值时， α剪枝
  - 后辈节点的α 值≥祖先节点的β值时， β剪枝
- 简记为：
  - 极小≤极大，剪枝
  - 极大≥极小，剪枝

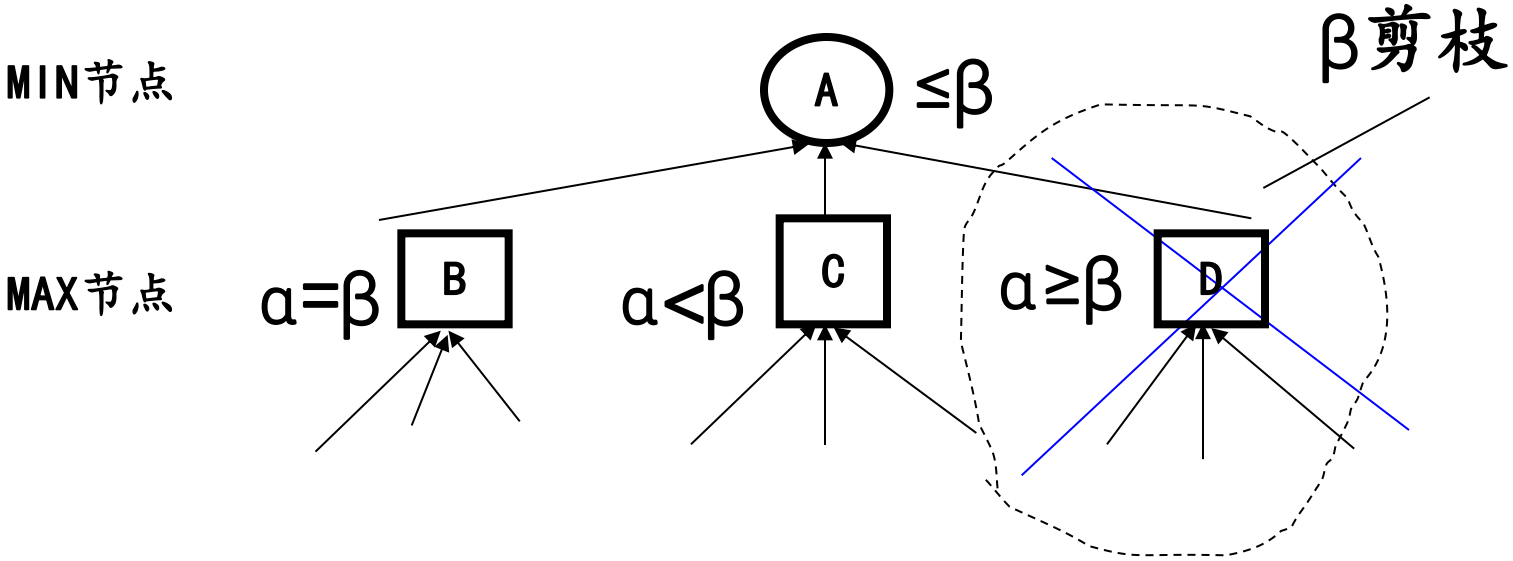
5.3 α-β剪枝

- α剪枝法
  - 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。

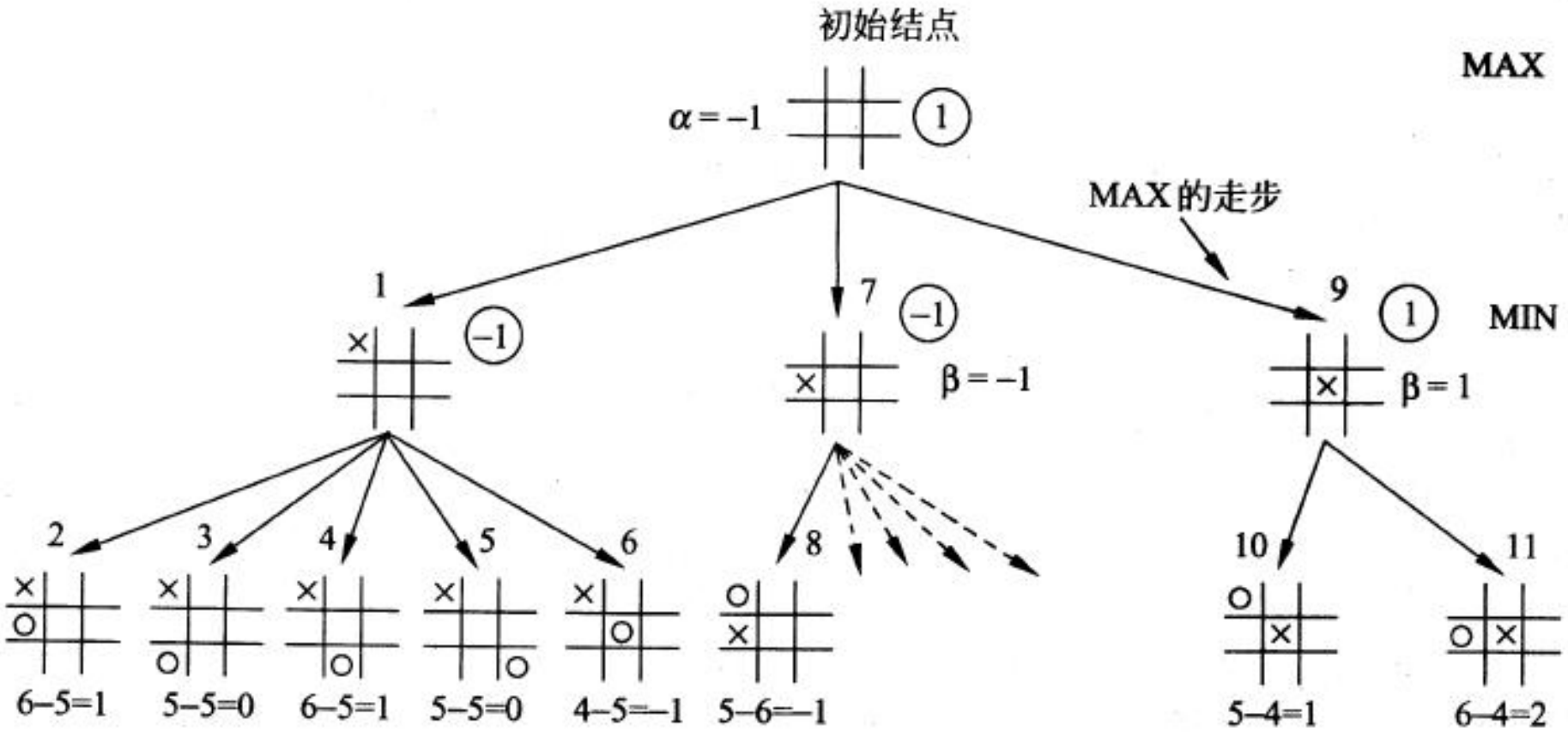


5.3 α-β剪枝

- β剪枝法
  - 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。

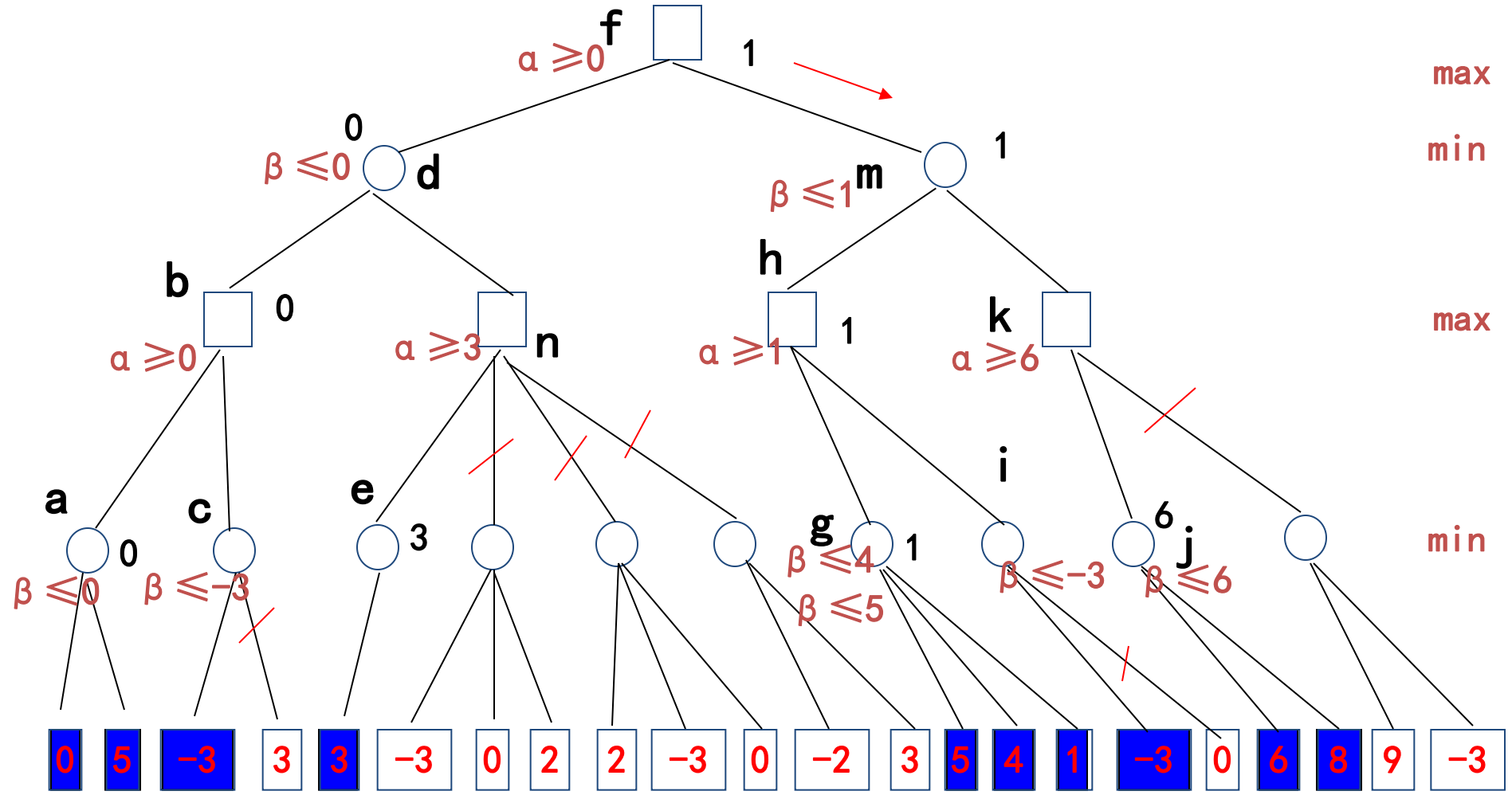


例：一字棋



一字棋第一阶段α-β剪枝方法

例： α-β剪枝


f
α ⩾0
1
max
0
β ⩽0
d
1
β ⩽1
m
min
b
0
α ⩾0
α ⩾3
n
h
1
α ⩾1
k
α ⩾6
max
a
0
β ⩽0
c
β ⩽−3
e
3
g
β ⩽4
1
β ⩽5
i
β ⩽−3
6
j
β ⩽6
min
0 5 −3 3 3 −3 0 2 2 −3 0 −2 3 5 4 1 −3 0 6 8 9 −3

5.4 本章小结

- 博弈是一类特殊的搜索问题
- 极大极小搜索算法
  - 基本的博弈搜索算法
  - 效率低，针对小规模问题
- α-β剪枝算法
  - 通过剪去明显不可能访问的分支，减小搜索空间，提高效率

谢谢

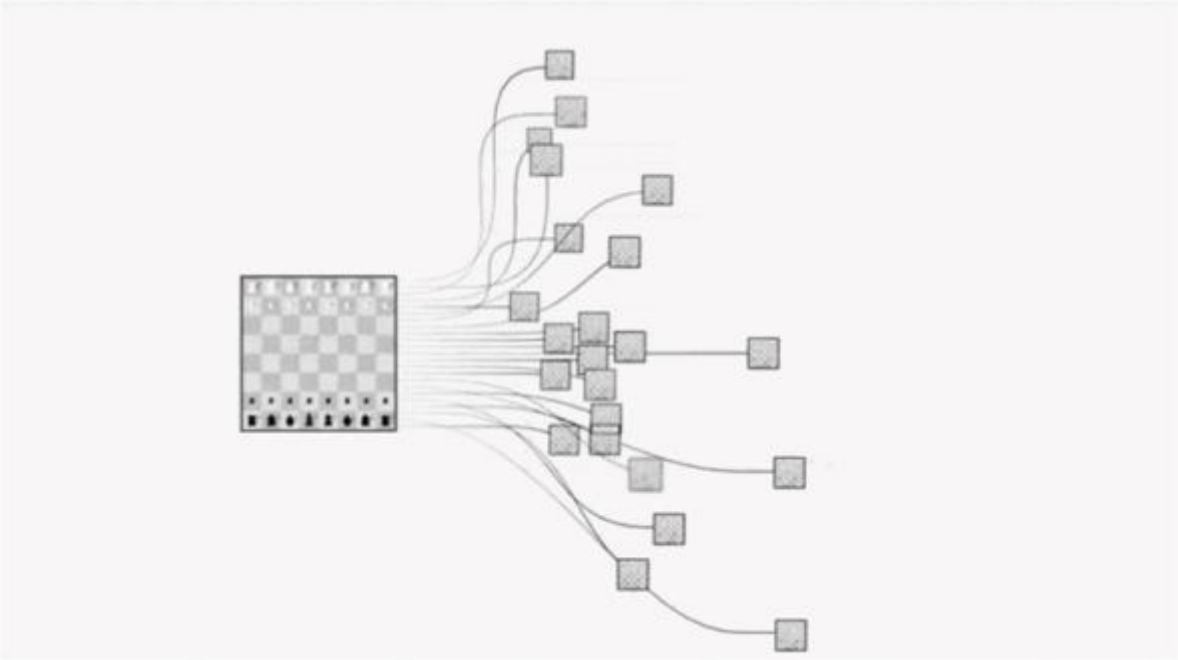
5.5 蒙特卡洛树搜索

- 蒙特卡洛树搜索全称 Monte Carlo Tree Search(MCTS), 是一种人工智能问题中做出最优决策的方法, 一般是在组合博弈中的行动规划形式. 它结合了随机模拟的一般性和树搜索的准确性.

- MCTS受到快速关注主要是由计算机围棋程序的成功以及其潜在的在众多难题上的应用所致. 超越博弈游戏本身, MCTS 理论上可以被用在以 {状态 state, 行动 action} 对定义和用模拟进行预测输出结果的任何领域.

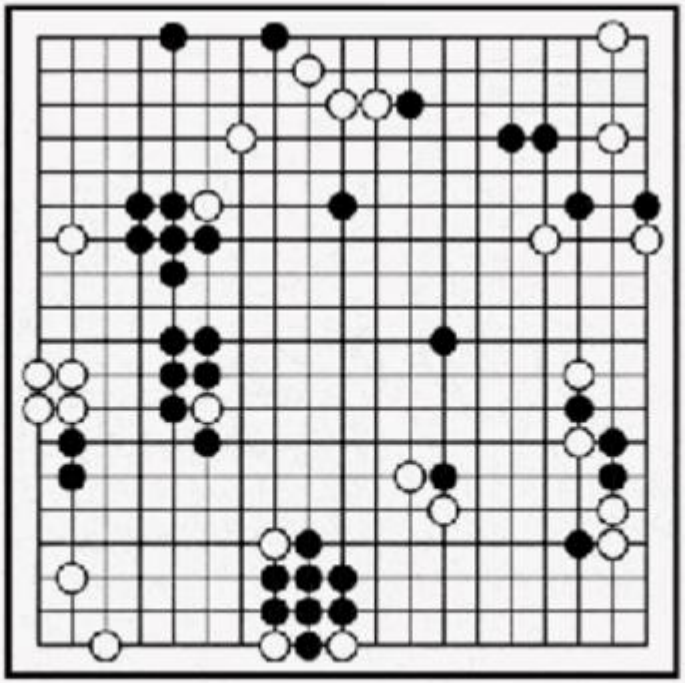
5.5 蒙特卡洛树搜索

- 人/机器下棋的基本决策过程: 根据当前棋面状态, 确定下一步动作
- 那么, 该下哪步才能保证后续赢球的概率比较大?
- 最容易想到的就是枚举之后的每一种下法, 然后计算每步赢棋的概率, 选择概率最高的就好.

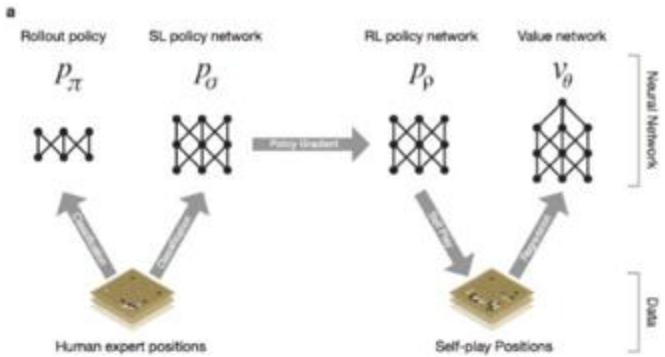


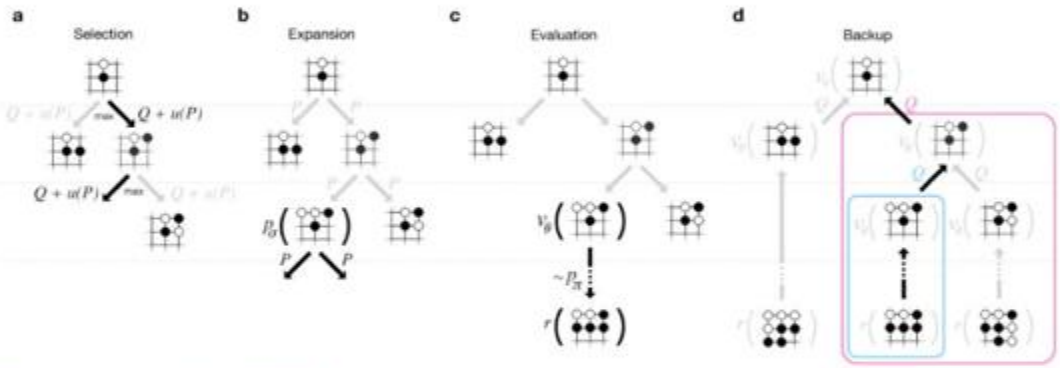
5.5 蒙特卡洛树搜索

- 中国象棋、国际象棋等都能暴力求解.
- 但是,对于围棋而言,状态空间实在太大,无法枚举.
- AlphaGo整合深度学习和蒙特卡洛树搜索解决该问题



5.5 蒙特卡洛树搜索


a
Rollout policy
SL policy network
RL policy network
Value network
p_π
p_σ
p_ρ
v_θ
Policy gradient
Natural Network
Data
Human expert positions
Self-play Positions
b
Policy network
Value network
$p_{\sigma/\rho}(a|s)$
$v_\theta(s')$
s
s'


a
Selection
b
Expansion
c
Evaluation
d
Backup
$Q + u(P)$
max
p_σ
v_θ
r
P
Q


nature
At last — a computer program that can beat a champion Go player PAGE 484
ALL SYSTEMS GO
SONGBIRDS À LA CARTE
SAFEGUARD TRANSPARENCY
WHEN GENES GOT 'SELFISH'


AlphaGo
Lee Sedol


Google

5.5 蒙特卡洛树搜索

- 基本的 MCTS 算法非常简单——根据模拟的输出结果，按照节点构造搜索树

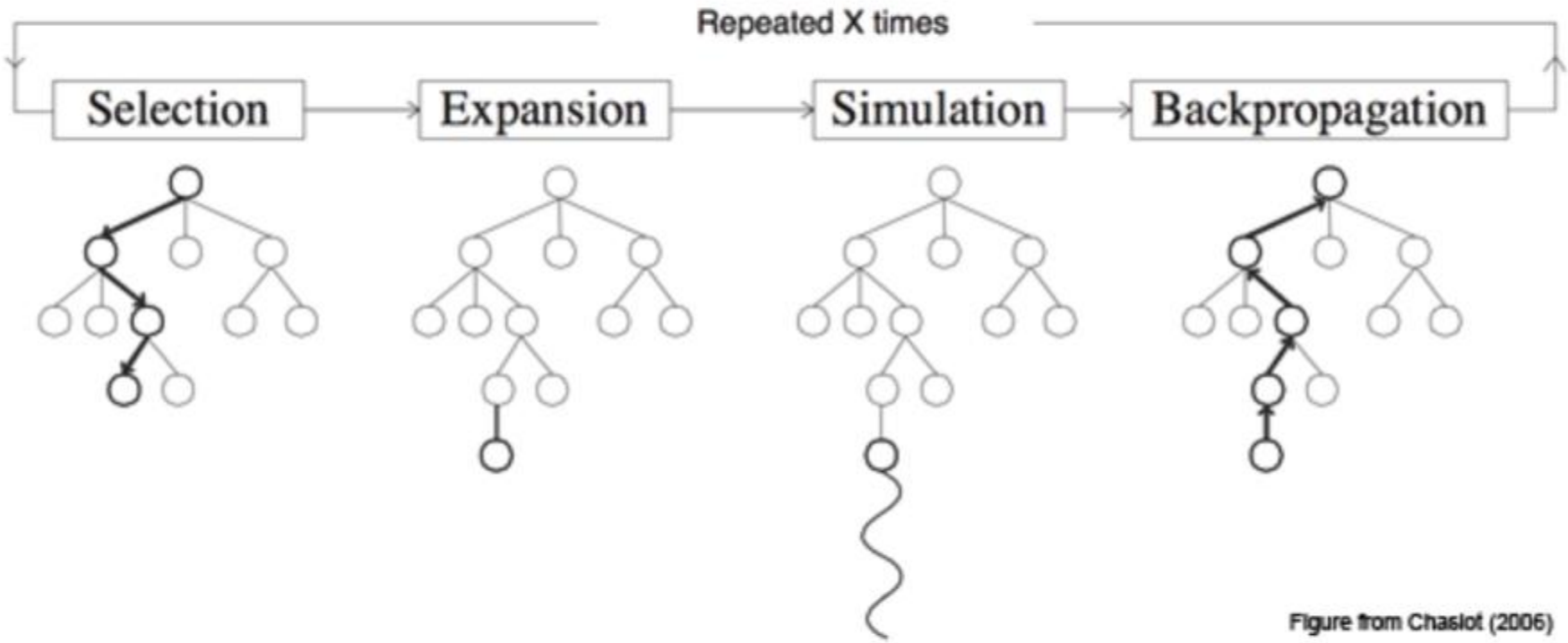


Figure from Chaslot (2006)

每个节点必须包含两个重要的信息：一个是根据模拟结果估计的值和该节点已经被访问的次数

5.5 蒙特卡洛树搜索

- 基本的蒙特卡洛树搜索(MCTS)算法

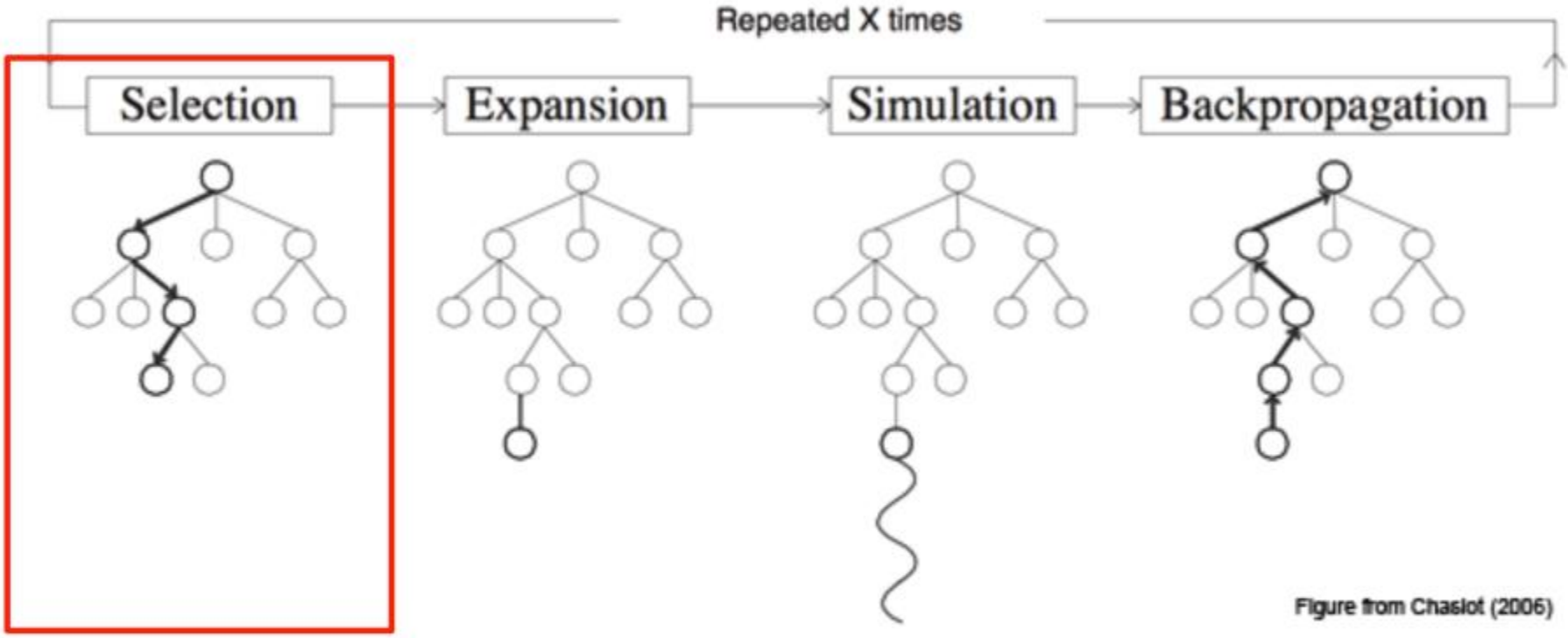
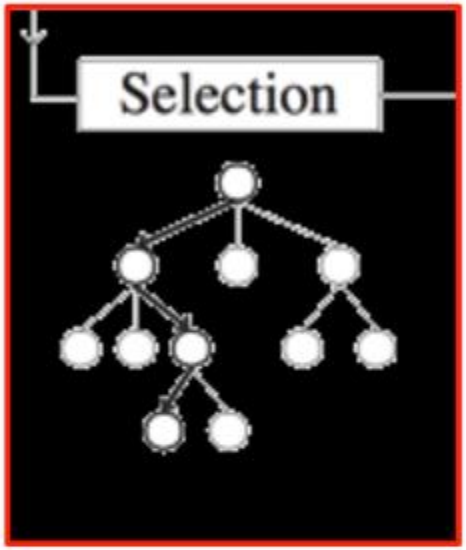


Figure from Chaslot (2006)

① 选择 Selection: 从根节点 R 开始, 递归选择最优的子节点直到达到叶子节点 L

5.5 蒙特卡洛树搜索

- 基本的蒙特卡洛树搜索(MCTS)算法



在树向下遍历时的节点选择操作是通过选择最大化某个量来实现，这类似于求解多臂赌博机问题(Multiarmed bandit problem)，其中的参与者必须选择一个赌博机(bandit)来最大化每一轮的估计的收益. 我们可以使用 Upper Confidence Bounds(UCB)公式来计算这个量：

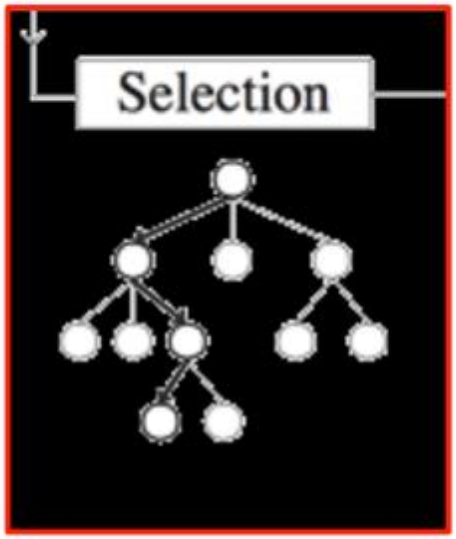
$$v_i + C \times \sqrt{\frac{\ln N}{n_i}}$$

其中，v_i是节点估计的值，n_i是节点被访问的次数，N是其父节点已经被访问的总次数，C是可调参数

48

5.5 蒙特卡洛树搜索

- 基本的蒙特卡洛树搜索(MCTS)算法



- 利用和探索的平衡:

UCB 公式对已知收益的利用和鼓励接触那些相对未曾访问的节点的探索之间进行平衡. 收益估计基于随机模拟, 所以节点必须被访问若干次来确保估计变得更加可信; MCTS 估计会在搜索的开始不大可靠, 而最终会在给定充分的时间后收敛到更加可靠的估计上, 在无限时间下能够达到最优估计.

$v_i + C \times \sqrt{\frac{\ln N}{n_i}}$ 其中, v_i 是节点估计的值, n_i 是节点被访问的次数, N 是其父节点已经被访问的总次数, C 是可调参数

5.5 蒙特卡洛树搜索

- 基本的蒙特卡洛树搜索(MCTS)算法

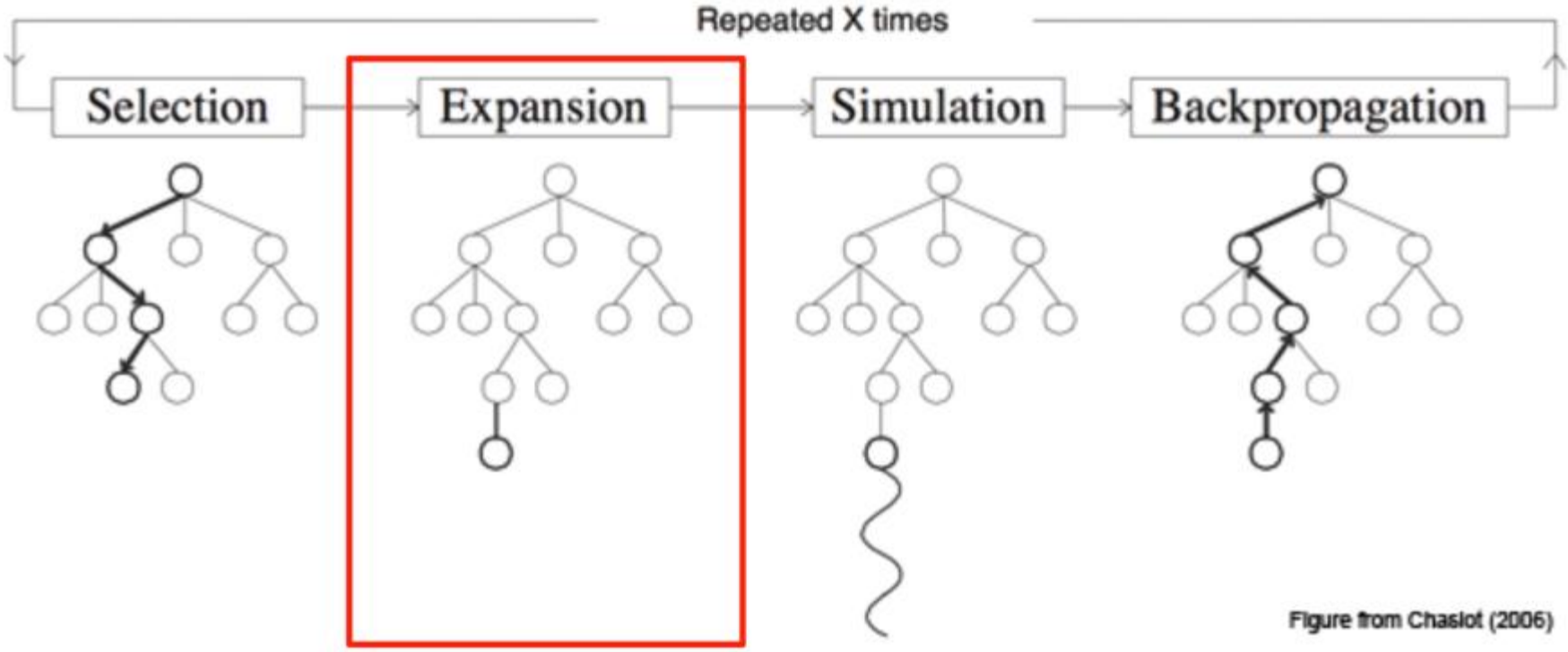


Figure from Chaslot (2006)

② 扩展 Expansion: 如果 L 不是一个终止节点(也即, 不会导致博弈游戏终止) 那么就创建一个或者更多的子节点, 选择其中一个 C

5.5 蒙特卡洛树搜索

- 基本的蒙特卡洛树搜索(MCTS)算法

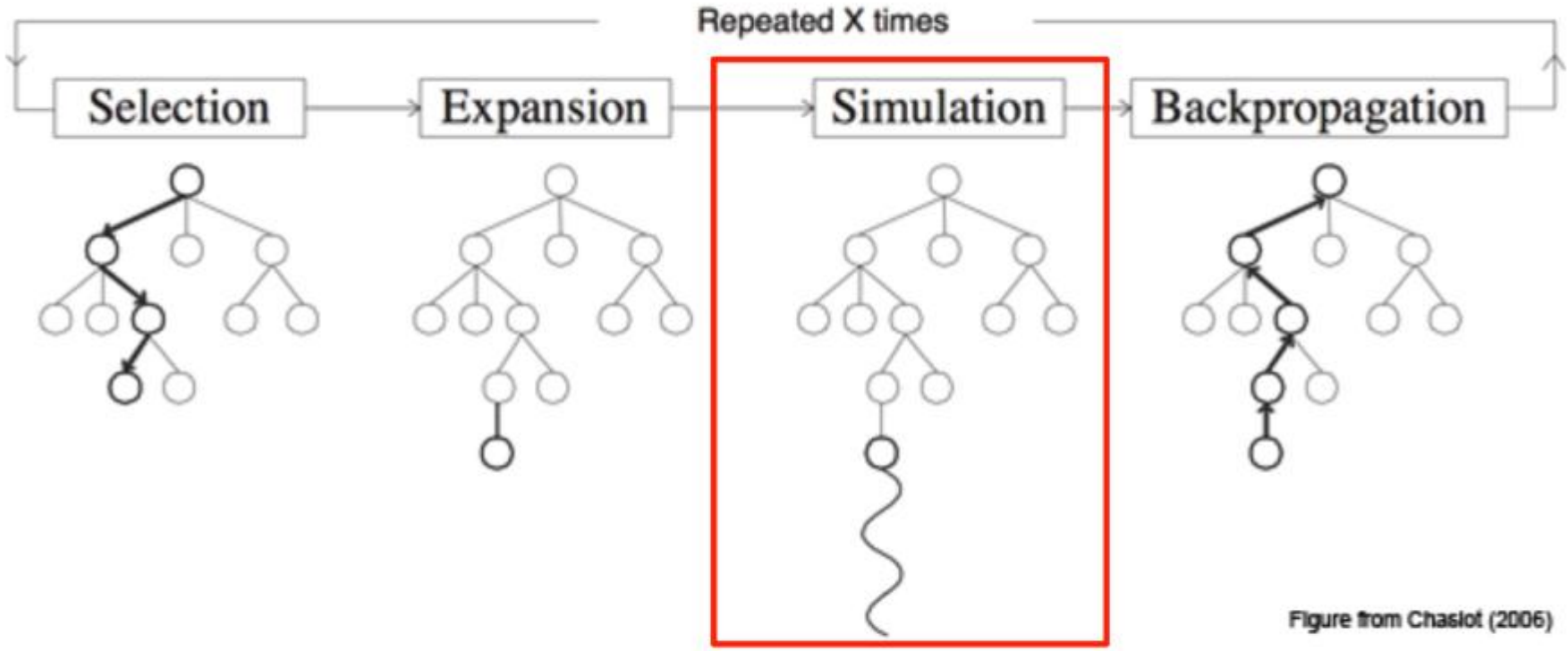


Figure from Chaslot (2006)

③ 模拟 Simulation: 从 C 开始运行一个模拟的输出，直到博弈游戏结束

5.5 蒙特卡洛树搜索

- 基本的蒙特卡洛树搜索(MCTS)算法

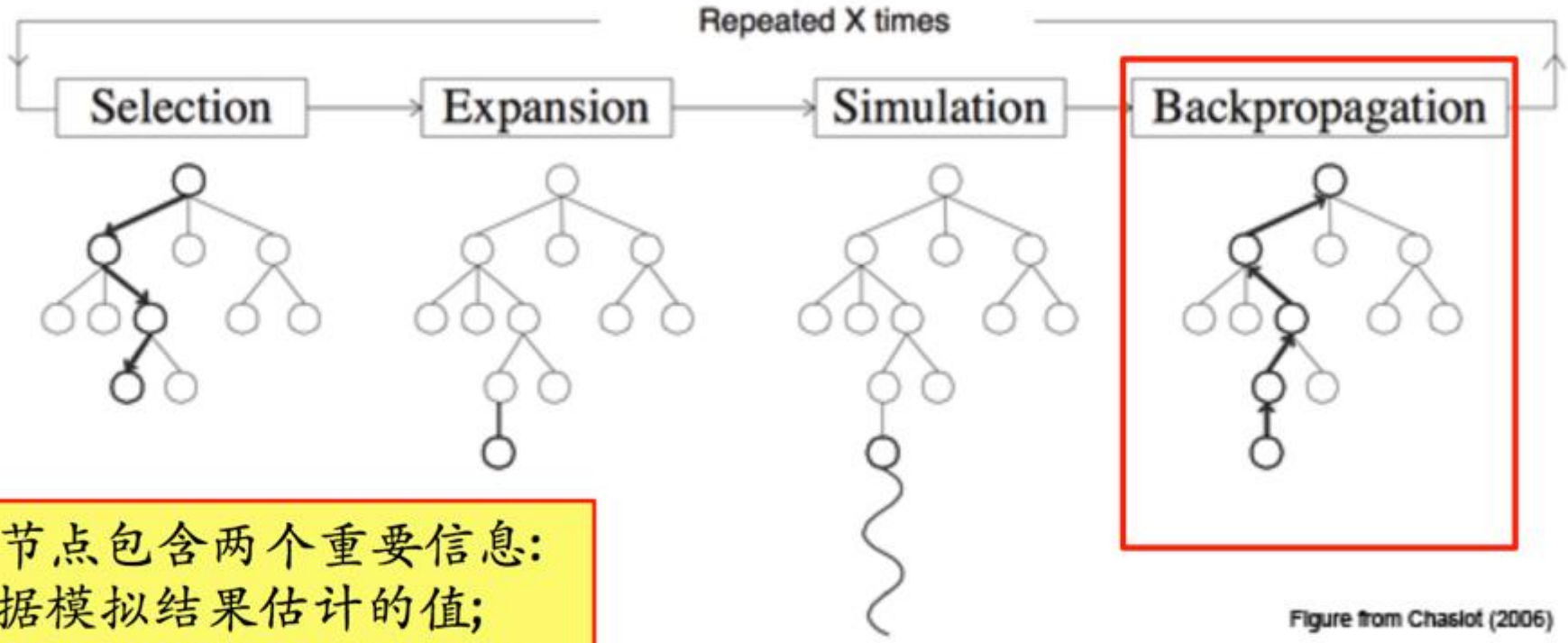


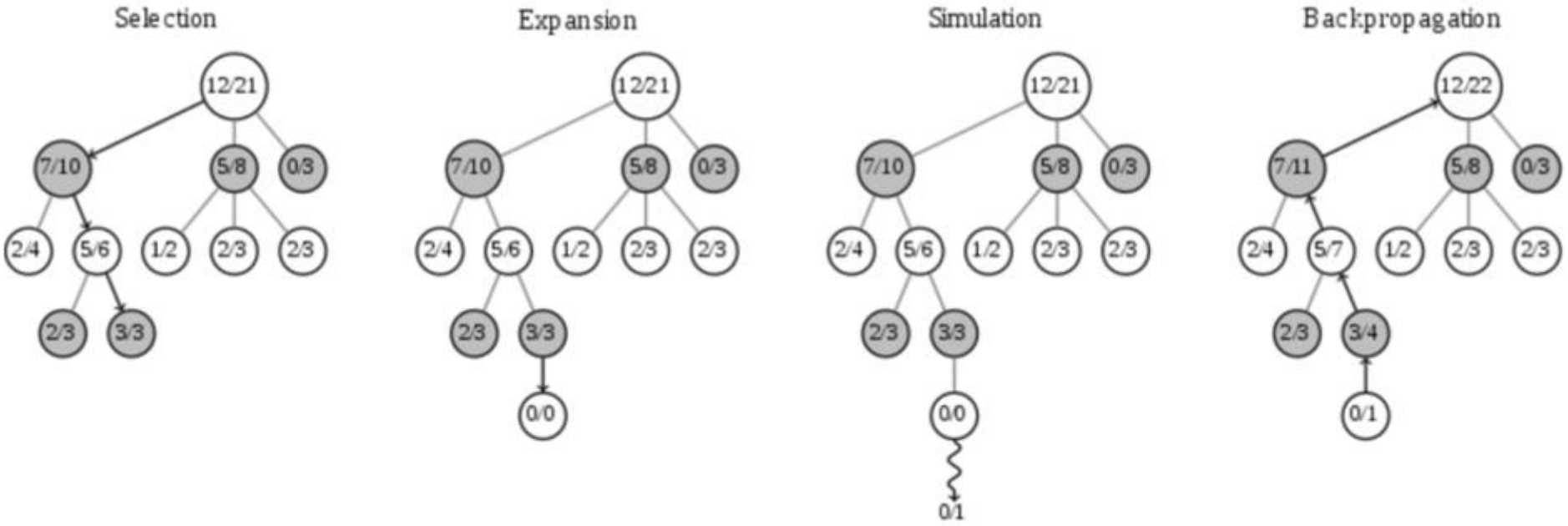
Figure from Chaslot (2006)

每个节点包含两个重要信息:
1)根据模拟结果估计的值;
2)该节点已经被访问的次数

④ **反向传播 Backpropagation**: 用模拟的结果输出更新当前行动序列, 也即更新访问过的节点上的信息.

5.5 蒙特卡洛树搜索

- 基本的蒙特卡洛树搜索(MCTS)算法



上图中每个节点代表一个局面。而 A/B 代表这个节点被访问 B 次，黑棋胜利了 A 次。例如一开始的根节点是 12/21，代表总共模拟了 21 次，黑棋胜利了 12 次。

5.5 蒙特卡洛树搜索

怎么选择节点？和从前一样：**如果轮到黑棋走，就选对于黑棋有利的；如果轮到白棋走，就选对于黑棋最不利的。但不能太贪心，不能每次都只选择"最有利的/最不利的"，因为这会意味着搜索树的广度不够，容易忽略实际更好的选择。**

因此，最简单有效的选择公式是这样的：$score = x_{child} + C \cdot \sqrt{\frac{\log(N_{parent})}{N_{child}}}$

其中 x 是节点的当前胜率估计（注意，如前所述，要考虑当前是黑棋走还是白棋走！），N 是节点的访问次数。C 是一个常数。C 越大就越偏向于广度搜索，C 越小就越偏向于深度搜索。注意对于原始的 UCT 有一个理论最优的 C 值，但由于我们的目标并不是最小化"遗憾"，因此需要根据实际情况调参。

我们看例子说明这是什么意思，就看之前的图吧。假设根节点是轮到黑棋走。那么我们首先需要在 7/10、5/8、0/3 之间选择：

1. 其中 7/10 对应的分数为 $7/10 + C \cdot \sqrt{\frac{\log(21)}{10}} \approx 0.7 + 0.55C$。
2. 其中 5/8 对应的分数为 $5/8 + C \cdot \sqrt{\frac{\log(21)}{8}} \approx 0.625 + 0.62C$。
3. 其中 0/3 对应的分数为 $0/3 + C \cdot \sqrt{\frac{\log(21)}{3}} \approx 0 + 1.00C$。
4. 可以注意到，C越大，就会越照顾访问次数相对较少的子节点。

5.5 蒙特卡洛树搜索

如果 C 比较小，我们将会选择 7/10，接着就要在 2/4 和 5/6 间选择。注意，由于现在是白棋走，需要把胜率估计倒过来：

1. 其中 2/4 对应的分数为 $(1-2/4)+C\cdot\sqrt{\frac{\log(10)}{4}}\approx 0.5+0.76C$。
2. 其中 5/6 对应的分数为 $(1-5/6)+C\cdot\sqrt{\frac{\log(10)}{6}}\approx 0.17+0.62C$。

那么我们下一步肯定应该选 2/4。所以说之前的图是错误的，因为制图的人并没有注意到要把胜率倒过来（有朋友会说是不是可以认为它的白圈代表白棋的胜率，但这样它的回溯过程就是错的）。

最后，AlphaGo 的策略网络，可以用于改进上述的分数公式，让我们更准确地选择需扩展的节点。而 AlphaGo 的价值网络，可以与快速走子策略的模拟结果相结合，得到更准确的局面评估结果。注意，如果想写出高效的程序，这里还有很多细节，因为策略网络和价值网络的运行毕竟需要时间，我们不希望等到网络给出结果再进行下一步，在等的时候可以先做其它事情，例如 AlphaGo 还有一个所谓 Tree policy，是在策略网络给出结果之前先用着。

5.5 蒙特卡洛树搜索

- 蒙特卡洛树搜索算法优点

① MCTS 不要求任何关于给定的领域策略或者具体实践知识来做出合理的决策. 这个算法可以在没有任何关于博弈游戏除基本规则外的知识的情况下进行有效工作; 这意味着一个简单的 MCTS 实现可以重用在很多的博弈游戏中, 只需要进行微小的调整.

② MCTS 执行一种非对称的树的适应搜索空间拓扑结构的增长. 这个算法会更频繁地访问更加有趣的节点, 并聚焦其搜索时间在更加相关的树的部分. 这使得 MCTS 更加适合那些有着更大的分支因子的博弈游戏, 比如说 19X19 的围棋, 这么大的组合空间会给标准的基于深度或者宽度的搜索方法带来问题, 所以 MCTS 的适应性说明它(最终)可以找到那些更加优化的行动, 并将搜索的工作聚焦在这些部分.

5.5 蒙特卡洛树搜索

- 蒙特卡洛树搜索算法优点

③ 算法可以在任何时间终止，并返回当前最有的估计. 当前构造出来的搜索树可以被丢弃或者供后续重用.

④ 算法实现非常方便.