计算机科学188：人工智能导论
2024年春季 Note 6

作者（其他所有注释）：尼基尔·夏尔马
作者（贝叶斯网络注释）：乔希·胡格和杰基·梁，由王蕾吉娜编辑
作者（逻辑注释）：亨利·朱，由考佩林编辑
学分（机器学习和逻辑注释）：部分章节改编自教材《人工智能：一种现代方法》。

最后更新时间：2023年8月26日

期望最大化算法

我们已经了解了极小化极大算法的工作原理，以及运行完整的极小化极大算法如何使我们能够针对最优对手做出最优响应。然而，极小化极大算法在其能够响应的情况上存在一些自然限制。由于极小化极大算法认为它是在应对最优对手，所以在无法保证对智能体行动做出最优响应的情况下，它往往过于悲观。这类情况包括具有内在随机性的场景，如纸牌或骰子游戏，或者对手随机或次优移动且不可预测的情况。在课程后半部分讨论马尔可夫决策过程时，我们将更详细地探讨具有内在随机性的场景。

这种随机性可以通过极小化极大算法的一种推广形式——期望最大化算法来表示。期望最大化算法在博弈树中引入了机会节点，与极小化节点考虑最坏情况不同，机会节点考虑的是平均情况。更具体地说，极小化节点只是简单地计算其孩子节点的最小效用，而机会节点计算期望效用或期望值。我们使用期望最大化算法确定节点值的规则如下：

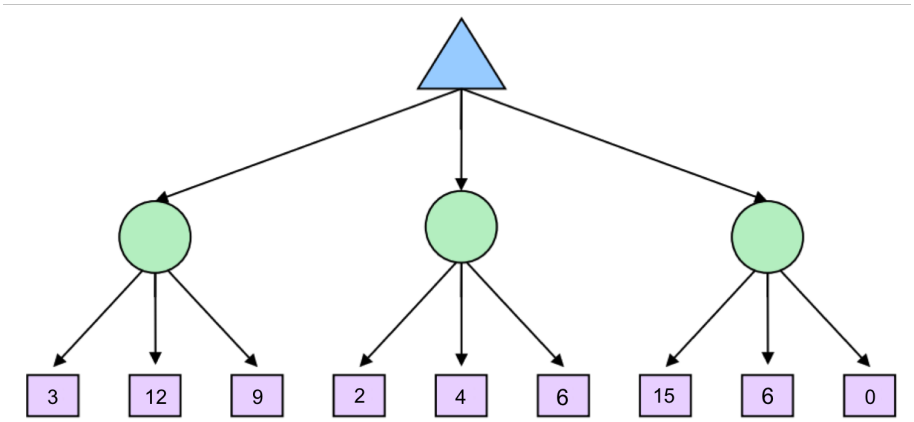
$$\begin{aligned} \forall \text{agent-controlled states}, \quad V(s) &= \max_{s' \in successors(s)} V(s') \\ \forall \text{chance states}, \quad V(s) &= \sum_{s' \in successors(s)} p(s'|s) V(s') \\ \forall \text{terminal states}, \quad V(s) &= \text{known} \end{aligned}$$

在上述公式中，$p\left(s' \mid s\right)$ 指的是给定的非确定性动作导致从状态 s 转移到 s' 的概率，或者对手选择导致从状态 s 转移到 s' 的动作的概率，这取决于所考虑的游戏及其博弈树的具体情况。从这个定义可以看出，极小极大算法只是期望极大算法的一种特殊情况。极小化节点只是概率节点，它们给自己最低值子节点分配概率1，给所有其他子节点分配概率0。一般来说，概率的选择要能恰当地反映我们试图建模的游戏状态，但我们将在未来的笔记中更详细地介绍这个过程是如何运作的。目前，可以合理地假设这些概率只是游戏固有的属性。

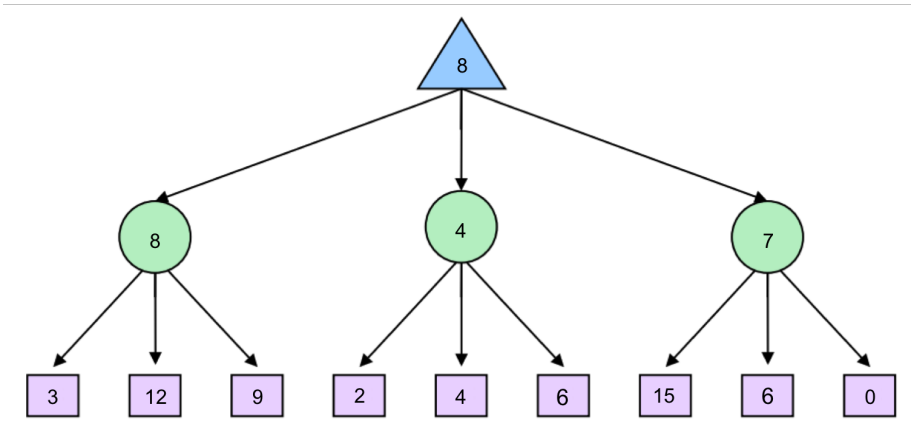
期望极大算法的伪代码与极小极大算法非常相似，只是做了一些小的调整，以考虑期望效用而不是最小效用，因为我们用概率节点取代了极小化节点：

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the agent is MAX: return max-value(state)
    if the agent is EXP: return exp-value(state)

def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v

def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

在继续之前，让我们快速浏览一个简单的示例。考虑以下期望最大化树，其中机会节点由圆形节点表示，而不是用于最大化者/最小化者的向上/向下的三角形。



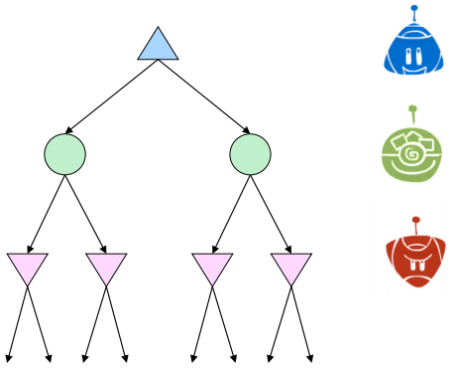
为简化起见，假设每个机会节点的所有子节点出现的概率均为 $\frac{1}{3}$ 。因此，根据我们用于确定值的期望最大化规则，我们可以看到从左到右，这3个机会节点的值分别为 $\frac{1}{3} \cdot 3 + \frac{1}{3} \cdot 12 + \frac{1}{3} \cdot 9 = 8, \frac{1}{3} \cdot 2 + \frac{1}{3} \cdot 4 + \frac{1}{3} \cdot 6 = 4$ 和 $\frac{1}{3} \cdot 15 + \frac{1}{3} \cdot 6 + \frac{1}{3} \cdot 0 = 7$ 。最大化者选择这三个值中的最大值8，从而得到如下填充完整的博弈树：



关于期望最大化算法的最后一点需要注意的是，一般来说，必须查看机会节点的所有子节点——我们不能像在极小极大算法中那样进行剪枝。与极小极大算法中计算最小值或最大值不同，单个值可能会使期望最大化算法计算出的期望值任意地偏高或偏低。然而，当我们对可能的节点值有已知的有限界时，剪枝是可行的。

混合层类型

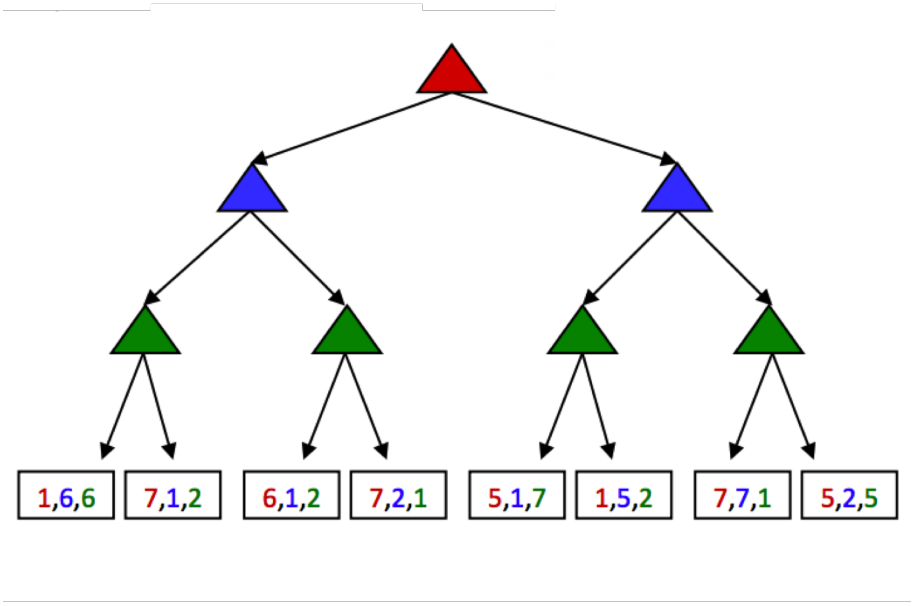
虽然极小化极大算法和期望极大算法分别要求交替出现最大化节点/最小化节点以及最大化节点/机会节点，但许多游戏仍然没有遵循这两种算法所规定的精确交替模式。即使在吃豆人游戏中，吃豆人移动后，通常会有多个幽灵轮流移动，而不是单个幽灵。我们可以通过根据需要非常灵活地在游戏树中添加层次来解决这个问题。在有四个幽灵的吃豆人游戏示例中，可以通过在第二个吃豆人/最大化节点层之前，先有一个最大化节点层，然后紧接着4个连续的幽灵/最小化节点层来实现。实际上，这样做本质上会促使所有最小化节点之间产生协作，因为它们会轮流进一步最小化最大化节点所能获得的效用。甚至可以将机会节点层与最小化节点层和最大化节点层结合起来。如果我们有一个有两个幽灵的吃豆人游戏，其中一个幽灵随机行动，另一个幽灵则最优行动，我们可以通过交替出现的最大化 - 机会 - 最小化节点组来模拟这种情况。



显而易见，节点分层存在相当大的稳健变化空间，这使得能够开发出适用于任何零和博弈的博弈树以及对抗搜索算法，这些算法是经过修改的期望最大化/极小化混合算法。

通用博弈

并非所有博弈都是零和的。实际上，在一场博弈中，不同的智能体可能有不同的任务，这些任务并不直接涉及彼此之间的严格竞争。这样的博弈可以用多智能体效用特征的树来构建。这些效用不是交替智能体试图最小化或最大化的单个值，而是表示为元组，元组中的不同值对应于不同智能体的唯一效用。然后，每个智能体在其控制的每个节点上试图最大化自己的效用，而忽略其他智能体的效用。考虑以下树：



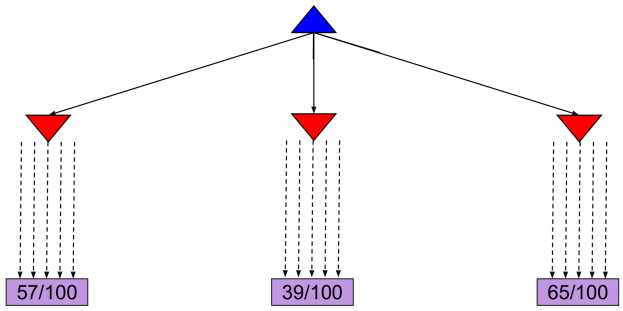
红色、绿色和蓝色节点分别对应三个不同的智能体，它们在各自层的可能选项中分别最大化红色、绿色和蓝色效用。通过这个例子最终在树的顶部得到效用元组(5,2,5)。具有多智能体效用的一般博弈是通过计算产生行为的一个主要例子，因为这样的设置会引发合作，因为在树的根部选择的效用往往会为所有参与的智能体产生合理的效用。

蒙特卡洛树搜索

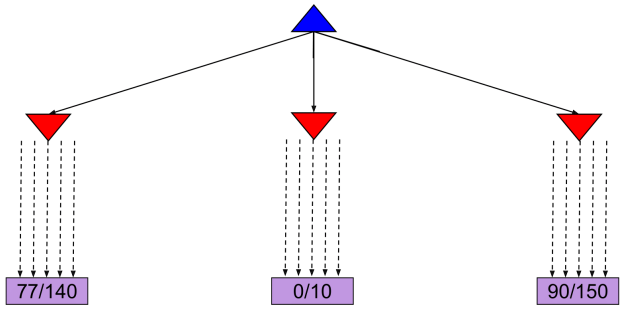
对于具有大分支因子的应用，如围棋，极小极大算法不再适用。对于此类应用，我们使用蒙特卡洛树搜索（MCTS）算法。MCTS基于两个理念：

- 通过随机模拟进行评估：从状态 s 使用一种策略（例如随机策略）多次进行游戏，并统计胜负次数。
- 选择性搜索：探索树的部分节点，不受视野限制，以改善根节点处的决策。

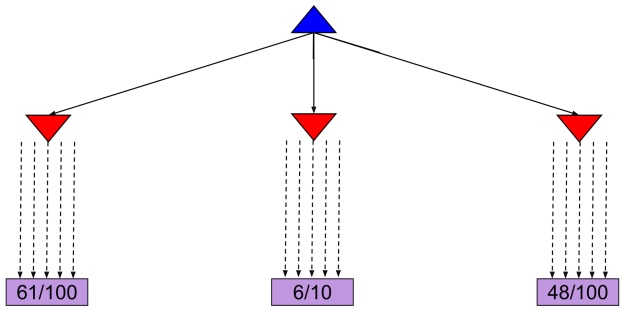
在围棋示例中，从给定状态开始，我们根据一种策略多次进行游戏直到结束。我们记录获胜的比例，它与该状态的值有很好的相关性。
考虑以下示例：



从当前状态出发，我们有三种不同的可用行动（左、中、右）。我们对每种行动都进行100次尝试，并记录每种行动的获胜百分比。经过模拟后，我们相当确信正确的行动是最佳行动。在这种情况下，我们为每个替代行动分配了相同数量的模拟。然而，经过几次模拟后，可能会清楚地发现某个行动不会带来很多胜利，因此我们可能会选择将这种计算资源用于对其他行动进行更多模拟。这种情况可以在下图中看到，我们决定将剩余的90次针对中间行动的模拟分配给左行动和右行动。



当某些行动产生相似的获胜百分比，但其中一个行动使用的模拟次数要少得多来估计该百分比时，就会出现一个有趣的情况，如下图所示。在这种情况下，使用较少模拟次数的行动的估计会有更高的方差，因此我们可能想要为该行动再分配一些模拟，以便对真正的获胜百分比更有信心。



UCB算法通过在每个节点 n 使用以下标准来捕捉“有前景”和“不确定”行动之间的这种权衡：

$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(PARENT(n))}{N(n)}}$$

其中 $N(n)$ 表示从节点 n 展开的总次数，$U(n)$ 表示玩家（父节点(n)）的总获胜次数。第一项体现了该节点的前景，而第二项体现了我们对该节点效用的不确定性。用户指定的参数 C 平衡了我们赋予这两项（“探索”和“利用”）的权重，并且取决于应用场景以及可能的任务阶段（在后期我们积累了许多试验后，可能会减少探索并增加利用）。

MCTS UCT算法在树搜索问题中使用UCB准则。更具体地说，它多次重复以下三个步骤：

1. 使用UCB准则从根节点开始逐层向下遍历树，直到到达未扩展的叶节点。

2. 向该叶节点添加一个新的子节点，然后从该子节点开始进行一次模拟，以确定该节点的获胜次数。
3. 将从子节点到根节点的获胜次数进行更新。

一旦上述三个步骤被充分重复，我们选择导致具有最高 N 的子节点的行动。请注意，由于UCT本质上会更频繁地探索更有希望的子节点，如 $N \to \infty$ 所述，UCT接近极小极大算法智能体的行为。

总结

在本笔记中，我们的关注点从考虑简单地从起点找到通往某个目标的路径的标准搜索问题，转变为考虑对抗性搜索问题，在这种情况下我们可能会有对手试图阻碍我们达成目标。我们考虑了两种主要算法：

- 极小极大算法 - 当我们的对手采取最优策略时使用，并且可以使用 $\alpha - \beta$ 剪枝进行优化。极小极大算法比期望极大算法提供更保守的行动，因此当对手情况不明时也往往能产生有利结果。
- 期望极大算法 - 当我们面对非最优对手时使用，通过对我们认为他们会采取的行动的概率分布来计算状态的期望值。

在大多数情况下，一直运行上述算法直到所考虑的博弈树的终端节点级别，计算成本过高，因此我们引入了评估函数的概念以实现早期终止。对于具有大分支因子的问题，我们描述了蒙特卡洛树搜索（MCTS）和上限置信区间（UCT）算法。此类算法易于并行化，允许使用现代硬件进行大量的模拟。
最后，我们考虑了一般博弈的问题，其中规则不一定是零和的。