基础算法

Customize Your Agent with Various Algorithms

王思图

SAST, DCST, Tsinghua University

2025-01-22

Outline

Outline

1.1 目标

本次游戏的形式化描述：给定一个环境：

$$\mathcal{E} = \Big(\mathcal{S} : \mathbb{N}_0^{d\times d}, \mathcal{A}_1 : \{0,1,2,3,4\}, \mathcal{A}_2 : \{0,1,2,3,4\}^3,$$
$$\mathcal{R} : \mathcal{S}\times\mathcal{A}_1\times\mathcal{A}_2\times\mathcal{S}\to\mathbb{R}, P : \mathcal{S}\times\mathcal{A}_1\times\mathcal{A}_2\to\mathcal{P}(\mathcal{S})\Big), \rho_0 : \mathcal{P}(\mathcal{S}))$$

双方的目标：玩家i希望找到策略π_i，使得最小累积奖励期望最大化

$$\pi_i^* = \arg\max_{\pi_i}\left[\left[\min_{\pi_{|i-1|}} R_s = \mathbb{E}\left[\sum_{t,a_{it}\sim\pi_i(a|\ s_t)} \mathcal{R}(s_t,a_{1t},a_{2t},s_{t+1})P(s_{t+1}|s_t,a_{1t},a_{2t})\right]\right]\right]$$

本次游戏的人话描述：给定一个地图，地图上有两个玩家，操作两种对象。玩家每回合同时和地图交互。地图根据当前状态和玩家的操作，返回一个分数并更新地图状态。玩家的目标是找到一个策略，使得在对方的策略下自己的累积奖励最大化。

1.1 目标

通常来讲，博弈算法都要求满足完全信息零和和马尔可夫性。我们的游戏并不满足零和性，但是我们可以通过一些技巧来转化为零和博弈。例如，我们规定玩家的分数为我方分数减对方分数，这样我们就可以将博弈转化为零和博弈。

1.2 算法设计

观察上面式子：一个自然想法是估计期望累积奖励，然后转移到估计的期望累积奖励最大的状态。这引出算法设计的两种主要思路：

- 在对局中动态估计期望奖励
- 通过固定模型估计期望奖励

这两种方法通常结合使用。例如，我们可以使用固定模型估计的期望奖励作为启发值进行搜索来动态估计期望奖励，使得搜索效率更高。

另一类方法是直接估计策略。对于离散状态空间，我们可以通过搜索的方法来估计策略，也可以手动实现一个状态机作为策略。

Outline

2.1 总览

通俗来讲就是使用 if-else，对于每类局面人为设计一个策略。SDK 通过寻路实现了一个简单的状态机方法。通常大家的代码里或多或少包含了一些人为设计的策略，通常用于特殊情况的处理。

Outline

3.1 总览

考虑期望的计算:

$$\mathbb{E}\left[\sum_{t,a_{it}\sim\pi_i(a|\ s_t)} \mathcal{R}(s_t,a_{1t},a_{2t},s_{t+1})P(s_{t+1}|s_t,a_{1t},a_{2t})\right]$$

一种直接的方法是遍历所有可能的对手策略，然后在对手策略下计算期望奖励。这引出了搜索的方法。

3.2 对抗搜索

3.2.1 思路

对抗搜索的基本想法是双方都会倾向于选择更优的策略。于是我们可以通过递归的构建游戏树，并沿着最大/最小奖励的路线来估计期望奖励。具体来说，我们从当前局面开始，展开接下来所有动作可能导致的下一个状态，重复进行该操作。这样游戏过程构成了一棵树，每条边代表了一个动作，每个节点代表一个局面，每个叶子节点代表游戏的一种结局。我们可以通过遍历这棵树递归的计算期望最大/最小奖励，随后选择该步期望奖励最大的动作。

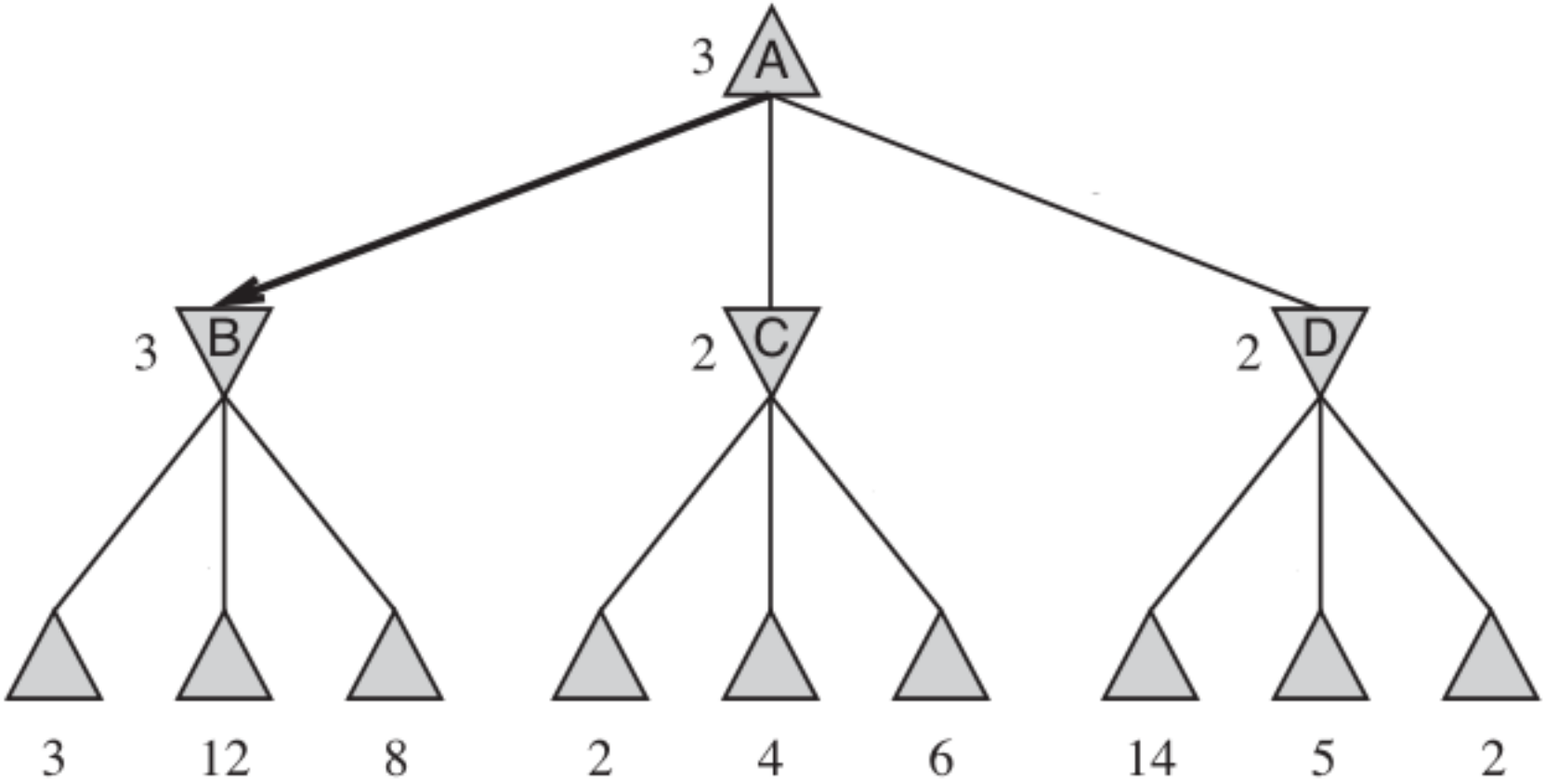
3.2 对抗搜索

3.2.2 算法

对抗搜索的算法如下：

- 从当前局面开始，递归的构建游戏树
- 从叶子节点开始，递归的计算最大/最小奖励
- 从根节点开始，选择期望奖励最大的动作

3.2 对抗搜索



3.3 改进：蒙特卡洛估值

3.3.1 思路

对抗搜索的一个问题是搜索树的规模可能会很大，导致搜索效率低下。一种改进的方法是蒙特卡洛估值。具体来说，我们可以通过随机模拟游戏的方式来估计期望奖励。这样我们可以通过多次模拟来估计期望奖励，从而减少搜索树的规模。

3.3.2 算法

蒙特卡洛估值的算法如下：

- 从当前局面开始，扩展一个节点
- 从该节点出发随机模拟游戏，估计期望奖励
- 重复多次，估计期望奖励
- 选择期望奖励最大的动作

3.4 改进：剪枝

3.4.1 思路

另一种改进的方法是剪枝。具体来说，我们可以通过一些启发式的方法来减少搜索树的规模。这样我们可以通过减少搜索树的规模来提高搜索效率。

3.4.2 算法

剪枝的算法如下：

- 从当前局面开始，递归的构建游戏树；如果某个节点的期望奖励已经小于当前最大/最小奖励，可以剪枝
- 从叶子节点开始，递归的计算最大/最小奖励
- 从根节点开始，选择期望奖励最大的动作

3.5 改进：终局特判

3.5.1 思路

当临近终局时，可能所有的状态估值都很大，但必胜的状态可能只有一个。这时我们可以通过特判的方式来提高搜索效率。

3.5.2 算法

终局特判的算法如下：人为给定一个算法，当遇到特定的局面时，直接采取特定的动作

Outline

4.1 总览

另一种思路是通过固定模型来估计期望奖励。具体来说，我们可以通过寻找一个模型来估计某个局面下执行某个动作的最终期望奖励，或估计到达某个局面的价值。这引出了基于模型的方法。

4.2 手动设计

4.2.1 思路

一种直接的方法是直接手动设计函数。具体来说，我们可以通过设计一个函数来估计某个局面下执行某个动作的最终期望奖励的相对大小（因为在基于价值的方法中，只有期望价值的相对大小是有意义的）。这种函数被称为状态-动作价值函数。

4.2 手动设计

4.2.2 算法

例如，我们可以通过以下函数来估计 Rollman 的状态-动作价值:

$$V(s, a_1, a_2) = \sum_{i, \text{ take action a1 and a2}} \text{dist}(\text{ghost}_i, \text{rollman})$$

或者，我们可以通过以下函数来估计 Rollman 的状态价值:

$$V(s) = \sum_i \text{dist}(\text{ghost}_i, \text{rollman})$$

4.3 强化学习

4.3.1 思路

另一种方法是通过强化学习来估计期望奖励。具体来说，我们可以通过设计一个模型来估计某个局面下执行某个动作的最终期望奖励的相对大小。这种模型被称为 Q 函数。SDK 实现了最简单的 Q-Learning 算法。

4.3.2 算法

- Minimax-Q
- Nash Q-Learning
- Friend-or-Foe Q-Learning
- WoLF Policy Hill-Climbing

Outline

5.1 参考资料

https://docs.net9.org/ai-ml/adversarial-search/ 对抗搜索

https://agent-guide.net9.org/instruction/pacman/ 开发引导