李堃秀 2022012110 Likx22@mails.tsinghua.edu.cn

一、简答题

1、 相比树搜索，图搜索作出了什么改进？

图搜索采用了用空间换时间的策略，通过引入已访问状态集合，记录已经搜索过的状态，避免了树搜索中可能存在的重复搜索，使得搜索过程更为高效。

2、 相比宽度优先搜索（BFS），深度优先搜索（DFS）和一致代价搜索（UCS）分别有什么优劣？

BFS：在路径代价相等的情况下，能够确保找到最短路径，但其空间复杂度较高

DFS：空间复杂度低，但不能够确保找到最短路径。在处理 n 皇后等最短路径长度等于搜索空间深度的问题中往往能够快速找到最优解

UCS：能够确保找到最短路径，适合解决带权路径问题，但需要额外空间来存储节点和代价

3、 在约束满足问题（Constraint Satisfaction Problems）中，搜索算法为什么每次要选择约束尽量多的变量（most constrained variable）和约束尽量少的值（least constrained value）？

有更多约束的变量，通常也会有更多产生冲突的可能，并会限制其他变量的选择，因此首先为该变量赋值，可以更早地发现冲突，尽早进行剪枝，避免无效搜索。

而选择约束更少的值，能够确保更大的其他变量的选择空间，从而在搜索过程中能有更多的选择余地，减少发生早期冲突的风险，使得搜索过程尽可能平滑，提高解的效率。

4、 随机集束搜索（Stochastic Beam Search）为什么要引入随机性？它是否能保证找到最优解？

引入随机性是为了在搜索过程中避免陷入局部最优解，从而增加找到全局最优解的可能性。

它不能保证找到最优解。

5、 Min-Max 搜索中，若被 Alpha-Beta 剪枝的节点状态值的符号反转，根节点决策有可能发生改变吗？为什么？

根节点决策有可能发生改变。若一个被 Alpha-Beta 剪枝的节点节点状态值的符号反转，意味着它的角色可能发生改变（如从 Max 变 Min 或反之）。这种变化可能导致剪枝策略变化，进而影响父节点的决策，最终也可能导致根节点决策发生改变。

二、A*算法的性质

在 A* 算法中，记搜索起点为 S，终点为 T。对于节点 n，记 g(n)表示当前搜索过程中从根节点到 n 的总代价，h*(n)为 n 到目标节点 T 的最小代价，h(n)为 n 节点的启发（heuristic）函数。在以下问题中，设 h(T)=0，且所有代价均非负。假设 A*算法每次选择边缘集里 f(n)=g(n)+h(n)最小的节点。若采用树搜索，试证明：

1. 若对任意节点 n 均满足 h(n)≤h*(n)+C1，其中 C1≥0 为常数，则 g(T)≤h*(S)+C1；

2. 若对任意节点 n 均满足 h(n)≤C2·h*(n)，其中 C2≥1 为常数，则 g(T)≤C2·h*(S)。

提示：算法未必会找到 S 到 T 的最优路径。讨论当 T 即将取出边缘集时，最优路径上位于边缘集中的点 n*。事实上，n* 满足 h*(S)=g(n*)+h*(n*)。

问题一：

设 n^* 为最优路径上的某点（即满足 $h^*(S) = g(n^*) + h^*(n^*)$ ）

令 n^* 为目标结点 T 被弹出时，最优路径最末的结点（即满足 $f(T) \leq f(n^*)$ ）

根据题设条件，对于结点 n^* 有：

$h(n^*) \leq h^*(n^*) + C_1$

以上不等式可以重写为：

$g(n^*) + h(n^*) \leq g(n^*) + h^*(n^*) + C_1$

由于 $h^*(S) = g(n^*) + h^*(n^*)$ 及 $f(T) = g(T) + h(T) = g(T)$ ，可得：

$g(T) \leq f(n^*) = g(n^*) + h(n^*) \leq h^*(S) + C_1$

证毕

问题二：

设 n^* 为最优路径上的某点（即满足$h^*(S) = g(n^*) + h^*(n^*)$）

类似问题一，有：

$g(T) = f(T) \leq f(n^*) = g(n^*) + h(n^*) \leq g(n^*) + C_2 h^*(n^*)$

对于 $n^* = S$ 的情况，即有 $g(T) \leq C_2 h^*(S)$

证毕

三、蒙特卡洛树搜索

1、井字棋（3×3 的三子棋）中，假设对局双方都绝对理性，先手有必胜策略吗？后手能保证不输吗？（直接回答）

先手没有必胜策略，后手能保证不输。

2、补充 ./mcts/uct_mcts.py 中标注 ./code 的内容，实现 MCTS 算法。

见 .code

3、适当修改 pit.py，在设定 C = 1.0, n_rollout = 7, n_search = 64 的情况下，分别汇报井字棋游戏中，MCTS 算法对弈 Random 策略、MCTS 算法对弈 AlphaBeta 算法和 MCTS 算法对弈 MCTS，在先手和后手情况下（共 3×2=6 组）的胜率和不输率。

Matchup	Player 1 Win	Player 2 Win	Draw	Player 1 No Lose (%)	Player 2 No Lose (%)
MCTS vs Random (P1)	77.50%	13.80%	8.70%	86.20%	22.50%
MCTS vs Random (P2)	19.10%	73.60%	7.30%	26.40%	80.90%
MCTS vs AlphaBeta (P1)	0.00%	89.20%	10.80%	10.80%	100.00%
MCTS vs AlphaBeta (P2)	99.90%	0.00%	0.10%	100.00%	0.10%
MCTS vs MCTS (P1)	41.40%	56.10%	2.50%	43.90%	58.60%
MCTS vs MCTS (P2)	42.90%	55.00%	2.10%	45.00%	57.10%

4、取消 ./players/uct_player.py 第 33 行处的注释，对比在 C =0.1, C =5.0 时 MCTS 输出的策略有何不同，分析 C 的取值是如何影响 MCTS 输出的策略的。

C=0.1 时输出：

[0.13541667 0.11284722 0.18576389 0.11284722 0.11458333 0.11284722
 0.11284722 0.11284722 0.   ]
[0.01492537 0.01492537 0.   0.01492537 0.01492537 0.01492537
 0.   0.92537313 0.   ]

[0. 0.01515152 0. 0.01515152 0.95454545 0.01515152
0. 0. 0.]
[0. 0.984375 0. 0.015625 0. 0. 0. 0.
0.]
C=5.0 时输出：
[0.12326389 0. 0.12326389 0.12847222 0.11805556 0.12326389
0.12847222 0.12847222 0.12673611]
[0. 0. 0.171875 0. 0.21875 0.171875 0.140625 0.140625
0.15625]
[0. 0. 0.21212121 0. 0. 0.37878788
0. 0.21212121 0.1969697]

在 C=0.1 时，从第一组策略开始，不同动作的权重方差均相比于 C=5.0 时大。这表明 MCTS 算法在较小的 C 值（0.1）下更加倾向于选择已知的高回报路径，而避免过多探索未知路径。这有利于快速收缩，但可能会错过一部分潜在的优势策略。

在 C=5.0 时，前两组策略的动作权重方差较小。这表明 MCTS 算法更倾向于探索所有的潜在可能，这可能降低时间效率（更慢收缩），但错过优策略的可能性更小。

5、 选取一组合适的 n_rollout、n_search 参数的取值，在 7×7 的围棋游戏中测试 MCTS 与 Random 策略对局先手和后手的胜率和不输率，并回答这两个参数对 MCTS 的搜索的速度和质量有何影响。

n_rollout	n_search	Player 1 (MCTS) Win	Player 2 Win	Draw	Player 1 No Lose (%)	Player 2 No Lose (%)
7	**64**	85	14	1	86.00%	15.00%
4	**64**	82	16	2	84.00%	18.00%
7	**32**	65	33	2	67.00%	35.00%
n_rollout	**n_search**	**Player 1 (MCTS) Win**	**Player 2 Win**	**Draw**	**Player 1 No Lose (%)**	**Player 2 No Lose (%)**
7	**64**	24	76	0	24.00%	76.00%
4	**64**	26	73	1	27.00%	74.00%
7	**32**	43	57	0	43.00%	57.00%

n_rollout 是在每次搜索过程中，对于每个叶节点执行的模拟（rollout）次数。增大 n_rollout 可以减少单次模拟带来的随机噪声，提高准确率，但会降低时间效率。

n_search 是指每次执行 search 时，MCTS 执行的完整搜索次数。增大 n_search 意味着更多的扩展、回传操作，能够探索到更多可能的潜在策略，有利于找到最优策略，但会降低时间效率。

四、理解 AlphaZero

在标准的围棋棋盘大小（19×19)下，由于状态空间极大，本次作业中使用的基于 UCB（Up per Confidence Bound）公式的 MCTS 无法在合理时间内得到高质量的策略。为了解决这个问题， AlphaZero 中，神经网络被嵌入 MCTS 以提高搜索的效率和质量。由于我们课程还未讲到神经网络，本次作业中的 MCTS 没有涉及到这部分内容。为了更好的了解大作业的整体结构，确保在后续作业中可以零成本复用本次作业的代码，请你

简单阅读 AlphaGoZero 的论文 2（其 MCTS 部分和 AlphaZero 的实现基本一致）和补充材料 3，其中与本次作业内容较为相关的部分是 AlphaGoZero 论文的“Reinforcement learning in AlphaGo Zero”章节和补充材料第 14 至 15 页的内容。请在完成阅读后在实验报告中回答以下问题：

1、 为什么 AlphaZero 使用了 MCTS 而不是 AlphaBeta 作为搜索主干？

我认为可能原因如下：

(1) 围棋的搜索空间过大：由于 AlphaBeta 算法依赖于穷尽性搜索，在这种情况下，剪枝效率较低，需要大量的存储和计算（在之前的作业中，模拟 5*4 的棋盘，AlphaBeta 在时间、空间的开销表现就已经难以接受）。而 MCTS 是基于蒙特卡洛模拟，可以更有效地处理大规模的状态空间。

(2) MCTS 算法不依赖于特定的评估函数：AlphaBeta 需要依赖启发式评估函数来估计局面优劣，这种评估函数需要人工设计。然而，在围棋的复杂策略中，人为设计评估函数过于困难，不如使用 MCTS 算法进行模拟。

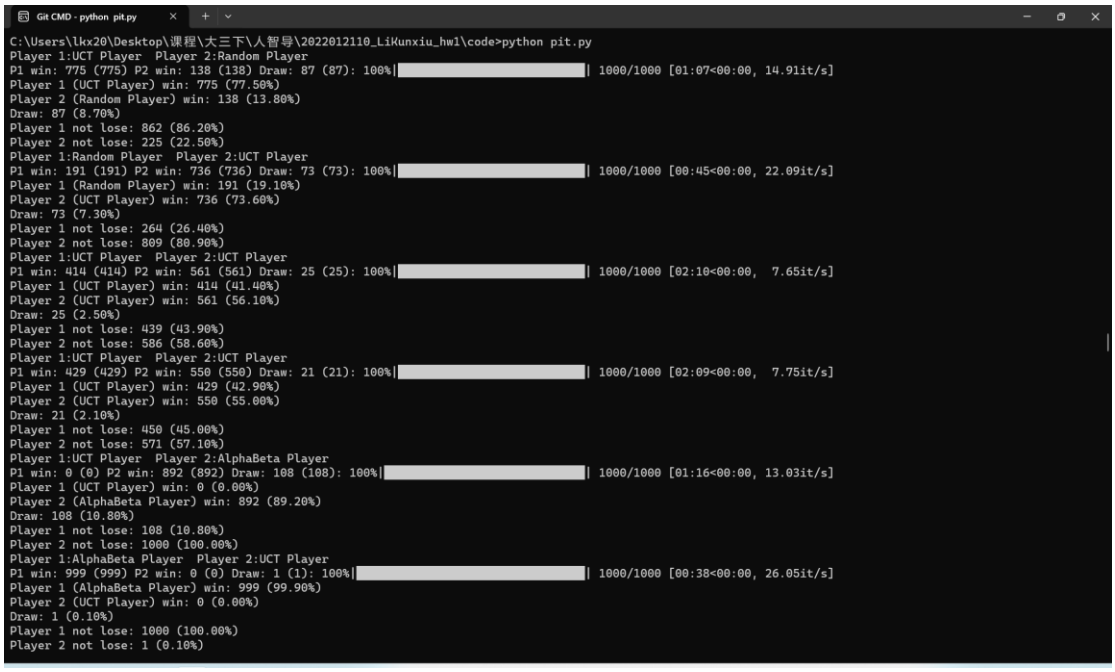
2、 AlphaZero 中使用的基于 PUCT（Predictor + UCB Applied to Trees）的 MCTS 与本次作业中基于 UCT（Upper Confidence Bounds Applied to Trees）的 MCTS 有哪些区别？

在标准的 UCT 中，树的选择策略基于 UCB 公式，选择操作主要考虑节点的访问次数和平均回报。在 AlphaZero 中，PUCT 对 UCT 进行了改进。PUCT 将神经网络的输出（策略预测和价值预测）结合进来，替代了传统 UCT 中仅依赖历史数据和回报的部分。

声明：在 codeing 过程中部分算法实现思路（伪代码和注释）借助了大模型生成，但结合依赖文件进行补全及 debug 等均为自主完成。

原始数据：

井字棋游戏中，MCTS 算法对弈 Random 策略、MCTS 算法对弈 AlphaBeta 算法和 MCTS 算法对弈 MCTS，在先手和后手情况下（共 3×2=6 组）：



n_rollout、n_search=（7，64）

```
(myenv) C:\Users\lkx20\Desktop\课程\大三下\人智导\AI2025_HW1\hw1\code>python -m pit
Player 1:UCT Player  Player 2:Random Player
P1 win: 85 (85) P2 win: 14 (14) Draw: 1 (1): 100%|| 100/100 [19:57<00:00, 11.97s/it]
Player 1 (UCT Player) win: 85 (85.00%)
Player 2 (Random Player) win: 14 (14.00%)
Draw: 1 (1.00%)
Player 1 not lose: 86 (86.00%)
Player 2 not lose: 15 (15.00%)
Player 1:Random Player  Player 2:UCT Player
P1 win: 24 (24) P2 win: 76 (76) Draw: 0 (0): 100%|| 100/100 [22:28<00:00, 13.48s/it]
Player 1 (Random Player) win: 24 (24.00%)
Player 2 (UCT Player) win: 76 (76.00%)
Draw: 0 (0.00%)
Player 1 not lose: 24 (24.00%)
Player 2 not lose: 76 (76.00%)
```

n_rollout、n_search=（4，64）

```
(myenv) C:\Users\lkx20\Desktop\课程\大三下\人智导\AI2025_HW1\hw1\code>python -m pit
Player 1:UCT Player  Player 2:Random Player
P1 win: 82 (82) P2 win: 16 (16) Draw: 2 (2): 100%|| 100/100 [09:40<00:00,  5.80s/it]
Player 1 (UCT Player) win: 82 (82.00%)
Player 2 (Random Player) win: 16 (16.00%)
Draw: 2 (2.00%)
Player 1 not lose: 84 (84.00%)
Player 2 not lose: 18 (18.00%)
Player 1:Random Player  Player 2:UCT Player
P1 win: 26 (26) P2 win: 73 (73) Draw: 1 (1): 100%|| 100/100 [11:33<00:00,  6.94s/it]
Player 1 (Random Player) win: 26 (26.00%)
Player 2 (UCT Player) win: 73 (73.00%)
Draw: 1 (1.00%)
Player 1 not lose: 27 (27.00%)
Player 2 not lose: 74 (74.00%)
```

n_rollout、n_search=（7，32）

```
(myenv) C:\Users\lkx20\Desktop\课程\大三下\人智导\AI2025_HW1\hw1\code>python -m pit
Player 1:UCT Player  Player 2:Random Player
P1 win: 65 (65) P2 win: 33 (33) Draw: 2 (2): 100%|| 100/100 [09:54<00:00,  5.94s/it]
Player 1 (UCT Player) win: 65 (65.00%)
Player 2 (Random Player) win: 33 (33.00%)
Draw: 2 (2.00%)
Player 1 not lose: 67 (67.00%)
Player 2 not lose: 35 (35.00%)
Player 1:Random Player  Player 2:UCT Player
P1 win: 43 (43) P2 win: 57 (57) Draw: 0 (0): 100%|| 100/100 [10:07<00:00,  6.07s/it]
Player 1 (Random Player) win: 43 (43.00%)
Player 2 (UCT Player) win: 57 (57.00%)
Draw: 0 (0.00%)
Player 1 not lose: 43 (43.00%)
Player 2 not lose: 57 (57.00%)
```