Comp90054 Report

AI Planning

for

Autonomy

Oral Presentation
https://youtu.be/7iSb7y4EEGQ

Description

1.Alpha zero(UCTS + Reinforcement learning(SARSA) + Greedy search + H fuc)

Procedure

The MCTS selects a not fully expanded node using UCB and then expands this node. After the expansion, the greedy simulator would sample whether this state leads to a winning state, and this result will be backpropagated to the root. The action with the highest q-value will be chosen eventually. Unlike the real Alphazero, our reward and policy score is calculated by heuristic and q-function respectively(as described below).

pros :1. The alpha-zero response was very quick to give a local maximum. 2. Yinsh has an almost infinite game state and most of which are meaningless. Using MCTS can help the agents focus on the potential useful states only.

cons 1. The simulation takes approximately 0.15s to find a reward. It is impossible to sample enough states to get a reasonable move. 2. The action might not be globally optimal. 3. The pre-trained on Qvalue takes a day-long to converge.

problem solved: 1. The reinforcement learning of the q-value is not gonna converge in such a short game, alternatively, the pre-trained Q-value is used. 2. Random simulators always give a tie prediction; therefore, the greedy simulator(driven by the pre-trained q-value) is used instead.

Potential improvement: 1. Training a real value network and real policy network. Make the Reinforcement learning happen during the tree search

2.Minimax Algorithm(DFS + Heuristic+game theory)

Procedure: The max player will generate all the states first and list out all the possible moves by MIN against each state. Then Max will run a DFS to find a movement where the maximizing the reward is minimized by Min.

strengths:1. It provides the optimal move assuming the opponent plays optimally.
2. The a-b pruning allows many meaningless states to be pruned.

weakness1. The branching factor of Minimax is high(see experiments). This might be timely and cause only a few plies can be looked at. 2. It highly relies on the heuristic.

Challenges: The deep copy is time to generate the game tree. Therefore, the Yinsh model is refactored. All the changes are directly applied to the board and the board can use actions recorded to revert to the original state. This reduces the time complexity to O(1).

Possible improvement 1. When the states are generated, a quick scoring function(don't have to be precise) can be used to score the state. The successors might be shorted based on the score. This can help ab-pruning prune more branches. 2. Choosing a better heuristic.

Not an agent but want to talk a bit more1: Heuristic

The heuristic is calculated as the 10000* [number of rings won] and 10* [number of markers] concerning the player. It will return inf if the play wins and -inf when the player loses.

Pros: 1. It is safe, it always gets notified when the game is over by assigning a large number to it. 2. It's quick. 3. It always gives a score even for unseen states.

Cons:1. The weight of features might not initialize optimally. 2. Some relevant features might not be included. 3. The engagement of the opponent introduces some noise.4.As (1,2,3), it is neither admissible and not optimal.

Potential improvements: Running the heuristic with different configurations on the same model and choosing a better setting based on observation.

Not an agent but want to talk a bit more2: RL

The reinforcement linear approximated learning(Sarsa) was run over minimax till it converges(using gradient descent, in a range between +/-100). [Markers controlled by start and end position], [newly formed lines] and [agent scores] were chosen as the features to be trained on while the state is determined by its heuristic score.

pros: Efficient, close to reality, with the features extracted from each state-action pair.
cons**:** converge too slow, have to be pre-trained.
potential improvements: Might abstract more relevant features with critical analysis.

Decision making

In our project, the following techniques are used: 1. Linear Q value approximation. 2. Pre-trained Q value function. 3. Sarsa reinforcement learning. 4. MCTS. 5. Heuristic. 6. Minimax. The justifications of why those techniques are being used can be seen below.

Linear Q value approximation

The state of the Yinsh game is almost infinite(5^85); therefore, it is impossible to make a Q-table for Yinsh. As an alternative, a list of features is abstract from the game state and their weight is approximated by the gradient descent. By doing this, the value of an unseen action can be connected to a set of circumstances seen and its score can be estimated.

Pre-trained linear Q-function

The game of Yinsh is quite short, ending in 62 moves. If we initialize the weight of each feature randomly, it will not have a chance to converge in a game-long period and it will neither contribute to the models' decision making.

Sarsa

Since the q function is trained on our models, we can't promise that the action considered as the best by our model is also being considered as the best for others. For that reason, Q-learning is not suitable. Additionally, training by Sarsa also might make our agent choose a less risky action(just like the cliff example in the lecture slides).

MCTS

Because there are a large number of potential actions to be taken at each state, minimax can become frighteningly difficult to choose an action in such a limited time for using deep plies. In contrast, MCTS can give a local maximum action even if a single epoch is run over. This is because MCTS is a universally applicable evaluation function that uses random sampling for deterministic problems to find the path of the highest potential outcome. For a similar reason, MCTS can see longer feature rewards than minimax.

Heuristic:

The Yinsh game has infinite states. It is not reasonable to manually assert each state and assign a score to each of them. Furthermore, the scenarios of the game board are complex and we want to reduce the information the board contains to a simple and manageable set of choices. This can be achieved by writing a heuristic function to abstract features from the board given.

Minimax

Since Yinsh is a zero-sum game with two agents, using minimax can consider the actions of the opponent. Giving an example, unlike the BFS, minimax can prevent the opponent from forming a sequence. This case can be illustrated clearly in tables 1 and 2. Moreover, minimax allows using a-b pruning to skip some moves which are worse than the previous one. This is because these moves won't produce a better choice and won't influence the final decision. Eventually, this may make its thinking time shorter by 1s.

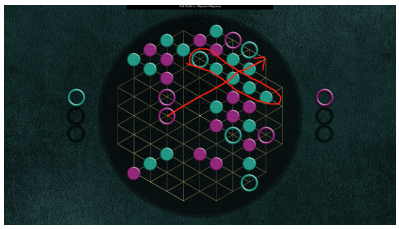


Table1 green is about to remove a ring

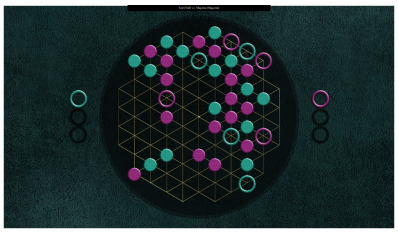


Table 2 red stop the green from removing the ring

Evaluations of the Algorithms

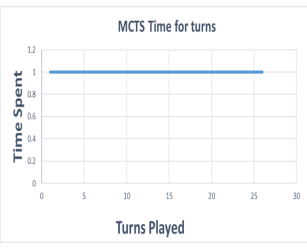
	Naïve	Minimax	Mcts	baseline
Minimax	100%	Nan	96%	97%
Mcts	75%	7%	Nan	10%

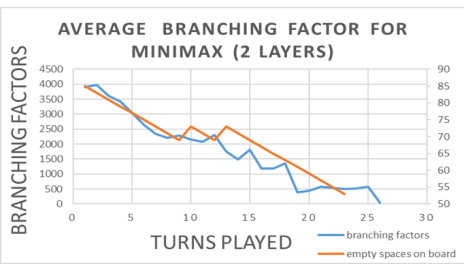
Table 3: win rate experiment result table.

Table 3 compares our agent(row)'s winning rate when tested against each other and with baseline. BFS with 5 seconds of thinking time was chosen as the baseline and the Naïve stands for random action selection. We simulated 100 tests for each of the pairs to minimize the influence of small probability events. The average turn of the minimax algorithm used to beat random is 23.2, and the average time taken to choose a good action for the minimax is around 0.300 seconds. **Additionally, the minimax player ranked 6th without teaching team players when this report was being written(24/5/2022). The winning rate, tie rate and the loss rate are 72%, 16% 14% respectively.**

The result listed in table 3 shows the win rate for all the algorithms in our experiment, as the table shows the minimax did a great job against all the algorithms, however, MCTS can only beat random action selection and lost most games against BFS with 5 seconds thinking time, probably because the MCTS failed to sample sufficient states within the time limit. **(The Alpha-zero fails to join the tournaments as it has such a complex algorithm[time complexity] and cannot make a good choice within 1s. It can only produce a general satisfactory result at around 5s)** According to the win rate result we got, minimax was chosen as the final implementation.

As shown in the Tabel4, the MCTS is guaranteed to give a local maximum in a specific thinking time(set to 0.95). This is because the MCTS updates the Q-value of each successor at the end of each epoch(selection ->expansion->simulation -> backpropagation). However, the sampling might not be sufficient to select a good action.





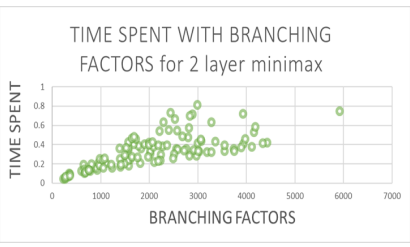


Table 4: MCTS time Table 5:branching factor for minimax Table 6: time spent for minimax

As shown in table 5, the average branching factor dropped as the game went on, but may have several spikes in the middle. When the game continues, empty places onboard would be occupied by markers, which results in fewer legal actions for the minimax algorithm to search.

To analyze the relationship between the branching factors of the minimax and the empty spaces on board, we also provided another line in orange showing the empty spaces on board when the game is on. Comparing the structure of the two lines is enough to conclude that the branching factors are dependent on empty spaces on board. A minimax algorithm with 4 layers would on average have a squared branching factor as we now have, which would make the time consumption unacceptable, so we limited the implementation of a 4 layered minimax according to the number of legal actions. Time spent by the minimax algorithm against the branching factors was also tested, and the result hints that the CPU could deal with around 6000 branching factors in one second in our experiment environment.

Improvements

MCTS agent

Building the actual policy network and value network instead of using q-value and heuristic approximation. Then finding records of actual Yinsh games from the internet to feed that two networks.

Minimax agent

The minimax player uses DFS; however, at the very end of the games, the branching factor is not very high. When coping with these states, the Iterative deepening might be used to help the player to get a more precise result(even might find out the goal state).

Innovation

Technique 1: Modify the state on the fly instead of using a deep copy

Using deep copy is very time-consuming. To solve this problem, two improvements were trie. 1. Using the pickle framework. However, the improvement is not significant. 2. We decided to change the board on the fly. The action getting to a state is recorded and this action was used to revert the new state to the original state. This on-the-fly modification makes the time consumption less than the deep-copy, saving around 9/10 of time.

Technique 2 + 3+ 4: Alphazero with a value network and a policy network

Unlike the normal MCTS covered in this subject, Alpha zero uses two neural networks: the value network and policy network to enhance the MCTS. However, training these two networks is not possible in 1 month. The reinforced SARSA learning and heuristic are used to approximate these two networks respectively. To be more specific, the pre-trained q function is used to estimate the score of each action. The H(state b) - H(state a) was used to estimate the reward from getting state a to state b.

Another innovation is that the greedy simulator is used instead of the random simulator. By observation, the random simulator gives a result of no more than a tie. Using the greedy simulator might help the tree estimate whether a state might lead to a win/ loss better.

Technique 5 Minimax algorithm + a-b pruning

Since this game is a zero-sum adversarial game, the minimax algorithm was used. The core assumption of minimax is maximizing the minimum gain while both players play admissibly. The player, who is known as max, would expand all the possible movements. Then max would imagine the opponent, who is known as min, would try to minimize the gain of max accordingly. Then the action resulting in the highest score, even minimized by min, would be chosen by max. Additionally, the a-b pruning will be used to prune the states where neither max nor min has an interest.