COMP 424: Project Report

Akshal Aniche

April 11th, 2019

1 Introduction

Game playing is a domain that has been central in the study of artificial intelligence. Games provide interesting frameworks to develop AI programs around, thanks to their clearly designed rules and the variety in types of games (for instance, perfect vs imperfect information and stochastic vs deterministic games).

One such game is the two-player game Pentago. Pentago is a game that has been perfectly solved: it's a first-player win game (Irving, 2014). In this project, we will explore a variant of Pentago, called Pentago-Swap: instead of rotating one of the quadrants, in this variant, players swap the positions of two quadrants. This variant doesn't guarantee a first player win. I will present an AI system designed to play this version of the game.

After describing my approach, consisting of three phases of the game, I will discuss the advantages and possible failures of my approach. Then, I will briefly compare it to other approaches I attempted. Finally I will explore possible improvements that can be explored to enhance this agent.

2 Proposed approach

2.1 Description

My agent's strategy is divided into three phases.
The first two phases are strictly offensive, whereas the third one is more neutral.

During the first three turns, the agent will attempt to place marbles in the center of the quadrants, while there are quadrants available. At the end of the first three turns, between two and three of the quadrants' centers are occupied by a marble placed by my agent. We will call such quadrants occupied. If the opponent tries to use a similar strategy of occupying the centers, then my agent will successfully occupy two of the four quadrants available.

For this purpose, at the first turn, a set of moves is generated (with swaps determined randomly) so that the agent can iterate through the set when selecting its move. If it doesn't find any legal moves in the set, then all of the quadrant centers are taken. If no quadrants are available, the agent moves on to the next phase.

The next phase is an offensive phase that lasts until the tenth turn. Here, the agent tries to align three marbles in one of the occupied quadrants. First, it will select the most promising of these quadrants, with the following scoring function, where α is a constant set to 0.25 :

$$f(Q) = \#(\text{own marbles in } Q) - \alpha * \#(\text{opponent marbles in } Q)$$

The agent restricts the moves considered to placing a marble in the highest scoring occupied quadrant. Since swapping quadrants doesn't affect the marbles inside the quadrant's position relative to one another, this becomes similar to Tic-Tac-Toe. With this assumption, and a further assumption that the opponent plays optimally in this theoretical game of Tic-Tac-Toe, the agent uses Minimax with $\alpha - \beta$ pruning (Russell & Norvig, 2016) to determine the move that will lead to a Tic-Tac-Toe win, and by extension, an alignment of 3 marbles inside a quadrant. In this algorithm, when scoring the leaves, losses, draws and wins that do not involve the central marble are scored 0, and other wins are scored 10.

When a desired alignment is achieved, or at the eleventh turn, whichever is earliest, the agent moves into the next phase.

The last phase consists in a Monte Carlo Tree Search method on Upper Confidence Trees (Couëtoux et al., 2011). The agent uses the standard Monte Carlo Tree Search algorithm on the complete Pentago board, where the tree policy also uses the UCT scoring function to choose from a node r a child node c to explore, where β is the constant defining the exploration rate, $\beta = 1.5$:

$$q(c) = \frac{score(c)}{\#visits(c)} + \beta\sqrt{\frac{\log(\#visits(r))}{\#visits(c)}}$$

During the expansion phase, when all the children of a node are generated, a game is simulated for all of them and they are scored.

During the rollout phase, the final board is scored this way: a win is worth 10 points, a draw is worth 5 points, a loss is worth 0 points. The move chosen is the move that gives the child c with the highest $q(c)$ value.

When executing each loop to find the best move, the agent has a time limit of 1.8 seconds to avoid timing out. The time limit being passed breaks the loop and the agent returns the best move it found with the information it computed so far.

For constructing and computing on the two trees types (Minimax tree and Monte Carlo Tree), I implemented two classes, resp. `student_player.TicTacToe` and `student_player.Node`.

2.2 Benefits and Drawbacks

The opening strategy of aiming for at least two quadrant centers makes it easier to win, because of the structure of the board. The quadrant centers are positions that stay quadrant centers even after swapping. On the other hand, for example, the lower left corner of the top right quadrant may be at the center of the board, but after swapping the top right and bottom left quadrants, it becomes the lower left corner of the entire board.

This strategy is only used for the first three moves, so it doesn't give the opponent time to build an advantageous position.

The second phase of the algorithm helps with the end goal of aligning five marbles. If we have three aligned in a quadrant and another quadrant with a marble in the center, we only need one marble and a few quadrant swaps to align five marbles.

Restricting the board to only consider the optimal quadrant optimizes space usage for the task. Using $\alpha - \beta$ pruning for this task helps with decreasing the running time in average since some branches are pruned and never explored.

However, the assumptions for this algorithm are too strong. We are assuming that the opponent will also restrict their moves to that quadrant and play optimally for aligning three marbles, which is unrealistic, and rarely true. For that reason, the optimal move can be missed, wasting turns during which an opponent can take advantage.

Finally, the Monte Carlo Tree Search method isn't as pessimistic as $\alpha - \beta$ pruning, which allows us to perform well against non optimal opponents in the later game.

Because we simulate a game from all the children of a node when they are created, during the first loop of the Monte Carlo Tree Search algorithm, the root's children have scores, which means that the first call to `findPromisingNode` can use the Upper Confidence Tree score to explore the most appropriate node, instead of one selected at random.

Using Upper Confidence Trees to select candidate play allows us to balance exploitation (exploring moves that have been effective) and exploration (trying moves that haven't been explored often).

This phase isn't exclusively offensive either: the move played is the one that appears to have the best long-term outcome. This creates an equilibrium between offensive and defensive play.

One disadvantage is that we might miss the optimal move again, which allows the opponent to gain an advantage, because of the randomness inherent to the Monte Carlo Tree Search. The time limit of 2 seconds also limits how much we can grow the tree, and thus the confidence we can have in the recommended move.

Overall, this strategy yields moves that are hard to defend against.

3 Other approaches

This approach achieves more than 99.5% wins against the random player, with the random seed unspecified.

In the development, I attempted a different scoring function for the boards in the Monte Carlo Tree Search algorithm. The score would be a linear function of the number of occurrences of the following characteristics in the board (for both the player and the opponent): 5 aligned, 4 aligned, 3 aligned, marble in the center of the quadrant. The coefficient for a given characteristic for the player is the negative of the coefficient for the same characteristic for the opponent.

The idea behind this scoring function was to score boards leading to a draw on a spectrum of how close to winning the agent was. However the scores calculated ended up in a very

noisy distribution where winning boards, losing boards, and draws weren't clearly distinct. The agent then worked with a function that lead to a poorer performance than the ternary scoring function, with around 85% wins against the random player.

4 Possible improvements

As discussed, a first attempt at using a scoring function with an image set of size larger than 3 was not an improvement. The image space being too large can be a possible explanation for this failure. We can explore different scoring criteria that lead to more separation between the nodes than the ternary function, without too much noise.

Instead of a linear combination of multiple features of the board, consider for example the longest alignment of marbles achieved by both players. Scoring the board according to which player achieved the longest alignment, and how many marbles are in that alignment could allow the agent to give preference to nodes where it gets a draw that is very close to a win over a draw close to a loss, in the Monte Carlo Tree Search algorithm.

To further improve the Monte Carlo Tree Search phase, after we've selected the most promising leaf to expand using the Upper Confidence Tree score, we can also select the first move from there using the same scoring function, instead of selecting a random move. However, this might not provide too much of an improvement, since the best child will be selected regardless in the next iteration if it scores better than the random child.

As we discussed in section 2.2, the second phase of my algorithm creates too many opportunities for the opponent to take advantage. For that phase, the only way we can improve the algorithm is by using a planning algorithm to implement both an offensive and a defensive strategy, without restricting the agent to a quadrant. This would be much faster to achieve if the history of moves was public. That way, the agent could look at the opponents last move to locate the changes in the board instead of iterating through the entire board to guess the last move executed.

Finally, except for the Monte Carlo Tree Search algorithm, the other phases do not consider what swaps would be most effective. The agent chooses two quadrants at random and swaps them. Studying more how swaps affect the early game would allow me to construct more sensible swaps for opening moves.

5 Conclusion

In conclusion, in this project, we attempt to create an agent to win at Pentago-Swap, a variant of the original Pentago. Even though Pentago is perfectly solved, this variant isn't, and so, I designed an algorithm in three phases.

The first phase consists of a set of opening moves that aim to control the centers of the four quadrants of the board. Then, the second phase tries to align three marbles within a quadrant while using the center marble. By imitating the implementation of the Minimax algorithm with $\alpha - \beta$ pruning for Tic Tac Toe, I aim to align three marbles so that the line

can be used with another quadrant whose center is controlled to align 5 marbles. Finally, using the Monte Carlo Tree Search algorithm on Upper Confidence Trees for the rest of the game, the agent selects the move that has the highest score.

The different phases of this algorithm have various advantages and disadvantages. A few of these advantages include selecting moves that are hard to defend against in the beginning and shifting towards a more balanced strategy in the later game. But this also means that the other player can easily take advantage of weak defensive spots.

To improve on this algorithm, we can explore different, more discriminating, scoring functions for all phases, as well as more comprehensive swaps. The most important improvement would be redesigning the second phase of the algorithm to make fewer, less strict assumptions. This would greatly improve the early performance of the agent, leading to stronger endgame moves.

In the end, exploring variants of classic games allows us to explore different strategies that do not apply to the original game. It helps contribute to creating stronger game playing AI algorithms.

References

Couëtoux, A., Hoock, J.-B., Sokolovska, N., Teytaud, O., & Bonnard, N. (2011). Continuous upper confidence trees. In *International Conference on Learning and Intelligent Optimization*, (pp. 433–445). Springer.

Irving, G. (2014). Pentago is a first player win: Strongly solving a game using parallel in-core retrograde analysis. *arXiv preprint arXiv:1404.0743*.

Russell, S. J., & Norvig, P. (2016). *Artificial Intelligence: a Modern Approach*, chap. 5.3. Malaysia; Pearson Education Limited, 3 ed.