CS3243R Project Report

Yao Yujian

Abstract

In this project, we investigate the approach of modeling Hearts as a Partially Observable Markov Decision Process by assuming opponents to be following a fixed strategy. Experiments are also run to test out the performance of this approach. The result, however, shows that for this model, a more sophisticated algorithm POMCP fails to out-perform the simpler algorithm with only one-step look-ahead.

1 Introduction

Hearts is a card game that's usually played among four. It uses the standard set of 52 cards, without the jokers. At the start of the game, each player is randomly dealt with 13 cards. Then each player passes 3 cards to the next one. The player with 2 of Club starts first. In each round, after one player starts, the others follow in clockwise manner until everyone has played a card. Other players also have to follow the suit of the first card, unless they do not have the suit at hand, in which case they are free to play any cards they have. After a round ends, the player with the largest card that has the same suit as the first card has to take all the cards that has been played in the round and add a score based on the cards he has taken. The next round will also start with this player. The player that starts a round can play any cards except hearts unless he has no cards of other suits, until when *heart is broken*, which happens when a player plays hearts. Cards in heart suit each has a score of 1 and Q of Spade has a score of 13, the other cards do not have any scores. A special scenario called *shooting the moon* happens when one player gathers all cards that has a score, i.e., earns 26 points. When this happens, he gets 0 points instead, and all other players each gets 26 points. The goal of the game is to minimize one's score.

Typically the game can be repeatedly played with the score of each player accumulated until at least one reaches a score of 100. At that point the player with the lowest score wins.[1]

Like most card games, the environment of Hearts is partially observable, as the players cannot see each other's cards at hand. With the information (the cards that each player has played, the cards the current player has), the number of possible samples is enormous (in the worst case, one only knows the 13 cards he has, so the number of possible card configurations other players can have is $\frac{39!}{13! \times 13! \times 13!} \approx 8 \times 10^{16}$). Moreover, at each round, there are usually large number of possible actions (in the worst case, it will be the number of cards the player current has). This means that using classical game theoretical minimax approach will be extremely slow due to the huge branching factor.

Therefore in this project, we investigate the approach with which the game is formulated as a Partially Observable Markov Decision Process (POMDP) by assuming that other players employ a fixed set of strategies.

[1]More details about the rules of the game can be found on http://en.wikipedia.org/wiki/Hearts.

2 Algorithms

This section describes the few algorithms we implemented in this project.

2.1 Greedy Heuristic

This algorithm greedily selects the best card in the set of all currently playable cards with the following rules:

1. If the player is the first player in the round, pick the smallest card
2. Otherwise
    a. If the player does not need to play the same suit as the first player
        1. If the player can play Q of Spade, pick Q of Spade
        2. If the player can play hearts, pick the largest card of hearts
        3. Otherwise, pick the largest card
    b. Otherwise
        1. if the player can pick a smaller card than the first card, pick the largest that is smaller than the first card
        2. Otherwise,
            a. If the player is not the last player, pick the smallest card
            b. Otherwise, pick the largest card

Note that there is no look-ahead involved.

The intuition is that the player should get rid of score cards as soon as possible, so that there will be less risk in the future that he will have to absorb those score. Similarly he should try to get rid of large cards as soon as possible as those can make him absorb scores in the future. Lastly, the player should try not to take in any score and/or become the first player in the next round if possible.

2.2 Simple Monte-Carlo Simulation

In this algorithm, all observations are used to generate some constraints on the configuration of each player's cards. Based on the constraints, the algorithm then randomly generates k number of samples. For each sample, the algorithm iteratively selects and simulates each possible move, after which it simulates the whole game from there by assuming all players to be following the greedy heuristics. The algorithm also maintains a total score for each move and adds increment each as it finishes a simulation. Then the move with the minimum total score is selected as the best move. The details on how to generate the constraints and then samples are available in the next section.

2.2.1 Generating Samples From Observations

All possible observations include:

1) the cards the player himself has,
2) the cards the player himself passed to the other player before the first round and
3) the cards each other player has played.

With 1) and 3) the player can know what cards other players cannot have. With 2) the player knows that the particular player who he has passed cards to will have that few cards, unless through 3) he sees that the cards have been played. 3) can be further exploited by considering the rules of the game. If a player plays a card with a different suit than the first card in a round, then it is clear that this player does not have cards of that suit any more.

Therefore with the observation, a player knows that 1) all three other players can only have cards in a known set of cards, 2) one player may have some cards and 3) certain players cannot have certain cards based on the rules of the game.

One can further make use of the observation and the assumption on the strategy that the players employ and generate even tighter constraints. For example, if the player always plays the smallest card when he has to start the round, then we can be sure that the player does not have cards smaller than the card he plays when he starts the round. However, this assumption makes sampling a lot less robust as it can generate unsatisfiable constraints when the other players do not follow the strategy exactly. Therefore it is not applied.

With the constraints, we can then randomly generate samples with the following algorithm:

Iterate through all cards that other players can have. For each card, randomly assign it to a player that can have the card. Note that we can mark a player as being unable to get any cards once the number of cards assigned to him become is equal to the number of cards he should have. Terminate and restart the process if such distribution becomes impossible, i.e., a card cannot be assigned to anyone. Otherwise, produce the sample if we have successfully assign all cards. This algorithm can be sped up by sorting the cards in descending order of numbers of players it cannot be assigned to before the distribution, i.e., applying a most constraint variable heuristic.

2.3 POMCP

2.3.1 Formulation as a POMDP

By assuming each opponents to be following the greedy heuristics, we can further formulate the game as a POMDP with the following attributes:

State $\mathcal{S}$: The cards and scores of all players, the cards that have been played but have not yet been taken by any player and whether heart has been broken. Note that in each state other than s_0, it will always be the current player's turn.

Action $\mathcal{A}$: The cards the current player can play

State Transition: After the player's action, assuming other players continue to play with the greedy heuristic, new rounds may be started and other players may further play other cards, until it is again the current player's turn, at which point a new state is considered to have been reached after the action.

Observation $\mathcal{O}$: The cards the current player has, the scores of all players and the cards that have been played

Reward $\mathcal{R}$: Negation of the player's increase in score until his next turn
Transition Probability: Deterministic, since all other players employ a deterministic strategy.
Initial State s_0: The start of the game.

2.3.2 Algorithm

In the project, we use the POMCP algorithm from *(Silver and Veness (2010))*. Some modifications are:

1. Since the game is finite (13 rounds per game), γ is set to 1.
2. In *rollout*, the game is not random but follows the same Monte-Carlo simulation in the previous section (all players use the greedy heuristic).
3. Instead of using a particle filter to generate samples, we use the same algorithm in Section 2.2.1.

3 Experiments

We conducted experiments to test the performance of each strategy. For both Simple Monte-Carlo and POMCP, we allow 0.5 seconds per round for sampling and simulation. We also add in a random AI so as to test the effect of disrupting the model.

For each experiment, we set up each player to follow a certain strategy, then let them play 1000 games and sum up all the score. Note that unlike in a typical game, in the experiments we do not terminate the game when one player reaches a score of 100.

Player	Algorithm	Score	Ratio to sum of all scores
1	Greedy	8445	0.299638093
2	Simple Monte-Carlo	4169	0.147920806
3	Greedy	7694	0.272991768
4	Greedy	7876	0.279449333
All	-	28184	1

Table 1: Simple Monte-Carlo vs. Three Greedy

Player	Algorithm	Score	Ratio to sum of all scores
1	Greedy	8172	0.287827557
2	POMCP	4154	0.146308819
3	Greedy	8287	0.291877994
4	Greedy	7779	0.27398563
All	-	28392	1

Table 2: POMCP vs. Three Greedy

As can be seen, by looking ahead, both POMCP and Simple Monte-Carlo out-performs the basic greedy strategy. However, it seems that the more sophisticated POMCP algorithm fails to out-perform the myopic simpler method. A possible explanation can be that POMCP incurs more overhead, and therefore does not generate as many samples. Or it can be that the run time is too short for POMCP to exploit enough simulation to effectively outmatch the simple Monte-Carlo method.

Player	Algorithm	Score	Ratio to sum of all scores
1	Random	11573	0.427174074
2	Simple Monte-Carlo	3785	0.139709139
3	Greedy	6088	0.224715783
4	Greedy	5646	0.208401004
All	-	27092	1

Table 3: Monte-Carlo vs. Two Greedy vs. Random

Player	Algorithm	Score	Ratio to sum of all scores
1	Random	11627	0.422677039
2	POMCP	3743	0.136069507
3	Greedy	6034	0.21935437
4	Greedy	6104	0.221899084
All	-	27508	1

Table 4: POMCP vs. Two Greedy vs. Random

Again the result is almost identical. As compared to the previous pair, however, we can see that the looking ahead algorithms performs slightly worse. In Tables 1 and 2, the ratio of the score of looking ahead algorithms to greedy is about $0.14/0.28 \approx 0.5$, whereas in Tables 3 and 4, the ratios become $0.13/0.21 \approx 0.6$. This shows that adding a random player does disrupt the computation of both algorithms, although they seem to be still quite tolerant to the disruption.

Player	Algorithm	Score	Ratio to sum of all scores
1	Simple Monte-Carlo	5202	0.18525641
2	POMCP	5156	0.183618234
3	Greedy	8756	0.311823362
4	Greedy	8966	0.319301994
All	-	28080	1

Table 5: POMCP vs. Simple Monte-Carlo vs. Two Greedy

Similarly, the performance of the two looking-ahead algorithm is almost identical. Interestingly though, the ratio of the score of looking ahead algorithms to greedy is about 0.6 again. So it seems that the disruption caused by another look-ahead algorithm is close to that caused by a random player.

Player	Algorithm	Score	Ratio to sum of all scores
1	Simple Monte-Carlo	3894	0.142637363
2	POMCP	4215	0.154395604
3	Random	12750	0.467032967
4	Greedy	6441	0.235934066
All	-	27300	1

Table 6: POMCP vs. Simple Monte-Carlo vs. Random vs. Greedy

In Table 6, however, there seemed to be a slight difference between the performance of the Simple Monte-Carlo method and that of the POMCP algorithm, although the difference is not very significant. If indeed this signifies a difference in performance, one may also conclude that the one step rollout appears to be more tolerant to disruptions (in this setup, only one player is actually following the heuristic), and thus performed better.

Again, the ratio of look-ahead to greedy is still about 0.6 for Simple Monte-Carlo, but increases to 0.65 for POMCP. So again an additional non-conforming player does not seem to introduce too much disruption.

Player	Algorithm	Score	Ratio to sum of all scores
1	Simple Monte-Carlo	6836	0.250880799
2	POMCP	7341	0.269414269
3	Simple Monte-Carlo	6864	0.251908397
4	Simple Monte-Carlo	6207	0.227796536
All	-	27248	1
Average of 1, 3, 4	-	6635.67	0.2435285770

Table 7: POMCP vs. Three Simple Monte-Carlo

Player	Algorithm	Score	Ratio to sum of all scores
1	POMCP	7588	0.276369464
2	Simple Monte-Carlo	6626	0.241331585
3	POMCP	6371	0.232043998
4	POMCP	6871	0.250254953
All	-	27456	1
Average of 1, 3, 4	-	6943.33	0.2528894716

Table 8: Three POMCP vs. Simple MOnte-Carlo

Again it seems that POMCP performs slightly worse than the Simple Monte-Carlo algorithm. The reason can again be that POMCP is less tolerant to disruptions as compared to the Simple Monte-Carlo algorithm.

4 Future Work

While it appears that POMCP did not perform as well as the simpler methods, it may be worthwhile to investigate the effect of changing some variables such as the run time (perhaps POMCP needs more sampling to converge) or the constant c in the POMCP algorithm which determines the tendency towards exploration or exploitation.

It is also possible to try out different rollout methods, for example, a random rollout instead of the currently deterministic rollout.

It may also be interesting to play the algorithm against opponents that uses a different strategy and test if such formulation can be a good enough approximation in real-life Hearts game play.

Lastly, a further improvement for the model in general is to consider a set of possible strategies instead of only the current single greedy heuristic.

5 Conclusion

In conclusion, by modeling the opponents to be using a known strategy, one can formulate the Hearts game as a POMDP. Experiments are run to test out the performance of the POMCP algorithm. While algorithms with look-ahead out-performs the greedy strategy, it seems that the more sophisticated algorithm does not out-perform the simpler version.

6 References

Silver, David, and Joel Veness. 2010. "Monte-Carlo Planning in Large POMDPs." In *Advances in Neural Information Processing Systems 23*, edited by J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, 2164–2172.