[COMP6247] Coursework Report

Feng Xie — fx1n18@soton.ac.uk — Student ID:30502322

1 Introduction

In the lemonade stand game, there are three sellers selling lemonade to customers in twelve islands. The locations of islands are distributed like hours in the clock. The sellers share a common aim which is to attract the most customers as customers will choose the nearest seller to buy lemonade. The sellers change their locations at night and are not allowed to change locations in the daytime. When sellers are changing locations, they have no idea which island the others sellers will choose(i.e. they choose a new location independently). But they are allowed to know the locations of other sellers at the end of the day. Therefore, the goal of sellers is to pick a location as far as possible away from the other two sellers.

At the start of each day, there are 12 customers on each island. Because there are 12 islands, the total utility each day is 144 dollars(i.e. Each customer pay one dollar to get the lemonade.). Every customer goes to the nearest lemonade stand, with ties split evenly. Consider each seller as an agent, each agent's payoff is directly proportional to the distance to the other agents[1]. In this way, the utility of each agent is actually the sum of its distances to the other two agents multiplied by six.

Each game lasts for 100 days(i.e. 100 rounds). And the game is repeated for 1000 times to get statistical significance. At the end of each 100 rounds, the cumulative rewards of sellers are cleared and the game is reset.

Figure 1 shows an example of lemonade stand game. Three sellers are located in the circles filled with black, blue and green. At the end of the day, the black seller receives 42 dollars. The blue seller receives 48 dollars and the green seller receives 54 dollars. The rewards sum up to 144.

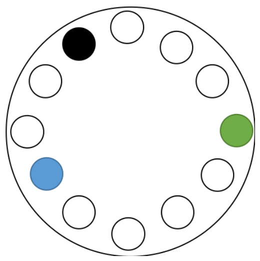
Figure 1: Lemonade Stand Game Example

2 Algorithm Design

2.1 General Analysis

The rules of Lemonade Stand Game are simple. What makes it difficult to win is the players playing the game. Since we don't know how the other agents move and what strategy they apply, it is very difficult to play a strategy which only considers the reward changes of our own and win the game. We might be exploited by others if we don't consider how they move.

This is a kind of constant-sum game. The total reward of each round is 144. It is difficult for one agent to get a high score because that explicitly decreases the score of the others considering that every one is try to achieve a high cumulative score. Therefore, the only way to get a high score in this game is to cooperate with one other agent and exploit the third, or accurately predict future actions of the agent's opponents[1].

Simple reinforcement learning algorithms like Q-Learning are not suitable for this game because the environment is dynamic. In Q-Learning, we update the Q-table at each time step and we assume that each state has a best action just as the Mountain Car problem where we try to drive the car to reach the top. In this game, when the agent arrives at the state that has been visited(i.e. The locations of the agents are the same.), the environment may have changed, which means that the other agents may choose different actions even at the same state. Q-Learning is not suitable when the environment is changing.

Figure 2 shows some strategic patterns[2] in the Lemonade Stand Game which help analyse and understand the design of strategies. The **Stick** pattern means the agent stays in the same location all the time. The **Collision** pattern refers to the situation where two agents choose the same location, making the third agent achieve the highest reward. The reward of three agents is the same in the **Equilateral** pattern where the distance between two agents is the same. The **Across** pattern is a collaborative pattern in which two agents sit opposite each other, making the third agent always get a reward of 36 no matter which location it chooses. The **Sandwich Offer** is a pattern that leads to the **Sandwich** pattern if the third agent recognises and accepts the offer. In Figure 2, the circle of 12 is the victim in the Sandwich pattern. He gets a reward of 12 which the other two get 66.

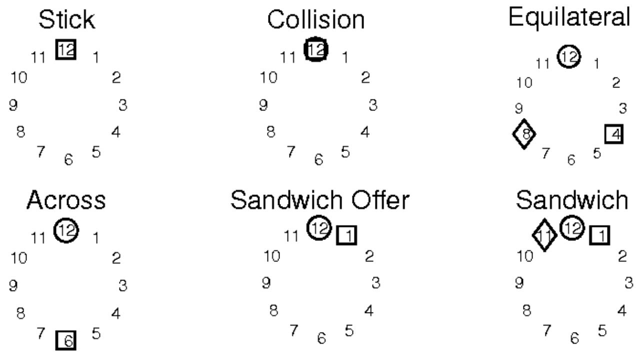

Figure 2: The strategic patterns introduced in [2]

2.2 FPL

The Follow the Perturbed Leader(FPL) algorithm serves as a simple strategy for comparison. It is simple because it neither cooperate with others nor predicts how the others move. At the end of each round, FPL observes the reward it gets for its chosen location and stores the cumulative reward for each location over time. It makes a decision by adding some random noise from an exponential distribution to the cumulative reward of each location and choosing the location with the biggest value.

The rationale behind FPL is following the location which has the most cumulative rewards so far. However, it has no idea how the other agents will move and may become the victim if it is exploited by the other agents.

2.3 Modified Constant

The idea is taken from [2] which has a list of the official tournament strategies and Modified Constant, or Pujara, is one of them.

Basically, it is modified from Constant strategy which chooses a random initial location and then Stick. Modified Constant also picks a random initial location, however, if the number of time steps with low utility is over a cutoff value, it will choose a new random location. And if a number of time steps with high utility occurs, the number of low utility record will be reset to 0. The low utility is set to 36 which is the reward of the third agent when it is trapped in the Across pattern described in 2.1. The high utility is set to 48 which is the reward of each agent gets in the pattern of Equilateral. The cutoff values for the low utility count and high utility count are set to 3. So when the number of low utility count is over 3, the agent will choose another random location, and when the number of high utility count is over 3, that means the agent has got a high reward at the current location for 3 times and the number of low utility count will be reset to zero. The number of high utility count will also be set to zero in this case.

The rationale behind this is that although the agent does not cooperate with others, it make itself difficult to predict by changing its location randomly under a certain condition. What makes it different from FPL is that it adjusts its location by monitoring the short-term change of rewards while FPL depends on the long-term cumulative rewards which could make FPL fail if the environment is changing and it depends too much on its past rewards.

2.4 EA^2

The idea is taken from [1]. Basically, it tries to find another agent to cooperate and exploit the third agent by making up the Across pattern as described in 2.1. In this way, the two agents that form a cooperation relationship share a high utility of 108 while the third agent can only receive a reward of 36 no matter where it moves.

The paper defines two ideal types of strategies that an agent is easy to cooperate with. The first type is **Stick**. That means an agent chooses a location and sticks to it. The other type is **Follow**, which means an agent is following an opponent by sitting opposite the opponent's last location. An agent that uses a Stick strategy is the one that barely move and an agent that applies a Follow strategy tends to choose locations within the best response area from the last time step.

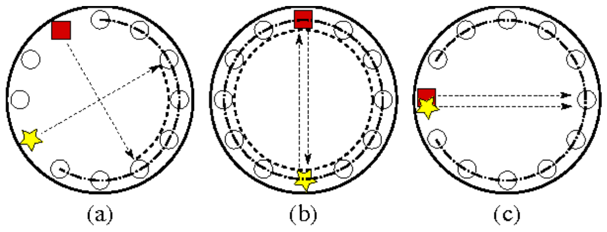

Figure 3: Best responses for different opponent configurations.[1] For an agent, the best response area is the dot-dashed segment.

EA^2 maintains three kinds of indices, s_i, f_i and f_{ij}, to indicate the opponent's proximity to playing either a stick or follow strategy. s_i measures if the opponent is playing a stick strategy. f_i measures if the opponent is generally following others and f_{ij} measures if player i is following player j. Generally, cooperation with an opponent with a high stick index involves sitting opposite the opponent and cooperation with an opponent with a high follow index involves sitting still and waiting for the opponent to play the appropriate action.

And if the EA^2 agent finds itself being exploited by the other two(i.e. The other two opponents have formed an Across strategic pattern), it will play a "carrot and stick" strategy to induce the opponent with a higher follow index to cooperate with it. The actions will be divided into two sides based on the position of Stick player(which has a lower follow index). And a measure of bias will be computed with recent best responses of the Follow player. The EA^2 will firstly sit on top of Follow player, trying to decrease its reward and then attempt to push the Follow player to the side with bias by playing the opposite side. In this way, the Follow player realizes that it will get more reward if it cooperates with EA^2.

The pseudo-code demonstrating how different types of opponents map actions is shown in Figure 4.

3 Evaluation

To measure the performance of the strategies described above, two simple testing strategies are implemented for comparison, along with two strategies, External Regret Matching(ERM) strategy and Constant strategy, which are already implemented in the framework.

One of the two simple testing strategies is Q-Learning strategy. The state is designed to be the locations of two opponents in the last round. Together with the current action of the agent, the Q-table is stored in a 3-dimension array. The action is chosen according to the epsilon-greedy algorithm. The epsilon, learning rate α, discount factor γ are set to 0.3, 0.05 and 0.95 respectively. At the end of each round, the agent observes the reward it gets and considers the locations of two opponents as the next state and updates the Q value for the current action in the Q-table.

The other testing strategy is PinchTheConstantOrElse(PTCO) which is derived from [2]. It initializes with a random location and identifies a fixed opponent if one present and aggressively attempts to give a Sandwich Offer(described in 2.1) to the other opponent for exploiting the fixed opponent. If the attempt fails, it will punish the non-fixed opponent by sitting opposite the fixed one, which forms an Across strategic pattern and limits the reward of the non-fixed agent to 36. In other cases, the agent sticks to its current

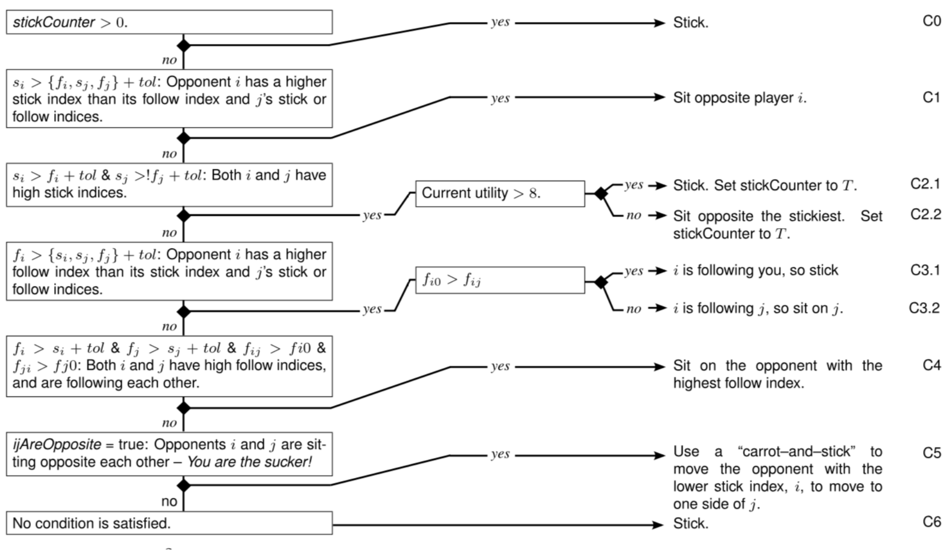
stickCounter > 0.
yes
Stick.
C0
no
$s_i > \{f_i, s_j, f_j\} + tol$: Opponent i has a higher stick index than its follow index and j's stick or follow indices.
yes
Sit opposite player i.
C1
no
$s_i > f_i + tol$ & $s_j >! f_j + tol$: Both i and j have high stick indices.
yes
Current utility > 8.
yes
Stick. Set stickCounter to T.
C2.1
no
Sit opposite the stickiest. Set stickCounter to T.
C2.2
no
$f_i > \{s_i, s_j, f_j\} + tol$: Opponent i has a higher follow index than its stick index and j's stick or follow indices.
yes
$f_{i0} > f_{ij}$
yes
i is following you, so stick
C3.1
no
i is following j, so sit on j.
C3.2
no
$f_i > s_i + tol$ & $f_j > s_j + tol$ & $f_{ij} > fi0$ & $f_{ji} > fj0$: Both i and j have high follow indices, and are following each other.
yes
Sit on the opponent with the highest follow index.
C4
no
ijAreOpposite = true: Opponents i and j are sitting opposite each other – *You are the sucker!*
yes
Use a "carrot–and–stick" to move the opponent with the lower stick index, i, to move to one side of j.
C5
no
No condition is satisfied.
Stick.
C6

Figure 4: Pseudo-code for EA2 [1]. The parameter tol controls the confidence with which a mapping must be held for a condition to be satisfied.

action. In order to identify the fixed opponent, in the implementation the agent looks back at the past time steps and if an opponent has stayed at the same location for 5 time steps it is said to be fixed.

Rank	Strategy	Utility	Standard Deviation
1	ERM	1.3456	0.1637
2	EA^2	1.3004	0.1608
3	ModifiedConstant	1.0340	0.1726
4	Constant	0.2182	0.2023
5	FPL	-0.5339	0.1788
6	Q-Learning	-1.4482	0.1599
7	PTCO	-1.9161	0.1283

Table 1: Results of different strategies

Table 1 shows the results of different strategies which are run with 100 rounds and 1000 repeats. The experiments are run with *LifelongLearningTournement* in the framework provided. We can see that EA^2 strategy outperforms most of the opponents using other strategies. The advantage of EA^2 is that it always tries to collaborate with an opponent and exploit the third so the strategies that don't consider how others move will suffer and be exploited. We can see from the table that the utility of EA^2 is much higher than Constant, FPL, Q-learning and PTCO. Besides, when EA^2 suffers from being exploited, it is able to break the others' collaboration by apply the "carrot and stick" method. Additionally, the collaboration with Across pattern which EA^2 applies is much more stable than Sandwich Offer which PTCO tries to achieve. Sandwich Offer is unstable because it is easy for the third agent to recognise the trap as well as escaping it. Therefore, EA^2 cannot be easily defeated. However, the disadvantage of EA^2 is that once the collaboration is formed, the high utility is shared between EA^2 and the other opponent so EA^2 doesn't have a bigger chance of getting the highest utility, which explains that in table 1, ERM has a slightly higher utility than EA^2. Another disadvantage of EA^2 is that it does not consider the future actions of the opponents and EA^2 is also indifferent to the types of opponents(i.e. Stick, Follow). The opponent EA^2 chooses to cooperate with is the one with higher statistical chances[3].

The utility of ModifiedConstant is a bit lower than the top two strategies in table 1 but it is still a lot higher than the others. Constant Strategy has a worse performance. Compared to Constant strategy, ModifiedConstant monitors the change of its reward at the current location and randomly picks a new location if a number of time steps with low utility occurs, which makes it more unpredictable and meanwhile, it achieves a high utility over time. In contrast, Constant strategy doesn't adapt to others' change and suffers from being exploited. But Constant strategy is better than FPL, Q-Learning or PTCO because the other three are not able to identify if the agent is staying at the same location. The disadvantage of ModifiedConstant is that although it tries to avoid being stuck in a location of low utility, it doesn't collaborate with others and thus can be exploited.

FPL takes actions according to the cumulative rewards on each location. One thing about looking back at the past rewards and choosing the location that has the best performance so far is that it helps avoid some apparent traps like the one set by PTCO. But the past usually cannot be trusted in this game because the opponent may be able to adapt to others' move and adjust its strategy. In this case, FPL will suffer because it depends too much on the past experience. Also, FPL is easy to be exploited because as rewards accumulate over time, it is hard for FPL to change locations unpredictably or in other words, it takes much more time to adapt to others' change.

4 Conclusion

In this report, three strategies are designed and evaluated for Lemonade Stand Game, including FPL, Modified Constant and EA^2. It was found that EA^2 outperformed most of the strategies mentioned in this report due to its ability to consistently find an appropriate opponent to cooperate with and form an Across pattern to exploit the third agent. In Lemonade Stand Game, it is almost not possible to always achieve a high utility without cooperating with others because otherwise the agent might be exploited by others.

As what is discussed above, EA^2 doesn't consider the future behavior of its opponents. In [3], a model-based RL algorithm is proposed to predict its opponents' future actions and select an opponent with higher predicted long-term utility to cooperate with. The performance turned out to be better than EA^2, which indicates that predicting future actions of opponents helps achieve a better cooperation strategy and a better performance.

References

[1] Adam M Sykulski, Archie C Chapman, Enrique Munoz De Cote, and Nicholas R Jennings. Ea2: The winning strategy for the inaugural lemonade stand game tournament. In *ECAI*, pages 209–214, 2010.

[2] Michael Wunder, Michael Littman, Michael Kaisers, and John Robert Yaros. A cognitive hierarchy model applied to the lemonade game. In *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[3] Enrique Munoz De Cote, Archie C Chapman, Adam M Sykulski, and Nicholas R Jennings. Automated planning in repeated adversarial games. *arXiv preprint arXiv:1203.3498*, 2012.