Winning the Lemonade Stand Game

Michael Robertson and **Fabio von Schelling Goldman**
{mirobertson,favonschellinggoldman}@davidson.edu
Davidson College
Davidson, NC 28035
U.S.A.

Abstract

We analyze the economical strategy lemonade stand game and design agents to compete in a class-wide tournament. After designing and testing implementations of published strategies, we designed agents created to counter the most successful published agents by targeting their weaknesses. However, because our in class preparatory tournament showed a trivially simple agent outperformed multiple more complex strategies, we modified our approach. Believing that strategies which attempt to predict other players behavior often over-analyze to their own determent, we ultimately selected one of the most simple agents, in the hopes it would succeed as the simple bot did in the preliminary tournament.

1 Introduction

Three players play the lemonade stand game on a circular board with twelve positions. Each player places their stand on a position, and one position can hold all three stands at the same time. On each of these positions sit two customers, who will patronize the lemonade stand closest to them, and split in the case that they are equidistant from two stands. Stands can be repositioned each round, and player with the greatest number of customers (points) over a set number of rounds wins the game.

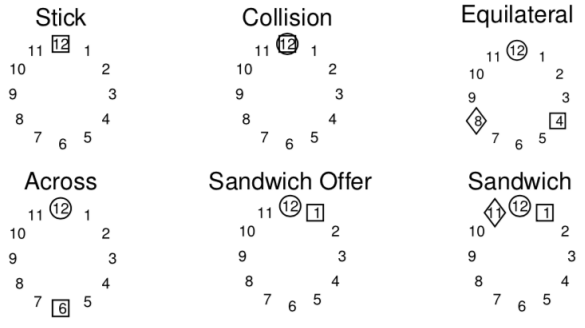

Figure 1: Six common scenarios in lemonade game, with circle, square, and diamond players. Image from Wunder et al. (2010)

We developed and tested different agents (bots) to play this game in preparation for a class-wide tournament. From a game theory perspective, we see that many Nash equilibria exist, including any equidistant spacing of the stands. However, such an existence only guarantees that perfectly rational players who are also aware that all other players have the same rationality would have no immediate incentive to change their position once in a equilibrium. Without knowledge of other agents reasoning, and without any confidence that the game would ever fall into any Nash equilibrium, we implemented several agent designs found in Wunder et al. and analyzed their performance by testing them amongst themselves as well as other basic agents(Wunder et al. 2010). We predicted that others in the class would enter these strategies in the tournament, and so we entered agents specifically designed to exploit weaknesses of most successful agent in (Wunder et al. 2010), EA^2.

2 Background

At our highest levels of implemented cognition, we attempt to predict others' reasoning based on the assumption that the rest of the class discovered and worked with the same papers we did. However, for the most basic functioning of our bots, we rely upon a few simple mechanisms designed to optimize our score.

Stick

As simple as its name suggests, a bot running *stick* simply repeats the same position it held the previous turn, regardless of all other conditions. The supplied "FiveOClockBot" exclusively implements this maneuver, and many of our bots incorporate it into their overall strategies.

Across

In this maneuver, the bot assumes that another player is implementing stick, and plays the position across from another.

Sandwich

Two players *sandwich* a third when they each realize that one player is sticking and the other also has sandwiching capabilities. To "offer" a sandwich, a player must notice another is sticking, then position themselves immediately next

to them. If the third player notices this offer, and places itself on the other side of the sticking player, a successful sandwich has occurred. The two sandwiching players will then presumably continue to sandwich for as long as possible. For long-term sandwiching to occur intentionally, three conditions must be met:

1. A player must stick without regard to its own score
2. One player must have the ability to recognize a sticking player as well as the ability to offer a sandwich
3. The other player must have the ability to recognize an offered sandwich and accept it.

Incorporating these basic strategies into more complex ones, we created the following bots.

Simple Strategies

- RandomBot: A bot that just randomly decides its position every round.
- FiveOClockBot: A bot that decides on one location and will play that position every round.
- ModConst: A bot that chooses a random initial position and every round performs a constant step size to determine its new position.

Complex Strategies

- ModifiedConstant: we implemented a Modified Constant bot inspired by the author "Pujara", who entered it in the tournament run in (Wunder et al. 2010). In our implementation, this bot sticks at a random spot, and only moves to a new random spot if it scores below 7 for three consecutive turns.
- CoOpp: This Cooperation strategy developed by "RL3" in (Wunder et al. 2010) randomly chooses a player to play across from, unless it detects a sticky player, in which case it will offer a sandwich. In the case that it sees an offer of a sandwich, it will accept the offer.
- ACTR: We were not able to find a paper detailing Lebiere and CMU's *ACTR*, but we were able to implement it generally by its short description in Wunder et al.'s tournament (Wunder et al. 2010). ACTR follows a simple cycling of three distinct strategies. It either sticks to its last position, goes opposite of the weakest opponent or goes opposite to the strongest opponent.
- EA^2: One of the bots we implemented was based on the EA^2 algorithm outlined in (Sykulski et al. 2010). Following their original pseudo code as seen in Figure 2 the algorithm makes use of three distinct formulas. S_i, which quantifies the likelihood that player i will stick in its last position, f_i, which quantifies the likelihood that player i will follow another player and $f_{i,j}$, which quantifies the

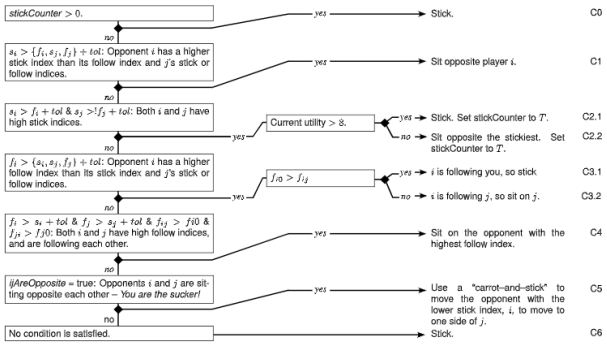

Figure 2: The original pseudo code as detailed by Sykulski et al.

likelihood that player i will follow player j.

$$s_i = -\sum_{k=2}^{t-1} \frac{\gamma^{t-1-k}}{\Gamma} d(a_i(k), a_i(k-1))^p \quad (1)$$

$$f_{ij} = -\sum_{k=2}^{t-1} \frac{\gamma^{t-1-k}}{\Gamma} d(a_i(k), a_j^*(k-1))^p \quad (2)$$

$$f_i = -\sum_{k=2}^{t-1} \frac{\gamma^{t-1-k}}{\Gamma} \min_{j=N\setminus i} d(a_i(k), a_i^*(k-1))^p \quad (3)$$

where $\Gamma = \sum_{k=2}^{t-1} \gamma^{t-1-k}$. d denotes a metric which provides the absolute difference based on t being the round, and the * denoting the opposite position. On a basic level EA-Squared tries to predict if a given player will be sticking to their past position and if not if they are playing a strategy in which they are following another player. Based on it's evaluation of these two conditions EA-Squared tries to optimize it's position for the next round.

In our implementation, we kept the suggested values of $\rho = 0.5, \gamma = 0.75,$ and$tol = 0.1$, however we performed two key differences compared to 2. Firstly we assumed in the second part that the ! is a typographical mistake and ignored it. Furthermore we did not employ the Stick and Carrot method to lure opponents off their current strategy and instead just took our past position and moved one back. The idea hereby was to provide a rather simplistic pattern that could possibly allure an opponent to vary from their current strategy at which point EA-squared hopefully would be able to gain control again.

- EA^{2}Beater: As we assumed the meta game would be dominated by the EA-Squared strategy we tried to design a bot to try and exploit this overabundance of a single strategy. Our main idea here was to try to focus on a weakness of EA-Squared which is that if the condition of ($s_i > f_i + tol$ & $s_j > f_j + tol$) is triggered by the player playing EA-Squared, they will set their *stickCounter* to a constant value of T. Assuming T is greater 0 this means that in any future round EA-squared will always stick in its current position. So in essence we wanted to create a bot that will stick until EA-Sqaured, if anybody is playing that strategy, would go into the branch of setting their

stickCounter and at that point switch strategies. The strategy we decided to switch to was maxUtil, which takes in both players last position and then tries all positions and returns the position that will provide the highest utility assuming both other players stick. We assumed this scenario as the only time that T is set, is if both other non EA players have a significantly high stick index.

3 Experiments

In addition to the warm up tournament data, we ran internal testing of our agents against each other. We generally classified our strategies in complex and simple approaches.

To help us decide which agent to enter into the tournament, we tested agents against each other, and recorded the average scores over five games of length 10,000, though scores often did not vary between rounds. We attempted to more directly observe the effects of the agents' strategies on each other by having them play with a FiveOClockBot rather than with two other complex agents. This testing set-up also confirmed the tendency of high-level players to seemingly "overthink" the game and lose to low-leveler players.

In the following table, the row/column intersection of two bots holds the winner of their matchup, with FiveOClockBot as the third player. In parentheses next to the name of the winning agent we report the winning score.

4 Results

The results of our 210 10,000-round tournaments with other agents are too lengthy to report here. However, despite (barely) having a winning average of 8.01 and placing 12th out of 22 by score average, in the middle of the pack, our EA^{2}Beater bot only actually won 31% of our tournaments, or 66 out of 210.

To determine which agent to enter into the tournament, we tested ours against each other, and recorded the average scores over three games of length 10,000, though scores varied little between rounds. We attempted to distill the effects of the agents strategies on each other by having them play with a FiveOClock bot. This testing set-up exposed the tendency of high-level players to seemingly "overthink" the game and lose to low-leveler players.

Bot	EA^2	ModifiedConstant	EA^2 Beater
ACTR	FOCB(9.6)	FOCB(9.6)	ACTR(10)
CoOpp	EA^2(11)	FOCB(11)	FOCB(11)

Table 1: Bot match ups, with the winner at the row-column intersection in the table, and winning scores in parentheses (FOCB = FiveOClockBot)

Bot	ModConst	EA^2 Beater	CoOpp
EA^2	EA^2(11)	FOCB(10)	EASquared(11)

Table 2: Bot match ups continued

Bot	EA^2 Beater	Bot	CoOpp
ModConst	EA^2 Beater(10)	ACTR	ACTR(11)

Table 3: Further Matchups

Our analysis disappointingly showed that EA^{2}Beater failed to beat EA-Squared, and the four victories of FiveOClockBot showed how susceptible complex predictive strategies can be to a simple sticking strategy.

The poor win percentage of EA^{2}Beater in the preliminary class tournament, along with its poor performance in our internal tests, ultimately led us to choose ModifiedConstant as our final entry.

5 Conclusions

We were frustrated that EA-SquaredBeater failed to beat EA^2 in many cases, and even lost to FiveOClockBot. We assume that the turn by turn interaction between EA-SquaredBeater and EA^2 often did not play in the way we had predicted. We reached the suprising conclusion that **stick** seemed to be a dominant strategy against a variety of levels of strategies. With more time, we would consider reexamining the design of EA-Squared Beater, as we have faith in the potential of attempting to implement all the complexity of EA^2 in normal play, and invoking a specific sub-strategy if we encounter another EA^2-type agent. For example, we could try to design a strategy to coax the opponent into creating a setup in which it is likely that we can spring our EA trap. We could also experiment with more versatile approaches, such as improving the utility until the trap is triggered or to improve the positioning once EA-Squared has been shutoff, for example by aggressively offering a sandwich to the third player.

6 Contributions

Michael Robertson implemented the CoOpp and ModifiedConstant agents, as well as the sandwich recognition, offering, and accepting mechanisms. Fabio von Schelling Goldman implemented ACTR, EA-Squared and EA-Squared Beater. Both ran multiple experiments and performed debugging, however Michael Robertson ran the final experiment which provided our results. Fabio wrote the abstract, parts of the background and experiments, and the conclusion. Michael wrote the introduction, the other parts of the background and experiments as well as the results. Both authors tested and debugged the Bots. Both authors proof-read and made small edits throughout the entire paper.

References

Sykulski, A. M.; Chapman, A. C.; de Cote, E. M.; and Jennings, N. R. 2010. Ea2: The winning strategy for the inaugural lemonade stand game tournament. In *ECAI*.

Wunder, M.; Littman, M. L.; Kaisers, M.; and Yaros, J. R. 2010. A cognitive hierarchy model applied to the lemonade game.