# Probabilistic Iterated Prisoner's Dilemma

Zachary Azar
Austin Conner
Andrew Garcia
Nanhua Jin

September 29, 2011

In our initial attack on the problem of the probabilistic prisoner's dilemma, we realized that although the problem is a simple one—merely choosing to defect or cooperate at each round in a sequence—it is sufficiently non-intuitive that we wouldn't be able to simply guess an optimal strategy. Research into prior work in the subject indicated an already existing body of knowledge regarding the formulation of strategies without non-deterministic point totals. Our initial thought was that the introduction of probabilistic point awarding would not significantly affect the game, except in extreme cases. However, this is a fuzzy line of thought. If we were to distinguish between 'normal' and 'abnormal' probability distributions in our code, that would not only be a subtle distinction to make, but a completely arbitrary one. In fact, this was a general trend in our consideration, that any particular design choice we made would in fact just be an arbitrary guess based on a limited understand of the dynamics of the problem.

With this in mind, we decided we would not describe a specific strategy, but rather a general one which would define a search space in which to find an optimal solution. Unfortunately, it is very difficult to truly describe a general strategy for the game, due to the amount of information available to the player. So by necessity, we strove to choose a form which did not exclude any strategies that we could think of for the non-probabilistic prisoner's dilemma, and extended it to be sensitive to unknown distributions. To describe the

general strategy, we first present a general strategy for the non-probabilistic prisoner's dilemma. For any discrete probability distribution $f : \mathbb{N} \to \mathbb{R}$, there is a corresponding non-deterministic strategy defined in the following way: $f(1)$ – probability to cooperate, $f(2)$ – probability to defect, $f(n)$, $n \geq 3$ – probability to do what the opponent did $n-2$ turns ago. So for instance always cooperate is represented by a function $c$ with $c(1) = 1$, $c(x) = 0$ for all $x \neq 1$. A tit-for-tat strategy, with 10% chance to 'forgive' by cooperating is defined by a function $d$, $d(1) = 0.1$, $d(3) = 0.9$, $d(x) = 0$ for all $x \neq 1, 3$. Many strategies can be represented this way. Notice that there are many more possible strategies which we could have included but didnt because they were very likely to be poor. We could have defined the function to encode values to do, for example, the opposite of what the opponent did $n$ turns ago, what we did $n$ turns ago, the opposite of what we did $n$ turns ago, and so on. We made the decision to keep our description simple to focus on the strategies we were more confident to actually be good.

We have transformed the problem of the non-probabilistic prisoner's dilemma into something related to a constraint satisfaction problem. The variables are the values that a function in the space takes for each natural number. Here, satisfying the constraints means performing well in competition. The subtle difference with normal constrain satisfaction problems is there is no objective fitness heuristic for an element in the space, it only performs *relative to the other algorithms in the space*. It's a competition. We can't even define an objective heuristic as success against every possible opponent because almost all of the opponents in the space are not going to be worthwhile. If such a heuristic was defined, it would crushingly overweight algorithm performance against ineffectual opponents, a situation it is not likely to encounter. This consideration restricts our considerations of heuristic search to the ones that consider populations.

Initially, we were planning to do our optimal strategy search online between every round as another component of our program (discussed in Section 3) updated the estimated probability distributions. In that case the above description would have been adequate since the probability distribution information would be implicitly considered in the search. However, as we considered this approach further, we realized it would be too computationally intensive to be realistically used. We started considering ways we could

offload the computation offline. We will show that the presented distribution-insensitive general algorithm representation is still sufficient for our consideration even when we restrict the search to offline. The probability distribution sensitivity arises naturally in our construction of the final strategy, as presented in section 4.

# 1 Offline Simulator

Since we will not know the probability distributions of the possible point losses until the competition, it is difficult to pick a strategy beforehand because there is too much margin for error. This is where our offline simulator came in. Since we did not know the probability distributions, we tested several algorithms against each other under each of a set of probability distributions corresponding to the ones which always give a particular value (infinite at a single point and 0 elsewhere). To represent particular testing conditions, we used a set of three values, each of which correspond to the expected value of the probability distributions of both players cooperating, both players defecting, and our player cooperating when the other player defects (if we defect and the other player cooperates the probability distribution is constant, so it added no value when making all possible combinations).

At first, we started with four hard coded algorithms: always cooperate, always defect, tit-for-tat, and blindly cooperate 30% of the time and tit-for-tat 70% of the time. To represent an algorithm in the simulator, we used a list of values corresponding to the values the associated $f$ takes at the indices. In addition to the four hard coded algorithms, we would add an arbitrary number of algorithms that were completely random in the sense that the probability in each of its positions was randomly generated.

After the first few batches of results of the simulator, we found that three algorithms were dominating: always cooperate, always defect and tit-for-tat. As a result, instead of using those three and an arbitrary number of random functions, we used every possible combination of the three along with an arbitrary number of random functions to essentially evolve our algorithm population. Once we had this immense pool of algorithms, we had

each algorithm play every other algorithm 100 times for every possible combination of the expected values. For each expected value combination, the algorithm with the most points was chosen and entered into a 3 dimensional matrix.

The 3D matrix is represented as several nested associated lists in lisp. Once all of this data was generated, it was output into a file and included in the lisp code. We discuss how our online algorithm makes use of this information in section 4. The end result is an algorithm which adapts to the environment in which the competition takes place in real time.

## 2 History Data Structure

The history of the opponent is stored in a large association list that gives the decision history for each player and the known data for each choice combination. The opponent history association list consists of an association list with the player ID as the key and the actions that player did every turn in a list, with the last turn being the first element of the list. These values in the list are either a symbol `C` if the opponent cooperated or symbol `D` if the opponent defected. A second set of data contains our player's choice against the opponent represented in the same manner.

The probabilities association list consists of three different entries, one for both players cooperating, both players defecting or exactly one player defects. Each of those entries will contain a list of point values gained per turn. For example, if cooperating will lose between 0-20, the values in the both cooperating will be between 80 and 100. This carries on for both the other two entries.

## 3 Probability Distribution Estimation

Estimation of unknown probability distributions from a set of data generated by it is not a mathematically determined task in the sense that assumptions in addition to the data are required to make an estimation. For this application, a useful approximation must

not over fit the data, but must provide enough resolution to actually suggest the form of the distribution. We let $P_{c,c}$ be the probability distribution of points lost when both players cooperate, and $P_{c,d}$, $P_{d,d}$ be defined similarly. Note that $P_{d,c}$ is unnecessary since there is no uncertainty in the values alloted in this case (always 100). As competition goes on, data points are collected for each of these distributions.

For any particular distribution, say $P$, let $D = d_1 \leq d_2 \leq \cdots \leq d_N$ be the set of data collected so far, and $L, U$ be the lower and upper bounds of the distribution, respectively. The distribution is estimated by partitioning the area between the bounds of the distribution by the data points, and defining the value between two data points such that the resulting function forms a frequency table over the partition. In particular,

$$P(x) = \begin{cases} \frac{1}{2N(d_1 - L)} & : x \in [L, d_1] \\ \frac{1}{N(d_{i+1} - d_i)} & : x \in [d_i, d_{i+1}],\ i \in [1, N-1] \\ \frac{1}{2N(U - d_N)} & : x \in [d_N, U] \end{cases}$$

Then

$$\begin{aligned} \int_L^U P(x)\,dx &= \int_L^{d_1} \frac{1}{2N(d_1 - L)}\,dx + \sum_{i=1}^{N-1} \int_{d_i}^{d_{i+1}} \frac{1}{N(d_{i+1} - d_i)}\,dx + \int_{d_N}^{U} \frac{1}{2N(U - d_N)}\,dx \\ &= \frac{d_1 - L}{2N(d_1 - L)} + \sum_{i=1}^{N-1} \frac{d_{i+1} - d_i}{N(d_{i+1} - d_i)} + \frac{U - d_N}{2N(U - d_N)} = \\ &= \frac{1}{2N} + \sum_{i=1}^{N-1} \frac{1}{N} + \frac{1}{2N} = \frac{1}{2N} + (N-1)\frac{1}{N} + \frac{1}{2N} = 1 \end{aligned}$$

So $P$ is normalized, as expected. This definition achieves the property of in a sense being high resolution where there is a lot of data, and low resolution where there is not. As a result, it is highly suggestive of the form of the data since it is both precise and does not overfit where there is little to go on.

## 4 Evaluation During Competition

The decide function takes an average of the elements of our algorithm matrix weighted by the probability distributions estimated at a particular point in the competition. It looks

up its current opponent's decision history and from the generated function formulates a probability to cooperate. Absent history of the opponent toward the beginning of the rounds is assumed to be constant cooperate (an arbitrary choice). A random number generator is then used to determine the final decision.

We could have created a deterministic strategy by instead of using a random number generator in this last step, to simple choose cooperate if its probability was determined greater than some threshold value, intuitively $\frac{1}{2}$. Whether or not this would have been a better strategy overall, we did not have the time to test. We chose the non-deterministic strategy because in a sense it allowed more subtlety in the behavior of the strategy. For instance, if it was defecting over some threshold value, it would not break the cycle even if it calculated it should cooperate 49% of the time.

Our group was excited to try to implement a truly genetic algorithm, but due to time constraints we essentially opted to have a single generation simulator and get our hands dirty to choose the graduating algorithms. However, even with this approach, we were able to learn a lot about effective strategies, and hopefully implemented our ideas in a way which will ultimately benefit our player in competition.