

# Exploring the Social Iterated Prisoners' Dilemma with Grammatical Evolution and Tournament Clustering

Chad Brewbaker<sup>1</sup>

Iowa State University  
Department of Computer Science  
Ames IA 50010, USA  
Email: crb002@iastate.edu

**Abstract.** We analyze the Social Iterated Prisoners' Dilemma as described by Toby Ord and Alan Blair. A grammatical model for genetic programming is introduced which bounds quantifier depth, thus making analysis and evolution computationally tractable. Populations are evolved and the best few are put into a super population. This super population is then clustered based on agent behaviors. Results seem to indicate that social strategies are harder to evolve than non-social strategies, tit-for-tat strategies dominate, and that there is evolution of cooperation.

## 1 Introduction

The motivation for this paper was twofold. First we wanted to implement Blair and Ord's model of Social Iterated Prisoners' Dilemma (SIPD) in a way that bounded quantifier depth. Without this restriction the simulation becomes intractable as we will describe later. Quantifier depth was limited with the help of a high level grammar specification language, similar to BNF, that allowed us to quickly implement the search space. Secondly, we wanted a more useful way of visualizing the populations than picking a few agents ad-hoc for their novelty. To do this we use a behavior consensus clustering technique which we claim yields us a high level view of the evolved population.

## 2 The Language

We borrow the language description and behavior examples from Blair and Ord[1].

$D$  is a three place predicate which evaluates to TRUE if the agent in argument number one defected against the agent in argument number 2 at the time given in argument number 3.

$b$  is a one place function symbol that decrements its value by one.

$p$  represents the time value of the last round played.

$r$  represents the agent that is considering whether or not to defect.

$c$  represents the agent that  $r$  is considering defecting against.

Other letters  $i, j, k, \dots$  are variables representing agents in the game or time.  $\neg$ (NOT),  $\wedge$ (AND),  $\vee$ (OR), and  $\exists$ (EXISTS) have their normal meaning in first-order logic.

Here are some well known strategies for iterated prisoner's dilemma:

ALL-D:  $D(r, c, p) \wedge \neg D(r, c, p)$

Always defect.

Tit-For-Tat:  $D(c, r, p)$

Defect if my opponent defected against me last round.

Tit-For-Two-Tats:  $D(c, r, p) \wedge D(c, r, b(p))$

Defect if my opponent has defected against me the last two rounds.

Grim:  $\exists t D(c, r, t)$

Defect if there exists a time in the past where my opponent defected against me.

These strategies use social information:

Vigilante:  $\exists j D(c, j, p)$

Defect if my opponent defected against someone last round.

Vulture:  $\exists j (\exists t (D(j, c, p)) \wedge \neg \exists l (D(c, j, l)))$

Defect if someone has defected against my opponent, but my opponent has never retaliated against them.

It should be noted that Axelrod's "standard" payoff matrix was used in this experiment.

		<i>Player B</i>	
	<i>Cooperate</i>		<i>Defect</i>
<i>Player A</i>			
<i>Cooperate</i>	3		0
<i>Defect</i>	5		1

### 3 The quantifier depth problem

A problem with evolving computationally tractable agents for SIPD is quantifier depth. To evaluate each existential quantifier,  $\exists$ , we might have to loop over its whole domain. Thus, if we have an  $n$  nested quantifier  $\exists_{a_1, a_2, \dots, a_n}()$ , we have an  $n$  deep nested loop to evaluate. Since the time to evaluate a nested quantifier grows exponential with  $n$  we need to limit quantifier depth.

To accomplish this we impose a the following grammar to limit the quantifier depth at two in the agent domain, and at depth one in the time domain.

Those familiar with BNF grammars can skip to the grammar. For everyone else, here is a quick tutorial.

BNF grammars consist of a set of nonterminals on the left hand side, and for each nonterminal there is a set of expressions on the right separated by a bar.

NON\_TERMINAL: EXP0 | EXP1 | EXP2 | EXP3;

The expressions on the right can evaluate to a combination of symbols and other nonterminal expansions.

Here is a simple example with START, NAME, and ANIMAL as nonterminals.

START: NAME is my ANIMAL.;

NAME: Felix | Scratchy | Andre;

ANIMAL: cat | dog | ferret | hamster;

Possible expansions in this grammar are:

Felix is my cat.

Andre is my ferret.

Scratchy is my dog.

For a deeper understanding consult the “Red Dragon” [8] by Aho, Sethi, and Ulman.

## 4 The Grammar

This is the grammar used to describe the search space. Nonterminals S, SA, SB, SC, SAC, and SBC correspond to various levels of quantifier depth. Nonterminals PL, PLA, and PLB correspond to a choice of players. Nonterminals TIME and TIMEA correspond to time. The start state is S.

The start state.

1) S:  $S \wedge S \mid S \vee S \mid \neg S \mid D(\text{PL}, \text{PL}, \text{TIME}) \mid \exists_i (\text{SA}) \mid \exists_k (\text{SC})$ ;

One agent quantifier deep.

2) SA:  $SA \wedge SA \mid SA \vee SA \mid \neg SA \mid \exists_j (\text{SB}) \mid \exists_k (\text{SAC}) \mid D(\text{PLA}, \text{PLA}, \text{TIME})$  ;

Two agent quantifiers deep.

3) SB:  $SB \wedge SB \mid SB \vee SB \mid \neg SB \mid D(\text{PLB}, \text{PLB}, \text{TIME}) \mid \exists_k (\text{SBT})$ ;

One time quantifier deep.

4) SC:  $SC \wedge SC \mid SC \vee SC \mid \neg SC \mid D(\text{PL}, \text{PL}, \text{TIMEA}) \mid \exists a (\text{SAC})$ ;

One agent and one time quantifier deep.

5) SAC:  $SAC \wedge SAC \mid SAC \vee SAC \mid \neg SAC \mid D(\text{PLA}, \text{PLA}, \text{TIMEA}) \mid \exists b (\text{SBC})$ ;

Two agent quantifiers and one time quantifier deep.

6) SBC:  $SBC \wedge SBC \mid SBC \vee SBC \mid \neg SBC \mid D(\text{PLB}, \text{PLB}, \text{TIMEA})$ ;

The player or his opponent.

7) PL:  $r \mid c$ ;

The player, his opponent, or quantifier i.

8) PLA: PL | i;

The player, his opponent, quantifiers i and j.

9) PLB: PL | i | j;

The quantifier k, or a time expression.

10) TIMEA: k | TIME;

The previous round, or a decremented time expression.

11) TIME: p | b(TIME);

To implement this grammar we used Flex and Bison, the GNU versions of Lex and Yacc, to write a compiler that takes our BNF grammar and outputs a function in the C programming language. The functions ExistI(), ExistJ(), ExistK(), b(), and D() were then filled in to complete this part of the program.

The internal representation used for expanding this function is a stack of integers. Every time a state is entered an integer is popped from the stack. The integer is evaluated modulo the number of expressions for that state, and the expression whose number comes up is evaluated. For more information on this technique refer to O'neil and Ryan's work with Grammatical Evolution[3].

If the integer stack is emptied then all open paths evaluate to FALSE, r, or p depending on the context. As with the language, a TRUE evaluation for this function means the agent defects, and a FALSE evaluation means that the agent cooperates. The history has a memory of ten rounds, and the initial history vector was set to cooperate.

For added behavior complexity we also impose a finite state automaton. At each state an expression is evaluated with our grammar. We use a lookup table that takes the choice to defect or cooperate, and current state to decide which state is used next. Separate state information is kept for each neighbor, and every state has a different integer stack. This simulation used three states.

The social network is defined as an 25x25 toroidal grid where agents interact with eight neighbors: {N,NE,E,SE,S,SW,W,NW}. Social information is confined to competing agents and their common neighbors. The grid is initialized with one half taken up by the first player, and the second half taken by the other.

For a game agents randomly play from 10 to 50 rounds. At every round all agents evaluate their current strategy for each neighbor and decide whether to defect or cooperate. After each round payoffs are tallied, and state information is updated. After the specified number of rounds the players scores are tallied and in parallel every cell is taken over the neighboring agent with the highest score. If the cell has a higher score than its neighbors nothing happens. Ties are broken randomly. After 25 games the agent occupying the largest number of cells is declared the winner.

Populations of 25 members are run through 50 generations of evolution. All

agents play each other, and the fitness function is the number of agents beaten. Mutation is set at 1%, and the worst five agents are replaced with random new agents. 30 populations were evolved and the best 3 members of each population were harvested for a super population.

## 5 Tournament Clustering

Agents were played against each other to form a binary win/loss matrix. Entry  $A_{i,j} = 1$  if agent  $i$  can beat agent  $j$ . Entry  $A_{i,j} = 0$  if agent  $j$  beat agent  $i$ . Thus, each agent has a row that represents its win-loss record over the entire population. These rows were then clustered by Hamming, or bit flip distance. Elements like  $A_{i,i}$  have no meaning since agents were not played against themselves. Thus, these are excluded from the Hamming distance. The agent closest to the center, called centroids, for each cluster were then extracted for analysis.

The clustering algorithm used was K-Means.

8

- 1) Randomly pick  $K$  agents to be called centroids.
- 2) do  $i=1:n$ 
  - 2a) Group every centroid with the set of agents closest to that centroid.
  - 2b) Remove all centroid labelings.
  - 2c) do  $j=1:K$ 
    - 2c.1) Find the agent closest to the average of group  $j$  and call it a centroid.
- 3) Output centroid agents.

## 6 Analysis of the Super Population

A tournament graph is a directed complete graph. The directed edge from vertex  $i$  to vertex  $j$  implies that agent  $i$  beat agent  $j$  on the toroidal social network. Here is the tournament graph for the centroids of our super population with 5 clusters:

	A	B	C	D	E
A		0	0	0	1
B	1		0	0	0
C	1	1		1	0
D	1	1	0		0
E	0	1	1	1	

<i>AgentA</i>	<i>Expression</i>	
<i>State0</i> :	$\neg D(c, r, b(b(p))) \wedge (D(r, c, p) \wedge \exists_i (D(r, i, b(p)) \vee D(c, i, b(p))))$	
<i>State1</i> :	<i>TRUE</i>	
<i>State2</i> :	$\exists_{i,j} D(i, i, p) \wedge D(r, j, b(p))$	
<i>State</i>	<i>TrueMove</i>	<i>FalseMove</i>
0	<i>GOTO0</i>	<i>GOTO2</i>
1	<i>GOTO1</i>	<i>GOTO1</i>
2	<i>GOTO1</i>	<i>GOTO2</i>

Agent A cooperates the first round because its history vector is initialized to cooperate. The second move is a cooperate because the agent cooperated with everybody on the first round. It stays in state 2 cooperating. Thus, this agent is equivalent to always cooperate.

<i>AgentB</i>	<i>Expression</i>	
<i>State0</i> :	<i>FALSE</i>	
<i>State1</i> :	$D(r, c, p)$	
<i>State2</i> :	<i>UNUSED</i>	
<i>State</i>	<i>TrueMove</i>	<i>FalseMove</i>
0	<i>GOTO1</i>	<i>GOTO1</i>
1	<i>GOTO1</i>	<i>GOTO0</i>
2	<i>GOTO2</i>	<i>GOTO0</i>

Agent B cooperates on the first round, goes to state two where it asks if it defected last round on its opponent, and then goes back to state 0. Thus, agent B has the always cooperate behavior.

<i>AgentC</i>	<i>Expression</i>	
<i>State0</i> :	$D(c, r, b(p))$	
<i>State1</i> :	<i>UNUSED</i>	
<i>State2</i> :	<i>FALSE</i>	
<i>State</i>	<i>TrueMove</i>	<i>FalseMove</i>
0	<i>GOTO2</i>	<i>GOTO2</i>
1	<i>GOTO0</i>	<i>GOTO2</i>
2	<i>GOTO2</i>	<i>GOTO0</i>

Agent C starts out cooperating because its history vector is initialized with cooperates. In the second round it cooperates. After that it switches between tit-for-tat from two rounds ago and cooperate.

<i>AgentD</i>	<i>Expression</i>	
<i>State0</i> :	$D(c, r, p)$	
<i>State1</i> :	$\neg D(c, r, b(b(p)))$	
<i>State2</i> :	$\exists_{j,k} (\neg D(c, j, p) \vee \neg D(c, j, k)) \wedge D(j, r, k)$	

<i>State</i>	<i>TrueMove</i>	<i>FalseMove</i>
0	<i>GOTO1</i>	<i>GOTO1</i>
1	<i>GOTO2</i>	<i>GOTO1</i>
2	<i>GOTO0</i>	<i>GOTO1</i>

Agent D is the only centroid which exhibits social behavior. Because its history vector is initialized with cooperates its first move is to cooperate. The second move is to defect. After that the employs a complex mix of tit for tat, defection if it was with cooperated with 4 rounds ago, and punishing opponents who did not defect against another agents who were defecting on agent D.

<i>AgentE</i>	<i>Expression</i>
<i>State0</i>	<i>D(c, r, b(p))</i>
<i>State1</i>	<i>UNUSED</i>
<i>State2</i>	<i>UNUSED</i>

<i>State</i>	<i>TrueMove</i>	<i>FalseMove</i>
0	<i>GOTO0</i>	<i>GOTO0</i>
1	<i>GOTO2</i>	<i>GOTO0</i>
2	<i>GOTO2</i>	<i>GOTO1</i>

Agent E does a simple tit-for-tat from two rounds ago.

## 7 Analysis

After documenting the centroid behaviors we should update the tournament graph to account for games in which players always cooperate against each other. We mark them with T, for tie.

	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
<i>A</i>		<i>T</i>	<i>T</i>	0	<i>T</i>
<i>B</i>	<i>T</i>		<i>T</i>	0	<i>T</i>
<i>C</i>	<i>T</i>	<i>T</i>		1	<i>T</i>
<i>D</i>	1	1	0		0
<i>E</i>	<i>T</i>	<i>T</i>	<i>T</i>	1	

First, only one out of 5 centroids, agent D , made use of social information. It would appear that social information is hard to evolve. Agent D's behavior is clearly the most complex of the five centroids. It should also be noted that agent D is only mediocre. Its win-loss record is balanced.

Secondly, agents that seem best out of the centroid group are agents E and C. They either tie or beat all the other strategies. Agent C uses a cooperative tit-for-tat, and agent E uses the regular tit-for-tat.

Third, 4 out of the 5 centroids will always cooperate if their opponent does. This indicates that there is evolution of cooperation.

## 8 Future Work

The first idea that comes to mind is throwing more CPU time to approximate asymptotic behavior of the SIPD. The one day of compute time on an Intel Celeron 1000mhz laptop used in this study is probably not sufficient to get any meaningful asymptotic results. Also, our choice of grammar greatly effects the probability of a certain agent behaviors being expressed.[2] The current grammar needs to be combinatorially analyzed and a better grammar should be generated if any statistical anomalies are found.

Furthermore, we still have not quantified the value of social information. How does one detect the use of social information? How hard is it to evolve good social behaviors?

Finally, the author is refining the suite of unix utilities developed for this experiment for public release. The first version should be out by August 2004. For updates check the author's website: <http://www.public.iastate.edu/~crb002/>

## 9 Acknowledgment

Thanks to Dan Ashlock for our discussions on using grammars to limit quantifier depth, adding finite state machines for more interesting behavior, and his helpful editorial comments.

## References

1. Toby Ord and Alan Blair, *Exploitation and peacekeeping: introducing more sophisticated interactions to the iterated prisoner's dilemma*, Proceedings of the 2002 Congress on Evolutionary Computation, 1606–1611
2. Daniel Ashlock, *Complex Adaptive Systems Text*, in preparation with Springer.
3. Connor Ryan and Michel O'Neil, *Grammatical Evolution*, IEEE Transactions on Evolutionary Computation, Volume 5, Issue 4, Aug. 2001
4. Robert Axelrod and William D. Hamilton, *The Evolution of Cooperation*, Science, New Series, Volume 211, Issue 4489 (Mar.27,1981)
5. Robert Axelrod and Douglas Dion, *The Further Evolution of Cooperation*, Science, New Series, Volume 242, Issue 4884,(Dec. 9, 1988)
6. Chad Brewbaker and Daniel Wengerhoff, *Optimization and Analysis with Centroidal Voronoi Tessellations*, NSF REU 2001 technical paper, Iowa State University, [http://www.math.iastate.edu/reu/2001/voronoi\\_paper/voronoi\\_paper.html](http://www.math.iastate.edu/reu/2001/voronoi_paper/voronoi_paper.html)
7. Max Gunzburger, Qiang Du, V. Faber *Centroidal Voronoi tessellations: applications and algorithms*; SIAM Review 41, 1999, 637-676
8. Aho, Sethi, and Ulman, *Compilers: Principals, Techniques, and Tools*, Addison-Wesley 1986, ISBN 0201100886