Coevolution Of Finite Automata With Errors

Christos A. Ioannou [*][†]

Friday 19th July, 2013

Abstract

We use a genetic algorithm to simulate the evolution of error-prone finite automata in the repeated Prisoner's Dilemma game. In particular, the automata are subjected to implementation and perception errors. The computational experiments examine whether and how the distribution of outcomes and genotypes of the coevolved automata change with different levels of errors. We find that the complexity of the automata is decreasing in the probability of errors. Furthermore, the prevailing structures tend to exhibit low reciprocal cooperation and low tolerance to defections as the probability of errors increases. In addition, by varying the error level, the study identifies a threshold. Below the threshold, the prevailing structures are closed-loop (history-dependent) and diverse, which impedes any inferential projections on the superiority of a particular automaton. However, at and above the threshold, the prevailing structures converge to the open-loop (history-independent) automaton Always-Defect (ALLD). Finally, we find that perception errors are more detrimental than implementation errors to the fitness of the automata. These results show that the evolution of cooperative automata is considerably weaker than expected.

JEL Classification: C72, C80, C90

Keywords: Automata, Repeated Games, Prisoner's Dilemma, Bounded Rationality, Algorithms

[*]I am indebted to Aldo Rustichini and Ket Richter for their continuous support and invaluable discussions. I would also like to thank John H. Miller, Scott E. Page, Larry Blume, David Rahman, Itai Sher, Ichiro Obara, Snigdhansu Chatterjee, Ioannis Nompelis, Shi Qi, Miltos Makris, Maria Kyriacou, Kostas Biliouris, and Panayotis Mertikopoulos as well as the seminar participants at the University of Vienna, Santa Fe Institute, University of Minnesota, and Washington University for their suggestions. Finally, I am grateful to the editor, Uwe Cantner, and two anonymous referees for their thoughtful and detailed comments, which significantly improved the paper. Errors are mine.

[†]Mailing Address: Department of Economics, University of Southampton, Southampton, SO17 1BJ, United Kingdom. Email: c.ioannou@soton.ac.uk

1 Introduction

The repeated Prisoner's Dilemma (PD) stage game has become the theoretical gold standard for investigating social interactions. Its importance stems from defying commonsense reasoning and highlighting the omnipresent conflict of interests among unrelated agents. While Robert Axelrod (1984) has argued that reciprocal cooperation is likely to evolve when individuals interact repeatedly, in real life, there is a plethora of situations in which non-cooperative outcomes evolve − some in the presence of frequent encounters. Consider, for example, the world of sports. In wrestling, the prevalent practice is for wrestlers intentionally to lose unnaturally large amounts of weight so as to compete against lighter opponents (Steen and Brownell 1990). In doing so, wrestlers are, clearly, not at their top level of physical and athletic fitness; yet they often end up competing repeatedly against the same opponents who follow the same practice (Franchini, Brito and Artioli 2012).

In this paper, we argue that the evolution of cooperation in the PD game is not a stable and robust result, but a product of the assumption that agents are immune from committing errors. In real life, agents are not hyper-rational, but engage in actions that are constrained by the limitations of human nature and the surrounding environment. Thus, oftentimes, they suffer from a measure of uncertainty about both their colleagues' and their own actions. In large and complex firms, for example, divisional managers are often physically removed from each other and, consequently, are unable to observe each other's behavior directly. Uncertainty in this context takes the form of errors in the transmission of information. Moreover, decision makers are prone to errors in the implementation of their own actions. Due to these disturbances, they may occasionally draw incorrect inferences about their peers' actions.

In this work, our objective is to use a genetic algorithm to simulate an evolving, error-prone population that plays the repeated PD game. We then assess, with computational experiments that incorporate different levels of errors, whether and how the distribution of outcomes and structures (genotypes) in the population changes. According to the thought experiment, a group of agents is set to play the PD game. Each agent is required to submit a strategy that is implemented by a type of finite automaton called a *Moore machine* (Moore 1956). The automaton specifies actions contingent upon the opponent's reported actions. The agents play the PD game against each other and against their twin in a round-robin structure. The automata are subjected to implementation and perception errors (defined in Section 3). With the completion of all round-matches, the actual scores and genotypes of the automata become common knowledge. Based on this information, agents update their automata for the next generation.

Under the proposed framework, the incorporation of implementation and perception errors is sufficient to reduce cooperative outcomes. In addition, the analysis identifies some broad characteristics of the genotypes in the presence of errors. First, the complexity of the automata, defined as the number of accessible states, is decreasing in the probability of errors. Second, the prevailing structures tend to exhibit low reciprocal cooperation and low tolerance to the opponents' defections as the probability of errors increases. Furthermore, by varying the error level, the study identifies a threshold. At and above the threshold, the prevailing structures converge to the open-loop (history-independent) automaton Always-Defect (ALLD).[1] However, below the threshold, the prevailing structures are closed-loop (history-dependent) and diverse, which impedes any inferential projections on the superiority of a particular automaton. Finally, we find that perception errors are more detrimental than implementation errors to the fitness of the prevailing automata, which signifies the importance of limiting perception errors first, to avoid suboptimal outcomes in our strategic interactions.

Our study aims to elicit an understanding of the patterns of reasoning of agent-based behaviors in the presence of errors. Conventional game theory rests on the foundation of hyper-rational agents with full ability to select the most-preferred action. Yet the latter is rarely justified as an empirically realistic assumption. Rather, it is usually defended on methodological grounds as the appropriate theoretical framework to analyze behavior. On the contrary, the incorporation of errors in the proposed context is a viable alternative and, consequently, one that merits further investigation.

[1]Oscar Volij (2002) provides a theoretical framework which confirms that the only evolutionary stable strategy is ALLD when agents' preferences are lexicographic (first, according to the limit of the means criterion, and second, according to the complexity of the automaton).

2 Related Literature

This work builds primarily on the computational simulation literature. Robert Axelrod pioneered this area with the computational tournaments in which game-playing algorithms were submitted to determine the best strategy in the repeated PD game (Axelrod 1987). Tit-For-Tat (TFT) was the champion in Axelrod's celebrated computer tournaments. TFT is a strategy that starts off by cooperating and then imitates the opponent's most recent action. TFT is not an infallible strategy. On the contrary, it has some weaknesses and, at times, has been defeated by other strategies. (See also Nowak and Sigmund 1992 & 1993.) For example, TFT placed eight out of thirteen strategies in the computer tournament of Bendor, Kramer and Stout (1991). In their study, the authors re-evaluated the performance of reciprocating strategies, such as TFT, and identified alternative strategies that could sustain cooperation in an environment with random shocks. They constructed their computer tournament in a manner similar to that of Axelrod. The winning strategy was Nice-And-Forgiving (NAF), which differs in many ways from TFT. First, NAF is nice in the sense that it cooperates as long as the frequency of cooperation of the opponent is above some threshold. Second, NAF is forgiving in the sense that, although NAF retaliates if the opponent's cooperation falls below the threshold level, it reverts to full cooperation before its opponent does, as long as the opponent meets certain minimal levels of cooperation.

The study also relates to the large literature on optimization routines. The genetic algorithm (Holland 1975) is one of many search techniques developed for solving hard combinatorial optimization problems in large search spaces. Other optimization techniques include: Simulated Annealing (Kirkpatrick, Gelatt and Vecche 1983); Tabu Search (Glover and Laguna 1993); and Stochastic Hill Climbing. Axelrod (1987) was the first to model the evolutionary process of the repeated PD game with a genetic algorithm. Nevertheless, his study was restricted by his use of strategies contingent on the action profiles of only the last three periods, and by his use of a fixed environment composed of only eight strategies. Marks (1992) and Miller (1996) circumvented these restrictions by using a variable environment in which strategies co-evolved as the strategic population changed. In addition, both authors used automata to enable the definition of many theoretically important strategies (for example, strategies relying on counting or triggers) that could not be defined under Axelrod's framework. This approach is used in our formulation as well. Aumann (1981) was the first to suggest using finite automata as the carriers of agents' strategies for the study of decision-making with bounded rationality. The first application originated in the work of Neyman (1985), who investigated a finitely-repeated game model in which the pure strategies available to the agents were those that could be generated by automata utilizing no more than a certain number of states.

Additionally, several researchers have studied the effect of complexity on the set of equilibria in repeated games with finite automata. Abreu and Rubinstein (1988), for example, showed that if agents' preferences are increasing in repeated-game payoffs and decreasing in the complexity of the strategies employed, then the set of Nash-equilibrium payoffs that can occur is dramatically reduced from the folk-theorem result (Fudenberg and Maskin 1986).[2] Yet the authors indicated that a wide variety of payoffs remained consistent with equilibrium behavior in the presence of complexity costs, including the ALLD strategy. In their seminal work, Binmore and Samuelson (1992), motivated by the non-existence of an evolutionary stable strategy in the PD game (see Boyd and Lorberbaum 1987), proposed a modified evolutionary stable strategies' solution concept. Under this solution concept, the ALLD strategy ceases to persist in equilibrium. Foster and Young (1990), on the other hand, developed the concept of stochastic stability, which requires a population to be immune to persistent random mutations. Stochastic stability has been successfully applied by Kandori, Mailath and Rob (1993) in the analysis of symmetric 2×2 games; by Vega-Redondo (1997) in the analysis of competition among firms; and by Ben-Shoham, Serrano and Volij (2004) in the analysis of a housing problem. Bergin and Lipman (1996), however, pointed out a weakness of the concept of stochastic stability: the actions selected out of the recurrent classes depend on the rate of mutation.

The rest of the paper is organized as follows. In Section 3, we derive theoretical predictions using Markov chains. Section 4 presents the concept of a finite automaton as the carrier of an adaptive agent's strategy. In Section 5, we explain the methodology, and in Section 6, we focus

[2]Abreu and Rubinstein (1988) defined the complexity of a strategy as the size of the minimal automaton implementing it, while Banks and Sundaram (1990) argued that the traditional number-of-states measure of complexity of an automaton neglects some essential features, such as informational requirements at a state. They proposed, instead, a criterion of complexity that takes into account both the size (number of states) and transitional structure of a machine. Under this proposition, they proved that the resulting Nash equilibria of the machine-game are now trivial: the machines recommend actions in every period that are invariably stage-game Nash equilibria.

on the results of the evolutionary process while elaborating on characteristics of the automata. In Section 7, we discuss the properties of the prevailing automata, which we then compare to those of previous studies. Section 8 concludes and offers direction for future research.

3 Theoretical Preliminaries

We first provide a theoretical approach to derive predictions. For the analysis that follows, we restrict attention to the two most fundamental strategies: Always-Defect (ALLD) and Tit-For-Tat (TFT). Needless to say, there are many other possible strategies, and some play an important role. Nevertheless, we believe that the interplay of these two particular strategies captures an essential aspect of the evolutionary dynamics. Thus, we propose to investigate the logic of reciprocation and non-cooperation by analyzing the relationship of the most basic conditional strategy (do whatever the other player did) with an extreme unconditional strategy that always defects.[3]

3.1 Markov Chains

We consider a finite but infinitely-evolving population inhabited by agents using either TFT or ALLD to play the PD game. (We defer to Section 2.3 our discussion of the proportions of each strategy within the population.) Each period of play leads to an outcome j ($j = 1, 2, 3, 4$): (C,C), (C,D), (D,C), and (D,D), where $j = 1$ corresponds to (C,C) and so forth. Note that the first position denotes the action taken by agent i and the second position that of agent $-i$. The transition rules are labeled by quadruples (s_1, s_2, s_3, s_4) of zeros and ones. In this context, s_j is 1 if the strategy plays *Cooperate* and 0 if the strategy plays *Defect*, after outcome j is realized. For instance, $(1, 0, 1, 0)$ is the transition rule of TFT and $(0, 0, 0, 0)$ is the transition rule of ALLD. For convenience, these rules are labeled S^{TFT} and S^{ALLD}, respectively. Suppose that the strategies are subjected to implementation and perception errors. Let ϵ denote the

[3]We conjecture that similar results hold for other reciprocal strategies, such as Win-Stay, Lose-Shift. Win-Stay, Lose-Shift, also known as Pavlov, repeats the previous action if the resulting payoff has met the aspiration level, and changes otherwise.

probability of committing an implementation error, and δ denote the probability of committing a perception error.

Let a stochastic strategy have transition rules $\mathbf{p} = (p_1, p_2, p_3, p_4)$, where p_j is any number between 0 and 1 denoting the *probability of cooperating* after the corresponding outcome of the previous period. The space of all such rules is the four-dimensional unit cube; the corners are just the degenerate transition rules. A rule $\mathbf{p} = (p_1, p_2, p_3, p_4)$ that is matched against a rule $\mathbf{q} = (q_1, q_2, q_3, q_4)$ yields a Markov process where the transitions between the four possible states[4] are given by the Markov transition matrix[5]

$$\begin{pmatrix} p_1 \cdot q_1 & p_1 \cdot (1-q_1) & (1-p_1) \cdot q_1 & (1-p_1) \cdot (1-q_1) \\ p_2 \cdot q_3 & p_2 \cdot (1-q_3) & (1-p_2) \cdot q_3 & (1-p_2) \cdot (1-q_3) \\ p_3 \cdot q_2 & p_3 \cdot (1-q_2) & (1-p_3) \cdot q_2 & (1-p_3) \cdot (1-q_2) \\ p_4 \cdot q_4 & p_4 \cdot (1-q_4) & (1-p_4) \cdot q_4 & (1-p_4) \cdot (1-q_4) \end{pmatrix}.$$

If $\mathbf{p}$ and $\mathbf{q}$ are in the interior of the strategy cube, then all entries of this Markov transition matrix are strictly positive; hence, there exists a unique stationary distribution $\pi^{\mathbf{p}/\mathbf{q}} = (\pi_1, \pi_2, \pi_3, \pi_4)$ such that $p_j^{(n)}$ is the probability of being in state i in the n^{th} period, and it converges to

$$\pi_j = \lim_{n\to\infty} p_j^{(n)} \; for \; j = 1, 2, 3, 4.$$

It follows that the payoff for agent i using $\mathbf{p}$ against agent $-i$ using $\mathbf{q}$ is given by

$$A(\mathbf{p}, \mathbf{q}) = R \cdot \pi_1 + S \cdot \pi_2 + T \cdot \pi_3 + P \cdot \pi_4, \tag{1}$$

where the coefficients arise from the generic PD payoff matrix in Table 1. The letter R stands for *Reward*, S for *Sucker's* payoff, T for *Temptation*, and P for *Punishment.* The payoffs are ordered such that $T > R > P > S$ and satisfy $R > \frac{T+S}{2}$. Notice that both the π_j and the payoffs are independent of the initial condition (in other words, of the actions of the agents

[4]To be consistent with the conventional notation in Markov chains, the word "outcome" is replaced with the word "state."

[5]A Markov transition matrix is a matrix used to describe the transitions of a Markov chain. Each of its entries is a nonnegative real number representing a probability. This is not to be confused with a transition function in the context of finite automata, which maps every two-tuple of state and opponent's action to a state. (The concept of a finite automaton will be formally defined in Section 4.)

Table 1: Generic Prisoner's Dilemma Payoff Matrix

	Cooperate	Defect
Cooperate	R, R	S, T
Defect	T, S	P, P

Notes: The letter *R* stands for *Reward*, *S* for *Sucker's* payoff, *T* for *Temptation*, and *P* for *Punishment.*

in the first period). For any error level $\epsilon, \delta > 0$, the payoff obtained by a strategy using a transition rule S^i against a strategy with transition rule S^{-i} can be computed via (1).[6] The transition rules of TFT and ALLD in the presence of errors are shown in Table 2.

Consider, first, an ALLD strategist paired with another ALLD strategist. The Markov transition matrix is

$$\begin{pmatrix} \epsilon^2 & \epsilon\cdot(1-\epsilon) & \epsilon\cdot(1-\epsilon) & (1-\epsilon)^2 \\ \epsilon^2 & \epsilon\cdot(1-\epsilon) & \epsilon\cdot(1-\epsilon) & (1-\epsilon)^2 \\ \epsilon^2 & \epsilon\cdot(1-\epsilon) & \epsilon\cdot(1-\epsilon) & (1-\epsilon)^2 \\ \epsilon^2 & \epsilon\cdot(1-\epsilon) & \epsilon\cdot(1-\epsilon) & (1-\epsilon)^2 \end{pmatrix}.$$

The invariant distribution of such a pair is $\pi^{ALLD/ALLD} = (\epsilon^2, \epsilon\cdot(1-\epsilon), \epsilon\cdot(1-\epsilon), (1-\epsilon)^2)$. The payoff for agent i using ALLD against agent $-i$ also using ALLD is given by $A(ALLD, ALLD) = R\cdot\epsilon^2 + S\cdot\epsilon\cdot(1-\epsilon) + T\cdot\epsilon\cdot(1-\epsilon) + P\cdot(1-\epsilon)^2$. Note that the payoff is *increasing* in ϵ and does not depend on perception errors. Therefore, increasing the probability of implementation errors increases the payoff of an ALLD pair.

Consider, next, a TFT strategist paired with another TFT strategist. The Markov transition matrix in this case is

$$\begin{pmatrix} (1-\epsilon)^2 & \epsilon\cdot(1-\epsilon) & \epsilon\cdot(1-\epsilon) & \epsilon^2 \\ \epsilon\cdot(1-\epsilon) & \epsilon^2 & (1-\epsilon)^2 & \epsilon\cdot(1-\epsilon) \\ \epsilon\cdot(1-\epsilon) & (1-\epsilon)^2 & \epsilon^2 & \epsilon\cdot(1-\epsilon) \\ \epsilon^2 & \epsilon\cdot(1-\epsilon) & \epsilon\cdot(1-\epsilon) & (1-\epsilon)^2 \end{pmatrix}.$$

The invariant distribution of such a pair is $\pi^{TFT/TFT} = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$. Thus, the distribution depends neither on implementation errors nor on perception errors. Consequently, the payoff

[6]The limit value of the payoff for $\epsilon \to 0$ and $\delta \to 0$ cannot be computed, as the transition matrix is no longer irreducible. Therefore, the stationary distribution π is no longer uniquely defined.

Table 2: Transition Rules With Errors

Tit-For-Tat (TFT)	S^{TFT}: $(1-\delta-\epsilon(1-2\delta), \delta+\epsilon(1-2\delta), 1-\delta-\epsilon(1-2\delta), \delta+\epsilon(1-2\delta))$ The first entry corresponds to the probability of cooperating after a period in which both agents cooperated. Agent i will cooperate if he commits neither an implementation error (which occurs with probability $1-\epsilon$) nor a perception error (which occurs with probability $1-\delta$) or if he commits both a perception and an implementation error. Therefore, the probability of cooperating after a period in which both agents cooperated is $1-\delta-\epsilon(1-2\delta)$. Analogous arguments hold for the other entries.
Always-Defect (ALLD)	S^{ALLD}: $(\epsilon, \epsilon, \epsilon, \epsilon)$ The fourth entry corresponds to the probability of cooperating after a period in which both agents defected. Agent i will defect unless he commits an implementation error (which occurs with probability ϵ). Notice that errors in perception have no effect here, as the strategy Always-Defect does not depend on agent $-i$'s actions. Therefore, the probability of cooperating after a period in which both agents defected is ϵ. Analogous arguments hold for the other entries.

also does not depend on implementation and perception errors as long as the errors are strictly positive. In fact, the payoff for agent i using TFT against agent $-i$ also using TFT is fixed at $A(TFT, TFT) = R \cdot \frac{1}{4} + S \cdot \frac{1}{4} + T \cdot \frac{1}{4} + P \cdot \frac{1}{4} = \frac{1}{4}(R + S + T + P)$.

Finally, consider what happens when a TFT is paired with an ALLD (and vice versa). Let the TFT strategist be the row player and the ALLD strategist be the column player. The Markov transition matrix is

$$\begin{pmatrix} \epsilon \cdot (1-\delta-\epsilon+2\epsilon\delta) & 1-\delta+\epsilon \cdot (\epsilon+3\delta-2\epsilon\delta-2) & \epsilon \cdot (\delta-2\epsilon\delta+\epsilon) & \delta-\epsilon \cdot (3\delta-2\epsilon\delta+\epsilon-1) \\ \epsilon \cdot (\delta-2\epsilon\delta+\epsilon) & \delta-\epsilon \cdot (3\delta-2\epsilon\delta+\epsilon-1) & \epsilon \cdot (1-\delta-\epsilon+2\epsilon\delta) & 1-\delta+\epsilon \cdot (\epsilon+3\delta-2\epsilon\delta-2) \\ \epsilon \cdot (1-\delta-\epsilon+2\epsilon\delta) & 1-\delta+\epsilon \cdot (\epsilon+3\delta-2\epsilon\delta-2) & \epsilon \cdot (\delta-2\epsilon\delta+\epsilon) & \delta-\epsilon \cdot (3\delta-2\epsilon\delta+\epsilon-1) \\ \epsilon \cdot (\delta-2\epsilon\delta+\epsilon) & \delta-\epsilon \cdot (3\delta-2\epsilon\delta+\epsilon-1) & \epsilon \cdot (1-\delta-\epsilon+2\epsilon\delta) & 1-\delta+\epsilon \cdot (\epsilon+3\delta-2\epsilon\delta-2) \end{pmatrix}.$$

The invariant distribution when a TFT is matched with an ALLD is messy and not intuitive at all. More specifically, the eigenvector is $(-\frac{\frac{\delta\epsilon}{4}-\delta\epsilon^2+\delta\epsilon^3+\frac{\epsilon^2}{2}-\frac{\epsilon^3}{2}}{\frac{\delta}{4}+\frac{3\epsilon}{4}-\frac{5\delta\epsilon}{4}+2\delta\epsilon^2-\delta\epsilon^3-\epsilon^2+\frac{\epsilon^3}{2}-\frac{1}{4}}, -\frac{\frac{\delta}{4}+\frac{\epsilon}{2}-\delta\epsilon+\delta\epsilon^2-\frac{\epsilon^2}{2}}{\frac{\delta}{4}+\frac{\epsilon}{2}-\delta\epsilon+\delta\epsilon^2-\frac{\epsilon^2}{2}-\frac{1}{4}}, \frac{-\epsilon}{\epsilon-1}, 1)$. Note that this daunting eigenvector has been derived with an eigenvalue of 1 but does not reflect probabilities, as the entries are greater than 1 for $\epsilon, \delta > 0$. A simple trick to obtain the probabilities of the invariant distribution is to divide each entry by the sum of the entries. Note that, to obtain the stationary distribution of an ALLD matched with a TFT, we interchange the second and third entries of the invariant distribution of a TFT paired with an ALLD.

In summary, an ALLD versus an ALLD earns a payoff that is increasing in implementation errors, whereas a TFT versus another TFT earns a fixed payoff regardless of the level of implementation and perception errors as long as these are strictly positive. A natural next step is to determine how the levels of implementation and perception errors affect the payoff of an agent using TFT when paired with an agent using ALLD – and, likewise, the payoff of an agent using ALLD when paired with an agent using TFT.

3.2 ALLD vs TFT

The objective here is to determine how the levels of errors affect the payoffs of a pair of agents using ALLD and TFT. The idea is to first calculate the invariant distribution at each error level. Second, we plug the distribution into the payoff function (1), with the numerical values of the PD payoff matrix in Table 3, in order to derive the corresponding payoffs of an ALLD

Table 3: Prisoner's Dilemma Payoff Matrix

	Cooperate	Defect
Cooperate	3, 3	0, 5
Defect	5, 0	1, 1

strategist and a TFT strategist. This way, one can determine the error levels for which one strategist type prevails over the other type, as well as any monotonicity properties if those exist. This information is demonstrated graphically in Figure 1.

In Figure 1(a), we assume the same level for both implementation and perception errors. Both strategies start off at a payoff of 1 when the error levels are zero, but then monotonically increase at different rates. In particular, the rate of increase of ALLD is much faster relative to the rate of increase of TFT. The gap between the payoffs of the two strategies reaches a peak at an error level of 21%, and then the rate of increase of ALLD slows down, while the rate of increase of TFT takes off. At around 32%, the payoff of ALLD reverses direction and moves downward, while the payoff of TFT continues moving uphill. At 50%, the payoffs of the two strategies cross at 2.25, after which the payoff of TFT overtakes the payoff of ALLD. At 79%, the gap between the payoffs of TFT and ALLD reaches its peak. ALLD reverses direction again at around the 80% error level, while TFT attains the highest payoff at 88%, with 3.17. The corresponding payoff for ALLD is 2.36. Then, the payoff of TFT decreases.

In Figure 1(b), we fix perception errors at 0 and vary the level of implementation errors. Similar to 1(a), both payoffs increase in the beginning, with ALLD exhibiting a much faster rate of increase than that of TFT. The distance between the payoffs of the two strategies is maximized at the 25% level. The payoff of ALLD reaches its peak at an implementation error level of 40% with a payoff of 2.33. Right afterwards, the payoff of ALLD reverses direction, reaching a payoff close to zero as the probability of implementation errors approaches 1. The payoff of TFT, however, continues the upward trend, reaching the highest point as the probability of implementation errors approaches 1. At this level, TFT acts as an ALLD, and ALLD as an Always-Cooperate (ALLC).

In Figure 1(c), we fix implementation errors at 0 and vary the level of perception errors. Recall that ALLD is not affected by perception errors. Thus, only two states attain strictly

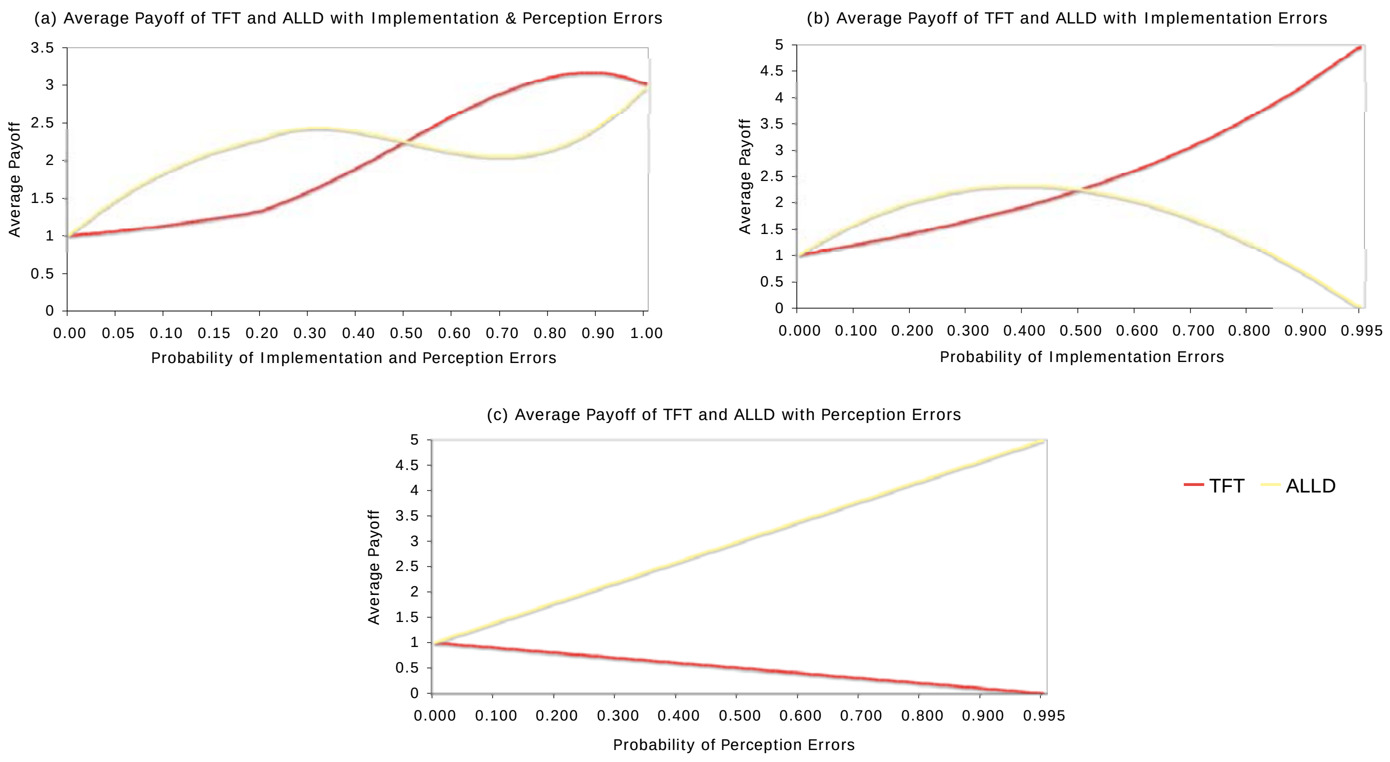

Figure 1: Average Payoff of TFT & ALLD

Notes: Graph 1(a) indicates the payoff of TFT and ALLD when the level of errors is the same. Graph 1(b) indicates the payoff of TFT and ALLD when the probability of implementation errors is varied, while perception errors are kept at 0. Graph 1(c) indicates the payoff of TFT and ALLD when the probability of perception errors is varied and implementation errors are kept at 0.

positive probability in the invariant distribution of TFT versus ALLD: (C,D), (D,D) – alternatively, (D,C) and (D,D) in the invariant distribution of ALLD versus TFT. Note that the payoffs of the two strategies move in opposite directions. As expected, the distance between the payoff of ALLD and TFT is maximized as the probability of perception errors approaches 1. At this level, ALLD always defects, while TFT continuously misperceives the action of ALLD as cooperation, instead of defection; consequently, TFT cooperates, thus earning the "sucker's" payoff.

3.3 Numerical Examples

Next, we provide numerical examples to facilitate a better understanding of the theoretical underpinnings. In these examples, we fix the population size at 30 agents. We strategically chose 30 to match the size of the population in the computational simulations conducted. (See Section 4.) The agents can use either TFT or ALLD and are paired in a round-robin structure; that is, all agents will be paired with one another in every possible combination. In addition, we assume that a strategy does play itself. Furthermore, to calculate the average payoff of each pair, we use the values of the payoff matrix in Table 3. Given this payoff structure, a pair of TFTs will earn 2.25 regardless of the error level as long as it is strictly positive. Finally, assume that x agents use TFT, and $30 - x$ agents use ALLD, where $x \in \mathbb{N}$ ($\mathbb{N} = \{1, 2, ..., 29\}$).

Example 1 ($\epsilon = \delta = 2\%$)
Assuming that implementation and perception errors are kept constant at 2%, first, we derive the payoff matrix of a population of ALLDs and TFTs. As indicated above, a TFT matched with a TFT will earn 2.25, while an ALLD matched with an ALLD will earn 1.06. A TFT paired with an ALLD will earn 1.02, while an ALLD paired with a TFT will earn 1.21. The payoffs are indicated in Table 4. The dynamics change once we assume different proportions within the population. In particular, given that x agents within the population use TFT, the average payoffs for an ALLD strategist and a TFT strategist are: $\mathcal{P}(ALLD) = \frac{((30-x)\times 1.06)+(x\times 1.21)}{30}$, and $\mathcal{P}(TFT) = \frac{((x)\times 2.25)+((30-x)\times 1.02)}{30}$, respectively.[7] The calculations indicate that as long as no more than one TFT enters the population, the ALLD strategists will earn a higher payoff than TFTs will. If two TFTs enter the population, then TFTs earn 1.10, and ALLDs earn 1.07. Furthermore, as the number of TFTs within the population increases, the gap in the payoffs widens; for example, if there are 15 TFTs in the population (and 15 ALLDs), then the ALLDs earn 1.13, and the TFTs earn 1.64, whereas if there are 29 TFTs (and 1 ALLD), then the TFTs earn 2.21 and the ALLD earns 1.20.

[7]The payoffs for TFT and ALLD for each possible value of x are provided in the Appendix.

Table 4: Payoff Matrix With $\epsilon = \delta = 2\%$

	TFT	ALLD
TFT	2.25, 2.25	1.02, 1.21
ALLD	1.21, 1.02	1.06, 1.06

Notes: The payoff matrix indicates the payoffs of a bi-morphic population consisting of ALLDs and TFTs. The payoffs reflect the invariant distributions for an error level fixed at 2% in both error types.

Example 2 ($\epsilon = \delta = 20\%$)

The payoff matrix corresponding to an implementation and perception error level of 20% is indicated in Table 5. A TFT that is paired with another TFT earns 2.25. An ALLD that is matched with another ALLD earns 1.56. A TFT paired with an ALLD earns 1.33, while an ALLD paired with a TFT earns 2.29. Analogous to the previous example, given that x agents within the population use TFT , the payoffs for an ALLD strategist and a TFT strategist are: $\mathcal{P}(ALLD) = \frac{((30-x)\times1.56)+(x\times2.29)}{30}$, and $\mathcal{P}(TFT) = \frac{((x)\times2.25)+((30-x)\times1.33)}{30}$, respectively. In this example, as long as there is at least one ALLD in the population, TFTs cannot bootstrap themselves. In other words, a deterministic selection process will enable ALLD to proliferate to the point where the entire population converges to a pure one implementing that strategy; consequently, TFTs will go extinct. If there are 29 ALLDs and one TFT, then, the payoffs are 1.58 and 1.36, respectively. If there are 29 TFTs and one ALLD, then, the payoffs are 2.22 and 2.27, respectively.

One may think that *Example 2* indicates a threshold effect above which TFTs cannot bootstrap themselves; that is, ALLD prevails. However, this is illusory.[8] Recall that, in Figure 1(a), the gap between the payoff of an ALLD and a TFT reaches its peak at an error level of 21%. When the gap between the payoffs of the two strategies is wide, it becomes very hard for the low earner to bootstrap himself; at some error levels when the gap is quite wide (such as the 20% error level in *Example 2*), it becomes impossible for the low earner to bootstrap himself given the population size of 30 agents. However, when the gap between the payoffs of the two strategies is small, then the low earner can, in fact, bootstrap himself as long as there is *sufficient* waiting time. (Recall *Example 1.*)

[8]We are grateful to an anonymous referee, who laid down the foundations to this analysis, and for pointing out the illusory nature of the process.

Table 5: Payoff Matrix With $\epsilon = \delta = 20\%$

	TFT	ALLD
TFT	2.25, 2.25	1.33, 2.29
ALLD	2.29, 1.33	1.56, 1.56

Notes: The payoff matrix indicates the payoffs of a bi-morphic population consisting of ALLDs and TFTs. The payoffs reflect the invariant distributions for an error level fixed at 20% in both error types.

4 Finite Automata

A finite automaton is a mathematical model of a system with discrete inputs and outputs. The system can be in any one of a finite number of internal configurations or "states." The state of the system summarizes the information concerning past inputs that is needed to determine the behavior of the system on subsequent inputs. The specific type of finite automaton used here is a Moore machine (Moore 1956). Let I denote the set of agents, A^i denote the set of i'*s* actions, A denote the cartesian product of the action spaces written as $A \equiv \overset{I}{\underset{i=1}{\times}} A^i$, and $g^i : A \rightarrow \Re$ denote the real-valued utility function of i. Thus, a *finite automaton* for an adaptive agent i in a repeated game of $G = (I, \{A^i\}_{i \in I}, \{g^i\}_{i \in I})$ is a four-tuple $(Q^i,\ q_0^i,\ f^i,\ \tau^i)$, where

- Q^i is a finite set of internal states;
- q_0^i is specified to be the initial state;
- $f^i : Q^i \rightarrow A^i$ is an output function that assigns an action to every state; and
- $\tau^i : Q^i \times A^{-i} \rightarrow Q^i$ is the transition function that assigns a state to every two-tuple of state and the other agent's action.[9]

[9]The transition function depends only on the present state and the other agent's action. This formalization fits the natural description of a strategy as agent i's plan of action in all possible circumstances that are consistent with agent i's plans. However, the notion of a game-theoretic strategy for agent i requires the specification of an action for every possible history, including those that are inconsistent with agent i's plan of action. To formulate the game-theoretic strategy, one would have to construct the transition function such that $\tau^i : Q^i \times A \rightarrow Q^i$ instead of $\tau^i : Q^i \times A^{-i} \rightarrow Q^i$.

In the first period, the state is q_0^i and the automaton chooses the action $f^i(q_0^i)$. If a^{-i} is the action chosen by the other agent in the first period, then the state of agent i's automaton changes to $\tau^i(q_0^i, a^{-i})$, and in the second period, agent i chooses the action dictated by f^i in that state. Then, the state changes again according to the transition function, given the other agent's action. Thus, whenever the automaton is in some state q, it chooses the action $f^i(q)$, while the transition function τ^i specifies the automaton's transition from q (to a state) in response to the action taken by the other agent.

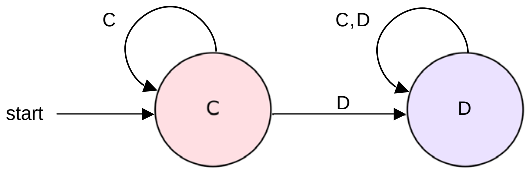

Figure 2: Grim-Trigger Automaton

Notes: The vertices denote the internal states of the automaton, and the arcs labeled with the action of the other agent indicate the transition to the states. The letter C stands for *Cooperate* and the letter D for *Defect*.

$Q^i = \{q_C, q_D\}$

$q_0^i = q_C$

$f^i(q_C) = C$ and $f^i(q_D) = D$

$\tau^i(q, a^{-i}) = \{^{q_C \quad (q, a^{-i}) = (q_C, C)}_{q_D \quad otherwise}$

For example, automaton (Q^i, q_0^i, f^i, τ^i) in Figure 2, carries out the Grim-Trigger strategy in the context of the PD game. The letter C stands for *Cooperate* and the letter D for *Defect.* The strategy chooses C, so long as both agents have chosen C in every period in the past, and chooses D otherwise. In the transition diagram, the vertices denote the states of the automaton, and the arcs labeled with the other agent's action indicate the transition to the states.

The study considers errors in the implementation of actions and errors in the perception of actions. It is, therefore, important to define formally implementation and perception errors.

Definition 1 *The automaton of agent i in the PD game commits an implementation error with probability ϵ, when, for any given state q, the automaton's output function returns the action $f^i(q)$ with probability $1-\epsilon$ and draws another action "$f^i(q)$" where $f^i(q) \neq$ "$f^i(q)$"otherwise.*[10]

[10]A more general definition would postulate that *the automaton of agent i commits an implementation error with probability ϵ, when, for any given state q, the automaton's output function returns the action $f^i(q)$ with*

That is, an implementation error level of ϵ indicates that with probability ϵ, the course of action dictated by the particular state of the automaton will be altered. For example, a cooperation dictated by the particular state will be implemented erroneously as a defection with probability ϵ. On the other hand, perception errors are defined as follows.

Definition 2 *The automaton of agent i in the PD game commits a perception error with probability δ, when, for any given opponent's action a^{-i}, the automaton inputs the opponent's action a^{-i} into the transition function with probability $1 - \delta$ and inputs the opponent's action "a^{-i}" into the transition function where $a^{-i} \neq$ "a^{-i}" otherwise.*

Thus, a perception error level of δ indicates that, with probability δ, an opponent's action is reported incorrectly, while with probability $1-\delta$, the opponent's action is perfectly transmitted.

Furthermore, the study considers automata that hold no more than eight internal states. The choice to keep the upper bound on the number of internal states at eight is a considered decision. First, such a bound is reasonable given complexity considerations. As Rubinstein (1986) indicates, agents seek to devise behavioral patterns that do not need to be constantly reassessed and that economize on the number of states needed to operate effectively in a given strategic environment. A more-complex plan of action is more likely to break down, is more difficult to learn, and may require more time to be executed. In fact, a number of studies (some with subjects in the laboratory) have pointed to the effectiveness of simple strategies over more-complex ones in a wide range of environments. (See Axelrod 1984; Rust, Miller and Palmer 1994; Selten, Mitzkewitz and Uhlich 1997.) Second, the choice of eight internal states allows for a sufficient variety of automata to emerge that incorporate a diverse array of characteristics.[11]

probability $1 - \epsilon$ and draws another action $a^i \in A^i \setminus f^i(q)$ randomly and uniformly otherwise. However, since the action space in the PD game consists of only two actions, the former definition suffices.

[11]To test the robustness of the results with respect to the size selection, computational experiments with automata that held 16 states were also conducted. The computations performed confirm that the results are robust.

5 Methodology

The search for an appropriate way to model agents' strategic choices has been a central topic in the study of game theory. The genetic algorithm is an attractive choice because it combines survival of the fittest with a structured information exchange that emulates some of the innovative flair of human search. Initially, the algorithm requires the natural parameter set of the optimization problem to be coded as a finite-length string over some finite alphabet. As a result, the algorithm is largely unconstrained by the limitations of other optimization methods that require continuity, the existence of derivatives, and uni-modality. More specifically, in order to perform an effective search for better structures, a genetic algorithm requires payoff values associated with the individual strings, in sharp contrast to calculus-based methods that require derivatives (calculated analytically or numerically). This characteristic makes the genetic algorithm a more canonical method than many other search schemes. As Goldberg (1989) indicates, a genetic algorithm searches from a rich database of points simultaneously, thus avoiding, to a large extent, suboptimal peaks in multi-modal search spaces. Finally, the genetic algorithm uses stochastic transition rules to guide the search. While randomized, the algorithm is no simple random walk. Instead, the genetic algorithm uses random choice as a tool to guide a search toward regions of the search space with expected improved performance.[12]

Darwinian mechanics are a continuation of the approach in which agents are neither fully rational nor knowledgeable enough to anticipate correctly the other agents' strategies. As Kandori, Mailath and Rob (1993) note, the process describes how agents adjust their plans of action over time as they learn from experience about the other agents' strategies. Yet the dynamics also reflect the limited ability of the agents to receive, decode and act upon the information they get in the course of the evolution. With the completion of all round-matches, not all agents change their strategies. The idea is that the agents' observations are imperfect; thus, changing one's strategy can potentially be costly. The existence of such uncertainties and adjustment costs, then, suggest the presence of inertia. In addition, agents are part of a system – they learn what constitutes a good strategy by observing what has worked well for other

[12]For a more-detailed discussion of genetic algorithms, see *An Introduction to Genetic Algorithms* (Mitchell 1998). No previous knowledge of genetic algorithms is required to understand the description of this genetic algorithm.

people. Hence, agents tend to emulate or imitate others' successful strategies. In turn, to the extent that there is substantial inertia present, only a small fraction of agents are changing their strategies simultaneously. In this case, those who do change their strategies are justified in making moderate changes. After all, they know that only a small segment of the population changes its behavior at any given point in time; hence, strategies that remain effective today are likely to remain effective for some time in the future.

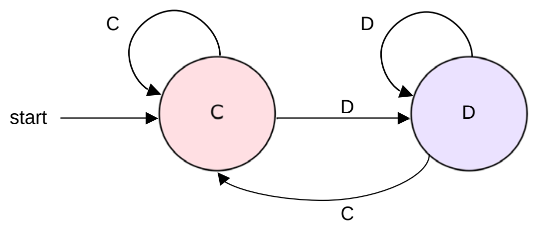

Figure 3: Tit-For-Tat Automaton

$$\underbrace{0}_{initial\ \ state}\quad \underbrace{1\ 0\ 1}_{state\ \ 0}\ \underbrace{0\ 0\ 1}_{state\ \ 1}\ \underbrace{0\ 0\ 0}_{state\ \ 2}\ \underbrace{0\ 0\ 0}_{state\ \ 3}\ \underbrace{0\ 0\ 0}_{state\ \ 4}\ \underbrace{0\ 0\ 0}_{state\ \ 5}\ \underbrace{0\ 0\ 0}_{state\ \ 6}\ \underbrace{0\ 0\ 0}_{state\ \ 7}$$

Initially, the natural parameter set of the optimization problem needs to be coded as a finite-length string. Each finite automaton here is, thus, represented by a string of 25 elements. The first element provides the starting state of the automaton. Eight three-element packets are then arrayed on the string, each packet representing an internal state of the automaton. The first bit within an internal state describes the action dictated by the particular state ($1 := cooperate, 0 := defect$). The next element gives the transition state if the opponent is observed to cooperate, and the final element gives the transition state if the opponent is observed to defect. Given that each string can utilize up to eight states, the scheme allows the definition of any automaton of eight or fewer states. For example, take the automaton that implements the Tit-For-Tat strategy in Figure 3. The automaton needs to remember only the opponent's last action and, thus, utilizes only two states; the last six states are redundant, as illustrated in the coding.[13]

[13]Given this layout, there are 2^{59} possible structures. Yet, the total number of unique structures is much less since some of the automata are isomorphic. For example, every two-state automaton, such as TFT, can be renamed in any of $(P_2^8)(2^{42})$ ways.

```
Specify error-level
Fix max-periods = 200

Create initial population: 30 agents (seed randomly)
Initiate round-robin tournament

For t = 1 to 500 do

    For all agent-pairs do
        For p = 1 to max-periods do
            Award utils to each agent based on the PD matrix
        End loop

        Output performance score
    End loop

    Apply subroutine for the offspring-population-creation
    Store agent results

End loop
```

Figure 4: Pseudocode of the Main Program

The mechanics of the genetic algorithm involve copying strings and altering states through the operators of selection and mutation. Each generation starts with a given population called the *parent population*. A new population of the same size, called the *offspring population*, is then constructed. In our formulation, the algorithm operates with a population of automata. Initially, a population of thirty automata is chosen at random. Then, each automaton is tested against the environment. Thus, each automaton aggregates a raw score based on the payoffs illustrated in Table 3. The pseudocode of the main program is summarized in Figure 4.

The offspring population is constructed from the parent population by selecting the automata that aggregated the top twenty scores. In addition, ten new structures are created via a process of selection and mutation. The process requires the draw of ten pairs of automata from the parent population (with the probabilities biased by their scores) and the selection of the better performer from each pair. Then, these ten automata undergo a process of mutation.[14] Mutation occurs when an element at a random location on the selected string changes

[14]Bergin and Lipman (1996) indicate that mutation rates represent either experimentation or computational errors. In our context, mutation rates represent experimentation. In addition, we assume that the mutation

value. Each element on the string is subjected to a 4% independent chance of mutation, which implies an expectation of one element-mutation per string.[15] The population is iterated for 500 generations. The subroutine of the creation of the offspring population is summarized in Figure 5.

```
Sort agents based on performance score

Copy top 20 agents to offspring-population

Select 10 agent-pairs via probabilities biased by performance scores

For each of 10 pairs do
    Create new agent as a copy of the winner of the pair's match
    Mutate new agent by switching one element at random

End loop
```

Figure 5: Subroutine of the Offspring-Population-Creation

6 Results

We conducted five main computational experiments, in order to assess how the distribution of outcomes and genotypes in the population changes with different levels of errors. In particular, the automata were subjected to a constant independent chance of implementation and perception errors of 4%, 3%, 2%, 1% and 0%, respectively. The results were averaged over 30 simulations conducted for each computational treatment. In addition, in the absence of a theoretical background on the functional form of the model, we used non-parametric methods to carry out the estimation. More specifically, we used local polynomial regression to fit the

rate is constant across strings, across agents and over time.

[15]A higher mutation rate increases the variation in the population; as a result, in such an environment, the automaton ALLD is hugely favored. The computational simulations performed with higher rates of mutation (8% and 12% independent chance of mutation per element, which implies an expectation of 2 and 3 elements-mutation per string, respectively) confirm this claim. On the other hand, the choice of a 4% independent chance of mutation is a conservative one that allows forms of innovation to appear in the structure of the automata while controlling for the variation in the population.

dependent variables over the course of the evolution. The dependent variables were found to exhibit non-stationary behavior that is attributed to the selection dynamics of the genetic algorithm. All the smoothed curves presented in Figure 6 utilize first-order polynomial functions, which are found in the literature to be quite effective in balancing the bias-variance trade-off. In addition, we used the Epanechnikov-Kernel weighting function. The bandwidth was chosen via the (data-based) rule-of-thumb bandwidth. The latter was preferred in the absence of any periodical patterns of the way agents behave across generations.

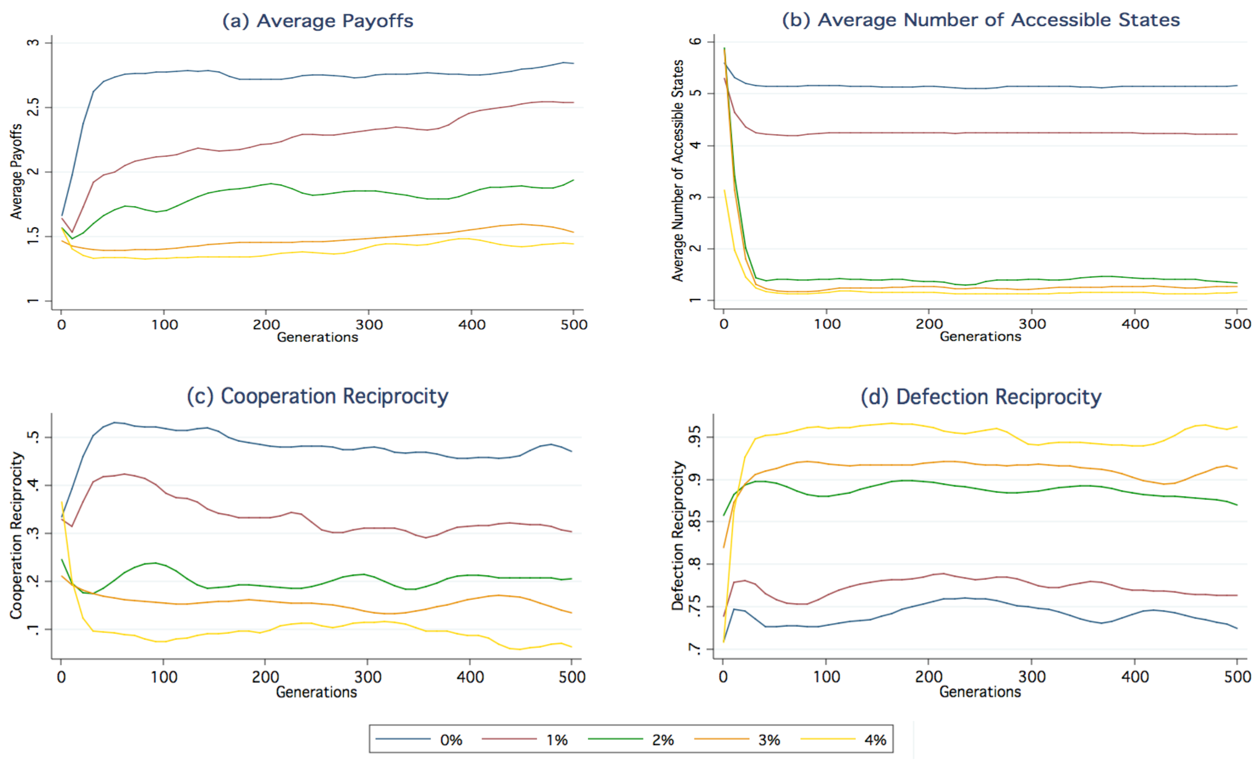

Figure 6: First-Order Polynomial Regressions

Notes: The simulations assess how the characteristics of genotypes change with different levels of errors. The smoothed curves utilize first-order polynomial functions with the Epanechnikov-Kernel weighting function.

6.1 Average Payoffs

In Figure 6(a), we use local polynomial regression to fit the average payoffs[16] (fitness) over the course of the evolution. The payoffs are found to exhibit non-stationarity that can be attributed to the dependence of the generational selection. In the early generations, the agents tend to use automata that defect continuously. The reason is that, at the start of the evolution, the automata are generated at random. In such an environment, the best strategy is always to defect. As a few generations elapse, though, automata in the less-error-prone treatments achieve consistent cooperation, which allows the payoffs to move higher. The paired differences tests establish that the means of the treatments are statistically different at a 1% level of significance. Thus, the incorporation of implementation and perception errors is sufficient to alter the evolution of cooperative outcomes.

6.2 Automaton Characteristics

We also ascertain broad characteristics of the structures that work "reasonably well" under different treatments. A *characteristic* is a property of the automata whose presence can be objectively determined by an examination of the corresponding graph. Note that the characteristics listed are also indicators of strategic ideas underlying the automata. The first summary measure is the size of the automaton, which is measured by its number of accessible states. A *state is accessible* if, given the automaton's starting state, there is some possible combination of actions that will result in a transition in that state.[17] In Figure 6(b), we fit the average number of accessible states per generation of the five treatments. Over the course of the evolution, the average number of accessible states is consistently less in the more-error-prone

[16]The average payoff of a given generation t is calculated as the sample average of the payoffs of the thirty members of the population and over the thirty simulations conducted for each treatment.

[17]This definition does not mandate that the opponent is part of the current population. Thus, we circumvent the problem that some states may not be reached, given the set of opponents. We clarify further our definition with an example. Suppose that, in some generation, t, the population consists of one-state automata that Always-Cooperate (ALLC) and one automaton that cooperates unless the opponent defects for seven consecutive periods, after which the automaton defects forever. Given our definition, such an automaton has eight accessible states. The ALLCs have one accessible state. We thank an anonymous referee for bringing to our attention the necessity of this clarification.

environments. This suggests that, under more-error-prone treatments, the average number of accessible states drops. Thus, if we assume that the average number of accessible states is a good measure of automaton-complexity, then a possible conclusion is that strategic simplification is advantageous in the presence of errors. The first descriptive characteristic of the automata is summarized below.

CHARACTERISTIC 1: *The average number of accessible states of the automata is decreasing in the probability of errors.*

Other summary measures that shed light on automaton composition are cooperation reciprocity and defection reciprocity. On the one hand, *cooperation reciprocity* is the proportion of accessible states that respond to an opponent's cooperation with cooperation. On the other hand, $defection\ reciprocity$ is the proportion of accessible states that respond to an opponent's defection with defection. The average cooperation reciprocity of the five treatments is fit into a local polynomial regression in Figure 6(c). As the likelihood of errors increases, the automata tend to reduce their cooperation reciprocity. This observation also accounts for the low payoffs in Figure 6(a). An alternative perspective to consider is the proportion of accessible states that respond to an opponent's cooperation with defection; this proportion is given by the expression $1 -$ *cooperation reciprocity*. The latter expression can be used as a proxy for the degree of "sneakiness" of the automata. Thus, automata with relatively low cooperation reciprocity are also relatively more sneaky, in the sense that these automata incorporate states that shoot for the "temptation" payoff more often.[18] Such exploitation can be camouflaged in the presence of errors, but not in their absence. The second descriptive characteristic is summarized next.

CHARACTERISTIC 2: *Cooperation reciprocity of the automata is decreasing in the probability of errors, or, alternatively, the degree of "sneakiness" is increasing in the probability of errors.*

Defection reciprocity of the five treatments is fit into a local polynomial regression in Figure 6(d). The pattern deduced is that defection is not tolerated in the error-prone environments. It is noteworthy that automata, in general, are more likely to reciprocate an opponent's defection with defection than to cooperate after the opponent cooperates. On the other hand, the proportion of accessible states that respond to an opponent's defection with cooperation is

[18]The "temptation" payoff is the payoff upon unilaterally defecting in the PD game.

$1 - defection\ reciprocity$. This expression signifies the degree of forgiveness of the automata. For example, a sizable $1 - defection\ reciprocity$ indicates that the automata, on average, incorporate quite a few states that respond to an opponent's defection with cooperation. Yet the computational experiments of Figure 6(d) indicate otherwise. The third descriptive characteristic is summarized below.

CHARACTERISTIC 3: *Defection reciprocity of the automata is increasing in the probability of errors, or, alternatively, the degree of forgiveness is decreasing in the probability of errors.*

6.3 Prevailing Automata

In this section, we examine the automata that prevailed in the computational treatments. Thus far, our analysis has described traits of the structures that survived over the course of the evolution. The results that have been highlighted rest on reasonable levels of error. Here, we carry out the analysis with abnormally high levels of error to get a spherical view of the impact of errors on the coevolved automata. ALLD and ALLC have been the clear winners of the simulations, but for substantially different levels of error. ALLD and ALLC are open-loop automata; that is, the actions taken at any time do not depend on the actions of the opponent. ALLD has been the winner in the treatments with reasonable error levels and ALLC in the treatments with very high error levels. This is *not* paradoxical. It is important to recognize that, at the high levels of error, ALLC essentially acts as if it were an ALLD in an environment with low levels of error.

In Figure 7,[19] we provide the proportion of wins of ALLD, ALLC, and Other automata for different error levels. This way, we can determine the automaton that dominates, if such exists, in each computational treatment. Recall that, for each computational treatment, we conducted 30 simulations. Thus, the proportion reflects the number of wins out of the 30 simulations. The structures that prevail in the 1% and 0% treatments are diverse. This result thwarts any possible attempt to discern a particular behavioral pattern that fares well in these specific treatments. Yet it is noteworthy that the diverse array of automata that prevail in the 1% and 0% treatments are all closed-loop (history-dependent). On the contrary, at the

[19]Table 7 in the Appendix provides the frequency of wins of ALLD, ALLC, and Other automata for each computational treatment.

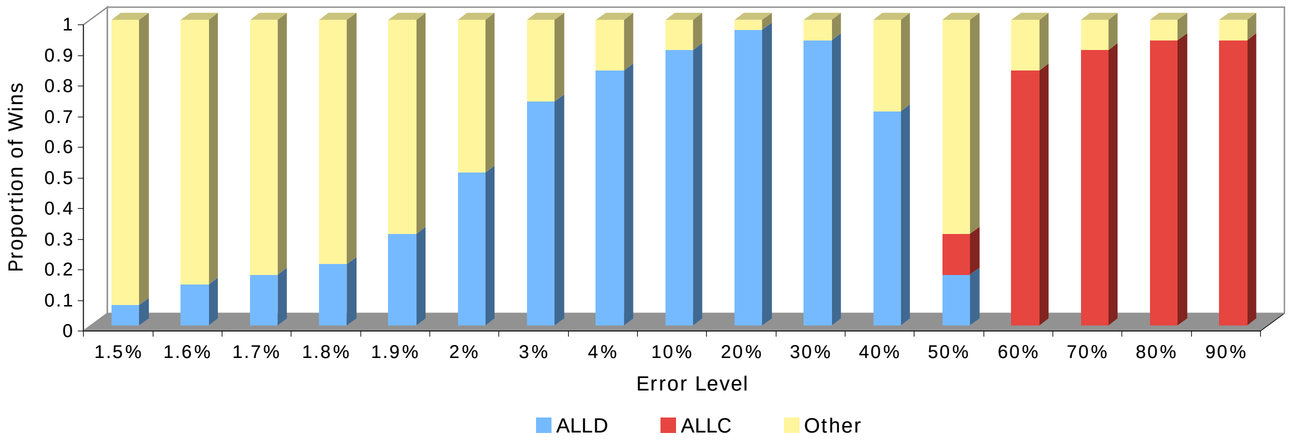

Figure 7: Prevailing Automata

Notes: On the vertical axis, we indicate the proportion of wins for ALLD, ALLC, and Other automata out of the 30 simulations conducted. The horizontal axis indicates the error levels.

2% error level, the prevailing structure is the open-loop ALLD. This error level needs to be highlighted given that, at this mark, ALLD prevails in 15 of the 30 simulations conducted; that is, ALLD attains a proportion of wins of 0.5. As the error level increases, the proportion of wins of ALLD increases and approaches 1 at the 20% error level. ALLD continues to dominate after the 20% error level, but the proportion steadily decreases until the error level of 50%. At the 50% error level, ALLC prevails in some simulations and ALLD in others. It might seem paradoxical that, at this mark, the prevailing automata cover both extremes of the spectrum. However, this is only an illusion, as at the 50% error level, both automata share the same behavior. From then on, ALLC is dominant. Yet, in practice, ALLC acts as an ALLD in low error levels. In summary, the effect of different error levels on the structure of the automata points towards the existence of a threshold error level at 2%; at and above the threshold, the prevailing structures converge to the open-loop automaton ALLD, whereas below the threshold, the prevailing structures are closed-loop and diverse.

6.4 Separation of Errors

Do both types of errors affect the payoffs of the prevailing automata in the same way? Moreover, is the threshold maintained under each type of error? These are the questions that we seek to answer here. Thus, we separate the two types of errors and conduct the analysis for each type independently at 4%, 3%, 2%, and 1%. Similar to the previous computational treatments, our analysis focuses on the same error levels as before. We do so for reasons of comparability and because we place greater emphasis on reasonable levels of errors. The computational experiments are shown in Figure 8. In particular, in Figure 8(a) we use first-order local polynomial regressions to fit the average payoffs over the course of the evolution for a constant and independent chance of perception errors. In Figure 8(b), we run first-order local polynomial regression to fit the average payoffs for a constant and independent chance of implementation errors.

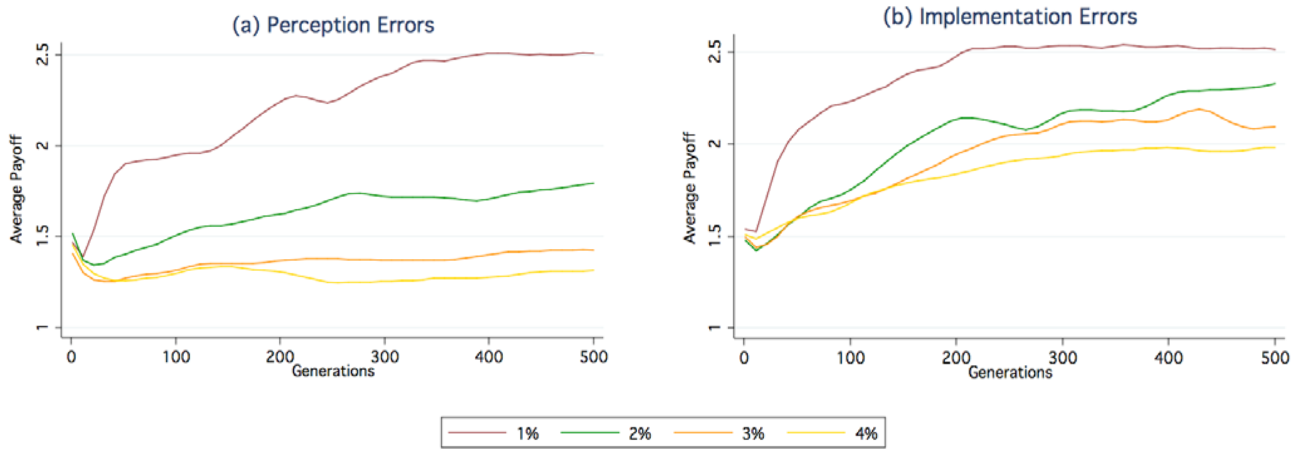

Figure 8: Separation of Errors

Notes: The simulations assess how the payoffs (fitness) of the automata change with (a) perception errors or (b) implementation errors. The smoothed curves utilize first-order polynomial functions with the Epanechnikov-Kernel weighting function.

As the likelihood of perception errors increases, the average payoffs of the prevailing automata decrease. The same result holds in the case of implementation errors. It is noteworthy that, in general, the average payoffs are higher in the presence of implementation errors than they are in the presence of perception errors. Thus, perception errors are more harmful than implementation errors to the fitness of the prevailing automata. This is made starkly evident if we compare the payoffs of each perception-error level (Figure 8(a)) with the payoffs for the corresponding error level when both types of errors exist (Figure 6(a)). Indeed, the average payoffs in Figure 8(a) are lower than those when both types of errors persist for the same error level. By contrast, the average payoffs in Figure 8(b) are higher than those when both types of errors persist for the same error level. (This interesting finding is discussed in the next section.)

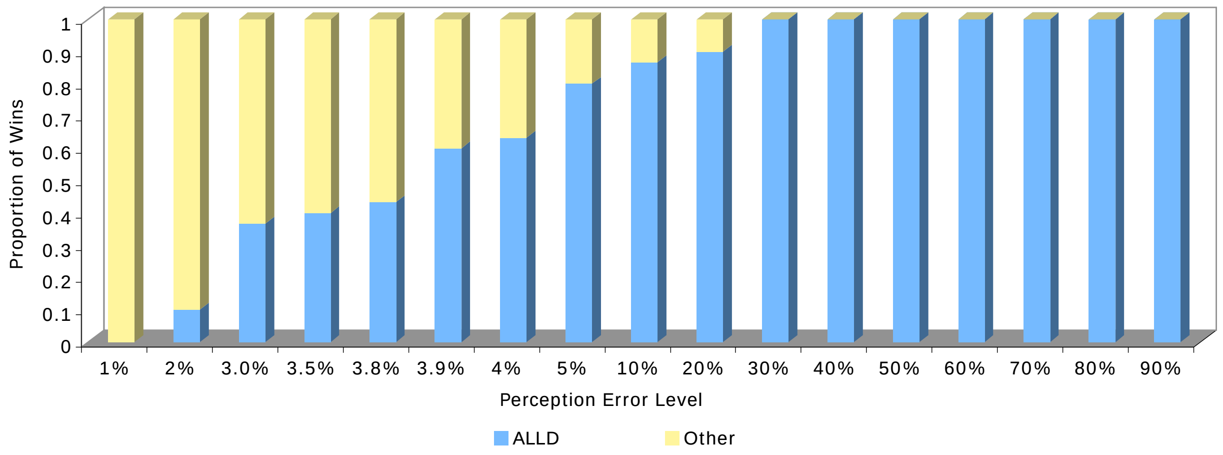

Figure 9: Prevailing Automata with Perception Errors

Notes: On the vertical axis, we indicate the proportion of wins for ALLD and Other automata out of the 30 simulations conducted. The horizontal axis indicates the perception-error levels.

Next, we provide the proportion of wins for ALLD, ALLC and Other automata for different levels of perception errors and implementation errors.[20] The proportion reflects the number of wins out of the 30 simulations conducted. In Figure 9, we provide the proportion of wins of ALLD and Other automata for different levels of perception errors. At the 1% and 2% error levels, no strategy is the clear winner. At 3%, the ALLD automaton begins to dominate. At 3.8%, the proportion of wins for ALLD is 0.43 and at 3.9% jumps to 0.6. At this level, ALLD wins 18 of the 30 simulations. We can safely assume that the proportion of 0.5 is attained somewhere between 3.8% and 3.9%. Note that the analogous proportion is attained at the 2% error level when both types of errors are allowed. The proportion of wins for ALLD increases as the likelihood of committing perception errors increases. At the 30% error level, ALLD reaches

[20]Table 8 in the Appendix provides the frequencies for each computational treatment for the perception errors, and Table 9 provides the frequencies for each computational treatment for the implementation errors.

a proportion of 1 (30 wins in 30 simulations), which is retained throughout the higher error levels. The reason that ALLD dominates in the presence of only perception errors is that the ALLD exploits errors committed by cooperating automata that misperceive its defections for cooperative behavior and, thus, attempt to cooperate with ALLD. Consequently, ALLD earns the "temptation" payoff, while the cooperative automata earn the "sucker's" payoff. This way, ALLDs proliferate and end up playing against each other.

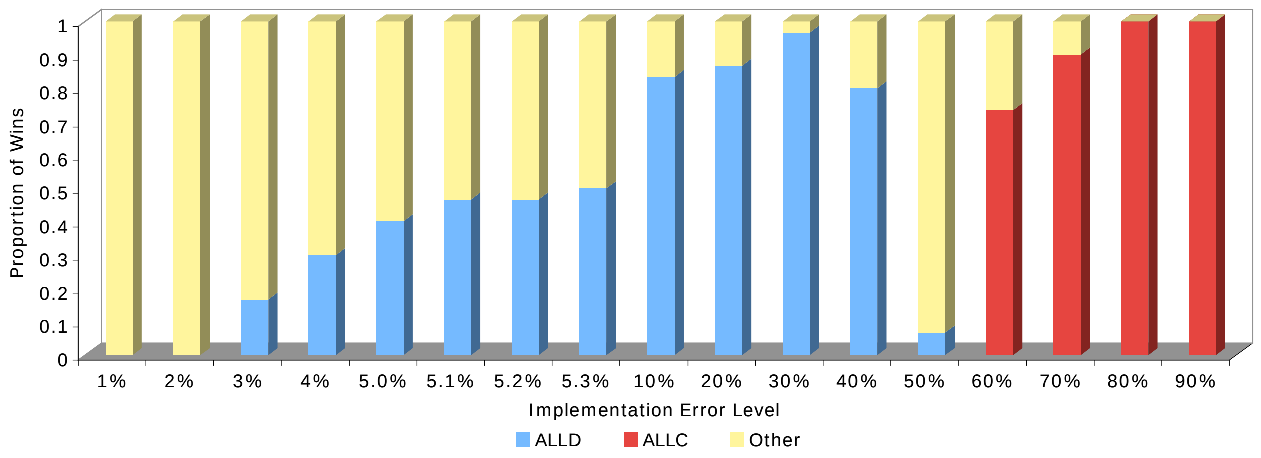

Figure 10: Prevailing Automata with Implementation Errors

Notes: On the vertical axis, we indicate the proportion of wins for ALLD, ALLC and Other automata out of the 30 simulations conducted. The horizontal axis indicates the implementation-error levels.

In Figure 10, we show the proportion of wins for ALLD, ALLC and Other automata for different levels of implementation errors. In sharp contrast to perception errors, implementation errors affect all automata, including ALLD. Thus, our results here are somewhat different from the results with only perception errors. At the 1% and 2% error levels, no strategy has an edge. At 3%, ALLD has a proportion of wins of 0.17, which jumps to 0.5 at 5.3%. At this point, the dominance of ALLD is clear. ALLD continues to increase its proportion of wins as the likelihood of committing implementation errors increases, until the 50% mark. Beyond this point, ALLC dominates. It is important to realize that, beyond this high error level, ALLC behaves as an ALLD in an environment with low error levels. This result is analogous to the one discussed in the previous section.

To facilitate comparison across the different environments, let us reiterate that the proportion of wins of 0.5 is attained at the 2% level when both types of errors are allowed, around the 3.9% level when only perception errors are allowed, and at the 5.3% level when only implementation errors are allowed. We conjecture that the lower error level when both types of errors coexist can be attributed to the additional randomness that two sources of uncertainty bring to the model, as opposed to when the perception and implementation errors act in isolation. Furthermore, it is not surprising that the model with only perception errors reaches the 0.5 proportion at a lower error level than the model with only implementation errors; ALLD is hugely favored with perception errors as it is immune to them, which is not the case with implementation errors.

7 Discussion

TFT was the winner in the *in silico* (with programmed automata) tournaments of Robert Axelrod (1984). The performance of TFT led Axelrod to identify some basic attributes that were necessary for the emergence and survival of cooperation. These were: (i) an avoidance of unnecessary conflict by cooperating as long as the other agent does; (ii) provocation in the face of an uncalled-for defection by the other; (iii) forgiveness after responding to a provocation; and (iv) clarity of behavior so that the other agent can adapt to your pattern of action. Unlike Axelrod, the study by Bendor, Kramer and Stout (1991) incorporated random shocks into a computer tournament. The winning strategy in that tournament was Nice-And-Forgiving (NAF), which differs in many ways from TFT. First, NAF is nice in the sense that it cooperates as long as the frequency of the opponent's cooperation is above some threshold. Second, NAF is forgiving (generous) in the sense that, although NAF will retaliate if the opponent's cooperation falls below the threshold level of cooperation, it will revert to full cooperation before its opponent does, as long as the opponent meets certain minimal levels of cooperation.

Yet, the success of NAF is not a robust result but is limited to the particular environment. As Bendor, Kramer and Stout (1991) note, the generosity of NAF creates a risk: other strategies may exploit NAF's willingness to give more than it receives. In other words, NAF can be suckered by a nasty strategy that is disinterested in joint gains.[21] As Axelrod and Dion (1988) aptly

[21]Due to its generosity, NAF lost in its pairwise play with every one of its opponents. In contrast to NAF's pattern, VIGILANT, the strategy that placed dead last in the tournament, beat every one of its partners in

note, in the presence of large amounts of errors, there is a trade-off: unnecessary conflict can be avoided by generosity, but generosity invites exploitation. Thus, strategies that are unilaterally generous will not fare well in every environment. Another objection to NAF's generosity is the lack of generalizability of this finding. With the exception of social-welfare models, in which agents are willing to exhibit generosity to increase social surplus, other models do not establish the presence of ad hoc generosity in the agents' decisions. In the difference-aversion models, for example, agents may be particularly sensitive and reactive to any indication that the other party is doing better or coming out ahead. Furthermore, in the reciprocity models, an agent's generosity is likely to be influenced by the other party's behavior. Many recent *in vivo* (with human subjects) experimental studies have demonstrated that the willingness to be generous in games is sensitive not only to the choice set available to the agent contemplating an action, but also to the behavior of the other agent that generated that choice set. (See, for example, Ioannou, Qi and Rustichini 2012.) Consequently, when an unfavorable outcome is attributed to the other party's greed, individuals will abandon cooperation. Finally, in the competitive (self-interest) models, agents approach the game with a preference for outcomes that maximize the difference between the parties. Thus, agents with a competitive motive may be particularly unlikely to regard generosity as an attractive or viable strategy in a PD game.

In addition, laboratory research has identified several psychological factors that might diminish generosity among human strategists. First, the fear of exploitation or desire to avoid the "sucker's" payoff[22] may make it difficult for individuals to risk generosity. Human experimental studies of the PD game, in particular, have found that agents will often cooperate until they have evidence or even the mere suspicion that the other party is taking advantage of them. (See discussions in Dawes and Thaler 1988.) The fear of exploitation may induce individuals to engage in a kind of defensive "stinginess," even though they recognize the merits of generosity. Furthermore, Pedro Dal Bó and Guillaume Fréchette (2011) provide compelling experimental evidence with human data to suggest that even in treatments where cooperation can be supported in equilibrium, the level of cooperation may remain at low levels even after

bilateral play. VIGILANT was a highly provocable and unforgiving strategy that retaliated sharply if it inferred that its partner was playing anything less than maximal cooperation.

[22]Recall that the "sucker's" payoff is the payoff upon unilaterally cooperating in the PD game and the "temptation" payoff is the payoff upon unilaterally defecting.

significant experience is obtained. The authors conclude that "these results cast doubt on the common assumption that agents will make the most of the opportunity to cooperate whenever it is possible to do so in equilibrium" (page 412).

The summary measures of the present study point to a very different direction from that in Bendor, Kramer and Stout (1991). In the computational experiments, cooperation reciprocity of the automata is relatively low, reflecting their readiness to exploit (to sneak on the opponent). Recall that $1-$*cooperation reciprocity* is a proxy for the degree of "sneakiness" of the automata. Therefore, automata with relatively low cooperation reciprocity are also relatively more sneaky. The reason is that an attempt to acquire the "temptation" payoff can be camouflaged in the presence of errors but not in their absence. Furthermore, in sharp contrast to NAF, the automata that evolve here are relentless punishers of defections. In fact, defection reciprocity of the automata climbs to a proportion close to one in the environments with high error levels, indicating their lack of forgiveness to defections.

Also of importance is the finding that the size of the automaton is decreasing in the probability of errors. Thus, we conjecture that in the presence of errors, strategic simplification is advantageous (if the number of accessible states is assumed to be a good measure of complexity). In the absence of errors, the behavior of well-informed agents responding with flexibility to every perturbation in the environment does not produce easily recognizable patterns, but, rather, is extremely difficult to predict. However, in the presence of errors, behavior is governed by mechanisms that restrict the flexibility to choose potential actions. These mechanisms simplify behavior to less-complex patterns, which are easier for an observer to recognize and to predict.

In real life, there exist a number of examples in which an agent's choice of a simple strategy in the presence of errors has proved quite successful. The historical series of publications on strategies to win at blackjack provide an interesting example. Edward Thorpe's book *Beat the Dealer* emphasized sophisticated card-counting and bet-variation methods. However, while no one has challenged the mathematical validity of these earlier more-complex methods, their actual use resulted in worse performance by most persons attempting to use them (which generated sizable unexpected profits for the casinos!). As a result, later books have steadily evolved towards more-rigidly-constructed methods (Heiner 1983).

On other occasions, agents apply simple heuristics when they find it too cumbersome to have complex decision rules to accommodate for the different contingencies that might arise. As Gigerenzer and Gaissmaier (2011) note,

> [i]n a number of large worlds [defined as environments where the conditions for rational decision theory are not met], simple heuristics are more accurate than standard statistical methods that have the same or more information (page 453).

For example, consider commercial retailers who need to distinguish those customers who are likely to purchase again in a given time frame (active customers) from those who are not (inactive customers). Academics often opt for the state-of-the-art Negative Binomial Distribution (NBD) model.[23] Yet most managers around the globe rely on the *hiatus heuristic*, which states that if a customer has not purchased within a certain number of months (the hiatus), the customer is classified as inactive; otherwise, the customer is classified as active. Wubben and Wangenheim (2008) compared the accuracy of the hiatus heuristic with that of the NBD model. For the heuristic, they used a hiatus of 9 months, which is common in the retailing industry, while for the NBD model, they estimated the parameters from 40 weeks of data and tested it over the following 40 weeks. The hiatus heuristic correctly classified 83% of the customers, whereas the NBD model classified only 75% correctly. Similar results have been found in the airline industry as well. These studies demonstrate the so-called "less-is-more" effect: an agent's overall performance may actually be improved by restricting flexibility to use information or to choose particular actions.

Finally, another interesting finding of the study is that perception errors are more detrimental than implementation errors to the fitness of the prevailing automata. Furthermore, the average payoffs of the automata are lower with perception errors than when both types of errors persist for the same error level. By contrast, the average payoffs of the automata are higher with implementation errors than when both types of errors persist for the same error level. Clearly, this observation is directly linked to the type of automata that survive and, in particular, the automaton ALLD. Perception errors do not affect ALLD. In fact, perception errors do not affect absorbing[24] (terminal) states. With perception errors, cooperative automata

[23]The Negative Binomial Distribution model assumes that purchases follow a Poisson process with a purchase parameter λ, that customers' lifetimes follow an exponential distribution with a dropout rate parameter μ, and that, across customers, purchase and dropout rates are distributed according to a gamma distribution.

[24]Once the automaton enters an absorbing state, it remains there for the duration of the game-play. Trigger

do not survive, as they are exploited by automata with defecting absorbing states. Thus, automata such as ALLD prevail and proliferate. Yet such automata lock themselves into endless vendettas (strings of defections) that decrease their payoffs considerably, which explains why, in the presence of only perception errors, the payoffs are lower than when both types of errors persist. However, implementation errors affect *all* automata, albeit in different ways. Note that implementation errors allow a mixture of cooperative and non-cooperative automata (such as ALLD) to survive, as the benefit of exploitation by the non-cooperative automata is *lower*. Consequently, the survival of cooperating and non-cooperating automata is instrumental in elevating the payoffs of the automata. These observations signify the importance of separating between the two types of errors.[25]

8 Conclusion

The study indicates, via an explicit evolutionary process simulated by a genetic algorithm, that the incorporation of implementation and perception errors is sufficient to alter the evolution of cooperative automata. In particular, we find that the prevailing structures tend to exhibit low reciprocal cooperation and low tolerance to defections as the probability of errors increases. Furthermore, the complexity of the automata is decreasing in the probability of errors, which suggests that strategic simplification is advantageous in the presence of errors. In addition, the study identifies a threshold error level. Below the threshold, the prevailing structures are closed-loop and diverse, which impedes any inferential projections on the superiority of a particular automaton. At and above the threshold, though, the prevailing structures converge to the open-loop automaton Always-Defect (ALLD). Finally, we find that perception errors are more detrimental than implementation errors to the payoffs of the automata. The latter finding makes it necessary to prioritize seeking ways to limit the level of perception errors first, so as to avoid suboptimal outcomes in our strategic interactions. Possible routes to limit perception errors could be through better training and transparent communication channels.

The dominance of one state automaton ALLD at or above the threshold error level is

automata have absorbing states as well as open-loop automata.

[25]We thank an anonymous referee for suggesting this vital direction.

undeniable. In fact, such dominance renders unnecessary any discussion about incorporating complexity costs based on the number of states accessed into the model. Such a venue would only expedite the dominance of ALLD. However, an interesting direction for future research would be to use the methodology prescribed to analyze the evolution of strategies in other 2×2 games. Ultimately, one would like to gather the prevailing strategies across each of these games and determine the single best strategy in a tournament consisting of a vast array of games. Having said this, we do hope, more than our specific findings and interpretations, that this work will help move computational research away from the study of misleading, hyper-rational agents and towards the study of agents with human-like characteristics and qualities. The results highlight that incorporation of implementation and perception errors in the modeling framework is an important and essential aspect of the evolutionary dynamics.

References

[1] Abreu, D., and Rubinstein, A. (1988). "The Structure of Nash Equilibrium in Repeated Games with Finite Automata," *Econometrica* 56, 1259-1282.

[2] Aumann, R. (1981). "Survey of Repeated Games," In Essays in Game Theory and Mathematical Economics in Honor of Oskar Morgenstern, Mannheim, Bibliographisches Institut.

[3] Axelrod, R. (1980). "Effective Choice in the Prisoner's Dilemma," *Journal Of Conflict Resolution* 24, 3-25.

[4] Axelrod, R. (1980). "More Effective Choice in the Prisoner's Dilemma," *Journal Of Conflict Resolution* 24, 379-403.

[5] Axelrod, R. (1984). The Evolution of Cooperation, Basic Books: New York.

[6] Axelrod, R. (1987). "The Evolution of Strategies in the Iterated Prisoner's Dilemma," In L. Davis (Ed.), Genetic Algorithms and Simulated Annealing, London: Pitman, and Los Altos, California: Morgan Kaufmann, 32-41.

[7] Axelrod, R., and Dion, D. (1988). "The Further Evolution of Cooperation," *Science* 242, 1385-1390.

[8] Axelrod, R. and Hamilton, W. D. (1981). "The Evolution of Cooperation," *Science* 211, 1390-1398.

[9] Banks, J. S., and Sundaram, R. K. (1990). "Repeated Games, Finite Automata, and Complexity," *Games and Economic Behavior* 2, 97-117.

[10] Bendor, J., Kramer, R. and Stout, S. (1991). "When in Doubt... Cooperation in a Noisy Prisoner's Dilemma," *Journal Of Conflict Resolution* 35, 691-719.

[11] Ben-Shoham, A., Serrano, R., and Volij O. (2004). "The Evolution of Exchange," *Journal of Economic Theory* 114, 310-328.

[12] Bergin, J., and Lipman, B. L. (1996). "Evolution with State-Dependent Mutations," *Econometrica* 64, 943-956.

[13] Binmore, K. G., and Samuelson, L. (1992). "Evolutionary Stability in Repeated Games Played by Finite Automata," *Journal of Economic Theory* 57, 278-305.

[14] Boyd, R., and Loberbaum, J. (1987). "No Pure Strategy is Evolutionary Stable in the Repeated Prisoner's Dilemma Game," *Nature* 327, 58-59.

[15] Cardoza, A. D. (1981). Winning Casino Blackjack for the Noncounter, Cardoza School of Blackjack: Santa Cruz.

[16] Dal Bó , P., and Fréchette, G.R. (2011). "The Evolution of Cooperation in Infinitely Repeated Games: Experimental Evidence," *American Economic Review* 101, 411-429.

[17] Dawes, R. M., and Thaler, R.H. (1988). "Anomalies: Cooperation," *Journal of Economic Perspectives* 2, 187-197.

[18] Dubey, L. B. (1980). No Need to Count: A Practical Approach to Casino Blackjack, A.S. Barnes & Co: San Diego.

[19] Epanechnikov, V. A. (1969). "Non-parametric Estimates of a Multivariate Probability Density," *Theory of Probability and its Applications* 14, 153-158.

[20] Foster, D.P., and Young, H. P. (1990). "Stochastic Evolutionary Game Dynamics," *Theoretical Population Biology* 38, 219-232.

[21] Franchini, E., Brito C.J., and Artioli, G.G. (2012). "Weight Loss in Combat Sports: Physiological, Psychological and Performance Effects," *Journal of the International Society of Sports Nutrition* 9, 52.

[22] Fudenberg, D., and Maskin, E. (1986). "The Folk Theorem in Repeated Games with Discounting and with Incomplete Information," *Econometrica* 54, 533-554.

[23] Gigerenzer, G., and Gaissmaier, W. (2011). "Heuristic Decision Making," *Annual Review of Psychology* 62, 451-482.

[24] Glover F., and Laguna M. (1993). "Tabu Search," In C.R. Reeves (Ed.), Modern Heuristic Techniques for Combinatorial Problems, Blackwell Scientific Publishing: Oxford, 70.

[25] Goldberg, D. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Company.

[26] Goldstein, J. (1991). "Reciprocity in Superpower Relations: An Empirical Analysis," *International Studies Quarterly* 35, 195-209.

[27] Hamo, Y., and Markovitch, S. (2005). "The Compset Algorithm for Subset Selection," In Proceedings of The Nineteenth International Joint Conference for Artificial Intelligence, 728-733.

[28] Heiner, R. (1983). "The Origins of Predictable Behavior," *American Economic Review* 73, 560-595.

[29] Holland, J. (1975). Adaptation in Natural and Artificial Systems, MIT Press.

[30] Ioannou, C., Qi, S., and Rustichini, A. (2012). "Group Outcomes and Group Behavior," Working Paper, University of Southampton.

[31] Kandori, M., Mailath, G., and Rob, R. (1993). "Learning, Mutation, and Long Run Equilibria in Games," *Econometrica* 61, 29-56.

[32] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). "Optimization by Simulated Annealing," *Science* 220, 671680.

[33] Ledolter, J. (2008). "Smoothing Times Series with Local Polynomial Regression on Time," Communications in Statistics - Theory and Methods, Vol. 37, 959-971.

[34] Marks, R.E. (1992). "Breeding Hybrid Strategies: Optimal Behavior for Oligopolists," *Journal Of Evolutionary Economics* 2, 17-38.

[35] Miller, J. (1996). "The Coevolution of Automata in the Repeated Prisoner's Dilemma," *Journal Of Economic Behavior & Organization* 29, 87-112.

[36] Mitchell, M. (1998). An Introduction to Genetic Algorithms, Cambridge, Massachusetts: MIT Press.

[37] Moore, E. (1956). "Gedanken Experiments on Sequential Machines," *Annals of Mathematical Studies* 34, 129-153.

[38] Nowak, M. A., and Sigmund, K. (1992). "Tit-For-Tat in Heterogeneous Populations," *Nature* 355, 250-253.

[39] Nowak, M. A., and Sigmund, K. (1993). "A Strategy of Win-Stay, Lose-Shift that Outperforms Tit-For-Tat in Prisoner's Dilemma," *Nature* 364, 56-58.

[40] Neyman, A. (1985). "Bounded Complexity Justifies Cooperation in the Finitely Repeated Prisoners Dilemma," *Economics Letters* 19, 227-229.

[41] Rubinstein, A. (1986). "Finite Automata Play the Repeated Prisoner's Dilemma," *Journal Of Economic Theory* 39, 83-96.

[42] Rust, J., Miller, J. H., and Palmer, R. (1994). "Characterizing Effective Trading Strategies," *Journal of Economic Dynamics and Control* 18, 61-96.

[43] Selten, R. (1975). "A Re-examination of the Perfectness Concept for Equilibrium Points in Extensive Games," *International Journal of Game Theory* 4, 25-55.

[44] Selten, R., Mitzkewitz, G., and Uhlich, R. (1997). "Duopoly Strategies Programmed By Experienced Traders," *Econometrica* 65, 517-555.

[45] Steen, S.N., and Brownell, K.D. (1990). "Patterns of Weight Loss and Regain in Wrestlers: Has the Tradition Changed?," *Medicine & Science in Sports & Exercise* 22, 762-768.

[46] Thorp, E. O. (1962). Beat the Dealer: A Winning Strategy for the Game of Twenty-One, Vintage Books: New York.

[47] Volij, O. (2002). "In Defense of Defect," *Games and Economic Behavior* 39, 309-321.

[48] Vega-Redondo, F. (1997). "The Evolution of Walrasian Behaviour," *Econometrica* 65, 375-384.

[49] Wu, J., and Axelrod, A. (1995). "How to Cope with Noise in the Iterated Prisoner's Dilemma," *Journal of Conflict Resolution* 39, 183-189.

[50] Wubben, M, von Wangenheim, F. (2008). "Instant Customer Base Analysis: Managerial Heuristics Often Get It Right," *Journal of Marketing* 72, 82-93.

Appendix

Numerical Examples

We provide the tables for the payoffs of ALLD and TFT for the numerical examples in Section 2.3. Recall that the examples pertain to a population size of 30 agents. The agents can use either TFT or ALLD and are paired in a round-robin structure. Additionally, we assume that a strategy does play itself. Furthermore, to calculate the average payoff of each pair, we use the values of the payoff matrix in Table 3. Finally, assume that x agents use TFT, and $30 - x$ agents use ALLD, where $x \in \mathbb{N}$ ($\mathbb{N} = \{1, 2, ..., 29\}$).

Table 6: Average Payoffs of ALLD & TFT

ε=δ=2%			ε=δ=20%		
x	*P*(ALLD)	*P*(TFT)	x	*P*(ALLD)	*P*(TFT)
1	1.06	1.06	1	1.58	1.36
2	1.07	1.10	2	1.61	1.39
3	1.07	1.14	3	1.63	1.42
4	1.08	1.18	4	1.66	1.45
5	1.08	1.23	5	1.68	1.48
6	1.09	1.27	6	1.71	1.51
7	1.09	1.31	7	1.73	1.54
8	1.10	1.35	8	1.75	1.58
9	1.10	1.39	9	1.78	1.61
10	1.11	1.43	10	1.80	1.64
11	1.11	1.47	11	1.83	1.67
12	1.12	1.51	12	1.85	1.70
13	1.12	1.55	13	1.88	1.73
14	1.13	1.59	14	1.90	1.76
15	1.13	1.64	15	1.93	1.79
16	1.14	1.68	16	1.95	1.82
17	1.14	1.72	17	1.97	1.85
18	1.15	1.76	18	2.00	1.88
19	1.15	1.80	19	2.02	1.91
20	1.16	1.84	20	2.05	1.94
21	1.16	1.88	21	2.07	1.97
22	1.17	1.92	22	2.10	2.00
23	1.17	1.96	23	2.12	2.04
24	1.18	2.00	24	2.14	2.07
25	1.18	2.05	25	2.17	2.10
26	1.19	2.09	26	2.19	2.13
27	1.19	2.13	27	2.22	2.16
28	1.20	2.17	28	2.24	2.19
29	1.20	2.21	29	2.27	2.22

Notes: On the left, we indicate the average payoffs of TFT and ALLD, as we vary the proportion of agents using TFT (and ALLD) for implementation and perception errors kept fixed at 2%. On the right, we indicate the average payoffs of TFT and ALLD, as we vary the proportion of agents using TFT (and ALLD) for implementation and perception errors kept fixed at 20%.

Prevailing Automata

In Section 5.3, we provide the proportion of wins for ALLD, ALLC and Other automata for the computational treatments conducted. In Table 7, we provide the frequency of wins for ALLD, ALLC and Other automata, out of a total of 30 simulations, for each computational treatment.

Table 7: Frequency of Wins

Error Level	ALLD	ALLC	Other
1.5%	2	-	28
1.6%	4	-	26
1.7%	5	-	25
1.8%	6	-	24
1.9%	9	-	21
2%	15	-	15
3%	22	-	8
4%	25	-	5
10%	27	-	3
20%	29	-	1
30%	28	-	2
40%	21	-	9
50%	5	4	21
60%	-	25	5
70%	-	27	3
80%	-	28	2
90%	-	28	2

Notes: The Table indicates the frequency of wins for ALLD, ALLC and Other automata out of 30 simulations conducted.

Separation of Errors

In Section 5.4, we provide the proportion of wins for ALLD, ALLC and Other automata for different levels of perception errors and implementation errors. In Table 8, we provide the frequency of wins for ALLD, ALLC and Other automata out of a total of 30 simulations for each computational treatment. Panel A indicates the frequency of wins when only the likelihood of committing perception errors is varied. Panel B indicates the frequency of wins when only the likelihood of committing implementation errors is varied.

Table 8: Frequency of Wins with Errors Separated

Panel A Error Level	ALLD	ALLC	Other	*Panel B* Error Level	ALLD	ALLC	Other
1%	0	-	30	1%	0	-	30
2%	3	-	27	2%	0	-	30
3.0%	11	-	19	3%	5	-	25
3.5%	12	-	18	4%	9	-	21
3.8%	13	-	17	5.0%	12	-	18
3.9%	18	-	12	5.1%	14	-	16
4%	19	-	11	5.2%	14	-	16
5%	24	-	6	5.3%	15	-	15
10%	26	-	4	10%	25	-	5
20%	27	-	3	20%	26	-	4
30%	30	-	0	30%	29	-	1
40%	30	-	0	40%	24	-	6
50%	30	-	0	50%	2	0	28
60%	30	-	0	60%	-	22	8
70%	30	-	0	70%	-	27	3
80%	30	-	0	80%	-	30	0
90%	30	-	0	90%	-	30	0

Notes: The Table indicates the frequency of wins for ALLD, ALLC and Other automata out of 30 simulations conducted. Panel A indicates the frequency of wins when only the likelihood of committing perception errors is varied. Panel B indicates the frequency of wins when only the likelihood of committing implementation errors is varied.