



<http://www.e-jemed.org/>

ISSN : 1298-0137

e - JEMED

The Electronic Journal
of Evolutionary Modeling
and
Economic Dynamics

Article number: 1002

Please cite this article as following:

Shu-Heng Chen, John Duffy, Chia-Hsuan Yeh, 2002, Equilibrium Selection via Adaptation: Using Genetic Programming to Model Learning in a Coordination Game, The Electronic Journal of Evolutionary Modeling and Economic Dynamics, n° 1002, <http://www.e-jemed.org/1002/index.php>

***Equilibrium Selection via Adaptation: Using Genetic Programming
to Model Learning in a Coordination Game***

<i>Shu-Heng Chen</i>	<i>John Duffy</i>	<i>Chia-Hsuan Yeh</i>
<i>National Chengchi University</i>	<i>University of Pittsburgh</i>	<i>Yuan Ze University</i>

Abstract

This paper models adaptive learning behavior in a simple coordination game that Van Huyck, Cook and Battalio (1994) have investigated in a controlled laboratory setting with human subjects. We consider how populations of artificially intelligent players behave when playing the same game. We use the genetic programming paradigm, as developed by Koza (1992, 1994), to model how a population of players might learn over time. In genetic programming one seeks to breed and evolve highly fit computer programs that are capable of solving a given problem. In our application, each computer program in the population can be viewed as an individual agent's forecast rule. The various forecast rules (programs) then repeatedly take part in the coordination game evolving and adapting over time according to principles of natural selection and population genetics. We argue that the genetic programming paradigm that we use has certain advantages over other models of adaptive learning behavior in the context of the coordination game that we consider. We find that the pattern of behavior generated by our population of artificially intelligent players is remarkably similar to that followed by the human subjects who played the same game. In particular, we find that a steady state that is theoretically unstable under a myopic, best-response learning dynamic turns out to be stable under our genetic--programming--based learning system, in accordance with Van Huyck et al.'s (1994) finding using human subjects. We conclude that genetic programming techniques may serve as a plausible mechanism for modelling human behavior,

and may also serve as a useful selection criterion in environments with multiple equilibria.

Keywords: Coordination Game, Equilibrium Selection, Genetic Programming, Adaptation.

JEL: C63, D83

Copyright: Suh-Heng Chen, John Duffy, Chia-Hsuan Yeh. 2002.

Equilibrium Selection via Adaptation: Using Genetic Programming to Model Learning in a Coordination Game*

Shu-Heng Chen
Department of Economics
National Chengchi University
Taipei 11623, Taiwan
E-mail: chchen@nccu.edu.tw

John Duffy
Department of Economics
University of Pittsburgh
Pittsburgh, PA 15260 U.S.A.
E-mail: jduffy+@pitt.edu

Chia-Hsuan Yeh
Department of Information Management
Yuan Ze University
Chungli, Taoyuan 320, Taiwan
E-mail: imcyeh@saturn.yzu.edu.tw

Abstract

This paper models adaptive learning behavior in a simple coordination game that Van Huyck, Cook and Battalio (1994) have investigated in a controlled laboratory setting with human subjects. We consider how populations of artificially intelligent players behave when playing the same game. We use the genetic programming paradigm, as developed by Koza (1992, 1994), to model how a population of players might learn over time. In genetic programming one seeks to breed and evolve highly fit computer programs that are capable of solving a given problem. In our application, each computer program in the population can be viewed as an individual agent's forecast rule. The various forecast rules (programs) then repeatedly take part in the coordination game evolving and adapting over time according to principles of natural selection and population genetics. We argue that the genetic programming paradigm that we use has certain advantages over other models of adaptive learning behavior in the context of the coordination game that we consider. We find that the pattern of behavior generated by our population of artificially intelligent players is remarkably similar to that followed by the human subjects who played the same game. In particular, we find that a steady state that is theoretically unstable under a myopic, best-response learning dynamic turns out to be stable under our genetic-programming-based learning system, in accordance with Van Huyck et al.'s (1994) finding using human subjects. We conclude that genetic programming techniques may serve as a plausible mechanism for modelling human behavior, and may also serve as a useful selection criterion in environments with multiple equilibria.

JEL Classification Nos. C63, D83.

*This project was initiated while Duffy was visiting National Chengchi University. A preliminary version of this paper, Chen, Duffy and Yeh (1996), was presented at the 1996 Evolutionary Programming Conference.

1 Introduction

The empirical usefulness of static equilibrium analysis is compromised when economic models have multiple equilibria. Consequently, extensive efforts have been made to identify ways of reducing the set of equilibria that are focal in models with multiple equilibria. There seems to be some consensus emerging that a sensible selection criterion for choosing among multiple equilibria is to determine which of the candidate equilibria are stable with respect to some kind of disequilibrium, “learning” dynamic.¹ A number of such learning dynamics have been proposed and used to reduce or eliminate multiple equilibria as empirically relevant candidates. However, the notion that these learning dynamics accurately reflect the behavior of individual economic agents or groups of agents has only very recently begun to be examined through a number of controlled laboratory experiments with human subjects.²

This paper focuses on results obtained from one such experiment conducted by Van Huyck, Cook and Battalio (1994) that tested the predictions of a class of selection dynamics in a generic coordination game against the behavior of human subjects who played the same coordination game. Van Huyck et al. postulated that one of two candidate learning processes could describe the behavior of human subjects playing the coordination game. The first learning process is a Cournot-type, myopic best-response dynamic, and the second is an inertial learning algorithm that allows for slowing changing beliefs.³ Both learning models are special versions of a large class of relaxation algorithms that have frequently appeared in the learning literature.⁴ Under certain parameterizations, these two learning processes yield different predictions for the stability of one of the game’s two Nash equilibria. Van Huyck et al.’s (1994) experimental results suggest that in those parameterizations where the two learning algorithms yield different predictions, the inertial learning algorithm provides a better characterization of the behavior of human subjects in the coordination game than the myopic best response dynamic.

In this paper, we adopt a computational approach, using Koza’s (1992, 1994) genetic programming techniques to model the behavior of artificial economic agents playing the same simple coordination game that was studied by Van Huyck et al. (1994). The computational approach that we

¹For references, see, e.g. the surveys by Kreps (1990), Sargent (1993), and Marimon (1997).

²For a survey of some of these experiments, see, e.g., Kagel and Roth (1995).

³Van Huyck et al. refer to this inertial learning dynamic as the “L-map” which is a reference to Lucas’ (1986) use of this type of learning dynamic.

⁴The class of relaxation algorithms includes, for example, the past averaging algorithm of Bray (1982) and Lucas (1986), and the least squares learning algorithm of Marcet and Sargent (1989).

take to modelling agent behavior allows for a considerably more flexible experimental design than is possible with experiments involving human subjects. Moreover, unlike most rule-based models of adaptive learning behavior, the artificial players in our genetic programming implementation of the coordination game are explicitly endowed with the ability to “think” nonlinearly, and are given all the “building blocks” necessary to construct a vast array of both linear and nonlinear forecasting rules including the myopic best response and the inertial learning algorithms considered by Van Huyck et al. (1994). Thus we know, at the outset, that our artificial players are capable of both choosing and coordinating upon linear or nonlinear forecasting rules that may result in stationary, periodic or aperiodic trajectories. We find that our more general computational approach to modelling learning behavior in the coordination game results in behavior that is qualitatively similar to that of the subjects in Van Huyck et al.’s (1994) coordination game experiment. Indeed, we think of our genetic programming implementation of learning in the coordination game as a kind of robustness check on the experimental results reported for the same game. Finally, we argue that the genetic programming techniques we illustrate in this application have certain advantages over other artificial intelligence techniques that have been applied to economic models, namely, genetic algorithms.

The coordination game found in Van Huyck et al. (1994) differs from previous coordination games that have been studied experimentally, (e.g. Cooper, DeJong, Forsythe and Ross (1990) and Van Huyck, Battalio and Beil (1990, 1991)) in that 1) the set of agent actions is considerably larger (indeed, there can be a continuum of possible actions), and 2) the stability of one of the game’s two equilibria cannot be ascertained *a priori*. The first difference makes it difficult to formulate and enumerate strategies that are based upon all possible actions as is often done in adaptive learning models. The second difference arises because Van Huyck et al. (1994) entertain the notion that agents might adopt nonlinear rules to choose actions. Because of these differences, learning models that have been used to explain behavior in the early coordination game experiments, for example, the linear learning models of Crawford (1991, 1995) and the genetic algorithm approach of Arifovic (1997) are not as well suited to the coordination game environment studied by Van Huyck et al. (1994). By contrast, we argue that the genetic-programming approach that we take to modelling learning behavior is particularly well suited to the coordination game environment of Van Huyck et al. (1994). We now turn to a description of this coordination game.

2 The Coordination Game

Consider the generic coordination game $\Gamma(\omega)$, studied by Van Huyck et al. (1994). There are n players, each of whom chooses some action $e^i \in [0, 1]$, $i = 1, 2, \dots, n$. The individual player i 's payoff function in every round of play, t , is described by:

$$\pi_{i,t}(e^i, e^{-i}) = c_1 - c_2|e^i - \omega M_t(e)[1 - M_t(e)]|, \quad (1)$$

where c_1 and c_2 are constants, $M_t(e)$ denotes the median of all n players' actions in round t , $\omega \in (1, 4]$ is a given parameter and e^{-i} denotes the vector of actions taken by the other $n - 1$ players in the same round. Both the payoff function and the set of feasible actions are assumed to be common knowledge.

It is clear from the structure of the payoff function that the individual player in this game should seek to minimize the expression that lies between the absolute value signs. That is, for a given value of the median, M , the individual player's best response function $b(M)$, is:

$$b(M) = \omega M(1 - M).$$

This best response function gives rise to two Nash equilibria: a corner equilibrium, where $e^i = 0$ for all i , and an interior equilibrium where $e^i = 1 - \frac{1}{\omega}$ for all i . The best response function $b(M)$ is easily recognized to be a member of the family of quadratic maps, where the degree of curvature is determined by the tuning parameter ω .

3 Selection Dynamics

Van Huyck et al. (1994) suggested that a certain class of "relaxation algorithms" that are frequently encountered in the learning literature could be used to characterize the evolution of play of this coordination game over time. This class of relaxation algorithms is described by the simple dynamical system:

$$\begin{aligned} M_t &= b(\widehat{M}_t), \\ \widehat{M}_t &= \widehat{M}_{t-1} + \alpha_t(M_{t-1} - \widehat{M}_{t-1}), \end{aligned}$$

where \widehat{M}_t is the representative agent's *expected* value for the median at time $t > 1$, and $\alpha_t \in [0, 1]$ is a given forgetting factor. Van Huyck et al. (1994) consider two specific versions of this relaxation

algorithm: 1) a “myopic” best response algorithm where $\alpha_t = 1$ for all $t > 1$, and 2) an “inertial” algorithm, where $\alpha_t = 1/t$ for all $t > 1$.

The myopic best response version of the dynamical system gives rise to a simple first order difference equation that characterizes the evolution of the median over time:

$$M_t = \omega M_{t-1}(1 - M_{t-1})$$

It is easily shown that for $1 < \omega < 3$, the interior equilibrium, $1 - \frac{1}{\omega}$ is attracting (locally stable) while the corner equilibrium, 0, is repelling (locally unstable) under the myopic best response dynamics. However for $\omega > 3$, the dynamics of the myopic best response algorithm become increasingly more complicated, resulting in a dense set of periodic trajectories for the median that follow the Sarkovskii order. When $\omega = 3.839$, the myopic best response algorithm gives rise to a stable cycle of period 3, which according to the famous theorem of Li and Yorke implies that there are cycles of all periods and an uncountable set of nonperiodic (chaotic) trajectories.⁵ Thus, for $\omega > 3$, the interior equilibrium is no longer stable under the myopic best response dynamics.

The inertial version of the relaxation algorithm gives rise to the dynamical system:

$$\begin{aligned} M_t &= b(\widehat{M}_t) \\ \widehat{M}_t &= \frac{t-1}{t}\widehat{M}_{t-1} + \frac{1}{t}M_{t-1} \end{aligned}$$

Note that the inertial learning algorithm differs from the myopic best response algorithm in that the inertial algorithm gives most weight to the previous *expected* value of the median whereas the myopic best response algorithm gives all weight to the previous *realized* value of the median. It is easily shown that for all feasible values for ω ($\omega \in (1, 4]$), the interior equilibrium $1 - \frac{1}{\omega}$ is a global attractor under the dynamics of the inertial learning algorithm.

Thus if $1 < \omega \leq 3$, both the myopic best response and the inertial learning algorithms predict that the interior equilibrium $1 - \frac{1}{\omega}$ will be the equilibrium that agents eventually coordinate upon. However, for $3 < \omega \leq 4$, the myopic best response algorithm predicts that the interior equilibrium will be unstable, whereas the inertial learning algorithm predicts that the interior equilibrium will continue to be stable.

⁵For a detailed analysis of the first order difference equation, $M_t = \omega M_{t-1}(1 - M_{t-1})$, that characterizes the myopic best response dynamic see, e.g. Devaney (1989).

4 Experimental Results and Experimental Design

Van Huyck et al. (1994) considered two experimental versions of the coordination game, $\Gamma(\omega)$, described above. In one version of the game, $\Gamma(2.4722)$, the interior equilibrium is predicted to be stable under both the myopic best response and inertial learning dynamics based on the choice of $\omega < 3$. In a second version of game, $\Gamma(3.86957)$, the interior equilibrium is predicted to be *unstable* under the myopic best response dynamics; starting from any initial condition, the myopic best response algorithm results in a chaotic trajectory for the median. By contrast, under the inertial learning dynamics, the interior equilibrium in the game $\Gamma(3.86957)$ is predicted to remain stable since $\omega \leq 4$. Thus, the second game, $\Gamma(3.86957)$, is the more interesting one, as the stability predictions of the myopic best response and inertial learning dynamics differ for this particular game.

Van Huyck et al. (1994) report results from 2 experimental sessions of $\Gamma(2.4722)$ and 6 experimental sessions of $\Gamma(3.86957)$ using 5 subjects in each session. In all eight sessions they found that almost all subjects quickly coordinated on the interior equilibrium; that is, the interior equilibrium is judged to be stable in all treatments. The authors thus conclude that the inertial learning algorithm is a better selection device in the coordination game than the myopic best response algorithm, since the prediction of the inertial learning algorithm regarding the stability of the interior equilibrium is always consistent with the experimental findings.

Van Huyck et al.'s conclusion that the inertial learning algorithm serves as a reasonable learning model/selection criterion is subject to some criticism. First, while it is true that the inertial learning algorithm converges to the interior equilibrium in the game $\Gamma(3.86957)$ (whereas myopic best response does not), the convergence trajectory taken by this algorithm is much too smooth when compared with the evolution of the median in the human subject experiments (see the experimental data reported in Appendix B of Van Huyck et al. (1994)). A second, related criticism is that it is apparent from the experimental data that the players in the coordination game do not all use the same learning scheme. If they all did use the same scheme, then for the same sequence of values for the median, we should expect to observe the same actions being taken. However, we observe players taking many different actions, especially in the early stages of the experiment, which suggests that they do not hold identical expectations. For this reason, it seems necessary to look beyond the predictions of representative agent learning models and to consider instead the performance of heterogeneous, multi-agent learning models. Our genetic-programming-based learning model is an example of this kind of multi-agent approach.

We note also that in implementing the coordination game experimentally, Van Huyck et al. (1994) made the simplifying assumption that the action set, e^i consists of only a *finite* set of *discrete* choices; players were asked to choose an action e^i from the set of integers $\{1, 2, \dots, 90\}$. Each subject's action was then mapped into the unit interval using the function $f(e^i) = (90 - e^i)/89$. The discreteness of the action set however, leads to some rather dramatic changes in the analysis of the myopic best response dynamics for the interesting case where $\omega > 3$. First, the discreteness of the action set rules out the possibility of chaotic trajectories which require the continuum of the unit interval. Indeed, the restriction that the median takes on one of 90 values implies that the median must repeat itself at least once every 91 periods. Second, the discreteness of the action set also leads to the possibility that the interior equilibrium of the game $\Gamma(3.86957)$ is locally *stable* under the myopic best response dynamics. In particular, the stability of the interior equilibrium of the discrete choice coordination game $\Gamma(3.86957)$ now depends crucially on the initial condition, i.e. the first median value M_1 . For almost all feasible values for $M_1 \in \{1, 2, \dots, 90\}$, the myopic best response dynamics for the game $\Gamma(3.86957)$ converge to a stable *seven* cycle, implying that the interior equilibrium is unstable. However, for some initial values the interior and the corner equilibria can also be locally *stable* under the discrete choice, myopic best response dynamics.⁶

In the genetic programming implementation of the coordination game that we explore in this paper, we do not have to discretize the action set. The computer programs that we evolve are all capable of choosing actions on the continuum of the unit interval. Therefore, unlike the experimental implementation of the coordination game we do not rule out the possibility of chaotic trajectories. Moreover, by allowing a continuum for the set of feasible actions, the coordination problem faced by our artificial agents is much more complicated than that faced by the experimental subjects whose actions were limited to a finite, discrete choice set. Finally, we consider a much larger size population of players than is practically feasible in an experiment with human subjects. This larger population size should again, make the coordination problem more difficult. Thus, our genetic programming implementation of learning in the coordination game can be viewed as a check of whether the experimental results are robust to a continuous action set with a large number of players that would

⁶In particular, for $M_1 \in \{24, 67\}$, the interior equilibrium is locally stable under the discrete choice, myopic best response dynamics, and for $M_1 \in \{1, 90\}$, the corner equilibrium is locally stable under these same dynamics. It is interesting to note that in one of Van Huyck et al.'s 6 treatments of the game $\Gamma(3.86957)$ – session 7 – the initial median was 24. With this value for M_1 , the discrete choice, myopic best response dynamic would predict that the system would stay at 24, the interior equilibrium forever, and indeed, this is roughly what occurred. See figure 13 of Van Huyck et al. (1994). Thus, one cannot dismiss altogether the possibility that discrete choice, myopic best response dynamic might also characterize the behavior of the experimental subjects in the game $\Gamma(3.86957)$.

be difficult to implement in an experiment involving human subjects.

5 Genetic Programming

Before describing how we model agent behavior in the coordination game using genetic programming techniques, we first provide a brief overview of genetic programming. A more detailed description of genetic programming, especially as it applies to the coordination game that we study, is provided in Appendix A.

5.1 An Overview

Genetic programming (GP) represents a new field in the artificial intelligence literature that was developed only recently by Koza (1992, 1994) and others.⁷ GP belongs to a class of evolutionary computing techniques based on principles of population genetics. These techniques combine Darwin’s notion of natural selection through survival of the fittest with naturally occurring genetic operations of recombination (crossover) and mutation. Genetic programming techniques have already been widely applied to engineering type optimization problems (both theoretical and commercial), but have seen comparatively little application to economic problems, which are often similar in nature. The few economic applications of GP thus far include Allen and Karjalainen (1999), Chen and Yeh (1997a,b), Dworman, Kimbrough and Laing (1996) and Neely et al. (1997).

While GP techniques are often viewed as an offshoot of Holland’s (1975) genetic algorithm (GA), GP techniques are perhaps more accurately viewed as a *generalization* of the genetic algorithm. The standard genetic algorithm operates on a population of structures, usually strings of bits. Each of the members of this population, the individual *bitstrings*, represent different candidate solutions to a well-defined optimization problem. The genetic algorithm evaluates the fitness of these various candidate solutions using the given objective function of the optimization problem and retains solutions that have, on average, higher fitness values. Operations of crossover (recombination) and mutation are then applied to some of these more fit solutions as a means of creating a new “generation” of candidate solutions. The whole process is repeated over many generations, in order to evolve solutions that are as close to optimal as possible. In analyzing the evolution of solutions over time, it is typical to report the solution in each generation that has the highest fitness value – this solution is designated the “best-of-generation” solution. The algorithm is ended when this

⁷See also Kinnear (1994).

best-of-generation solution satisfies a certain criterion (e.g. some tolerance) or after some maximum number of iterations has been reached.

Theoretical analyses of genetic algorithms suggest that they are capable of quickly locating regions of large and complex search spaces that yield highly fit solutions to optimization problems. That is because the genetic operators of the GA work together to optimize on the trade-off between discovering new solutions and using solutions that have worked well in the past (Holland (1975)).⁸

Koza's idea in developing genetic programming techniques was to take the genetic algorithm a step further and ask whether the same genetic operators used in GAs could be applied to a population of *computer programs* so as to evolve highly fit computer programs. There are several advantages to using computer programs rather than bitstrings as the structures to be evolved. First, the computer programs of GP have an *explicit, dynamic structure* that can be easily represented in a *decision tree format*. By contrast, the bitstrings of GAs typically encode passive yes/no type decisions or parameter values for prespecified, often static functional forms. The dynamic nature of the computer programs of GP makes them capable of a much more sophisticated and nonlinear decision making than is generally possible using the bitstrings of GAs. Second, the computer programs of GP are *immediately implementable structures*; as such, they can be readily interpreted as the forecast rules used by a heterogeneous population of agents. For example, in GP, a computer program used by player i in round t , $gp_{i,t}$, might take the form:

$$gp_{i,t} = 0.31 + M_{t-1}(M_{t-1} - M_{t-2}).$$

Here, M_{t-j} represents the value of the median j periods in the past. Given these lagged median values, this program can be immediately executed and delivers a forecast of the median in period t , equal to the value of $0.31 + M_{t-1}(M_{t-1} - M_{t-2})$. This forecast then becomes the action taken by player i in round t . Note that this program is readily interpreted as the agent's *forecast function*. By contrast, the bitstrings used in GAs are not immediately implementable and their interpretation is less clear; these bitstrings first have to be decoded and then the decoded values must be applied to some prespecified functional form before the solution the bitstrings represent can be implemented. Finally, while the length of the bitstrings used in GAs is fixed, the length of the computer programs used in GP is free to vary (up to some limit, of course) providing for a much richer range of

⁸For an introduction to the theory of genetic algorithms, see, e.g. Goldberg (1989) or Mitchell (1996). Economic applications of genetic algorithms can be found in the work of Arifovic (1994, 1995, 1996, 1997), Arthur et al. (1997), Bullard and Duffy (1998ab), Dawid (1996), Miller (1996) and Tesfatsion (1997) among others and are also discussed in Sargent (1993) and Birchenhall (1995).

structures.⁹

Koza chose to develop GP techniques using the Lisp programming language because the syntax of Lisp allows computer programs to be easily manipulated like bitstrings, so that the same genetic operations used on bitstrings in GAs can also be applied to the computer programs that serve as the evolutionary structures in GP. Moreover, the new computer programs that result from application of these genetic operations are immediately executable programs.

Lisp has a single syntactic form, the symbolic expression (S-expression), that consists of a number of *atoms*. These atoms are either members of a *terminal* set, that comprise the *inputs* (e.g. data) to be used in the computer programs, or they are members of a *function* set that consists of a number of prespecified functions or operators that are capable of processing any data value from the terminal set and any data value that results from the application of any function or operator in the function set. Each Lisp S-expression has the property that it is immediately executable as a computer program, and can be readily depicted as a rooted, point-labeled tree. Moreover, the S-expressions are easily manipulated like data; cutting a tree at any point and recombining the cut portion with another tree (S-expression) results in a new S-expression that is immediately executable. A more detailed discussion of Lisp S-expressions is provided in Appendix A.

As Koza and others have noted the use of Lisp is not necessary for genetic programming; what is important for genetic programming is the implementation of a *Lisp-like environment*, where individual expressions can be manipulated like data, and are immediately executable. For the results reported in this paper, we have chosen to implement the Lisp environment using Pascal 4.0.¹⁰

5.2 Using Genetic Programming to Model Learning in the Coordination Game

In this section, we explain how we use genetic programming to model population learning in the coordination game. Discussion of some of the more technical details of our implementation can be found in Appendix A. The version of genetic programming used here is the *simple genetic programming* that is described in detail in Koza (1992).

Let GP_t , denote a population of trees (S-expressions), representing a collection of players' forecasting functions. A player i , $i = 1, \dots, n$, makes a decision about his action for time t using a *parse tree*, $gp_{i,t} \in GP_t$, written over the *function* and *terminal* sets that are given in Table 1.

⁹While in principle it is possible to represent dynamic, variable length expressions using the bitstrings of genetic algorithms, this has not been the practice. See Angeline (1994) for a further discussion.

¹⁰Other programming languages, e.g. C, C++, and Mathematica have also been used to implement Lisp environments.

Table 1: Tableau for the GP-Based Learning Algorithm

Population size	500
The number of initial trees generated by the full method	250
The number of initial trees generated by the grow method	250
The maximum depth of a tree	17
Function set	$\{+, -, \times, \%, Exp, Rlog, Sin, Cos\}$
Terminal set	$\{\mathfrak{R}, M_{t-1}, M_{t-2}, M_{t-3}, M_{t-4}, M_{t-5}\}$
The maximum number in the domain of Exp	1,700
The number of trees created by reproduction	50
The number of trees created by crossover	350
The number of trees created by mutation	100
The probability of mutation	0.0033
The probability of leaf selection under crossover	0.5
The maximum number of generations	1,000
Fitness Criterion	Payoff Function: $\pi_{i,t}$

As Table 1 indicates, the *function set*, includes the standard mathematical operations of addition (+), subtraction (−), multiplication (×) and protected division (%), and also includes the exponential function (*Exp*) a protected natural logarithm function (*Rlog*) and the sin and cosine functions (*Sin* and *Cos*).¹¹ This set of operators and functions is the one that the artificial agents in our experiments are “endowed” with.

The *terminal set* includes the set of constants and variables that the artificial agents may use in combination with the operators and functions from the function set to build forecast rules. As indicated in Table 1, the terminal set includes the random floating–point constant \mathfrak{R} which is restricted to range over the interval $[-9.99, 9.99]$, as well as the population *mean* choice of action lagged up to h periods, i.e., M_{t-1}, \dots, M_{t-h} . Note that in our version of the coordination game, M refers to the *mean* rather than the *median* choice of action as in Van Huyck et al. (1994).¹² The choice of the

¹¹The protected division operator protects against division by zero by returning the value 1 if its denominator argument is 0; otherwise, it returns the value from dividing its first argument (the numerator) by its second argument (the denominator). Similarly, the protected natural logarithm function avoids nonpositive arguments by returning the natural logarithm of the absolute value of its argument, and returning the value 0 if its argument is 0. The exponential function, which takes the argument x and returns the value e^x , allows a maximum argument value of 1,700 as indicated in Table 1. Such function modifications and restrictions are necessary to avoid ill–defined forecasts; these types of modifications are quite standard in the GP literature. See, e.g., Koza (1992).

¹²Van Huyck et al. (1994) used the median rather than the mean in the coordination game because in experiments involving small numbers of human subjects (they only had 5 subjects in each experimental session), the mean can be

lag length, h , determines players' ability to recall the past. We set h equal to 5, so that agents may consider as many as 5 past lagged values of the mean in their forecast functions.

The forecasting functions that players may construct and use are linear and nonlinear functions of M_{t-1}, \dots, M_{t-h} , \mathfrak{R} , and, as we shall see later, they may also be functions, in whole or in part, of past forecast rules $gp_{i,t}(M_{t-1}, \dots, M_{t-h})$. We note that the set of forecast functions that our artificial players may adopt includes the myopic best response, but not the inertial learning algorithm, as the latter requires knowledge the previous period's forecast of the mean value, \hat{M}_{t-1} .

Indeed, Chen and Yeh (1997a) have shown that GP techniques can be used to uncover a variety of nonlinear data generating functions. In one demonstration, they generated a time series for the nonlinear, chaotic dynamical system $x_{t+1} = 4x_t(1 - x_t)$, which is the same as our myopic best response law of motion with $\omega = 4$. They then used a GP-based search in an effort to recover this exogenously given system. Fitness was based on how close the forecast functions in the population came to matching the given time series behavior, and the GP function and terminal sets were nearly identical to those used in this paper. Chen and Yeh (1997a) report that the GP-based search was able to uncover the data generating process in no more than 19 generations. In this paper, by contrast, the data generating process for the mean is *endogenously* determined by the actions chosen by all of the individual players. Nevertheless, it is nice to know that a GP-based search algorithm can deduce a nonlinear data generating function such as the myopic best response law of motion.

The decoding of a parse tree $gp_{i,t}$ gives the forecasting function used by player i at time period t , i.e., $gp_{i,t}(\Omega_{t-1})$ where Ω_{t-1} is the information set containing past mean values through time $t - 1$. Evaluating $gp_{i,t}(\Omega_{t-1})$ at the realization of Ω_{t-1} gives the mean action predicted by player i in round t , i.e., $gp_{i,t}$. Without any further restriction, the range of $gp_{i,t}$ is $(-\infty, \infty)$. However, since the action space for each player is restricted to $[0, 1]$, we must restrict $gp_{i,t}$ so that it also lies in $[0, 1]$. We chose to implement this restriction in two different ways. Our first approach was to use the *symmetric sigmoidal activation function* to map $(-\infty, \infty)$ to $[0, 1]$ so as to obtain a valid mean forecast, $\hat{M}_{i,t}$, for player i in round t , i.e.

$$\hat{M}_{i,t} = \frac{1}{1 + e^{-gp_{i,t}}}$$

A second approach that we also considered was a simple *truncated linear transformation* where player

easily influenced by the behavior of a single subject. By contrast, the computational coordination game experiments that we perform involve hundreds of artificial agents, so that the use of the mean rather than the median is no longer a concern.

i 's round t forecast was determined as follows:

$$\widehat{M}_{i,t} = \begin{cases} gp_{i,t} & \text{if } 0 \leq gp_{i,t} \leq 1, \\ 1 & \text{if } gp_{i,t} > 1, \\ 0 & \text{if } gp_{i,t} < 0. \end{cases}$$

Using either of these two approaches ensures that player i 's mean choice of action lies in the feasible $[0, 1]$ interval.

Once we have all n players' mean action choices (equivalent to their mean forecasts), it is possible to determine the actual value of the mean in round t , $M_t = \frac{1}{n} \sum_{i=1}^n \widehat{M}_{i,t}$. Given this mean value, we can calculate each player's *fitness value* in round t . The *raw fitness* of a parse tree $gp_{i,t}$ is determined by the value of the player's payoffs earned in round t as determined by the payoff function $\pi_{i,t}$, given in equation (1). To avoid negative fitness values, each raw fitness value is adjusted to produce an *adjusted fitness* measure $\mu_{i,t}$ that is described as follows:

$$\mu_{i,t} = \begin{cases} \pi_{i,t} + 0.25 & \text{if } \pi_{i,t} \geq -0.25, \\ 0 & \text{if } \pi_{i,t} < -0.25. \end{cases}$$

In making this adjustment, we are effectively eliminating from the population forecast functions $gp_{i,t}$ that lose more than \$0.25, since these rules will have comparatively lower adjusted fitness values (equal to 0) than rules that did not perform so poorly. Our decision to make the above adjustment to the fitness measure was due to the following consideration. In the early rounds of a game, players have very limited experience with the environment so their expectations essentially amount to random guessing. As a consequence, many of the players will lose money. If we only considered players with forecast functions that earned positive payoffs, the selection process would quickly come to be dominated by those few players (forecast functions) that were lucky enough to earn positive payoffs in the initial stages of the game. However, we want to maintain some heterogeneity in the population and avoid the possibility of *premature convergence*, a problem that can occur in populations lacking sufficient heterogeneity. For this reason, we allow some players to earn negative payoffs, but we restrict such losses so that they do not exceed \$0.25. After a few generations when most of the players have begun to earn positive payoffs, this protection no longer plays any effective role. We have experimented with adjustment values other than 0.25. While small adjustment values do not significantly alter our simulation results, very large adjustment values do affect our results because these large values effectively nullify the adjusted fitness measure as an indicator of the relative success of a forecast function. Later in the paper, we examine what happens when we replace the adjustment value of 0.25 in the adjustment scheme described above with the

much larger value of 200.00. However, unless otherwise indicated, all of the simulation results we report below involve an adjustment value equal to 0.25.

Once all of the adjusted fitness values are determined, each adjusted fitness value $\mu_{i,t}$ is then normalized. The *normalized fitness* value $p_{i,t}$ is given by:

$$p_{i,t} = \frac{\mu_{i,t}}{\sum_{i=1}^n \mu_{i,t}}.$$

It is clear that the normalized fitness value is a *probability measure*. Moreover, $p_{i,t}$ will vary directly with the performance of the parse tree $gp_{i,t}$; the better the parse tree performs (in terms of its payoff), the higher is its normalized fitness value. The normalized fitness values $p_{i,t}$ are used to determine the next generation of agents (parse trees) GP_{t+1} from the current generation GP_t through application of the three primary genetic operators, i.e., *reproduction*, *crossover*, and *mutation*. We now describe these three genetic operators.

1. **Reproduction:**

The reproduction operator makes copies of individual parse trees from generation GP_t and places them in the next generation GP_{t+1} . The criterion used for copying is the normalized fitness value $p_{i,t}$. If $gp_{i,t}$ is an individual in the population GP_t with normalized fitness value $p_{i,t}$, then each time the reproduction operator is called, $gp_{i,t}$ will be copied into the next generation with probability $p_{i,t}$. The reproduction operator does not create anything new in the population and the “offspring” generated by reproduction constitute only part of the population of the next generation of trees, GP_{t+1} . As specified in Table 1, the reproduction operator is used to create only 10% (50 out of 500) of the next generation. The rest of the offspring are generated by the other two operators, *crossover* and *mutation*.

2. **Crossover:**

The crossover operation for the genetic programming paradigm is a sexual operation that starts with two parental parse trees that have been randomly selected from the population GP_t based upon their normalized fitness values as described above. Crossover involves exchanging different parts of these “parents” to produce two new “offspring.” This exchange begins by randomly and independently selecting a single point on each parental parse tree using a uniform distribution described below. By the syntax of Lisp, each point (atom) of a parse tree could be either a *leaf* (terminal) or a *inner code* (function). Thus, the point (atom) selected for

crossover could either be a terminal or a function. As specified in Table 1, the probability that the crossover point is a terminal or a function is the same, i.e., one-half. Given that a terminal or function is to be the point chosen for crossover, the probability that any terminal or function is chosen as the crossover point is uniformly distributed. For example, if the crossover point is to be a terminal, and there are three terminals in the parse tree, the probability that any one of the three terminals is chosen for the crossover point is one-third. Unlike reproduction, the crossover operation adds new individuals (new forecasts rules) to the population. As indicated in Table 1, crossover is responsible for creating 70% (350 out of 500) of the next generation of parse trees, GP_{t+1} .

3. Mutation:

The operation of mutation also allows for the creation of new individuals. The mutation operator begins by selecting a parse tree $gp_{i,t}$ from the population GP_t based once again upon normalized fitness values $p_{i,t}$. Each point (atom) of the selected parse tree is then subjected to mutation (alteration) with a small, fixed probability. As specified in Table 1, this fixed probability of mutation is 0.0033. To ensure that the resulting expression is a syntactically and semantically valid Lisp S-expression, terminals can only be altered to another member from the terminal set and functions can only be altered to another member from the function set possessing the same number of arguments. The altered individual forecast rule (parse tree) is then copied into the next generation of the population. As indicated in Table 1, mutation is responsible for creating 20% (100 out of 500) of the next generation of parse trees.

The three operators combined create the population GP_{t+1} by copying, recombining and mutating the parse trees that make up the population GP_t . Once the new population GP_{t+1} has been created, the decoding of each parse tree $gp_{i,t+1}$ is performed to obtain the new mean, M_{t+1} . Once the new mean is determined, the raw, adjusted and normalized fitness values for each parse tree can be determined using the payoff function (1), and the GP operators can then be applied to create the population GP_{t+2} . The algorithm continues with successive generations, up to the maximum number of generations. We set the maximum number of generations equal to 1,000 as indicated in Table 1.

The initial S-expressions were randomly generated using both of the methods suggested by Koza

Table 2: Parameter Values for the Coordination Game
Used in the Genetic Programming Simulations

Parameter	Case 1	Case 2
ω	2.47222	3.86957
c_1	0.5	0.5
c_2	1.0	1.0
n	500	500
e_I^*	0	0
e_{II}^*	0.59551	0.74157

e_I^* : The optimal action under the strict equilibrium $e^i = 0 \quad \forall i$.

e_{II}^* : The optimal action under the strict equilibrium $e^i = 1 - \frac{1}{\omega} \quad \forall i$.

(1992) – the *full* method and the *grow* method.¹³ Together, these two initialization methods provide for a great diversity of initial programs. Table 1 indicates that each method was responsible for creating one-half (250) of the initial population of trees, GP_1 .

We consider the same two coordination games studied by Van Huyck et al. (1994), although as mentioned previously, we do not restrict the action set to a finite set of discrete choices. Furthermore, we have many more (artificial) players. We refer to the game $\Gamma(2.4722)$ studied by Van Huyck et al. (1994) as Case 1 and the other game these authors considered, $\Gamma(3.86957)$, as Case 2. The exact parameterizations of these two cases are reported in Table 2.

6 Simulation Results

Our simulation experiments were organized as follows. For each of the two different transformation functions – the symmetric sigmoidal transformation function and the truncated linear transformation function – we conducted 10 simulations for a total of 20 simulations. Within each group of 10 simulations, 5 of the simulations were conducted under the Case 1 parameterization and 5 were conducted under the Case 2 parameterization.

We focus our attention first on the 10 simulations that we conducted using the *symmetric sigmoidal transformation function*. Means and standard deviations from these 10 simulations are reported in Table 3. In this table, simulation 1.1 refers to our first simulation of Case 1, while simulation 2.1 refers to our first simulation of Case 2, and so on. Time series for the mean, M_t , from a single simulation of Case 1 and Case 2 are plotted in Figures 1A–1B. These time series

¹³See Appendix A for a brief description of these two methods. See also Koza (1992) pp. 92–93.

Table 3: GP Simulation Results Using the Symmetric Sigmoidal Transformation

Case		Simulation				
		1	2	3	4	5
1	\overline{M}_a	0.5917	0.5897	0.5910	0.5907	0.5901
	$\delta_{M,a}$	0.0118	0.0107	0.0091	0.0094	0.0122
	$\delta_{M^*,a}$	0.0124	0.0122	0.0102	0.0105	0.0133
1	\overline{M}_b	0.5958	0.5925	0.5936	0.5933	0.5946
	$\delta_{M,b}$	0.0037	0.0026	0.0033	0.0028	0.0021
	$\delta_{M^*,b}$	0.0038	0.0039	0.0038	0.0035	0.0023
2	\overline{M}_a	0.7406	0.7411	0.7415	0.7450	0.7447
	$\delta_{M,a}$	0.0061	0.0067	0.0069	0.0064	0.0064
	$\delta_{M^*,a}$	0.0061	0.0067	0.0069	0.0073	0.0071
2	\overline{M}_b	0.7394	0.7403	0.7399	0.7434	0.7431
	$\delta_{M,b}$	0.0033	0.0039	0.0040	0.0034	0.0024
	$\delta_{M^*,b}$	0.0039	0.0041	0.0043	0.0039	0.0029

\overline{M}_a = the average of M_t of a simulation from Generation 1 to 1,000.

\overline{M}_b = the average of M_t of a simulation from Generation 201 to 1,000.

$\delta_{M,a}$ = standard deviation about the M_a of a simulation from Generation 1 to 1,000.

$\delta_{M,b}$ = standard deviation about the M_b of a simulation from Generation 201 to 1,000.

$\delta_{M^*,a}$ = standard deviation about the *strict interior equilibrium* $1 - \frac{1}{\omega}$ from Generation 1 to 1,000.

$\delta_{M^*,b}$ = standard deviation about the *strict interior equilibrium* $1 - \frac{1}{\omega}$ from Generation 201 to 1,000.

plots are typical of the other simulations we conducted for the two cases. As these figures clearly indicate, the time series for M_t in both cases of the GP-based coordination game (Case 1 and Case 2) tend to converge to a neighborhood of the strict interior equilibrium $1 - \frac{1}{\omega}$, i.e., 0.59551 for Case 1 and 0.74157 for Case 2. In addition, the transition to $1 - \frac{1}{\omega}$ is remarkably brief; If one considers $(0.99 - \frac{1}{\omega}, 1.01 - \frac{1}{\omega})$ as a neighborhood of $1 - \frac{1}{\omega}$ then, for all simulations, it takes no more than 50 generations to move into this neighborhood.

A second finding is that while M_t does not converge to $1 - \frac{1}{\omega}$ in a strict sense, due to the constant mutation rate, there appears to be a force that serves to stabilize the movement of M_t in a very small band around the interior equilibrium. In other words, GP-based coordination games have a self-stabilizing feature. These properties are also revealed by Table 3.

As Table 3 reveals, in almost all of our simulations, the average of the means, M_t , from generation 201 to 1,000, i.e. \overline{M}_b , does not deviate from the interior equilibrium value, $1 - \frac{1}{\omega}$, by more than 0.5%. Note also that if we compare the standard deviations, $\delta_{M,a}$ with $\delta_{M,b}$ or $\delta_{M^*,a}$ with $\delta_{M^*,b}$ for each simulation, we see that after the first 200 periods of learning, the stability of the mean in all of the GP-based coordination games improves.

Figure 1A : The Time Series of the Mean Choice of Action of the Coordination Game

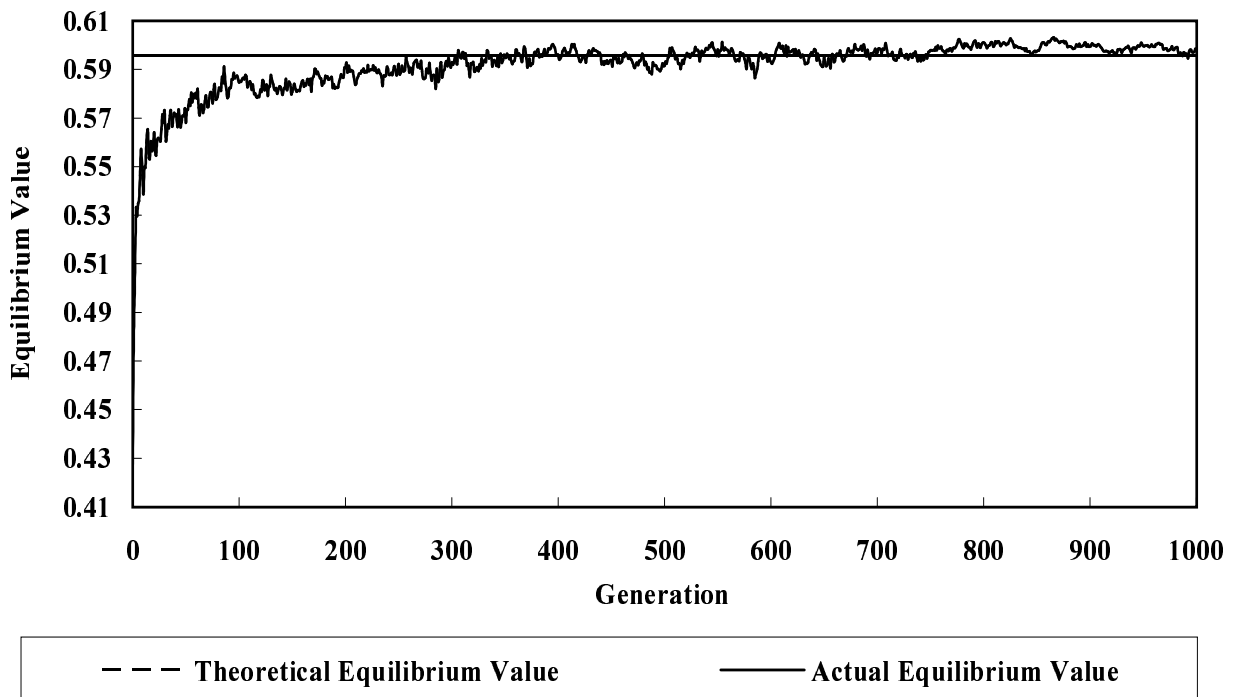
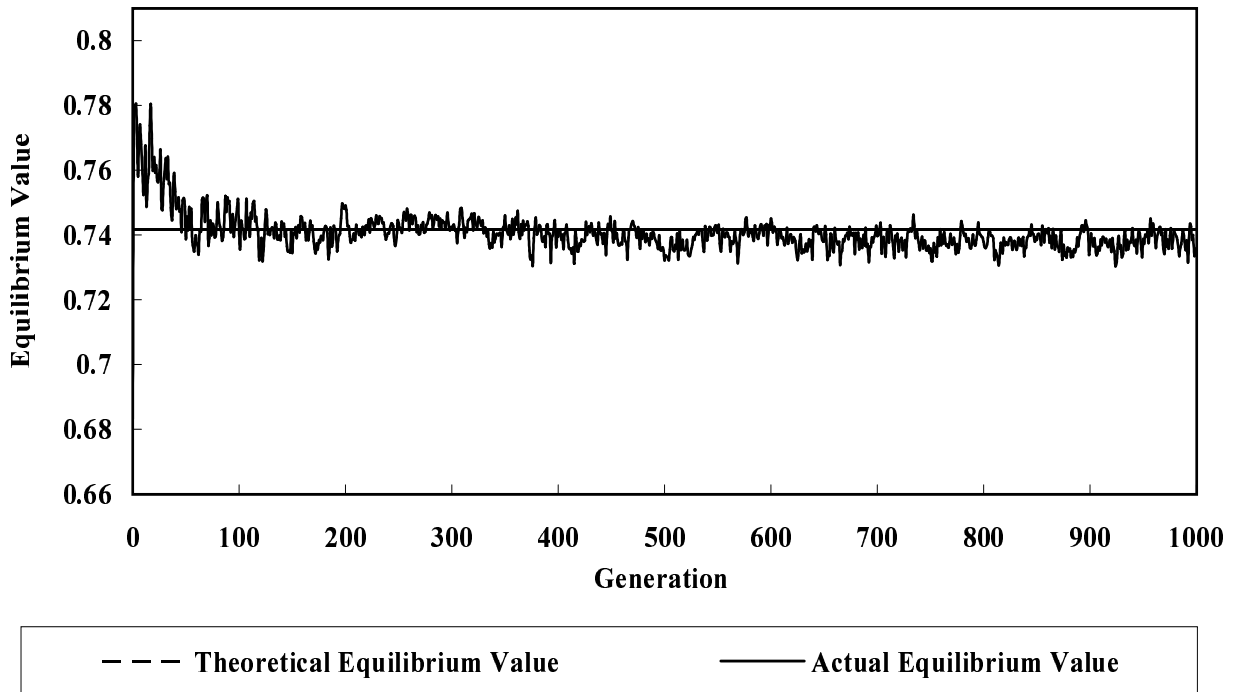


Figure 1B : The Time Series of the Mean Choice of Action of the Coordination Game



A third result is that the chaotic trajectories for $\Gamma(3.86957)$ that are predicted by the myopic best response dynamic are *not apparent* in any of our simulations of Case 2. However, by comparing $\delta_{M,b}$ or $\delta_{M^*,b}$ across Case 1 and Case 2 in Table 3, we find that the standard deviations in Case 2 are generally somewhat larger than those in Case 1. Indeed, a rank order test reveals that $\delta_{M^*,b}$ is significantly larger in Case 2 as compared with Case 1 ($p \leq .10$).¹⁴ This difference between the two cases is also apparent from a visual comparison between Figures 1A and 1B. Thus, while it appears that the aggregate outcome from the GP simulations is similar for both treatments, there appears to be some evidence that the coordination problem in Case 2 is more difficult for our artificial players than is the coordination problem in Case 1.

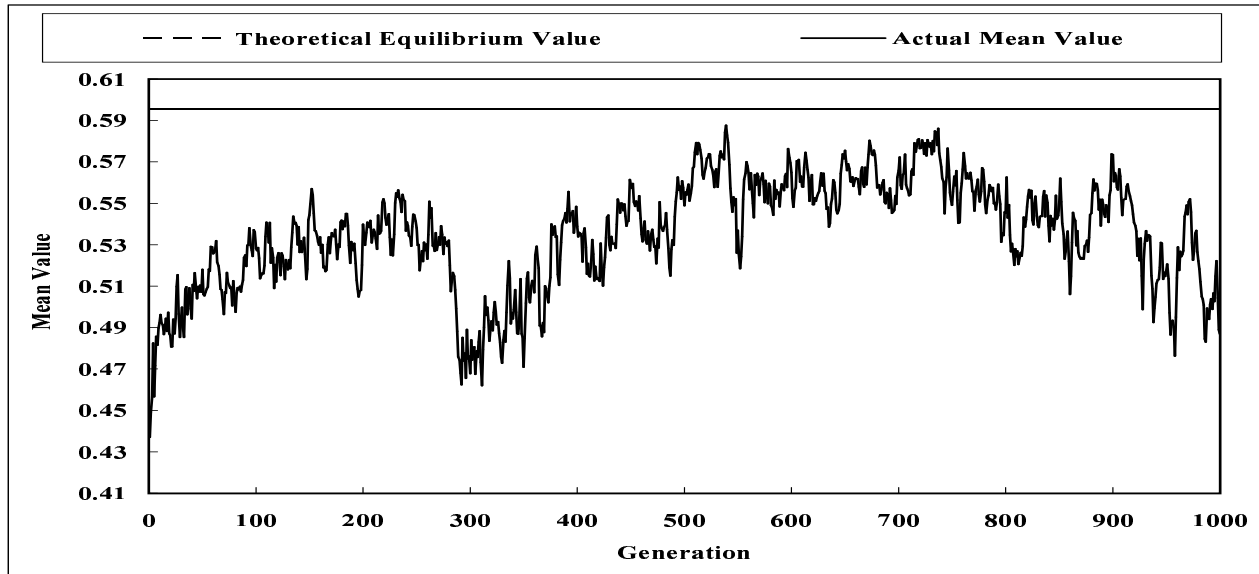
We have also considered how sensitive our results are to the use of payoff fitness as the main determinant of successive generations of forecast rules through application of the reproduction, crossover and mutation operations. Recall that we made an adjustment to the raw fitness values, so as to avoid excluding rules with negative payoffs. In all of our simulations we used an adjustment factor of 0.25. We have also performed a simulation exercise where we considered what happens when we used a much larger adjustment factor of 200.00. That is, we adjusted raw fitness values, $\pi_{i,t}$ as follows:

$$\mu_{i,t} = \begin{cases} \pi_{i,t} + 200.00 & \text{if } \pi_{i,t} \geq -200.00, \\ 0 & \text{if } \pi_{i,t} < -200.00. \end{cases}$$

The effect of this adjustment is to nullify the usefulness of fitness as an indicator of the relative success of individual forecast functions. That is because the raw, unadjusted fitness values, $\pi_{i,t}$ can only take on values in the range $[-.50, .50]$. (See the payoff function (1) and the parameterizations of this function given in Table 2). Adding 200.00 to these raw fitness values makes them essentially indistinguishable from one another, even after the adjusted fitness values have been converted into the normalized fitness values that are used to determine application of the reproduction, crossover and mutation operations.

Thus, the experiment where the adjustment value is set at 200.00 rather than at 0.25 serves as a test of whether relative fitness values are the driving force behind the results reported above. Indeed, this experiment is a test of the explanatory power of GP techniques. Figure 2 presents the time series for the mean from the single experiment involving Case 1 where we set the adjustment factor equal to 200.00 rather than 0.25. We see in this figure that the mean just wanders about randomly

¹⁴No significant difference was found for $\delta_{M,b}$ between Cases 1 and 2. See Siegel and Castellan (1988) for an explanation of the nonparametric, robust rank order test used here.



and has not settled down after 1,000 iterations. In particular, the mean does not approach either the corner equilibrium (0) or the interior equilibrium of Case 1 (.59551). We may conclude from this exercise that the reliance of the genetic operators on relative fitness values is a driving force behind our simulation results, i.e. that fitness of forecast functions matters.

In addition to considering the dynamics of the mean choice of action, it is also interesting to examine the evolution of the population of forecast functions, i.e., GP_t . The *length* of the best-of-generation forecast function (Lisp S-expression) varies pretty widely. The length of a forecast function is measured by counting the number of elements (atoms) that are used in the program.¹⁵ Programs with longer lengths are more complex than those with shorter lengths, so the length of the program serves as a measure of the complexity of the forecast rule. Initially, the length of the best-of-generation program is rather small, but over time, the length increases substantially.

Consider, Simulation 2.5 for example. (Simulation 2.5 corresponds to the fifth simulation of Case 2). The length of the shortest best-of-generation program (S-expression) in this simulation is 15 and it appears in generations 16, 23, 27, 29, 32, 35, 37, 40, 41 and 50:¹⁶

$$\begin{aligned} gp_{\text{best},16} &= ((M_{t-5} + M_{t-4}) * M_{t-4}) \\ gp_{\text{best},23} &= ((M_{t-5} + M_{t-4}) * M_{t-4}) \end{aligned}$$

¹⁵The length of a Lisp S-expression is distinct from the *depth* of a Lisp S-expression in tree form. See Appendix A for a further discussion.

¹⁶All of the GP programs below are represented as algebraic expressions (so that they can be more easily understood) rather than in the Lisp S-expression form in which they are encoded for GP.

$$\begin{aligned}
gp_{\text{best},27} &= ((M_{t-5} + M_{t-3}) * M_{t-4}) \\
gp_{\text{best},29} &= (M_{t-3} * (M_{t-5} + M_{t-4})) \\
gp_{\text{best},32} &= ((M_{t-1} + M_{t-3}) * M_{t-3}) \\
gp_{\text{best},35} &= (M_{t-5} * (M_{t-5} + M_{t-4})) \\
gp_{\text{best},37} &= (M_{t-3} * (M_{t-4} + M_{t-5})) \\
gp_{\text{best},40} &= (M_{t-3} * (M_{t-4} + M_{t-5})) \\
gp_{\text{best},41} &= (M_{t-2} * (M_{t-5} + M_{t-5})) \\
gp_{\text{best},50} &= (M_{t-5} * (M_{t-4} + M_{t-5}))
\end{aligned}$$

Programs with such a small program length continued to appear frequently after generation 50 but they were no longer selected as best-of-generation programs. Instead, increasingly complicated programs with lengths over 200 were more likely to be selected as the best-of-generation; the longest best-of-generation program with a length of 459 appeared in generation 554:

$$\begin{aligned}
gp_{\text{best},554} = & (SinCos((CosCosCos(CosM_{t-3} * M_{t-3}) * CosSin(CosCos(Cos(\\
& CosM_{t-4} \% M_{t-4}) * Cos(M_{t-2} + SinSinM_{t-1}))) + (M_{t-5} * CosM_{t-2} \\
&))) * CosCosCosSin((M_{t-3} * M_{t-3}) - CosM_{t-3}) \% CosSinCos((\\
& (Cos(CosM_{t-1} * Cos(ExpSinCosM_{t-4} * M_{t-2})) \% CosSinCosSin \\
& (M_{t-4} \% CosSinCosM_{t-1})) * CosSinRLog((Sin(M_{t-4} * CosCosM_{t-1} \\
&) * Cos(M_{t-4} \% ((CosM_{t-5} + M_{t-2}) * (CosM_{t-3} * M_{t-4})))) * Cos((\\
& Cos(M_{t-3} * CosM_{t-5}) * CosCosM_{t-4}) * M_{t-1}))) * Cos(Sin(Sin \\
& Cos(CosSinCosM_{t-1} * (CosSinM_{t-2} * CosCosM_{t-4})) + ExpSin \\
& CosCosSin((M_{t-2} * M_{t-3}) - CosM_{t-3})) * Cos((SinM_{t-3} + M_{t-5}) \\
& * ((M_{t-4} + (M_{t-3} + ((M_{t-2} * SinM_{t-2}) * M_{t-1}))) \% CosM_{t-3}))))))
\end{aligned}$$

At generation 1,000 of simulation 2.5, the length of the individual programs was found to vary over the interval [3, 297]. This wide variation in program length implies that considerable heterogeneity remains in the population of forecast rules even after many generations.

Our findings that the best-of-generation programs become increasingly more complicated over time and that heterogeneity does not appear to diminish with time are perhaps attributable to our use of the symmetric sigmoidal activation function to map forecasts into the unit interval. The symmetric sigmoidal transformation function effectively “squashes” the output of forecast rules so that forecasts always lie within the unit interval. As a result, the forecasts of the various rules and their associated fitness values may not be all that distinct from one another even though the rules themselves may differ considerably. One consequence is that simple forecast rules, e.g. $gp_t = M_{t-i}$, $i = 1$ or 2 , may be unable to effectively compete with more complicated rules (programs with

longer lengths), since these more complicated rules are better able to differentiate themselves from the simpler rules after being squashed, and therefore, these more complicated rules stand a better chance of being chosen for reproduction than the simpler rules.

As an alternative to the symmetric sigmoidal transformation function, we also considered the performance of our GP-based learning algorithm when the simple truncated linear transformation (discussed above) is used in place of the symmetric sigmoidal transformation. The truncated linear transformation is essentially a *linear* mapping into the unit interval whereas the symmetric sigmoidal transformation comprises a nonlinear mapping. Thus, with the truncated linear transformation there is less “squashing” of forecasts and associated fitness values. Indeed, squashing only occurs for forecasts that exceed the bounds of the unit interval; forecasts that lie within the unit interval are unaltered, and therefore remain more distinct (in terms of fitness) than under the symmetric sigmoidal transformation.

We conducted 10 simulations using the *truncated linear transformation function* in place of the symmetric sigmoidal transformation function – 5 simulations of Case 1 and 5 simulations of Case 2.¹⁷ Means and standard deviations from these 10 simulations are reported in Table 4. Here again, simulation 1.1 refers to our first simulation of Case 1, while simulation 2.1 refers to our first simulation of Case 2, and so on. Time series for the mean, M_t , from a single simulation of Case 1 and Case 2 are plotted in Figures 3A–3B. These time series plots are typical of the other simulations we conducted for the two cases using the truncated linear transformation.

From Table 4 and Figures 3A–3B, we see that our use of the truncated linear transformation in place of the symmetric sigmoidal transformation results in several significant differences. First, a comparison between Figures 3A–3B and Figures 1A–1B and between the results in Tables 4 and 3 reveals that after 1,000 generations, the GP learning algorithm with the truncated linear transformation is generally closer to achieving the interior equilibrium than is the GP learning algorithm with the symmetric sigmoidal transformation. This quicker convergence to the interior equilibrium is more consistent with the experimental findings of Van Huyck et al. (1994). Second, we observe that the deviation of the mean from the interior equilibrium, $1 - \frac{1}{\omega}$, in both Case 1 and Case 2 is *much smaller* when we use the truncated linear transformation in place of the symmetric sigmoidal transformation; in all 10 simulations, the average of the means, M_t , from generation 201 to 1,000, i.e. \overline{M}_b , does not deviate from the interior equilibrium value, $1 - \frac{1}{\omega}$ by more than 0.01%.

¹⁷Here we are using an adjustment factor of 0.25 once again to obtain adjusted fitness values.

Figure 3A : The Time Series of the Mean Choice of Action of the Coordination Game

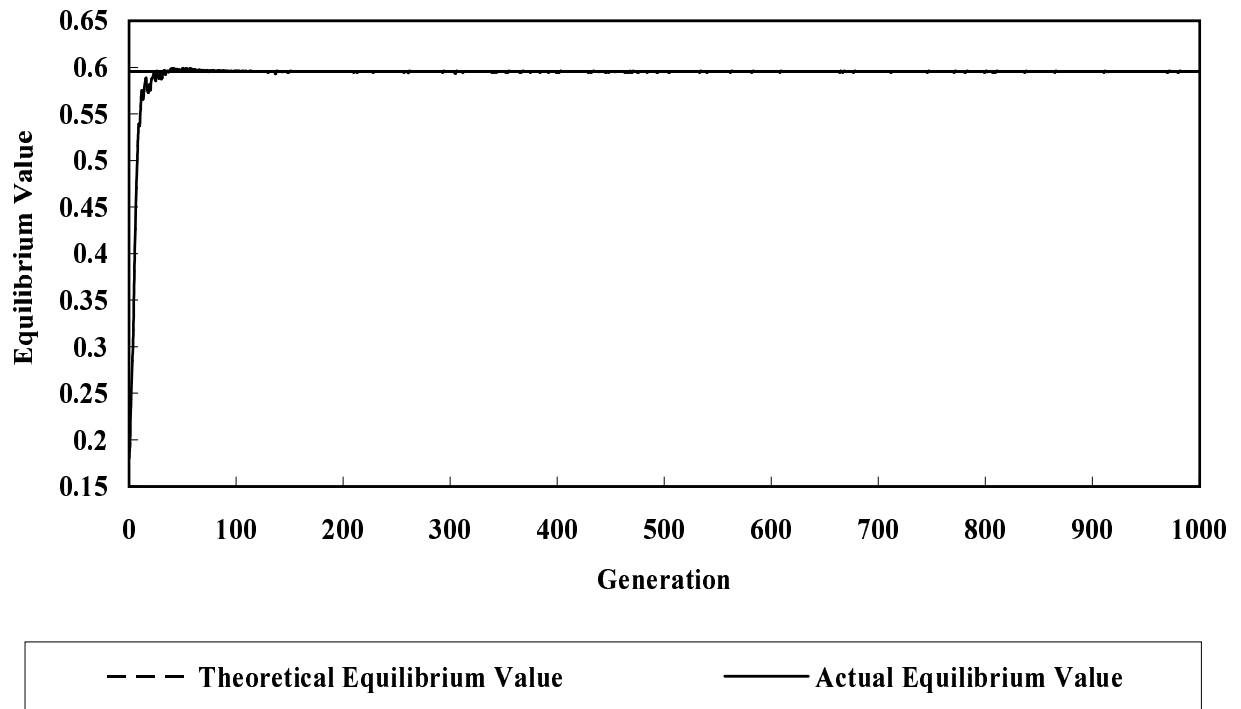


Figure 3B : The Time Series of the Mean Choice of Action of the Coordination Game

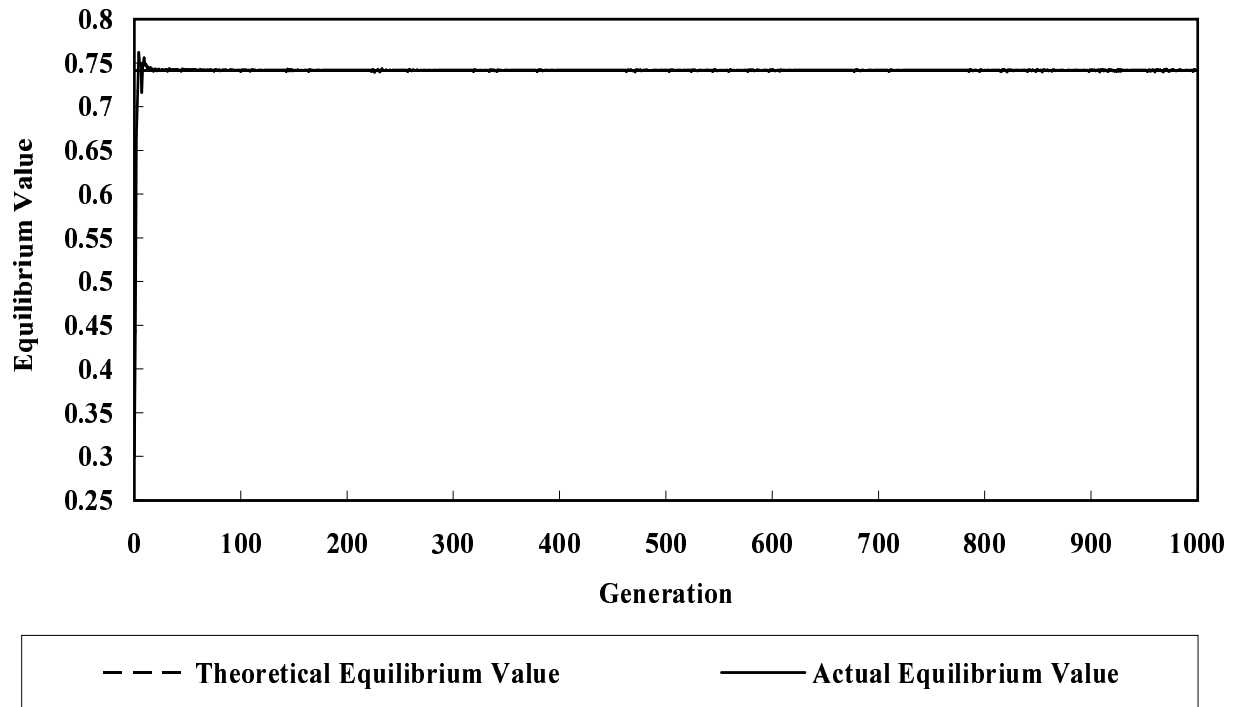


Table 4: GP Simulation Results Using the Truncated Linear Transformation

Case		Simulation				
		1	2	3	4	5
1	\overline{M}_a	0.59281308	0.59279258	0.59328333	0.59295982	0.59249145
	$\delta_{M,a}$	0.02706819	0.02707828	0.02422021	0.02663368	0.02761726
	$\delta_{M^*,a}$	0.02720234	0.02721443	0.02432245	0.02675562	0.02778190
1	\overline{M}_b	0.59543389	0.59544433	0.59546691	0.59546368	0.59545463
	$\delta_{M,b}$	0.00033369	0.00033578	0.00025092	0.00028152	0.00029720
	$\delta_{M^*,b}$	0.00034227	0.00034215	0.00025460	0.00028531	0.00030232
2	\overline{M}_a	0.74079979	0.74081541	0.74069191	0.74081630	0.74080296
	$\delta_{M,a}$	0.01710630	0.01744866	0.01870916	0.01680324	0.01777976
	$\delta_{M^*,a}$	0.01712365	0.01746498	0.01872978	0.01682015	0.01779631
2	\overline{M}_b	0.74154575	0.74154649	0.74155158	0.74155041	0.74154058
	$\delta_{M,b}$	0.00043555	0.00045807	0.00044765	0.00048606	0.00050182
	$\delta_{M^*,b}$	0.00043622	0.00045867	0.00044803	0.00048645	0.00050268

\overline{M}_a = the average of M_t of a simulation from Generation 1 to 1,000.

\overline{M}_b = the average of M_t of a simulation from Generation 201 to 1,000.

$\delta_{M,a}$ = standard deviation about the M_a of a simulation from Generation 1 to 1,000.

$\delta_{M,b}$ = standard deviation about the M_b of a simulation from Generation 201 to 1,000.

$\delta_{M^*,a}$ = standard deviation about the *strict interior equilibrium* $1 - \frac{1}{\epsilon}$ from Generation 1 to 1,000.

$\delta_{M^*,b}$ = standard deviation about the *strict interior equilibrium* $1 - \frac{1}{\epsilon}$ from Generation 201 to 1,000.

This tighter distribution of forecasts around the interior equilibrium is again more consistent with the experimental findings of Van Huyck et al. (1994).¹⁸

As in the case of the symmetric sigmoidal transformation, we find that the chaotic trajectories for the game $\Gamma(3.86957)$ that are predicted by the myopic best response dynamic are not apparent in any of our simulations of Case 2. We also find once again that a comparison of the standard deviations, $\delta_{M,b}$ or $\delta_{M^*,b}$ across Cases 1 and 2 in Table 4 reveals that these standard deviations are slightly larger in Case 2 than in Case 1. A rank order test confirms that both $\delta_{M,b}$ and $\delta_{M^*,b}$ are significantly larger in Case 2 than in Case 1 ($p \leq .01$ in both cases). This difference is also present, though difficult to see, in a visual comparison between Figures 3A and 3B. We conclude that the coordination problem remains somewhat more difficult in Case 2 than in Case 1 regardless of whether we use truncated linear transformation or the symmetric sigmoidal transformation.

Finally, we note that under the truncated linear transformation, the length of the best-of-

¹⁸More direct comparisons between the experimental data and the simulated data from the GP-based learning algorithm are not really possible due to differences in the two experimental designs (e.g. the GP algorithm allows forecasts on the continuum of the unit interval, while the experimental subjects were limited to a finite set of discrete choices).

generation programs are considerably smaller than those discovered under the sigmoidal transformation. Simulation 2.2 (our second simulation of Case 2) is typical of the other simulation results we obtained using the truncated linear transformation. In this simulation, the longest best-of-generation program appeared in generation 2, and had a length of 29:

$$gp_{\text{best},2} = \text{Exp}(\text{Exp}(\text{Exp}(\text{Exp}((M_{t-3} * M_{t-5}) - \text{Exp}M_{t-1}) * M_{t-1}) - (M_{t-2} \% \text{Rlog}(-2.66020 + M_{t-5}))))$$

Following generation 100 of simulation 2.2, no best-of-generation program had a length that exceeded 3. In fact, the best-of-generation programs after generation 100 were always of the simplest form:

$$gp_{\text{best},t>100} = M_{t-i}, \quad i = 1, 2, 3, 4 \text{ or } 5.$$

Given such simple forecast rules, it is easy to understand why the distribution of forecasts becomes more tightly concentrated around the interior equilibrium when we use the truncated linear transformation.

7 Summary and Conclusions

We have considered a simple coordination game where the actions of the individual players are modeled and updated using GP techniques. Our GP-based coordination game allows for a considerably more flexible experimental design than is possible in experiments with human subjects. In particular, we do not have to restrict the choice set to a finite set of discrete actions, and we can have large numbers of players, e.g. $n = 500$. Moreover, players in our genetic programming implementation are explicitly endowed with the ability to formulate a vast number of both linear and nonlinear forecasting rules for the mean, including the myopic best response rule. This more flexible design allows for a possibly dense set of periodic and chaotic trajectories for the mean for values of $\omega > 3$. Despite this more flexible design, the evolution of play in our GP-based coordination game remains quite similar to that observed in the experiments that Van Huyck et al. (1994) conducted with human subjects. The mean choice of action eventually settles down to a small neighborhood of the interior equilibrium, even in Case 2, where the myopic best response dynamic predicts that this interior equilibrium should be unstable. There is evidence however, that the coordination problem that our artificial agents face in Case 2 is somewhat more difficult than the coordination problem they face in Case 1, as indicated by the different standard deviations about the mean/interior equilibrium for these two cases.

While these results cast some doubt on the plausibility of the myopic best response dynamic as a selection criterion (or any other learning schemes that would predict the interior equilibrium to be unstable), it is not yet clear that the myopic best response dynamic should be rejected on the basis of a “bad” prediction for a single game, namely $\Gamma(3.86957)$, or that the alternative, inertial learning algorithm should be accepted as a plausible selection dynamic on the same basis. While the inertial learning dynamic predicts that the interior equilibrium is always stable, the predicted trajectory for the mean/median is much too smooth when compared with the same trajectory from the experimental data. Moreover, the notion that a single, representative-agent-type learning algorithm can be used to characterize the evolution of the mean/median is at odds with the initial heterogeneity that is apparent in the experimental subjects’ actions. Finally, since our GP-based learning algorithm always “converges” to the interior equilibrium it is, by the criterion of Van Huyck et al. (1994), just as plausible a selection dynamic as the inertial learning algorithm. The initial heterogeneity of the forecasts that arise from our population-based GP algorithm makes it all the more plausible as a characterization of the experimental data.

We also note that the predictions of our GP-based learning model, especially those involving the truncated linear transformation, compare quite favorably with some new coordination game experiments that Van Huyck, Battalio and Rankin (1996) have recently conducted with human subjects. These new experiments differ from the previous experiments conducted by Van Huyck et al. (1994) in that subjects are not informed of the game’s payoff function π ; the only information available to subjects is their own past action/payoff history and the discrete action set that they may choose from. The purpose of this new experimental treatment is to place the human subjects in an environment that is as close as possible to that of artificial learning algorithms such as GP. In this new treatment, the human subjects learn to coordinate on the interior equilibrium *even more quickly* than in the previous treatment where subjects are informed of the payoff function π , of the game. Van Huyck, Battalio and Rankin (1996) compare the experimental behavior in the new treatment with the behavior of a representative-agent-type, linear, stochastic reinforcement algorithm. While this algorithm eventually achieves a neighborhood of the interior equilibrium, it takes much longer to achieve this equilibrium (750 iterations) than it takes the experimental subjects. By contrast, our multi-agent GP-based learning algorithm converges much more quickly to a neighborhood of the interior equilibrium (usually within 50 iterations) so that it comes closer to mimicking the behavior of the experimental subjects.

Finally, we note that our findings for the coordination game are consistent with some other coordination experiments that have involved overlapping generations economies. Marimon, Spear and Sunder (1993) for example, report that experimental subjects are unable to coordinate on two-state sunspot equilibria, choosing instead to settle upon the steady state of an overlapping generations economy. Similarly, Bullard and Duffy (1998b) simulate behavior using a genetic algorithm-based learning model in an overlapping generations economy and find that their population of artificial agents is able to eventually coordinate on steady state and low-order cycles for inflation rates but not on the higher order periodic equilibria of their model. This paper extends these earlier findings by suggesting that it may not be possible for agents to coordinate on aperiodic, *chaotic* trajectories.

Appendix A: Lisp S-Expressions and Genetic Programming

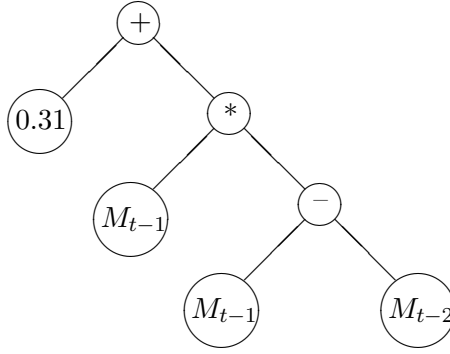
In this appendix, we briefly illustrate how genetic programming techniques are applied to the Lisp symbolic expressions (S-expressions) that serve as the forecast rules for our heterogeneous population of artificial agents.¹⁹ Lisp S-expressions, which are written in the syntax of the Lisp programming language, are immediately implementable computer programs. These S-expressions consist of either *atoms* or *lists*. Atoms include variables, constants, mathematical operators or functions, while lists are collections of atoms or lists, grouped within parentheses, (). Constant atoms (e.g. the constant 2) evaluate to their constant value, while variable or function atoms evaluate to the value taken by the variable or function. The evaluation of lists, such as (+ 2 3) proceeds from left to right. In the expression (+ 2 3), the addition operator, +, is applied to the two constants 2 and 3; this expression yields the value 5. The prefix operator syntax of Lisp – where the operator precedes the operands – allows for considerable flexibility. In particular, this syntax allows an operator such as addition, +, to take on any number of arguments, including zero arguments. For example, the Lisp S-expression (+ 2 (- 1 3) 1) yields the value 1, while (+ 2) yields the value 2, and (+) yields the value 0. The *length* of a Lisp S-expression is determined by counting from the leftmost to the rightmost position the number of elements in the string that makes up the S-expression, including spaces. For example, the Lisp S-expression (+ 2 (- 1 3) 1) has length 15. Generally, the greater the length of a Lisp S-expression, the more complex is the program.

For an example that is more relevant to our application, consider the following S-expression:

$$(+ 0.31 (* (M_{t-1} (- M_{t-1} M_{t-2})))) \quad (1)$$

Here, M_{t-j} denotes the value of the mean action variable at time $t - j$. Note that lagged values of the mean action are included in the *terminal* set that agents are allowed to draw from in assembling their forecast rules, as are floating point constant values drawn from a subset of the real line (\mathfrak{R}): $[-9.99, 9.99]$ (See Table 1). The above Lisp S-expression (program) can also be represented as a rooted, directed *parse tree*:

¹⁹For a more complete description, see Koza (1992, especially Chapter 6).



The *depth* of a parse-tree such as the one depicted above, is defined as the length of the longest path from root to endpoint. Given this definition, it is easy to verify that the depth of the parse tree depicted above is 4.

Algebraically, we would write the Lisp S-expression (1) or the parse-tree depicted above as:

$$0.31 + M_{t-1}(M_{t-1} - M_{t-2})$$

Implementation of expression (1) yields a forecast $gp_{i,t} = 0.31 + M_{t-1}(M_{t-1} - M_{t-2})$. This forecast is then transformed into a valid mean forecast $\widehat{M}_{i,t} \in [0, 1]$ using either the symmetric sigmoidal transformation or the truncated linear transformation as described in the text. This mean forecast $\widehat{M}_{i,t}$ is the action taken by agent the i in round t who possesses forecast rule (1).

Once all n forecasts for period t have been determined, it is possible to calculate the *actual* mean action at time t , $M_t = \frac{1}{n} \sum_{i=1}^n \widehat{M}_{i,t}$. Using the value of M_t it is possible to determine each forecast rule's raw fitness, π_i^t , and to obtain the normalized fitness value $p_{i,t}$ as described in the text.

Given payoffs for each forecast rule, it is possible to apply the first of the genetic operators, reproduction. As discussed in the text, the reproduction operator makes copies of a number of forecast rules that reside in the population of trees at time t , GP_t . The criterion for choosing from among these trees is the normalized fitness value, $p_{i,t}$. Each tree i with normalized fitness value $p_{i,t}$ is chosen by the reproduction operator with probability $p_{i,t}$. Thus, trees with higher normalized fitness values have a higher probability of being chosen for inclusion in the set of forecast rules that make up the population of rules at time $t + 1$. In our application, the reproduction operator is applied 50 times at each date to create 10% (50 out of 500) of the trees that will enter in the population at time $t + 1$. The remaining 450 trees that enter the population at time $t + 1$ are the result of the two other genetic operators, *crossover* and *mutation*. We now describe how these two other genetic

operations are applied to the population of Lisp S-expressions.

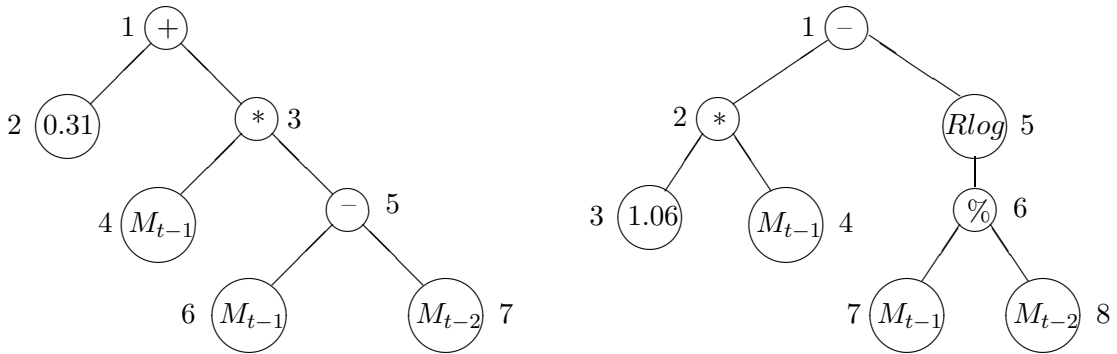
Let us suppose, for illustrative purposes, that the forecast rule described by the S-expression (1) is chosen for crossover based on its normalized fitness value, and is randomly paired with the following S-expression (also chosen for crossover on the basis of its normalized fitness value):

$$(- (* 1.06 M_{t-1}) (Rlog (% M_{t-1} M_{t-2}))) \quad (2)$$

The % sign indicates application of the *protected division* operator; the first argument that follows this operator is divided by the second argument provided that the second argument is not equal to 0. If the second argument is 0, the % operator returns the value 1. Similarly, the *Rlog* operator is a *protected natural logarithm function* that returns the natural logarithm of the absolute value of its argument if this argument is not equal to 0; otherwise, it returns the value 0.²⁰ Algebraically, we would write the S-expression (2) as:

$$1.06M_{t-1} - \ln \left| \frac{M_{t-1}}{M_{t-2}} \right|$$

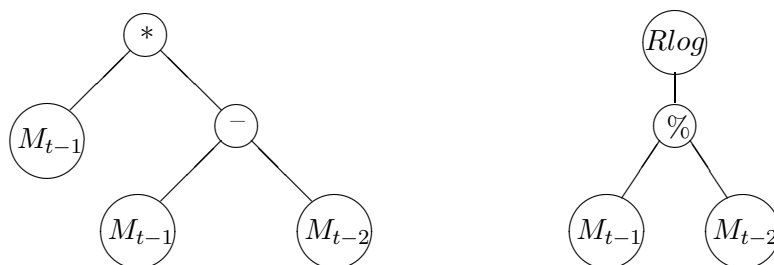
assuming that M_{t-2} and $M_{t-1}/M_{t-2} \neq 0$. The two paired S-expressions (1) and (2) are depicted below in their tree forms, where we have numbered the different atoms (nodes) of each tree in a depth first, left-to-right manner.



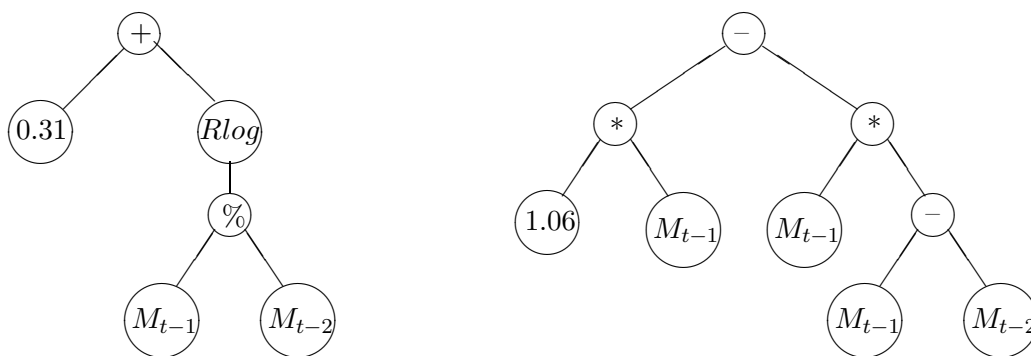
Using the above tree forms for expressions (1–2) enables us to illustrate how the *crossover* operator works. In the first tree, three of the atoms (1, 3 and 5) are *functions* drawn from the function set while four of the atoms (2, 4, 6 and 7) are *terminal values* drawn from the terminal set. (See Table

²⁰Slight modifications of this sort have to be made to ensure that each of the arguments in the function set are able to accept values returned by functions in the function set or by values in the terminal set. This *closure property* of the functions in the function set ensures that any S-expressions that result from genetic operations of crossover and mutation are syntactically legal S-expressions.

1 for the function and terminal sets). Similarly, for the second tree, four atoms (1, 2, 5 and 6) are functions while the other four atoms (3, 4, 7 and 8) are terminal values. In our specification of the crossover operation, the probability that the crossover point occurs at a function atom or at a terminal atom is the same, i.e. it is equal to one-half. Let us suppose that in both trees illustrated above, it is determined that the crossover point should be a function. Then, among the function atoms of the first tree, the atoms (1, 3 and 5) have equal (i.e. one-third) chance of being chosen as crossover points. The four function atoms of the second tree (1, 2, 5 and 6) also have an equal (i.e. one-fourth) chance of being chosen as crossover points. Suppose that in the first tree, function atom 3 is chosen as the crossover point while in the second tree, function atom 5 is chosen as the crossover point. Note that the crossover points need not be the same. The trees are then cut at these two points, yielding the two fragments:



These two fragments are then swapped and recombined at the crossover points of the two trees, resulting in two new trees, (*two new forecast rules*):



Algebraically, the first new rule can be written as:

$$0.31 + \ln \left| \frac{M_{t-1}}{M_{t-2}} \right|$$

while the second new rule would be written as:

$$1.06M_{t-1} - M_{t-1}(M_{t-1} - M_{t-2})$$

In practice, the crossover operation is performed on the Lisp S-expressions themselves rather than on the tree representations of these expressions. We illustrate by considering the same two S-expressions considered above:

$$\begin{aligned} & (+ 0.31 (* (\mathbf{M}_{t-1} (- \mathbf{M}_{t-1} \mathbf{M}_{t-2})))) \\ & (- (* 1.06 M_{t-1}) (\mathbf{Rlog} (\% \mathbf{M}_{t-1} \mathbf{M}_{t-2}))) \end{aligned}$$

All atoms and lists to the right of the randomly chosen crossover points are shown in bold type. The fragments to the right of each crossover point are swapped resulting in two new S-expressions:

$$\begin{aligned} & (+ 0.31 (Rlog (\% M_{t-1} M_{t-2}))) \\ & (- (* 1.06 M_{t-1}) (* (M_{t-1} (- M_{t-1} M_{t-2})))) \end{aligned}$$

Note that the crossover operation will often result in two *offspring* forecast rules that are of different lengths (and complexity) than the original *parent* rules. This difference results from allowing crossover to occur at two different points, and is one way in which genetic programming differs from genetic algorithms, which do not permit such variable length structures.

The two new rules that result from the application of the crossover operation would enter the population of decision rules that yield forecasts in the next round of play, time $t + 1$. In our application, crossover is used to create 70% (350 out of 500) of the trees that enter the population at time $t + 1$. Thus, after application of the reproduction and crossover operations, we have already obtained 80% (400 out of 500) of the trees that will make decisions in the next period. The remaining 20% (100 out of 500) of the next generation of trees are created through application of the *mutation* operator.

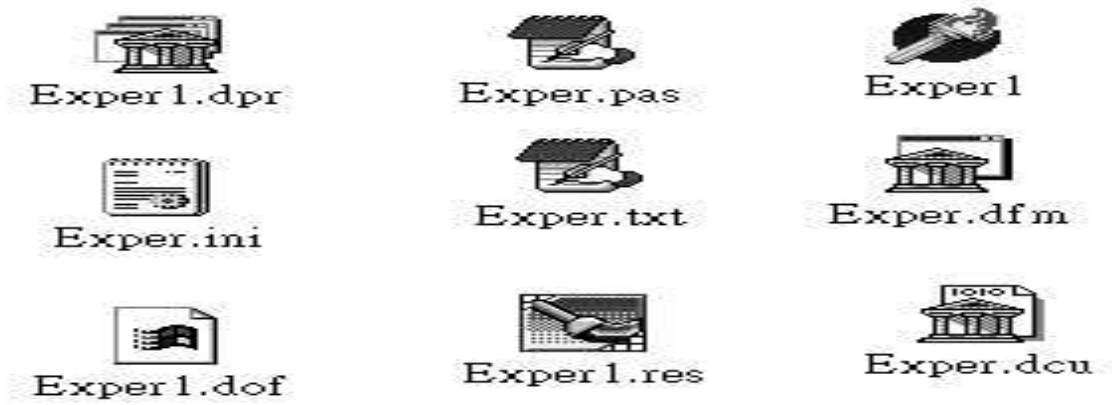
The mutation operator begins by choosing a forecast rule (tree) from the time t population on the basis of normalized fitness values. Given a tree that has been selected for mutation, every atom (node) of that tree is subjected to a small probability of mutation. In our application, the probability of mutation of each atom is fixed at .0033. If an atom is to be mutated, the algorithm first checks whether the atom is a function or a terminal value. If the atom is a function, then it is replaced by another function in the function set, with each of the other functions in this set having an equal

chance of being chosen. If it is a terminal atom, then it is replaced by another terminal value from the terminal set, with each of the other terminal values having an equal chance of being chosen. Once the mutation operator has been applied to every atom of the selected tree, the resulting tree enters the population of trees that yield forecasts in the next round of play, GP_{t+1} .

The initial population of trees, GP_1 is created using one of two methods proposed by Koza (1992, pp. 92–3). In the *full* initialization method, the initial trees have the property that every path from root to endpoint is of full i.e. *maximum* depth. Table 1 indicates that the maximum depth of a tree is set equal to 17. However, for initialization purposes only, the maximum depth of a tree was not allowed to exceed 6. Under the full method, all nodes of the tree that are less than the maximum depth are chosen randomly from the function set only. The final nodes of the tree, at the maximum depth, are chosen randomly from the terminal set. In the *grow* initialization method, initial trees can be of various depths, subject to the constraint that they not exceed the maximum depth. Thus, each node is chosen randomly from either the function or the terminal set; a path ends with the choice of a value from the terminal set. If a member of the terminal set has not been chosen for the node prior to the maximum node allowed, then the final node choice for the path is constrained to be from the terminal set. The initialization scheme we used is the “ramped half and half” method detailed in Koza (1992). Under this scheme, equal numbers of trees are generated using a maximum initial depth that ranges from 2 to 6, so that 20% (100) of all initial trees are generated under the condition that the maximum depth is equal to 2, another 20% are generated under the condition that the maximum depth is equal to 3, etc. on up to a depth of 6. For each of the five maximum depth levels, 50% (50) of initial trees are generated using the full method and the other 50% (50) are generated using the grow method. Thus, under our initialization scheme, the full method is used to create one-half (250) of the initial trees and the grow method is used to create the other half (250) of the initial trees, as we have indicated in Table 1. Following the initial round, trees could achieve a depth greater than 6, up to a maximum depth of 17 via the crossover operation.

The genetic programming algorithm may be summarized as follows. Given an initial population of 500 trees (rules) GP_1 , the forecasts $\widehat{M}_{i,t}$, from each of the rules are determined, and the mean action at time t , M_t is determined. Next, the payoffs for each forecast rule π_t^i are determined from which it is possible to construct the normalized fitness values, $p_{i,t}$ as described in the text. Given the normalized fitness values for all 500 rules, the three operations of reproduction, crossover and mutation are applied to the population GP_1 to obtain, respectively, 10%, 70% and 20% of the

C:\DELPROG\ACE\GAME\GAME3



population of rules that will play the same game in the next round (generation), GP_2 . This process is then repeated until the maximum number of rounds (generations), set at 1,000, has been reached.

Appendix B: Source Code and Extensions

In the coordination game application, the GP method is able to locate and converge upon the interior Nash equilibrium of the game, consistent with the behavior of subjects in Van Huyck et al.'s experiment. However, there are *lots of parameters* in GP, and the results we obtained using the GP method might be sensitive to the particular parameter values we chose. While we know from our own experimentation that our findings are robust to modest changes in the GP parameter values we report in the paper, we have not conducted an exhaustive sensitivity analysis. We leave it to other interested researchers to pursue such an exercise, and provide a link below to the source code we used for this paper.

B.1: Source Code

The software **AIE-GAME** is available from the following web address:

<http://www.aiecon.org/software.htm> The software is composed of 12 files as shown in Figure 4. Among them, **Exper1.exe** is the main executable file. The parameters indicated in Table 1 and Table 2 are stored in the file **Exper.ini** (Figure 5). The dynamics of the game are initialized by using the initial values of the mean choice of action specified in the file **p.txt**. To run the program, the files **Exper1.exe** and **Exper.ini** have to be put under the same directory, and the location of the

```

[Common]
Data_File=D:\WORKAREA\TreeData\DATA\P.TXT
Tree_Path=D:\WORKAREA\TreeData\DATA
Stop_No=0
BinarySet=+\-\*\%\/
UnarySet=Exp\Log\Sin\Cos\
X_Uar=X01\X02\X03\X04\X05\
Const_Item=R

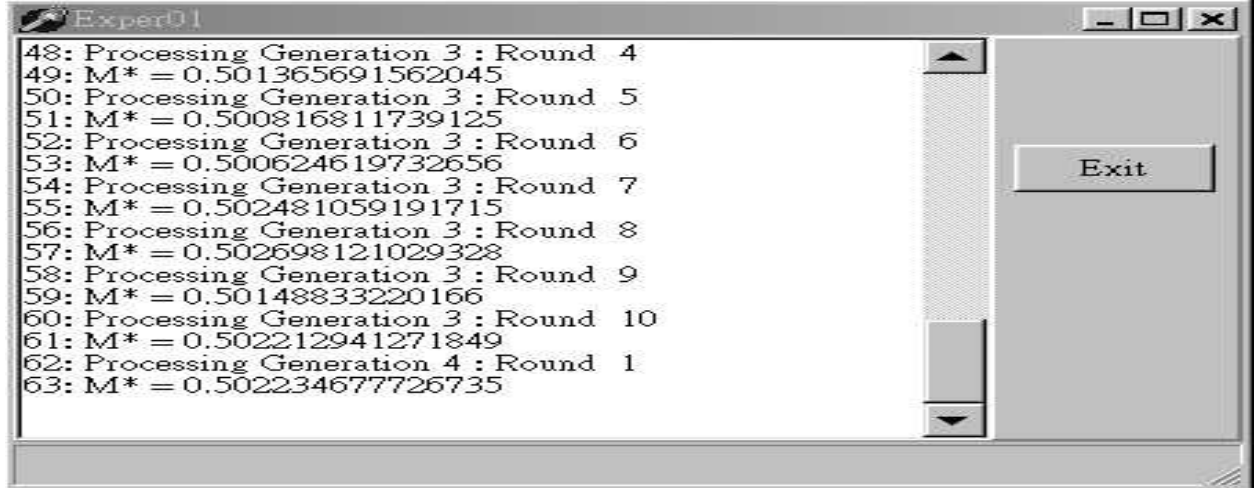
[Traders]
Preserve=0
Reproduct=50
NewBaby=0
Mutation=100
MutProb=3.3
DepthLimit=17
LeaveProb=5
MutMode=0
SelectionMode=0
MaxExp=1700.0
C1=0.5
C2=1.0
Omega=2.47222
MaxNeg=-0.25
Round_No=10
Gen_Number=1000

```

file p.txt should be declared in the Exper.ini (Figure 5). Once the file Exper1.exe is executed, the results are immediately shown on the screen (Figure 6).

AIE-GAME is quite user-friendly because most of the parameters can be directly modified through the file **Exper.ini**. However, if one would like to go further, the file containing the main GP operations is **Exper.pas**. For example, if one would like to modify the fitness function, then one has to click the **Exper1.dpr**, which shall in turn open the *project file*, **Exper.pas**. Then one can move the cursor down to Lines 809 (Figure 7), 1593, and 1525, which are the locations of the fitness function, and change it directly from there.

Depending on the computing environment, running genetic programming can be very time-consuming. For example, for a typical run of 25,000 generations, it takes four hours on Pentium III 650, 256 MB SDRAM, whereas it can take 100 minutes on Pentium 133, 16 MB DRAM even for a run of 1,000 generations. These run-time speeds limit the possibility of large-scale simulation. That is why most of results shown in this paper are based on only 5 runs. Those interested in conducting further experiments are encouraged to download the software and confirm our results. It is our belief



that the main contribution of this paper is not give to provide a final word on the subject, but to provide an innovative platform to facilitate more extensive research in this direction. Below, we give two examples of how this software can be easily modified so as to examine two interesting variants of the coordination game model.

B.2: A Buffer Design

In the evolutionary design adopted in Section 5.2, individual rules are assigned a fitness value based solely on their performance in the *immediate past round* which is used to determine their probability of passing into the next generation, either via reproduction, cross-over or mutation. As a result, the rules are functions whose *validity* is based on a single evaluation of the rules' performance. Inevitably, some rules may survive or die simply because of this single-shot fitness evaluation rule. Alternatively, rule fitness might be evaluated over *several rounds*, rather than on the basis of a *single* trial, so that “genetic operations” of the model are applied only after several rounds, during which the cumulated strength of the rules serve as the fitness measure. Specifically, one may consider the following alternative to the raw fitness measure.

The *raw fitness* of a parse tree $gp_{i,t}$ is determined by the value of the player's payoffs earned over k consecutive rounds, i.e.,

$$\Pi_{i,t} = \sum_{j=0}^{k-1} \pi_{i,t+j}, \quad (3)$$

where $\pi_{i,t}$ is given in equation (1).

With this design, each rule is assigned a “strength”, resulting from the cumulation of past fitness

```


0      10      20      30      40      50      60      70
procedure ComputeMstar;
var
  i : integer;
  Uij : extended;
begin
  Qstar := 0.0;
  for i := 1 to Individual do
    Qstar := Qstar + Qis[i];
  Mstar := Qstar / Individual;

  if Mstar < 0.0 then
    Mstar := 0.0;

  for i := 1 to Individual do
  begin
    Profits[i] := C1 - C2 * Abs(Qis[i] - Omega * Mstar * (1 - Mstar));
    TProfits1[i] := TProfits1[i] + Profits[i];
  end;
end;

```

Ln 809 Col 73 EOL Lines 1792 Insert DOS

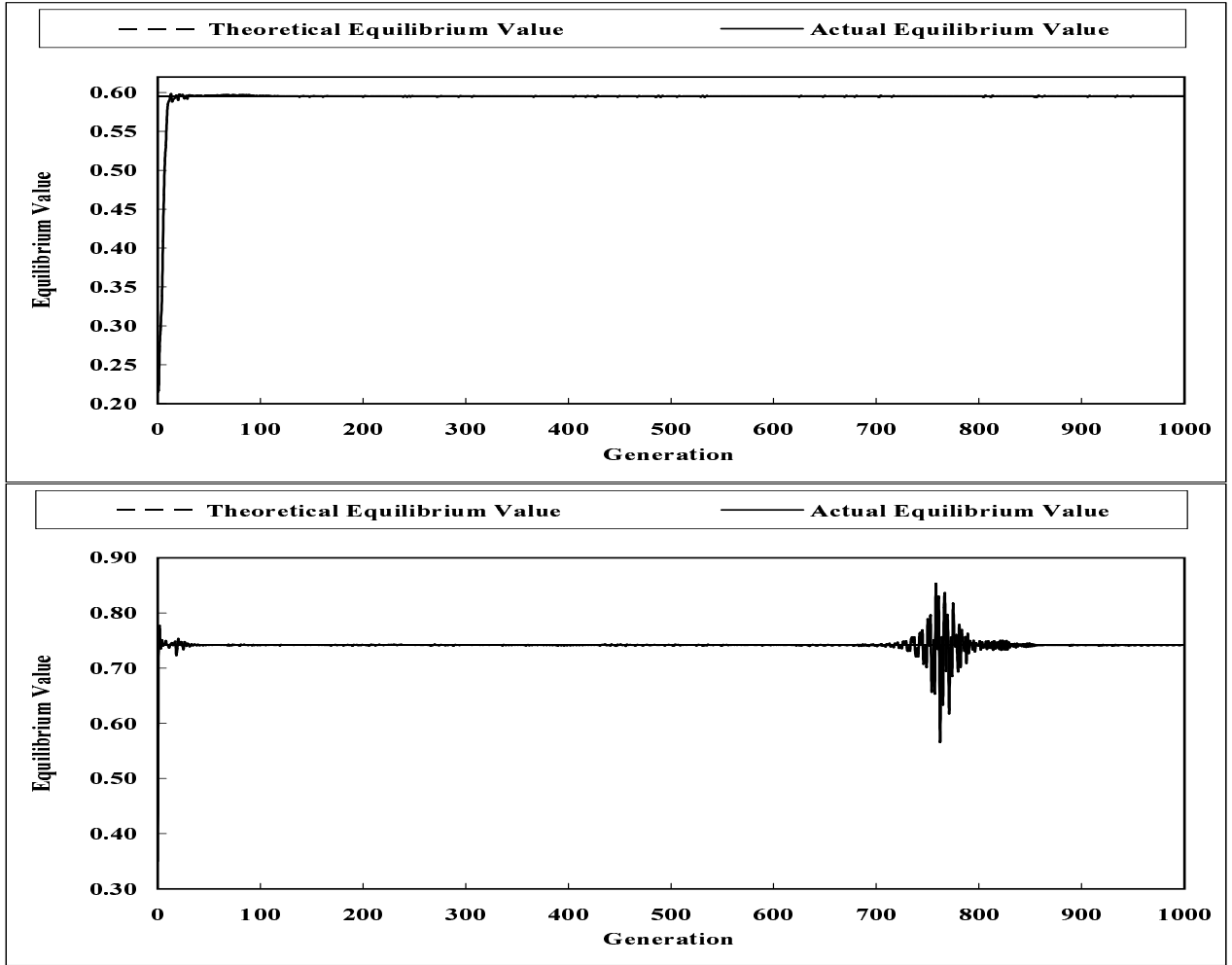


measure over k rounds, and this is used to determine the probability (normalized fitness) of passing to the next generation. The crucial role of the strength is to *buffer* potentially successful rules from temporary shocks, due to exceptional reasons. We consider the *buffer design* an interesting follow-up to this study. In fact, it is quite easy to conduct such experiments with our software. To do so, end-users only need to supply the line “*Round_No*” in the file **Exper.ini** (See Figure 5).

B.3: Catastrophe Payoffs

The GP mechanism proposed in Section 5.2., computes the rule response, then “squashes” it into the unit interval, adjusts the raw fitness to eliminate extremely poor performance, and finally normalizes the resulting values. This mechanism makes it extremely complicated to relate the rules’ structure to a measure of the rule’s effectiveness (fitness). It is, therefore, desirable to simplify the mapping from rule results to rule fitness in order to better understand why we obtain certain rules rather than others.

This problem may be fixed in several different ways. For example, it is not necessary to limit the response to the unit interval. One may use a very punitive reward mechanism (*catastrophe payoff*) for rules providing values outside the unit interval. In this case, rules producing non-sensible results



The upper diagram corresponds to Case 1 defined in Table 2, whereas the lower diagram corresponds to Case 2.

may be immediately removed in favor of other rules. An specific design could be as follows.

$$\pi_{i,t} = \begin{cases} \pi_{i,t} & \text{if } 0 \leq M_{i,t} \leq 1, \\ \pi_{i,t} - 0.25 & \text{otherwise.} \end{cases} \quad (4)$$

In this design, a *catastrophe payoff* -0.25 is assigned to those rules providing values outside the unit interval. End-users can easily choose some other catastrophe value by supplying a different parameter value in the file **Exper.ini** (See Figure 5). We ran a few simulations driven by catastrophe payoffs without further squashing response rules. We see in Figure 8 that our findings can be quite sensitive to this change.

References

- [1] Allen F. and R. Karjalainen (1999). Using Genetic Algorithms to Find Technical Trading Rules. *Journal of Financial Economics*, 51(2), 245–71.
- [2] Angeline, P. (1994). Genetic Programming and Emergent Intelligence. Chapter 4 of Kinnear (1994).
- [3] Arifovic, J. (1994). Genetic Algorithm Learning and the Cobweb Model. *Journal of Economic Dynamics and Control*, 18, 3–28.
- [4] Arifovic, J. (1995). Genetic Algorithms and Inflationary Economies. *Journal of Monetary Economics*, 36, 219–243.
- [5] Arifovic, J. (1996). The Behavior of the Exchange Rate in the Genetic Algorithm and Experimental Economies. *Journal of Political Economy*, 104, 510–541.
- [6] Arifovic, J. (1997). Strategic Uncertainty and the Genetic Algorithm Adaptation. In H. Amman et al. (eds.), *Computational Approaches to Economic Problems*. Boston: Kluwer Academic Press, 225–36.
- [7] Arthur, W.B., J.H. Holland, B. LeBaron, R. Palmer and P. Tayler (1997). Asset Pricing Under Endogenous Expectations in an Artificial Stock Market. In W.B. Arthur et al. (eds.), *The Economy as a Evolving Complex System II*. Reading, MA: Addison-Wesley, 15–44.
- [8] Birchenhall, C.R. (1995). Genetic Algorithms, Classifier Systems and Genetic Programming and Their Use in Models of Adaptive Behavior and Learning. *Economic Journal*, 105, 788–795.
- [9] Bray, M. (1982). Learning, Estimation, and the Stability of Rational Expectations. *Journal of Economic Theory*, 26, 318–339.
- [10] Bullard, J. and J. Duffy (1998). A Model of Learning and Emulation with Artificial Adaptive Agents. *Journal of Economic Dynamics and Control*, 22, 179–207.
- [11] Bullard, J. and J. Duffy (1998). On Learning and the Stability of Cycles. *Macroeconomic Dynamics*, 2, 22–48.

- [12] Chen, S., J. Duffy and C. Yeh (1996). Genetic Programming in the Coordination Game with a Chaotic Best-Response Function. In: *Proceedings of the 1996 Evolutionary Programming Conference*, San Diego, CA.
- [13] Chen, S., and C. Yeh (1997a). Toward a Computable Approach to the Efficient Market Hypothesis: An Application of Genetic Programming. *Journal of Economic Dynamics and Control*, 21, 1043–1063.
- [14] Chen, S., and C. Yeh (1997b). On the Coordination and Adaptability of the Large Economy: An Application of Genetic Programming to the Cobweb Model. In P. Angelino and K.E. Kinnear, Jr., (eds.), *Advances in Genetic Programming II*, Chapter 22. Cambridge, MA: MIT Press.
- [15] Cooper, R., D. DeJong, R. Forsythe and T. Ross (1990). Selection Criterion in Coordination Games: Some Experimental Results. *American Economic Review*, 80, 218–233.
- [16] Crawford, V.P. (1991). An Evolutionary Interpretation of Van Huyck, Battalio and Beil’s Experimental Results on Coordination. *Games and Economic Behavior*, 3, 25–59.
- [17] Crawford, V.P. (1995). Adaptive Dynamics in Coordination Games. *Econometrica*, 63, 103–143.
- [18] Dawid, H. (1996), *Adaptive Learning by Genetic Algorithms*, Lecture Notes in Economics and Mathematical Systems No. 441. New York: Springer.
- [19] Devaney, R.L. (1989). *An Introduction to Chaotic Dynamical Systems*, 2nd Ed.. Reading, MA: Addison–Wesley.
- [20] Dworman, G., S.O. Kimbrough, and J.D. Laing (1996). On Automated Discovery of Models Using Genetic Programming: Bargaining in a Three-Agent Coalitions Game. *Journal of Management Information Systems*, 12, 97–125.
- [21] Goldberg D.E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison–Wesley.
- [22] Holland J.H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Ann Arbor: University of Michigan Press.

- [23] Kagel, J.H. and Roth, A.E. (1995), eds., *Handbook of Experimental Economics*. Princeton, NJ: Princeton University Press.
- [24] Kinnear, K.E. Jr. (1994), (ed.) *Advances in Genetic Programming*. Cambridge, MA: MIT Press.
- [25] Koza, J.R. (1992). *Genetic Programming*. Cambridge, MA: MIT Press.
- [26] Koza, J.R. (1994). *Genetic Programming II*. Cambridge, MA: MIT Press.
- [27] Kreps, D.M. (1990). *Game Theory and Economic Modelling*. New York: Oxford University Press.
- [28] Lucas, R.E. Jr. (1986). Adaptive Behavior and Economic Theory. *Journal of Business*, 59, S401–S426.
- [29] Marcet, A. and T.J. Sargent (1989). Convergence of Least Squares Learning Mechanisms in Self Referential Linear Stochastic Models. *Journal of Economic Theory*, 48, 337–368.
- [30] Marimon, R., (1997). Learning From Learning in Economics. In D.M. Kreps and K.F. Wallis, eds., *Advances in Economics and Econometrics: Theory and Applications*, Vol. 1, Seventh World Congress, Econometric Society Monographs, No. 26, Cambridge: Cambridge University Press.
- [31] Marimon, R., S.E. Spear and S. Sunder (1993). Expectationally Driven Market Volatility: An Experimental Study. *Journal of Economic Theory*, 61, 74–103.
- [32] Miller, J.H. (1996). The Coevolution of Automata in the Repeated Prisoner’s Dilemma. *Journal of Economic Behavior and Organization*, 29, 87–112.
- [33] Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press.
- [34] Neely, C.J., P. Weller and R. Dittmar. Is Technical Analysis in the Foreign Exchange Market Profitable?: A Genetic Programming Approach. *Journal of Financial and Quantitative Analysis*, 32, 405–26.
- [35] Sargent, T.J. (1993), *Bounded Rationality in Macroeconomics*. New York: Oxford University Press.
- [36] Siegel, S. and N.J. Castellan, Jr. (1988). *Nonparametric Statistics for the Behavioral Sciences*, 2nd Ed. New York: McGraw Hill.

- [37] Tesfatsion, L. (1997). A Trade Network Game with Endogenous Partner Selection. In H. Amman et al. (eds.), *Computational Approaches to Economic Problems*. Boston: Kluwer, 249-269.
- [38] Van Huyck, J.B., R.C. Battalio, and R. Beil (1990). Tacit Coordination Games, Strategic Uncertainty and Coordination Failure. *American Economic Review*, 80, 234-248.
- [39] Van Huyck, J.B., R.C. Battalio, and R. Beil (1991). Strategic Uncertainty, Equilibrium Selection Principles and Coordination Failure in Average Opinion Games. *Quarterly Journal of Economics*, 106, 885–910.
- [40] Van Huyck, J.B., J.P. Cook, and R.C. Battalio (1994). Selection Dynamics, Asymptotic Stability, and Adaptive Behavior. *Journal of Political Economy*, 102, 975–1005.
- [41] Van Huyck, J.B., R.C. Battalio, and F.W. Rankin (1996). Selection Dynamics and Adaptive Behavior Without Much Information. Working paper, Texas A&M University.