ISU Economic Report No. 39
revised June 1999

A C++ Platform for the Evolution of Trade Networks[1]

DAVID McFADZEAN
Kumo Software Corporation
1400, 520-5th Avenue S.W., Calgary, AB T2P 3R7 CANADA
david@kumo.com

and

LEIGH TESFATSION
Department of Economics
Iowa State University, Ames, IA 50011-1070
tesfatsi@iastate.edu

Abstract. This paper presents a general C++ platform for the implementation of a *trade network game* (TNG) that combines evolutionary game play with preferential partner selection. In the TNG, successive generations of resource constrained traders choose and refuse trade partners on the basis of continually updated expected payoffs, engage in risky trades modelled as two-person games, and evolve their trade strategies over time. The modular design of the TNG platform facilitates experimentation with alternative specifications for market structure, trade partner matching, trading, expectation formation, and trade strategy evolution. The TNG platform can be used to study the evolutionary implications of these specifications at three different levels: individual trader attributes; trade network formation; and social welfare.

Key words. C++ platform, trade networks, evolutionary game, partner matching, endogenous interactions, agent-based computational economics, artificial life.

1 Introduction

Agent-based computational economics (ACE) is the computational study of economies modelled as evolving decentralized systems of autonomous interacting agents.[2] A key concern

[1]To appear in *Computational Economics*, Special Issue on Programming Languages. For program code and related materials, see http://www.econ.iastate.edu/tesfatsi/. Corresponding author: L. Tesfatsion.

[2]ACE is a specialization to economics of the basic artificial life paradigm (Levy, 1992). See Tesfatsion (1997b) for a brief survey of work in both artificial life and ACE.

of ACE researchers is to understand the apparently spontaneous appearance of global regularities in economic processes, such as the unplanned coordination of trading activities in decentralized market economies that economists associate with Adam Smith's invisible hand. The challenge is to explain how these global regularities arise from the local interactions of autonomous agents channeled through actual or potential economic institutions rather than through fictitious coordinating mechanisms such as imposed equilibrium conditions.

The modelling of evolutionary economies has of course been pursued by many previous researchers. See, for example, the numerous interesting studies surveyed by Witt (1993) and Nelson (1995). In addition, as detailed in Friedman (1991), economists have recently been focusing on the potential economic applicability of evolutionary game theory in which a fixed number of strategy types reproduce in direct proportion to their relative fitness. Nevertheless, tractability issues have generally forced the use of relatively simple types of models which either directly posit aggregate behavioral relations or impose strong restrictions on the potential variability of behavior across agents; see Kirman (1992).

Exploiting the recent advent of more powerful computational tools, ACE researchers such as Duong (1996), Epstein and Axtell (1996), Tesfatsion (1997a), and Vriend (1995) have been able to extend this previous work in three key ways. First, ACE models generally consist of heterogeneous agents who determine their interactions with other agents and with their environment on the basis of internalized data and behavioral rules. Second, a broad range of agent interactions is typically permitted, with predatory and cooperative associations taking center stage along with price and quantity relationships. Third, the evolutionary process is generally expressed by means of genetic (recombination and/or mutation) operations acting directly on agent characteristics. These evolutionary selection pressures result in the continual creation of new modes of behavior and an ever-changing network of agent interactions.

The present paper focuses on the particular ACE model developed by Tesfatsion (1997a) to study the endogenous formation and evolution of trade networks. This model, referred to as the *trade network game* (TNG), extends to an economic setting an earlier model (Ashlock et al., 1996; Stanley et al., 1994) combining evolutionary game play with endogenous partner selection. In the TNG, successive generations of resource-constrained traders choose and refuse trade partners on the basis of continuously updated expected payoffs, engage in risky trades modelled as two-person games, and evolve their trade strategies over time.

The primary purpose of the paper is to present a C++ platform that implements the TNG. This TNG platform is supported by *SimBioSys*, a general class framework for evolutionary simulations developed by McFadzean (1995) that has a number of capabilities important for ACE modelling.

In particular, the TNG platform permits traders to be instantiated as autonomous endogenously interacting software agents (tradebots) with internal behavioral functions and with internally stored information that includes identifiers for other tradebots. The tradebots can therefore display anticipatory behavior (expectation formation), and they can communicate with each other at event-triggered times. The tradebots use these and other platform capabilities to determine their trade partners and to evolve their trade behavior. The modular design of the TNG platform permits experimentation with alternative specifications for market structure, trade partner matching, trading, expectation formation, and trade behavior evolution. All of these specifications can potentially be grounded in tradebot-initiated activities. The TNG platform thus facilitates the general ACE study of trade networks.

The general features of *SimBioSys* are outlined in Section 2. The basic TNG model is outlined in Section 3, making use of the particular TNG module specifications detailed in Tesfatsion (1997a). Section 4 describes how the TNG has been implemented with the support of *SimBioSys*. Section 5 illustrates how the TNG implementation can be used to study evolutionary TNG outcomes at three different levels of analysis: individual trader attributes; trade network formation; and social welfare. Concluding remarks are given in Section 6.

2 SimBioSys: An Evolutionary Simulation Framework

SimBioSys is a general C++ framework[3] for evolutionary simulations developed by McFadzean (1995). The framework permits a user to construct a virtual spatial environment inhabited by any number of evolving populations of autonomous agents. The following discussion emphasizes the particular features of this framework that have been used to implement the TNG.

SimBioSys is designed to handle simulations comprising the following four features:

[3]C++ is a popular object-oriented extension of the C programming language; see, for example, Lippman (1991). The discussion below does not presume any prior knowledge of C++.

- A *world* defining the virtual environment where the simulation occurs;
- *Populations* of autonomous agents inhabiting the world;
- *Programs* driving the behavior of the agents;
- *Genetic mechanisms* emulating natural selection which act on the agents' programs.

Agents are entities capable of displaying some kind of active autonomous behavior. A population of agents of a particular species is represented as a population of computer programs inhabiting a world. This world can comprise other agent species as well as passive objects such as geographically distributed trails, obstacles, and energy sources (food). Agent interactions with each other and with passive objects are driven by their programs. Specifically, the program of each agent takes the agent's perceptions as input and computes an intention for the agent that the world resolves into an action. The evolutionarily significant activities of the agents are birth, death, interactions, migration, and sexual reproduction (genetic recombination and mutation) among agents of each species. These activities are represented as operations acting on the agents' programs.

As a simple example, consider how *SimBioSys* has been used to evolve the foraging behavior of an isolated ant. The behavior of each ant in an initial ant population is driven by a program that implements a finite state machine, i.e., a state transition table that determines an intended action for each possible state of the world. The fitness of each ant is measured by the degree to which, in some prespecified number of time steps, it successfully traverses a winding, broken, increasingly difficult trail in a cellular grid environment. Each ant can sense whether or not the cell just ahead of it constitutes part of the trail. After sensing the cell ahead of it, the ant must take one of four possible actions: move forward one cell; turn right (without moving); turn left (without moving); or do nothing. The programs of the initial ant population are randomly determined. After each ant in the population completes a prespecified number of attempts to traverse the trail, the average fitness of each ant is computed. The programs of the most successful ants are then retained while the programs of the less successful ants are replaced with genetic variations (recombinations and mutations) of more successful programs. The process then repeats.

The static structure of *SimBioSys* is expressed through its class definitions and relationships and the dynamic structure of *SimBioSys* is expressed through its hierarchical simulation cycles. These aspects of *SimBioSys* will now be described in turn.

2.1 *SimBioSys Class Definitions and Relationships*

The programming language C++ is organized around the concepts of class and class derivation. A *class* is a module that contains data members, possibly of different types, together with a set of member functions that operate on this data. A class that contains member functions whose implementation is incomplete is called an *abstract base class*. An abstract base class is thus an incompletely implemented class from which more completely implemented classes can be derived.

For example, one might construct an abstract base class, Insect, whose member functions express general characteristics commonly shared by insects (e.g., foraging behavior) without providing an implementation for these characteristics. Instances of classes derived from Insect, such as Ant and Wasp, could then inherit the member functions of Insect but also provide specialized implementations for them. For example, Ant and Wasp might implement distinct foraging behaviors.

SimBioSys is a C++ class library comprising both abstract base classes and specialized derived classes for constructing worlds, autonomous agent populations, programs, and genetic mechanisms. The static relationships among the various *SimBioSys* classes are depicted in Figure 1. The label "HasA" or "HasSome" on a directed arrow connecting a class A to another class B indicates that class A includes one or more derived class instances of class B as data members; and the label "IsA" on a directed arrow connecting a class A to another class B indicates that A is derived from B.

— INSERT FIGURE 1 ABOUT HERE —

At the highest level, *SimBioSys* represents the simulation as an abstract base class, bioSimulation. This class contains member functions and data for the construction of a world, one or more populations of agents that inhabit the world, and instruments for the design and control of the user interface.

An abstract base class, bioWorld, is responsible for the physics governing the virtual environment of the simulation. Derived class instances of bioWorld implement specific environments, such as a rectangular grid or a torus. An abstract base class, bioPopulation, identifies general data and operations required for the initial construction and genetic reproduction of the agent populations that inhabit the world. For example, bioPopulation includes the size and average fitness of a population as data members, and it defines member functions for setting the size of the population and for sorting the population by fitness.

An abstract base class, bioThing, represents all of the inhabitants of the world. These inhabitants are either passive objects or active autonomous agents. The bioThing class identifies certain general operations common to all inhabitants and provides for the storage and retrieval of the current positions and orientations of the inhabitants.

An abstract base class, bioAgent, is a derived bioThing class that represents the subset of world inhabitants who are agents. This class sets general protocols for communication and interaction among agents, and between agents and passive objects. Each derived class instance of bioAgent constructs a program that allows the represented agent to perceive its local environment and to act in response to this perception. The program thus acts as the agent's brain. An abstract base class, bioProgram, sets general protocols for the communication between an agent and its program. One advantage of separating the function of the program into the class bioProgram is the ability to substitute different implementations, such as finite state machines, artificial neural networks, and Turing machines, without changing any other aspect of *SimBioSys*.

Finally, an abstract base class, bioGType, identifies the basic recombination and mutation operations used in the genetic reproduction of agent populations. These operations act directly on agent genotypes, which are intrinsic characteristics of agents expressed as bit strings.[4] Derived class instances of bioGType implement operations for specific genotypical forms, either haploid (single bit string form) or diploid (double bit string form). A class derived from bioAgent, bioPType, stores an instance of bioGType that is used by bioPopulation to construct an agent's program before the agent is added to the world.

[4]A *bit string* is a sequence of 0's and 1's: for example, $(1,0,1,1)$. A bit string is thus structurally analogous to a biological chromosome. See Goldberg (1989) for a detailed discussion of the use of bit string expressions in genetic algorithms.

```
int main () {
    Initialize world and agent populations;
    For (B = 0,...,BMAX-1) {          // Enter the breeding cycle loop.
        For (E = 0,...,EMAX-1) {      // Enter the environmental cycle loop.
            For (A = 0,...,AMAX-1) {  // Enter the action cycle loop.
                Do agent actions;
            }
            Environmental step;
        }
        Breeding step;
    }
    Return 0;
}
```

Table 1: Pseudo-Code for the SimBioSys Simulation Cycles

2.2 *SimBioSys Simulation Cycles*

As depicted in Table 1, a *SimBioSys* simulation run consists of the execution of a hierarchy of cycle loops. At the top level, a *breeding cycle loop* is executed. Each breeding cycle consists of an *environmental cycle loop* and a *breeding step*, and each environmental cycle consists of an *action cycle loop* and an *environmental step*.

During each action cycle, each agent perceives its local environment, sends its perception to its internal program, and translates the output of the program into a specific intended action. The intended actions of all agents are collected by the world and resolved into realized actions that affect the next state of the world. After the completion of the action cycle loop, the environmental step is executed. During the environmental step, statistical data may be collected and recorded, agent fitnesses may be calculated, and processes independent of agent actions may alter the environment. For example, changes in the "weather" may affect the quantity and geographical placement of energy sources.

After the completion of the environmental cycle loop, the breeding step is executed. During the breeding step, the phenotypes (agent programs) associated with each agent population are sorted by fitness, parent phenotypes are chosen, and selected phenotypes within the population are replaced by genetically altered (recombined and mutated) versions of parent phenotypes. These evolved phenotypes are then assigned to a new generation of agents, a new breeding cycle commences, and the whole process repeats.

Various artificial life applications of *SimBioSys* are reported in McFadzean (1995). For example, *SimBioSys* has been used to simulate the cellular automaton Game of Life, the earlier described model of an isolated foraging ant, and a social foraging ant model in which two distinct types of ants simultaneously compete for a finite supply of food in a cellular grid. As will be seen in the next two sections, the trade network game application presents *SimBioSys* with several new challenges. Most importantly, the process of choosing and refusing potential trade partners requires the traders to display anticipatory behavior and to engage in event-driven communication with each other.

3 The Basic Trade Network Game

This section outlines the basic features of the trade network game (TNG). The implementation of the TNG with the support of *SimBioSys* is taken up in Section 4.

The TNG consists of a collection of traders that evolves over time. As depicted in Table 2, each trader in the initial trader generation is constructed and assigned a random trade strategy. The traders then enter a nested pair of cycle loops.

At the top level, a *generation cycle loop* is executed. Each generation cycle begins with a configuration step during which each trader is configured with various user-supplied parameter values. The traders then enter into a *trade cycle loop*. In each trade cycle the traders undertake three basic activities: the determination of trade partners, given current expected payoffs; the carrying out of potentially risky trades; and the updating of expected payoffs based on any new payoffs received during trade partner determination and trading. At the end of the trade cycle loop the traders enter into an *environmental step* during which information about the individual traders is assessed and printed out. At the end of the environmental step, an *evolution step* is executed during which evolutionary selection pressures are applied to the current trader generation to obtain a new trader generation with evolved trade strategies. This new trader generation then enters into a new generation cycle and the whole process repeats.[5]

The TNG currently uses the particular specifications for market structure, trade partner determination, trade, expectation updating, and trade behavior evolution detailed in (1997a). For completeness, these specifications are reviewed below.

[5]A generation cycle in the TNG corresponds to a breeding cycle in *SimBioSys* and a trade cycle in the TNG corresponds to an action cycle in *SimBioSys*. In the TNG there is no environmental cycle loop; rather, there is a single environmental cycle comprising the trade cycle loop and the environmental step.

```
int main () {
  Init() ;                      // Construct initial trader generation
                                //   with random trade strategies.
  For (G = 0,...,GMAX-1) {      // Enter the generation cycle loop.
                                // Generation Cycle:
    InitGen();                  //   Configure traders with user-supplied
                                //     parameter values (initial expected
                                //     payoff levels, resource quotas,...).
    For (I = 0,...,IMAX-1) {    //   Enter the trade cycle loop.
                                //   Trade Cycle:
      MatchTraders();           //     Determine trade partners,
                                //       given expected payoffs,
                                //       and record refusal and
                                //       wallflower payoffs.
      Trade();                  //     Implement trades and
                                //       record trade payoffs.
      UpdateExp();              //     Update expected payoffs
    }                           //       using newly recorded payoffs.
                                //   Environmental Step:
    AssessFitness();            //     Assess and output trader information.
    Dump();                     //     Output fitness statistics for the
                                //       current trader generation.
    EvolveGen();                //   Evolution Step: Evolve a new trader generation.
  }
  Return 0 ;
}
```

Table 2: Pseudo-Code for the TNG

Alternative market structures are currently imposed in the TNG through the prespecification of buyers and sellers and through the prespecification of quotas on offer submissions and acceptances. More precisely, the set of players for the TNG is the union $V = B \cup S$ of a nonempty subset B of *buyer traders* who can submit trade offers and a nonempty subset S of *seller traders* who can receive trade offers, where B and S may be disjoint, overlapping, or coincident. In each trade cycle, each buyer m can submit up to O_m trade offers to sellers and each seller n can accept up to A_n trade offers from buyers, where the offer quota O_m and the acceptance quota A_n can be any positive integers.

Although highly simplified, these parametric specifications permit the TNG to encompass two-sided markets, markets with intermediaries, and markets in which all traders engage in both buying and selling activities. For example, the buyers and sellers might represent workers and employers, lenders, banks, and borrowers, or barter traders. The offer quota O_m indicates that buyer m has a limited amount of resources (labor time, deposits, apples,...) to offer, and the acceptance quota A_n indicates that seller n has a limited amount of resources (job openings, investment earnings, oranges,...) to provide in return.

Three illustrations are sketched below.

Case 1: A Labor Market With Endogenous Layoffs and Quits

The set B consists of M workers and the set S consists of N employers, where B and S are disjoint. Each worker m can make work offers to a maximum of O_m employers, or he can choose to be unemployed. Each employer n can hire up to A_n workers, and employers can refuse work offers. Once matched, workers choose on-the-job effort levels and employers choose monitoring and penalty levels. An employer fires one of its current workers by refusing future work offers from this worker, and a worker quits his current employer by ceasing to direct work offers to this employer. This TNG special case thus extends the standard treatment of labor markets as assignment problems (Roth and Sotomayor, 1990) by incorporating subsequent strategic (efficiency wage) interactions between matched pairs of workers and employers and by having these interactions iterated over time.

Case 2: Intermediation with Choice and Refusal

The buyer subset B and the seller subset S overlap but do not coincide. The pure buyers in $V - S$ are the depositors (lenders), the buyer-sellers in $B \cap S$ are the intermediaries (banks), and the pure sellers in $V - B$ are the capital investors (borrowers). The depositors offer funds to the intermediaries in return for deposit accounts, and the intermediaries offer loan contracts to the capital investors in return for a share of investment earnings. The degree to which an accepted offer results in satisfactory payoffs for the participants is determined by the degree to which the deposit account and loan contract obligations are fulfilled.

Case 3: A Labor Market with Endogenously Determined Workers and Employers

The subsets B and S coincide, implying that each trader can both make and receive trade offers. Each trader v can make up to O_v work offers to traders at other work sites and receive up to A_v work offers at his own work site. As in Case 1, the degree to which any accepted work offer results in satisfactory payoffs for the participant traders is determined by subsequent work site interactions. Ex post, four pure types of traders can emerge: (1) pure workers, who work at the sites of other traders but have no traders working for them at their own sites; (2) pure employers, who have traders working for them at their own sites but who do not work at the sites of other traders; (3) unemployed traders, who make at least one work offer to a trader at another site but who end up neither working at other sites nor having traders working for them at their own sites; and (4) inactive (out of the work force) traders, who neither make nor accept any work offers.

The determination of trade partners in the TNG is currently implemented using a modified version of the well-known Gale-Shapley deferred acceptance mechanism (Gale and Shapley, 1962). This modified mechanism, hereafter referred to as the *deferred choice and refusal* (DCR) mechanism, presumes that each buyer and seller currently associates an expected payoff with each potential trade partner. Also, each buyer and seller is presumed to have an exogenously given *minimum tolerance level*, in the sense that he will not trade with anyone whose expected payoff lies below this level.

The DCR mechanism proceeds as follows. Each buyer m first makes trade offers to a maximum of O_m most-preferred sellers he finds tolerable, with at most one offer going to any one seller. Each seller n in turn forms a waiting list consisting of a maximum of A_n of the most preferred trade offers he has received to date from tolerable buyers; all other trade offers are refused. For both buyers and sellers, selection among equally preferred options is settled by a random draw. A buyer that has a trade offer refused receives a negative *refusal payoff*, R; the seller who does the refusing is not penalized. A refused buyer immediately submits a replacement trade offer to any tolerable next-most-preferred seller that has not yet refused him. A seller receiving a new trade offer that dominates a trade offer currently on his waiting list substitutes this new trade offer in place of the dominated trade offer, which is then refused. A buyer ceases making trade offers when either he has no further trade offers refused or all tolerable sellers have refused him. When all trade offers cease, each seller accepts all buyer trade offers currently on his waiting list. A trader that neither submits nor accepts trade offers during this matching process receives a *wallflower payoff*, W.

The buyer-seller matching outcomes generated by the DCR mechanism exhibit the usual static optimality properties associated with Gale-Shapley type matching mechanisms (Roth and Sotomayor, 1990). First, any such matching outcome is core stable, in the sense that no subset of traders has an incentive to block the matching outcome by engaging in a feasible rearrangement of trade partners among themselves (Tesfatsion, 1997a, Proposition 3.2). Second, define a matching outcome to be B-optimal if it is core stable and if each buyer matched under the matching outcome is at least as well off as he would be under any other

		Player 2	
		c	d
Player 1	c	(C,C)	(L,H)
	d	(H,L)	(D,D)

Table 3: Payoff Matrix for the Prisoner's Dilemma Game

core stable matching outcome. Then, in each TNG trade cycle, the DCR mechanism yields the unique B-optimal matching outcome as long as each trader has a strict preference order over the potential trade partners he finds tolerable (Tesfatsion, 1997a, Proposition 3.3).

Trades are currently modelled in the TNG as prisoner's dilemma (PD) games. For example, a trade may involve the exchange of a good or service of a certain promised quality in return for a loan or wage contract entailing various payment obligations. A buyer participating in a trade may either cooperate (fulfill his trade obligations) or defect (renege on his trade obligations), and similarly for a seller. The range of possible payoffs is the same for each trade in each trade cycle: namely, L (the sucker payoff) is the lowest possible payoff, received by a cooperative trader whose trade partner defects; D is the payoff received by a defecting trader whose trade partner also defects; C is the payoff received by a cooperative trader whose trade partner also cooperates; and H (the temptation payoff) is the highest possible payoff, received by a defecting trader whose trade partner cooperates. More precisely, the payoffs are assumed to satisfy $L < D < 0 < C < H$, with $(L + H)/2 < C$. The payoff matrix for the PD game is depicted in Table 3.

The TNG traders are currently assumed to use a simple form of learning algorithm to update their expected payoffs on the basis of new payoff information. Specifically, whenever a trader v receives a trade or refusal payoff P from an interaction with a potential trade partner k, trader v forms an updated expected payoff for k by taking a convex combination of this new payoff P and his previous expected payoff for k. In this way, trader v keeps a running tab on the payoff outcomes of his interactions with k.

The trade behavior of each trader, whether he is a pure buyer in $V - S$, a buyer-seller in

$B \cap S$, or a pure seller in $V - B$, is currently characterized by a finite-memory pure strategy for playing a PD game with an arbitrary partner an indefinite number of times, hereafter referred to as a *trade strategy*. Each trader thus has a distinct trading personality even if he engages in both buying and selling activities. At the commencement of each trade cycle loop, traders have no information about the trade strategies of other traders; they can only learn about these strategies by engaging other traders in repeated trades and observing the payoff histories that ensue. Moreover, each trader's choice of an action in a current trade with a potential trade partner is determined entirely on the basis of the payoffs obtained in past trades with this same partner. Thus, each trader keeps separate track of the particular state he is in with regard to each of his potential trade partners.

The evolution of the traders in each generation cycle is meant to reflect the formation and transmission of new ideas rather than biological reproduction. Specifically, successful trade strategies are mimicked and unsuccessful trade strategies are replaced by variants of more successful strategies. As clarified in the next section, this evolutionary process is currently implemented by means of a standardly specified genetic algorithm applied separately to each subpopulation of distinct trader types: pure buyers, buyer-sellers, and pure sellers.

4 TNG Implementation

This section describes the current *SimBioSys* implementation of the TNG—namely, TNG Version 103—hereafter referred to as the TNG platform. The static and dynamic structures of the TNG platform are discussed in turn.

4.1 TNG Class Definitions and Relationships

The static structure of the TNG platform is expressed through definitions and relationships for three principal classes:

- *tngSimulation*, which manages the overall simulation;
- *tngPopulation*, which manages the evolution of the traders;
- *tngTradeBot*, which simulates a single trader, either a pure buyer, a buyer-seller, or a pure seller.

These classes are derived from the *SimBioSys* abstract base classes discussed in Subsection 2.1. Specifically, referring to Figure 1, tngSimulation is derived from bioSimulation, tngPopulation is derived from bioPopulation, and tngTradeBot is derived from bioPType, which in turn is derived from bioAgent. The TNG platform constructs a single instance of tngSimulation, which in turn constructs a single instance of tngPopulation; and tngPopulation then constructs a collection of tngTradeBot instances hereafter referred to as *tradebots*.[6]

In the current implementation of the TNG, the only aspect of a tradebot that evolves over time is its trade strategy for playing iterated prisoner's dilemma games with other tradebots. Consequently, this trade strategy is implemented as the tradebot's program and is constructed as a derived class instance of the *SimBioSys* class bioProgram.

Specifically, each trade strategy is represented as a finite state machine (FSM) with a fixed starting state. The FSMs for two illustrative trade strategies are depicted in Figure 2. The first trade strategy, Tit-for-Two-Tats, is a nice trade strategy that starts by cooperating and only defects if defected against twice in a row. The second trade strategy, Rip-Off, is an opportunistic trade strategy that evolved in an experiment[7] with an initial population of Tit-for-Two-Tats to take perfect advantage of the latter strategy by defecting every other time.

— INSERT FIGURE 2 ABOUT HERE —

The heart of the TNG platform is the representation of each trader as a tradebot, i.e., as an instance of the class tngTradeBot. All of the trade-related activities of a tradebot are implemented as tradebot methods, meaning they are implemented by means of member functions of tngTradeBot.

A schematic description of the internal structure of a tradebot is given in Table 4. Three features of this description are of particular interest. First, the general accessibility of each member function or datum of a tradebot can be controlled by declaring it to be public, protected, or private.[8] Second, institutional constraints regarding the determination of trade

[6]Currently the TNG platform does not exploit the capability provided by the *SimBioSys* abstract base class bioWorld to situate the tradebots in a virtual spatial environment subject both to biological processes (e.g., plant growth) and to physical laws (e.g., conservation of energy).

[7]The experimental discovery of Rip-Off was made by Daniel Ashlock of Iowa State University during the preparation of Stanley et al. (1994).

[8]Technically, however, any tradebot instance can access the private member functions and data of other tradebots within the scope of a tngTradeBot method.

```
class tngTradeBot
{
    Public Access:
        // Internalized Institutional Rules
                Methods for determining my trade partners;
                Methods for conducting my trades;
        // Other Publicly Accessible Methods Used by This TradeBot
                Methods for constructing my trade strategy;
                Methods for updating my expected payoffs;
                Methods for calculating my fitness score.
    Private Access Only:
                Data about myself;
                Data I have recorded about other tradebots;
        // Data permitting this tradebot to communicate with other tradebots
                Identifiers for all tradebots.
};
```

Table 4: Schematic Description of a Tradebot

partners and the conduct of trades are expressed through the form of the tradebot's member functions and data. Third, each tradebot stores an identifier for itself and for each other tradebot, which permits the tradebot to identify itself to other tradebots it interacts with and to pass messages to other tradebots at event-driven times. The particular importance of these features for the implementation of trade activities in the TNG will be clarified below.

4.2 TNG User-Supplied Parameter Values

Prior to running the TNG platform, the user can supply values for the TNG parameters in a configuration file, tng.ini. A sample specification of tng.ini is given in Table 5.

Note that tng.ini currently only accommodates identical offer quotas for all buyer tradebots, identical acceptance quotas for all seller tradebots, and identical initial expected payoff levels for all tradebots. Also the minimum tolerance level for each tradebot does not currently appear in tng.ini; it is hardcoded to 0.

The constraints on the user-supplied parameter values in tng.ini are as follows. GMax and IMax can be any positive integers, RandomSeed can be any unsigned integer, MutationRate can be any nonnegative number strictly less than 1, FsmStates and FsmMemory can be any positive integers, and TraderCount can be any integer greater than 1. SellerCount and BuyerCount can be any positive integers not exceeding TraderCount that sum to a number at least as great as TraderCount. SellerQuota can be any nonnegative integer not exceeding BuyerCount, and BuyerQuota can be any nonnegative integer not exceeding SellerCount.

```
// VIRTUAL ENVIRONMENT PARAMETERS
    GMax = 50                   // Total number of generations.
    IMax = 150                  // Number of trade cycles in each trade cycle loop.
    RandomSeed = 20             // Seed for pseudo-random number generator.
    MutationRate = .005         // GA bit toggle probability.
    FsmStates = 16              // Number of internal FSM states.
    FsmMemory = 1               // FSM memory (in bits) allocated to past move recall.
    TraderCount = 30            // Total number of tradebots.
    SellerCount = 30            // Number of pure sellers and buyer-sellers.
    BuyerCount = 30             // Number of pure buyers and buyer-sellers.
    Elite = 20                  // Number of elite tradebots in each subpopulation.
    RefusalPayoff = -0.6        // Payoff R received by a refused tradebot.
    WallflowerPayoff = +0.0     // Payoff W received by an inactive tradebot.
    BothCoop = +1.4             // Mutual cooperation PD payoff, C.
    BothDefect = -0.6           // Mutual defection PD payoff, D.
    Sucker = -1.6               // Lowest possible PD payoff, L.
    Temptation = +3.4           // Highest possible PD payoff, H.
// TRADEBOT PARAMETERS
    BuyerQuota = 1              // Buyer offer quota.
    SellerQuota = 30            // Seller acceptance quota.
    InitExpPayoff = +1.4        // Initial expected payoff level.
```

Table 5: Sample Specification for the TNG Configuration File

Elite can be any nonnegative integer less than the number of pure buyers (if positive), the number of buyer-sellers (if positive), and the number of pure sellers (if positive). Finally, InitExpPayoff, RefusalPayoff, WallflowerPayoff, BothCoop, BothDefect, Sucker, and Temptation can be any real numbers.

4.3 TNG Dynamic Structure

The TNG platform uses the *SimBioSys* hierarchy of simulation cycles shown in Table 1 to implement the dynamic structure of the TNG shown in Table 2.

The first step executed by the TNG main program, Main(), is the creation of derived instance of the class tngSimulation, called tng. Main() next invokes a tng member function, Init(), which configures the virtual environment with user-supplied parameter values extracted from the configuration file tng.ini and constructs an initial generation of tradebots with randomly determined trade strategies.

The generation cycle loop then commences. At the start of the first generation cycle, Main() invokes a tng member function, InitGen(), that prompts each tradebot to configure itself with user-supplied parameter values extracted from tng.ini. The result is that each tradebot is publicly identified as a pure buyer, a buyer-seller, or a pure seller, and each tradebot is characterized by parameter values in accordance with its type. In particular, in the current TNG implementation, each pure buyer and buyer-seller has an identical offer quota, each pure seller and buyer-seller has an identical acceptance quota, and each tradebot has an identical initial expected payoff level that represents its initial assessment for each of its potential trade partners.

The initial generation of tradebots then successively participates in a trade cycle loop, an environmental step, and an evolution step. During each trade cycle the tradebots determine their trade partners, engage in trades, and update their expectations. During the environmental step the current generation of tradebots is sorted by fitness scores. Finally, during the evolution step, the trade strategies associated with the least fit tradebots are replaced by variants of more successful trade strategies for each trader type. The tradebots then enter into a new generation cycle and the entire process repeats. The implementation of these cycle and step activities will now be described in greater detail.

Implementation of TNG Trade Partner Determination

The determination of trade partners in the TNG platform is executed through the tng member function MatchTraders(). The precondition of MatchTraders() is that each tradebot associates an expected payoff with each of its potential trade partners. The postconditions of MatchTraders() are first, that trade partners have been determined in such a way that no offer or acceptance quota has been violated, and second, that refusal and wallflower payoffs have been recorded by the tradebots that received them.[9]

MatchTraders() currently implements the DCR mechanism outlined in Section 3 with the minimum tolerance level of each tradebot hardcoded to 0 for simplicity. The specialized tradebot member functions that are invoked by MatchTraders() for the implementation of the DCR mechanism are PrepareOffers(), SubmitOffers(), TakeOffer(), AcceptOffer(), and OfferRejected().

MatchTraders() first invokes each buyer's PrepareOffers() member function. This activation prompts each buyer to undertake certain preparatory actions for entering into a trade cycle. For example, each buyer sets the offer status of each of its potential trade partners to

[9]Preconditions and postconditions specify the interface (inputs and outputs) of a member function that must be respected by all implementations of the member function.

NoOffer.

After each buyer has completed its preparations, MatchTraders() invokes each buyer's SubmitOffers() member function. This activation prompts each buyer to make trade offers to tolerable most-preferred sellers with offer status NoOffer, one offer per seller, up to a maximum number of offers as determined by its offer quota. In addition, each buyer changes the offer status of all sellers receiving its trade offers to OfferMade. A buyer communicates a trade offer to a seller by invoking the seller's TakeOffer() member function, which prompts the seller to provisionally add the buyer's trade offer to its current waiting list. During this offer process, each buyer keeps a running count of the current trade offers it has outstanding to sellers so that it never exceeds its offer quota.

After all buyers have completed this first round of trade offers, MatchTraders() invokes each seller's AcceptOffers() member function. This activation prompts a seller to sort the trade offers currently on its waiting list in accordance with expected payoffs, to provisionally accept up to its maximally allowed number of tolerable most preferred trade offers as determined by its acceptance quota, and to refuse the rest. A seller communicates a refusal to a buyer by invoking the buyer's OfferRejected() member function. This activation prompts the buyer to record a refusal payoff from this seller, to update its payoff count and payoff sum with this seller, to change the offer status of this seller to OfferRejected, and to decrement the number of its outstanding trade offers by one.

After all sellers have had a chance to issue refusals, MatchTraders() again invokes each buyer's SubmitOffers() member function. Buyers who have received refusals, and hence who have fewer outstanding trade offers than permitted by their offer quotas, then enter into a new offer round with sellers, and the entire process repeats. After a finite number of offer rounds,[10] a point is reached where buyers make no further trade offers to sellers. Each seller then accepts the trade offers on its current waiting list, and MatchTraders() terminates the

[10]By Proposition 3.1 in Tesfatsion (1997a), the DCR mechanism always terminates on or before offer round MN, where M is the number of buyers (both pure buyers and buyer-sellers) and N is the number of sellers (both pure sellers and buyer-sellers).

matching process.

Implementation of TNG Trades

Trade activity in the TNG platform is executed by the tng member function Trade() with help from the auxiliary tng member functions MediateTrade() and DilemmaPayoff(). The precondition of Trade() is that all trade partners have been determined. The principal postconditions of Trade() are: first, that each pair of trade partners has engaged in a trade; second, that each tradebot has recorded its trade payoffs and its trade partners' actions; and third, that each tradebot has updated its payoff count, payoff sum, and expected payoff for each tradebot with whom it has traded.

In the current TNG implementation, trades are implemented as prisoner's dilemma (PD) games. In particular, Trade() directs each seller to activate a specialized tradebot member function, PlayPD(). This results in the subsequent activation of MediateTrade() for each buyer on the seller's current waiting list. For any particular buyer and seller pair, MediateTrade() activates the member functions GetAction() for both the buyer and the seller. GetAction() queries a tradebot's trade strategy regarding what action it should take in the current trade (PD game). After actions have been retrieved for the buyer and seller, MediateTrade() activates DilemmaPayoff(), which retrieves PD payoffs for the buyer and seller as a function of their actions. This action and payoff information is then passed back to the buyer and seller through invocation of their Buy() and Sell() member functions, respectively, which prompts them to record the actions taken by their trade partners and to update their payoff counts, payoff sums, and expected payoffs for these trade partners.

Implementation of TNG Expectation Updating

The updating of expectations in the TNG platform is executed by the tradebot member function UpdateExp(). For each tradebot, the preconditions of UpdateExp() are that this tradebot has just received a payoff from an interaction with another tradebot, that it has recorded this payoff, and that it has updated its payoff count with the other tradebot to reflect the receipt of the new payoff. The postcondition of UpdateExp() is that this tradebot has updated its expected payoff for the other tradebot.

In the current TNG implementation, UpdateExp() uses a simple criterion filter[11] to implement the direct updating of expected payoffs as new payoffs are obtained. A tradebot's member function UpdateExp() is invoked each time the tradebot receives a payoff from an interaction with another tradebot, whether this payoff is a trade payoff or a refusal payoff.

Suppose, for example, that a tradebot v receives a payoff P from an interaction with another tradebot k, either a refusal payoff R transmitted through OfferRejected() or some type of trade payoff transmitted through Buy() or Sell(). Tradebot v immediately updates its current payoff count with k, $N_v(k)$, via the assignment statement

$$N_v(k) \quad \leftarrow \quad N_v(k) + 1 \ . \tag{1}$$

Tradebot v then invokes its member function UpdateExp(). This activation first results in the updating of $\omega_v(k)$, the *memory weight* that v currently associates with k, via the assignment statement

$$\omega_v(k) \quad \leftarrow \quad N_v(k)/[N_v(k) + 1] \ . \tag{2}$$

The expected payoff $U_v(k)$ that v currently associates with k is then updated via the assignment statement

$$U_v(k) \quad \leftarrow \quad \omega_v(k)U_v(k) + [1 - \omega_v(k)]P \ . \tag{3}$$

Relations (1) through (3) together imply that tradebot v gives equal weight to each payoff that it has obtained in past interactions with tradebot k. Note, also, that an increase in the memory weight $\omega_v(k)$ implies an increase in the weight put on past payoffs relative to the current payoff. Thus, larger memory weights lead to more inertia in the trade partner selection process in the sense that it becomes increasingly difficult for current payoffs to affect the expected payoffs of tradebots as payoff histories accumulate.

The trade strategies that characterize the trade behavior of any two potential trade partners v and k in repeated trades are currently represented as finite state machines. Moreover, trade strategies are restricted to having a finite memory depth, m, for some fixed finite integer m as determined by user specification. That is, conditional on any current FSM state, the action dictated for v in any current trade with k can depend at most on the last m

[11]As detailed in Tesfatsion (1979), a criterion filter is an algorithm for the direct updating of an expected return function on the basis of past return outcomes, without recourse to the usual interim updating of probability assessments via Bayes' rule.

actions of k in previous trades with v.[12] Letting s denote the (finite) number of FSM states, it follows that the sequence of payoffs received by v and k in repeated trades must eventually enter into a cycle of period no greater than $[s * 2^m]^2$, the maximum possible number of distinct joint conditioning configurations for action determination by v and k. The expected payoff (3) thus converges to the true average payoff obtained by v in interactions with k as the number of interactions between v and k becomes arbitrarily large.

Implementation of a TNG Environmental Step

The environmental step in the TNG platform is executed by the tng member function AssessFitness(). The precondition of AssessFitness() is that each tradebot has a positive total payoff count. The postconditions of AssessFitness() are that the fitness score of each tradebot has been calculated, various information about each tradebot has been printed out, and the tradebots have been sorted by their fitness scores.

In the current TNG implementation, AssessFitness() first invokes each tradebot's member function CalcFitness(). This activation prompts the tradebot to calculate and record its fitness score, measured by the total sum of payoffs divided by the total number of payoffs received by this tradebot during the previous trade cycle loop. AssessFitness() next invokes each tradebot's member function Dump(), which prompts the tradebot to print out various information it has recorded concerning each of its potential trade partners. Finally, AssessFitness() invokes a bioPopulation member function SortByFitness(), which currently uses a quick sort routine to sort the current tradebot generation by their fitness scores.

Implementation of a TNG Evolution Step

The evolution of tradebots in the TNG platform is executed by the tng member function EvolveGen(). The precondition of EvolveGen() is that the current generation of tradebots has been sorted by fitness scores. The postcondition of EvolveGen() is that the current generation of tradebots has been replaced by a new generation of tradebots with evolved

[12]Note that the history of actions by v and k in all previous trades determines the FSM state v is currently in with respect to k and hence indirectly can affect the action taken by v in any current trade with k well. In the current TNG implementation, if m is greater than the number of previous trades that v has undertaken with k, then the actions of k in the non-existent trades are assumed to be cooperations. The integer m is determined by the user's specification of FSMmemory (the number of bits allocated to past move recall) in the TNG configuration file.

trade strategies.

In the current TNG implementation, EvolveGen() invokes a bioPopulation member, Breed(), which applies a standardly specified genetic algorithm (GA) to each of the distinct trader type subpopulations (pure buyers, buyer-sellers, pure sellers) of the current tradebot generation.[13] In particular, for each subpopulation, the trade strategies associated with a certain number of the most fit tradebots (the Elite as user-specified in the TNG configuration file tng.ini) are retained unchanged for the next tradebot generation, and the remaining trade strategies are replaced by genetically altered variants of selected parent trade strategies.

To illustrate, suppose tng.ini specifies that each tradebot generation consists of 24 tradebots, all buyer-sellers, and that the elite tradebots are the 16 most fit tradebots and the mutation rate is .005. Then, in each evolution step, eight parent pairs are selected (with replacement) from the current generation of 24 tradebots, where the probability that any particular tradebot is selected to be a parent is proportional to its relative fitness within the current generation. The trade strategy (FSM) associated with each parent tradebot is coded as a bit string.[14] For each parent pair (mom and dad), two bit positions x and y are randomly selected for mom, and the bits in positions x through y for mom are exchanged with those of dad to obtain an offspring bit string of the same length. Each bit in this offspring bit string is then mutated (toggled from 0 to 1 or vice versa) with probability .005.

The result of these genetic operations on the 8 parent pairs is the creation of 8 new offspring bit strings which decode to 8 new offspring trade strategies (FSMs). The trade strategies currently associated with the 8 least fit tradebots in the current generation are then discarded and replaced with these 8 new offspring trade strategies, and the trade strategies currently associated with the 16 elite tradebots in the current generation are retained

[13]See Goldberg (1989) for a detailed discussion of the design and implementation of genetic algorithms.

[14]For example, suppose tng.ini specifies that each FSM has three internal states and a 1-bit memory; see, e.g., the FSM representation of the Rip-Off trade strategy in Figure 2. Then, for any given FSM, one bit is needed for the binary representation of the initial move, either cooperate (1) or defect (0), which always results in a transition to the fixed starting state 1. Also, each of the three possible internal FSM states 1, 2, and 3 has two labelled exit arrows, one determining a current move and next state if the previous move of the current opponent was to cooperate, and one determining a current move and next state if the previous move of the current opponent was to defect. The binary representation of the current move takes one bit and the binary representation of the next state takes two bits. Consequently, letting the identification of the current internal state be determined by ordered position in the bit string, a bit string of length $[1+3\times[2\times(1+2)]] = 19$ is sufficient to represent the FSM. The current TNG implementation is able to reduce this length by making appropriate use of C++ bitwise operations.

unchanged. Each of the 24 tradebots in the next generation is then assigned either an offspring or an elite trade strategy initialized to its fixed starting state.

The elitism, recombination, and mutation operations performed on the bit string encodings for the current generation of trade strategies have the following effects on these trade strategies. Elitism preserves successful move combinations, recombination rearranges blocks of move combinations with relatively little disruption of the blocks, and mutation changes individual moves. Consequently, the behavioral effects of these genetic operations are that successful trade strategies are mimicked, unsuccessful trade strategies are discarded, and new trade strategies are introduced for experimentation.

5 Experimental Study of the TNG

The TNG platform detailed in the previous sections permits the experimental study of TNG outcomes at three different levels of analysis. First, individual tradebot attributes can be directly sampled to assess the evolution of trader types. Second, detailed information concerning who is trading with whom can be collected to track the endogenous formation and evolution of trade networks. Third, the fitness scores attained by the tradebots in each successive generation can be used to assess social welfare.

With regard to the first level of analysis, the only attribute of a tradebot that evolves in the current TNG implementation is its trade strategy. Trade strategies are represented as finite state machines (FSMs), and genetic operations are applied to bit string encodings of these FSMs to evolve the trade strategies over time. These bit string encodings can be directly examined to determine each tradebot's attitude toward strangers and to assess the type of trade behavior that it is potentially capable of expressing in repeated trades.[15]

To illustrate, the FSM representation for a Rip-Off tradebot in Figure 2(b) reveals that a Rip-Off initially defects against any previously untested trade partner but is capable of expressing a wide variety of behaviors in repeated trades. For example, a Rip-Off ends up mutually cooperating with another Rip-Off, repeatedly alternating cooperations and defections with a Tit-for-Two-Tats as in Figure 2(a), and mutually defecting with a trade partner

[15]The feasibility of carrying out the latter assessment in a thorough way of course decreases sharply as the number of states permitted in the FSM representations is increased. In the current implementation the number of FSM states is essentially arbitrary, limited only by hardware-specific memory limits.

that always defects. Consequently, the "type" of trader represented by a Rip-Off is not simple to characterize; the expressed trade behavior of a Rip-Off depends on the trade strategy of its trade partner and the particular trade history that they currently share.

A direct examination of the trade strategies of individual tradebots can therefore yield useful and interesting information concerning their potential trade behavior, information that is not always possible to infer from their expressed trade behavior. In some circumstances, this information may even permit the analytical determination of trade networks, i.e., who will actually end up trading with whom during the course of a trade cycle loop.

For example, Figure 3 depicts the four possible trade networks that can ultimately form among three Rip-Off (Rip) tradebots and two Tit-for-Two-Tats (TFTT) tradebots during the course of an arbitrarily long trade cycle loop, assuming the refusal payoff R is strictly negative and that each tradebot has a buyer offer quota 1 and a seller acceptance quota 4. Each TFTT ultimately submits its trade offers only to other TFTTs, for TFTTs mutually cooperate with each other whereas a Rip defects against a TFTT every other time. Nevertheless, as the exogenously given initial expected payoff level U^o is increased, the TFTTs are ultimately unable to refuse trade offers directed at them by the predacious Rips. The reason is that a TFTT with a high U^o value is overly optimistic with regard to the payoff it initially expects from a Rip. Although the TFTT subsequently lowers this expected payoff each time the Rip defects against it, for sufficiently high U^o values these defections are infrequent enough to guarantee that the expected payoff never falls below the TFTT's zero minimum tolerance level. Consequently, as U^o increases, the Rips are increasingly able to parasitize the TFTTs and hence secure a relatively higher fitness level.[16]

— INSERT FIGURE 3 ABOUT HERE —

In general, however, the ability to choose and refuse trade partners on the basis of continually updated expected payoffs results in such complicated contingencies that the prediction of trade network formation from an *a priori* assessment of individual trade strategies is extremely difficult. It is therefore important to be able to obtain information on the trade networks that form during each TNG simulation run.

To aid in this second level task, at the end of each trade cycle loop the tng member

[16]See Tesfatsion (1997a) for a more detailed analysis of this 5-tradebot TNG.

function AssessFitness() prompts each tradebot to output the data it currently has stored for each of its potential trade partners. This data includes: the sum of payoffs received from interactions with this potential trade partner; the number of payoffs received from interactions with this potential trade partner; the current expected payoff associated with this potential trade partner; the number of trades actually undertaken with this potential trade partner; and, if trades have taken place, a complete ordered list of the trade partner's actions (cooperate or defect) in each of these trades. The user can thus observe who has traded with whom, and how often, during the course of the trade cycle loop, as well as the degree to which the trades have resulted in mutual cooperation, exploitation (successful defection against a cooperating partner), or mutual defection.

Finally, with regard to the third level of analysis, social welfare, various fitness statistics can be calculated for each successive generation of tradebots. For example, in the current implementation of the TNG the minimum, average, and maximum fitness scores attained by the tradebots during the course of each trade cycle loop are calculated and recorded. These fitness scores give a rough idea of the level of social welfare attained by each successive generation of tradebots.

To illustrate, Figure 4 depicts the minimum, average, and maximum fitness scores attained by 50 successive generations of tradebots starting from initially random trade strategies, given the parameter settings in Table 5. These parameter settings were chosen to match as closely as possible the "standard scenario" used as a benchmark case by Ashlock et al. (1996, Table III) for an IPD game with choice and refusal of partners. The latter authors found that rapid convergence to mutual cooperation took place for this benchmark case, and the TNG results qualitatively replicate this finding. The reason for this rapid convergence becomes clear from an inspection of individual strategies: the refusal mechanism helps cooperative players protect themselves against defectors without having to defect themselves, and the choice mechanism helps cooperative players direct their offers to other cooperative types. Consequently, nice players are not as evolutionarily disadvantaged as they tend to be with random or round-robin partner matching.

— INSERT FIGURE 4 ABOUT HERE —

More generally, the fitness scores attained by the tradebots can be used to judge the relative merits of alternative market structures with regard to inducing good social outcomes. For example, preliminary TNG experimental findings reported in Tesfatsion (1997a) suggest that the conventional optimality properties used by economists to evaluate market structures in static contexts, such as core stability and Pareto efficiency, may be inadequate measures of optimality from an evolutionary perspective. In particular, use of the deferred choice and refusal (DCR) partner matching mechanism discussed in Section 3 and Section 4 imposes high transactions costs on the tradebots that these standard measures do not take into account.

6 Concluding Remarks

The TNG platform has been developed to facilitate the study of trade in decentralized market economies from a bottom-up perspective. The key feature of the TNG platform is that traders are modelled as autonomous endogenously interacting software agents ("tradebots") with internally stored data and with internal behavioral functions that can evolve over time.

The platform design is modular and extensible, reflecting the view that the current implementation is only a first step in a long journey to come. Specifically, in the current implementation the tradebots are market situated, but explicit price setting behavior has not yet been introduced. In addition, no attempt has yet been made to exploit the capability of the underlying *SimBioSys* abstract base classes to situate the tradebots within a virtual spatial environment.

In the past, tractability issues have generally forced economists to focus their attention on specialized aspects of economic behavior, without detailed consideration of psychological, sociological, biological, or physical constraints. The recent development of agent-based frameworks such as the TNG platform raise new questions concerning the appropriate bounds of economic analysis, for these frameworks can potentially model social activity from a much more inclusive perspective. Indeed, as stressed in Epstein and Axtell (1996), such frameworks may at last provide a common paradigm for social science as a whole.

References

Ashlock, D., Smucker, M. D., Stanley, E. A., and Tesfatsion, L. (1996). Preferential partner selection in an evolutionary study of prisoner's dilemma. *BioSystems*, 37, 99–125.

Duong, D. V. (1996). Symbolic interactionist modeling: The coevolution of symbols and institutions. In A. Meystel (ed.), *Intelligent Systems: A Semiotic Perspective*, pp. 349–354. National Institute for Standards and Technology.

Epstein, J. and Axtell, R. (1996). *Growing Artificial Societies: Social Science from the Bottom Up*, MIT Press.

Friedman, D. (1991). Evolutionary games in economics, *Econometrica*, 59, 637–666.

Gale, D. and Shapley, L. (1962). College admissions and the stability of marriage, *American Mathematical Monthly*, 69, 9–15.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA.

Kirman, A. (1992). "Whom or What Does the Representative Individual Represent?," *Journal of Economic Perspectives*, 6 (Spring), 117–136.

Levy, S. (1992). Artificial life: A report from the frontier where computers meet biology, Pantheon Books, N.Y.

Lippman, S. B. (1991). *C++ primer*, Second edition, Addison-Wesley.

McFadzean, D. (1995). *SimBioSys: A class framework for evolutionary simulations*, Master's Thesis, Department of Computer Science, University of Calgary, Alberta, Canada.

Nelson, R. (1995). Recent evolutionary theorizing about economic change, *Journal of Economic Literature*, 33, 48–90.

Roth, A. and Sotomayor, M. A. O. (1990). *Two-sided matching: A study in game-theoretic modeling and analysis*, Cambridge University Press, Cambridge.

Stanley, E. A., Ashlock, D., and Tesfatsion, L. (1994). Iterated prisoner's dilemma with choice and refusal of partners, In C. Langton (ed.), *Artificial Life III*, 131–175, Proceedings Volume 17, Santa Fe Institute Studies in the Sciences of Complexity, Addison Wesley.

Tesfatsion, L. (1979). Direct updating of intertemporal criterion functions for a class of adaptive control problems, *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-9, 143–151.

Tesfatsion, L. (1997a). A trade network game with endogenous partner selection, ISU Economic Report No. 36, April 1995, abbreviated version appears as pp. 249–269 in H. Amman et al. (eds.), *Computational Approaches to Economic Problems*, 249–269, Kluwer Academic Publishers, Dordrecht.

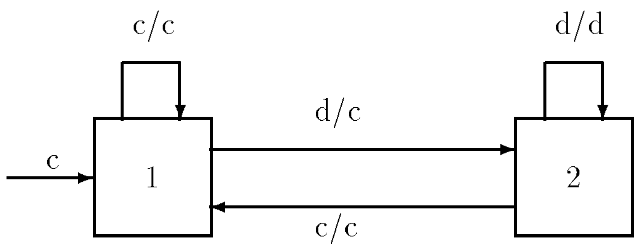
Tesfatsion, L. (1997b). How economists can get alife, In W. Brian Arthur, Steven Durlauf, and David Lane (eds.), *The Economy as an Evolving Complex System, II*, 533–564, Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Volume XXVII, Addison-Wesley.

Vriend, N. (1995). Self-organization of markets: An example of a computational approach, *Computational Economics*, 8), 205–231.

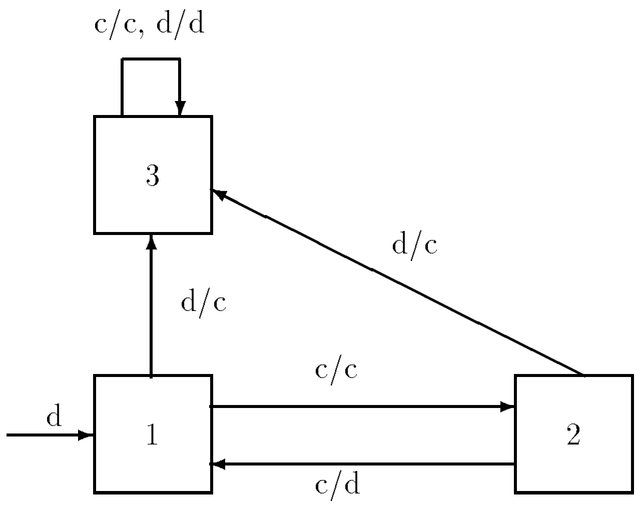
Witt, U. (1993). *Evolutionary Economics*. Edward Elgar, London.

A hard copy of Figure 1 is available from the second author upon request.

Figure 1. Static Structure of *SimBioSys*: Class Relationships



(a) Tit-for-Two-Tats



(b) Rip-Off

Figure 2. The FSM Representations for Two Illustrative Trade Strategies: (a) A nice trade strategy that starts by cooperating and only defects if defected against twice in a row; (b) An opportunistic trade strategy that starts by defecting and defects every other time unless defected against. An arrow label x/y means that y is the next move to be taken in response to x, the last move of one's opponent.

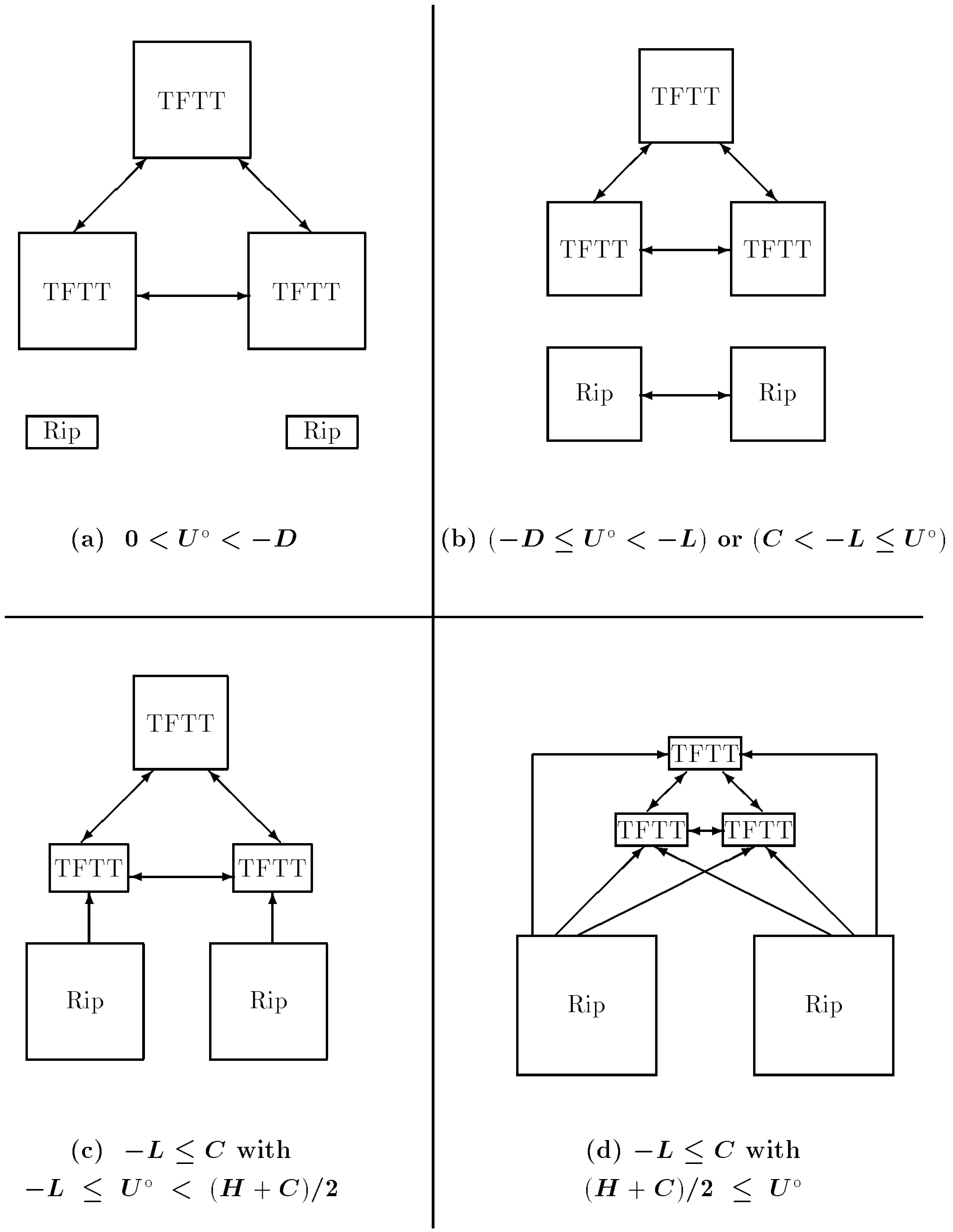

TFTT
TFTT
TFTT
Rip
Rip
(a) $\mathbf{0 < U^\circ < -D}$
TFTT
TFTT
TFTT
Rip
Rip
(b) $(\boldsymbol{-D \leq U^\circ < -L})$ **or** $(\boldsymbol{C < -L \leq U^\circ})$
TFTT
TFTT
TFTT
Rip
Rip
(c) $\boldsymbol{-L \leq C}$ **with**
$\boldsymbol{-L \leq U^\circ < (H+C)/2}$
TFTT
TFTT
TFTT
Rip
Rip
(d) $\boldsymbol{-L \leq C}$ **with**
$\boldsymbol{(H+C)/2 \leq U^\circ}$

Figure 3. Long-Run Trade Networks for an Illustrative 5-Tradebot TNG as a function of the initial expected payoff U^0 and the PD payoffs $L < D < 0 < C < H$. A directed arrow indicates the submission of a trade offer. A relatively larger box indicates a definitely higher fitness score for a sufficiently long trade cycle loop. In case (d), the Rip-TFTT interactions are stochastic if $(H + C)/2 = U^0$ and deterministic if $(H + C)/2 < U^0$.

A hard copy of Figure 4 is available from the second author upon request.

Figure 4. TNG (Version 103) simulation run depicting the minimum, average, and maximum fitness scores attained by 50 successive generations of tradebots starting from initially random trade strategies. By generation 11 the average fitness score closely fluctuates around the mutually cooperative level 1.4. The parameter settings for this run are explained more fully in Table 5.