A Computational Laboratory for Evolutionary Trade Networks

David McFadzean, *Associate Member, IEEE*, Deron Stewart, and Leigh Tesfatsion

Abstract—This study presents, motivates, and illustrates the use of a computational laboratory (CL) for the investigation of evolutionary trade network formation among strategically interacting buyers, sellers, and dealers. The CL, referred to as the Trade Network Game Laboratory (TNG Lab), is targeted for the Microsoft Windows desktop. The TNG Lab is both modular and extensible and has a clear easily operated graphical-user interface. It permits visualization of the formation and evolution of trade networks by means of real-time animations. Data tables and charts reporting descriptive performance statistics are also provided in real time. The capabilities of the TNG Lab are demonstrated by means of labor-market experiments.

Index Terms—Agent-based computational economics, buyer-seller trade networks, C++ class framework, computational laboratory, evolution, labor-market experiments, network animation.

I. INTRODUCTION

AGENT-BASED computational modeling is gaining increased acceptance among social science researchers, as evidenced by the growing number of journal articles and books, designated conferences, and formally instituted research groups that highlight this methodology for social science applications.[1] A key stumbling block, however, is that many social scientists do not know how to get started with computational modeling because they lack a programming background.

Languages such as Starlogo are simple and easily learned, but they are not powerful enough for many social and economic applications. Java and C++ are powerful general-purpose languages, but they are difficult to master. Authoring tools such as AgentSheets, Swarm, and Ascape provide useful repositories of software for building multiagent interactive systems, but their primary appeal is to more experienced programmers.[2]

A computational laboratory (CL) is a framework that permits the computational study of interactions among autonomous structurally differentiated entities by means of controlled and replicable experiments.[3] CLs represent a middle ground between authoring tools and fully customized application software. A CL can present a clear and easily manipulated graphical-user interface (GUI) that allows an inexperienced user to test systematically the sensitivity of a system to changes in a wide variety of key parameters without becoming immersed in implementation details. On the other hand, a CL can be designed to be both modular and extensible. Thus, as users gain more experience and confidence, they can begin to experiment with alternative module implementations to broaden the range of system applications encompassed by the CL.

This study presents a particular CL designed for the study of trade network formation in a variety of market contexts. The CL, referred to below as the Trade Network Game Lab (TNG Lab),[4] comprises buyers, sellers, and dealers who repeatedly search for preferred trade partners, engage in risky trades modeled as noncooperative games, and evolve their trade strategies over time.

The top layer of the TNG Lab consists of a GUI that permits the user to systematically test changes in key market parameter values, e.g., number of traders of each type, capacity constraints, trade payoffs, transaction costs, inactivity costs, learning parameters, and number and length of trading periods. The effects of these market parameter settings on trade network formation are visualized through a real-time animation and the user is able to set animation physics parameters to control this visualization. In addition, data tables and charts are provided that report various market performance measures in real time. This top layer is supported by three lower layers consisting of a general class framework, extension classes, and an event model. These lower layers are extensible and modular, permitting more experienced users to support a much wider range of applications than reflected by the current TNG Lab GUI.

The basic objective of this study is to explain the architecture of the TNG Lab and to demonstrate its capabilities and usefulness by means of illustrative experiments. Another objective,

[1]For a general introduction to agent-based computational social science, see [1]. The online *Journal of Artificial Societies and Social Simulation (JASSS)*, freely available at http://jasss.soc.surrey.ac.uk/, regularly provides surveys of ongoing work in computational social science. See also the website on agent-based computational economics (ACE) at http://www.econ.iastate.edu/tesfatsi/ace.htm. ACE is the computational study of economies modeled as evolving systems of autonomous interacting agents. The extensive resources available at the ACE website include surveys, an annotated syllabus of readings, software, teaching materials, information on conferences and special journal issues, and pointers to individual researchers and research groups.

[2]See http://www.econ.iastate.edu/tesfatsi/acecode.htm for an annotated list of pointers to these and other software tools for constructing multiagent systems.

[3]The felicitous phrase "computational laboratory" was apparently first introduced and formally defined by Dibble [2], a strong advocate for the use of CLs in human geography.

[4]For research papers, tutorials, user instructions, source code, and executables pertaining to the TNG Lab, TNG/SimBioSys, and SimBioSys, visit the TNG home page at http://www.econ.iastate.edu/tesfatsi/tnghome.htm.

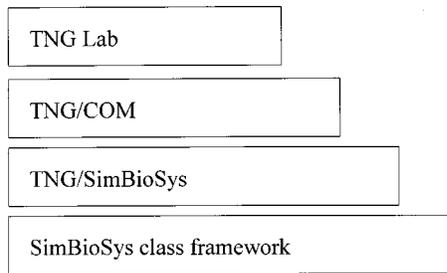| TNG Lab |
| TNG/COM |
| TNG/SimBioSys |
| SimBioSys class framework |

Fig. 1.   Architecture of the TNG lab.

however, is to use the example of the TNG Lab to encourage the routine construction and use of CLs for social science applications. This second objective will require increased interdisciplinary efforts between social scientists and programmers interested in evolutionary computation, efforts which could prove to be highly stimulating and beneficial for both groups.

II. OVERVIEW OF THE TNG LAB ARCHITECTURE

The TNG Lab is constructed in a four-layer architecture. This architecture is illustrated in Fig. 1.

The bottom and most fundamental layer of the TNG Lab is the SimBioSys class framework, a general C++ toolkit for developing simulations involving the evolution of populations of autonomous agents [3]. The basic features of SimBioSys are outlined in Section III. The SimBioSys class framework implements a design pattern for evolutionary simulations, controlling overall dynamics of the system, but it does not specify any detailed or specific behaviors. To create a useful application, the framework must be extended by subclassing several key classes.

The second layer, TNG/SimBioSys, provides extension classes that implement the market protocols and behavioral rules of the trade network game (TNG) [4]. This layer forms a complete application sufficient for generating interesting research results [5], [6], but it lacks a friendly interface. Configuration data is read from an input file and simulation results are captured in output files. The basic features of the TNG and the TNG/SimBioSys layer are described in Sections IV and V.

The third layer, TNG/COM, wraps the simulation functionality in a Microsoft component interface that allows it to be called and controlled by external programs. The component interface also introduces an event model that enables interactive applications. This layer is described in more detail in Section VI. The drawback of using the Microsoft component model is that the framework is no longer platform-independent at this level. This tradeoff was considered acceptable given that ultimately we are targeting the Microsoft Windows desktop.

The fourth and final layer implements a GUI for the simulation. Simulation parameters can be entered in a form-based screen, although they can also still be saved and read from an input file. An animation screen and a physics screen allow the researcher to visualize the simulation dynamics while they evolve and to fine-tune this visualization to the application at hand. A results screen displays simulation output in a table format while a chart screen displays the same data graphically, both in real time. These aspects of the TNG Lab are explained and illustrated in Sections VII and VIII.

III. SIMBIOSYS

As detailed in [3], the SimBioSys class framework is designed to handle simulations comprising the following four features:

1) a *world* defining the virtual environment where the simulation occurs;
2) *populations* of agents inhabiting the world;
3) *programs* driving the behavior of the agents;
4) *evolutionary mechanisms* emulating natural selection that act on the agents' programs.

Agents are entities capable of displaying some kind of active autonomous behavior. A population of agents of a particular type is represented as a population of computer programs. In addition to multiple populations of agents, the world can also include passive entities such as spatially distributed trails, obstacles, and energy sources.

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 entities 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 interactions among agents and for interactions between agents and passive entities. 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 program implementations, such as finite-state machines (FSMs), artificial neural networks, and Turing machines, without changing any other aspect of SimBioSys.

Finally, an abstract base class, bioGType, identifies the basic elitism, recombination, and mutation operations used in the genetic reproduction of agent populations. These operations act directly on agent genotypes, which are intrinsic characteristics of

```
int main () {
    Initialize world and agent populations;
    For (G = 0,...,GMax-1) {      // Enter generation cycle loop.
        For (E = 0,...,EMax-1) {      // Enter environment cycle loop.
            For (A = 0,...,AMax-1) {   // Enter action cycle loop.
                Do agent actions;
            }
            Environment Step;
        }
        Evolution Step;
    }
    Return 0;
}
```

Fig. 2.   SimBioSys simulation cycles.

agents expressed as bit strings. Derived class instances of bioG-Type 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.

As depicted in Fig. 2, applications built using the SimBioSys framework typically execute three nested simulation loops: generation cycles contain environment cycles, which in turn contain action cycles. In each action cycle, every agent in the simulation is given the opportunity to make a single move based on its current state and its perception of the environment. After a set number of action cycles, an environment step gives the environment an opportunity to change the state of the world. After a set number of environment cycles, an evolution step is executed during which each agent's relative fitness is evaluated and new populations of agents are separately generated (by type) based on the previous populations. It is this last step that introduces evolutionary change into the simulation.

IV. TRADE NETWORK GAME

This section briefly outlines the basic features of the TNG. A more extensive discussion of these features can be found in [4].

The TNG models the formation and evolution of trade networks among heterogeneous buyers, sellers, and dealers strategically interacting within a market context. The TNG differs in four essential respects from standard market models in economics.

First, the TNG is a process model whose structure at each point in time is given by the internal states and behavioral rules of the traders rather than by a system of demand, supply, and equilibrium equations. The TNG traders must act in accordance with physical feasibility constraints and accounting identities by construction. However, the only equations that explicitly appear in the TNG are those used by the traders themselves to represent aspects of their world and to implement their behavioral rules.

Second, the TNG traders continually adapt their behavior in response to interactions with other traders and with their environment in an attempt to satisfy their needs and wants. That is, behavioral rules are state conditioned and the traders coadapt their behavior in an intricate dance of interactions. The TNG can therefore exhibit self-organization.

Third, the evolutionary process is represented in the TNG as natural selection pressures acting directly on trader attributes rather than as population-level laws of motion. These natural selection pressures induce the TNG traders to engage in continual open-ended experimentation with new rules of behavior, i.e., the TNG traders coevolve.

Fourth, starting from given initial conditions, all events in the TNG are contingent on trader-initiated interactions and occur in a path-dependent time line. Consequently, the market system described by the TNG develops over time in a manner analogous to the growth of a culture in a petri dish.

The TNG accommodates three distinct trader types: 1) *(pure) buyers*, who only engage in buying activities; 2) *(pure) sellers*, who only engage in selling activities; and 3) *dealers*, who can engage in both buying and selling activities. Buyers can only buy from sellers or dealers and sellers can only sell to buyers or dealers, but dealers can buy from sellers, sell to buyers, or buy or sell with each other.

Alternative market structures are imposed in the TNG by prespecifying the number of traders of each type, together with their capacity (resource) constraints. For example, a two-sided market is obtained if the number of dealers is set to zero, an intermediary market is obtained if all three trader types are present, and an "endogenous-type" market is obtained if every trader is a dealer who can switch from buying to selling or vice versa as the current situation warrants. In the two-sided market, the buyers and sellers might represent workers with limited amounts of labor time and employers with limited job openings.[5] In the intermediary market, the buyers might represent lenders (bond purchasers) with limited funds, the dealers might represent financial firms with limited service capacity, and the sellers might represent borrowers (bond suppliers) with limited collateral. In the endogenous-type market, the traders might be resource-constrained agents who must each decide whether to become a firm (hire workers) or to work for others.

Each trader in the TNG is modeled as an autonomous agent with internalized social norms (market protocols), internally stored state information, and internal behavioral rules. Although each trader has this same general internal structure, trader types can differ from each other in terms of their specific market protocols, fixed attributes, and initial endowments and each trader can acquire different state information and evolve different trade behavioral rules over time on the basis of its own unique past experiences.

Activities in the TNG are divided into a sequence of *generations*. Each trader in the initial generation is assigned a rule ("personality") governing its behavior in its trade interactions, an initial expected utility assessment for each of its potential trade partners, and a capacity constraint on the number of trade offers it can make or accept at any given time depending on the trader's type. The traders then repeatedly engage in three types of activities for a certain specified number of rounds: 1) a search for, and determination of, preferred trade partner matches on the basis of current expected utility assessments; 2) trade interactions with trade partners, modeled as noncooperative games; and 3) an updating of expected utility assessments to take into account any newly incurred search costs, inactivity costs, and trade payoffs. The traders of each type then separately evolve

---

[5]This example is used in Section VIII to demonstrate the capabilities of the TNG Lab.
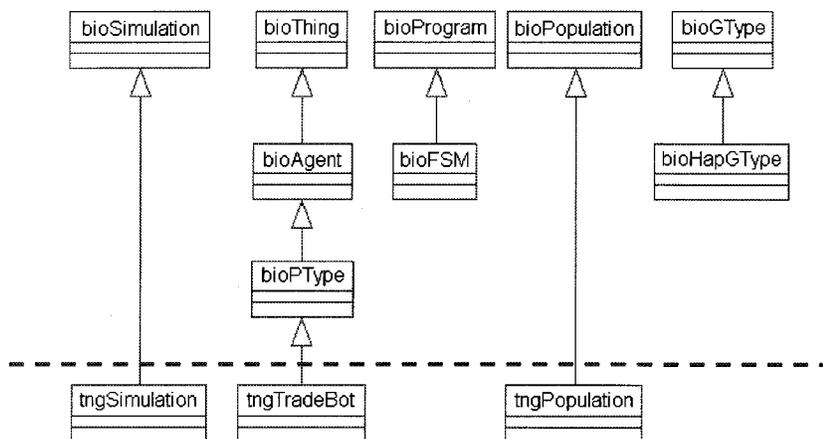
Fig. 3.   TNG/SimBioSys class structure.

(structurally modify) their trade behavioral rules based on the past net payoff outcomes secured with these rules and a new generation commences.

V. TNG/SIMBIOSYS

This section briefly outlines how the activities of the TNG traders are implemented with the support of SimBioSys. A more detailed discussion of the resulting TNG/SimBioSys class framework can be found in [5].

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

1) *tngSimulation*, which manages the overall simulation;
2) *tngPopulation*, which manages the evolution of the traders;
3) *tngTradeBot*, which simulates a single trader (either a buyer, a seller, or a dealer).

These classes are derived from the SimBioSys abstract base classes discussed in Section III. Specifically, as depicted in Fig. 3, tngSimulation is derived from bioSimulation, tng-Population is derived from bioPopulation, and tngTradeBot is derived from bioPType, which in turn is derived from bioAgent. TNG/SimBioSys constructs a single instance of tngSimulation, which in turn constructs a single instance of tngPopulation, and tngPopulation then constructs a collection of traders as tngTradeBot instances.[6]

The key aspect of TNG/SimBioSys is the representation of each trader as a *tradebot*, i.e., as an instance of the class tng-TradeBot. A schematic description of the internal structure of a tradebot is given in Fig. 4. Three features of this description are of particular interest.

First, social norms (market protocols) regarding the determination of trade partners and the conduct of trades are expressed as member functions of tngTradeBot that are commonly inherited and implementated by all tradebots of a given type. Second, additional aspects of the trade behavior of each tradebot are expressed as individualized behavioral rules, i.e., as member functions of tngTradeBot inherited by the tradebots whose imple-

[6]As developed to date, TNG/SimBioSys 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).

```
class tngTradeBot
{
  Internalized Social Norms:
    Market protocols for communication;
    Market protocols for search and matching;
    Market protocols for trade interactions.
  Internal Behavioral Rules:
    My rules for gathering and processing information;
    My rule for determining my trade behavior;
    My rule for updating my expected utility assessments;
    My rule for measuring my utility (fitness) level;
    My rules for modifying my rules.
  Internally Stored State Information:
    My attributes;
    My endowments;
    My beliefs and preferences;
    Addresses I have for myself and other tradebots;
    Additional data I have about other tradebots.
};
```

Fig. 4.   Internal structure of a tradebot.

mentations can differ from one tradebot to another both within and across tradebot types. These differences can occur both through tradebot-specific initial configurations and through evolutionary change. Third, each tradebot stores addresses for other tradebots. This permits each tradebot to identify itself to other tradebots it interacts with and to pass messages to other tradebots at event-driven times.

In principle, all of the behavioral rules of a tradebot could be subject to evolutionary selection pressures. As developed to date, however, TNG/SimBioSys only permits the evolution of each tradebot's rule for determining its trade behavior.

The dynamic structure of TNG/SimBioSys is depicted in Fig. 5. The simulation begins with an initialization step during which each tradebot is constructed, assigned a randomly specified trade behavioral rule, and configured with various user-supplied parameter values according to its type (buyer, seller, or dealer). For simplicity, in the current implementation of TNG/SimBioSys, it is assumed that traders of each type are identically configured. Thus, all buyers are configured with the same parameter values and similarly for all sellers and all dealers. The tradebots then enter into a *generation-cycle loop* comprising three types of events: a "trade-cycle loop," an "environment step," and an "evolution step."

```
int main () {
  Init();                     // Construct initial tradebot
                              //   populations (buyers, sellers,
                              //   and dealers) with randomly
                              //   specified trade rules, and
                              //   configure each tradebot
                              //   with user-supplied parameter
                              //   values (initial expected
                              //   utility levels, capacities,...).
  For (G = 0,...,GMax-1) {    // Enter generation cycle loop.
                              // Generation Cycle:
    For (A = 0,...,AMax-1) {  //   Enter trade cycle loop.
                              //   Trade Cycle:
      MatchTraders();         //     Determine trade partners,
                              //       given expected utility
                              //       levels, and record search
                              //       and inactivity costs.
      Trade();                //     Implement trades and
                              //       record trade payoffs.
      UpdateExp();            //     Update expected utility levels
    }                         //       using newly recorded
                              //       costs and trade payoffs,
                              //       and begin new trade cycle.
                              //   Environment Step:
    AssessFitness();          //     Tradebots assess their fitnesses.
    EvolveGen();              //   Evolution Step:
                              //     Tradebot populations separately
                              //       evolve their trade rules, and
                              //       a new generation cycle begins.

  }
  Return 0;
}
```

Fig. 5. TNG/SimBioSys simulation cycles.

The *trade-cycle loop* consists of a user-specified number of successive trade cycles. In each trade cycle, the tradebots undertake three basic activities: 1) a search for and determination of preferred trade partners, given current expected utility levels; 2) trade interactions with trade partners, modeled as noncooperative games; 3) and the updating of expected utility levels based on any new costs incurred and/or payoffs received during trade partner determination and trading.

At the end of the trade-cycle loop, the tradebots enter into an *environment step*. In this step, each tradebot assesses its fitness, measured as the total payoff (net of costs) that it earned during the preceding trade-cycle loop.

At the end of the environment step, an *evolution step* is executed. In this step, each member of each distinct tradebot population (buyers, sellers, or dealers) evolves its trade behavioral rule. Specifically, in addition to engaging in inductive learning by experimentation with the use of new trade behavioral rules, each tradebot also engages in social learning by mimicking aspects of the behavioral rules used by more successful tradebots of its own type. Experimentation and mimicry for each tradebot type are currently implemented by means of a genetic algorithm (GA) involving standardly specified elitism, mutation, and recombination operations.[7]

---

[7]As currently implemented, all buyers in TNG/SimBioSys have identical structural attributes apart from their evolving trade behavioral rules, and similarly for sellers and dealers. Social learning is then implemented by having each tradebot mimic the trade behavior of other successful tradebots of the same type. Since each tradebot in TNG/SimBios is uniquely tagged and tracked throughout each simulation run, however, more general structural specifications and learning implementations are possible. Note that the usefulness of mimicry as a learning mechanism is substantially reduced in market contexts in which the traders within each trader type have distinct structural attributes (e.g., differentiated capacities, payoffs, or costs). Consequently, for such applications, the current implementation of TNG/SimBioSys should be modified to permit the tradebots to engage in individual learning on the basis of their own unique past experiences.

```
int main () {
  Init();
  FireSimRunning();
  For (G = 0,...,GMax-1) {
    FireGenerationBegin();
    For (A = 0,...,AMax-1) {
      FireTradeCycleBegin();
      MatchTraders();
      Trade();
      UpdateExp();
      FireTradeCycleEnd();
    }
    AssessFitness();
    EvolveGen();
    FireGenerationEnd();
  }
  FireSimFinished();
  Return 0;
}
```

Fig. 6. TNG/COM main loop pseudocode with event firing.

At the end of the evolution step, each evolved tradebot updates its initial expected utility assessment for each of its potential trade partners. It does this by taking a weighted average of the expected utility it assigned to this potential trade partner at the beginning of the latest generation and the expected utility it currently assigns to this potential trade partner. The memory of each evolved tradebot is wiped clean apart from its tag identifier (name) and its updated initial expected utility assessments for its potential trade partners. The three evolved tradebot populations then enter into a new generation cycle and the whole process repeats.

As seen by comparing Fig. 2 with Fig. 5, a generation-cycle loop in TNG/SimBioSys corresponds to a generation-cycle loop in SimBioSys and a trade-cycle loop in TNG/SimBioSys corresponds to an action-cycle loop in SimBioSys. However, there is no environment-cycle loop in TNG/SimBioSys. Rather, for each generation, there is in effect a single environment cycle consisting of a trade-cycle loop, an environment step, and an evolution step.

VI. TNG/COM

In the two-layer implementation (TNG/SimBioSys), the cycles are executed without interruption, generating a batch output of simulation results. One challenge of wrapping the functionality in a component layer was to introduce an event model into the cycle dynamics that would enable a graphical front-end application to display simulation results in real time and interactively.

The introduction of the event model was accomplished by replacing the main SimBioSys loop with another implementation that fires events at key points in the hierarchy of cycles. Pseudocode for the TNG/COM main loop with event firing is depicted in Fig. 6. For simplicity, the events that signal that the simulation has been paused or stopped have been omitted from Fig. 6. In the actual TNG/COM code, the inclusion of these events is accomplished by splitting the main() function into event handlers.

The calls to FireXxx() raise a corresponding event Xxx, which is (optionally) handled by a controlling program, in this case, the TNG Lab. Other substantially different interactive applications could also be built on the TNG/COM.
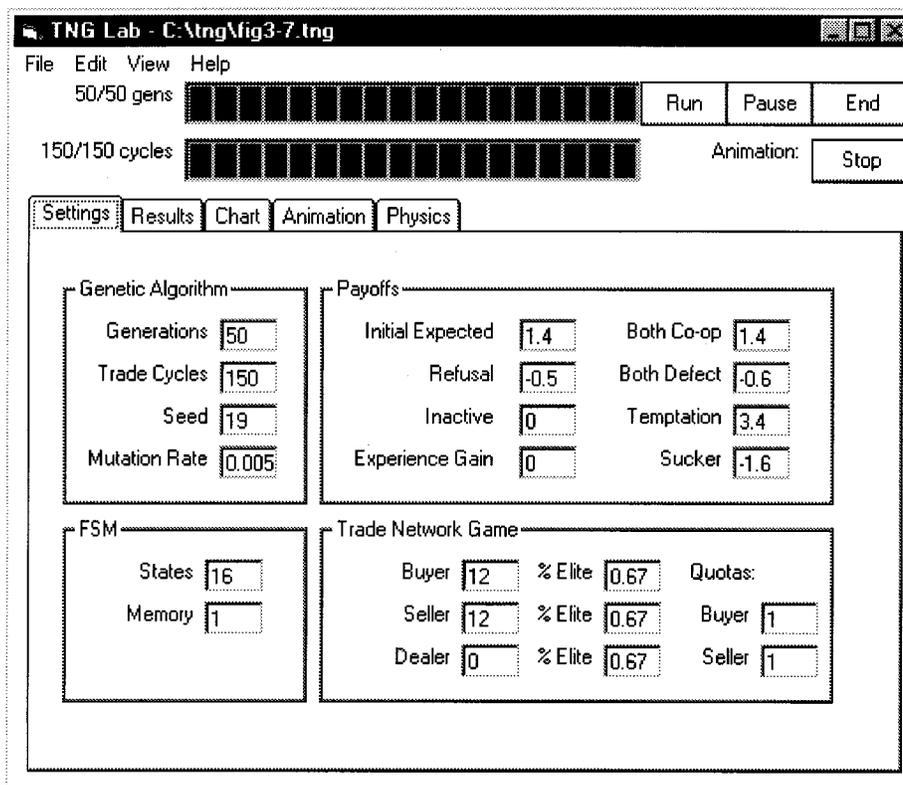
Fig. 7.   TNG lab GUI settings screen.

When the events fire at the TNG/COM level, control is passed back up to the higher level application, giving it an opportunity to update its interactive displays and/or act on user input such as a menu selection or a pushbutton press to pause the simulation. As can be observed in the Fig. 6 pseudocode, an appropriate event is fired at the beginning and end of each major cycle. The SimRunning event is fired after the simulation has been initialized to allow the controlling application to do necessary initialization based on the configuration parameters. The corresponding SimFinished event gives the controlling application a chance to clean up and to carry out and report any final calculations.

VII. TNG LAB GRAPHICAL-USER INTERFACE

The TNG Lab GUI consists of five distinct screens. A *settings screen* permits the user to set key market parameter values. A *results screen* permits the user to view simulation performance data in tabular form in real time. A *chart screen* permits the user to view simulation performance data in graphical form in real time. An *animation screen* permits the user to view the evolution of trade networks in a real-time animation. Finally, a *physics screen* permits the user to set animation physics parameters to control the network visualization. The TNG Lab GUI opens in the settings screen. The user can then use tabs to enter or exit each of the other screens as desired.

More precisely, the settings screen permits the user to set key market parameter values for each TNG/SimBioSys simulation run. As will be illustrated in the context of a concrete labor-market application in Section VIII, these values control market structure, payoffs, FSM representation for trade rules, and the form of the GA learning mechanism. A screen shot of the settings screen for a two-sided market simulation run with equal numbers of buyers and sellers is shown in Fig. 7.

As seen in Fig. 7, the market structure parameters include the total number of buyers, the total number of sellers, the total number of dealers, the buyer quota level (for buyers and dealers), and the seller quota level (for sellers and dealers). The payoff parameters include four trade (prisoner's dilemma) payoffs, an initial expected utility assessment, a refusal payoff, an inactivity payoff, and an experience gain parameter. The initial expected utility assessment is the assessment used by each tradebot for each potential trading partner at the start of the first generation. The refusal payoff is a nonpositive transactions cost incurred by a buyer whenever one of its offers to buy is refused. The inactivity payoff is the payoff incurred by a tradebot who neither makes nor accepts offers during the course of a trade cycle. The inactivity payoff can be positive, zero, or negative depending on the application. For example, the inactivity payoff might be positive in the context of a market with welfare support. The experience gain parameter is the weight that each tradebot applies to its most recent experiences when updating its current expected utility assessments at the end of an evolution step in preparation for the start of a new generation.

Also, the FSM parameters include the number of internal states for the FSM representation and a memory parameter controlling how many past moves of a current trade partner are recalled (along with the current FSM state) to condition the choice
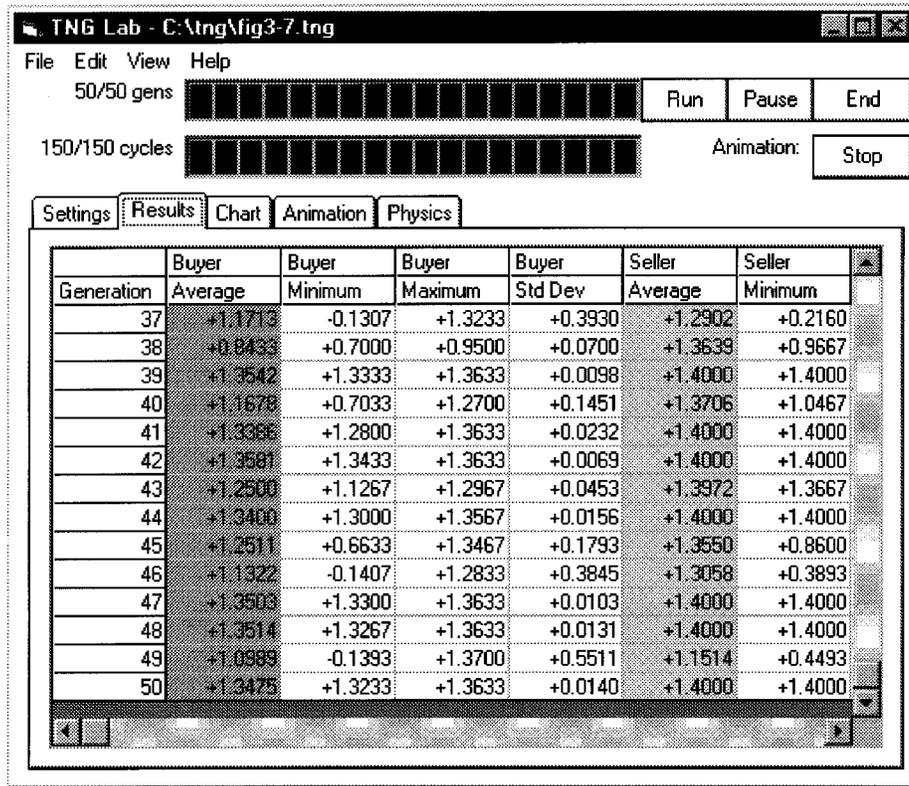
	TNG Lab - C:\tng\fig3-7.tng						
File Edit View Help							

50/50 gens [██████████████████]   [Run] [Pause] [End]

150/150 cycles [██████████████████]   Animation: [Stop]

[Settings] [Results] [Chart] [Animation] [Physics]

Generation	Buyer Average	Buyer Minimum	Buyer Maximum	Buyer Std Dev	Seller Average	Seller Minimum
37	+1.1713	-0.1307	+1.3233	+0.3930	+1.2902	+0.2160
38	+0.8433	+0.7000	+0.9500	+0.0700	+1.3639	+0.9667
39	+1.3542	+1.3333	+1.3633	+0.0098	+1.4000	+1.4000
40	+1.1678	+0.7033	+1.2700	+0.1451	+1.3706	+1.0467
41	+1.3366	+1.2800	+1.3633	+0.0232	+1.4000	+1.4000
42	+1.3581	+1.3433	+1.3633	+0.0069	+1.4000	+1.4000
43	+1.2500	+1.1267	+1.2967	+0.0453	+1.3972	+1.3667
44	+1.3400	+1.3000	+1.3567	+0.0156	+1.4000	+1.4000
45	+1.2511	+0.6633	+1.3467	+0.1793	+1.3550	+0.8600
46	+1.1322	-0.1407	+1.2833	+0.3845	+1.3058	+0.3893
47	+1.3503	+1.3300	+1.3633	+0.0103	+1.4000	+1.4000
48	+1.3514	+1.3267	+1.3633	+0.0131	+1.4000	+1.4000
49	+1.0989	-0.1393	+1.3700	+0.5511	+1.1514	+0.4493
50	+1.3475	+1.3233	+1.3633	+0.0140	+1.4000	+1.4000

Fig. 8.   TNG lab GUI results screen.

of a current action. The GA learning parameters include the total number of generations, the total number of trade cycles per generation, a mutation rate, and an elite percentage separately specified for each tradebot type (buyer, seller, and dealer). In addition, a seed value has to be set to initialize the pseudorandom number generator.

The results screen permits the user to view fitness data for the tradebots as each simulation run proceeds. Mean and standard deviation calculations for the fitness of each tradebot type and for all tradebots together are provided in separate columns. The key columns of interest are the four columns that provide average fitness data for buyers, sellers, dealers, and all tradebots together. These key columns are highlighted in colors that are used consistently throughout the TNG Lab GUI: namely, blue for buyers, yellow for sellers, green for dealers, and red for all tradebots together. Only columns for tradebot types actually present in a current simulation run are activated.

Fig. 8 shows a screen shot of the results screen for the two-sided market simulation run in Fig. 7. This screen shot was taken at the end of the simulation run with the results screen scrolled to data for the final generations.

The chart screen permits the user to view in separate charts the average, maximum, and minimum fitness levels achieved by tradebots of each type and by tradebots as a whole, as each simulation run proceeds. The charts for average fitness are color coded using the same colors that were used for the results screen: blue for buyers, yellow for sellers, green for dealers,

and red for all tradebots. Only charts for tradebot types actually present in a current simulation run are activated.

Fig. 9 shows a screen shot for one of the charts provided by the chart screen for the two-sided market simulation run in Fig. 7. This screen shot was taken at the end of the simulation run and uses color-coded line charts to depict the average fitness levels achieved by buyers and sellers in each generation.

The animation screen was introduced to allow researchers an opportunity to gain insight into the dynamics of trade network formation by watching the tradebots interact with each other in real time. The abstract game simulation modeled in the lower layers of the architecture (TNG/SimBioSys) is modeled as a physical simulation in the top layer (TNG Lab).[8]

Each tradebot is represented in the animation screen by a point mass and is displayed as a letter with a numerical subscript. The letters "B," "S," and "D" stand for "buyer," "seller," and "dealer," respectively. The numerical subscript serves to differentiate tradebots of the same type. The letters with numerical subscripts are color-coded in conformity with the results screen and the chart screen, namely, blue for buyers, yellow for sellers, and green for dealers.

---

[8]Network visualization has been an active field of research for over 40 years. The animation physics for the TNG Lab was greatly influenced by previous work on network visualization for iterated prisoner's dilemma games with choice and refusal of partners; see [7] and [8]. For pointers to other work on network visualization, visit the Formation of Economic and Social Networks website at http://www.econ.iastate.edu/tesfatsi/netgroup.htm.
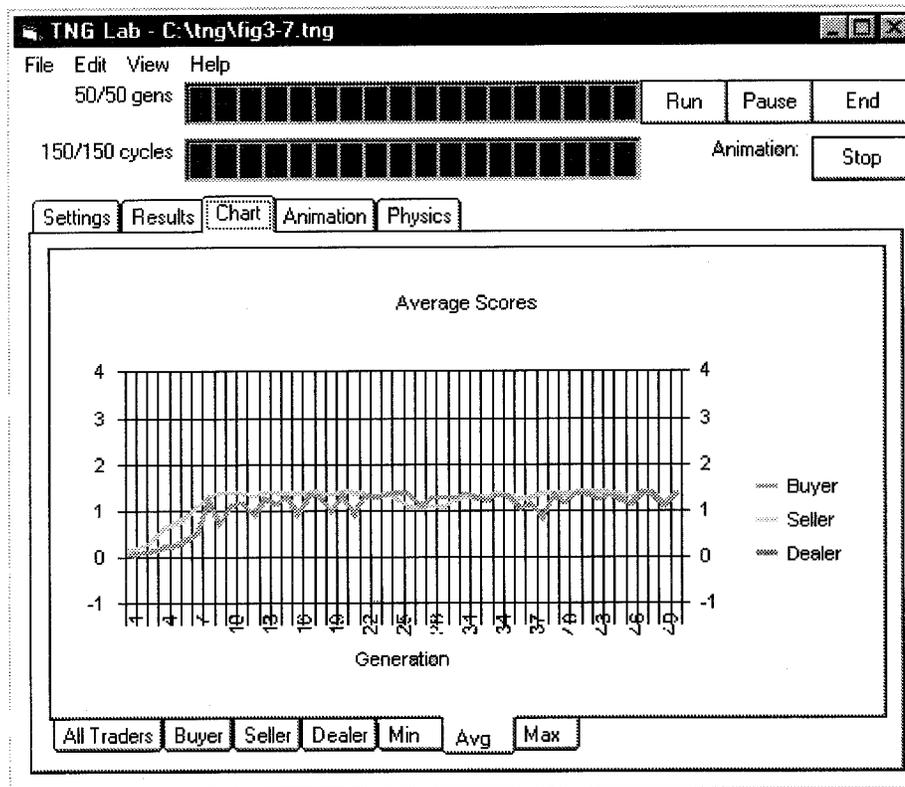
Fig. 9. TNG lab GUI chart screen.

Each step in the animation coincides with the end of a trade cycle. The forces on each tradebot due to bonds with other tradebots and repulsion forces are summed up and the resulting vector is applied to the tradebot's position in the arena. The result of these simple calculations is the emergence of a dynamic trade network visualization. Groups of tradebots are seen to make initial deals with each other and to tentatively form a network. One or more of these tradebots may ultimately engage in too many defections, however, breaking the alliance. The ostracized defectors then move elsewhere, seeking more profitable trades with new partners. The visual effect is quite compelling.

Specifically, at the end of each trade cycle A, the relationship between any two tradebots who are potential trade partners is classified as follows. The two tradebots are in a *latched relationship* if the tradebots have traded with each other at least once in each of the last A/FL trade cycles and if each tradebot currently has a nonnegative expected utility assessment for the other. A latched relationship is implemented as a spring with a relatively short rest length and displayed as a solid line. The two tradebots have a *transient relationship* either if they have not traded with each other at all in any of the last A/FT trade cycles or if at least one of the two tradebots currently has a negative expected utility assessment for the other. Once a relationship between two tradebots is classified as transient, the relationship is not depicted visually and any bond that previously existed between the two tradebots is destroyed. Finally, the two tradebots have a *recurrent relationship* if their relationship is neither latched nor transient. A recurrent relationship is implemented

as a spring with a relatively long rest length and displayed as a dashed line.

Three additional forces are also introduced in the simulation in order to enhance the visualization. First, each tradebot acts as a point charge, repelling every other tradebot with a force that varies inversely with the square of the distance separating them. This prevents groups of tradebots from overlapping, which would obscure the visualization. Second, the walls of the arena (the inside borders of the window containing the animation) have a repelling effect on each tradebot with a force that varies inversely with the square of the perpendicular distance separating the tradebot from the wall. This forces pushes each tradebot back into the arena when other forces threaten to push it out of sight. Finally, a frictional force that is proportional and opposite to a tradebot's current velocity is introduced to dampen oscillations.

Each step in the animation coincides with the end of a trade cycle. The forces on each tradebot due to bonds with other tradebots and repulsion forces are summed up and the resulting vector is applied to the tradebot's position in the arena. The result of these simple calculations is the emergence of a dynamic trade network visualization. Groups of tradebots are seen to make initial deals with each other and to tentatively form a network. One or more of these tradebots may ultimately engage in too many defections, however, breaking the alliance. The ostracized defectors then move elsewhere, seeking more profitable trades with new partners. The visual effect is quite compelling.

A screen shot of the animation screen for the two-sided market simulation run in Fig. 7 is shown in Fig. 10. For this
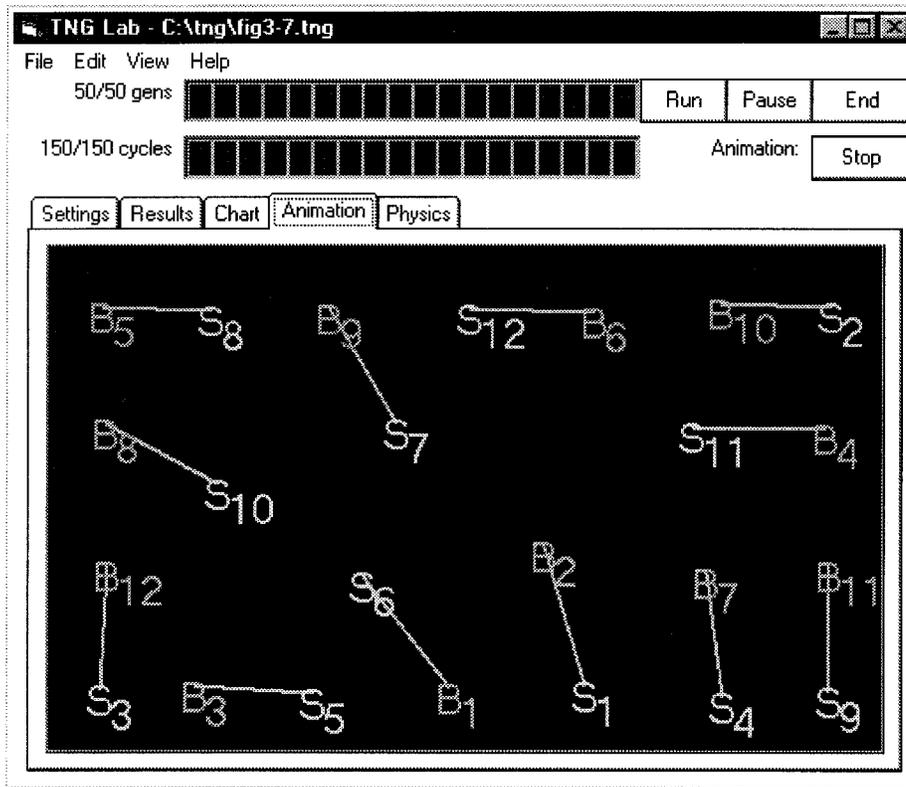
Fig. 10.   TNG lab GUI animation screen.

run, the 12 buyers and 12 sellers manage to self-organize into 12 disjoint buyer-seller pairs by about the 20th generation and this network formation then persists throughout the remaining 30 generations. Fig. 10 gives the still display of the network formation at the end of the final (50th) generation.

Finally, the physics screen permits the user to set the frequency theshold parameters FL and FT, the spring rest lengths and strengths for both latched and recurrent relationships, the repulsion forces (trader, boundary), and the frictional force. These physics parameters permit the user to tailor the network visualization in the animation screen to the application at hand. A screen shot of the physics screen for the two-sided market simulation run in Fig. 7 is shown in Fig. 11.

VIII. LABOR-MARKET APPLICATION

This section first outlines a labor-market framework that has been implemented with the support of TNG/SimBioSys [6]. Experiments conducted with this framework are then used to illustrate the capabilities of the TNG Lab. These experiments address an important unresolved issue in current labor-market research, referred to as the "excess heterogeneity" problem [9]. Briefly, the issue is why observationally equivalent workers and employers have markedly different earnings and employment histories.

A. Labor-Market Framework

The labor market is a two-sided market consisting of NW work suppliers ("buyers" of job openings) and NE employers ("sellers" of job openings), where NW and NE are arbitrary

positive integers. Each work supplier is assumed to have the same (work) quota WQ, where WQ is the maximum number of potential work offers that each work supplier can have outstanding at any given time.[9] Similarly, each employer is assumed to have the same (employment) quota EQ, where EQ is the maximum number of job openings that each employer can provide at any given time.

As in Fig. 5, activities in the labor market are divided into a sequence of *generations*. Each work supplier and employer in the initial generation is assigned a randomly generated rule governing its worksite behavior, an initial expected utility assessment for each of its potential worksite partners, and a quota governing its size. The work suppliers and employers then enter into a *trade-cycle loop* during which they repeatedly search for preferred worksite partners on the basis of their current expected utility assessments, engage in worksite interactions modeled as prisoner's dilemma games, and update their expected utility assessments to take into account newly incurred job search costs, inactivity costs, and worksite payoffs. At the end of the trade-cycle loop, the work suppliers and employers each separately evolve (structurally modify) their worksite behavioral rules based on the past net payoff outcomes secured with these rules and a new generation then commences.

Matches between work suppliers and employers are determined using a one-sided offer auction. Each work supplier first

---

[9]When WQ exceeds one, each work supplier can be interpreted as some type of information service provider (e.g., broker or consultant) that is able to supply services to at most WQ employers at a time or as some type of union organization that is able to oversee work contracts with at most WQ employers at a time.
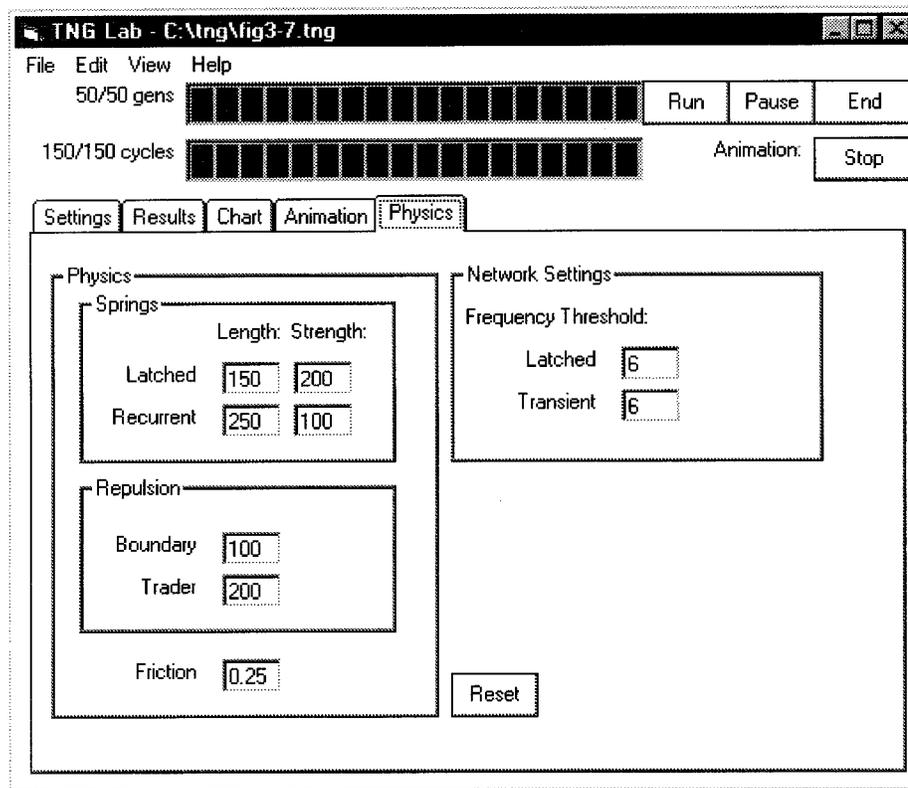
Fig. 11. TNG lab GUI physics screen.

submits work offers to a maximum of WQ employers it ranks as most preferable on the basis of expected utility and who it judges to be tolerable in the sense that their expected utility is not negative. Each employer then selects up to EQ of the work offers it has received to date that it finds tolerable and most preferable on the basis of expected utility and it places these selected work offers on a waiting list; all other work offers are refused. Work suppliers who have work offers refused then redirect these work offers to any tolerable preferred employers who have not yet refused them, and the process repeats. Once an employer stops receiving new work offers, it accepts all work offers currently on its waiting list.

A work supplier incurs a job search cost in the form of a negative *refusal payoff R* each and every time that an employer refuses one of its work offers during a trade cycle; the employer who does the refusing is not penalized. A work supplier or employer who neither submits nor accepts work offers during a trade cycle receives an *inactivity payoff* zero for the entire trade cycle. The refusal and inactivity payoffs are each assumed to be measured in utility terms.

If an employer accepts a work offer from a work supplier in any given trade cycle, the work supplier and employer are said to be *matched* for that trade cycle. Each match constitutes a mutually agreed upon contract stating that the work supplier shall supply labor services at the worksite of the employer until the beginning of the next trade cycle. These contracts are risky in that outcomes are not assured.

Specifically, work suppliers and employers can each shirk on the worksite to the detriment of the other and can possibly improve their own welfare by doing so. Work suppliers can reduce

their disutility of work in the short run by not working as hard as their employers expect and employers can enhance their profit in the short run by not providing benefits their work suppliers expect to receive. Offsetting these incentives are factors that discourage shirking. Employers can punish shirking work suppliers by firing them (i.e., by refusing their future work offers) and work suppliers can punish shirking employers by quitting (i.e., by redirecting their future work offers elsewhere).

These various possibilities are captured by having each matched work supplier and employer engage in a worksite interaction modeled as a two-person prisoner's dilemma game. The work supplier can either cooperate (exert high work effort) or defect (shirk). Similarly, the employer can either cooperate (provide good working conditions) or defect (shirk). The range of possible worksite payoffs is assumed to be the same for each worksite interaction in each trade cycle, namely, a cooperator whose worksite partner defects receives the lowest possible payoff L (sucker payoff), a defector whose worksite partner also defects receives the next lowest payoff D (mutual defection payoff), a cooperator whose worksite partner also cooperates receives a higher payoff C (mutual cooperation payoff), and a defector whose worksite partner cooperates receives the highest possible payoff H (temptation payoff).

The worksite payoffs are assumed to be measured in utility terms and to be normalized about the inactivity payoff zero so that $L < D < 0 < C < H$. Thus, a work supplier or employer that ends up either as a sucker with payoff L or in a mutual defection relation with payoff D receives negative utility, a worse outcome than inactivity (unemployment or vacancy). The worksite payoffs are also assumed to satisfy the usual prisoner's

dilemma regularity condition $(L + H)/2 < C$, guaranteeing that mutual cooperation dominates alternating cooperation and defection on average.

Each trader, whether a work supplier or an employer, uses a simple reinforcement learning algorithm to update its expected utility assessments on the basis of new payoff information during the course of each trade-cycle loop. Specifically, a trader v assigns an initial expected utility U^o to each potential worksite partner z with whom it has not yet interacted. Each time an interaction with z takes place, v forms an updated expected utility assessment for z by summing U^o together with all payoffs received to date from interactions with z (including both worksite payoffs and refusal payoffs) and then dividing this sum by one plus the number of interactions with z.

The personality of each trader, as expressed in its worksite interactions, is governed by a *worksite (behavioral) rule* that is maintained throughout the course of each trade-cycle loop. These worksite rules are represented as finite-memory pure strategies for playing a prisoner's dilemma game with an arbitrary partner an indefinite number of times. At the commencement of each trade-cycle loop, traders have no information about the worksite rules of other traders; each trader can only learn about these rules by engaging other traders in repeated worksite interactions and observing the actions and payoff outcomes that ensue. Each trader keeps separate track of its interaction history with each potential worksite partner and each trader's choice of an action in a current worksite interaction with another trader is determined on the basis of its own past interactions with this other trader plus its initial expected utility assessment of the trader. This means, in particular, that a trader can end up revealing different aspects of its personality to different worksite partners due to differences in their interaction histories. For example, a work supplier may develop a mutually cooperative relationship with one employer while at the same time it is shirking on the job with a second employer.

At the end of each trade-cycle loop, the *utility (fitness)* of each trader is measured by normalized total net payoff, i.e., by total net payoff divided by the fixed number of trade cycles constituting each trade-cycle loop. For work suppliers, total net payoff is measured by total net worksite payoffs plus the (negative) sum of any incurred refusal payoffs. For employers, total net payoff is measured simply by total net worksite payoffs.

The work suppliers and employers then separately evolve their worksite rules by means of elitism, mutation, and recombination operations biased in favor of more successful (fit) traders. Elitism ensures that the most successful worksite rules are retained unchanged from one generation to the next. Mutation ensures that work suppliers and employers continually experiment with new worksite rules (inductive learning). Recombination ensures that work suppliers and employers continually engage in mimicry (social learning). Specifically, if the use of a worksite rule successfully results in a high fitness for a trader of a particular type, then, through recombination operations, other traders of the same type will tend to modify their own worksite rules to more closely resemble the successful rule.

At the end of the evolution step, each work supplier or employer v updates its initial expected utility assessment for each

of its potential trade partners z by taking a weighted average of the expected utility it assigned to z at the beginning of the latest generation and the expected utility it now assigns to z at the end of this latest generation. For example, suppose the current generation is $G \geq 0$, the expected utility assigned by v to z at the beginning of G was $U^o(G)$, and the expected utility now assigned by v to z at the end of generation G is $U^*(G)$. Then

$$U^o(G+1) = [1 - e] \cdot U^o(G) + e \cdot U^*(G) \qquad (1)$$

where the *experience gain* e lies between zero and one.[10]

B. Illustrative Experimental Findings

Consider the special case of a labor market with a balanced concentration, i.e., a labor market for which the number NW of work suppliers equals the number NE of employers. Each work supplier has a quota WQ on the number of work offers it can have outstanding at any given time and each employer has a quota EQ on the number of job openings it can provide at any given time. Define the (relative) job capacity ($JCAP$) of this economy to be the ratio of total potential job openings ($NE \cdot EQ$) to total potential work offers ($NW \cdot WQ$). Under the balanced concentration assumption, $JCAP$ simply reduces to the size ratio EQ/WQ.

This section reports TNG Lab experimental findings regarding the effects of systematic variations in $JCAP$ when the labor market consists of twelve work suppliers and twelve employers. The primary purpose in reporting these findings is to convey in general terms the capabilities of the TNG Lab in facilitating interesting socioeconomic research. Consequently, the findings will be explained and motivated here at a general intuitive level. A detailed and rigorous discussion of these and many additional related findings can be found in [6].

Three distinct $JCAP$ treatments are tested by specifying three distinct settings for the quotas WQ and EQ as follows: 1) $WQ = 2$ and $EQ = 1$, implying $JCAP = 1/2$ (tight job capacity); 2) $WQ = 1$ and $EQ = 1$, implying $JCAP = 1$ (balanced job capacity); and 3) $WQ = 1$ and $EQ = 2$, implying $JCAP = 2$ (excess job capacity). Apart from these quota changes and changes in the seed value for the pseudorandom number generator, all other TNG/SimBioSys parameters in the settings screen are maintained at fixed values throughout all experiments. Fig. 7 displays the settings screen for a balanced job capacity experiment with the "buyer quota" WQ set to one, the "seller quota" EQ set to one, and the seed value set to 19.

Intuitively, work suppliers should be favored when $JCAP$ is large, since job openings are then potentially in excess supply. Conversely, employers should be favored when $JCAP$ is small, since job openings are then potentially in excess demand. The following plausible hypothesis will, therefore, be tested.

JCAP Hypothesis: As $JCAP$ increases, all else equal, the average utility level (fitness) attained by work suppliers increases

---

[10]For simplicity, it is assumed that all tradebots use the same fixed values for the initial expected utility level $U^o(0)$ and for the experience gain parameter e, as specified by the user on the settings screen for the TNG Lab GUI. Eventually, however, it would be interesting to permit individual tradebots to have different $U^o(0)$ values and to evolve e over time on the basis of their own unique experiences.

TABLE I
EXPERIMENTAL FINDINGS FOR VARYING JOB CAPACITY LEVELS. (a) TIGHT JOB CAPACITY ($JCAP = 1/2$). (b) BALANCED JOB CAPACITY ($JCAP = 1$). (c)
EXCESS JOB CAPACITY ($JCAP = 2$)

D^o	% of Runs	AGGRESSIVE		P-INACTIVE		P-NICE		UTILITY		MPOWER	
		w	e	w	e	w	e	w	e	w	e
1-7	55%	2% (4%)	5% (1%)	23% (9%)	2% (5%)	74% (11%)	95% (12%)	0.87 (.13)	1.38 (.04)	-38% (9%)	-1% (3%)
12-14	20%	10% (18%)	75% (43%)	38% (9%)	4% (7%)	48% (25%)	50% (42%)	0.88 (.21)	1.46 (.22)	-37% (15%)	+4% (15%)
24	25%	100% (0%)	100% (0%)	100% (0%)	100% (0%)	0% (0%)	0% (0%)	-0.12 (0)	-0.02 (0)	-109% (0%)	-101% (0%)

(a)

D^o	% of Runs	AGGRESSIVE		P-INACTIVE		P-NICE		UTILITY		MPOWER	
		w	e	w	e	w	e	w	e	w	e
1	5%	0% NA	0% NA	8% NA	0% NA	100% NA	100% NA	0.45 NA	1.37 NA	-68% NA	-2% NA
12	70%	12% (26%)	31% (44%)	0% (0%)	0% (0%)	85% (26%)	85% (26%)	1.23 (.19)	1.33 (.14)	-12% (13%)	-5% (10%)
13-15	25%	63% (45%)	33% (40%)	13% (4%)	15% (3%)	48% (40%)	67% (35%)	1.08 (.30)	0.97 (.29)	-23% (21%)	-31% (21%)

(b)

D^o	% of Runs	AGGRESSIVE		P-INACTIVE		P-NICE		UTILITY		MPOWER	
		w	e	w	e	w	e	w	e	w	e
0	5%	17% NA	0% NA	0% NA	0% NA	83% NA	100% NA	1.42 NA	1.30 NA	1% NA	-7% NA
4-6	25%	2% (3%)	0% (0%)	0% (0%)	0% (0%)	98% (0%)	98% (0%)	1.37 (.00)	1.38 (.04)	-2% (0%)	-1% (2%)
14-17	70%	100% (0%)	29% (45%)	2% (3%)	29% (9%)	10% (26%)	60% (32%)	1.74 (.32)	0.61 (.27)	+26% (20%)	-56% (19%)

(c)

and the average utility level (fitness) attained by employers decreases.

Table I reports findings for each of the three tested $JCAP$ treatments. For each treatment, 20 runs were generated using 20 different seed values for the pseudorandom number generator. As will be clarified further below, the first two columns of Table I report observed network formation clusters, the next six columns report means and standard deviations for observed behavioral attributes supported by these network formation clusters, and the final four columns report means and standard deviations for observed welfare and market power outcomes supported by these network formation clusters.

The first column of Table I, labeled D^o, classifies observed network formations in accordance with a distance measure D^o. The origin zero of this distance measure corresponds to a benchmark "competitive" network formation in which any unemployment is distributed uniformly across work suppliers, any vacancies are distributed uniformly across employers, and all traders always cooperate (implying all relationships are recurrent). By construction, larger D^o values imply larger deviations away from this competitive network formation.

The second column of Table I, labeled "% of Runs," gives the percentage of the twenty runs for each treatment that lie within the indicated range of D^o values. The first interesting aspect to note about the Table I results is that, for each $JCAP$ treatment, the network formations lie in two or three sharply distinguished distance clusters with one cluster markedly dominating the others in percentage terms. The dominant distance cluster

(55%) for tight job capacity is the D^o interval 1–7, the dominant distance cluster (70%) for balanced job capacity is the D^o level 12, and the dominant distance cluster (70%) for excess job capacity is the D^o interval 14–17.

Thus, for each treatment, the network histogram is spectral in form with two or three isolated peaks; there is no smooth bell-shaped central tendency distribution. Figs. 12–14 illustrate the three distinct types of networks that arise for the tight job capacity treatment with $JCAP = 1/2$: namely, largely recurrent relations, a mix of latched and recurrent relations, and fully transient relations. Each figure presents an animation screen still display of the network formation in the final generation of a single simulation run. The only change from one run to the next is a change in the seed value for the random number generator.

Next, consider the Table I columns labeled "Aggressive," "P-Inactive," and "P-Nice." These columns report means and standard deviations for the behavioral characteristics of work suppliers and employers in the final generation of each run. For each treatment, these behavioral attributes are separately calculated for each distinct distance cluster indicated in column one.

The Aggressive columns report means and standard deviations for the percentages of work suppliers and employers in the final generation who defect against work partners who have not defected against them in any previous trades. The P-Inactive columns report means and standard deviations for the percentages of work suppliers and employers who become persistently inactive (unemployed or vacant) by the final generation. Finally, the P-Nice columns report means and standard deviations for the
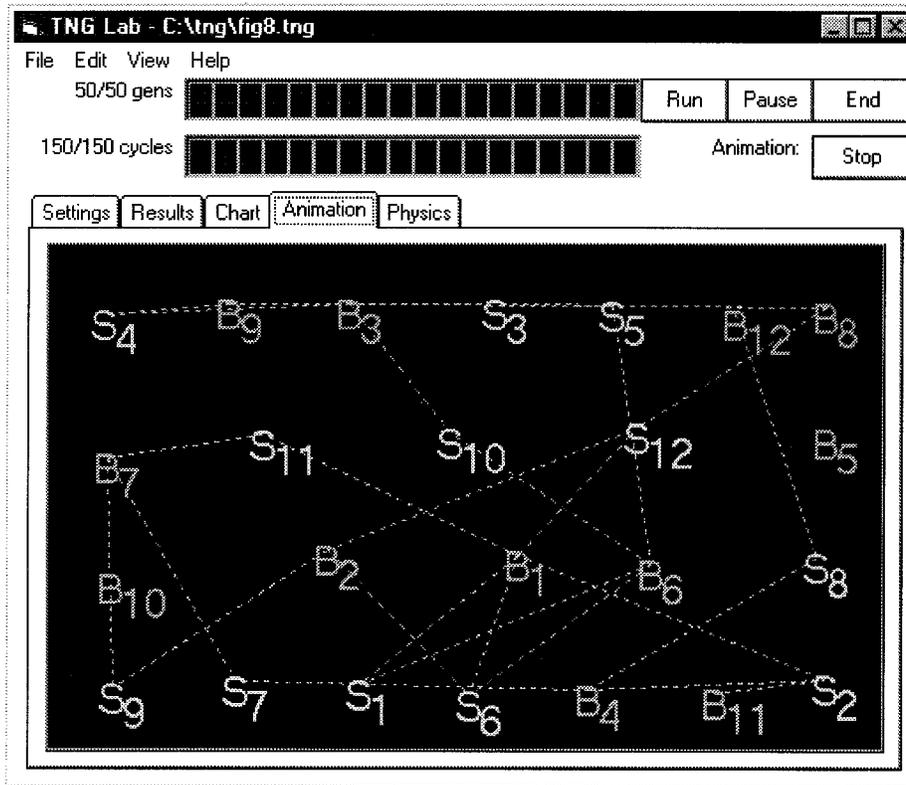
Fig. 12.   Network formation under tight job capacity with D^o in 1–7.

percentages of work suppliers and employers in the final generation who become persistently cooperative.[11]

The final columns of Table I labeled "Utility" and "MPower" report means and standard deviations for the utility (fitness) and market power levels achieved by the final generation of work suppliers and employers. These levels are separately calculated for each distinct distance cluster indicated in column one. Market power is measured in percentage terms as the difference between the utility a trader attains in the experimental labor market and the utility the trader would instead attain in the competitive network formation, in ratio to the latter utility.

Now consider the $JCAP$ hypothesis. Restricting attention to dominant distance clusters in Table I, it is seen that this hypothesis receives strong support. The mean utility level attained by work suppliers in the dominant distance cluster increases dramatically as $JCAP$ increases from 1/2 to 2, whereas the mean utility level attained by employers declines.

As shown in [6, Table VI], the $JCAP$ hypothesis is still supported even when, for each treatment, the data for the dominant distance cluster is pooled with the data for less dominant distance clusters. Nevertheless, Table I shows how misleading it can be simply to pool data across distance clusters for each treatment. For example, in the tight job capacity case reported in Table I(a), complete coordination failure is seen to occur in 25% of the runs, namely, in distance cluster $D^o = 24$. In this distance cluster, 100% of the work suppliers end up persistently

unemployed and 100% of the employers end up persistently vacant.

The problem in the tight job capacity case is that each work supplier has a hard time finding job openings because jobs are scarce relative to work offers. Consequently, each work supplier tends to accumulate many (negative) refusal payoffs during the job search process. Moreover, employers have an incentive to defect on the worksite, which can induce the evolution of protective defecting behavior in work suppliers. If a work supplier accumulates too many refusals from any one employer or if the worker and employer are both aggressive and engage in mutual defection in their first worksite interaction, then the work supplier will cease making work offers to this employer because the expected utility it assigns to this employer will drop below zero. If this happens for too many employers, the work supplier will simply withdraw altogether from the labor market, preferring unemployment (at inactivity cost zero) to the risk of sustaining additional negative payoffs. The latter situation is exactly what occurs in every run included in distance cluster $D^o = 24$. By the final (50th) generation, every worker and employer has evolved into an aggressive agent who defects against every new worksite partner. This mutual defection behavior quickly leads to complete coordination failure.

Finally, consider the implications of Table I for the "excess heterogeneity" issue highlighted at the beginning of this section. The issue in question is why observationally equivalent workers and employers are observed to have markedly different earnings and employment histories. Recall that, for each treatment reported in Table I, all work suppliers have observationally identical structural attributes at the start of the first generation and

---

[11]The technical meaning of the important qualifier "persistently" is carefully explained in [6]. The objective is to avoid classifying behaviors on the basis of transient attributes.
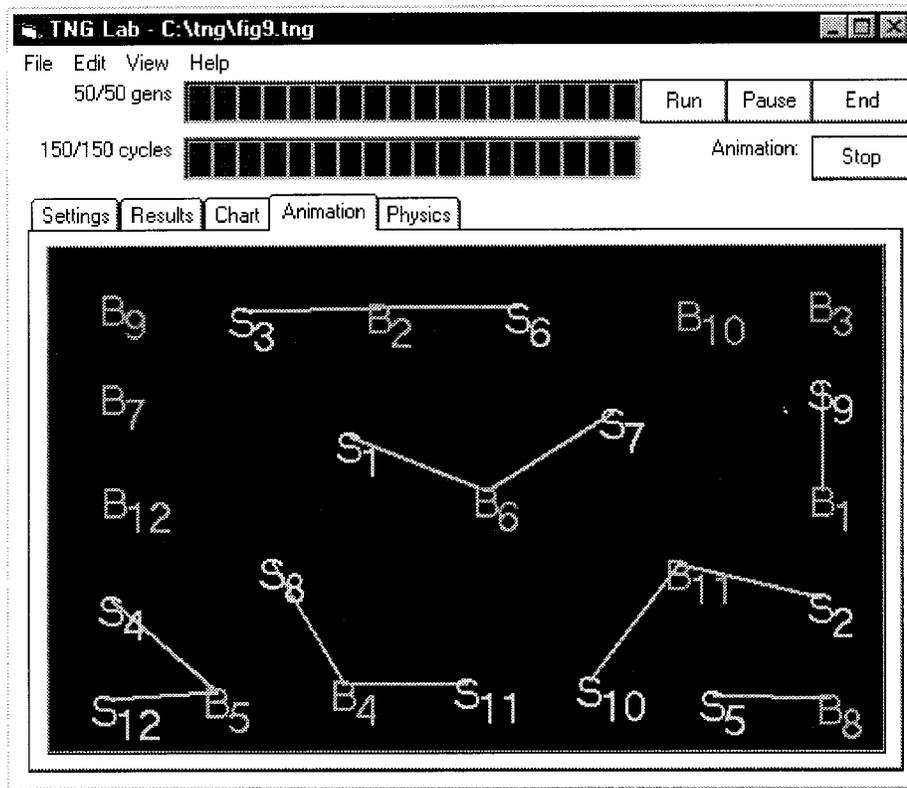
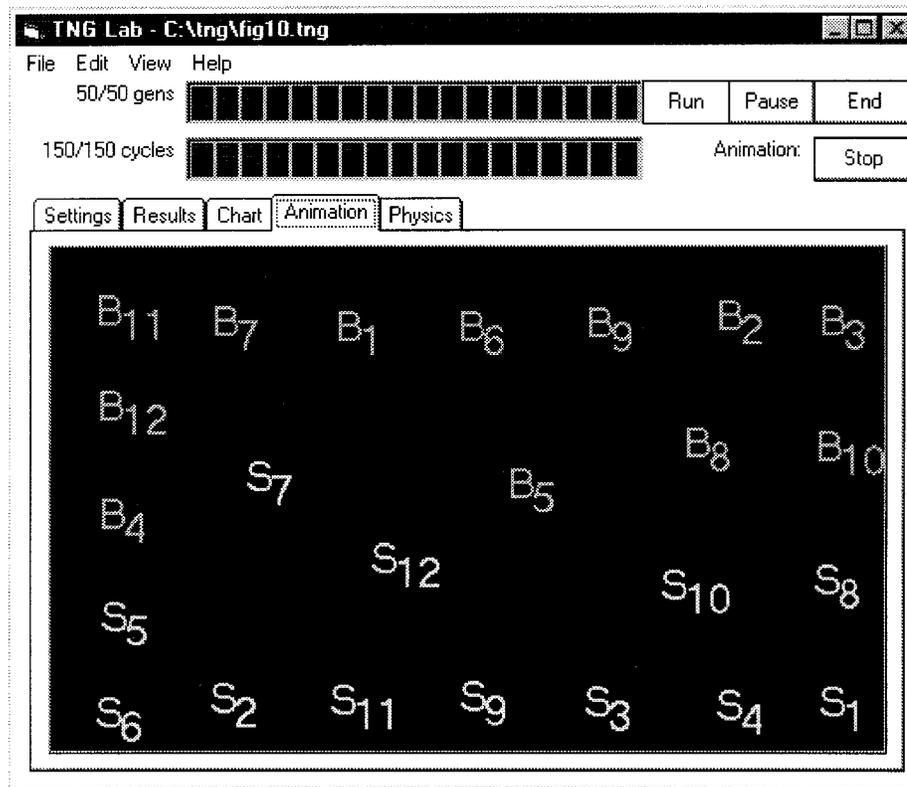Fig. 13. Network formation under tight job capacity with D^o in 12–14.

Fig. 14. Network formation under tight job capacity with $D^o = 24$.

similarly for all employers. The work suppliers and employers separately evolve their worksite rules over time, but these rules are not directly observable by other traders in any given trade-cycle loop. Consequently, standard labor-market theories pur-

porting to explain the distribution of wages and profits on the basis of observable structural attributes would presumably predict that these work suppliers and employers should be earning approximately similar wages and approximately similar profits.

To the contrary, however, Table I indicates a strong degree of path dependency (hysteresis) in this labor market, resulting in wage and profit distributions that have spectral rather than central tendency features. Specifically, for each job capacity treatment reported in Table I, the labor market is capable of evolving multiple distinct clusters of network formations with markedly different utility (wage and profit) levels for work suppliers and employers. Indeed, as indicated by the high standard deviations for some of the mean utility outcomes reported in Table I, there can be a rather substantial degree of within-cluster variability in wages and profits as well.

As indicated in Table I and elaborated in [10], the variability in wages and profits (utility levels) observed in the current labor-market context arises from two sources: network hysteresis effects and behavioral hysteresis effects. Regarding network hysteresis effects, temporary shocks in the form of idiosyncratic worksite interactions can result in persistently heterogeneous network relationships for traders who have identical observable worksite behaviors and structural attributes. Regarding behavioral hysteresis effects, temporary shocks in the form of idiosyncratic worksite interactions can result in persistently heterogeneous worksite behaviors for traders who have identical observable structural attributes. Either effect can support persistently heterogeneous earnings patterns and employment histories across employed work suppliers and across nonvacant employers.

IX. CONCLUDING REMARKS

This study presents, motivates, and illustrates the use of the TNG Lab, an agent-based CL for the study of evolutionary trade networks. In doing so, it constructively demonstrates how CLs can be used to explore complex socioeconomic processes that are difficult to model using standard analytical and statistical tools. In particular, a CL permits exploration to proceed at three levels of analysis: 1) interaction patterns (who is interacting with whom and with what regularity); 2) interaction behaviors (how do agents behave within any given interaction pattern); and 3) welfare outcomes (what consequences arise for individual agents and for society as a whole as a result of these interaction patterns). It is hoped that this study will encourage the increased use of CLs for serious social science research.

ACKNOWLEDGMENT

REFERENCES

[1] J. M. Epstein and R. Axtell, *Growing Artificial Societies: Social Science from the Bottom Up*. Cambridge, MA: MIT Press, 1996.

[2] C. Dibble, "Theory in a Complex World: GeoGraphic Smallworlds as Computational Laboratories," Ph.D. dissertation, Dept. Geography, Univ. California, Santa Barbara, CA, 2001.

[3] D. McFadzean, "SimBioSys: A Class Framework for Evolutionary Simulations," M.Sc. thesis, Dept. Comput. Sci., Univ. Calgary, Calgary, AB, Canada, 1995.

[4] L. Tesfatsion, "A trade network game with endogenous partner selection," in *Computational Approaches to Economic Problems*, H. Amman, B. Rustem, and A. Whinston, Eds. Norwell, MA: Kluwer, 1997, pp. 249–269.

[5] D. McFadzean and L. Tesfatsion, "A C++ platform for the evolution of trade networks," *Comput. Econom.*, vol. 14, pp. 109–134, 1999.

[6] L. Tesfatsion, "Structure, behavior, and market power in an evolutionary labor market with adaptive search," *J. Econom. Dyn. Control*, vol. 25, no. 3–4, pp. 419–457, 2001.

[7] E. A. Stanley, D. Ashlock, and L. Tesfatsion, "Iterated prisoner's dilemma with choice and refusal of partners," in *Artificial Life III*, C. Langdon, Ed. Reading, MA: Addison-Wesley, 1994, vol. XVII, Santa Fe Inst. Studies Sci. Complexity, pp. 131–175.

[8] M. D. Smucker, E. A. Stanley, and D. Ashlock, "Analyzing social network structures in the iterated prisoner's dilemma game with choice and refusal," Dept. Comput. Sci., Univ. Wisconsin, Madison, WI, Tech. Rep. CS-TR-94–1259, 1994.

[9] J. M. Abowd, F. Kramarz, and D. N. Margolis, "High wage workers and high wage firms," *Econometrica*, vol. 67, pp. 251–333, 1999.

[10] L. Tesfatsion, "Hysteresis in an evolutionary labor market with adaptive search ," in *Evolutionary Computation in Economics and Finance*, S.-H. Chen, Ed. New York: Springer-Verlag, 2001, to be published.

David McFadzean (A'96) received the B.Sc. degree in electrical engineering and the M.Sc. degree in computer science from the University of Calgary, Calgary, AB Canada, in 1987 and 1995, respectively.

For the past several years, he has been a Software Developer building decision analysis tools for the energy industry. He is currently Vice President, Technology, for Javien Canada, Inc., Calagary, AB Canada, an Internet startup company that is developing a next-generation infrastructure for distributed collaborative applications. His current research interests include artificial life and complex adaptive systems.

Deron Stewart received the B.Sc. degree in engineering physics from Queen's University, Kingston, ON Canada, in 1988.

He was a Programmer with Merak, Inc., Calgary, for the past five years, building C++/MFC Windows applications. He is currently a self-employed Consultant Programmer, Gabriola, BC Canada. His current research interests include artificial life and evolutionary theory.

Leigh Tesfatsion received the Ph.D. degree in economics with a minor in mathematics from the University of Minnesota, Minneapolis, in 1975.

She is currently a Professor of Economics with Iowa State University, Ames, with a courtesy appointment as a Professor of Mathematics. Her current research interests focus on agent-based computational economics (ACE), the computational study of economies modeled as evolving systems of autonomous interacting agents, which she is applying to the study of market power in labor markets and to the study of restructuring in electricity markets.

Dr. Tesfatsion is a Co-Editor in charge of the Complexity-at-Large section of *Complexity* as well as an Associate Editor of *Journal of Economic Dynamics and Control*, *Journal of Public Economic Theory*, *Applied Mathematics and Computation*, and IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION.
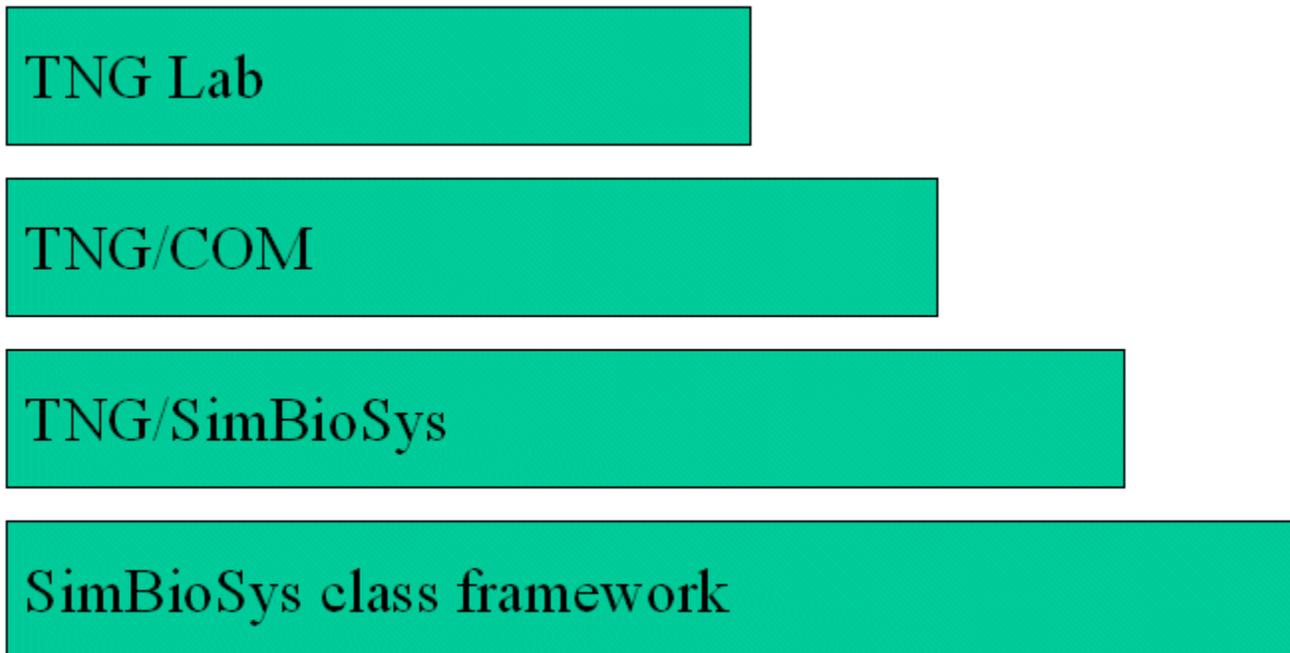
Fig. 1
Architecture of the TNG Lab

TNG Lab

TNG/COM

TNG/SimBioSys

SimBioSys class framework

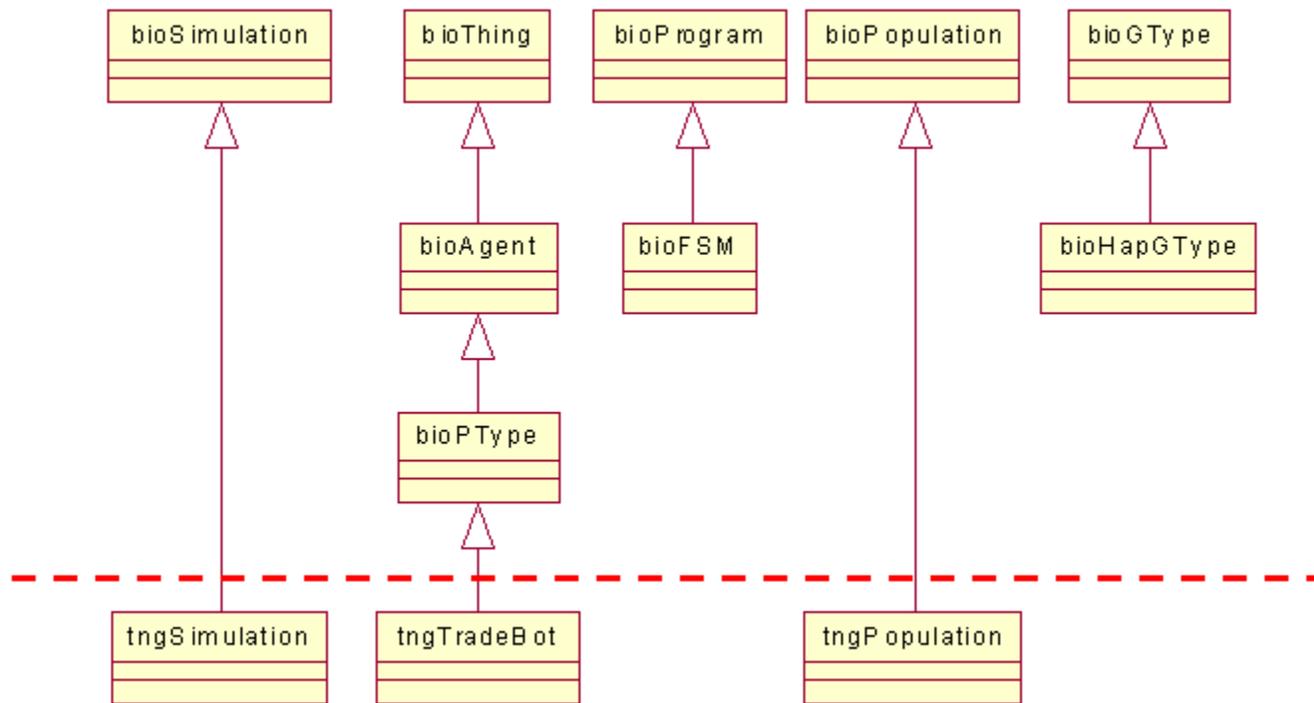# Fig. 2
TNG/SimBioSys Class Structure

Fig. 3
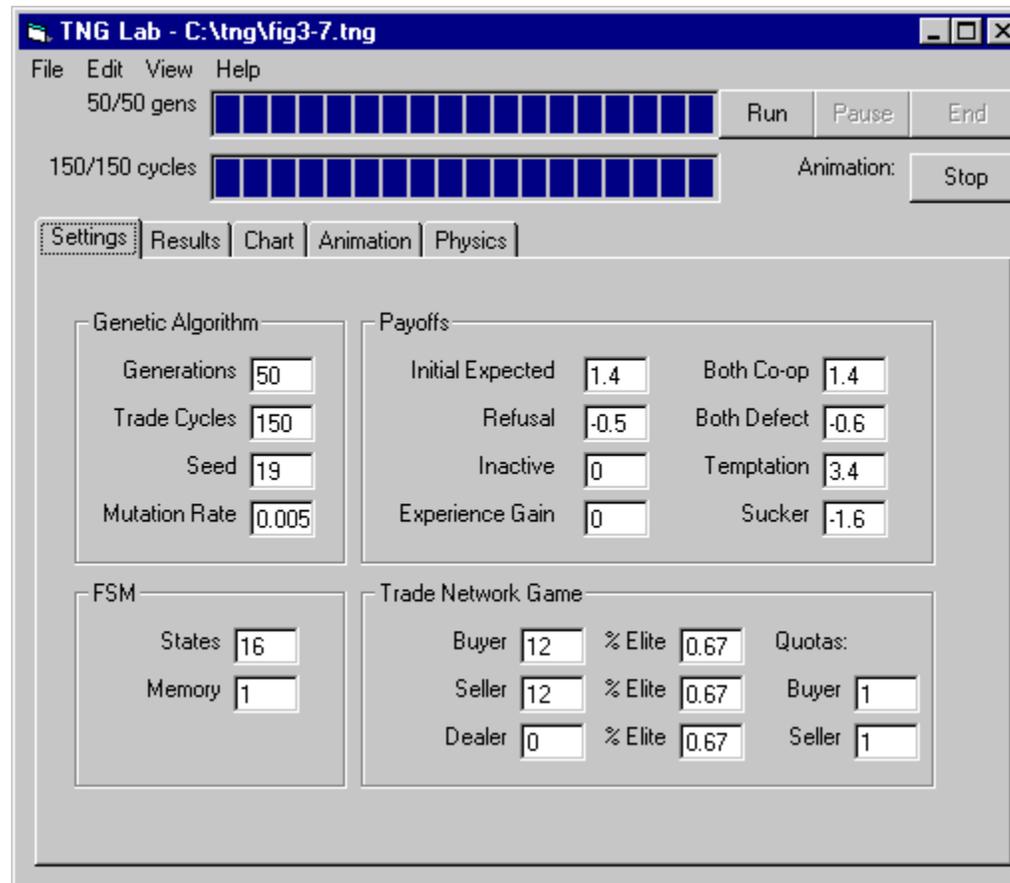# TNG Lab GUI Settings Screen

Fig. 4
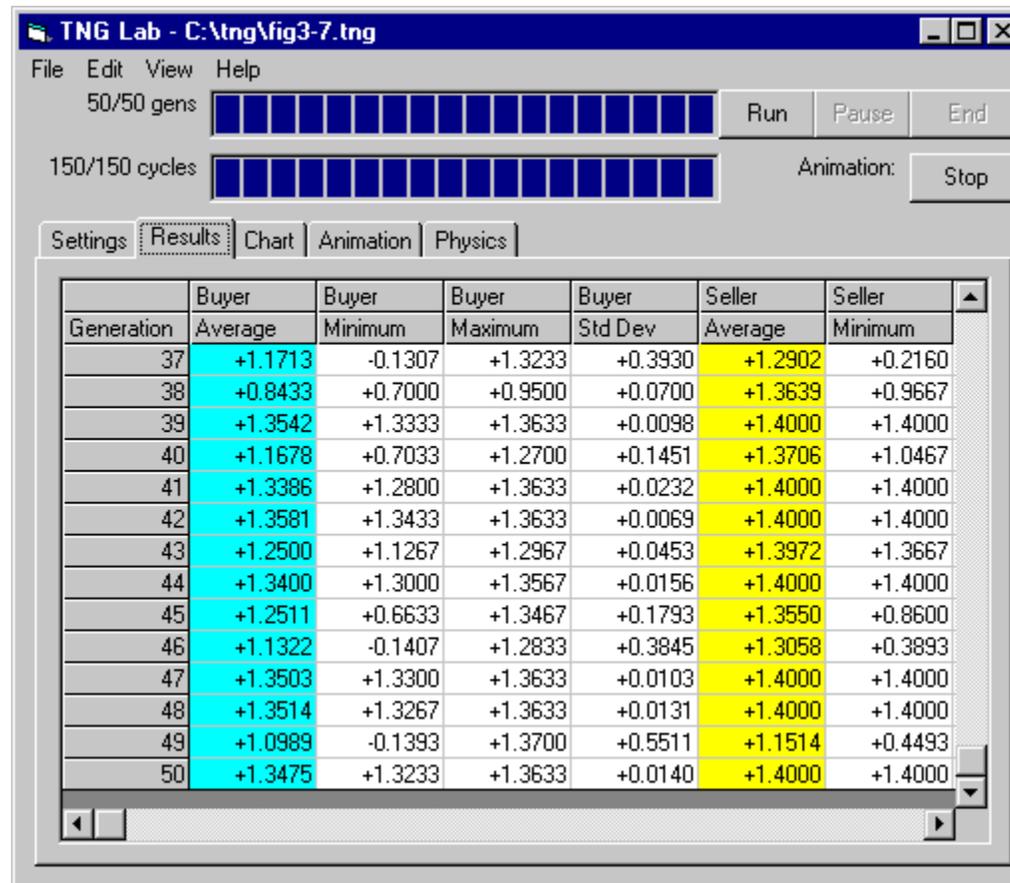## TNG Lab GUI Results Screen
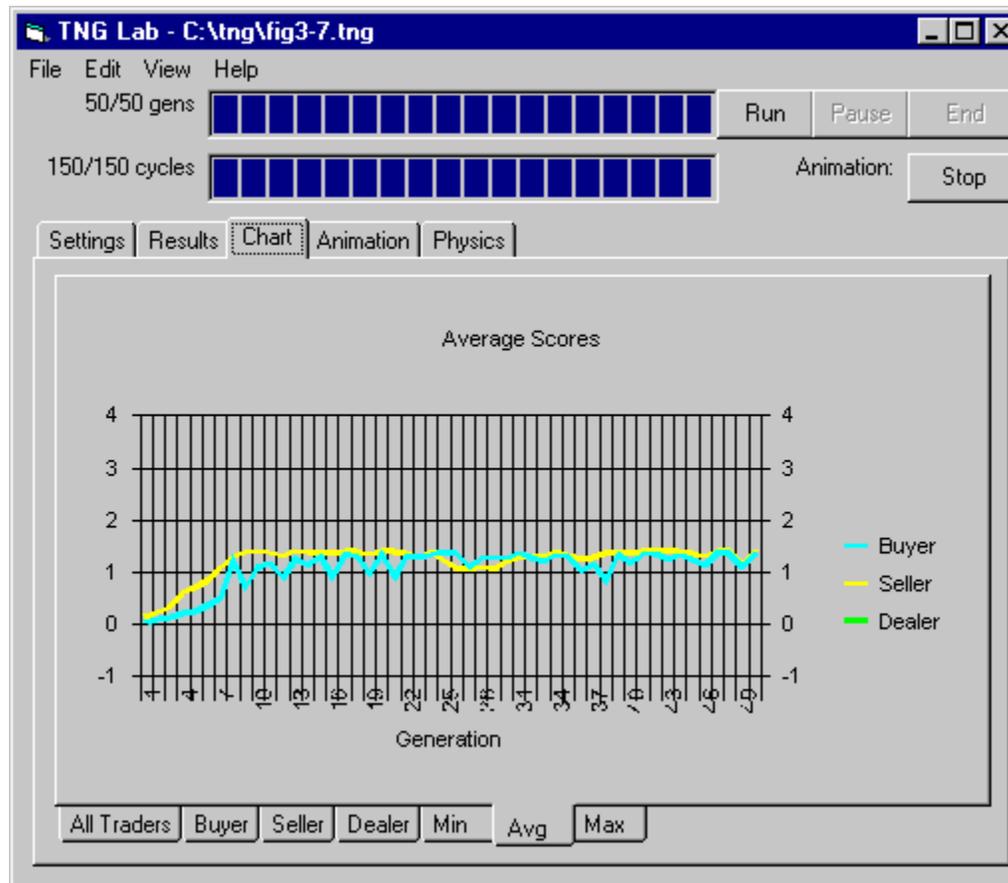
Fig. 5
TNG Lab GUI Chart Screen
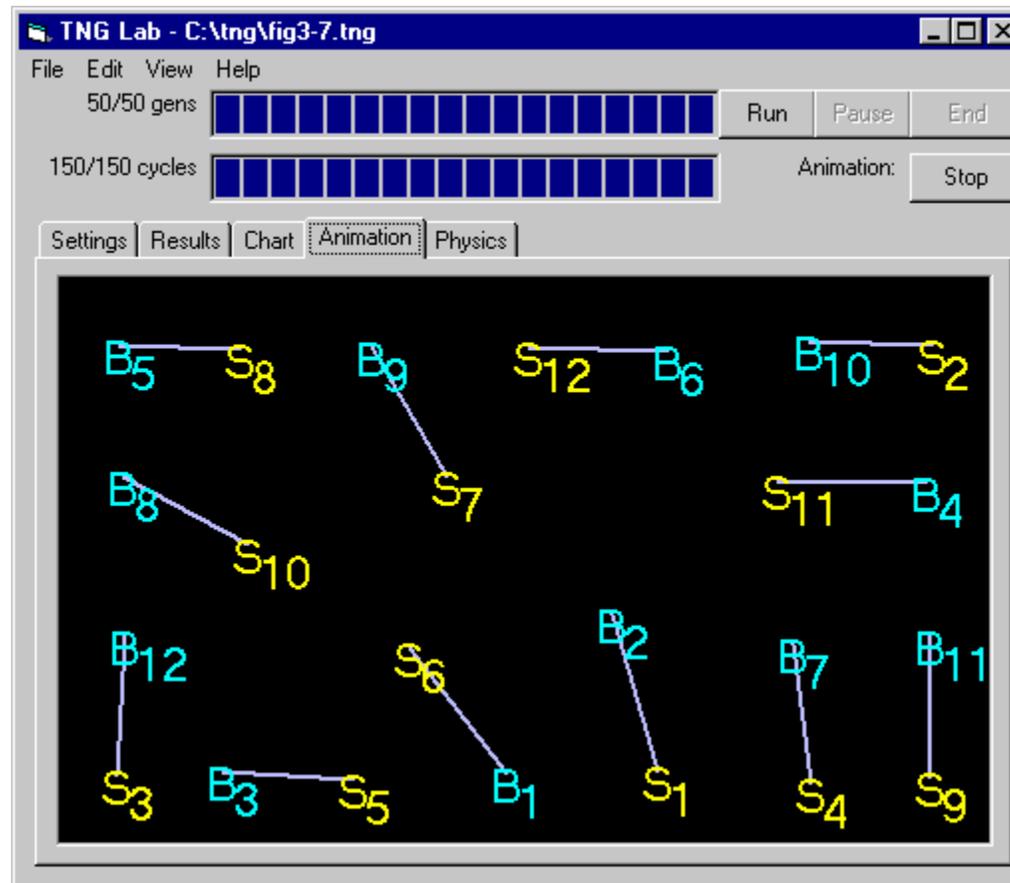
Fig. 6
TNG Lab GUI Animation Screen

Fig. 7
TNG Lab GUI Physics Screen
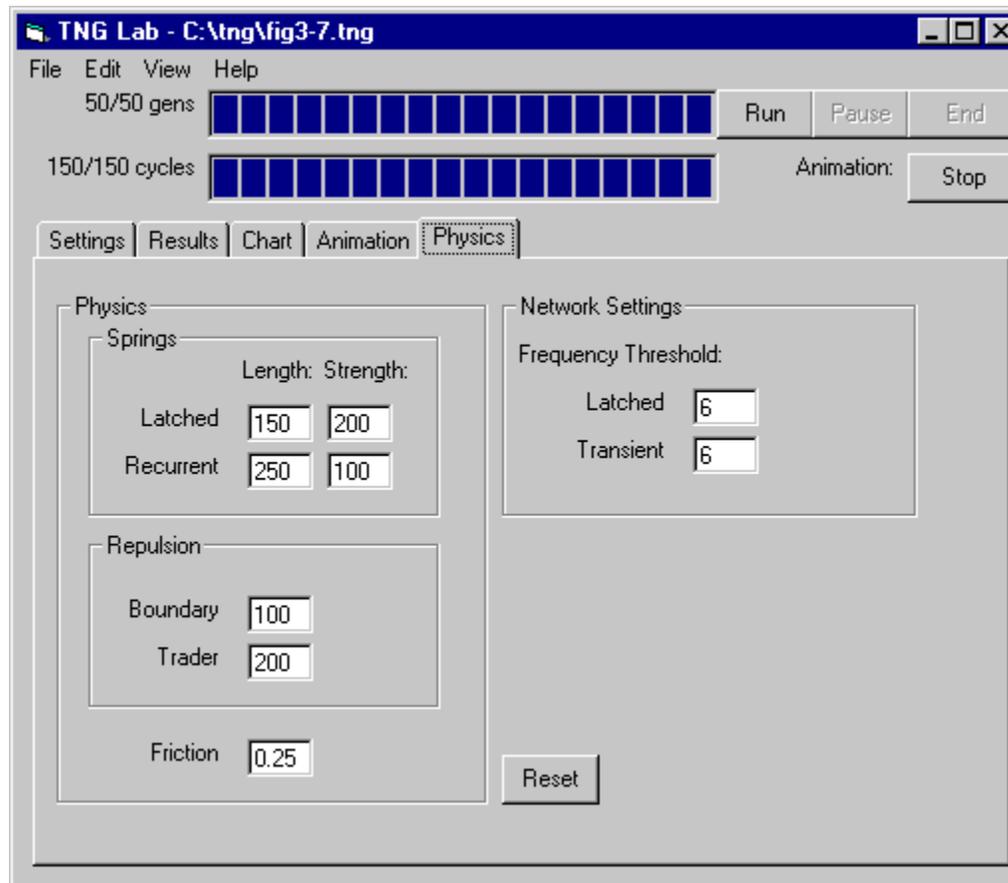
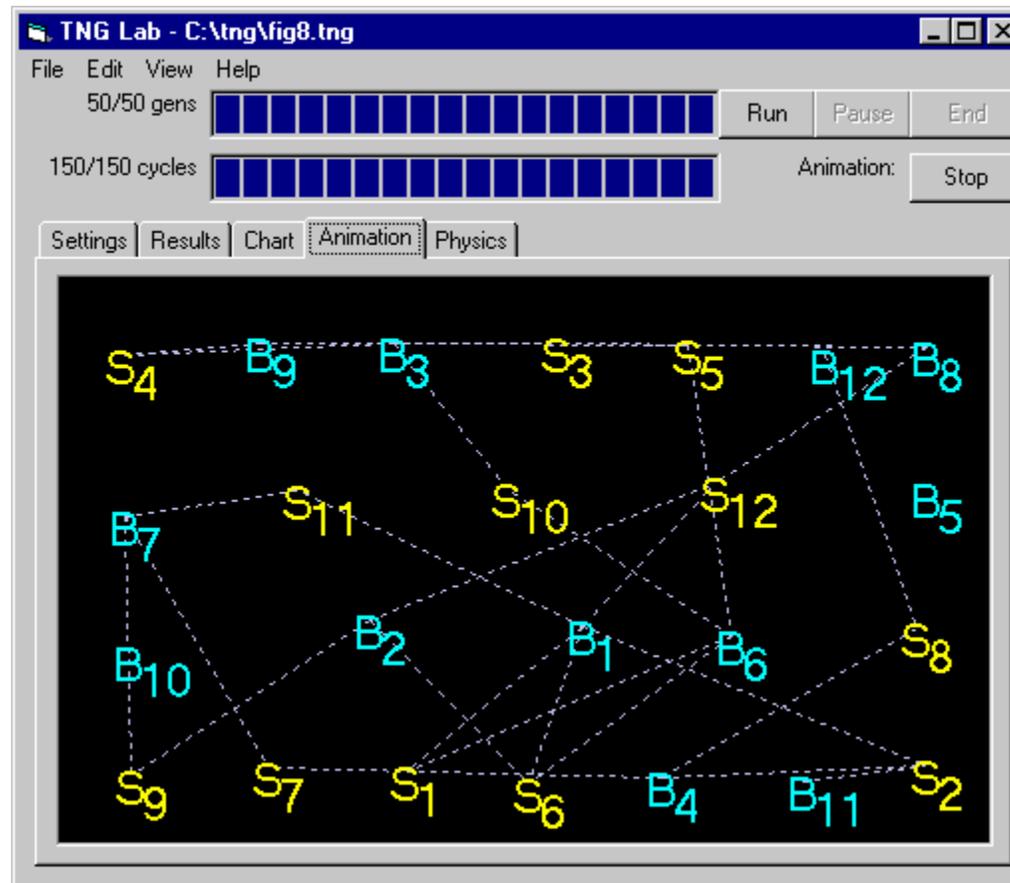# Fig. 8: Network Formation under Tight Job Capacity with D^o in 1-7

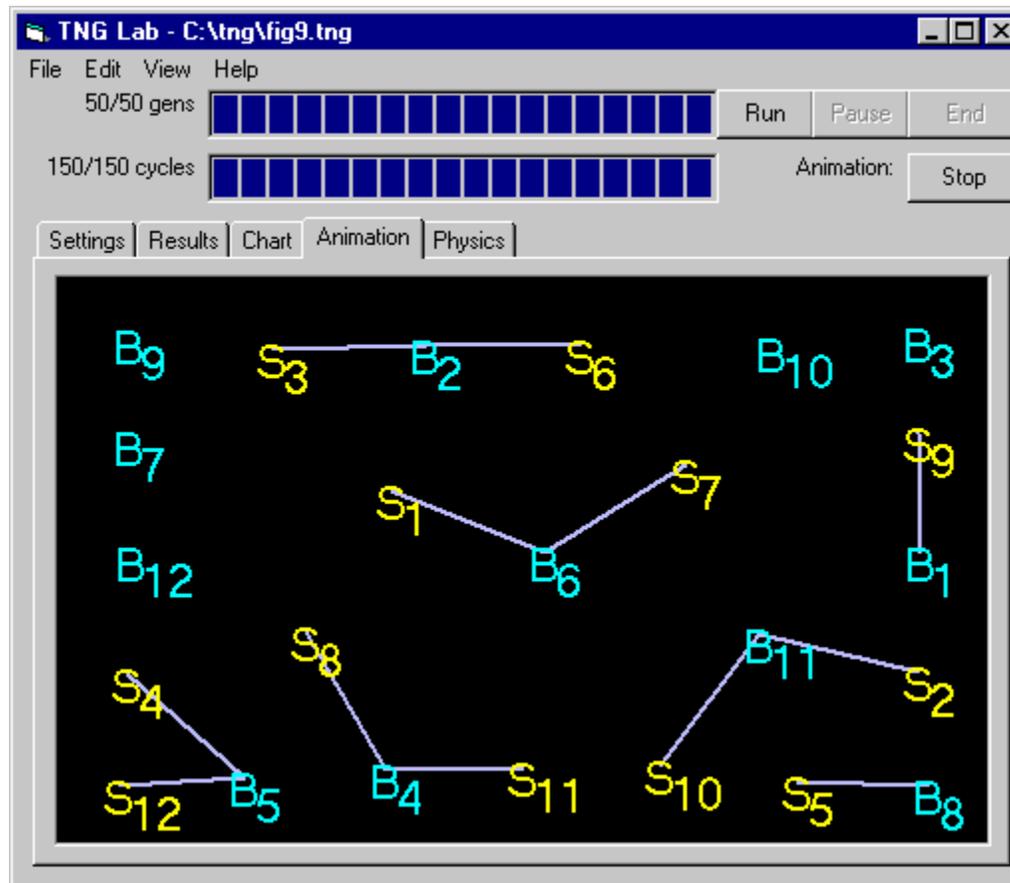# Fig. 9: Network Formation under Tight Job Capacity with D^o in 12-14

Fig. 10: Network Formation under Tight Job Capacity with $D^o = 24$