Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

INTRODUCTION & OVERVIEW OF "ARTIFICIAL LIFE" -- EVOLVING INTELLIGENT AGENTS FOR MODELING & SIMULATION

A. Martin Wildberger

Electric Power Research Institute
3412 Hillview Ave.
Palo Alto, CA 94304-1395 U.S.A.

ABSTRACT

"Artificial Life," despite its biological analogy and the hyperbole that its name implies, is really a collection of methods for building discrete event simulations with evolving multiple agents. It consists mainly of representing parts of systems or natural phenomena as individual active objects that may be both persistent and self-modifiable, operating on them with genetic algorithms or other evolutionary computing techniques and treating their multi-dimensional parameter (state) space discretely, by using cellular automata or similar coupled map lattices. The attractiveness of these methods for general purpose modeling and simulation lies in their ability to produce complex emergent phenomena out of a small set of relatively simple rules, constraints and relationships couched in either quantitative or qualitative terms. This tutorial includes brief introductions to both GA and CA, a description of a tool kit for building multi-agent simulations, and an outline of a current application to electric power systems and to the evolution of the electric power industry itself. References are included to demonstration software and source code available on the internet.

1 BACKGROUND

For at least fifteen years, many simulations, and simulation tools, have employed multiple agents, usually called "actors" or "demons" and represented by separate processes, which operate independently and interact by communicating selected information through messages. Until recently, these agents and their interactions have been specifically modeled to represent relatively unchanging functions, operations or physical entities. For most of these simulations, consistency and reproducibility were essential requirements. In general, artificial intelligence techniques were used only when necessary to model human decision making and were couched in the form of policy rules to be followed, for instance, by processes simulating additional crew members in a team trainer.

More recently, some researchers have been building intelligent agent simulations with some form of self-modifiability in an attempt to model natural or societal processes whose complexity would otherwise lead to exponentially increasing computational demands. The chief difference is that these agents change as the simulation progresses. They "evolve" by "adapting" their behavior in both competitive and cooperative ways to meet general goals which are assigned by the simulation designer but whose details may also change through the same "evolutionary" process. Like Monte Carlo, this simulation technique uses statistical methods to obtain an end result that is, nevertheless, deterministic and reproducible within the bounds of numerical precision, although the route and the CPU time taken may not be identical over all runs of the simulation.

These simulations go by various names such as: artificial life, animats or softbots, all of which may suggest more capability than is actually being delivered, and some of the practitioners of these simulations have been guilty of unrealistic "hype." Nevertheless, this approach has been particularly successful in modeling biological, bio-molecular and ecological phenomena, as might be expected since it is loosely based on a biological analogy. What is more surprising is the success these methods have had in modeling economic and financial processes such as commodity markets and currency exchanges.

The attractiveness of these methods for general purpose modeling and simulation lies in their ability to produce complex emergent phenomena out of a small set of relatively simple rules, constraints and relationships couched in either quantitative or qualitative terms. Inventing the right set of the local rules to achieve the desired global behavior is not always easy -- although it often seems obvious afterward. For instance, in a simulation of freeway traffic flow, simple rules that govern the actions of individual vehicles ("travel at such-and-such a speed," "don't hit anything") result in global behavior that is highly complicated (structured "platoons" of fast-moving autos form, as do convoluted traffic jams). In other examples, flocking arises in a simulation of multiple individual birds, and elasticity in polymers emerges from a simulation of molecular interactions.

A variety of evolutionary computational methods are employed in these simulations, but most are variations on John Holland's genetic algorithms (GA). (Holland 1975) The GA structure, based on individual membership in "species," most easily lends itself to modeling individuals as unique instances of more general (evolving) classes. These simulations typically include displays of the agent behavior over time in a physical space, if appropriate, or, more often, in a state or phase space. Therefore, cellular automata (CA) and similar spatially extended, coupled map lattices are also frequently used as another tool for intelligent agent based modeling.

2 GENETIC ALGORITHMS

Genetic algorithms (GA) provide a technique for the simulation of complex systems by modeling them in the form of multiple, interacting, relatively simplistic agents, represented simply by bit strings of ones and zeros, that evolve and adapt through competition and cooperation in a (usually) stable environment defined by the simulation-builder, modeled as another bit string and called a "fitness function" by analogy with biological evolution. The bits may stand for characteristics, capabilities, or relatively simple strategies which can be selected and recombined to form different individuals by "crossover" (cutting up two strings and patching them together) or by "mutations" (random changes in bit settings). By comparing randomly selected individuals' relative success with respect to the fitness function and then creating a new "generation" of individuals from crossover and mutation biased toward using the most successful individuals, eminently successful individuals are eventually "evolved."

Genetic algorithms are examples of a very broad class of systems that can be described as "adaptive non-linear networks" (Holland 1975) Some artificial neural network paradigms also fit into this class. All these systems consist of a large number of units that interact in a non-linear, competitive manner, and are modified by their own interaction or by external operators so that the overall system adapts to its environment. That environment is conventionally defined in terms of the system's input, its output, and some criterion, either internal or external to the system, by which its success in adapting can be measured. The competitive nature of this process does not necessarily imply a zero-sum game in which individual units within the system only benefit at the expense of other units. Combining and/or cooperating among units can be beneficial to the system, but ultimately the whole system adapts through change, growth, or even loss of individual units.

Genetic algorithms are based loosely on the theory of evolution, genetic diversity, and the "survival of the fittest." To use genetic algorithms to model individual agents, it is first necessary to invent, select, or otherwise define, a number of possible candidate strategies, rules or procedures to make available to each agent. Each agent may also be given some number of possible candidate attributes, qualities or characteristics. If any of these strategies or attributes are relatively complex in itself, it should be described in terms of its constituent parts or aspects, some of which it may have in common with other strategies or attributes. These (whole or partial) strategies and attributes may be considered analogous to biological genes, and an individual agent defined in terms of whether it has or does not have particular "genes".

Conceptually, all the agents evolve in parallel over a series of generations through competition and "selective breeding." Each generation of individuals is tested as to how well they perform on the basis of a specified fitness function, which results in an adjustment of their current "strength", the measure of their success. The next generation of new individuals are produced by genetic recombination and some mutation of the most successful individuals. Most unsuccessful individuals do not reproduce at all. Eventually almost all individuals belong to the same successful set. If we wish to maintain, or to evolve, two or more different classes of agents, we can limit the interbreeding to members of the same class (or species).

2.1 Mathematical Formulation Of GA

Mathematically, genetic algorithms are similar to other forms of "hill climbing" optimization using randomized gradients. However, they allow a convenient representation of a multi-dimensional problem space when different coordinates of that space are measured by continuous, discrete, and symbolic values. Genetic algorithms also provide for direct parallel implementation whereas conventional algorithms require the additional effort to convert them from their original sequential form.

To begin describing the computational implementation of genetic algorithms, it is first necessary to define formal classifiers. First, take as primitive elements, all possible binary strings: $m_1 m_2 \ldots\ldots m_j \ldots\ldots m_k$ (of length k, where m_j is taken from the set {1,0}, and m_j is either zero or one). These binary strings are called "messages". Next, define a "condition" to be any subset of messages specified by a string of the form: $c_1 c_2 \ldots\ldots c_j \ldots\ldots c_k$ (of length k, where c_j is taken from the set {1,0,#}, c_j is either zero, or one, or #, and the number sign, #, is a pure symbol that can intuitively be taken to mean: "don't care whether it's zero or one"). The subset of messages that meet a particular condition have, for all the positions in them:

(1) $m_j = 1$ when $c_j = 1$

(2) $m_j = 0$ when $c_j = 0$

(3) m_j is either 1 or 0 when $c_j = \#$

For instance, the message: "0001100" meets the condition: "00##100". (In this example, k = 7.)

Conditions can be combined according to all the usual Boolean operations: **and**, **or**, **not**, and any combination of them. Any set of one or more such conditions is defined to be a "classifier" on the space of messages.

The definition of "genetic classifiers" requires another level of abstraction. A "schema" is defined to be any subset of conditions. For length k = 11, an example of a schema is: s = *0##1****** where "*" means "don't care whether it's zero, one, or #". Members of the subset specified by a schema are called "instances" of that schema. Examples of two possible instances of the schema s would be: x = 10##1001010 and y = 00##1010011. A schema defines a subset of the set of all possible conditions, whereas each condition defines a subset of the set of all possible messages. Any set of one or more such schema is defined to be a "genetic classifier" on the space of messages.

As these genetic classifiers evolve through many generations, they must be tested for how well they meet the original fitness function. At time t, test the available instances of s and record their "strengths"; i.e.: their fitness. Then assign a "payoff" value v(s,t) to s at time t by averaging the strengths of all its instances. Let v(t) be the average strength at time t of all instances of all schema under consideration and let m(s,t) be the number of instances available of schema s at time t. Then bias the selective breeding so that, at some future time, t + T, the number of instances of s will increase or decrease according to the ratio of the strength of s to the average strength of all schema: m(s,t + T) = b[v(s,t)/v(t)]m(s,t) where b is an arbitrary (but strictly positive) constant.

The selection of strings for reproduction is performed randomly, but the probability of a given string being selected is made proportional to the normalized value of its strength ratio as described above. The total number of strings selected for each generation can be controlled so as to maintain the size of the population within tractable limits. Since a single condition is an instance of 2^k substring schema, explicit calculation of these ratings would be impractical. Genetic operators accomplish this same effect implicitly by constructing hybrid offspring from combinations of substring schema carried by high strength parents.

The simplest method of string reproduction is to copy without change the strings selected by the random process. This clones the most successful instances, and causes the "fittest" schema to grow exponentially until the population limit is reached. This approach usually converges rapidly but it assumes that one of the trial solutions is, in fact, the best. It does not provide for the "discovery" of any other solution than those candidates included by the algorithm designer. Other "genetic operators" combine parts of more than one string (selected randomly as above) to form wholly new individuals. These approaches are mostly variations on a basic operation called "crossover". Crossover is applied to a pair of strings as follows: select at random a position i, $1 < i < k$, and exchange the segments to the left of position i in the two strings. Crossover is considered to be analogous with sexual reproduction. Clearly, this method can be expanded by breaking the two strings in more than one place and/or by selecting pieces from more than two strings. In general, the simplest technique is as effective as the more complex ones, although the latter may speed convergence in certain cases.

Operators of the crossover type mix aspects of the trial solutions in an attempt to produce a better result than any one of the original set could achieve. However, they add nothing really new to that trial set. Another type of genetic operators, "mutation", can be used to produce random changes in a string, thus potentially inventing a wholly new strategy or characteristic. Experience with this type of operation indicates that the rate of mutation must be kept relatively low for best results.

2.2 Other Evolutionary Computing Methods

Two similar computing approaches have developed in parallel to GA. One, the "evolutionary programming" technique of Lawrence J. Fogel (1966) is based on evolution at the "behavioral" (performance) level--where selection for fitness actually takes place--rather than at the genetic level of GA. In this method, evolutionary change is produced solely through random mutation of the most successful candidates: no explicit connection is made between "behavioral" fitness and "genetic" change. Despite this difference, evolutionary programming generally produces results similar to GA. The second approach involves "evolutionary strategies," a technique introduced by Ingo Rechenberg (1973), and Hans-Paul Schwefel (1975). This method focuses on evolution of machine learning through trial-and-error search procedures with automated reinforcement of successful strategies.

Although evolutionary programming, evolutionary strategies, and GA developed separately and continue to represent areas of independent research, they are beginning to be viewed as different facets of evolutionary computing rather than fundamentally different techniques.

2.3 Cautions on the Use of GA

Genetic algorithms, and other evolutionary computing methods, must be used with care. They cannot be guaranteed to produce an absolutely optimum solution. It is also difficult to predict how many generations it will take to produce a satisfactory result. The inherent combinatoric explosion of which they are capable, is only avoided because most possible strings are never tested. Genetic algorithms are most beneficial when used in a parallel processing environment. Like neural networks, they can easily be simulated on a single sequential processor, but either the time or the computer power must be available to test hundreds of solutions in

each generation until a satisfactory set is obtained. Designing the genetic operators, describing candidate solutions as strings of genes, and defining the pay-off function are still more of an art than a science. The process must be tuned to guard against "super" individuals winning too soon and causing the irrevocable loss of genetic material. Furthermore, problem spaces can be constructed (using Walsh functions) whose maxima will never be found by a genetic algorithm.

3 CELLULAR AUTOMATA

Cellular automata (CA) are massively parallel computing engines that provide tools for simulating phenomena characterized by simple local rules and complicated, unpredictable global behavior. CA can produce complex "emergent" behaviors even though they do not appear to have been included in the original CA design. CA in use range from J.H. Conway's classic computer game "Life" (Conway 1982) to the ECHO and SWARM (Santa Fe Institute 1996) series of programs designed by researchers at the Santa Fe Institute (SFI) to simulate some of the emergent phenomena mentioned previously.

CA are built from an array of ordered sites; input data or boundary conditions are represented by the initial site configuration. Values assigned to sites change synchronously in discrete steps over time by application of relatively simple, local rules. This process is somewhat similar to the evolution of successive GA generations except that no external fitness function is used to evaluate computations: all information is endogenous.

Construction of a CA and examination of its operation is relatively simple; "programming" it to produce meaningful results, which requires selection of the correct initial configuration and proper behavioral rules, is not--and the programming considerations are usually difficult to express formally. Despite considerable effort, the mathematical fundamentals of CA design are still not well understood. However, the ability of GA to evolve good solutions to poorly formulated problems may provide an approach to improved CA design. Melanie Mitchell (Mitchell and Forrest 1993) and colleagues at SFI are investigating GA that can evolve an initial CA configuration and a set of local rules appropriate for specific computational tasks.

3.1 Mathematical Formulation of CA

The simplest CA can be built by starting with a linear array of zeros and ones: for example, 01100101, and applying a rule to construct another such array. For instance, the rule: "If both nearest neighbors are equal, change; otherwise stay the same," produces, sequentially:

01100101
11100010
10101001
11010001

In this example, it is assumed that the first and last cells are neighbors: i.e., the array is circular.

Although it is clear that this deterministic, finite machine will eventually repeat itself, it is not immediately obvious how soon repetition will occur, nor is it easy to predict the length and membership of the limit cycle into which any finite CA must eventually fall. Small changes in the starting values and in the rules can have significant effects. By displaying each new linear array beneath (or above) the last and using contrasting colors (or black and white) to represent the ones and zeros, it is possible to watch the sequence of developing patterns and immediately recognize stable states or limit cycles. CA of higher dimensions and/or more complex rules can evolve to chaotic, aperiodic patterns that are the analog of strange attractors in dynamical systems. Since these extremely simple CA produce global effects from strictly local operations, they make it possible to model the emergence of the most complex known phenomena, perhaps life itself, from multiple, simple activities by many individual elements that appear to be affecting only their immediate neighbors.

3.2 CA Notation and Terminology

The simplest form of finite CA is made up of a lattice of L cells, each of which, at time t, can be in one of K states. A single fixed rule is used to update the state of each cell. The rule may be different for each cell, but the rule associated with each cell is not subject to change. The rule for each cell operates on that cell simultaneously with all the rest to convert all their states, (called the configuration or global state of the CA) at time t, to a new configuration at time t+1. The rule is a mapping from the current state of the cell and that of its nearest neighbors to the updated state of the cell in question. The number of neighbors that enter into the rule is known as the cell's radius, r. Note that the terminology in this paper is used by most, but not all researchers in the field.

The rule is usually the same for all cells and is often expressed as a table. The following is a rule table for a binary-state (0,1) nearest neighbor (r=1) CA. All 8 possible neighborhood patterns are shown on the left while the right hand bit shows the rule's output bit, which becomes the center cell's value at the next time step.

000 -> 0
001 -> 0
010 -> 0
011 -> 1
100 -> 0
101 -> 1
110 -> 1
111 -> 1

This rule is known in shorthand as #232 because the output values for sequentially increasing inputs make up the binary version of that number, i.e.:
$11101000_2 = 232_{10}$

CA can be extended to any number of dimensions, the number of cells allowed to become infinite, and their rules made as complicated as desired. The rules can include probabilities, and uncertainty (noise) can also be allowed to affect the accuracy with which values can be read.

3.3 Visualizing CA

There are two common ways to visualize CA. A state space (or phase space) plot is best for showing space-time patterns, but a state transition graph is more effective in illustrating CA behavior as a non-linear dynamical system. Software is available on the Internet (Wuensche 1996) that demonstrates these visualizations and allows the user to explore various rules and initial configurations for CA.

3.3.1 Space-Time Patterns in State Space

A finite sized CA may be bounded in any way that is desired, but the boundary cells will then require special rules to handle their unique situations. The simplest CA architecture is a small, 1-D array of cells of length L, with binary values (K=2) and a small local neighborhood ($r<<L$). This array is treated as circular, thus creating periodic boundary conditions which do not require any special rules for cells 1 and L since they are considered to be neighbors. Evolution of the CA may be represented an a sequence of configurations wrapped around a cylinder.

However, it is convenient to split the cylinder between cells 1 and L, flatten it out, and represent its evolution as a 2-D space-time pattern, with space running across the screen or page, and time progressing downward. By painting pixels in differing colors depending on the value of each cell (or black and white in the case of a binary valued CA), it is easy to detect patterns, observe static, periodic, or complex (possibly chaotic) behavior, and to compare the effects of different rules and different initial configurations. Although many researchers have developed rule classification schemes on the basis of such space-time patterns, these effects are necessarily phenomenological and not all observers may detect the same patterns.

3.3.2 State Transition Graphs

A finite, deterministic CA must eventually repeat a configuration (or global state) that occurred at some earlier time. Its trajectory will then become trapped in a repeating sequence of the same configurations. This becomes a cyclic attractor, with a period of 1 or more, but no greater than K^L. All configurations are either part of one such attractor or belong to some transient sequence of configurations that lead to an attractor. Configurations at the beginning of such a transient cannot be reached from any other such global state of the CA and can only exist if the creator of the CA initializes it in one of those configurations. (These configurations have become known as "garden-of-Eden" global states.) All the configurations on a cyclic attractor, along with all configurations on transient trajectories leading to that attractor, make up its basin of attraction. Any particular CA may have one or many such basins, but it is impossible to transition from a configuration in one of these basins to a configuration in any other one. All of the basins of attraction for a particular CA make up that CA's basin field.

If we represent each configuration of a CA by drawing a small circle, and connect each of those circles to all the other circles that represent just those configurations that can transition to the first state, we will produce a directed graph that links all of the configurations belonging to each basin of attraction for that CA. This graph of the entire basin of attraction field displays all the possible dynamics of that CA. The set of basins partitions all the configurations of the CA into disjunct classes. The CA cannot evolve from one basin to another.

In any finite CA, it is always theoretically possible to compute the complete basin of attraction field by exhaustively working either forward (as described above) or backward (by computing all the pre-images of each configuration). However, the time to compute pre- or post- images increases exponentially with lattice size so that this approach is impractical for larger arrays. Wuensche and Lesser (1992) have developed a computational shortcut, the "reverse algorithm," which directly computes pre-images and improves the computation time by several orders of magnitude. They have also defined a parameter, Z, as the probability that the next unknown cell in a partial pre-image can be uniquely determined from the values of the cells already known. A relatively high value for Z implies the ability to compute all configurations of a CA without an exhaustive search.

3.4 Using CA as System Simulators

The remarkable computing capability of CA make them useful tools for modeling and simulation, especially for phenomena whose local actions are easily described but exhibit complex behavior of mixed deterministic and random origins.

As a very simple example of pattern recognition or image enhancement that is, nevertheless, actually used in pixel-based image processing, consider this CA that automatically computes whether its initial configuration contained more black or white sites. The CA is represented as a looped string of m ordered, black-or-white sites. It follows the rule:

the color of any site at time = t_{n+1} is the color of the majority of the site and its two neighbors at time = t_n

Eventually the CA becomes all white or all black, reflecting the initial majority.

In an area of more engineering interest, Professor Kai Nagel of the University of Cologne has constructed a CA model for freeway traffic. (Nagel and Rasmussen 1994) The deterministic version produces a high/low density phase transition at the point of maximum throughput with self-organized criticality driven by the speed of the slowest car. Adding random noise causes spontaneous formation of traffic jams. Their distribution responds to a scaling law near the critical point and expansion of the scaling region can be characterized by the percentage of drivers using "cruise control" to reduce their fluctuations at high speed.

Lattice gases (LGA) constitute a distinct subclass of CA that can simulate complex fluid dynamics governed by non-linear equations such as the Navier-Stokes equation. There have been some significant recent developments in programming languages and computer architectures that strongly support CA (and especially LGA) based parallel computations. (Margolus 1995, Toffoli 1995)

3.5 Modeling Wave Equations with CA

In a project sponsored by EPRI at San Jose State University, Prof. Rudy Rucker and his students are developing techniques for using complex cellular automata to model certain aspects of the electric power grid. The long term goal is to perfect a distributed-computation simulation of the global behavior of a circuit (for instance, its stability) based on the local behavior of the individual components, and to apply this approach to power quality (harmonics, load induced transients, etc.) where CA cells might represent individual loads (such as electrical appliances) within a building.

A cellular-automaton implementation of partial differential equations has been developed, including: the heat equation, the wave equation (or telegrapher's equation), the damped driven oscillator, the damped driven oscillator coupled with the wave equation, and the Fermi-Pasta-Ulam non-linear soliton wave. These simulations have never before been implemented as pure cellular automata. The CA implementation makes it possible to rapidly explore alternate parameter settings, to breed and mutate parameter settings, and to view space-time diagrams of the simulation in real time. These simulations can run rapidly on a standard desktop Windows computer. Research is continuing on how best to use these techniques for real world problems, as well as on extending these techniques to branching networks and to higher-dimensional systems. Version 3.0 of CAPOW (Cellular Automata Power Simulator) has been posted on the Internet. (Rucker 1996)

This modeling approach, while it makes use of computations carried out by local rules on a spatially extended lattice, does not include any explicit intelligence or agency in the individual cells. However, the possibility of using genetic algorithm-based agents to guide the evolution of the CA is being explored. The rules developed in these CA-based experiments can also be provided as methods for the agent-objects used to model components in a distributed agent simulation of the electric power grid. (See Section 5 below.)

4 SWARM -- A TOOL FOR MULTI-AGENT SIMULATIONS

The SWARM Simulation System has been developed by the Santa Fe Institute as a tool kit for the study of complex adaptive systems using multi-agent discrete event simulation. The full source code is available free. (SFI 1996) It requires the use of the GNU C Compiler, UNIX, and X-Windows.

The basic unit of a SWARM simulation is the agent, which may be any entity that can generate events that affect itself and other agents. Simulations consist of groups of many interacting agents, called "swarms," which may, themselves, be grouped into more comprehensive swarms. The logical structure of swarms of agents interacting through discrete events is implemented in Objective C, an object-oriented (OO) language. As in most OO programming, software consists primarily of definitions of various classes of objects. An object is then a combination of instance variables for the object's state and methods (services, procedures) that implement the object's behavior. Each object carries with it its own state variables, but the generic definition of its behavior is provided by the class.

The SWARM system itself is an object framework: a set of class libraries that are designed to work together. These include support libraries with potential use outside of SWARM as well as domain-specific libraries that provide facilities for building two dimensional discrete lattices, genetic algorithms, and neural networks. Agents are created by taking a class from the SWARM libraries, specializing it for the particular modeling domain, and then instantiating it, with each instantiated object an agent. A schedule of discrete events involving the agent-objects defines a process occurring over time and is implemented as a partially ordered series of actions to be performed by objects on objects. In the simplest case, a model consists of one swarm inhabited by a group of agents and a schedule of activity for those agents. In SWARM, the environment is itself modeled by one or more other agents. In the general case, the environment for each agent consists of many or all of the other agents, some of whom might have a larger influence than others.

In addition to all the common OO facilities, SWARM implements a "probe" facility which allows

any object's state to be read or set and any method to be called in a generic fashion, without requiring additional user code. Probes are used to make data analysis tools work in a general way and are also the basis of graphical tools to inspect objects in a running system. SWARM also provides data collection tools in the form of observer agents, special objects whose purpose it is to observe other objects via the probe interface. The observer agents themselves are a swarm, a group of agents and a schedule of activity. By combining this swarm with a model swarm running as a subswarm of the observer, a full experimental apparatus is created. By using hierarchical swarms to separate data collection from the model, the model itself remains pure and self-contained, a simulated world under glass. Different observer swarms can be used to implement different data collection and experimental control protocols, but the model itself remains unchanged.

The SWARM software package is currently being used in a research project started in 1996 with the support of the Electric Power Research Institute (EPRI). Professor V. C. Ramesh at the Illinois Institute of Technology is investigating a multiple agent approach to contingency analysis. All electric utility control centers run computer simulations continuously to test whether any single accident or failure will cause any security constraint to be violated for longer than a specified short time period and thus lead to the possibility of a serious blackout. Prof. Ramesh is retaining the conventional model of the problem as constrained non-linear flow, but he is building (five to ten) individual intelligent agent solvers, each of which will attack the solution of the model in a different way. He expects that their cooperation and competition will achieve a faster convergence than could any one alone.

5 MODELING THE ELECTRIC POWER GRID

The North American power network may realistically be considered to be the largest machine in the world since its transmission lines connect all the electric generation and distribution on the continent. A design for distributed control of an electric power system by intelligent agents operating locally with minimal supervisory control is being developed systematically to include modeling, computation, sensing and control. (Wildberger 1994)

The distributed intelligent agent model of the electric power grid is being developed to serve two purposes:

1. Concurrent simulation of the grid for real-time distributed control using new developments in active, high-power, electronic control devices and self-calibrating, self-diagnostic sensors. (Wildberger 1994)
2. "What if" studies and computer experiments intended to provide insight into the evolution of the entire electric power industry under various forms of organization imposed or evolved as the result of varying degrees of deregulation, competition and unbundling of services.

Since intelligent agents have been used successfully to model the traders in commodities and the commodity futures markets, model development began by using artificial agents represent the buyers and sellers of bulk power. Software developed by Prof. Gerald Sheblé at Iowa State University, based on a form of genetic algorithm, permits the user to assign combinations of relatively simple strategies to a small number of agent-traders and to compare their profits and losses as the market clears. Rather than the bilateral trading between utilities which has been the rule in the past, this models the "power pool" arrangements that are now being established both by institutions and private parties. The trading model is being extended to include futures trading and retail as well as wholesale contracts, but, in order to include the effects of each transaction on power flow and stability in the electrical network itself, the network's physical components must also be modeled as intelligent agents.

The basic concept is to represent each component of the network by an agent of limited intelligence which seeks to ensure its own survival while optimizing its performance in the context of all the other agents. For instance, a single bus will strive to stay within its voltage and power flow limits while still carrying the voltages and power flows imposed on it by the combination of other agents representing generators, loads, transformers, etc. Specific classes of components will have additional survival constraints which may not be exceeded or, if exceeded, would require replacement or off-line maintenance. For instance, the chemical state of the oil bath in a large transformer must stay within specified limits for safe operation. It's state may be tested periodically by taking samples, or instrumentation may be installed for continuous monitoring. The cost-benefit of this instrumentation for each transformer is one example of management information this distributed model and simulation is intended to provide.

More complex components, such as a generating plant or a substation, must be modeled as class and object hierarchies of simpler components. An Integrated Knowledge Framework (IKF) has been developed that describes the management and operational functions at a generic, coal-fired, steam power plant as an OO model containing 422 classes, interconnected through three different types of relations. (EPRI 1996) This framework identifies the data, information and knowledge required for the important functions typically performed at a fossil power plant, as well as the flow of that data, information and knowledge between the function that generates it and those that use it. Although the IKF was not developed solely as a design for distributed agent control and it contains many classes which would not be appropriate for instantiation as active objects, this model defines, at an abstract level, all the structure needed for an

intelligent agent model of the generation aspects of electric power.

6 SUMMARY

What is generally called "Artificial Life," is essentially a combination of methods for building discrete event simulations with evolving multiple agents As such, it provides another important tool for Simulationists to use in the modeling and analysis of many complex adaptive systems produced by humans either through engineering design or as a (possibly unintended) result of their social and economic behavior. The tools for artificial life simulations include a variety of evolutionary computing techniques operating on a coupled map lattice. Genetic algorithms and cellular automata, respectively, are the most common ones used. These methods are particularly useful in modeling and simulating very large and complex systems for two major reasons:

1. their ability to produce complex emergent phenomena out of a small set of relatively simple rules, constraints and relationships couched in either quantitative or qualitative terms
2. their inherent parallel, distributed structure

REFERENCES

Conway, J.H. 1982. What is Life? In *Winning Ways for Your Mathematical Plays*. eds. E. Berlekamp, J.H. Conway and R. Guy, Vol. 2, Chap. 25. New York, NY: Academic Press

Electric Power Research Institute. 1996. Integrated Knowledge Framework (IKF) for Coal-Fired Power Plants. EPRI Technical Report TR-106211-V1/2/3, (Mar). Pleasant Hill, CA: EPRI Dist. Ctr.

Fogel, L.J., A.J. Owens, and M.J. Walsh. 1966. *Artificial Intelligence Through Simulated Evolution*. New York: Wiley

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

Koza, J.R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge. MA: MIT Press.

Koza, J.R. 1994. *Genetic Programming II: Scalable Automatic Programming by Means of Automatically Defined Functions*. Cambridge. MA: MIT Press

Margolus, N. 1995. Ultimate Computers. In *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing* (San Francisco, Feb. 15-17) eds. D.H. Bailey et al. 181-186. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Mitchell, M. and S. Forrest. 1993. Genetic Algorithms and Artificial Life. Technical Report SFI 93-11-072, Santa Fe Institute, Santa Fe, NM.

Nagel, K. and S. Rasmussen. 1994. Traffic at the Edge of Chaos. Technical Report SFI 94-06-032. Santa Fe Institute, Santa Fe, NM.

Rechenberg, I. 1973. Evolutionsstrategie: *Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Frommann-Holzboog

Reynolds, C.W. 1987. Flocks, Herds, and Schools: A Distributed Behavioral Model." In Proceedings of SIGGRAPH'87. *Computer Graphics* V 21(4): 25-34.

Rucker, R. (rucker@jupiter.sjsu.edu). 1996. "CAPOW3B?.ZIP," from WEB site: http://www.mathcs.sjsu.edu/capow/.

Santa Fe Institute (sfi@santafe.edu). 1996. ftp://ftp.santafe.edu/pub/swarm/, or from WEB site: http://www.santafe.edu/projects/swarm/.

Schwefel, H-P. 1981. *Numerical Optimization of Computer Models*. Chichester, UK: Wiley

Toffoli, T. 1995. Fine-Grained Models and Massively-Parallel Architectures: The Case for Programmable Matter. In *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing* (San Francisco, Feb. 15-17) eds. D.H. Bailey et al., 181-186. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Von Neumann, J. 1949. Theory of Self-Reproducing Automata. In *1949 University of Illinois Lectures on the Theory and Organization of Complicated Automata*. (reprinted 1966) ed. A.W. Burks. Urbana, IL: University of Illinois Press

Wildberger, A.M. 1994. Automated Management for Future Power Networks: A Long-Term Vision. *Public Utilities Fortnightly* **132**, 20 (Nov): 38-41.

Wuensche, A. 1996. http://www.santafe.edu/~wuensche

Wuensche, A. and M.J. Lesser. 1992. *The Global Dynamics of Cellular Automata: an Atlas of Basin of Attraction Fields of One-Dimensional Cellular Automata*. (diskette included). Santa Fe Institute Studies in the Sciences of Complexity. Reference Vol. I. Reading, MA: Addison -Wesley

BIOGRAPHY

Dr. A. Martin Wildberger manages Strategic Research and Development projects in Applied Mathematics and Information Science at the Electric Power Research Institute (EPRI). He also provides institute-wide applied research support in mathematical modeling and computer simulation.

Dr. Wildberger received his B.S. degree (cum laude) from Fordham University, his M.S. degree from the U.S. Naval Postgraduate School Engineering School and his Ph.D. from the Catholic University of America. He has published or presented over sixty technical papers and has been, since 1987, the editor of a monthly column in *Simulation* entitled: "AI and Simulation"