Optimizing System Behavior using Genetic Algorithms

Robert M Sholtes
Decision Dynamics, Inc
8601 Georgia Ave. Suite 806
Silver Spring, MD 20910, USA
Tel: 301 565 4040
Fax: 301 565 4045

Abstract

This paper explores the use of genetic algorithms (GAs) for optimizing system dynamics models. System dynamics offers a unique and powerful approach to identifying the most successful policies for managing complex problems. Unfortunately, policy makers too often avoid the use of models because of the high level of expertise required to operate the models and the time and expense which results from trial and error testing of a multitude of policy options in order to discover the best policies. The role of system dynamics models as decision-support tools would be greatly strengthened if model users could simply identify the goals for the system being modeled and have the system dynamics model identify the best management actions. Current analysis and optimization techniques used with system dynamics models are not capable of automatically determining which policies most nearly produce the desired system behavior. One emerging optimization technique, (GAs), offers great promise in automating the identification of the best policies for selected system goals.

The paper is divided into three sections. The first explains how genetic algorithms work. The second section demonstrates how GAs can be used to optimize system performance for a more complex model. The paper concludes with a discussion of the advantages and limitations of GAs as they relate to the needs of the system dynamics community.

Optimizing System Behavior using Genetic Algorithms

Introduction

System dynamics offers a unique and powerful approach to identifying the most successful policies for managing complex problems. Unfortunately, policy makers too often avoid the use of models because of the high level of expertise required to operate the models and the time and expense which results from "trial and error" testing of a multitude of policy options in order to discover the "best" policies. The role of system dynamics models as decision-support tools would be greatly strengthened if model users could simply identify the goals for the system being modeled and have the system dynamics model identify the best management actions. Current analysis and optimization techniques used with system dynamics models are not capable of automatically determining which policies most nearly produce the desired system behavior. One emerging optimization technique, genetic algorithms (GAs), offers great promise in automating the identification of the best policies for selected system goals. This paper explores the use of GAs as a tool for optimizing system dynamics models.

This paper is divided into three sections. The first explains how genetic algorithms work. A simple example shows how GAs can be used to determine the value for a single model parameter in order to achieve dynamic equilibrium. The second section demonstrates how GAs can be used to optimize system performance for a more complex model. The Kaibab Plateau model is used for these exercises. The paper concludes with a discussion of the advantages and limitations of GAs as they relate to the needs of the system dynamics community.

1. How GAs work

What are GAs?

Genetic algorithms are highly parallel, mathematical, adaptive search procedures based on the processes of natural selection and genetics. GAs apply the evolutionary concept of "survival of the fittest" to optimize mathematical problems which are not easily optimized by traditional numeric methods. One unique feature of GAs is that they are "blind". That is, the technique is not problem dependent. This makes GAs applicable to any system dynamics model. Although simple in concept and structure, GAs are highly effective in optimizing complex systems.

How do GAs differ from traditional optimization methods?

GAs are different from more traditional optimization methods used by system dynamicists in three ways: [Goldberg 1989]

1. GAs work with a coding of the parameter set, not directly with the model parameters.
2. GAs search from a population, not a single point.
3. GAs use probabilistic rather than deterministic transition rules

1. Coding

GAs employ binary strings to represent the model parameters used in an optimization. For example, a model parameter whose value can range from 0 through 31 would be coded as a five digit binary string for use by the GA. Each parameter to be included in the optimization is added to the string much as genetic information is stored in bits of DNA. Strings permit GAs to optimize a large number of parameters as easily as they can optimize a single model parameter. Strings are also used to facilitate the genetic processes of reproduction, crossover and mutation.

2. Population

GAs work with a population of binary strings rather than a single string for optimization. Each member string of the population is evaluated for fitness before proceeding with the next round of optimization. More fit member strings have a greater chance of producing children for the subsequent generation. The use of a population prevents optimizing to local minima or maxima and provides greater robustness while searching for optima within the solution space.

3. Probability

Probability plays a role in determining the best members of a population for reproduction, crossover and mutation. While probability is central to GAs operational efficiency, GAs are more than just a random search through the solution space. [Goldberg 1989] Succeeding generations in a GA optimization show a steady improvement in system performance over the course of the optimization that reflects more than mere chance.

GAs in operation

The basic steps required to apply the classic (simple) GA to system dynamics models is outlined below:

```
Code the policy parameters as a finite length bit string
Create an initial population of size N with randomly set parameter bit strings
While a suitable solution has not been found {
        For each member in the population {
                Decode the string into model input values
                Run a simulation using the decoded values
                Calculate the fitness for the member based on simulation results
        }
        Using probability selection, choose members for mating
        While the new population is smaller than N members {
                Randomly choose two parents from the mating pool
                Apply the genetic operation of crossover to create two child strings
                Apply the genetic operation of mutation to each child
                Add the child strings to the new population
        }
}
```

To demonstrate the application of GAs, consider a search for the value of Percent_In which provides a steady-state solution for the simple model shown in Figure 1. Although the solution is obvious in this example, Percent_In = 1/Avg Lifetime, the GA does not know this. While estimating the value of a single model constant is not the type of problem one would normally use GAs to solve, it does provide an excellent example for demonstrating how GAs can be applied to system dynamics models.[1]

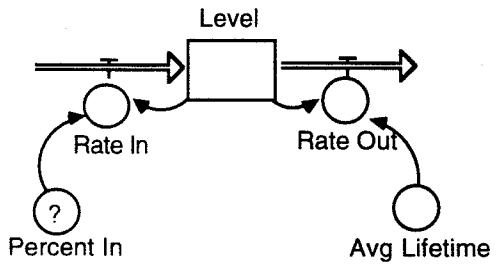

Figure 1 - Determine Percent_In for steady-state

Before the GA can search for the optimal value of Percent_In, an initial population must be generated. The GA uses a five bit binary string to represent the value for Percent_In, where Percent_In = 1/string value. Initial string values are generated using a Bernoulli random variable with probability = 0.5.

[1] All GA research presented in this paper was completed using the Microsoft Visual C++ compiler for Windows.

#	String	String_Value	Percent_In	Fitness
1	10110	22	0.0454	0.88
2	10010	18	0.0555	0.72
3	10011	19	0.0526	0.76
4	00101	5	0.2000	0.20
:	:	:	:	:
39	11000	24	0.0417	0.96
40	01011	11	0.0909	0.44
			Sum Fitness:	32.12

Figure 2 - Generating an initial population

Figure 2 shows a portion of the initial population. The binary string, the decimal string value, the decoded value for Percent_In and the fitness are shown for each member of the initial population.

While binary strings are useful for applying the GA algorithms, they must be translated into decimal values for use by the model being optimized. A decoding algorithm must be written for each parameter. Numerous techniques exist for decoding bit strings. [Goldberg 1989] The actual decoding scheme chosen depends on the range of values and precision required for each parameter.

Once decoded, the value of Percent_In can be used by the system dynamics model. The model is run once for each member of the current population using the member's decoded value of it's bit string. The results of each simulation are evaluated and a fitness value is assigned to each member of the population. In this example, fitness is determined by measuring the percentage difference between the value of the rate in and the rate out. The best fitness occurs when the rates are equal, yielding a fitness value of 1. Strings which provide less than optimal results have a lower fitness value. Figure 2 shows the fitness values for members of the initial population. For example member #39 has a fitness equal to 0.96.

Relative fitness is used to select strings for creation of the next generation. Individuals with higher fitness are more likely to be selected for reproduction. There are many methods available for choosing who will generate offspring. [Goldberg 1989] The examples described in this paper use the simplest method called roulette selection. This method approximates the use of a biased roulette wheel where members with higher fitness receive a proportionately greater chance of selection.

A new generation is created through reproduction. Reproduction occurs by selecting two members from the mating pool at random to form a mating pair. For each mating pair, the crossover and mutation operations are applied to the strings to create two child strings. To demonstrate the process of reproduction, assume that members #3 and #39 from the initial population are chosen for mating. First, the two parents are paired. Next, a cross-over point is randomly generated. In this example, a crossover point is chosen between the fourth and fifth bit string position. This operation creates two string segments for each parent string. The first child is generated by combining the first string segment of parent #3 with the second string segment of parent #39. The second child combines the first string segment from parent #39 with the second string segment of parent #3. The decimal value, Percent_In and fitness are shown in Figure 3 for the two new strings.

Parent	String	Children	Value	Percent_In	Fitness
#3	1001 | 1	10010	18	0.0555	0.72
#39	1100 | 0	11001	25	0.0400	1.00

Figure 3 - An example of reproduction

Before adding the children to a new population, each string undergoes the process of mutation. Mutation randomly changes bits within the newly created strings, albeit at a very low probability such as 0.5%. Mutation maintains variability within the population, reducing the chances that the population will converge prematurely on one possible sub optimal solution. Figure 4 shows one example of mutation. In this example, the fourth bit of the child string is mutated.

	Before Mutation	After Mutation
Child:	10011	10001

Figure 4 - An example of mutation

Mutation prevents one superior individual in a population from dominating the optimization process. There are alternative numeric techniques which prevent sub-optimization (such as larger populations or more sophisticated reproduction methods). However, mutation introduces enough variability into a simple GA to push the population toward better solutions.

Bringing it all together

The steps of decoding, simulation, fitness evaluation and reproduction are repeated from generation to generation as the GA searches for the best solution. Figure 5 concludes this introduction to GAs by showing the results from the search for the best value of Percent_In given an average lifetime of 25. The results show convergence of the average population fitness toward the optimum. The results of this exercise correctly identify the best value for Percent_In as 0.40.

		Fitness Values	
Generation: 1	Min: 0.000000	Max: 0.961538	Avg: 0.796714
Generation: 2	Min: 0.076923	Max: 0.961538	Avg: 0.830217
Generation: 3	Min: 0.000000	Max: 0.961538	Avg: 0.814076
Generation: 4	Min: 0.000000	Max: 0.961538	Avg: 0.827518
Generation: 5	Min: 0.000000	Max: 0.961538	Avg: 0.864780
:			
:			
Generation: 47	Min: 0.684211	Max: 0.958333	Avg: 0.922189
Generation: 48	Min: 0.684211	Max: 1.000000	Avg: 0.930825
Generation: 49	Min: 0.000000	Max: 1.000000	Avg: 0.893270
Generation: 50	Min: 0.214286	Max: 1.000000	Avg: 0.924980
Generation: 51	Min: 0.750000	Max: 1.000000	Avg: 0.955440

Figure 5 - Summary of GA results for optimizing Percent_In

2. Optimizing Model Behavior

With an understanding of GAs in place, it is time to turn our attention to the real power of GAs: optimizing behavior for a complex system. Two test cases are presented. The first optimizes the values of four constant parameters. The second test shows how GAs can be used to generate data for model table functions. Both tests are conducted using the Kaibab Plateau model. [Goodman 1983, Roberts 1983]

The Kaibab Plateau model was chosen for three reasons. First, it is a model most system dynamicists are familiar with. Using a familiar model permits modelers to evaluate the utility of GAs without needing to first evaluate the validity of the model used for optimization. Second, a number of well-established policy questions have been identified for the Kaibab model. The tests described here are based, in part, on these documented policy questions. [Roberts 1983] Finally, the Kaibab model was considered a good model for testing GAs since the model is capable of producing a wide range of dynamic behavior.

Test 1: Fixed deer and predator hunt rates

The Kaibab Plateau is home to deer, deer predators and numerous other fauna. In this first test, the GA is used to help establish policies which support a diverse eco-system. A diverse eco-system cannot exist if the deer population dominates on the plateau. In this test case, then, the policy goal is a steady population of 10,000 deer. This population size is small enough to ensure food for other animals, yet large enough to support healthy deer and predator populations and meet the needs of sport hunters. The target year for an initial population of 10,000 deer is 1910 with the simulation time running from 1900 through 1950.

To achieve this policy, four variables may be altered. They are the hunt rate for deer, the hunt rate for predators, the year which hunting is allowed to start for deer and the year in which predator hunting may begin. Once established, the hunting rates will not be changed over the next fifty years.
To conduct the optimization, the four policy variables are coded as follows:

Deer and Predator hunt rate: 7 bit string with the hunt rate equal to (1/200)
Hunt start years: 5 bit string with the hunt start year equal to (1900 + string value)

The coding for the hunt rate permits annual hunting rates between 0% and 63.5%. The coding for the hunting starts limits the values between 1900 (the start of the simulation) through 1931. This formulation assumes that any hunting which starts beyond 1931 would be unable to generate the desired deer population.

Fitness is calculated by summing the square of differences between the desired deer population of 10,000 head and the actual deer population generated by the simulation. This value is normalized and converted into a fitness value with 1 representing a perfectly constant deer population. Zero fitness is associated with results where the average difference between desired and actual population is more than twice the desired population.

Figure 6 shows the results for this first optimization test. The optimized policy limits the growth of deer population. The optimized policy prevents the deer population from reaching the desired size of 10,000 by 1910. However, the population does remain stable from 1920 through 1950. The optimized policy also ensures there is an adequate food supply for deer, predators and other plateau animals.

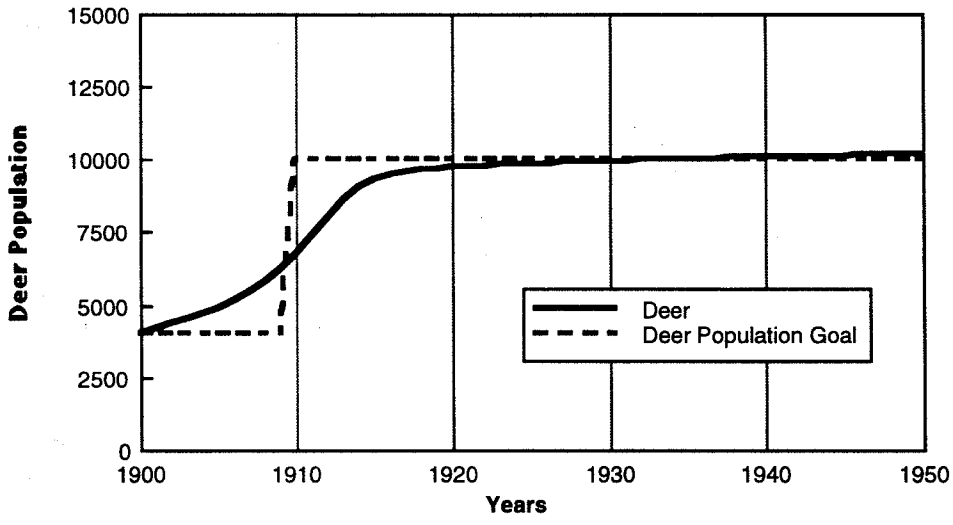

Figure 6 - Optimizing the deer population using constant hunting rates

The simulation results shown in Figure 6 are achieved using the following policies:

An annual deer hunt rate of 0.5% starting in year 1917, and
An annual predator hunt rate of 5% starting in year 1904.

The results from this first test are very encouraging. GAs demonstrate an ability to optimize parameter values given a quantifiable objective function. The second test demonstrates the utility of GAs for minimizing the impact of past policy actions.

Test 2: Improving Current Conditions

The second test provides a more demanding test for the GA. In this test, the GA is used to minimize the long-term impact of errant policy. A baseline policy, which mimics historical practices on the plateau, permits predator hunting at an annual rate of 40% starting in 1906 while no deer hunting is permitted. Assume, however, that policy makers have access to models with optimization techniques in 1920, the beginning of a deer population explosion. Having simulated the likely impact of the current policy, they discover the deer population will rapidly expand and then "crash" as deer starve, ending with a stable population of 20,000 deer. The kaibab land managers wonder if there is an alternative policy which can prevent the impending devastation to the deer population. The new goal is a stable, healthy population of 40,000 deer. The GA is used to find hunting rates which, when applied beginning in 1920, will achieve a sustainable deer population without suffering the consequences of a population boom and crash.

Two policy actions are used to reduce the deer population. First, a new predator hunting rate is applied at the start of 1920. Second, deer hunting is permitted starting in 1920. The annual deer hunting rate is based on the deer population density.

The predator hunting rate is implemented by the GA as a 7 bit binary string with a value equal to (string value / 200). This bounds the annual predator hunting rate between 0% and 63.5%. The deer hunting rate is implemented in the Kaibab model as a table function. The GA is used to determine the best values for each data point in the table function. Input to the deer hunt rate is the deer population density. This value typically ranges between 0 and 0.1 deer per acre. A total of eleven data points are used to define the deer hunt rate table function. Each data point is coded as a five bit string whose value is equal to (string value / 50).

Figure 7 shows the simulation results using the policies determined by the GA. The plot shows the desired deer population of 40,000 head, the simulation results using the baseline policy and the results based on the GA optimization. The deer population reaches its goal within four years after implementing the deer hunting policy and remains stable through 1950.

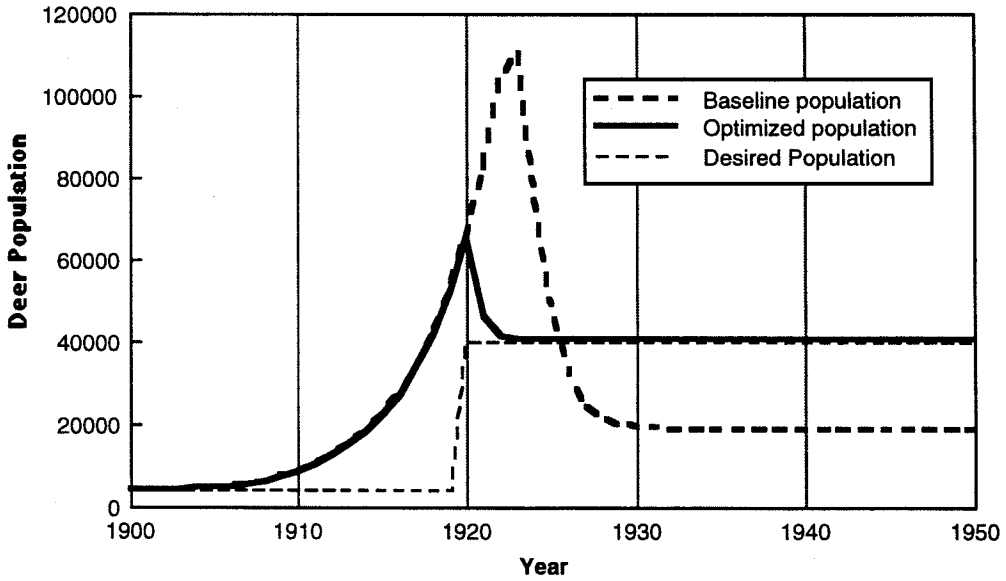

Figure 7 - Optimizing the deer population using GA-derived policies

The GA call for increasing the predator hunt rate from the initial rate of 40% to 45%. Sensitivity tests show that changes in the predator hunt rate only marginally impacts the deer population because the predator population is nearly zero by the year 1920. The reduction in deer population is predominantly attributable to the deer hunting rate. Figure 8 shows the optimized deer hunt rate table used to achieve the reduction in the deer population to a stable 40,000 deer.

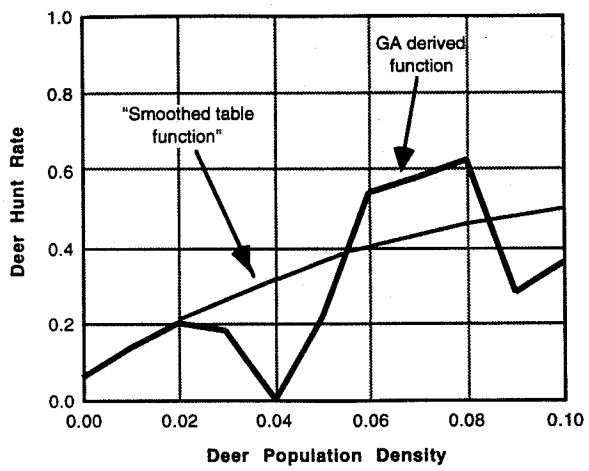

Figure 8 - Deer Hunt Rate Multiplier

The optimized deer hunting rate is not the type of table function most modelers would estimate using traditional estimation techniques [Graham 1980]. To verify that this is, in fact the most effective table, a smoothed version of the table function more typical of the tables used by modelers was also run on the Kaibab model. The results from the smoothed table produce a stable deer population of roughly 24,000 deer; 16,000 fewer than desired. To further verify the need for an irregularly shaped table function, the value of the deer_population_density was checked. Obviously, if the input value remains within a small range, a variety of table functions could generate the same model results. Review of model behavior show the deer population density varies significantly, generating values between 0.04 and 0.09 from 1920 through the end of the simulation. This input range verifies that the irregular, optimized table function is needed to generate the desired results.

Although the model is optimized to support 40,000 head of deer, one wonders how healthy the population is. The Kaibab model uses the ratio of Food_per_deer to measure the relative health of the population. One food unit per deer represents the minimum amount of food needed to prevent starvation. Figure 9 plots the Food_per_deer for the baseline policy versus the optimized policy from 1920 through 1950. The optimized deer hunting policy not only sustains twice as many deer as the baseline behavior, it does so with six times the available food per deer.

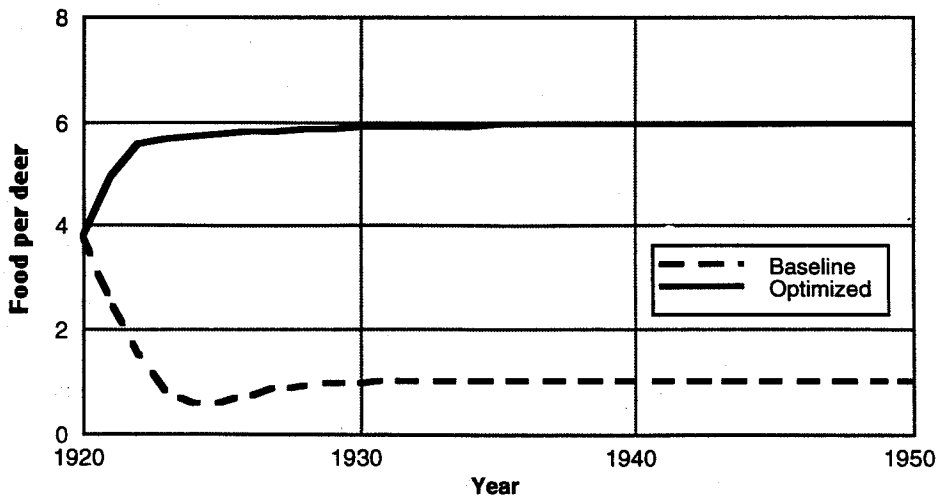

Figure 9 - Comparing the health of the kaibab deer population

Summary of results

The results of the two tests demonstrate the power and utility of GAs as an analytic support tool for system dynamics models. In particular, the results from test 2 show that, just as system dynamics models often produce counter intuitive behavior, the optimal policy actions also defy common experience.

A multitude of interesting and enlightening policy tests can be conducted using the Kaibab model (or any other model for that matter) - far more than can be discussed in a paper of this length. The remainder of this paper discusses a number of applications for GAs, discusses possible reasons for the limited use of GAs within the system dynamics community and looks to the future integration of GA techniques with modeling environments.

3. The future of GAs and System Dynamics

Applying GAs to system dynamics models

Preliminary research shows GAs offer great potential for improving the utility of system dynamics models. Specifically, GAs offer three unique analytic capabilities currently lacking in the system dynamics community.

1. Optimizing a simulation given a set of policy objectives
2. Sub-optimizing a model to narrow the acceptable range of parameter values
3. Estimating likely parameter values for unknown quantities given historical data

1. Optimization

The research presented in this paper focused on the first capability - optimizing parameter values given a policy objective. In this role, GAs demonstrate their versatility and utility by permitting modelers to optimize any number of parameters as easily as optimizing a single parameter. For example, test 1 optimized four key model parameters while test 2 provided twelve parameter values in the form of table functions and model constants. Model optimizations of the type presented in this paper offer great benefits to model users who are not well versed in model operation, but rely on model results for critical decision-making.

2. Sub-optimization

There are times when a full optimization effort is either impractical or not desired. In these instances, GAs still offer utility by providing reasonable initial estimates for model parameter values. For example, in test 1, it would have been possible to run a GA with a small population size for only a few generations in order to estimate likely values for hunt rates and hunting start years. With these estimates in place, a modeler could test the sensitivity of the estimates on

model behavior. For instance, does a predator hunting rate of 5.5% significantly alter the outcome of the simulation results? If not, there may be no need to fully optimize the model. Sub-optimization using GAs provides knowledgeable model users with a faster, more efficient method of testing the sensitivity of parameter values than do traditional methods.

3. Parameter estimation

GAs can be used in conjunction with historical data to estimate unknown parameter values. Confidence is increased in models capable of recreating historical data. Too often, the "soft" parameters used in a system dynamics model are hard to quantify given inaccurate or insufficient data. In these cases, GAs can be used to find reasonable values for unknown parameter values. For example, given a model of an industry, how does one estimate management's delay time in responding to a change in expected orders? Using the recreation of historical data as the objective, a GA can identify the most reasonable value of the delay time. This capability, while powerful, does present a moral dilemma for model developers. Given a loosely bounded objective, GAs can, in theory, optimize any model to any historical data. For example, one could use GAs to fit the Kaibab deer population to historical employment levels for a given company and impress upon users the notion that employment rates can be correlated with food availability on the plateau. This example is admittedly absurd, but it does show how GAs could be used to validate a model as "reasonable" while using an "unreasonable" model structure. If, however, a model has already proven its accuracy, GAs are useful for determining unknown parameter values for a new data set.

Limitations of GAs

GAs almost seem too good to be true. Although simple in theory and application, GAs do possess two serious drawbacks, both of which are technology related. First, GAs are processor intensive. Consider the results from test case 1. Using a population size of 60 with a total of 50 generations requires a total of 3,000 model runs. Fortunately, the Kaibab model runs rapidly on modern desktop computers. Assume, however, one wanted to apply GAs to a more complex model - one requiring thirty seconds to run on the fastest available, affordable PC. 3,000 runs at one-half minute each translates into 25 hours of dedicated computer time.

The second limitation is the need to code the algorithms for decoding each parameter and for generating the fitness for each model one wishes to optimize. The more popular modeling languages, such as iThink, DYNAMO or Vensim, cannot easily support custom software code. Translating models into a language such as C or Pascal and then applying the GA can be time consuming and error-prone.

Fortunately, continuing advances in computer hardware and software are expected to soon overcome these two limitations.

The future of GAs and system dynamics

As recently as five years ago, modelers would never have expected they could easily build, test and run complex simulation models on a computer in their own office. Nobody who ever ran DYNAMO on a mainframe would ever consider conducting thousands of model runs to conduct optimization. Technology has changed all of that. With the advent of ever more powerful desktop computers GAs could become a standard tool of system dynamicists. As a case in point, the optimizations run for this paper were conducted using a laptop personal computer. Hardware is no longer a true limitation on doing effective model optimization. Even the example of 25 hours for a complex model optimization is not unacceptable. Analysts could simply start an optimization Friday night and when they return to work Monday morning, the answer would be waiting. Such a scenario compares favorably with the time and cost associated with an analyst manually conducting a host of model runs, with little certainty of finding the best policy.

Advances in modeling software, combined with improved software inter-operability can eliminate the need to create custom optimization routines. In the future it should be possible to take your favorite model and import it into an optimization program or have the GAs built directly into your simulation software. Tools are now being developed to make the design of policy goals and parameter selection as simple as pointing-and-clicking. Sophisticated GAs (which provide better

results than those presented in this paper) can be coded once and used whenever needed. Software advances employing GAs will transform models from confusing programs understood by only the model developer into useful analysis tools usable by anybody needing answers and insights.[2]

Technology no longer limits the adoption of system dynamics models as practical policy analysis tools. Complex models, once the domain of expensive mainframe computers, can be placed on the desktop of every decision maker. Advanced numerical techniques, such as GAs, promise to make the models usable by a wide range of people who in the past have felt overwhelmed by the complexity of the models. GAs promise to help make system dynamics models an integral part of policy making at any level and to serve as a support tool during the model development process.

References

Coyle, R. Geoffrey. 1985. The use of optimization methods for policy design in a system dynamics model. *System Dynamics Review* 1(1): 81-91.
Goldberg, David E. 1989. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley Publishing Company, Inc.
Goodman, Michael R. 1983. *Study Notes in System Dynamics*. Cambridge, Mass: M.I.T. Press.
Graham, Allen K. 1980. *Parameter Estimation in System Dynamics Modeling.* Elements of the System Dynamics Method. 143-161. Cambridge, Mass: M.I.T. Press.
Peterson, David W. 1980. Statistical Tools for System Dynamics. Elements of the System Dynamics Method. 224-241. Cambridge, Mass: M.I.T. Press.
Riolo, Rick L. 1992. *Survival of the fittest bits*. University of Michigan, Ann Arbor.
Roberts, Nancy, David Andersen, Ralph Deal, Michael Garet, William Shaffer. 1983. *Introduction to Computer Simulation A System Dynamics Modeling Approach.* Addison-Wesley Publishing Company, Inc.
Wallbridge, Charles T. 1989. *Genetic algorithms: what computers can learn from Darwin.* Technology Review 92(1): 46-47.

[2] The author is currently working on a GA interface for system dynamics models. The interface, as designed, permits users to select model parameters for optimization, enter parameter bounds and graphically establish the dynamic behavior they seek to achieve.