Genetic Algorithms as Virtual User Managers

Rasmiprava Singh 1, Snehalata Baede 2 and Sujata Khobragade3

[1] **National Institute of Technology,Raipur CG,India**

[2] **National Institute of Technology,Raipur CG,India**

[3] **National Institute of Technology,Raipur CG,India**

Abstract

A growing issue in genetic algorithm research involves understanding the paths through the solution space that are explored. This work presents a software design architecture that can aid in the explanation process. This architecture encourages more complete user interfaces on the problem domain application, which facilitates the integration of a genetic algorithm, at the same time, taking advantage of substantial code re-use. The steps taken to increase the usability of the problem domain application can aid in the visualization of the evolutionary paths explored by the genetic algorithm. This effort on the designer's part results in improvements in overall accessibility of the problem domain application and the evolutionary process.

Keywords:*Virtual User, visualization, genetic algorithm,domain* .

1. Introduction

Automated software testing products use scripting languages to describe user behaviors [Bei90,Mer98a,Mer98b]. Executing the scripts emulates users interacting with the test application. Variables within the scripts can change virtual user behaviors. The standard parameterization technique generates a brute force testing method for the application in question. Yet there exist problem domains where brute force is not feasible, including : situations where a variable's range of values is too large; domains where many variables need to be parameterized. While discussing these problems with a consultant in the automated software testing industry, it became clear that their testing techniques could benefit from a genetic algorithm (GA). Instead of a brute force approach for testing, the GA could guide the selection of the parameters used by the virtual users. The GA would search for virtual users that maximized error conditions.

GAs could also benefit from taking a virtual user point of view. Since the GA is essentially a generate and test search method, a generated virtual user could test each potential solution. Granted, GAs are applied to many optimization problems where a user interface is ill fitted. But as the popularity of GAs has grown, the domains that GAs are utilized in has grown as well. In these new domains understanding the subtleties of a solution might involve more than just a fitness value and its genetic representation. To understand the ``how" of a solution one might benefit from a visual interface for the problem domain application.

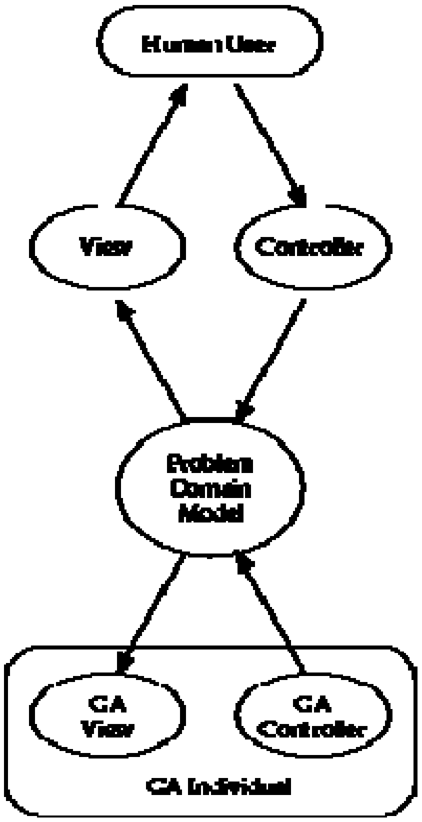
Likewise, the set of users applying GAs is growing more diverse. Communicating the evolutionary processes and advantages becomes more challenging as domains broaden. Often visualization is the most effective aid in fostering an understanding of the underlying phenomena. Recent work [BB98] towards visualizing the evolutionary path greatly aids the understanding of evolutionary processes. This work compliments their approach.

This paper describes an approach where each individual in a genetic algorithm is treated as a virtual user. The GA search involves finding the virtual users that manipulate the problem domain application in the most appropriate fashion. True, this paradigm demands more discipline on behalf of the system architect in the early stages of design. However this effort does not go unrewarded, as it leads to a system that is easier to test, explore, understand, and share. The payoff is not an optimized GA, but instead a greater accessibility to the problem domain and the results generated by the GA.

System architecture

An individual in a genetic algorithm acting as a virtual user does not demand a sophisticated scripting system that understands user interfaces. If one uses the Model-View-Controller (MVC) design pattern [Gol84,BMR6#1+96] in the program representing the problem space, an interface is clearly defined that can be utilized by code designed for either a human or virtual user. Figure 1 lays out the basic genetic algorithm as a virtual user manager (GAVUM) architecture. This mechanism leads to relatively painless implementations of virtual users.

Figure: A graphical representation of the system architecture. The user's application is composed of the Model, View and Controller. The user affects the Model through the Controller and the Model's state is visualized through the View. The GAVUM mechanism integrates a GA with the problem domain application by designing the GA Individual to use the Model's View and Controller interfaces.



In the MVC design pattern, the Model object contains the heart of the problem domain application. It encapsulates all the data and functions underlying the phenomena to model. When something ``interesting" happens in the Model, it updates all the View objects associated with it. ``Interesting" is defined by the problem domain and the View itself. An example of these two objects would be a simple model oven. The Model would be tracking the temperature of the oven: calculating heat flow and transfer. As the temperature in the oven changes, it would notify all of its associated View objects with the temperature in degrees Fahrenheit. One View object could be coded to display the temperature, another View object could re-calculate the temperature into Celsius before displaying it, while another View could display the temperature as a color. A Model can have any number of Views, but each View can only have one Model.

Adding a Controller object to this application would allow us to alter the oven's behavior. The Model object provides a set of methods through which its behavior and/or state can be changed. The Controller object could map keystrokes and mouse actions in the user interface to the appropriate method calls. The user could be presented with a knob labeled with temperatures. As the knob's value changes, through user actions, the Controller object would alter the settings within the Model. Again, the mapping from Controller to Model is many to one. There could also be a text field where the user can type in a desired temperature. Either way, the change in temperature of the oven would affect the model and in turn, be visualized through the various Views.

Once the MVC structure is in place, coding the GA virtual users is straightforward. The process involves these steps:

- Create a Controller object to map a genetic representation to the appropriate method calls in the Model object. Different genomes would encode for different parameters passed to the Model methods.

- Create a View object that can interpret the data from the Model and calculate a fitness value. Depending on the way the Model is implemented, fitness values can be derived after one update or after many updates.

- Glue the Controller and View together in a GA Individual object. This interfaces the GA with the problem application by defining exactly how a fitness value is generated from a genetic representation.

There is no runtime penalty for visualization for the virtual users, as their data does not need to be displayed. Mutation and crossover operations can act upon the genome in one's preferred fashion. Similarly, one's favorite selection methods can then drive the GA, to evolve the virtual users.

This architecture doesn't define any changes to the actual evolutionary computational method. Instead, it improves on the interface among the evolutionary computation, the problem domain application, and the researcher. It also fosters reusability at the level of the GA and the problem domain application. GA Individuals seen as virtual users can control a variety of application types, all being evolved by an unmodified GA. The Model object, which defines the majority of the domain application, is used by both human and virtual users. With one code base, changes to the application are less complicated and consistency is maintained without any effort.

1. Case studies

As mentioned earlier, the number of domains that utilize GAs is growing. The domain of multi-agent systems is an example of a new area benefitting from GAs. Multi-agent systems are difficult to design and analyze because the group's behavior often relies on subtle traits in the local behavior of the individuals. GAs allow designers to search through the space of individual behaviors for desired group behaviors. This work presents three different examples of GAs applied to multi-agent systems using the GAVUM architecture.

1.1 Resource Allocation

Sen et. al. [SRA96] suggest that limited knowledge in multi-agent systems can be beneficial in certain cases. Their work involves the resource allocation problem, where N autonomous agents must distribute themselves among M resources linked in a circular chain . During a time step each agent chooses a resource to use. The performance of the system can be measured by the number of time steps it takes for the agents to converge upon and maintain the optimal state starting from random initial distributions.

In Sen's implementation, each agent's behavior is controlled by identical probabilistic functions. f_{ii} determines the probability of an agent staying at the current resource and f_{ij} defines the probability of moving from resource *i* to resource *j*. These are the control functions:

$$f_{ii} = \frac{1}{1 + \tau \exp \frac{r_i - \alpha}{\beta}} \qquad f_{ij} = \frac{1}{1 + \tau \exp \frac{r_i - r_j - \alpha}{\beta}}$$

Here, r_i represents the current number of agents at resource *i*; α, β, and τ are control parameters. Sen introduces a window parameter, which limits the number of resources an agent has knowledge of at any time. This window value defines how many neighboring resources the agent has access to. The window is centered around the agent, so a window value of 3 would allow agents access to the number of agents at the adjacent resources. The window value also defines legal values of *j* for each *i* in the control functions. With Sen's control parameters, larger windows led to slower convergence to the optimal state, and smaller windows led to faster convergence. To explain their results, Sen presents the number of agents at one resource over time. As the number of sites visible to the agents increases, the number of agents at the resource varies longer.

This counter intuitive phenomena brought forth an attempt to reproduce the results [Bar98a]. First, the Model object was created to represent the resources and the agents. Next, the Controller was designed, creating a user interface through which the control parameters and window value could be set. A View of the resources displays the the number of agents at each resource and the current time step. These components together create the problem domain application. With this program, users can alter the variables and explore the effects the control parameters and window values have on group behavior.

It would take one user too long to explore the space of potential control parameters over all the window values. Thus, the GA was introduced, to emulate the actions of many users. The GA's task was to find the virtual user that would create the fastest settling group of agents. The mutation and crossover operations alter the genetic representation which changes the α, β, and τ values set by the GA Controller. The GA View object would ignore the updates until the optimal state was reached. Once the optimal state was reached, the current time step was used as a fitness value. In some cases, the optimal state was never maintained. For those cases, the simulation was halted after 30,000 time steps. To gather more accurate statistics, several iterations of the model needed to be run. Additional logic was added to the GA Individuals, enabling them to repeat their experiments in the model.

Running the GA found a virtual user with a set of control parameters that provided superior performance regardless of the window values. The GA results were immediately examined by plugging the best individual's parameters into the problem domain application. The agents quickly settled into the optimal state as expected.

By examining the time series of fitness values, an increase in fitness was apparent. Yet this reveals very little in this domain. But with this system architecture, one can pick out the best individual every 10 time steps and plug the virtual user's values into the stand-alone application. This ends up being much more informative, as the group behavior starts out being very volatile in the early generations. It becomes clear that in these volatile agent situations, the window size has little to do with group behavior. The group performs equally poorly at all window sizes. As the GA individuals evolve, one can observe agent volatility decrease. First for only some window values, then for all the window values. This information gives many hints about the nature of the problem space, allowing the user to generate more hypotheses about the behavior of the system. This example illustrates the GAVUM architecture's ability to enhance the knowledge acquisition process.

1.2 Game Playing Artificial Intelligence

Designing artificial intelligence routines for games is another application of the GAVUM architecture. Though the AI in some of today's commercial games might use more information than what is provided to the user, that hasn't always been the case [Sam63]. Additionally, in cognitive science research, AI models are designed with the human player's abilities and techniques in mind [KM92]. These facts support using virtual users to evolve game playing AIs.

A simple client/server game of Snake was written as a separate project to learn about networking. Snake is a simple game, where a user controls a snake's movement in a 2-d world. A snake increases in size when food is eaten, and it dies when the snake's head comes in contact with any non-food object in the world. Users interested in playing the game launch a snake View in order to see the world. Launching a snake Controller enables the user to control a snake in the game. Both client programs connect to the snake world Model object, which doubles as the server. With the client-server architecture, multiple snakes can be introduced into the game.

Using the GAVUM architecture, a GA individual then can use a snake View (to understand the world and its place in it) and a snake Controller to control its own snake. The GA individual becomes a real-time, interactive player in a networked game of snake. The snake game reports statistics about players, like the longest length attained, number of lives, etc. The GA individual then can use this data to calculate its fitness. Once its fitness is calculated (longest length achieved during 10 games), it quits the snake game. The selection process of the GA decides its fate.

This example shows displays how the GAVUM architecture can get co-evolution for free. The networked architecture of the game enables co-evolution in the GA, as the individuals are competing against other snakes controlled by other GA Individuals. Additionally, the GAVUM system even allows visualization of the co-evolutionary paths. The user can open up their own snake View of the game, in order to watch the GAVUM snakes play and evolve. The user can also join the game, affecting the fitness of the individuals in the GA. Many benefits are reaped simply by using a different program structure. This architecture could be directly applied to the iterated prisoner's dilemma domain as well as its related tournament models.

1.3 Artificial Life Models

My research focuses on artificial life (AL) models. These models rely on visualization to express their behavior. Again, through the model's API designed for use with the Views and Controllers, the GA individual has control over the Model, without having to compute the display. My models are designed to facilitate examining the evolution of coordination in multi-agent systems [Bar98b]. In my model, survival is the task facing homogenous populations of agents. The model's agent environment has many parameters that can be adjusted, offering a variety of ways to increase its hostility. The behavior of the agents themselves can be adjusted via the Controller. The agent's behavior is what the GA virtual users adapt.

Each GA run evolves agents best fit for a given environment. In these experiments, GA individuals are created that define new environmental conditions in which to evolve agents. The researcher now takes the position of defining meta-experiments, with the virtual users performing sets of experiments. This allows one to study the environmental effects on the agents and their evolution. Each GA run creates an evolutionary path that represents a set of experiments, each resulting in agents performing to varying degrees of success. And then collectively, the GA runs define another set of evolutionary paths across meta-experiments. Hopefully, the automation of the experiments that is intrinsic in the GAVUM architecture will increase the rigor of AL research. As I extend the model to allow heterogeneous populations, the same GA will be used, with some modifications to the GA Individual object. Again, co-evolution will be a convenient side effect.

This AL domain highlights another benefit of this paradigm. With the other models, reproducibility was not as much of an issue, because the code was simpler and there were fewer variables in play. But with artificial life models, programmers are only limited by their imagination and the programs quickly get rather complex. Reproducibility has been a problem facing the AL community from the beginning. By encouraging the MVC and virtual user system, programmers might find it worthwhile to introduce useable Views and Controllers. Then the problem domain applications can stand-alone and be shared with other researchers. Instead of trying to duplicate the code, others can use the same code to repeat the experiments and run their own new ones.

2. Discussion

All too often, researchers are too excited to start evolving individuals in a GA, and not enough effort is put into figuring out how to explain the results. The GAVUM architecture provides a step towards insuring that the most can be learned from the evolutionary computation. Diligent application of the GAVUM paradigm will also allow others to use the problem domain application and to explore the problem domain themselves. With the Java programming language and applets that can be run through web browsers, accessibility of the problem domain application can become trivial. URLs of all work discussed here is soon to come.

The GAVUM architecture is currently being discussed with instructors of an artificial intelligence class as a way of introducing the class to GAs. First, students will be introduced to modelling through a particular problem domain application. Through hands on experience with the application, the students gain a deeper understanding of the domain problems that the model represents. Then, as they become familiar with the program, they would be asked to create their own metric for comparing two instances of their models with each other. In essence, each student is implicitly creating a fitness function. Next, the students would be introduced to a method of finding individuals that maximize performance according to a metric - the GA. They can then code up their own GA individual.

The students would already be familiar with the kinds of information expected by the Controller, as they have had to supply the same information to the program when they were using it. Through creating their own metric for comparison, they have used information from the View. The students just have to formalize this information in actual code. This clearly defines assignments in the sense of coding routines with well documented input and output. Additionally, these methods probably would not be too technically demanding, which can be a benefit in introductory classes. But the elements of the model that are tested, how the individuals are evaluated, and the genetic representations are completely open ended. Each student has the freedom to explore the areas of the model that are personally interesting.

After running the experiments, (most likely editing their GA individuals along the way), the students can then discuss the various advantages and disadvantages of their different representations, fitness functions, and evolutionary techniques (mutation and crossover operators and rates, plus the selection methods). Throughout the assignments, the process reinforces the benefits of the scientific method.

Hopefully the advantages of the GAVUM architecture are apparent and seem worth the design and coding effort. *Programming as if people mattered* [Bor91] is a honest, practical book that stresses the need to bridge the gap between programmers and users. The same need is found in the GA community. Although GAs came from a programmer-rich environment, one can no longer assume that users of GAs will only be other programmers, due to the growing diversity in the GA user population. Both the GA and the problem domain application should be considered tools to be used by others for learning - making visualization all the more important.

Currently, work is being conducted to integrate the GA into the actual problem domain application. User interface issues need to be refined in order to merge the View and Controller of both the GA and the problem domain application. Future work is directed towards better formalizations of the model. There are most likely better ways to generalize the API in order to create more re-useable code. Finally, applying this approach to other domains would help evaluate the effectiveness of the GAVUM architecture.

2.1.1 Acknowledgements

I am grateful to Catherine Kim at Mercury Interactive for numerous discussions about automated software testing, virtual users, and their potential relationships with genetic algorithms. This research is supported by the National Science Foundation under grants GER93-54898 and CDA93-03189.

References

[1]. Baray.Effects of individual decision schemes on group behavior.
In *The Third International Conference on Multi-Agent Systems*, New York, NY, 1998. IEEE Press.

[2].Baray.
Effects of population size upon emergent group behavior.
In *Complex Systems '98 Conference Proceedings*, 1998.

[3].M.A. Bedau and C.T. Brown.
Visualizing evolutionary activity of genotypes.
Technical Report 98-03-23, The Santa Fe Institute, Santa Fe, NM, 1998.

[4].Beizer.
Software Testing Techniques.
International Thomson Publishing, Boston, MA, 1990.

[5].Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal.
Pattern Oriented Software Architecture: A System of Patterns.
John Wiley and Sons, New York, NY, 1996.

[6]. N.Borenstein.
Programming as if people mattered.
Princeton University Press, Princeton, NJ, 1991.

[7].Adele Goldberg.
Smalltalk-80: the interactive programming environment.
Addison-Wesley, Reading, MA, 1984.

[8].D.Kirsh and P. Maglio.
Reaction and reflection in tetris.
In J. Hendler, editor, *Artificial intelligence planning systems: Proceedings of the First Annual International Conference (AIPS92)*, San Mateo, CA, 1992. Morgan Kaufman.

[9].Mercury Interactive.
Load Runner : Controller Users Guide.
Mercury Interactive Press, 1998.
[10].Mercury Interactive.
Load Runner : Creating Vuser Scripts.
Mercury Interactive Press, 1998.

[11].A.L.Samuel.
Some studies in machine learning using the game of checkers.
In E.A. Feigenbaum and J. Feldman, editors, *Computers and Thought*. McGraw-Hill, New York, NY, 1963.

[12].S. Sen, S. Roychowdhury, and N. Arora.
Effects of local information on group behavior.
In Mario Tokoro, editor, *The Second International Conference on Multi-Agent Systems*, Menlo Park, CA, 1996. AAAI Press.

First Author: Msc (computer Science) On 2006, M.Tech (Computer Technology) On 2010,National Institute of Technology, Raipur, CG Two international paper, Generally interest in wireless, image processing and genetic algorithm.

Second Author Msc (Mathematics), MCA (Computer Application), National Institute of Technology, Raipur, CG Two international paper, Generally interest in wireless, image processing and genetic algorithm.

Third Author BSC (Electronics), MCA (Computer Application),National Institute of Technology, Raipur, CG Two international paper, Generally interest in wireless, image processing and genetic algorithm.