GATutor: A Graphical Tutorial System for Genetic Algorithms*

Charles Prince
Roger L. Wainwright
Dale A. Schoenefeld
Travis Tull

Department of Mathematical and Computer Sciences
The University of Tulsa
{prince,rogerw,dschoen,ttull}@euler.mcs.utulsa.edu

ABSTRACT

In this paper we discuss the design and implementation of GATutor, a graphical tutorial system for genetic algorithms (GA). The X Window/Motif system provides powerful tools for the development of a user interface with a familiar feel and look. We implemented the Traveling Salesman Problem (TSP) and the Set Covering Problem (SCP) as two example GA problems in the tutorial. The TSP problem uses an order-based chromosome representation (permutation of *n* objects), while the SCP uses bit strings. The user has numerous buttons to select the GA parameters. These include (a) type of initial population: random or from a file, (b) mode: steady-state or generational, (c) population size, (d) maximum number of generations or trials, (e) generation gap, (f) selection mode, (g) selection bias, (h) selection of the crossover operation from a choice of several possibilities, (i) mutation method, (j) mutation rate, (k) replacement method, (l), elitism, etc. The user has the ability to do a step by step execution or to do a continuous run. The screen layout provides visual representation of the chromosomes in the population with the ability to scroll. This gives the user the option of varying one or two GA parameters to visually see the effect on the algorithm. One of most important features of this tutorial is the set of help screens that explain, with examples, all of the options for each of the GA parameters. This package has already been very useful for teaching the fundamental features of GAs in many different courses, and it has been very valuable in our GA research projects.

* Research supported by OCAST Grants AR2-004 and ARO-038 and Sun Microsystems Inc.

1. GENETIC ALGORITHMS

Genetic algorithms search through the solution space by emulating biological selection and reproduction. Each new generation is created by a biased reproduction, the more "fit" members of the population have a better chance of reproduction. The parameters of the model to be optimized are encoded into a finite length string, usually a string of bits or a string of integers. Each parameter is represented by a portion of the string. The string is called a chromosome, and each bit (integer) is called a gene. Each chromosome is given a measure of "goodness" by the fitness function. The fitness of a chromosome determines its ability to survive and reproduce offspring. The "least fit" or weakest chromosomes of the population are displaced by more fit chromosomes. The fitness function drives the population toward better solutions.

Genetic algorithms use probabilistic rules to evolve a population from one generation to the next. The transition rules from one generation to the next are called genetic recombination operators. These include *reproduction*, selection of the more "fit" chromosomes; *crossover*, where the genetic material of two chromosomes are exchanged in some manner; and *mutation*, where, infrequently a random gene in a chromosome is altered. Genetic algorithms retain information from one generation to the next. This information is used to prune the search space and generate plausible solutions within the specified constraints.

Genetic Algorithms have been applied to a wide variety of problems. These include facility scheduling, production and job scheduling, circuit board layout design, communication network design, keyboard layout, missile control, missile evasion, gas pipeline control, pole balancing, machine meaning, designing neural networks, classifier systems, robotics, signal processing and filter design,

image processing, and combinatorial optimization such as TSP, bin packing, set covering, graph bisection, routing, package placement. For the interested reader, Davis, Goldberg, and Rawling provide an excellent in depth study of genetic algorithms [2,4,9].

2. GATutor MOTIVATION

Studying and developing genetic algorithms for solving combinatorial optimization problems is one of the major research areas in our department. As a result, we developed our own genetic algorithm package: LibGA [1]. LibGA provides a user-friendly workbench for order-based genetic algorithm research. LibGA is a collection of routines written in the C programming language on a Unix platform. It includes a rich set of genetic operators for reproduction, crossover and mutation. Routines are provided which implement both generational and steady-state genetic algorithms using the genetic operators, allowing researchers to compare the two approaches. Other features of LibGA include elitism, generation gap, and the ability to implement a dynamic generation gap. Other routines are provided for initialization, reading the configuration file and generating reports.

In addition we also developed a parallel genetic algorithm package, HYPERGEN [5]. HYPERGEN is a distributed genetic algorithm developed for a hypercube. HYPERGEN distributes the initial population evenly among the processors. Each processor (island) executes a sequential GA on its subpopulation, performing crossover and mutation. HYPERGEN is a modular collection of routines for generating the initial population, and designating the evaluation function. HYPERGEN also provides routines for selection (based on a bias function), reproduction, mutation, migration interval, migration rate and summary statistics. The sequential GA on each processor and the periodic migration of genetic material between processors is performed automatically for the user.

In several of our undergraduate courses such as Data Structures, Analysis of Algorithms, and Discrete Mathematics, the topic of genetic algorithms is presented to some extent. We believe techniques such as genetic algorithms and simulated annealing for solving combinatorial optimization problems deserve attention, especially in undergraduate courses just as much as more traditional algorithmic techniques such as dynamic programming, branch and bound and backtracking. In the undergraduate courses, GAs are presented in just one or two lectures, just to give an introduction. In graduate courses, the topic may be presented over several lectures and covered in more detail. The authors attempted to locate a good source for an overview for GAs, but could not locate anything. We found ourselves trying to generate all of the introductory material (in text form) from numerous sources. This proved to be very time consuming and difficult. It was also not a very effective learning tool for the students.

Other educators have had recent success in developing graphical tutorials for various disciplines. Lim and Hunter [7] developed a graphical database design tool, DBTool, for an introductory database course. Schweitzer [11] developed interactive visualization tools for a computer graphics course. Kurtz *et al.* [6] developed a tutoring system for denotational semantics. Newsome and Pancake [8] developed a graphical computer simulator for systems programming courses, and Fritz [3] developed hypercard applications for teaching information systems. Therefore, we elected to develop a graphical tutorial system for genetic algorithms, GATutor. The authors are not aware of any other graphical genetic algorithm tutorial package.

GATutor was designed to assist in the instruction of the foundations and principles of genetic algorithms. Specifically, GATutor is: (1) Designed to be used by anyone after a typical introduction lecture on genetic algorithms in some course. Note the lecture may not necessarily include any instruction on GATutor at all, just fundamental terms, concepts and definitions. (2) Designed for a general audience, not just computer science students. Thus, no programming is required and very few computer skills are required by the user. (3) Portable and easy to install. GATutor was developed in C under X Windows/Motif. (4) Modular enough to easily be able to add new features and options to the package by different people over several years. (5) Comprehensive, with a full set of help menus to explain, with examples, all of the numerous GA terms, definitions, operators and parameters. (6) A self contained package where students can visually see the effect of executing a genetic algorithm. (7) An environment where students can be creative and investigate what happens when different GA parameters are tried in various combinations. This "research" type of setting is extremely helpful in developing early research skills in undergraduates.

GATutor can be used for several different purposes. The primary purpose for GATutor is to educate our students in the area of genetic algorithms. GATutor helps develop the skill and understanding of genetic algorithms for those undergraduate and graduate students interested in either reading more about GAs or developing some insight into their behavior. This package is also very helpful for our seniors and first year graduate students who are interested in doing research in genetic algorithms. This package is also very helpful for students in other disciplines who are applying GAs in their research work. In our university, students in electrical engineering, petroleum engineering, geosciences and business are currently using GAs in ongoing research projects. Added to HYPERGEN and LibGA, GATutor completes our trio of in-house developed GA software packages for education and research in GAs.

A secondary, but very important, use for GATutor is to educate individuals who are not typical university students. For example, each summer the university conducts summer academies for middle school and high school students. The students are divided into small groups of three or four students and assigned a simple research project. When the authors were involved in this program, our students used a GA package to work on the traveling salesman problem. The students also prepared a short written and oral report. Unfortunately GATutor was not available at that time, but will prove very useful for future summer academies. The authors are also involved with summer institutes for mathematics and science high school teachers. For the past two summers we conducted a discrete mathematics institute for mathematics teachers of grades 7-12. We devoted one lecture to GAs in an attempt to show new techniques for solving problems. They were absolutely fascinated by the topic. GATutor was not finished in time for our last institute, but undoubtedly will prove very "entertaining" in future teacher institutes. This also opens the door for invitations to high school classrooms to talk to students about computer science and mathematics. One final and important use for GATutor is to demonstrate GA techniques to visiting students from local middle and high schools touring our university and department. This is an excellent demonstration package. In short, the summer academies for students, summer institutes for mathematics and science teachers, along with the visitations from local schools are extremely effective recruiting tools for our future students.

3. AN OVERVIEW OF GATutor

3.1 TSP AND SET COVERING PROBLEMS

Given a set of n points in a plane corresponding to the location of n cities, find the minimum distance closed path that visits each city exactly once. This is called the Traveling Salesman Problem (TSP). In GATutor it is assumed that each city is directly connected to every other city by Euclidian distance. In the Traveling Salesman Problem a chromosome is simply a permutation of the numbers 1..n representing the sequence in which the cities are visited. Problems that can be represented in a chromosome as a permutation of n integers are called order based problems.

The Set Covering Problem (SCP) problem is the problem of finding the minimum number of columns in a Boolean matrix such that all rows of the Boolean matrix are "covered" by at least one element from some column, and the sum of the costs associated with the covering columns is optimal (minimum cost in our case). A Boolean matrix is a rectangular matrix of zeros and ones. A row is covered if it contains a one in any of the selected columns. For a matrix with n columns, the number of combinations to try is 2^n, since each element of the power set is a possible solution.

Consider Figure 1 which shows an example Boolean matrix with $m = 7$ rows and $n = 8$ columns. Columns 2, 3, 4 and 5 form a cover with a cost of 19; columns 1, 2, 5 and 8 also form a covering with a cost of 23. The optimal cost is 10 formed by columns 4, 5, and 7.

Given a SCP problem with m rows and n columns, a chromosome defining a possible covering is depicted as a bit string of length n. The ith bit of the chromosome corresponds to the ith column of the Boolean matrix representing the problem. A one bit means the column is included in the covering, and a zero bit means the column is not included in the covering. There are obviously 2^n possible chromosomes. A feasible solution is defined as any covering of the rows of the Boolean matrix. For example, in Figure 1, the bit strings (01111000), (11001001), and (11111111) are feasible solutions. An infeasible solution is represented by a bit string that does not define a covering, such as (00100100), (11010100), (10101011), and (00000000). Hence, the optimal solution is a feasible solution with the minimum cost. Further discussion of GAs applied to the SCP problem can be found in [10].

GATutor features genetic algorithm implementations of these two classic problems, TSP and SCP. These problems are easy to understand and represent the two most common chromosome representations of problems, permutation of integers and bit strings.

3.2 USER INTERFACE

The initial screen and main window for GATutor gives the user all of the options required to execute a genetic algorithm (see Figure 2). The screen is divided into four parts, **Problem Parameters, GA Parameters, GATutor Commands** and two **Scroll Windows** at the bottom of the screen. In the **Problem Parameters**, the user may select the *Problem*: TSP or SCP, the *Size* of the problem, and the type of *Data*. In this case, the user has selected a 10 city TSP problem with user-built data. When user-built data is selected, another screen appears where a point and click method is used by the user to place points anywhere in the screen. In this case the final number of points specified by the user becomes the *Size* of the problem.

There are numerous **GA parameters** that the user may specify. All of these options have default values. In Figure 2, this user has specified that the *Initial Population* is to be generated randomly. The *Mode* is generational. The other option is steady-state. The *Pool Size* is set at 20, and the *Number of Generations* is set at -1. A positive value will cause the GA to terminate after that number of generations (or trials for steady-state mode). A negative value means the GA will continue to run forever. In this case, the tutorial will be terminated by using one of the **GATutor Commands**. In the example shown in Figure 2, the

Generation Gap is not used. The *Selection Method* is roulette. Other selection options are uniform random and rank biased. The *Selection Bias* is 1.8. The range is from 1.0 to 2.0. The *Crossover Method* is Order1. All of the options are user selectable from menus. In addition, all options for all parameters have help screens that serve as a complete tutorial system. In Figure 2 the *Crossover Rate* is set to 1.0. The *Mutation Method* is swap. Other mutation options include simple inversion and simple random. The *Mutation Rate* is set to zero (off) in this case, and the population pool *Replacement Method* is append. Other replacement options include first weaker, by rank and weakest. Finally, *Elitism* is on and the *Random Number Generator Seed* is 1.

The **GATutor Commands** allow the user to either *Step* through the GA one generation (or trial) at a time, *Run* continuously, *Reset* the problem, or *Quit* the tutorial. Two graphical scroll bars are provided at the bottom of the screen depicting the population pool at any given moment. This is shown as a permutation list (TSP) or list of bit strings (SCP) in the left window, and a corresponding graphical display in the right window for each chromosome. The right window illustrates the current population in "row major order". Both windows scroll vertically and horizontally for a complete view of the entire population.

GATutor is a viable tool in terms of its functionality and availability to aid students in learning about genetic algorithms in many different courses at various levels. There are numerous uses for the package. We have found that GATutor is very easy to use; students enjoy using it, and they seem to learn extensively on their own. Several additions and options are planned for future enhancements of GATutor. For example, the architecture of GATutor will make it convenient to provide visualizations for problems other than the TSP and SCP problems. Thus, in addition to being a useful educational tool, GATutor is also an invaluable research tool. Finally, GATutor is portable and easy to install and is available at no cost by sending email to rogerw@penguin.mcs.utulsa.edu.

ACKNOWLEDGEMENTS

This research has been supported by OCAST Grants AR2-004 and ARO-038. The authors wish to acknowledge the support of Sun Microsystems Inc.

REFERENCES

[1] A.L. Corcoran and R.L. Wainwright, "LibGA: A User-friendly Workbench for Order-based Genetic Algorithm Research", *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing*, pp. 111-118, 1993, ACM Press.

[2] L. Davis, ed., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

[3] J.M. Fritz, "Hypercard Applications for Teaching Information Systems", *Proceedings of the Twenty-Second Technical Symposium on Computer Science Education, SIGCSE Bulletin* Volume 23, Number 1, March, 1991, pp. 55-61.

[4] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning,* Addison-Wesley, 1989.

[5] L.R. Knight and R.L. Wainwright, "HYPERGEN: A Distributed Genetic Algorithm on a Hypercube", *Proceedings of the 1992 Scalable High Performance Computing Conference, SHPCC'92,* Williamsburg, Va., April 26-29, 1992.

[6] B.L. Kurtz, R.L. Oliver and E.M. Collins, "The Design, Implementation, and Use of DSTutor: A Tutorial System for Denotational Semantics", *Proceedings of the Twenty-Second Technical Symposium on Computer Science Education, SIGCSE Bulletin* Volume 23, Number 1, March, 1991, pp. 169-177.

[7] B.L. Lim and R. Hunter, "DBTool: A Graphical Database Design Tool for an Introductory Database Course", *Proceedings of the Twenty-Third Technical Symposium on Computer Science Education, SIGCSE Bulletin* Volume 24, Number 1, March, 1992, pp. 24-27.

[8] M. Newsome and C.M. Pancake, "A Graphical Computer Simulator for Systems Programming Courses", *Proceedings of the Twenty-Third Technical Symposium on Computer Science Education, SIGCSE Bulletin* Volume 24, Number 1, March, 1992, pp. 157-162.

[9] G. Rawling, ed., *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, 1991.

[10] D.A. Sekharan and R.L. Wainwright, "Manipulating Subpopulations of Feasible and Infeasible Solutions in Genetic Algorithms", *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing*, pp. 118-125, 1993, ACM Press.

[11] D. Schweitzer, "Designing Interactive Visualization Tools for the Graphics Classroom", *Proceedings of the Twenty-Third Technical Symposium on Computer Science Education, SIGCSE Bulletin* Volume 24, Number 1, March, 1992, pp. 299-303.

Rows	Columns							
	1	2	3	4	5	6	7	8
1	1	0	1	1	0	1	0	0
2	0	1	1	0	0	1	1	0
3	0	0	0	1	1	0	1	1
4	1	1	1	0	1	0	0	1
5	0	0	0	1	1	0	1	0
6	0	0	1	0	1	0	0	1
7	0	1	0	1	0	1	0	0
Cost	6	4	7	3	5	6	2	8

Figure 1: Example SCP Representation as a Boolean Matrix

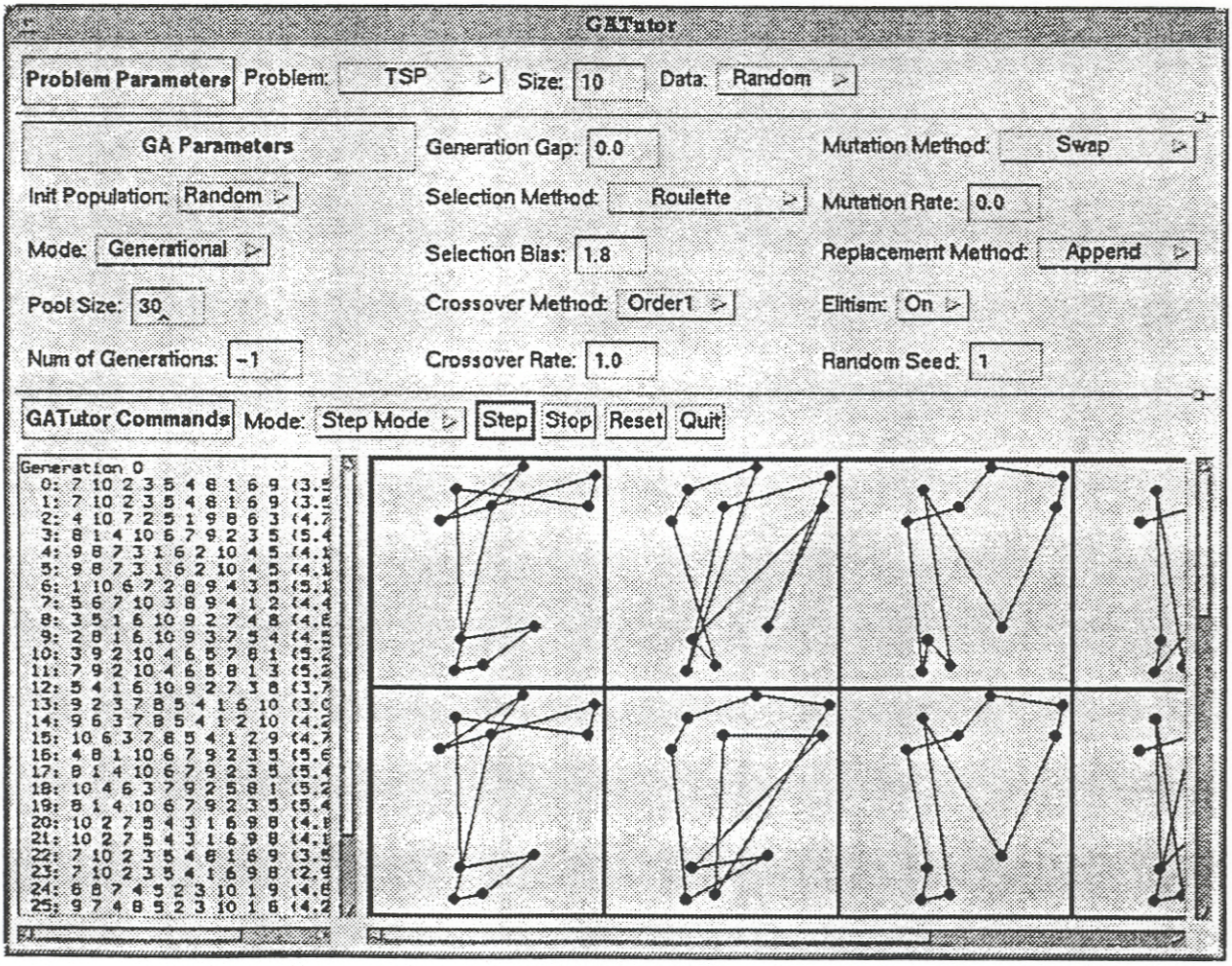

Figure 2: GATutor Primary Window