N91-24050

# GENETIC ALGORITHMS

Lui Wang
Software Technology Branch
Lyndon B. Johnson Space Center
National Aeronautics and Space Administration
Houston, TX 77058
wang@mpad.span.nasa.gov

Steven E. Bayer
Engineering Technology Group
The MITRE Corporation
1120 NASA Road One, Suite 600
Houston, TX 77058
bayer@mpad.span.nasa.gov

## ABSTRACT

Genetic algorithms are highly parallel, mathematical, adaptive search procedures (i.e., problem-solving methods) based loosely on the processes of natural genetics and Darwinian survival of the fittest. This paper introduces basic genetic algorithm concepts, discusses genetic algorithm applications, and presents results from a project to develop a software tool that will enable the widespread use of genetic algorithm technology.

## INTRODUCTION

### Background

Genetic algorithms (GAs) were pioneered by John Holland in his research on adaptation in natural and artificial systems (1). This research outlined a logical theory of adaptive systems. In essence, biological adaptive systems strive to optimize single individuals or entire species for specific environments to increase the chance of survival. Holland simulated the methods used when biological systems adapt to their environment in computer software models–the genetic algorithms–to solve optimization and machine learning problems. The following paragraphs briefly discuss two types of adaptation strategies which are observed in many biological systems and inspired the basic framework of genetic algorithms.

Adaptation. One form of adaptation pertains to the way an individual changes within its environment to promote survival. Examples include the development of antibodies specific to certain diseases, or the enlargement of muscles needed for daily activities. The way we learn, and the neural changes that accompany learning, is another example of how an individual adapts within its environment. The effects of this form of adaptation are not imprinted on the genome (the genetic makeup of a species); that is, they are not passed on from generation to generation. On the other hand, *individual* adaptation does promote the survival of the individual within an environment–*survival of the fittest*–and enhances that individual's net reproductive advantage through a *natural selection* where *fitter* members of a population are more likely to reproduce.

All species have used adaptive search for millions of years, through an evolutionary search process, to improve the way a species lives and survives within its environment. Therefore, adaptation also refers to evolution and modification of an entire species to fit its environment. This is the process of making a species environmentally *fit*. An appropriate example can be seen in the way many plant species have evolved their flower to resemble a female bee or wasp that attracts the male counterpart and promotes pollination. This evolutionary or *species* adaptation is imprinted on the genome and is passed on to subsequent generations.

Thus natural, biological systems continuously use adaptive search to improve genomes–that is, to improve the species–and to promote the survival of *fitter* individuals and genomes through natural selection.

Genetic Algorithms. Genetic algorithms are highly parallel, mathematical, adaptive search procedures (i.e., problem-solving methods) based loosely on the processes of natural genetics and Darwinian survival of the fittest. These algorithms apply genetically-inspired operators to populations of potential solutions in an iterative fashion, creating new populations while searching for an optimal (or near-optimal) solution to the problem at hand. Population is a key word here: the fact that many points in the space are searched in parallel sets genetic algorithms apart from other search operators. Another important characteristic of genetic algorithms is the fact that they are very effective when searching (e.g., optimizing) function spaces that are not smooth or combine several measurements from different types of sensors and maintain a desired state of this non-linear system. The concept can easily be expanded to detect potential component failures and generate immediate advisory messages for corrective actions. Suitability of currently available fuzzy hardware for real-time monitoring and diagnosis is also being investigated.

## SUMMARY

Applications of fuzzy logic in autonomous orbital operations are described in this paper with past accomplishments at JSC. Current ongoing as well as future activities planned are also described. The main objective of all these activities is to **increase autonomy** in orbital operations and thus achieve a higher level of **operational efficiency** desired for future space operations. The approach is to develop modular control that can be upscaled for greater autonomy in an integrated environment. The initial step is to develop a software controller and then to integrate it with hardware at the appropriate level. As the activities progress, detail testing is performed to check out implementation and integration of components. Our preliminary results promise a very successful utilization of fuzzy logic in autonomous orbital operations.

## REFERENCES

1. Zadeh, L. : "Fuzzy Sets", Information and Control, vol. 8, pp. 338-353, 1965.
2. Klir G. J. ; and Folger T. A. : Fuzzy sets, Uncertainty, and Information, Prentice Hall, New Jersey, 1988.
3. Lea, R. N. ; and Giarratano, J. : An Expert System Program Using Fuzzy Logic For Shuttle Rendezvous Sensor Control, proceedings of ROBEXS'86, pp. 327-329, 1986.
4. Lea, R. N. ; and Jani, Y. : Spacecraft Attitude Control System Based on Fuzzy Logic Principles, proceedings of ROBEXS'89, 1989.
5. Lea, R. N. ; Togai, M. ; Teichrow, J. ; and Jani, Y. : Fuzzy Logic Approach to Combined Translational and Rotational Control of a Spacecraft in Proximity of the Space Station, Proceedings of the IFSA'89, pp. 23-29 1989.
6. Rogers, M. ; and Hoshiai, Y. : The Future Looks 'Fuzzy', NEWSWEEK, May 28, 1990.
7. Johnson, R. C. : Clear Leader Emerges : Japan at fuzzy fore, EETimes, Sept. 11, 1989.
8. Armstrong, L. ; and Gross, N. : Why 'Fuzzy Logic' beats black-or-white thinking, Science & Technology section, BUSINESS WEEK, May 21, 1990.
9. Yasunobu, S. ; and Miyamoto, S. : "Automatic Train Operation System by Predictive Control", Industrial Applications of Fuzzy Control, Sugeno, M. (Ed.), 1-18, North-Holland: Amsterdam, 1985.
10. Perkins, C. ; Teichrow, J. ; and Horstkotte, E. : Fuzzy-C development system : A complete overview, Togai InfraLogic Inc., SOAR-89 conference held at Johnson Space Center, Houston, July 25-27, 1989.
11. Teichrow, J. ; and Horstkotte, E. : Fuzzy-C compiler User's manual, v2.0b, Togai InfraLogic Inc., Irvine, California, April 1989.
12. Lee, C. C. ; and Berenji, H. R. : An Intelligent Controller Based On Approximate Reasoning And Reinforcement Learning, Proc. of IEEE Int. Symposium on Intelligent Control, Albeny, NY 1989.
13. Lea, R. N. : Automated Space Vehicle Control for Rendezvous Proximity Operations, Telematics and Informatics, vol. 5, no. 3, pp 179-185, 1988.
14. Lea, R. N. : Applications of fuzzy sets to Rule-based Expert System Development, Telematics and Informatics, vol. 6, nos. 3/4, pp 403-406, 1989.
15. Edwards, H. C. ; and Bailey, R. : The Orbital Operations Simulator User's Guide, LinCom corporation, ref. LM85-1001-01, June 87.
16. Video Conference Demonstration from Johnson Space Center, International Workshop On Fuzzy Systems Applications (IFSA-88), Iizuka, Fukuoka, Japan, August 20-24 1988.
17. Lea, R. N., Giarratano, J., Fritz, R. H., and Jani, Y. K. : Fuzzy Logic Control for Camera Tracking System, Proceedings of the 8th International Congress of Cybernetics and Systems, New York, June 1990.
18. Pal, S. K. : 'Fuzziness, Image Information and Scene Analysis' in An Introduction to Fuzzy Logic Applications in Intelligent Systems edited by R. R. Yager & L. A. Zadeh, Kluwer Academic Publishers (to appear).

continuous–functions which are very difficult (or impossible) to search using calculus based methods. Genetic algorithms are also blind: that is, they know nothing of the problem being solved other than payoff or penalty (i.e., objective function) information.

The basic iterative model of the genetic algorithms is shown in figure 1. A new population is created from an existing population by means of *evaluation, selection,* and *reproduction.* This process repeats itself until the population converges on an optimal solution or some other stopping condition is reached.

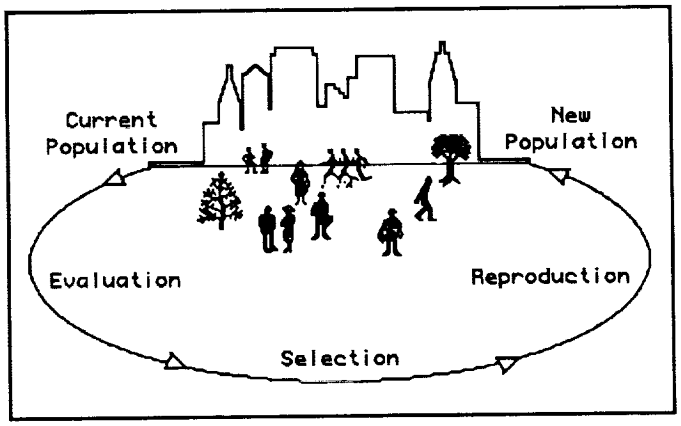


Figure 1. The Iterative Genetic Algorithm Model.

The initial population consists a set of individuals (i.e., potential solutions) generated randomly or heuristically. In the classical genetic algorithm, each member is represented by a fixed-length binary string of bits (a *chromosome*) that encodes parameters of the problem. This encoded string can be decoded to give the integer values for these parameters.

Once the initial population has been created, the evaluation phase begins. The genetic algorithms require that members of the population can be differentiated according to *goodness* or *fitness.* The members that are more fit are given a higher probability of participating during the selection and reproduction phases. Fitness is measured by decoding a chromosome and using the decoded parameters as input to the objective function. The value returned by the objective function (or some transformation of it) is used as the fitness value.

During the selection phase, the population members are given a target sampling rate which is based on fitness and determines how many times a member will mate during this generation–that is, how many offspring from this individual will be created in the next population. The target sampling rate (usually not a whole number) must be transformed into an integer number of matings for each individual. There are many ways of determining the target sampling rate and the actual number of matings. Suffice it to say that individuals that are more fit are given a reproductive advantage over less fit members.

During the reproduction phase, two members of the mating pool (i.e., members of the population with non-zero mating counts) are chosen from random and genetic operators are applied to their genetic material to produce two new members for the next population. This process is repeated until the next population is filled. The recombination phase usually involves two operators: crossover and mutation. A simple crossover operation is illustrated in figure 2. During crossover, the two parents exchange substring information (genetic material) at a random position in the chromosomes to produce two new strings.

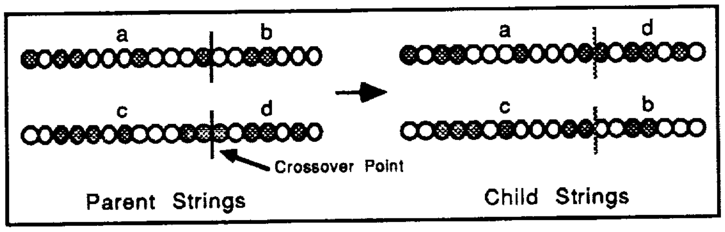


Figure 2. The Crossover Operation.

Crossover occurs according to a crossover probability, usually between 0.5 and 1.0. The crossover operation searches for better *building blocks* within the genetic material which combine to create optimal or near-optimal problem parameters and, therefore, problem solutions, when the string is decoded.

The mutation operation, shown in figure 3, is a secondary genetic algorithm. It is used to maintain deversity in the population–that is, to keep the population from prematurely converging on one solution–and to create genetic material that may not be present in the current population. The mechanics of the mutation operation are simple: for each position in a string created during crossover, change the value at that position according to a mutation probability. The mutation probability is usually very low–less than 0.05.

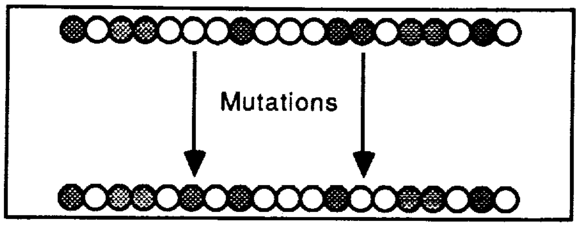


Figure 3. The Mutation Operation.

## Genetic Algorithm Applications

Since genetic algorithms provide a set of efficient, domain-independent search methods, they have been used for a wide range of applications. Table 1 lists some of the GA applications ranging from science and engineering to business and social sciences applications. The following sections briefly describe several of these applications.

Engineering. Goldberg (3) applied genetic algorithms and classifier systems to optimization and machine learning problems in natural gas pipeline control. He focussed on a 10-compressor, 10-pipe, steady-state, serial pipeline problem. The object was to minimize the power consumed subject to maximum and minimum pressure and pressure ratio constraints. Goldberg and Samtani (4) used a simple genetic algorithm to optimize a 10-member plane truss. The objective was to minimize the weight of the structure subject to maximum and minimum stress constraints on each member. In both cases, optimal or near-optimal results were obtained.

Medical Image Processing. Fitzpatrick, et al. (5), used genetic algorithms to perform image registration for an arterial examination system known as digital subtraction angiography. Using this system, a physician examines arteries using two x-rays: one taken of the artery unaltered and one taken after injection of a dye into the artery. The two x-ray images are subtracted one pixel at a time; the result is a picture of the interior wall of the artery. Movement of the artery between the time each x-ray is taken results in a distorted image. Fitzpatrick, et al., used genetic algorithms to find a set of equations that transform or *register* the two images.

Robot Path Planning. A Mobile Transporter system is being designed for on-orbit use with Space Station Freedom which will be capable of traversing the station's truss structure. The Mobile Transporter's function is to facilitate space station maintenance tasks and transportation of material around the station. The Software Technology Branch has investigated the use of genetic algorithms for Mobile Transporter path planning (6). The objectives of these activities are to produce an optimum trajectory for the Mobile Transporter that avoids collisions with objects attached to the truss and to minimize the length of the path between the Mobile Transporter and the target position.

Machine Learning. Genetic algorithms have been used in an area of machine learning called *classifier systems*. Classifier systems learn if-then production rules that guide the performance of a production system. Holland has used classifier systems in studies of economic models, specifically mathematical stock market models. The genetic algorithm creates new rules for trading and selling stocks.

TABLE 1. Genetic Algorithm Applications in Search and Optimization [taken from (2)]

| Year | Investigators | Description |
|---|---|---|
| | BIOLOGY | |
| 1967 | Rosenberg | Simulation of the evolution of single-celled organism populations |
| 1970 | Weinberg | Outline of cell population simulation including metalevel GA |
| 1984 | Perry | Investigation of niche theory and speciation with GAs |
| 1986 | Grosso | Simulation of diploid GA with explicit subpopulations and migration |
| 1987 | Sannier and Goodman | GA adapts structures responding to spatial and temporal food availability |

to create genetic material that may not be present in the current population. The mechanics of the mutation operation are simple: for each position in a string created during crossover, change the value at that position according to a mutation probability. The mutation probability is usually very low–less than 0.05.

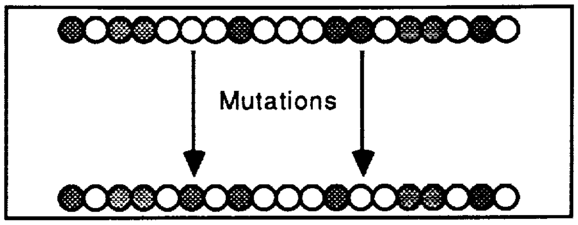


Figure 3. The Mutation Operation.

## Genetic Algorithm Applications

Since genetic algorithms provide a set of efficient, domain-independent search methods, they have been used for a wide range of applications. Table 1 lists some of the GA applications ranging from science and engineering to business and social sciences applications. The following sections briefly describe several of these applications.

Engineering. Goldberg (3) applied genetic algorithms and classifier systems to optimization and machine learning problems in natural gas pipeline control. He focussed on a 10-compressor, 10-pipe, steady-state, serial pipeline problem. The object was to minimize the power consumed subject to maximum and minimum pressure and pressure ratio constraints. Goldberg and Samtani (4) used a simple genetic algorithm to optimize a 10-member plane truss. The objective was to minimize the weight of the structure subject to maximum and minimum stress constraints on each member. In both cases, optimal or near-optimal results were obtained.

Medical Image Processing. Fitzpatrick, et al. (5), used genetic algorithms to perform image registration for an arterial examination system known as digital subtraction angiography. Using this system, a physician examines arteries using two x-rays: one taken of the artery unaltered and one taken after injection of a dye into the artery. The two x-ray images are subtracted one pixel at a time; the result is a picture of the interior wall of the artery. Movement of the artery between the time each x-ray is taken results in a distorted image. Fitzpatrick, et al., used genetic algorithms to find a set of equations that transform or *register* the two images.

Robot Path Planning. A Mobile Transporter system is being designed for on-orbit use with Space Station Freedom which will be capable of traversing the station's truss structure. The Mobile Transporter's function is to facilitate space station maintenance tasks and transportation of material around the station. The Software Technology Branch has investigated the use of genetic algorithms for Mobile Transporter path planning (6). The objectives of these activities are to produce an optimum trajectory for the Mobile Transporter that avoids collisions with objects attached to the truss and to minimize the length of the path between the Mobile Transporter and the target position.

Machine Learning. Genetic algorithms have been used in an area of machine learning called *classifier systems.* Classifier systems learn if-then production rules that guide the performance of a production system. Holland has used classifier systems in studies of economic models, specifically mathematical stock market models. The genetic algorithm creates new rules for trading and selling stocks.

TABLE 1. Genetic Algorithm Applications in Search and Optimization [taken from (2)]

| Year | Investigators | Description |
|---|---|---|
| | BIOLOGY | |
| 1967 | Rosenberg | Simulation of the evolution of single-celled organism populations |
| 1970 | Weinberg | Outline of cell population simulation including metalevel GA |
| 1984 | Perry | Investigation of niche theory and speciation with GAs |
| 1986 | Grosso | Simulation of diploid GA with explicit subpopulations and migration |
| 1987 | Sannier and Goodman | GA adapts structures responding to spatial and temporal food availability |

continuous–functions which are very difficult (or impossible) to search using calculus based methods. Genetic algorithms are also blind: that is, they know nothing of the problem being solved other than payoff or penalty (i.e., objective function) information.

The basic iterative model of the genetic algorithms is shown in figure 1. A new population is created from an existing population by means of *evaluation*, *selection*, and *reproduction*. This process repeats itself until the population converges on an optimal solution or some other stopping condition is reached.

The initial population consists a set of individuals (i.e., potential solutions) generated randomly or heuristically. In the classical genetic algorithm, each member is represented by a fixed-length binary string of bits (a *chromosome*) that encodes parameters of the problem. This encoded string can be decoded to give the integer values for these parameters.

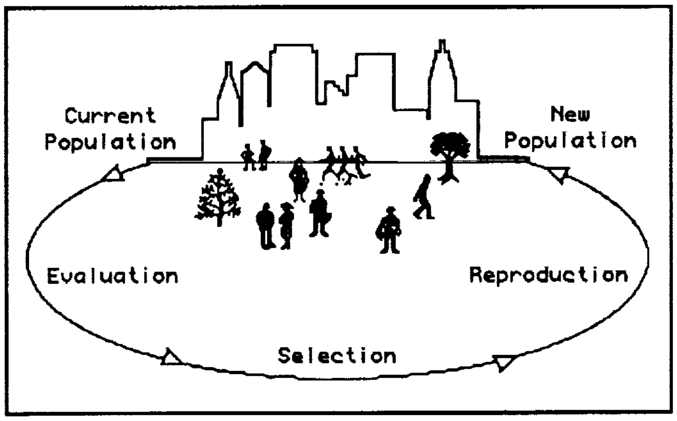


Figure 1. The Iterative Genetic Algorithm Model.

Once the initial population has been created, the evaluation phase begins. The genetic algorithms require that members of the population can be differentiated according to *goodness* or *fitness*. The members that are more fit are given a higher probability of participating during the selection and reproduction phases. Fitness is measured by decoding a chromosome and using the decoded parameters as input to the objective function. The value returned by the objective function (or some transformation of it) is used as the fitness value.

During the selection phase, the population members are given a target sampling rate which is based on fitness and determines how many times a member will mate during this generation–that is, how many offspring from this individual will be created in the next population. The target sampling rate (usually not a whole number) must be transformed into an integer number of matings for each individual. There are many ways of determining the target sampling rate and the actual number of matings. Suffice it to say that individuals that are more fit are given a reproductive advantage over less fit members.

During the reproduction phase, two members of the mating pool (i.e., members of the population with non-zero mating counts) are chosen from random and genetic operators are applied to their genetic material to produce two new members for the next population. This process is repeated until the next population is filled. The recombination phase usually involves two operators: crossover and mutation. A simple crossover operation is illustrated in figure 2. During crossover, the two parents exchange substring information (genetic material) at a random position in the chromosomes to produce two new strings.

Crossover occurs according to a crossover probability, usually between 0.5 and 1.0. The crossover operation searches for better *building blocks* within the genetic material which combine to create optimal or near-optimal problem parameters and, therefore, problem solutions, when the string is decoded.

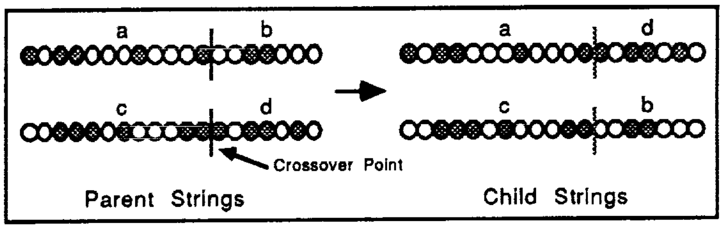


Figure 2. The Crossover Operation.

The mutation operation, shown in figure 3, is a secondary genetic algorithm. It is used to maintain deversity in the population–that is, to keep the population from prematurely converging on one solution–and

TABLE 1. (Continued.)

| Year | Investigators | Description |
|---|---|---|
| | COMPUTER SCIENCE | |
| 1967 | Bagley | Parameter search in hexapawn-like game evaluation function via GA |
| 1979 | Raghavan and Birchard | GA-based clustering algorithm |
| 1982 | Gerardy | Probabilities automation identification attempt via GA |
| 1984 | Gordon | Adaptive document description using GA |
| 1985 | Rendell | GA search for game evaluation function |
| 1987 | Raghavan and Agarwal | Adaptive document clustering using GAs |
| | ENGINEERING & OPERATIONS RESEARCH | |
| 1981 | Goldberg | Mass-spring-dashpot system identification with simple GA |
| 1982 | Etter, Hicks, and Cho | Recursive adaptive filter design using a simple GA |
| 1983 | Goldberg | Steady-state and transient optimization of gas pipeline using GA |
| 1985 | Davis | Bin-packing and graph-coloring problems via GA |
| 1985 | Davis | Outline of job shop scheduling procedure via GA |
| 1985 | Davis and Smith | VLSI circuit layout via GA |
| 1985 | Fourman | VLSI layout compaction with GA |
| 1985 | Goldberg and Kuo | On-off, steady-state optimization of oil pump-pipeline system with GA |
| 1986 | Glover | Keyboard configuration design using a GA |
| 1986 | Goldberg and Samtani | Structural optimization (plane truss) via |
| 1986 | Goldberg and Smith | Blind knapsack problem with simple GA |
| 1986 | Minga | Aircraft landing strut weight optimization with GA |
| 1987 | Davis and Coombs | Communications network link size optimization using GA |
| 1987 | Davis and Rittter | Classroom scheduling via simulated annealing with metalevel GA |
| | IMAGE PROCESSING & PATTERN RECOGNITION | |
| 1970 | Cavicchio | Selection of detectors for binary pattern recognition |
| 1984 | Fitzpatrick, et al. | Image registration via GA to minimize image differences |
| 1985 | Englander | Selection of detectors for known image classification |
| 1985 | Gillies | Search for image feature detectors via GA |
| 1987 | Stadnyk | Explicit pattern class recognition using partial matching |
| | PHYSICAL SCIENCES | |
| 1985 | Shaefer | Nonlinear equation solving with GA for fitting potential surfaces |
| | SOCIAL SCIENCES | |
| 1979 | Reynolds | GA-like adaptation in model of prehistoric hunter-gatherer behavior |
| 1981 | Smith and De Jong | Calibration of population migration model using GA search |
| 1985 | Axelrod | Simulation of the evolution of behavioral norms with GA |
| 1985 | Axelrod | Iterated prisoners dilemma problem solution using GA |

## THE SPLICER PROJECT

This section introduces the Splicer Project. It presents background material and discusses the objectives of the project, the approach taken, results to date, current status and possibilities for future work.

### Background

The Splicer Project is a project within the Software Technology Branch at NASA's Johnson Space Center. The purpose of the project is to develop a tool that will enable the widespread use of genetic algorithm technology.

The charter of the Software Technology Branch is to develop and/or acquire generic software tools for emerging technologies. Genetic algorithms are just one of the many technologies being investigated within the Software Technology Branch: other areas and tools are expert systems (CLIPS), neural networks (NETS), fuzzy logic, scheduling (COMPASS), software reuse, and intelligent computer-aided training.

The MITRE Corporation supports the Software Technology Branch on multiple projects and is responsible for evaluating the viability and robustness of genetic algorithms and for supporting the Software Technology Branch with respect to the development and acquisition of software tools related to this technology.

## Objectives

The Software Technology Branch is interested in applying genetic algorithms within various domains: e.g., robot path planning, job shop scheduling. The original goal of the Splicer Project was to create a flexible, *generic* tool. As such, the tool would:

- Implement the basic genetic algorithms defined in the literature
- Define the interfaces for and allow users to develop interchangeable fitness modules
- Provide a graphic, event-driven user interface

Subsequent goals include the following:

- Distribution of the tool in the public domain
- Support for multiple computing platforms
- Extension of the tool for additional genetic algorithm functionality
- Use of the tool for genetic algorithm research
- Augmentation of the tool and special user interfaces for specific application domains

## Approach

Design. The design chosen for the Splicer tool is shown in figure 4. This design consists of four components: a genetic algorithm kernel and three types of interchangeable libraries or modules: representation libraries, fitness modules, and user interface libraries.

A *genetic algorithm kernel* was developed that is independent of representation (i.e., problem encoding), fitness function, or user interface type. The GA kernel comprises all functions necessary for the manipulation of populations. These functions include the creation of populations and population members, the iterative population model, fitness scaling, parent selection and sampling, and the generation of population statistics. In addition, miscellaneous functions are included in the kernel (e.g., random number generators).

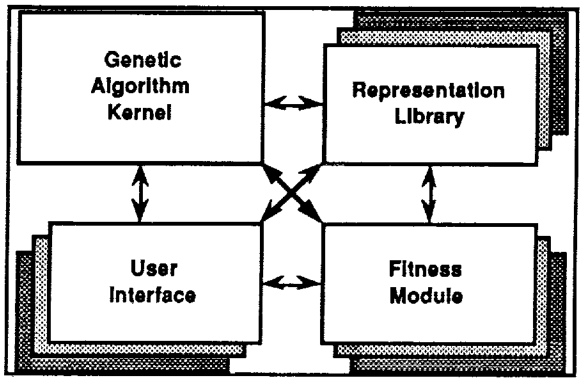


Figure 4. Splicer Project Design

Different types of problem-encoding schemes and functions are defined and stored in interchangeable *representation libraries*. This allows the GA kernel to be used for any representation scheme. At present, the Splicer tool provides representation libraries for binary strings and for permutations. These libraries contain functions for the definition, creation, and decoding of genetic strings, as well as multiple crossover and mutation operators. Furthermore, the Splicer tool defines the appropriate interfaces to allow users to create new representation libraries (e.g., for use with vectors or grammars).

Fitness functions are defined and stored in interchangeable *fitness modules*. Fitness modules are the only piece of the Splicer system a user will normally be required to create or alter to solve a particular problem. Within a

The charter of the Software Technology Branch is to develop and/or acquire generic software tools for emerging technologies. Genetic algorithms are just one of the many technologies being investigated within the Software Technology Branch: other areas and tools are expert systems (CLIPS), neural networks (NETS), fuzzy logic, scheduling (COMPASS), software reuse, and intelligent computer-aided training.

The MITRE Corporation supports the Software Technology Branch on multiple projects and is responsible for evaluating the viability and robustness of genetic algorithms and for supporting the Software Technology Branch with respect to the development and acquisition of software tools related to this technology.

## Objectives

The Software Technology Branch is interested in applying genetic algorithms within various domains: e.g., robot path planning, job shop scheduling. The original goal of the Splicer Project was to create a flexible, *generic* tool. As such, the tool would:

- Implement the basic genetic algorithms defined in the literature
- Define the interfaces for and allow users to develop interchangeable fitness modules
- Provide a graphic, event-driven user interface

Subsequent goals include the following:

- Distribution of the tool in the public domain
- Support for multiple computing platforms
- Extension of the tool for additional genetic algorithm functionality
- Use of the tool for genetic algorithm research
- Augmentation of the tool and special user interfaces for specific application domains

## Approach

Design. The design chosen for the Splicer tool is shown in figure 4. This design consists of four components: a genetic algorithm kernel and three types of interchangeable libraries or modules: representation libraries, fitness modules, and user interface libraries.

A *genetic algorithm kernel* was developed that is independent of representation (i.e., problem encoding), fitness function, or user interface type. The GA kernel comprises all functions necessary for the manipulation of populations. These functions include the creation of populations and population members, the iterative population model, fitness scaling, parent selection and sampling, and the generation of population statistics. In addition, miscellaneous functions are included in the kernel (e.g., random number generators).

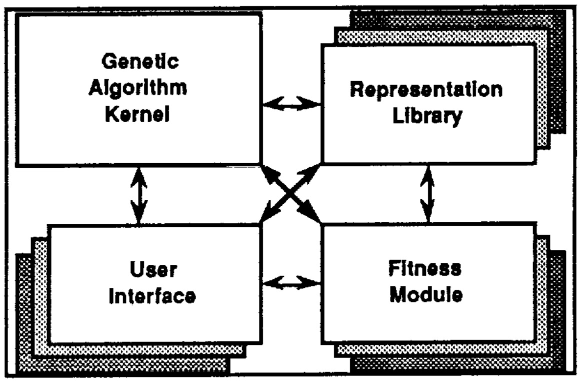


Figure 4. Splicer Project Design

Different types of problem-encoding schemes and functions are defined and stored in interchangeable *representation libraries*. This allows the GA kernel to be used for any representation scheme. At present, the Splicer tool provides representation libraries for binary strings and for permutations. These libraries contain functions for the definition, creation, and decoding of genetic strings, as well as multiple crossover and mutation operators. Furthermore, the Splicer tool defines the appropriate interfaces to allow users to create new representation libraries (e.g., for use with vectors or grammars).

Fitness functions are defined and stored in interchangeable *fitness modules*. Fitness modules are the only piece of the Splicer system a user will normally be required to create or alter to solve a particular problem. Within a

TABLE 1. (Continued.)

| Year | Investigators | Description |
|---|---|---|
| | COMPUTER SCIENCE | |
| 1967 | Bagley | Parameter search in hexapawn-like game evaluation function via GA |
| 1979 | Raghavan and Birchard | GA-based clustering algorithm |
| 1982 | Gerardy | Probabilities automation identification attempt via GA |
| 1984 | Gordon | Adaptive document description using GA |
| 1985 | Rendell | GA search for game evaluation function |
| 1987 | Raghavan and Agarwal | Adaptive document clustering using GAs |
| | ENGINEERING & OPERATIONS RESEARCH | |
| 1981 | Goldberg | Mass-spring-dashpot system identification with simple GA |
| 1982 | Etter, Hicks, and Cho | Recursive adaptive filter design using a simple GA |
| 1983 | Goldberg | Steady-state and transient optimization of gas pipeline using GA |
| 1985 | Davis | Bin-packing and graph-coloring problems via GA |
| 1985 | Davis | Outline of job shop scheduling procedure via GA |
| 1985 | Davis and Smith | VLSI circuit layout via GA |
| 1985 | Fourman | VLSI layout compaction with GA |
| 1985 | Goldberg and Kuo | On-off, steady-state optimization of oil pump-pipeline system with GA |
| 1986 | Glover | Keyboard configuration design using a GA |
| 1986 | Goldberg and Samtani | Structural optimization (plane truss) via |
| 1986 | Goldberg and Smith | Blind knapsack problem with simple GA |
| 1986 | Minga | Aircraft landing strut weight optimization with GA |
| 1987 | Davis and Coombs | Communications network link size optimization using GA |
| 1987 | Davis and Rittter | Classroom scheduling via simulated annealing with metalevel GA |
| | IMAGE PROCESSING & PATTERN RECOGNITION | |
| 1970 | Cavicchio | Selection of detectors for binary pattern recognition |
| 1984 | Fitzpatrick, et al. | Image registration via GA to minimize image differences |
| 1985 | Englander | Selection of detectors for known image classification |
| 1985 | Gillies | Search for image feature detectors via GA |
| 1987 | Stadnyk | Explicit pattern class recognition using partial matching |
| | PHYSICAL SCIENCES | |
| 1985 | Shaefer | Nonlinear equation solving with GA for fitting potential surfaces |
| | SOCIAL SCIENCES | |
| 1979 | Reynolds | GA-like adaptation in model of prehistoric hunter-gatherer behavior |
| 1981 | Smith and De Jong | Calibration of population migration model using GA search |
| 1985 | Axelrod | Simulation of the evolution of behavioral norms with GA |
| 1985 | Axelrod | Iterated prisoners dilemma problem solution using GA |

## THE SPLICER PROJECT

This section introduces the Splicer Project. It presents background material and discusses the objectives of the project, the approach taken, results to date, current status and possibilities for future work.

### Background

The Splicer Project is a project within the Software Technology Branch at NASA's Johnson Space Center. The purpose of the project is to develop a tool that will enable the widespread use of genetic algorithm technology.

fitness module, a user can create a fitness function, set the initial values for various Splicer control parameters (e.g., population size), create a function which graphically draws the best solutions as they are found, and provide descriptive information about the problem being solved. The tool comes with several example fitness modules.

The Splicer tool provides three *user interface libraries*: a Macintosh user interface, an X Window System user interface, and a simple, menu-driven, character-based user interface. The first two user interfaces are event-driven and provide graphic output using windows.

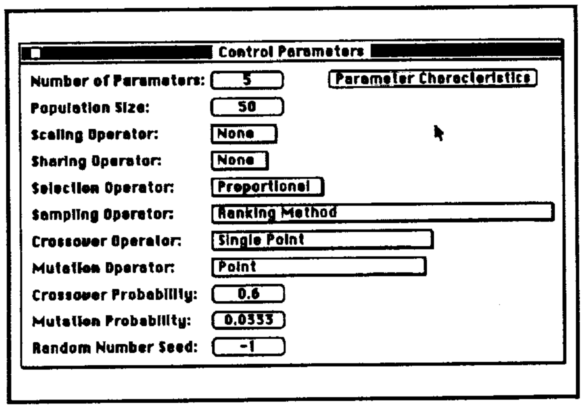
Implementation. The C programming language was chosen for portability and modularization. The original prototype was developed on an Apple Macintosh™ using Symantec's Think C™. This included the development of the Macintosh user interface. The GA kernel, representation libraries, and fitness functions were then ported to a Sun 3/80™ and SPARC™. An X Windows System user interface was then developed using X and the Hewlett-Packard Widget Set.

## Results

This section will present the results to date of the Splicer Project. This will be done using brief descriptions and pictures of components from the Macintosh interface (components of the X Window System interface are very similar).

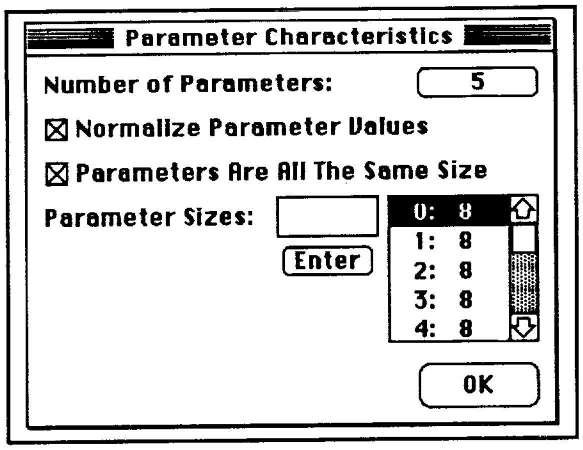
Control Parameters. The **Control Parameters** dialog box, shown in figure 5, allows the user to set the values of multiple parameters that control the operation of the Splicer tool (e.g., population size, crossover operator, mutation probability). This is accomplished in two ways: numeric parameters have buttons associated with them that pop up dialog boxes to allow the user to enter a new value; genetic operators (e.g., the fitness scaling operator) have pop up menus associated with them that allow the user to select from a list of operators.

The **Parameter Characteristics** button on the **Control Parameters** dialog box is used to pop up another dialog box that allows the characteristics of the individual problem definition parameters to be changed (for permutations there are no characteristics to change, therefore this button is disabled). The **Parameter Characteristics** dialog box for binary strings is shown in figure 6. This dialog box allows the user to specify the number of problem parameters and their size in number of bits. It also allows the parameter values to be normalized during decoding to create floating point numbers.



Number of Parameters: 5
Population Size: 50
Scaling Operator: None
Sharing Operator: None
Selection Operator: Proportional
Sampling Operator: Ranking Method
Crossover Operator: Single Point
Mutation Operator: Point
Crossover Probability: 0.6
Mutation Probability: 0.0333
Random Number Seed: -1

Figure 5. Control Parameters Dialog Box.



Number of Parameters: 5
☒ Normalize Parameter Values
☒ Parameters Are All The Same Size
Parameter Sizes:
0: 8
1: 8
2: 8
3: 8
4: 8

Figure 6. The Parameter Characteristics Dialog Box for Binary Strings.

Program Control. The operation of the Splicer program is controlled using options on the **Control** menu, shown in figure 7, as well as other menus (some of these are shown below). The **Control Parameters** dialog box is displayed using the **Set Control Parameters...** option. To create a random population, the **Create Population** option is selected. The **Run** option starts the execution of the genetic algorithms on the existing population. To begin again with a clean slate, the **Reinitialize...** option is used.

Operators. The Splicer program provides multiple genetic operators for fitness scaling, (in the future: fitness sharing), parent selection, parent sampling, crossover, and mutation. These operators can be changed at any time, even while the genetic algorithms are executing, and are selected using either the **Operators** menu (shown in figure 8) or the menu buttons on the **Control Parameters** dialog box.

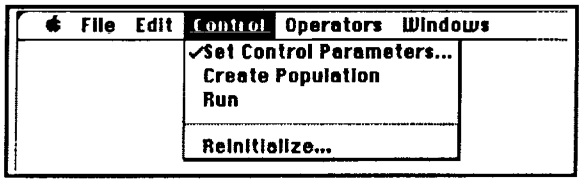


Figure 7. The Control Menu.

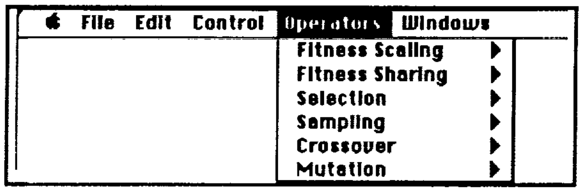


Figure 8. The Operators Menu.

Fitness Scaling. Splicer provides a linear fitness scaling option, as shown in figure 9. Fitness scaling is useful near the end of a genetic algorithm run when all members of the population have high fitness. Scaling spreads the fitness values and gives fitter members a higher reproductive advantage. Scaling can be turned off.

Fitness Sharing. While there is a place holder on the operators menu for fitness sharing, this option has not been implemented yet.

Parent Selection. Parent selection can be performed using either proportional selection (i.e., using relative fitness values) or using linear rank selection (where population members are simply ranked according to fitness), as shown in figure 10.

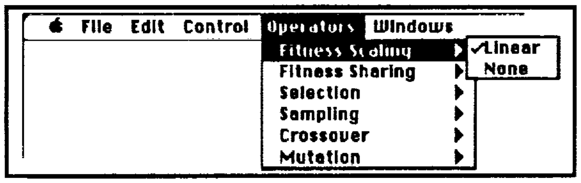


Figure 9. The Fitness Scaling Menu.

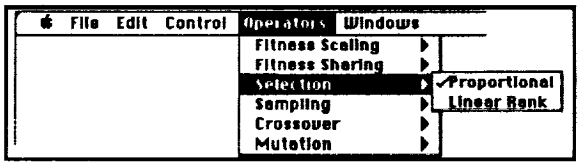


Figure 10. The Selection Menu.

Parent Sampling. Parents can be sampled using one of the methods shown in figure 11.

Crossover. The availability of crossover operators depends upon the representation library being used. The crossover operators provided by the binary string library are shown in figure 12. The crossover operators provided by the permutation library are shown in figure 13.

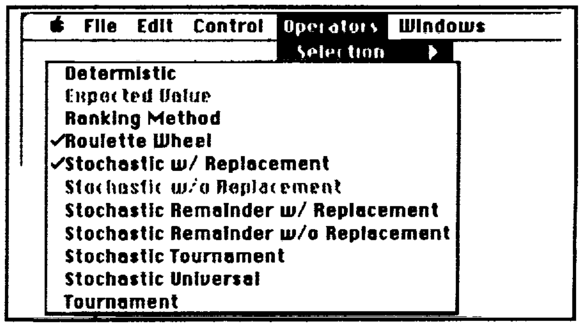


Figure 11. The Sampling Menu.

Mutation. Similarly, the availability of mutation operators depends upon the representation library being used. The mutation operators provided by the binary string library are shown in figure 14. The mutation operators provided by the permutation library are shown in figure 15.

Program Control. The operation of the Splicer program is controlled using options on the **Control** menu, shown in figure 7, as well as other menus (some of these are shown below). The **Control Parameters** dialog box is displayed using the **Set Control Parameters...** option. To create a random population, the **Create Population** option is selected. The **Run** option starts the execution of the genetic algorithms on the existing population. To begin again with a clean slate, the **Reinitialize...** option is used.

Operators. The Splicer program provides multiple genetic operators for fitness scaling, (in the future: fitness sharing), parent selection, parent sampling, crossover, and mutation. These operators can be changed at any time, even while the genetic algorithms are executing, and are selected using either the **Operators** menu (shown in figure 8) or the menu buttons on the **Control Parameters** dialog box.

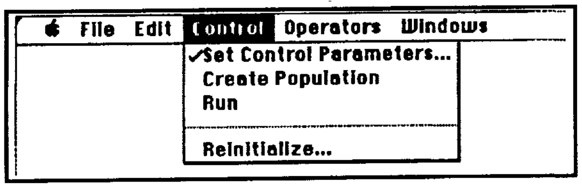

File Edit Control Operators Windows
✓Set Control Parameters...
Create Population
Run
Reinitialize...

Figure 7. The Control Menu.

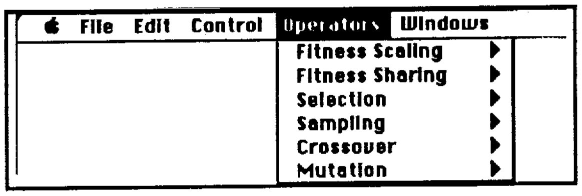


Figure 8. The Operators Menu.

Fitness Scaling. Splicer provides a linear fitness scaling option, as shown in figure 9. Fitness scaling is useful near the end of a genetic algorithm run when all members of the population have high fitness. Scaling spreads the fitness values and gives fitter members a higher reproductive advantage. Scaling can be turned off.

Fitness Sharing. While there is a place holder on the operators menu for fitness sharing, this option has not been implemented yet.

Parent Selection. Parent selection can be performed using either proportional selection (i.e., using relative fitness values) or using linear rank selection (where population members are simply ranked according to fitness), as shown in figure 10.

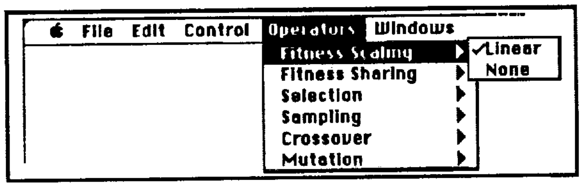


Figure 9. The Fitness Scaling Menu.

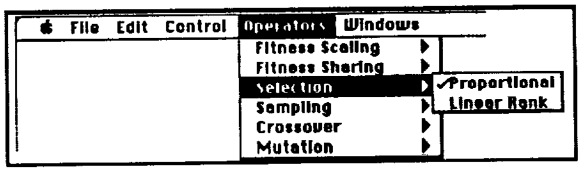


Figure 10. The Selection Menu.

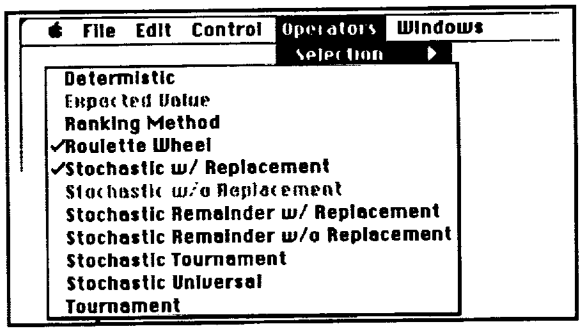

File Edit Control Operators Windows
Selection
Determistic
Expected Value
Ranking Method
✓Roulette Wheel
✓Stochastic w/ Replacement
Stochastic w/o Replacement
Stochastic Remainder w/ Replacement
Stochastic Remainder w/o Replacement
Stochastic Tournament
Stochastic Universal
Tournament

Figure 11. The Sampling Menu.

Parent Sampling. Parents can be sampled using one of the methods shown in figure 11.

Crossover. The availability of crossover operators depends upon the representation library being used. The crossover operators provided by the binary string library are shown in figure 12. The crossover operators provided by the permutation library are shown in figure 13.

Mutation. Similarly, the availability of mutation operators depends upon the representation library being used. The mutation operators provided by the binary string library are shown in figure 14. The mutation operators provided by the permutation library are shown in figure 15.

fitness module, a user can create a fitness function, set the initial values for various Splicer control parameters (e.g., population size), create a function which graphically draws the best solutions as they are found, and provide descriptive information about the problem being solved. The tool comes with several example fitness modules.

The Splicer tool provides three *user interface libraries*: a Macintosh user interface, an X Window System user interface, and a simple, menu-driven, character-based user interface. The first two user interfaces are event-driven and provide graphic output using windows.

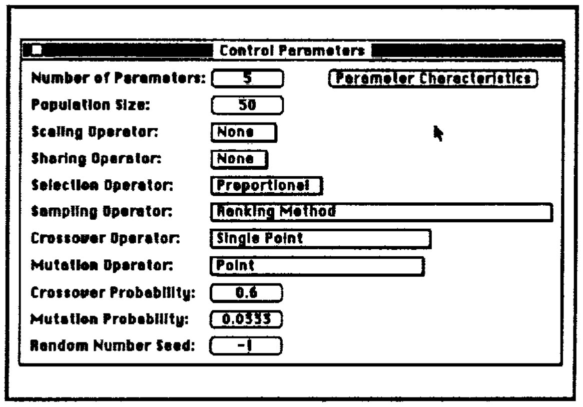
Implementation. The C programming language was chosen for portability and modularization. The original prototype was developed on an Apple Macintosh™ using Symantec's Think C™. This included the development of the Macintosh user interface. The GA kernel, representation libraries, and fitness functions were then ported to a Sun 3/80™ and SPARC™. An X Windows System user interface was then developed using X and the Hewlett-Packard Widget Set.

## Results

This section will present the results to date of the Splicer Project. This will be done using brief descriptions and pictures of components from the Macintosh interface (components of the X Window System interface are very similar).

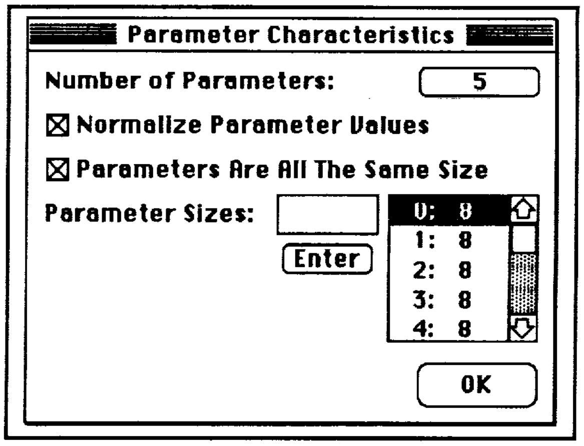
Control Parameters. The **Control Parameters** dialog box, shown in figure 5, allows the user to set the values of multiple parameters that control the operation of the Splicer tool (e.g., population size, crossover operator, mutation probability). This is accomplished in two ways: numeric parameters have buttons associated with them that pop up dialog boxes to allow the user to enter a new value; genetic operators (e.g., the fitness scaling operator) have pop up menus associated with them that allow the user to select from a list of operators.

The **Parameter Characteristics** button on the **Control Parameters** dialog box is used to pop up another dialog box that allows the characteristics of the individual problem definition parameters to be changed (for permutations there are no characteristics to change, therefore this button is disabled). The **Parameter Characteristics** dialog box for binary strings is shown in figure 6. This dialog box allows the user to specify the number of problem parameters and their size in number of bits. It also allows the parameter values to be normalized during decoding to create floating point numbers.



Number of Parameters: 5
Population Size: 50
Scaling Operator: None
Sharing Operator: None
Selection Operator: Proportional
Sampling Operator: Ranking Method
Crossover Operator: Single Point
Mutation Operator: Point
Crossover Probability: 0.6
Mutation Probability: 0.0333
Random Number Seed: -1

Figure 5. Control Parameters Dialog Box.



Number of Parameters: 5
☒ Normalize Parameter Values
☒ Parameters Are All The Same Size
Parameter Sizes:
0: 8
1: 8
2: 8
3: 8
4: 8

Figure 6. The Parameter Characteristics Dialog Box for Binary Strings.

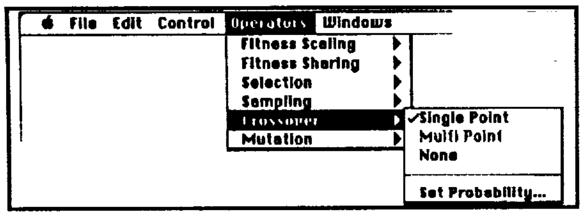


Figure 12. The Crossover Menu Using the Binary String Library.

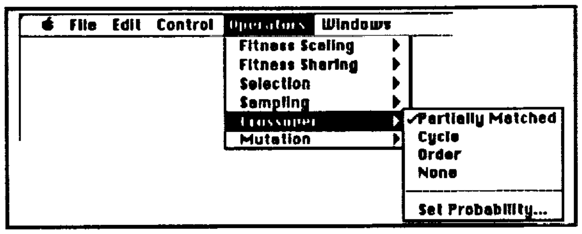


Figure 13. The Crossover Menu Using the Permutation Library.

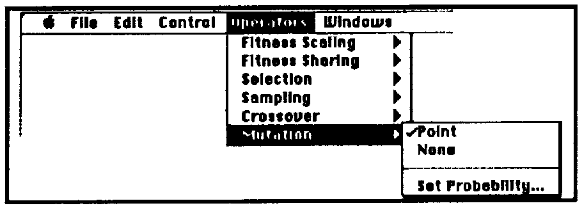


Figure 14. The Mutation Menu Using the Binary String Library.

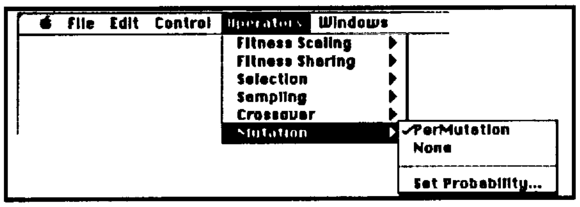
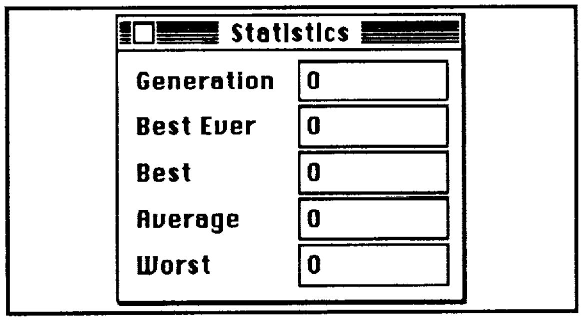


Figure 15. The Mutation Menu Using the Permutation Library.

Output. Various windows present information to the user as the genetic algorithms execute. These windows are described briefly in the following sections.

The Statistics Window. The **Statistics Window**, shown in figure 16, displays the current generation number along with numeric objective function measures of the best solution ever found and the best, average, and worst members of the current population.

The Fitness Window. The **Fitness Window**, shown in figure 17, displays the fitness distribution of the current population. Fitness values are normalized, using the best ever fitness value, to create fitness values between 0.0 and 1.0. This interval is divided into 0.1 size bins and the percentage of the population in each bin is presented as a histogram.



Statistics

| | |
|---|---|
| Generation | 0 |
| Best Ever | 0 |
| Best | 0 |
| Average | 0 |
| Worst | 0 |

Figure 16. The Statistics Window.

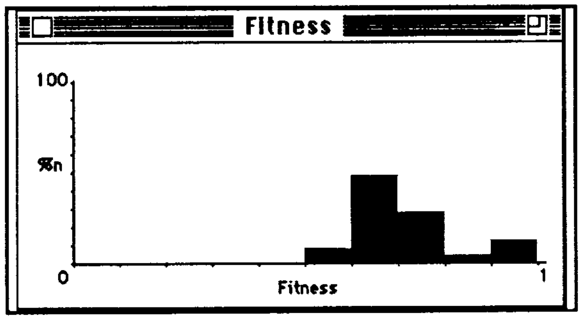


Figure 17. The Fitness Window.

The Objective Window. The **Objective Window**, shown in figure 18, displays the numeric information from the **Statistics Window** but in historic form as a strip chart.

The User Window. The **User Window**, shown in figure 19, is controlled by the developer of the fitness module. The Splicer program provides simple drawing commands that the developer can use to draw in this window. The user is notified whenever a "best ever" solution has been found. One way this window can be used is to graphically display information about this solution. For example, in the window shown, the best

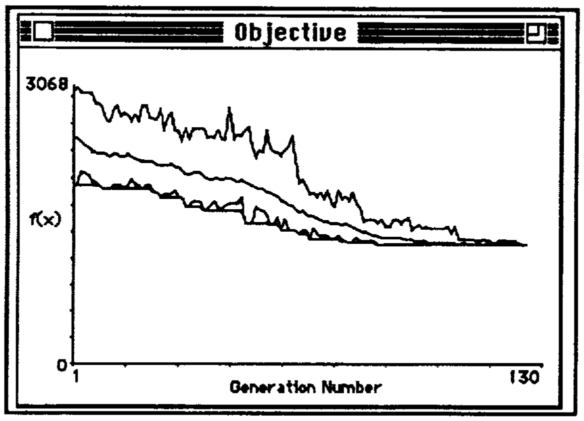


Figure 18. The Objective Window.

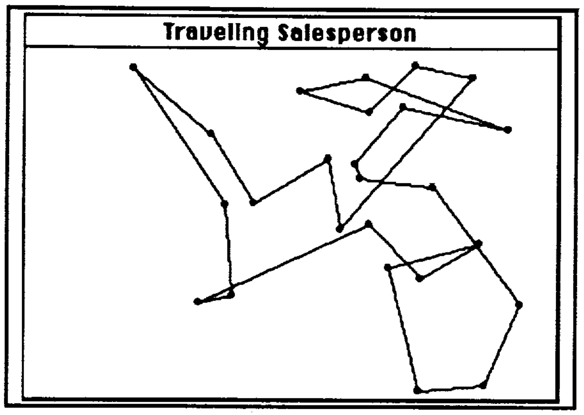


Figure 19. The User Window.

solution (so far) for a 25 city traveling salesperson problem is drawn. The numeric value of this solution would simultaneously appear in the **Statistics Window.**

The Trace Window. The **Trace Window**, shown in figure 20, presents the algorithm used by the genetic algorithms and highlights the current activity the genetic algorithms are performing by filling the appropriate box.

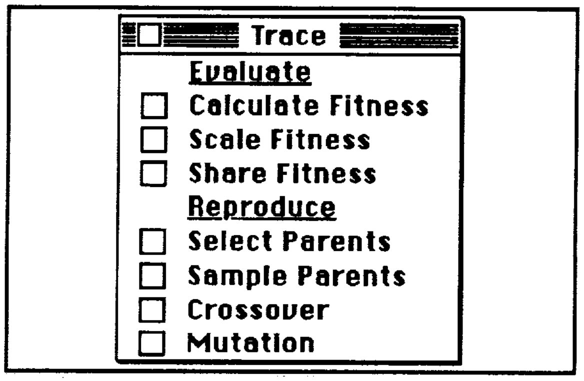


Figure 20. The Trace Window.

Status. An early version of the Macintosh-based Splicer tool was released for beta testing in July, 1990. During the beta testing period, the tool was ported to a Sun 3/80 and an X Window System user interface was developed. Significant new functionality, derived from the literature and suggestions from beta test feedback, and bug fixes were then incorporated into both versions. Documentation (user and reference manuals) is currently being developed. Version 1.0 of the Splicer genetic algorithm tool (both Macintosh and X Windows) will be released within the near future.

Future Work. Future work for this project includes the following:

- Additional functionality (e.g., *steady state* GAs, fitness sharing, other crossover or mutation operators)
- Genetic algorithm research
- Application of GAs and the Splicer tool within specific domains (e.g., job shop scheduling)
- Augmentation of the Splicer tool to create an application-specific tool within a particular domain

## REFERENCES

1. Holland, John H.: Adaptation in Natural and Artificial Systems. University of Michigan Press (Ann Arbor), 1975.

2. Goldberg, David E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wessley Publishing Company, Inc. (Reading), 1989.

3. Goldberg, David E.: Computer-aided Gas Pipeline Operation using Genetic Algorithms and Rule Learning. Doctoral Dissertation, University of Michigan, 1983.

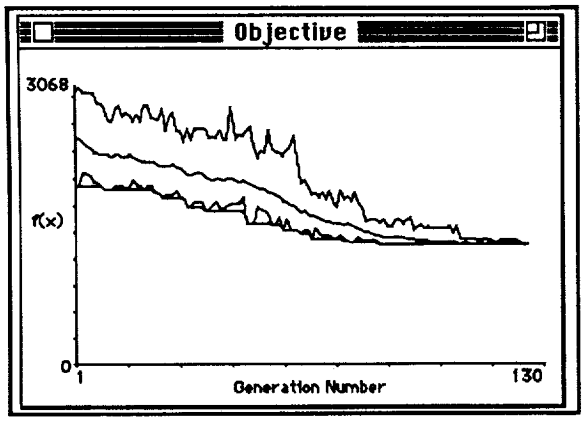


Figure 18. The Objective Window.

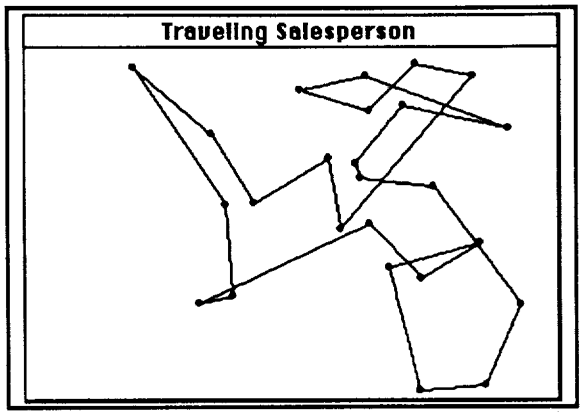


Figure 19. The User Window.

solution (so far) for a 25 city traveling salesperson problem is drawn. The numeric value of this solution would simultaneously appear in the **Statistics Window**.

<u>The Trace Window</u>. The **Trace Window**, shown in figure 20, presents the algorithm used by the genetic algorithms and highlights the current activity the genetic algorithms are performing by filling the appropriate box.

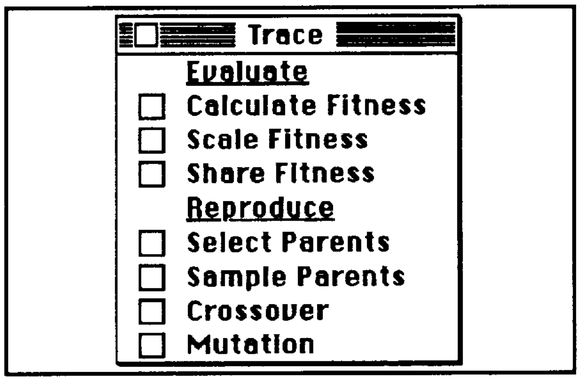


Figure 20. The Trace Window.

<u>Status</u>. An early version of the Macintosh-based Splicer tool was released for beta testing in July, 1990. During the beta testing period, the tool was ported to a Sun 3/80 and an X Window System user interface was developed. Significant new functionality, derived from the literature and suggestions from beta test feedback, and bug fixes were then incorporated into both versions. Documentation (user and reference manuals) is currently being developed. Version 1.0 of the Splicer genetic algorithm tool (both Macintosh and X Windows) will be released within the near future.

<u>Future Work</u>. Future work for this project includes the following:

- Additional functionality (e.g., *steady state* GAs, fitness sharing, other crossover or mutation operators)
- Genetic algorithm research
- Application of GAs and the Splicer tool within specific domains (e.g., job shop scheduling)
- Augmentation of the Splicer tool to create an application-specific tool within a particular domain

## REFERENCES

1. Holland, John H.: Adaptation in Natural and Artificial Systems. University of Michigan Press (Ann Arbor), 1975.

2. Goldberg, David E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wessley Publishing Company, Inc. (Reading), 1989.

3. Goldberg, David E.: Computer-aided Gas Pipeline Operation using Genetic Algorithms and Rule Learning. Doctoral Dissertation, University of Michigan, 1983.

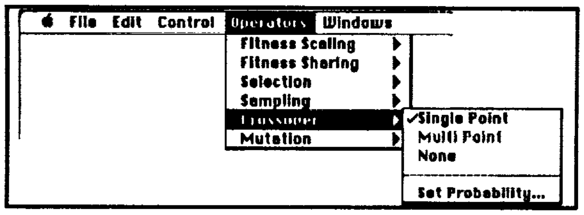


Figure 12. The Crossover Menu Using the Binary String Library.

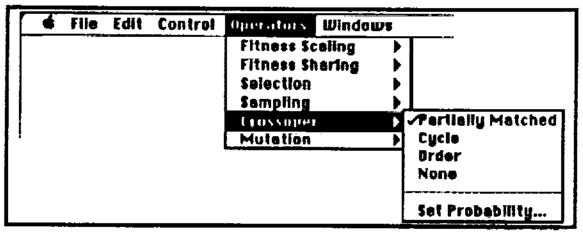


Figure 13. The Crossover Menu Using the Permutation Library.

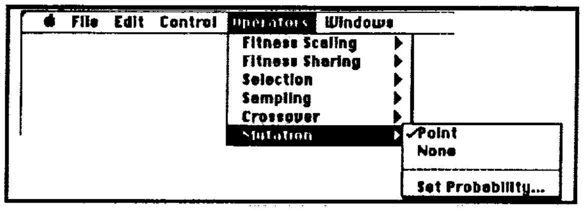


Figure 14. The Mutation Menu Using the Binary String Library.

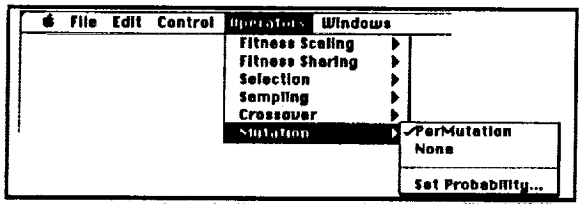
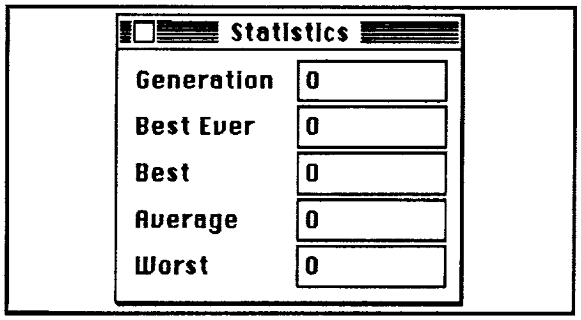


Figure 15. The Mutation Menu Using the Permutation Library.

Output. Various windows present information to the user as the genetic algorithms execute. These windows are described briefly in the following sections.

The Statistics Window. The **Statistics Window**, shown in figure 16, displays the current generation number along with numeric objective function measures of the best solution ever found and the best, average, and worst members of the current population.

The Fitness Window. The **Fitness Window**, shown in figure 17, displays the fitness distribution of the current population. Fitness values are normalized, using the best ever fitness value, to create fitness values between 0.0 and 1.0. This interval is divided into 0.1 size bins and the percentage of the population in each bin is presented as a histogram.


Statistics

| | |
|---|---|
| Generation | 0 |
| Best Ever | 0 |
| Best | 0 |
| Average | 0 |
| Worst | 0 |

Figure 16. The Statistics Window.

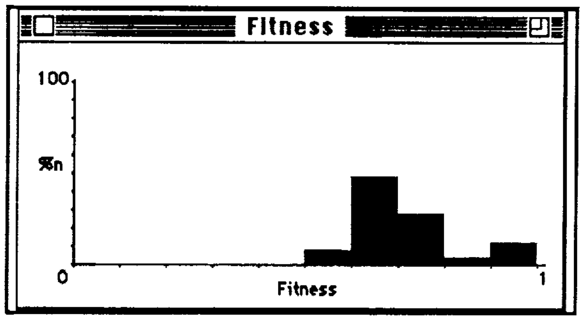


Figure 17. The Fitness Window.

The Objective Window. The **Objective Window**, shown in figure 18, displays the numeric information from the **Statistics Window** but in historic form as a strip chart.

The User Window. The **User Window**, shown in figure 19, is controlled by the developer of the fitness module. The Splicer program provides simple drawing commands that the developer can use to draw in this window. The user is notified whenever a "best ever" solution has been found. One way this window can be used is to graphically display information about this solution. For example, in the window shown, the best

4. Goldberg, David E.; and Samtani, M. P.: Engineering Optimization via Genetic Algorithms. Proceedings of the Ninth Conference on Electronic Computation, 1986, p. 471-482.

5. Fitzpatrick, J.M.; Greffenstette, J.J.; and Van Gucht, D.: Image registration by genetic search. Proceedings of IEEE Southeast Conference, 1984, pp. 460-464.

6. Baffes, P.; and Wang, L.: Mobile Transporter Path Planning Using a Genetic Algorithm Approach. Proceedings of SPIE's Cambridge Symposium on Advances in Intelligent Robotics Systems, Cambridge, MA, 1988.

N91-24051

# INTEGRATED VERTICAL BLOCH LINE (VBL) MEMORY

R. R. Katti, J. C. Wu, and H. L. Stadler
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

## ABSTRACT

Vertical Bloch Line (VBL) Memory is a recently conceived, integrated, solid-state, block-access, VLSI memory which offers the potential of 1 Gbit/cm$^2$ areal storage density, data rates of hundreds of megabits per second, and submillisecond average access times simultaneously at relatively low mass, volume, and power values when compared to alternative technologies. VBLs are micromagnetic structures within magnetic domain walls which can be manipulated using magnetic fields from integrated conductors. The presence or absence of VBL pairs are used to store binary information. At present, efforts are being directed at developing a single-chip memory using 25 Mbit/cm$^2$ technology in magnetic garnet material which integrates, at a single operating point, the writing, storage, reading, and amplification functions needed in a memory. This paper describes the current design architecture, functional elements, and supercomputer simulation results which are used to assist the design process.

## INTRODUCTION

Vertical Bloch Line (VBL) Memory[1, 2, 3] is a solid-state, radiation-hard, nonvolatile, block access, magnetic VLSI memory. Research and development efforts for this novel memory are being pursued in the United States, Europe, and Japan. Table 1 shows the potential storage density that is achievable with VBL memory. The densities are a function of stripe width and line feature width, which are defined respectively by the magnetic garnet material and the lithographic process.

In a VBL memory, information is stored using VBL pairs in magnetic stripe domains in garnets. The presence or absence of a Vertical Bloch Line pair in a bit-cell location defines a binary "1"and "0," respectively. Input to the chip is performed by converting currents into magnetic bubbles and then into VBL pairs. Output sensing is performed by converting VBL pairs into magnetic bubbles and sensing magnetic bubbles magnetoresistively.

## PRESENT DEVICE DESIGN

### Fabrication

The present design uses the magnet garnet, $(BiYGdHoCa)_3(FeGeSi)_5O_{12}$, as the storage medium. The thickness, stripe width, collapse field, saturation magnetization, and anisotropy field of the film is approximately 2.4 $\mu$ m, 2.4 $\mu$ m, 230 Oe, 450 Oe, and 1800 Oe, respectively. The film is grown epitaxially on a non-magnetic gadolinium-gallium-garnet (GGG) substrate. These films are transparent but also have a large Faraday rotation, so that magnetic stripes, magnetic bubbles, and, under certain conditions, VBLs can be observed magneto-optically with polarized light using the Faraday effect in a polarized light microscope.

The magnetic garnet has perpendicular magnetic anisotropy so that the magnetization lies perpendicular to the film plane, with the bulk of the film magnetized in one direction, and the stripes magnetized in the opposite direction. A magnetic domain wall is the boundary between the stripe's magnetization and the magnetization of the rest of the film. A twist of magnetization in the domain wall in the plane of the film is a VBL, and two such twists form a VBL pair. If the chirality, or sense of rotation, of the VBLs in the wall is the same, the VBL pair is stable, with a size calculated to be much less than 1 $\mu$ m. The VBL pair is bound together energetically by VBL demagnetizing field energy and magnetic exchange energy.