AD_______________

MIPR 96MM6722

TITLE: Emerging Diseases as Complex Adaptive Systems/Pilot

PRINCIPAL INVESTIGATOR: John J. Grefenstette, Ph.D.

CONTRACTING ORGANIZATION: Naval Research Laboratory
Washington, DC 20375-5337

REPORT DATE: October 1997

TYPE OF REPORT: Final

PREPARED FOR: U.S. Army Medical Research and Materiel Command
Fort Detrick, Maryland 21702-5012

DISTRIBUTION STATEMENT: Approved for public release;
distribution unlimited

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** October 1997	**3. REPORT TYPE AND DATES COVERED** Final (15 May 96 - 30 Sep 97)

4. TITLE AND SUBTITLE Emerging Diseases as Complex Adaptive Systems/Pilot	**5. FUNDING NUMBERS** 96MM6722
6. AUTHOR(S) John J. Grefenstette , Ph.D.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Research Center Washington, DC 20375-5337	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Medical Research and Materiel Command Fort Detrick, Maryland 21702-5012	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200*

We designed and tested a computational model of emerging viruses that simulates the evolution of real biological viruses. Knowledge of viral molecular replication mechanisms was used to instruct the development of this genetic algorithm-based C language "Virtual Virus" (VIV). The model consists of populations of hundreds to thousands of variable length virtual virus genomes that replicate, mutate, recombine, and evolve. Each virus genome is composed of an artificial polynucleotide string in which arbitrary "nucleotide" triplets encode English letters rather than amino acids, and in which sequences are translated into words or phrases, rather than into polypeptides or proteins. The three word/phrases "COREPROTEIN," "POLYMERASE," and "ENVELOPE," which can be present in any order on the string, together comprise the selected phenotype. Run-on and overlapping reading frames are permitted. Fitness is assigned to each string according to the encoded spelling score. Probability of replication at each generation is directly related to string fitness. VIV populations seeded with random strings regularly evolve terse, high spelling score genomes within a few hundred to a thousand generations. By systematically varying evolutionary operators in the VIV model we observed several reproducible features relevant to the evolution and emergence of biological viruses: (1) adaptation (fitness slope) proceeds most rapidly at mutation rates close to one per genome, and falls off rapidly at rates either higher or lower than unity (2) when added to mutation, recombination in any form speeds adaptation, and (3) homologous recombination is superior to random cross-over recombination. The VIV model is designed so that it can be conveniently modified to incorporate a variety of additional evolutionary operators, such as genome segmentation, genomic secondary structures, insertions and deletions, and feed-back loops and hypercycles. Sub-population phylogenies and individual ancestries can be recalled, displayed, and analyzed with a JAVA-based visualization tool.

14. SUBJECT TERMS Emerging diseases, viral evolution, computational model, evolutionary computation.	**15. NUMBER OF PAGES** 53
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	Unlimited

FOREWORD

Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the U.S. Army.

____ Where copyrighted material is quoted, permission has been obtained to use such material.

____ Where material from documents designated for limited distribution is quoted, permission has been obtained to use the material.

____ Citations of commercial organizations and trade names in this report do not constitute an official Department of Army endorsement or approval of the products or services of these organizations.

____ In conducting research using animals, the investigator(s) adhered to the "Guide for the Care and Use of Laboratory Animals," prepared by the Committee on Care and Use of Laboratory Animals of the Institute of Laboratory Resources, National Research Council (NIH Publication No. 86-23, Revised 1985).

____ For the protection of human subjects, the investigator(s) adhered to policies of applicable Federal Law 45 CFR 46.

____ In conducting research utilizing recombinant DNA technology, the investigator(s) adhered to current guidelines promulgated by the National Institutes of Health.

____ In the conduct of research utilizing recombinant DNA, the investigator(s) adhered to the NIH Guidelines for Research Involving Recombinant DNA Molecules.

____ In the conduct of research involving hazardous organisms, the investigator(s) adhered to the CDC-NIH Guide for Biosafety in Microbiological and Biomedical Laboratories.

[illegible] 9/30/97

PI - Signature Date

Contents

1 Introduction

1.1 Subject

This report constitute the final report for the project "Emerging Diseases as Complex Adaptive Systems", sponsored by the Walter Reed Army Institute of Research (WRAIR) and performed by the Navy Center for Applied Research in Artificial Intelligence (NCARAI) at the Naval Research Laboratory (NRL). This project addresses the issue of emerging virus epidemics by focusing on methods of viral evolution. This report describes the computational model that has been developed and presents pilot computational studies on a series of questions related to the evolvability of viruses.

1.2 Purpose

The objectives of this project were to:

- Understand emerging virus epidemics by focusing on methods of viral evolution, and
- Develop and evaluate novel evolutionary modeling approaches.

The approach taken was to develop computational models of virus evolution that may provide a means to answer fundamental questions concerning the evolved structure of the viral genotypes, the dynamics of cross-species infection, and the role of alternative recombination strategies exhibited by viruses. This pilot study applies the methods of evolutionary algorithms to the problem of modeling biological systems, extending the optimization-based genetic algorithm in the direction of a more plausible model of biological evolution.

1.3 Scope of Research

In this pilot study, evolutionary models of computation were extended to the domain of viral evolution. Previous work on evolutionary computation models, including techniques such as genetic algorithms, evolution strategies, and evolutionary programming, has generally focused on optimization and learning methods in engineering and science applications. There have been few applications of these methods to the problem of modeling biological systems. This project thus provides a partial bridge between the two fields, extending the optimization-based genetic algorithm in the direction of a more plausible model of biological evolution.

The scope of the research during the period of 15 May 1996 to 30 September 1997 was to:

1. Design and implement an evolutionary computation model of viral infection.
2. Perform pilot computational studies with the model to investigate questions of interest in the evolution of emerging infections.

These objectives have been accomplished. The pilot computational simulation, described below, illustrate the kind of investigations that the computer model enables. While a complete investigation of the model is beyond the scope of this effort, Section 4 contains our recommendations for further investigation using the model that are expected to provide important new insights into the evolution of emerging diseases.

1.4 Background

1.4.1 Viral Evolution

RNA viruses are - by several orders of magnitude - the most genetically labile "life forms" [22, 32]. Mutation rates for RNA viruses are typically on the order of one error per 10,000 nucleotides replicated, compared to one per ten million nucleotides for larger DNA-based life forms like vertebrates [11]. Since the average genome length of RNA viruses is only 10,000 nucleotides, and all are shorter than 40,000 nucleotides, almost all new viral RNA strands differ from their parent strand by one or more nucleotides.

The nucleotide sequence a viral genome be thought of as an information string of 10,000 bits with four alternative states (A,C,G, U or T) per bit. The total evolutionary potential for such a system is the universe of all possible 10,000-bit strings. These can be hypothetically arranged in a "sequence space" so that each string is adjacent to its 30,000 one-step nearest neighbors. The total dimensions of this space is 4 to the 10,000th power, a number that is greater than all atoms in the universe. Obviously, most regions in this hypothetical RNA sequence space does not support viral RNA replication, but many regions do permit replication, and within these there are local optima. These optima can be conceptualized as peaks on a fitness landscape in nucleotide sequence space [36, 43], as illustrated in Figure 1.

Similarly, evolution can be thought of as the process whereby sequence space is explored, with successful variants colonizing the fitness peaks. Because the mutation rate of RNA replication is so high, evolutionary time for exploration of sequence space for RNA based life can be measured in weeks and years, compared to the millennia required for DNA based life. Rephrased, evolution of RNA life occurs on a scale that can be comprehended - and studied - within human dimensions. The disparity between rates of RNA and DNA evolution - the difference in RNA and DNA "evolvability" - probably accounts for the fact that most of the new emerging diseases are RNA viruses. RNA based genomes have sufficient plasticity to permit rapid host switching. See Table 1.

This high evolvability of RNA may also account for the fact that almost all known arthropod-borne viruses - viruses that alternately replicate within vertebrate and arthropod cells - have RNA genomes. Although there are numerous DNA viruses of vertebrates and numerous DNA viruses of arthropods, remarkably there is only a single known DNA arbovirus (African Swine Fever Virus) [33]. It is likely that for most arthropod-borne viruses the sequence space fitness peak for growth in arthropod cells is close to, but not perfectly congruent with, that for growth in vertebrate cells, and mutation is required to trampoline back and forth through sequence space between the two host-specific optima.

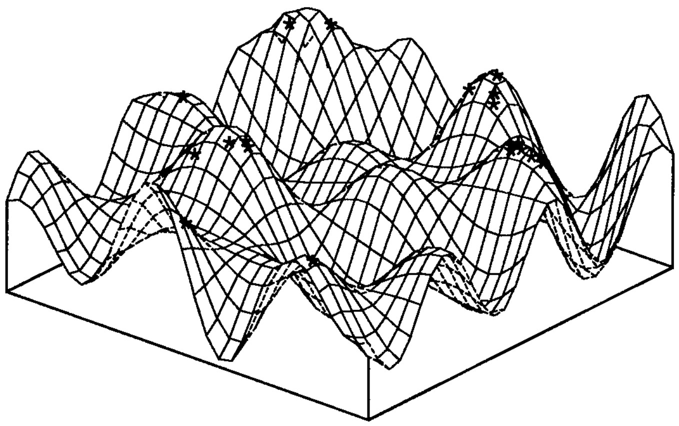


Figure 1: Representation of evolving bit strings in sequence space. Here the sequence space is shown only in two dimensions, the x and y axes. For a string of length L, the strings would evolve through an L-dimensional sequence space, but this is impossible to draw on a two-dimensional paper surface. Fitness is represented as the height on the z axis. In this example populations of strings are colonizing several local fitness optima.

Mutation alone may be insufficient to permit movement through some regions of sequence space [12, 13, 27]. By definition, even single step mutations from a local fitness optimum are less fit than their parents. Particularly in rugged fitness landscapes, genomes only slightly removed from the local optimum may be totally unfit, so that exploration of the surrounding space becomes impossible. Recombination between genomes on separated fitness optima permits such an "evolutionary broad jumping" type of sequence space exploration; recombinant progeny may fall on previously totally unexplored fitness peaks [26]. Naturally occurring recombination (or reassortment) has been closely studied in RNA viruses with segmented genomes such as influenza. Recombination has also recently been shown to occur commonly among HIV strains [4, 5]. The role of recombination in nature is less well studied for other RNA viruses, but convincing examples can been found wherever they have been sought [1, 6].

The biological consequences of such recombination, whether by reassortment of segmented genomes or true recombination through crossing over, may be the generation of novel variants with new epidemiological properties. For influenza, change by mutation, widely known as "drift," is of minor epidemiological significance, while change by recombination often results in a "shift" with an epidemiological impact felt on a global scale [28]. It is now clear that "shifts" in influenza come about through recombination (reassortment) of RNA sequences from bird and pig influenza viruses with sequences from human viruses. The same model may be

Virus	Genome Size: ν (number of nt or bp)	Error Rate: $(1-q)$ (per replication per nt)	Error Rate: $\nu(1-q)$ (per replication per genome)
RNA			
Bacteriophage Q_β	4200	3×10^{-4}	1.3
Polio-1 virus	7400	3×10^{-4}	0.2
Vesicular stomatitis virus	11000	1×10^{-4}	1.1
Influenza-A virus	14000	6×10^{-5}	0.8
Sendai virus	15000	3×10^{-5}	0.5
HIV-1 (AIDS virus)	10000	1×10^{-4}	1.0
Avian myeloblastosis virus	7000	5×10^{-5}	0.4
DNA			
Bacteriophage M13	6400	7×10^{-7}	4.6×10^{-3}
Bacteriophage γ	48500	8×10^{-8}	3.8×10^{-3}
Bacteriophage T4	166000	2×10^{-8}	3.3×10^{-3}
E. coli	4.7 mill	7×10^{-10}	3.3×10^{-3}
Yeast *(Saccharomyces cerevisine)*	13.8 mill	3×10^{-10}	3.8×10^{-3}
Neurospora crassa	41.9 mill	1×10^{-10}	4.2×10^{-3}
Human	3 billion	$\approx 10^{-12}$	$\approx 3 \times 10^{-3}$

Table 1: Error Rates and Genome Sizes of RNA Viruses as Compared to DNA-based Life Forms (modified from Eigen [11])

applicable to many other RNA viruses. All RNA viruses apparently can and do recombine, but the epidemiological significance of recombination is less clear. Recent studies of the nucleotide sequences of HIV strains from around the world have shown that recombination may be just as dominant an evolutionary force for this virus as it is for influenza [37, 38]. Non-human primate lentiviruses may contribute to the human HIV gene pool [25].

Existing RNA viruses only occupy a tiny fraction of the available RNA sequence space. Clearly a dominant constraint is the ability of the proteins encoded by the viral RNA to functionally interact with host cell constituents. However, there are probably several other structural (protein/protein, protein/RNA, etc) constraints on sequence space exploration.

One possible constraint is that certain genome organizations may be necessary to support evolvability through recombination or reassortment. Figure 2 shows several different genomic organization structures of different RNA virus families [4]. There are at least six major types of genomic organizations: diploid, non-segmented, pseudo-segmented, modular segmented, fully segmented, and partite. The evolutionary pathways that have given rise to these various genomic structures are not known. Equally important, it is unknown if these genomic organizations reflect particular "evolvability" adaptations by viruses to specific ecological niches.

Sequencing of viral genomes is now technically routine and viral nucleotide sequence data

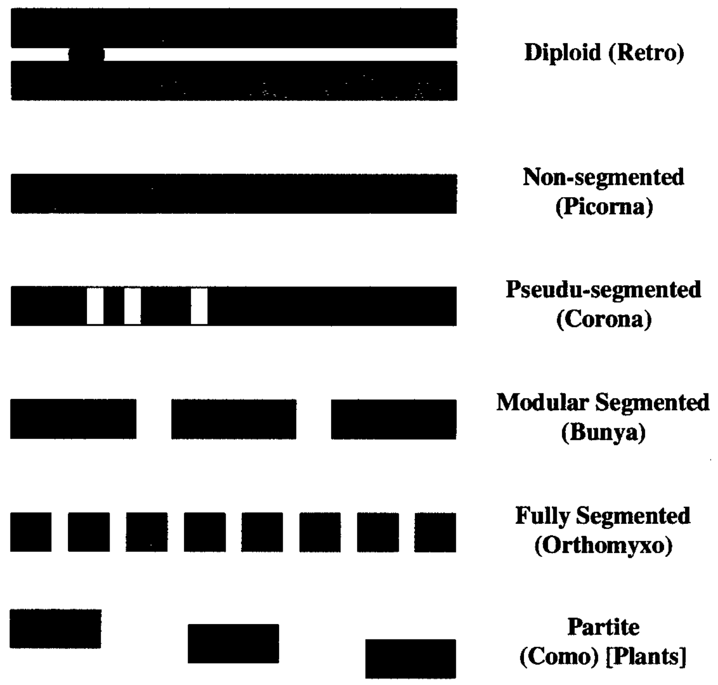

Diploid (Retro)
Non-segmented (Picorna)
Pseudu-segmented (Corona)
Modular Segmented (Bunya)
Fully Segmented (Orthomyxo)
Partite (Como) [Plants]

Figure 2: Viral Representationm Strategies

is accumulating at an exponential pace. We anticipate that it will be possible to use viral genomic sequence data to guide the construction of computational models of virus evolution that have predictive potential.

1.4.2 Genetic Algorithms

Genetic algorithms (GAs) are heuristic learning models based on principles drawn from natural evolution and selective breeding. Some features that distinguish genetic algorithms from other optimization methods are:

- A *population* of *structures* that can be interpreted as candidate solutions to the given problem;
- The *competitive selection* of structures for reproduction, based on each structure's fitness as a solution to the given problem;
- Idealized *genetic operators* that alter the selected structures in order to create new structures for further testing.

In many applications in optimization and search, these features enable the genetic algorithm to rapidly improve the average fitness of the population and to quickly identify the high performance regions of very complex search spaces. In practice, genetic algorithms may be combined with local search techniques to create a high-performance hybrid search algorithm.

```
procedure GA
begin
    initialize population;
    while termination condition not satisfied do
    begin
        select parents from population;
        create copies of selected parents;
        apply genetic operators to offspring;
        perform evaluations of offspring;
        insert offspring into population;
    end
end.
```

Figure 3: A Genetic Algorithm

A genetic algorithm simulates the dynamics of population genetics by maintaining a population of structures that evolves over time in response to the observed performance of its structures in their operational environment. A specific interpretation of each structure (e.g. as a collection of parameter settings, a condition/action rule, etc.) yields a point in the space of alternative solutions to the problem at hand, which can then be subjected to an *evaluation* process and assigned a measure called its *fitness*, reflecting its potential worth as a solution. The search proceeds by repeatedly selecting structures from the current knowledge base on the basis of fitness and applying idealized *genetic search operators* to these structures to produce new structures (*offspring*) for evaluation. The basic paradigm is shown in Figure 3. For more detailed discussions, see [14, 23].

Population $P(0)$ may be initialized using whatever knowledge is available about possible solutions. In the absence of such knowledge, the initial population should represent a random sample of the search space. When each structure in the population has been evaluated, a new population of structures is formed in two steps. First, structures in the current population are *selected* to be reproduced on the basis of their relative fitness. That is, highly fit structures may be replicated several times and poorly performing structures may not be replicated at all. In the absence of any other mechanisms, the resulting selective pressure would cause the best performing structures in the initial knowledge base to occupy a larger and larger proportion of the knowledge base over time.

Next the selected structures are altered using idealized *genetic operators* to form a new set of structures for evaluation. The primary genetic search operator is the *crossover* operator, which combines the features of two *parent* structures to form two similar *offspring*. There are many possible forms of crossover. The simplest version operates by swapping corresponding segments of a string or list representation of the parents. For example, if the parents are represented by the lists:

$$(a_1 a_2 a_3 a_4 a_5)$$

and

$$(b_1 b_2 b_3 b_4 b_5)$$

then crossover might produce the offspring

$$(a_1a_2b_3b_4b_5)$$

and

$$(b_1b_2a_3a_4a_5).$$

Other forms of crossover operators have been defined for other representations [18]. Specific decisions as to whether both resulting structures are to be entered into the knowledge base, whether the precursors are to be retained, and which other structures, if any, are to be purged define a range of alternative implementations.

A *mutation* operator, which alters one or more components of a selected structure, provides the means for introducing new information into the knowledge base. Again, a wide range of mutation operators have been implemented, ranging from completely random alterations to more heuristically motivated local search operators. In most cases, mutation serves as a secondary search operator that ensures the reachability of all points in the search space.

The search power of the genetic algorithm derives from the efficient exploitation of the wealth of information that the testing of structures provides with regards to the interactions among the components comprising these structures. Specific configurations of component values observed to contribute to good performance (e.g. a specific pair of parameter settings, a specific group of rule conditions, etc.) are preserved and propagated through the structures in the knowledge base in a highly parallel fashion. This, in turn, forms the basis for subsequent exploitation of larger and larger such configurations. Intuitively, we can view these structural configurations as the regularities in the space that emerge as individual structures are generated and tested. Once encountered, they serve as *building blocks* in the generation of new structures. That is, GAs actually search the space of all feature combinations, quickly identifying and exploiting combinations that are associated with high performance. The ability to perform such a search on the basis of the evaluation of completely specified candidate solutions is called the *implicit parallelism* of GAs. This leads to a focused exploration of the search space wherein attention is concentrated in regions that contain structures of above average utility. The knowledge base, nonetheless, is widely distributed over the space, insulating the search from susceptibility to stagnation at a local optima.

Although many genetic algorithm applications have been in the areas of function optimization, parameter tuning, scheduling and other combinatorial problems [8], genetic algorithms have also been applied to many traditional machine learning problems, including concept learning from examples, learning weights for neural nets, and learning rules for sequential decision problems. At NRL, we have investigated many aspects of genetic algorithms, ranging from the study of alternative selection policies [17] and crossover operators [10, 41], to performance studies of genetic algorithms for optimization in non-stationary environments [19]. Much of our effort has been devoted to the development of practical learning systems that use genetic algorithms to learn strategies for sequential decision problems [9, 16]. In our SAMUEL system [18], the "chromosome" of the genetic algorithm represents a set of condition-action rules for controlling an autonomous vehicle or a robot. The fitness of a rule set is measured by evaluating the performance of the resulting control strategy on a simulator. This system has successfully learned highly effective strategies for several tasks, including evading a predator, tracking a prey, seeking a goal while avoiding obstacles, and defending a goal from threatening agents. Experiments have shown that genetic algorithms provide an efficient way to learn strategies that take advantage of subtle regularities in the behavior of opposing agents. We are now beginning to investigate the more general case in which the behavior of the external agents changes over time. In particular, we are interested in learning competitive strategies against an opponent that is itself a learning agent. This is, of course, the usual situation in natural environments in which multiple species compete for survival. Our initial studies lead us to expect that genetic learning systems can successfully adapt to changing environmental conditions.

While the range of applications of genetic algorithms continues to grow more rapidly each year, the study of the theoretical foundations is still in an early stage. Holland's early work [23] showed that a simple form of genetic algorithm implicitly estimates the utility of a vast number of distinct subspaces, and allocates future trials accordingly. Specifically, let H be a *hyperplane* in the representation space. For example, if the structures are represented by six binary features, then the hyperplane denoted by H =0#1### consists of all structures in which the first feature is absent and the third feature is present. Holland showed that the expected number of samples (offspring) allocated to a hyperplane H at time $t+1$ is given by:

$$M(H,t+1) \geq M(H,t) * \frac{f(H,t)}{\overline{f}} * (1 - P_d(H))$$

In this expression, $M(H,t)$ is the expected proportion of the population in hyperplane H at time t, $f(H,t)$ is the average fitness of the current samples allocated to H, $\overline{f}$ is the average fitness of the current population, and $P_d(H)$ is the probability that the genetic operators will be "disruptive" in the sense that the children produced will not be members of the same subspace as their parents.[1] The usual interpretation of this result is that subspaces with consistently higher than average payoffs will be allocated exponentially more trials over time, while those subspaces with below average payoffs will be allocated exponentially fewer trials. This *implicit parallelism* can be shown to arise in any genetic algorithm that satisfies certain minimal conditions [17]. In addition, Ros [39] presents an initial PAC analysis of a class of genetic concept learners. There are many remaining opportunities for formal analysis of genetic algorithms. Some crucial open questions include:

- How quickly do genetic algorithms converge to an approximately optimal solution for various classes of problems? Behavior in the limit is known, but concrete convergence results are known only for trivial classes of problems and for the simplest forms of genetic algorithms.

- For which classes of problems does recombination (e.g., crossover) provide a measurable advantage over mutation alone?

[1]The effects of mutation are generally neglected in a first-order analysis. Considerable attention has been given to estimating the probability that a particular application of crossover will be disruptive [10].

- Given a fixed amount of computational resources, what are the optimal trade-offs among population size, number of generations, and (for probabilistic problems) evaluation accuracy?
- How much noise in the fitness functions can genetic algorithms tolerate?
- How well do genetic algorithms track non-stationary environments?
- What is the role of mating restrictions, e.g., mating with similar structures or among a spatially segregated sub-population, in promoting robust search and learning?

This report describes some pilot computational studies that shed light on these general issues for a genetic algorithm that has been modified to model the dynamic of viral infections.

1.4.3 Related Work

The evolutionary computation model described here includes a unique combination of several novel features:

- Variable length genomes
- Non-coding regions
- Ability to encode evolvability parameters such as mutation rate and crossover rate.

Some previous work in evolutionary computation has considered these features in isolation. For example, some recent work addresses the generalization to variable length structures, including [14, 18, 20, 24, 34, 47]. The role of non-coding genetic material has been addressed in [2, 21, 29, 31, 35, 42, 44, 45, 46]. Preliminary studies of the adaptation of genetic operators include [7, 30, 40].

The remainder of this report is organized as follows: Section 2 describes the materials and methods used in the project. This includes and description of the VIV virus modeling software and the VIS visualization tool. Section 3 presents a series of pilot computational studies that address the objectives listed above. Section 4 discusses the progress to date and outlines areas for future research. The Appendices provide more detailed descriptions of the software developed under this project.

2 Materials and Methods

2.1 Project Resources

This project was performed at the Navy Center for Applied Research in Artificial Intelligence, located at the Naval Research Laboratory in Washington, DC. NRL equipment used during the course of this project included five Sun Workstations running SunOS 4.2.1 and Solaris 2.4.1, both being versions of the Unix operating system. The VIV software system developed under this project was written in C and compiled using gcc (version 2.7.1). The VIV visualization tool was written in Java, using Java Developer's Kit (JDK) 1.0.2.

2.2 VIV: A Computational Model of Virus Evolution

This section gives a brief overview of the Virtual Virus (VIV), a model of virus evolution. This model has been developed to provide a computational framework for exploring fundamental questions concerning the evolved structure of the viral genotypes, the dynamics of cross-species infection, and the role of alternative recombination strategies exhibited by viruses.

VIV models one or more co-evolving populations of viruses. It is assumed that each population consists of N individual viral genomes. The population evolves over time by means of a genetic algorithm, shown in Figure 3. Each genome is evaluated by computing its fitness on a given fitness landscape representing a host species for the virus. Viruses reproduce based on their fitness, with the expected number of offspring based on the virus's fitness relative to the population mean fitness. Offspring undergo mutations, including point mutations, insertions and deletions, as well as recombination with other members of the virus population.

Rather than focusing on a particular virus-host model, VIV represents an abstract model of a class of viral fitness landscapes. We believe that an abstract approach is the best route to obtaining general results applicable to large classes of evolutionary systems. At the same time, the VIV model has been designed to reflect the most relevant information-processing relationships in the evolving system.

Any application of a genetic algorithm requires a specification of (at least) the following elements:

- the representation of genetic information,
- the definition of the fitness function,
- the algorithm for selecting parents for reproduction,
- the mutation operators, and
- the recombination operators.

In all of these elements, we have attempted to include the most relevent features of the biological system to the VIV simulation. As a result, we have extended the usual genetic algorithms in several novel directions. The differences between VIV and the standard genetic algorithm are described in the remainder of this section. The final subsection briefly describes the visualization tool for the VIV model (described in more detail in Appendix B).

2.2.1 Representation of Genetic Information

Because the VIV model is intended to explore issues concerning both the genomic sequence and the secondary structure of viruses, the model features a genotype-to-phenotype mapping that captures several important features occurring in biological systems:

- In nature, the phenotype is determined by the sequence of amino acids produced as a result of the transcription and translation of the genetic sequence. In the VIV model, we define the phenotype in terms of an analogous translation process.
- A four-letter genomic alphabet. While studies of genetic algorithms usually assume a binary alphabet, VIV adopts the standard four letter alphabet,

  $$G = \{\mathtt{A}, \mathtt{T}, \mathtt{C}, \mathtt{G}\}.$$

  That is, a genome in VIV consists of a sequence:

  AACGTTATA...CGCACTG

  While the binary alphabet is sufficient to encode any problem, the use of a four letter alphabet provides roughly the same degree of redundancy that occurs in nature in the mapping between the primary genetic sequence and the resulting phenotype.
- The natural genetic code is degenerate, meaning there may be more than genetic sequence that codes for the same amino acid. In VIV, the mapping from genotype to phenotype is also many-to-one. In particular, we adopt a mapping T from triplets over the genetic alphabet (i.e., *codons*) to a phenotypic alphabet A:

  $$T : G \times G \times G \longrightarrow A$$
- In nature, the phenotypic alphabet is defined over the 20 amino acids occuring in proteins, plus START and STOP codes. For the purpose of our abstract model, we adopt an analogous alphabet:

  $$A = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, ..., \mathtt{X}, \mathtt{Y}, \mathtt{Z}, \_, ., +\}.$$

  consisting of the 26 letters of the Roman alphabet along with the three punctuation marks underscore (_), period (.) and plus (+). The underscore represents the START codon and the period represents the STOP code. While this represents a slight increase in the size of the target alphabet A from 22 to 29, it still supports all of the above mentioned features of the natural genetic code.

codon	output	codon	output	codon	output	codon	output
TTT	A	CTT	I	ATT	Q	GTT	Y
TTC	A	CTC	I	ATC	Q	GTC	Y
TTA	B	CTA	J	ATA	R	GTA	Z
TTG	B	CTG	J	ATG	R	GTG	Z
TCT	C	CCT	K	ACT	S	GCT	A
TCC	C	CCC	K	ACC	S	GCC	E
TCA	D	CCA	L	ACA	T	GCA	I
TCG	D	CCG	L	ACG	T	GCG	O
TAT	E	CAT	M	AAT	U	GAT	U
TAC	E	CAC	M	AAC	U	GAC	Y
TAA	F	CAA	N	AAA	V	GAA	+
TAG	F	CAG	N	AAG	V	GAG	+
TGT	G	CGT	O	AGT	W	GGT	_
TGC	G	CGC	O	AGC	W	GGC	_
TGA	H	CGA	P	AGA	X	GGA	.
TGG	H	CGG	P	AGG	X	GGG	.

Figure 4: The VIV Artificial Genetic Code

As indicated above, the VIV model assumes an artificial genetic code. The motivation for this departure from biology is that the current state of knowledge does not permit an accurate model of the function of arbitrary proteins on the basis of their amino acid sequence. Therefore, it was necessary to make a set of simplifying assumptions that would permit the definition of a fitness landscape for our evolutionary model. Our approach has been to adopt an artificial genetic code that maps the genome to an output alphabet (in this case, the English alphabet), and to define a fitness landscape based on the resulting output strings. The particular artificial genetic code in VIV is shown in Figure 4.

As in the natural genetic code, this artificial code has the property that there is more information in the first two positions of each codon than in the third position. The codons for each consonant differ only in the third position. Two of the three codons for the other symbols also have the first two positions in common. This is meant to reflect the similar pattern of information content in the genetic code, where the third position of a codon contains less information than the first two positions. In addition, not all output symbols have the same number of codons. Each consonant has two codons, each vowel (`A`, `E`, `I`, `O`, `U` and `Y`) has three codons, and the punctuation marks each have two codons, giving a total of 64 codons.

The mapping T can be applied to a genetic sequence to obtain a string over the alphabet A. In VIV, as in nature, three output strings over A can be derived from a given genome sequence by starting in any of the first three initial positions. That is, VIV models the three reading frames occuring in natural genomes.

2.2.2 The Fitness Function

In nature, the environmental fitness landscape for a virus is defined by the set of environmental conditions in the host species to which the virus must adapt in order to reproduce. As in the modeling of protein function, the current state of knowledge does not permit an accurate model of the environment in which viruses operate. Therefore, it is necessary to make a set of simplifying assumptions that would permit the definition of a fitness landscape for our evolutionary model.

The fitness landscape in VIV is defined by a set of *target phenotypes*, or words over the output alphabet A. That is, the target phenotype specifies a phenotype that is highly fit in terms of its ability to infect a given host species. The fitness of a given virus is determined by a *compatibility index* that measures the similarity between the viral phenotype and the target phenotype. Alternative fitness landscapes associated with different species can be specified by alternative target phenotypes. For example, the target phenotype for a given viral environment might be specified as the set:

$$\{\texttt{COREPROTEIN} + \texttt{ABC}, \texttt{POLYMERASE} + \texttt{ABC}, \texttt{ENVELOPE} + \texttt{ABC}\}$$

where the phenotype `COREPROTEIN+ABC` represents the ability of the expressed gene to yield the core protein for the virus in the environment of the `ABC` species.

The compatibility index (or it raw fitness) computation is as follows:

1. All coding regions in the viral genome are identified, and all product strings are computed (i.e., transcribed).
2. For each coding region, compute the compatibility score with respect to each given target term by comparing the spelling of the product string with the spelling of the target term.
3. For each target term, the compatibility score of the target term is the highest compatibility score of any coding region with respect to this term.
4. The compatibility index of the genome is then a weighted average of the compatibility scores of all the terms in the target phenotype.

2.2.3 Selection

Selection is the process of choosing individuals for reproduction in an evolutionary algorithm. One popular form of selection is called *proportional selection*, which involves creating a number of offspring in proportion to an individual's fitness. This approach was proposed and analyzed by Holland [23], and has been used widely in many implementations of evolutionary algorithms. Proportional selection provides a natural counterpart to the usual practice in population genetics of defining an individual's fitness in terms of its number of offspring.

The selection process is implemented in three distinct steps, namely,

1. Map the objective function to fitness.
2. Create a probability distribution proportional to fitness.
3. Draw samples from this distribution.

The evaluation process of individuals in an evolutionary algorithm begins with the user-defined *objective function*,

$$f : A_x \rightarrow \Re$$

which typically measures some cost to be minimized or some reward to be maximized. The definition of the objective function is, of course, application dependent. In general, the objective function should provide enough information to drive the selective pressure of the evolutionary algorithm. For example, "needle-in-a-haystack" functions, i.e., functions that assign nearly equal value to every candidate solution except the optimum, should be avoided. The objective function in VIV is the compatibility score, described above.

The *fitness function*

$$\Phi : A_x \rightarrow \Re_+$$

maps the raw scores of the objective function to a non-negative interval. The fitness function is often a composition of the objective function and a scaling function g:

$$\Phi(x) = g(f(x))$$

As an evolutionary algorithm progresses, the population often becomes dominated by high-performance individuals with a narrow range of objective values. In this case, the fitness functions described above tend to assign similar fitness values to all members of the population, leading to a loss in the selective pressure toward the better individuals. To address this problem, *fitness scaling* methods that accentuate small differences in objective values are often used in order to maintain a productive level of selective pressure. VIV uses *Sigma scaling* [14]), defined as follows:

$$\Phi(a_i(t)) = \begin{cases} f(a_i(t)) - (\overline{f}(t) - c * \sigma_f(t)) & \text{if } f(a_i(t)) > (\overline{f}(t) - c * \sigma_f(t)) \\ 0 & \text{otherwise} \end{cases}$$

where $\overline{f}(t)$ is the mean objective value of the current population, $\sigma_f(t)$ is the (sample) standard deviation of the objective values in the current population, and c is a constant, say $c = 2$. The idea is that $\overline{f}(t) - c * \sigma_f(t)$ represents the least acceptable objective value for any reproducing individual. As the population improves, this statistic tracks the improvement, yielding a level of selective pressure that is sensitive to the spread of performance values in the population.

Once the fitness values are assigned, the next step in proportional selection is to create a probability distribution such that the probability of selecting a given individual for reproduction is proportional to the individual's fitness. That is,

$$Pr_{prop}(i) = \frac{\Phi(i)}{\sum_{i=1}^{\mu} \Phi(i)}.$$

VIV employs a so-called generational GA, in which the entire population is replaced during each generation, so the above probability distribution is sampled μ times. Baker [3] developed an algorithm called *stochastic universal sampling* (SUS) that exhibits less variance than repeated calls to the roulette-wheel algorithm [14].

2.2.4 Mutation Operators

In the traditional genetic algorithm, mutation is usually implemented as a probabilistic operator that randomly alters individual bits within a binary genome. Length altering mutations are usually not considered, although there are exceptions [18]. In contrast, VIV contains several biologically motivated forms of mutation, including:

- Random substitutions for individual bases,
- Deletions,
- Repetitions,
- Inversions.

Currently, preliminary forms of these operators are included in the model. For the purpose of the pilot studies below, only point mutations (random substitutions on individual bases) have been included in studies to date.

2.2.5 Recombination

In traditional genetic algorithms, the members of the population have a fixed length, and recombination in a GA works as follows:

- Given two individuals of the population, pick a point (or multiple points) randomly.
- Line up the individuals and perform the crossover(s) at these points.

For example, given parents **AATTGCACGGG** and **TCGCCCGCTAA** and a crossover point of 5,

```
AATTGCACGGG
    ^
TCGCCCGCTAA
    ^
```

the offspring produced would be:

```
AATTCCGCTAA and TCGCGCACGGG.
----+++++++     ++++-------
```

(- indicates base element from parent 1 and + indicates base element from parent 2).

So-called two-point crossover is similar in that two individuals of the same length would be lined up but two points would be chosen as crossover points. The parents **AATTGCACGGG** and **TCGCCCGCTAA**, when crossed at positions 5 and 8 (chosen randomly),

```
AATTGCACGGG
    ^  ^
TCGCCCGCTAA
    ^  ^
```

would produce the offspring

```
AATTCCGCGGG and TCGCGCACTAA.
----+++----     ++++---++++
```

In VIV, a new crossover operator called *homologous 1-point crossover* has been designed which crosses two variable length parents in regions of similarity. This type of crossover was inspired by biological mechanisms. The homology matching algorithm works as follows:

1. Given two individuals, called parent1 and parent2, pick a point at random on parent1.
2. Use window of bases around the selected point on parent1 to match up against the same size window on parent2, running through each possible starting positions for the window on parent2.
3. Records the window on parent2 giving the best match against parent1.
4. Based on the best match score, decide whether to perform crossover between the parents.

If the match score is less than 50%, the parents are not recombined. Otherwise, the probability of crossover increases from 0 (for a match score of 50%) to 1 (for a complete match). If crossover occurs, a point is picked randomly within the aligned window of the parents, dividing each parent into two segments. One offspring comprises the first segment of parent1 and the second segment of parent2, and the second offspring comprises the first segment of parent2 and the second segment of parent1. Finally, if either offspring violates the maximum-length constraint, then the crossover is aborted.

For example, given the following individuals:

```
Parent1: ATTTCGCTCAGGTAAATGCGCG

Parent2: GGGTTTCGATTTCATGGTAGCAAAAATTAG
```

Suppose we center a window of size 6 at position 4:

```
Parent2: GGGTTTCGATTTCATGGTAGCAAAAATTAG
            *
```

Looking for the best match on parent1, we find that the window beginning with the second base element matches best. We line up the parents according to the match.

```
Parent1:   ATTTCGCTCAGGTAAATGCGCG
            *    *
Parent2: GGGTTTCGATTTCATGGTAGCAAAAATTAG
            *    *
```

Now randomly pick a crossover point within this window, say 2.

```
Parent1:   ATTTCGCTCAGGTAAATGCGCG
            *^   *
Parent2: GGGTTTCGATTTCATGGTAGCAAAAATTAG
            *^   *
```

and perform the crossover:

```
Offspring1: ATTTCGATTTCATGGTAGCAAAAATTAG
            --++++++++++++++++++++++++++

Offspring2: GGGTTTCGCTCAGGTAAATGCGCG
            ++++--------------------
```

(- indicates base element from parent 1 and + indicates base element from parent 2).

2.2.6 Multiple-Population Infection

The VIV fitness model supports the investigation of cross-species infection. As shown in Figure 5, any number of interacting populations may be defined. Each population represents the virus population infecting a specific species. Each species can have a distinct fitness landscape, specified by the target phenotypes for that species. The model permits interactions among the multiple evolving populations through a virus transfer mechanism. Briefly, at periodic intervals, individual viruses may be transferred between the evolving populations. This permits the study of the effects of viral recombination in a multiple species system.

2.2.7 VIV Visualization Tool

To facilitate the analysis of data produced by the computer simulations, we have developed a visualization tool, called VIS. The two main objectives of the visualization are:

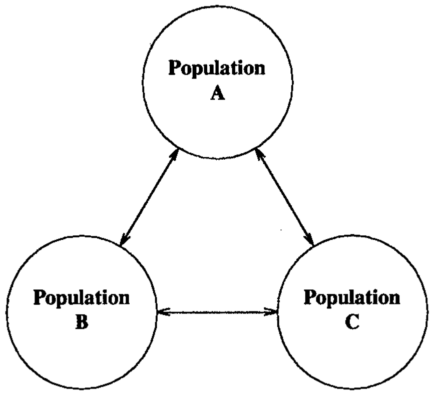


Figure 5: A Multi-population Model

- Easy access to desired information and easy transitions between related pieces of information.
- The development of novel and alternative methods for displaying multi-dimensional data in a coherent and informative manner.

In particular, VIS assists the user in:

- Observing the effects of individual genetic operators.
- Monitoring the average or best performance (fitness of the members of the population) throughout a run.
- Tracking the discovery, growth and sharing of partial solutions.
- Comparing selection pressure, operator effectiveness, and other characteristics at various time during a GA run.
- Tracing the ancestry or progeny of an individual.

The VIS program allows the user to "navigate" through a VIV run (both forwards and backwards), focus on "snapshots" of any instant of a run, and observe individual events from the run. These capabilities are expected to allow the user to examine the causes and effects of individual events and to follow specific trails of information exchange in a particular evolution, ultimately leading to a better understanding of the process of viral evolution.

The VIS program is written in the Java programming language and is available as either an applet or an application. When executed as an applet, VIS is only able to access input files from the NRL server. When executed as an application, VIS should be able to read any correctly formatted input files. A detailed user's guide is provided in the Appendix.

3 Pilot Computational Studies

This section describes some of the pilot computational studies performed with VIV. The first three subsections below describe an initial set of studies that test the internal consistency of the model and to compare it with previous evolutionary computation models. In these initial studies, a single control parameter, e.g., mutation rate, crossover rate, or length bias, was systematically altered over a pre-defined set of values. There was a single population of viruses evolving in a fixed environment. Besides indicating the effect of these parameters on the model, these studies also provide a baseline against which to compare later studies, which will concern multi-population infections. An example of a multi-population study is included as the final section below.

Unless otherwise noted, all the studies below used the following set of model parameters (explained in Appendix A):

- Population size = 500
- Generations = 2000
- Mutation rate = 0.003
- Crossover rate = 1.0
- Initial genome lengths = [100, 500]
- Maximum genome length = 7500
- Crossover operator = 1 point homologous crossover
- Mutation operator = random base substitution
- Number of evolving populations = 1

3.1 Effects of Mutation Rate in a Single Population

The goal of this study is to investigate the effect of different fixed mutation rates in a fixed, single-species environment. The target phenotype was {`COREPROTEIN, POLYMERASE, ENVELOPE`}. Some interesting questions to address are:

1. Are higher or lower mutation rates more useful in a single population for this domain?
2. Are mutation rates correlated with length of the individuals?
3. Is crossover necessary to produce better results for all mutation rates tested?

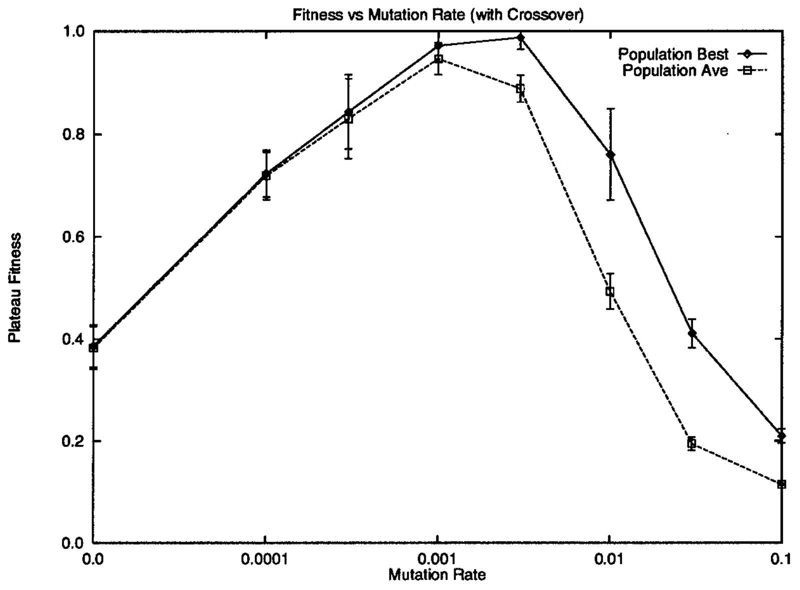


Figure 6: Effects of Mutation Rates (with Crossover enabled)

3.1.1 Experimental Design

Mutation rates for all individuals were fixed over the entire run at one of the following values:

$$\{0.0, 0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1\}.$$

The mutation rate gives the probability that a random substitution mutation will occur at each base element of the individual. For each mutation rate, 10 independent runs were performed using different seeds for the random number generators. Two sets of experiments were run using these mutation rates, one with a crossover rate of 1.0 and one with a crossover rate of 0.0 (no crossover).

3.1.2 Computational Results

The *best plateau fitness* refers to the fitness of the best individual in the final population, and the *average plateau fitness* refers to the average fitness in the final population. Both measures were averaged over 10 independent runs of the program. Figure 6 shows the best and average plateau fitness obtained with each mutation rate with a crossover rate of 1.0. (Error bars indicate one standard deviation over the 10 runs.) Figures 7 shows the best and average plateau fitness obtained with each mutation rate with a crossover rate of 0.0.

Regarding the best individuals of each run, the best results were obtained using a mutation rate of 0.003. Generally, the average length of the individuals in the final population was about

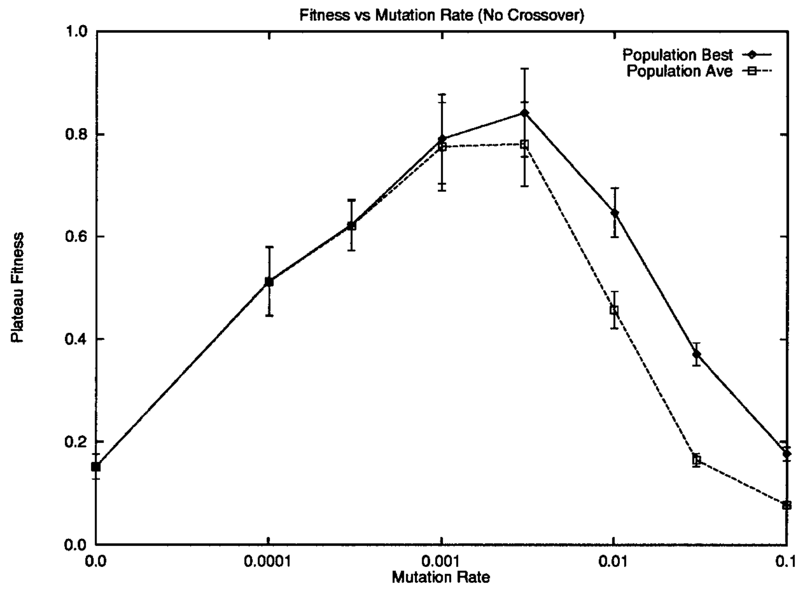


Figure 7: Effects of Mutation Rates (with Crossover disabled)

500 nucleotides. Thus the mutation rate of 0.003 yields on average about 1.3 mutations per individual during each generation. This is fairly consistent with biological observations of one mutation per generation per individual.

The best-individual performance increases consistently as the mutation rate increases from 0.0 to 0.003. Higher mutation rates yield steadily worsening performance. Removing crossover leads to decreased fitness with all rates of mutation tested.

Figure 8 shows the effects of the mutation rate on genome length. It is interesting that as the mutation rate increases beyond 0.001, the plateau genome length also increases. An explanation for this phenomenon is not obvious. Assuming that the coding regions are roughly the same size, there would not seem to be any clear evolutionary advantage in longer genomes, since the extra mutations are mainly occurring in non-coding regions. This data suggests that the plateau genome length evolves in response to the base mutation rate. Further investigations are required to identify the source of the selective pressure toward longer genomes. One conjecture is that the non-coding regions may be used to some advantage as redundant coding regions.

3.2 Effects of Recombination in a Single Population

The goal of this study is to investigate the effect of different fixed recombination rates in a fixed, single-species environment.

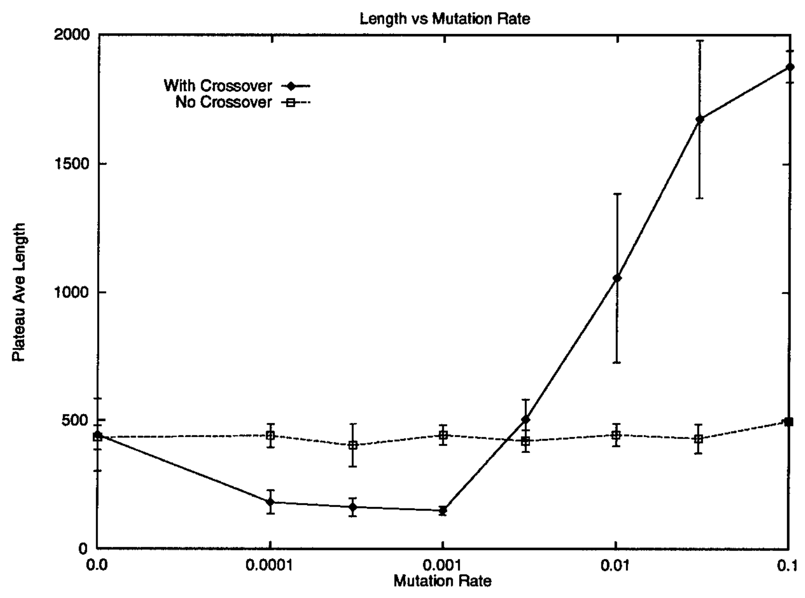


Figure 8: Effects of Mutation Rates on Genome Length

3.2.1 Experimental Design

Experiments were conducted to determine the best overall crossover rate for this problem domain. The type of crossover performed was 1-point homologous crossover. Crossover rates of 0.0, 0.1, 0.3, 0.6 and 1.0 were used. A rate of 1.0 means that 100% of the population will have the potential to be crossed each generation, depending on how well the crossover regions match each other. Other parameters were set to default values as described above. For each crossover rate, 10 runs were performed.

3.2.2 Computational Results

Figure 9 shows the best and average plateau fitness obtained with each crossover rate. All of the results are averaged over 10 runs.

With respect to the best individual, the crossover rate of 1.0 gave the best performance, and further, each performance curve indicated better results as the crossover rate increased from 0.0 to 1.0.

With respect to the average fitness of individuals, the crossover rate of 0.6 and 0.3 produced slightly better results than a crossover rate of 1.0. There appears to be no significant difference resulting from crossover rates of 0.3, 0.6 or 1.0, but having at least some crossover appears to be necessary to produce the best results.

It should be noted that the nominal crossover rate, shown here, differs from the effective

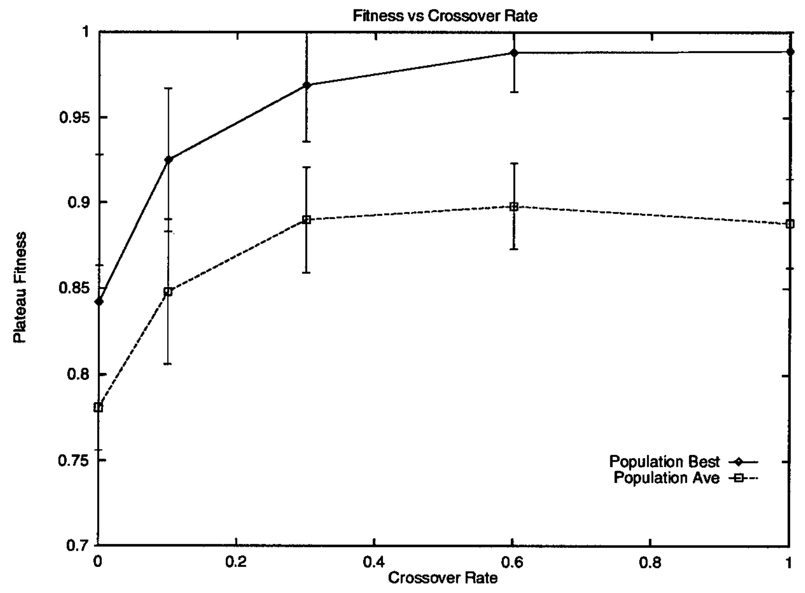


Figure 9: Effects of Recombination Rates

crossover rate. The nominal crossover rate determines the probability that crossover is attempted between two parents. When there is insufficient similarity between the two parents, no crossover occurs. Thus the effective crossover rate may be less than the nominal rate.

3.3 Effects of Length Bias in a Single Population

This study investigates the effect of different fixed length biases in a fixed, single-species environment.

3.3.1 Experimental Design

The *maximum genome length* determines the attenuation of the fitness. The fitness declines linearly based on length until it reaches 0 at the maximum genome length. Experiments were conducted to determine the effect of changing the maximum genome length for this problem domain. The maximum genome lengths were fixed over the entire run at one of the following values:

$$\{1000, 2500, 5000, 7500, 22500, 25000\}.$$

For each value for the maximum genome length, 10 runs were performed. All other parameters were set to default values as described above.

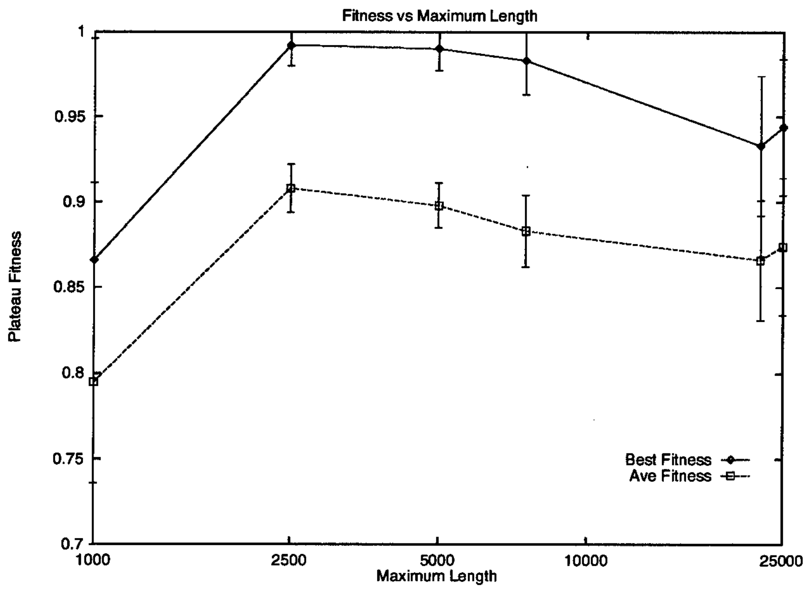


Figure 10: Effects of Length Bias

3.3.2 Computational Results

Figure 10 shows the best and average plateau fitness obtained with each value for the maximum genome length. All of the results are averaged over 10 runs.

A maximum genome length of 2500 to 5000 appears to give the best results. Performance declined both for values lower than 2500 and greater than 5000. A maximum length of 1000 is too probably severe, leading to premature convergence of the population to sub-optimal solutions. Maximum lengths of 7500 and higher are not restrictive enough, and performance begins to degrade there as well.

3.4 A Study of Mutation Rates in Multi-population Infections

This study was aimed at understanding the evolution of evolvability, i.e., the ability of viruses to rapidly adapt to new environments. In this study, three populations of viruses co-evolved. The fitness of individuals with each population was measured against a specific target phenotype. The target phenotypes were as follows:

$$\{\mathtt{COREPROTEIN}+\mathtt{HUMAN}, \mathtt{POLYMERASE}+\mathtt{HUMAN}, \mathtt{ENVELOPE}+\mathtt{HUMAN}\}$$

$$\{\mathtt{COREPROTEIN}+\mathtt{CHIMP}, \mathtt{POLYMERASE}+\mathtt{CHIMP}, \mathtt{ENVELOPE}+\mathtt{CHIMP}\}$$

$$\{\mathtt{COREPROTEIN}+\mathtt{MONKEY}, \mathtt{POLYMERASE}+\mathtt{MONKEY}, \mathtt{ENVELOPE}+\mathtt{MONKEY}\}$$

Note that the target phenotypes have strong similarities, as well as significant differences. As a result, a virus that has evolved in any one of these environment will be relatively highly fit in any other (compared to random genomes).

During the evolutionary runs, viruses transferred among the evolving populations as follows:

1. At intervals of 10 generations, 20 viruses were selected at random from each population. Ten of the selected viruses were transferred to each of the other two populations, replacing randomly selected viruses from the destination population.
2. The transferred viruses were evaluated for fitness within the receiving population.
3. Finally, the transferred viruses were selected for reproduction based on their fitness, and recombined with native viruses.

This procedure allowed cross-fertilization between the evolving population.

Two questions of immediate interest are:

1. Are higher mutation rates more useful in the multi-population scenario than in the single-population scenario?
2. Given the opportunity, will higher mutation rates evolve in multi-population scenarios?

A series of experiments were run in which the mutation rate was systematically varied over a range of values, in order to identify the most favorable value for the multi-population scenario.

3.4.1 Experimental Design

VIV was run in multi-population mode, with three populations. The interpopulation transfer rate was 10 individuals transferred to each of the other two populations every 10 generations. The population size was 500 individuals per population. Each run comprised 2000 generations. Mutation rates were:

$$\{0.0, 0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1\}$$

For each mutation rate, 5 runs were performed.

3.4.2 Computational Results

The results are shown in Figure 11. All of the results are averaged over 5 runs. According to this pilot study the mutation rate that provides the best multi-population infection does not significantly differ from the best mutation rate for the single-population case. Further study is required in order to identify factors that may influence the overall mutation rate in multi-population scenarios.

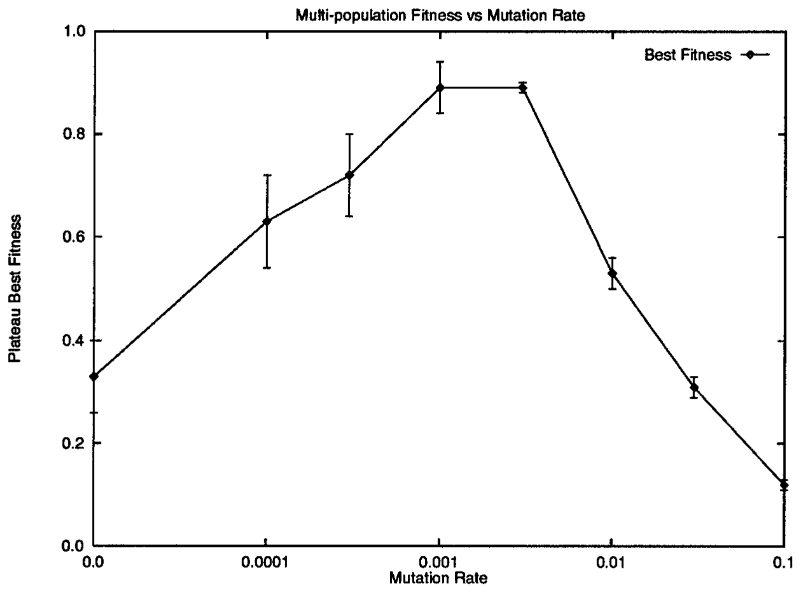


Figure 11: Effects of Mutation Rate in Multi-host Environment

4 Conclusions

The objectives of this project were to understand emerging virus epidemics by focusing on methods of viral evolution, and to develop and evaluate novel evolutionary modeling approaches. A computational model of virus evolution has been developed that enables the study of fundamental questions concerning the evolved structure of the viral genotypes, the dynamics of cross-species infection, and the role of alternative recombination strategies exhibited by viruses. The VIV software system applies the methods of evolutionary algorithms to the problem of modeling biological systems, extending the optimization-based genetic algorithm in the direction of a more plausible model of biological evolution. Pilot computational studies on a series of questions related to the evolvability of viruses have been performed. Further effort with the model is now required in order to address specific hypotheses about the evolution of emerging disease.

4.1 Recommendations

During this pilot project, we have developed valuable computational tools for the study of viral evolution and emerging infections. However, extensive computational experiments with the model were beyond the scope of this project. Many promising directions for further research suggest themselves, including:

- Evolvability selected as an adaptation to differing fitness landscapes, including:
  - Single peak vs. multi-peak landscapes,
  - Smooth vs. rugged landscapes, and

  - Fixed vs. changing landscapes.

  The results would provide insights into the effects of different mutation rates, for example, on viruses whose host population was itself shifting or evolving in response to the virus. All of these classes of landscapes can be investigated by varying the target phenotypes during the run.

- Effects of secondary structure in fitness, mutation and recombination. Genome secondary structure, e.g., the presence of stems and loops, plays a role in both mutation and recombination in nature. Modeling the mutation and recombination operations that depend on secondary structure may lead to further understanding of the role of non-coding regions as regulatory mechanisms in viral evolution. An initial implementation of secondary structure was implemented as part of this project, but much remains to be done.

- Effects of mating restrictions and speciation. Our initial model includes no mating restrictions within each viral population. It would be reasonable to model mating restrictions based on sequence similarity or secondary structure. Mating restrictions are likely to play a key role in the development of separate genome organization, such as shown in Figure 2.

- Evolvability of mutation rate, processivity and segmentation. Only preliminary computer studies of the evolvability of mutation rates were possible within this pilot studies.

This project has achieved its goal of developing a novel evolutionary modeling approach to the study of emerging infections. A computational model of virus evolution has been developed that enables the study of fundamental questions concerning the evolved structure of the viral genotypes, the dynamics of cross-species infection, and the role of alternative recombination strategies exhibited by viruses. The pilot computational studies illustrate just a few of the possible ways to use the model to investigate issues related to the evolution of viruses. For example, the results shown in Figure 8 suggest that the genome length may evolve in response to the base mutation rate. The VIV computational model will enable the study of such questions in a highly observable simulation environment. As the model is refined, such studies can be expected to provide additional insight into issues concerning viral evolution.

The VIV model provides the computational infrastructure for an extended examination of these issues and their relationship to the evolution of emerging disease. We look forward to pursuing these issues in future projects.

References

[1] R. Ahmen, C.S. Hahn, T. Somasundarem, L. Villarete, M. Matloubian, and J. H. Strauss, 1991. Molecular basis of organ-specific selection of viral variants using chronic infection. *J. of Virology* **65**:4242-4247.

[2] D. Andre and A. Teller, 1996. A study in program response and the negative effects of introns in genetic programming. *Proc. of the 1st Annual Conference on Genetic Programming*, 12-20.

[3] J. E. Baker, 1987. Reducing bias and inefficiency in the selection algorithm. *Proceedings of the Second International Conference on Genetic Algorithms*, ed. J Grefenstette, 14-21. Hillsdale, NJ: Lawrence Erlbaum Associates.

[4] D. S. Burke, 1997. Recombination in HIV: An important viral evolution strategy. *Emerging Infectious Diseases* **3**:253-259.

[4] D. S. Burke, 1997. Personal communication.

[5] D. S. Burke and F. E. McCutchan, 1996. Global distribution of human immunodeficiency virus-1 clades. In (Devita, V.T., Jr, S. Hellman, S. A. Rosenberg, editors) *AIDS: Bilogy, Diagnosis, Treatment and Prevention*, Fourth Edition. Lippincott-Raven Publishers.

[6] D. D. Clarke, E. A. Duarte, A. Moya, S. F. Elene, E. Domingo and J. Holland, 1993. Bottlenecks and population passages cause profound fitness differences in RNA viruses. *J. Virology* **67**:222-228.

[7] L. D. Davis, 1989. Adapting operator probabilities in genetic algorithms. *Proceeding of the Third International Conference on Genetic Algorithms.*

[8] L. Davis (Ed.), 1991. *Handbook of Genetic Algorithms*, Van Nostrand Reinhold.

[9] K. A. De Jong, 1990. Genetic-algorithm-based learning. In *Machine Learning: An artificial intelligence approach, Vol. 3,* Y. Kodratoff and R. Michalski (Eds.), Morgan Kaufmann.

[10] K. A. De Jong and W. M. Spears, 1992. A formal analysis of the role of multi-point crossover in genetic algorithms. *Annals of Mathematics and Artificial Intelligence 5(1)*, 1-26.

[11] M. Eigen, 1993. The origin of genetic information: Viruses as models, *Gene* **135**:37-47.

[12] J. Felsenstein, 1974. The evolutionary advantage of recombination. *Genetics* **78**:737-56.

[13] J. Felsenstein, 1976. The evolutionary advantage of recombination, II: Individual selection for recombination. *Genetics* **83**:845-59.

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

[15] D. Goldberg, K. Deb and B. Korb, 1989. Messy genetic algorithms: motivation, analysis, and first results, *Complex Systems, 3,* 493-530.

[16] J. J. Grefenstette, 1988. Credit assignment in rule discovery systems based on genetic algorithms. *Machine Learning 3(2/3),* 225-245.

[17] J. J. Grefenstette, 1991. Conditions for implicit parallelism. In *Foundations of Genetic Algorithms,* G. Rawlins (Ed.), Morgan Kaufmann.

[18] J. J. Grefenstette, C. L. Ramsey and A. C. Schultz, 1990. Learning sequential decision rules using simulation models and competition. *Machine Learning 5(4)*, 355-381.

[19] J. J. Grefenstette, 1992. Genetic algorithms for changing environments. *Proceedings of Parallel Problem Solving from Nature-2*, R. Maenner and B. Manderick (Eds.), North-Holland, 137-144.

[20] I. Harvey, 1992. The SAGA cross: the mechanics of crossover for variable-length genetic algorithms. in *Parallel Problem Solving from Nature, 2*, 269-278.

[21] T. Haynes, 1996. Duplication of coding segments in genetic programming. *Proc. of the 13th National Conference on Artificial Intelligence*, 344-349.

[22] J. Holland, K. Spindler, F. Horodyski, E. Grabau, S. Nichol and S. VandePol, 1982. Rapid evolution of RNA genomes. *Science* **215:**1577-85.

[23] J. H. Holland, 1975. *Adaptation in natural and artificial systems.* Ann Arbor: University of Michigan Press.

[24] H. Iba, H. deGaris, and T. Sato, 1994. Genetic programming using a minimum description length principle, in *Advances in Genetic Programming*, 265-284.

[25] M. J. Jin, H. Hui, D. L. Robertson, 1994. Mosaic genome structure of simian immunodeficiency virus from west African green monkeys. EMBOJ **13:**2935-47.

[26] S. A. Kauffman, 1993. *The Origins of Order.* New York: Oxford University Press. 114-17.

[27] A. S. Kondrashov, 1984. Deleterious mutations as an evolutionary factor. I. The advantage of recombination. Genet Res (Camb) **44:**199-217.

[28] F. M. LaForce, K. L. Nichol, N. J. Cox, 1994. Influenza: Virology, epidemiology, disease and prevention. Am J Prev med **10:**31-44.A

[29] James R. Levenick, 1991. Inserting introns improves genetic algorithm success rate: Taking a clue from biology, *Proc. of the 4th International Conference on Genetic Algorithms*, 123-127.

[30] J. R. Levenick, 1995. Metabits: generic endogenous crossover control. *Sixth International Conference on Genetic Algorithms*, 88-95.

[31] R. K. Lindsay and A. S. Wu, 1996. Testing the robustness of the genetic algorithm on the floating building block representation, *Proc. of the 13th National Conference on Artificial Intelligence.*

[32] S. Morse and A. Schluederberg, 1990. Emerging viruses: The evolution of viruses and viral diseases. JID **162:**1-7.

[33] F. A. Murphy, C. M. Fauquet, D.H. L. Bishop, S. A. Ghabrial, A. W. Jarvis, G. P. Martelli, M. A. Mayo and M. D. Summers (eds.), 1995. *Virus Taxonomy: Classification and Nomenclature of Viruses: Sixth Report of the International Committee on Taxonomy of Viruses.* Springer-Verlag, New York, 586 pp.

[34] P. Nordin and W. Banzhaf, 1995. Complexity compression and evolution, Proc. of the 6th International Conference on Genetic Algorithms, 310-317.

[35] P. Nordin, F. Francone, and W. Banzhaf, 1996. Explicitly defined introns and destructive crossover in genetic programming, in *Advances in Genetic Programming 2*, Chpt. 6, 111-134.

[36] W. B. Provine, 1986. *Sewall Wright and Evolutionary Biology.* Univ of Chicago Press, Chicago, pp. 307-317.

[37] D. L. Robertson, B. H. Hahn, P. M. Sharp, 1995. Recombination in AIDS viruses. J. Mol Evol **40:**249-59.

[38] D. L. Robertson, P. M. Sharp, F. E. McCutchan, B. H. Hahn, 1995. Recombination in HIV-1 (letter). Nature **374:**124-26.

[39] J. Ros, 1993. Learning Boolean functions with genetic algorithms: A PAC analysis. In *Foundations of Genetic Algorithms-2*, D. Whitley (Ed.), Morgan Kaufmann.

[40] J. D. Schaffer and A. Morishima, 1987. An adaptive crossover distribution mechanism for genetic algorithms. *Proceeding of the Second International Conference on Genetic Algorithms*, 36-40.

[41] W. M. Spears and K. A. De Jong, 1991. An analysis of multi-point crossover. In *Foundations of Genetic Algorithms*, G. Rawlins (Ed.), Morgan Kaufmann.

[42] M. Wineberg and F. Oppacher, 1996. The benefits of computing with introns, *Proc. of the 1st Annual Conference on Genetic Programming*, 410-415.

[43] S. Wright, 1932. The roles of mutation, inbreeding, crossbreeding, and selection evolution. Proceedings of the Sixth International Congress on Genetics **1:**356-66.

[44] A. S. Wu, 1995. *Non-coding segments and floating building blocks for the genetic algorithm.* Dissertation, University of Michigan.

[45] A. S. Wu and R. K. Lindsay, 1995. Empirical studies of the genetic algorithm with non-coding segments, *Evolutionary Computation, 3:2.*

[46] A. S. Wu and R. K. Lindsay, 1996. A comparison of the fixed and floating building block representation in the genetic algorithm. *Evolutionary Computation 4:2.*

[47] B. T. Zhang and H. Muhlenbein, 1995. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation, 3:1*, 17-35.

A The VIV Software System

A.1 Running VIV

VIV is executed via a UNIX command line as follows:

```
% viv &
```

All output is directed to various output files, described below. After completion of the program, the VIS visualization tool can be used to navigate through the data produced during the run. The remainder of this Appendix describes the input and output files for the VIV program.

A.2 Input Files

Several runtime parameters allow the user to control how experiments are done, how data is saved and display, and how various components operate. Runtime parameters are specified in two files: **params.def** and **params**. At startup, the program first reads the **params.def** file, setting the runtime variables according to the default values in that file. Then it reads the **params** files, overwriting any values specified. Each line in the parameter files has the form:

parameter = value

Lines that begin with a pound-sign (#) are comments and are ignored.

The following paragraphs define each parameter, giving its range of values and its default values. For each parameter, the default value is shown as the (first) value after the equal sign. For parameters with string values shown below, only one of the listed strings will be accepted (except for filename parameters). Generally, there is little or no error checking on numeric parameters.

`popsize = 500`
Number of genomes per population. Program variable: **Popsize**.

`gens = 2000`
Max number of generations performed. Program variable: **Maxgens**.

`best_interval = 10`
Generations between printing out the best genome. Program variable: **Best_interval**.

`max_length = 7500`
Maximum length of a genome. Program variable: **Length**.

`transfer_npops = 1`
Number of population undergoing evolution. Program variable: **Transfer_npops**.

`min_init_length = 100`
: Minimum length for randomly generated genomes in the initial population Program variable: **Min_init_length**.

`max_init_length = 500`
: Maximum length for randomly generated genomes in the initial population Program variable: **Min_init_length**.

`m_rate = 0.003`
: The probability that a base undergoes a random substitution during each reproductive event. Program variable: **Mu_rate**.

`c_rate = 1.0`
: The probability that a individual participates in crossover during each reproductive event. Program variable: **C_rate**.

`cross_type = hom_1pt`
: Crossover operator. Default is one-point homologous crossover. Program variable: **Cross_type**.

`seed = 1`
: Random number seed. Program variable: **Seed**.

`vis = 1`
: If set, produce output required by the VIS visualization tool. (See Appendix B). Program variable: **Vis**.

A.3 Output Files

VIV normally creates several output files, described in the following subsections. The following descriptions follow the default behavior of VIV. However, the names of these files and the frequency of updating them can also be controlled by runtime parameters.

A.3.1 LOG File

The **log** file records the start and finishes time for each run of VIV, for example:

```
VIV started on sun19.aic.nrl.navy.mil
Thu Jun 26 09:41:30 1997
```

```
VIV finished
Thu Jun 26 11:42:46 1997
```

A.3.2 OUT File

The **out** file normally contains one line of statistics per generation. A partial **out** file is shown below:

```
Gen  Trials    Best  Online Offline     Ave     Std    Base   Ext   MaxL  MinL  AveL  StdL  AveM
  0     500  0.1304  0.0468  0.1224  0.0468  0.0257  0.0000     0    499   100   307   113  0.003
 10    5500  0.2057  0.0877  0.1712  0.1197  0.0312  0.0269     1    647    94   403    96  0.003
 20   10500  0.2944  0.1221  0.2192  0.2005  0.0448  0.0643     3    746   183   452    98  0.003
 30   15500  0.3329  0.1563  0.2531  0.2424  0.0392  0.1228     2    753   228   439   105  0.003
 40   20500  0.3564  0.1834  0.2782  0.2835  0.0372  0.1693     7    748   224   437    98  0.003
 50   25500  0.4117  0.2068  0.3004  0.3195  0.0396  0.2094     8    861   253   490   111  0.003
```

The fields in the **out** file are:

`Gen`
: Generation counter.

`Trials`
: Trial counter (number of evaluations performed).

`Best`
: The best value of any genome in the current population.

`Online`
: Online performance. This is the running average of the best-so-far values.

`Offline`
: Offline performance. This is the average value of all genomes.

`Ave`
: The mean value in the current population.

`Std`
: The standard deviation of values in the current population.

`Base`

The baseline value used to measure fitness. The fitness of a genome is its raw value minus the baseline.

`Ext`

The number of exterminated genomes (in the previous generation). A genome is exterminated if its value is below the baseline value. Exterminated genomes have no offspring.

`MaxL, MinL, AveL, StdL`

Maximum, minimum, mean and standard deviations of the lengths for genomes in current population.

`AveM`

The average mutation rate (per base) in current population.

A.3.3 RAW File

The **raw** file normally contains one line per generation, showing the best, average and standard deviation of the raw fitness scores in the current population. A partial **raw** file is shown below:

```
 Gen  Trials    Best    Ave    Std
   0     500 0.1366 0.0490 0.0272
  10    5500 0.2212 0.1266 0.0332
  20   10500 0.3130 0.2134 0.0475
  30   15500 0.3525 0.2576 0.0416
  40   20500 0.3823 0.3011 0.0399
  50   25500 0.4486 0.3420 0.0440
```

The first field is the generation counter. The second field is the trial counter for the current generation.

A.3.4 BEST File

VIV normally produces one **best** file containing the best genome in each generation. A partial **best** file is shown below:

```
=====================================

Generation 1990  Trial 995150  Raw Fitness 1.000  Fitness 0.985  Length 109  M_rate 0.0030

genotype: AAGGCTCTCGTATAGCCCGGATAGCGACGTATGCACAGGGGTCGGCGTCCAGTCCACGCCATGGCTACCGCCGGGCGGGC
          GCCCAAAAATATCCGCGTCGATATGGGCT

frame 0: VAIZFK.FPORMX_POLYMERASE.POKVUQOYUR_
frame 1: XIDEWLUWYZGT.Y_YNCTLHJLL_.OLVRCODRHA
frame 2: _COREPROTEIN.DOCWLOM_EOPO_ENVELOPE.

GAG in frame 2: _COREPROTEIN.docwlom_eopo_envelope.
POL in frame 0: vaizfk.fpormx_POLYMERASE.pokvuqoyur_
ENV in frame 2: _coreprotein.docwlom_eopo_ENVELOPE.

=====================================

Generation 2000  Trial 1000292  Raw Fitness 1.000  Fitness 0.985  Length 110  M_rate 0.0030

genotype: AAGGTTCTCGCATAGCCCGAATAGCGACGTATGCACAGGGGTCGACGCCCAGTTCACGCCATAGCTACCGCCGGAGGCGGT
          GCCCAGAAGTATCCGCGTCGATATGGGCT

frame 0: VYIIFK+FPORMX_POLYMERASE.__ENVELOPE.
frame 1: XADMWLUWYZGT.YYENATLFJLL+OZKXWQOYUR_
frame 2: _COREPROTEIN.DTKWDOMWEOPXPGL+ZCODRHA

GAG in frame 2: _COREPROTEIN.dtkwdomweopxpgl+zcodrha
POL in frame 0: vyiifk+fpormx_POLYMERASE.__envelope.
ENV in frame 0: vyiifk+fpormx_polymerase.__ENVELOPE.

=====================================
```

B The VIS Visualization Tool

B.1 Introduction

To fully understand the implications of the VIV model, it is necessary to examine the individual events and interactions that occur during a VIV run as well as evaluate the outcome of a run. We are developing a visualization tool (VIS) in conjunction with the VIV model to facilitate the analysis of GA data. The two main goals of the VIS project are:

- The design of tool capabilities that allow easy access to desired information and easy transitions between related pieces of information.
- The development of novel and alternative methods for displaying multi-dimensional data in a coherent and informative manner.

The VIS program is meant to be a tool with which one can easily "navigate" through a GA run (both forwards and backwards), focus on "snapshots" of any instant of a run, and observe individual events from the run. These capabilities are expected to allow the user to examine the causes and effects of individual events and to follow specific trails of information exchange in a VIV run, ultimately leading to a better understanding of the simulated virus evolution.

The VIS program is written in the Java programming language and is available as either an applet or an application. When executed as an applet, VIS is only able to access input files provided on the NRL server. When executed as an application, VIS should be able to read any correctly formatted input files.

To use VIS to examine a VIV run, turn on the `vis` flag in the VIV parameters. This action causes VIV to print all run data to a set of files, placed in a single directory. This directory is specified as the input directory when starting a VIS session. VIS is then able to access data as necessary based on user interactions and requests.

Though the VIS program capabilities and displays are currently focused on the VIV project, much of the functionality is easily generalizable to be useful for general GA data.

B.2 Getting Started

B.2.1 Starting the applet

To start the VIS applet, go to `http://www.aic.nrl.navy.mil/ aswu/vis/vistool2/vis2.html`. Enter the name of the directory containing the data files from the desired run and click on the "Enter" button.

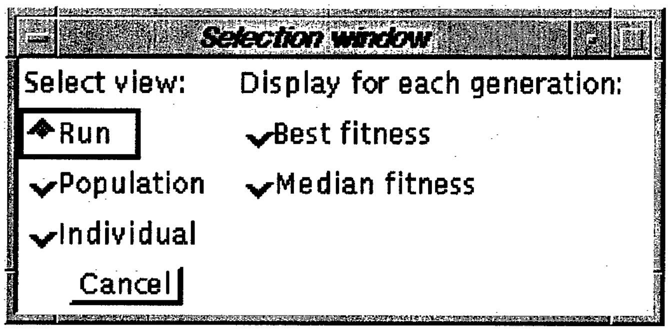
B.2.2 Starting the application

To start the VIS application, enter the following command: `java Vis2 &`. When the **Vis2** window appears, enter the name of the directory containing the data files from the desired run and click on the "Enter input directory" button.

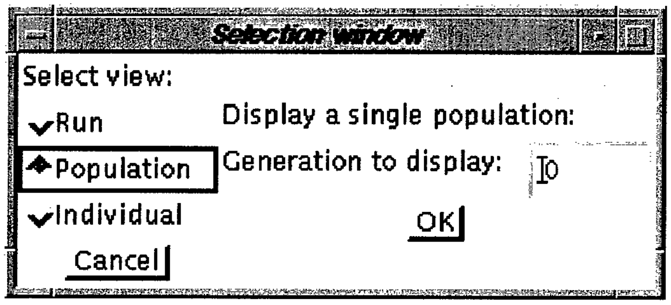
B.2.3 Opening a display window

Each time an input directory is entered, VIS will display a selection window. The selection window prompts the user to open one or more display windows. The three types of display windows available: **Run**, **Population**, and **Individual**; they are described in section B.5.

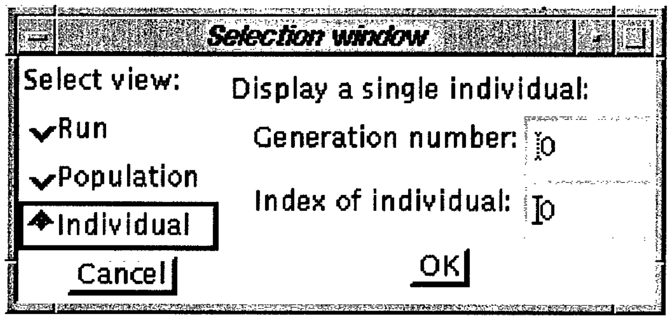
To open a **Run** window, the user must select the information to be displayed for each generation of the run. Current choices are the best individual and the median individual.



To open a **Population** window, the user must specify the generation to be displayed.



To open an **Individual** window, the user must specify the generation and index of the individual to be displayed.



All display windows opened from a particular selection window (or its progeny display windows) will display data from the same input directory. The user may open multiple selection windows to display data from more than one input directory.

B.3 Terminating a session

To terminate the VIS application, simply select the **Quit** button from the **Vis2** window. VIS will close all display windows and terminate the program.

To close a specific display window, select the **Close** option from the **Window** menu. This action closes the specified window, but does not affect any other windows.

B.4 Input files

VIS expects multiple data files for each GA run. All files should be placed in a single directory. This directory is the input directory that is specified when starting the program.

Each run has a file called `vis_params`. This file stores the number of generations in the run, the size of the population, and the genotype format. The VIS program currently supports two genotype formats. Binary format individuals have genotypes consisting of only two bases, represented by zero and one. ACGT format individuals have genotypes consisting of four bases, represented by the bits ranging from zero to three.

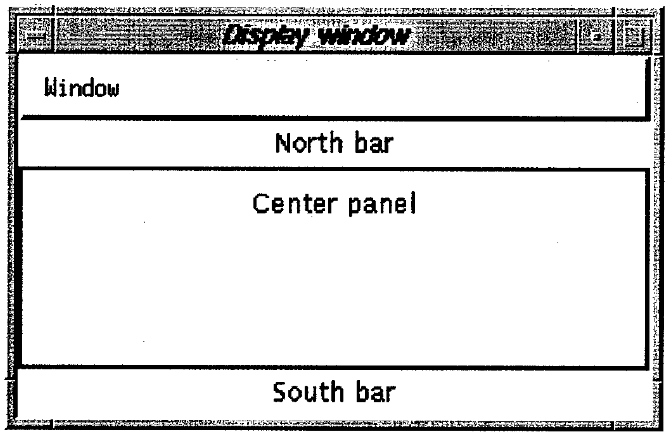
For each generation of the run, four additional files are expected: `vis_generation.`n, `vis_best.`n, `vis_median.`n, and `vis_cross.`n, where n is the generation number. The file `vis_generation.`n stores all of the data from generation n. This data includes general statistics from that generation as well as complete information about each individual in the generation. The file `vis_best.`n stores the information about the individual with the highest fitness in generation n. The file `vis_median.`n stores the information about the individual with the median fitness of generation n. The file `vis_cross.`n stores the crossover points of all individuals in which crossover occurred. Generation 0 indicates the initial population.

B.5 Display windows

VIS displays data in one of three types of windows: **Run**, **Population**, or **Individual**. These windows may be opened from the selection box that appears after in input directory is entered, or they may be opened by clicking on select display items in the display windows themselves.

B.5.1 Window organization

Display windows are laid out in three sections: south bar, center panel, and north bar.



South bar

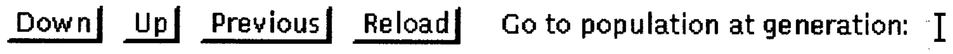
The south bar contains a description of what the window is currently displaying. For example, an **Individual** window displaying the fifth individual from the fourth generation would display "Generation 4, Individual 5" in the south bar.

Center panel

The center panel displays the primary data of the window. The contents of the center panel for each type of window will be described in the sections below on the specific types of display windows.

North bar

The north bar contains buttons for navigating among the individuals of a generation or among the generations of a run. Users may use these buttons to change the individual or generation that is displayed in the window. Out-of-range index values are ignored. The north bar currently does not appear in **Run** windows.



- The **Down** button causes the window to display the individual/generation with the next lowest index from the current individual/generation.

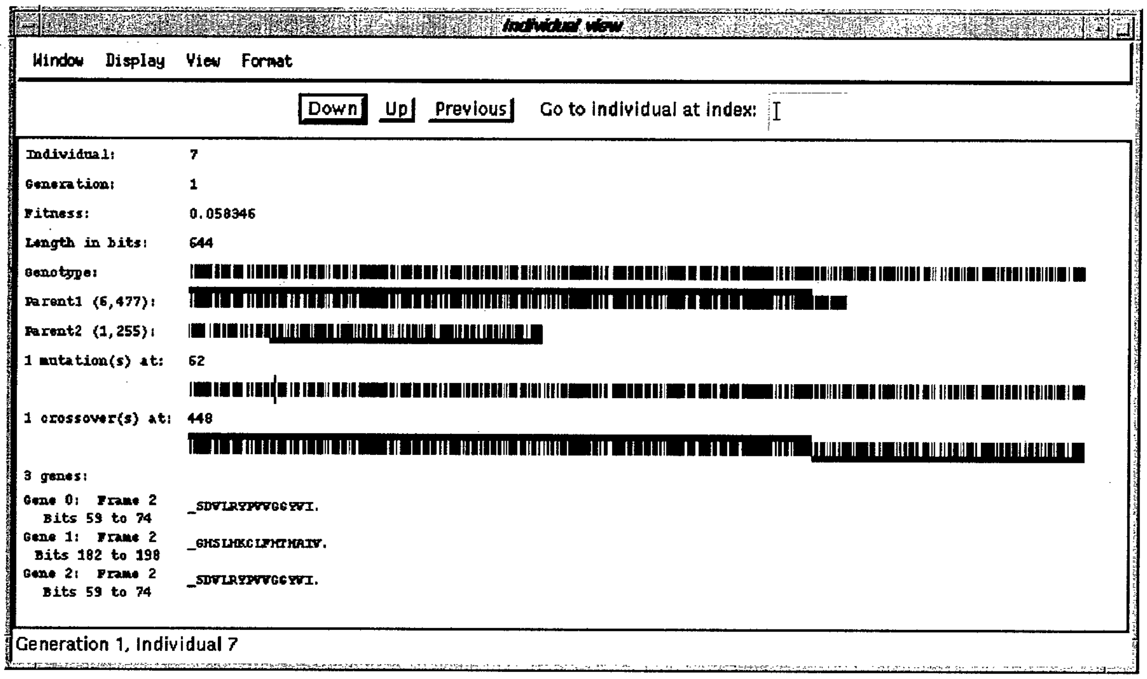
- The **Up** button causes the window to display the individual/generation with the next highest index from the current individual/generation.
- The **Previous** button causes the window to display the individual that was displayed previous to the currently displayed individual.
- The **Reload** button re-loads and re-displays the current generation.
- The **Go to** field allows the user to select any individual/generation (within range) to be displayed.

B.6 Individual window

The **Individual** window displays the data for a single individual. The **Display** menu (section B.9.2) allows the user to select one of two center panel displays.

B.6.1 Data

Displays general data for an individual.


Individual view
Window Display View Format
Down Up Previous Go to individual at index:

```
Individual:           7
Generation:           1
Fitness:              0.058346
Length in bits:       644
Genotype:
Parent1 (6,477):
Parent2 (1,255):
1 mutation(s) at:     62

1 crossover(s) at:    448

3 genes:
Gene 0:  Frame 2      _SDVLRYPVVGGYWI.
  Bits 59 to 74
Gene 1:  Frame 2      _GHSLHKCLFMTMAIV.
 Bits 182 to 198
Gene 2:  Frame 2      _SDVLRYPVVGGYWI.
  Bits 59 to 74
```

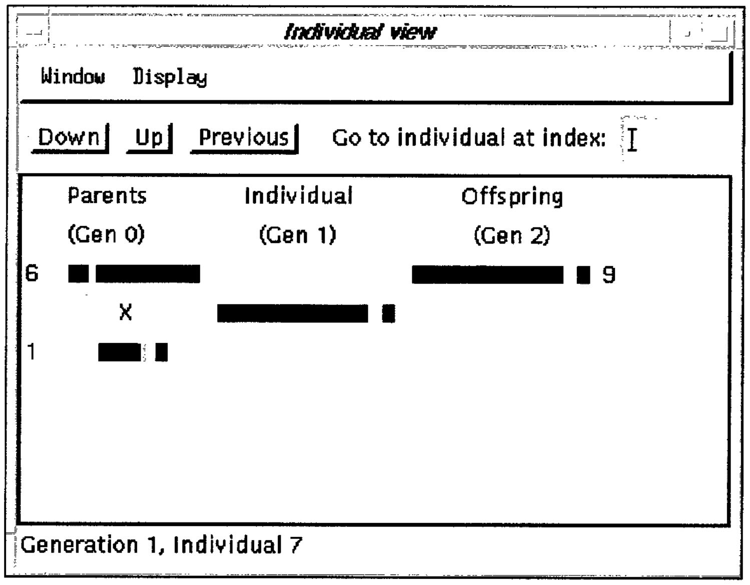
Generation 1, Individual 7

- The index of the individual. Each individual is arbitrarily assigned an index number to distinguish it from other individuals.
- The generation to which the individual belongs.
- The fitness of the individual.

- The length of the individual in bits.
- The genotype of the individual. Genotype formats are described in section B.9.3. The **Color coded** format is shown above.
- The genotype, index, and length of the individual's parents. Users may click on a parent genotype to open a new **Individual** window for that parent.
- The mutations, if any, involved in creating this individual. Mutation locations listed and are marked in color.
- The crossover points, if any, involved in creating this individual. If crossover occurred, the portion that each parent contributed to the individual is indicated in color. If crossover did not occur, the individual was cloned from `Parent1`.
- The genes and their locations and reading frames on this individual. Users may select to display only the genes or the genes in relation to the rest of their reading frame.

B.6.2 Family

Displays the current individual, its parents, and its offspring in **Gene location** format. Users may select parents or offspring to open a new **Individual** window displaying the **Family** of the selected individual.

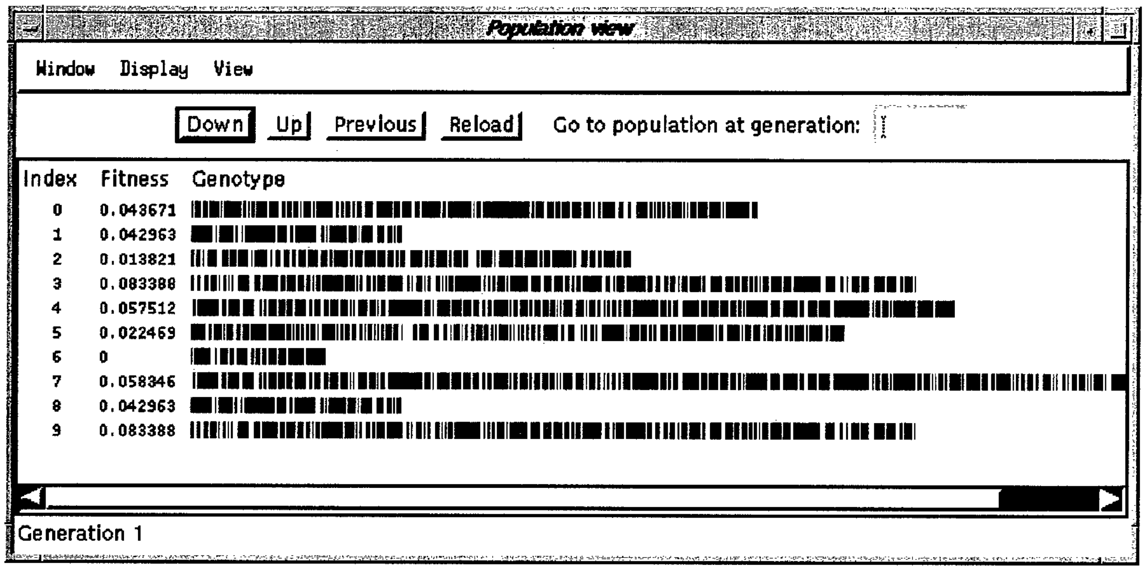


B.7 Population window

The **Population** window displays data about a single population. The **Display** menu (section B.9.2) allows the user to select one of three center panel displays.

B.7.1 Individuals

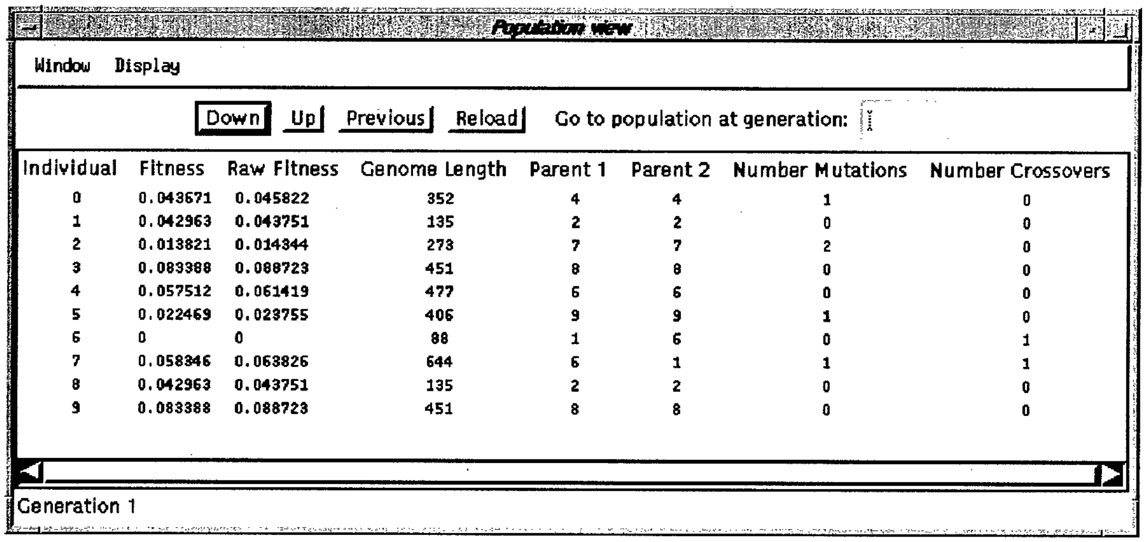
Displays each individual of the population along with its fitness and index. The default format by which individuals are displayed is by bit value; the examples below show the **Color coded** format. The format may be changed with the **View** menu (section B.9.3). Users may click on an individual to bring up an **Individual** window for the selected individual.



Index	Fitness	Genotype
0	0.043671	
1	0.042963	
2	0.013821	
3	0.083388	
4	0.057512	
5	0.022469	
6	0	
7	0.058346	
8	0.042963	
9	0.083388	

B.7.2 Statistics

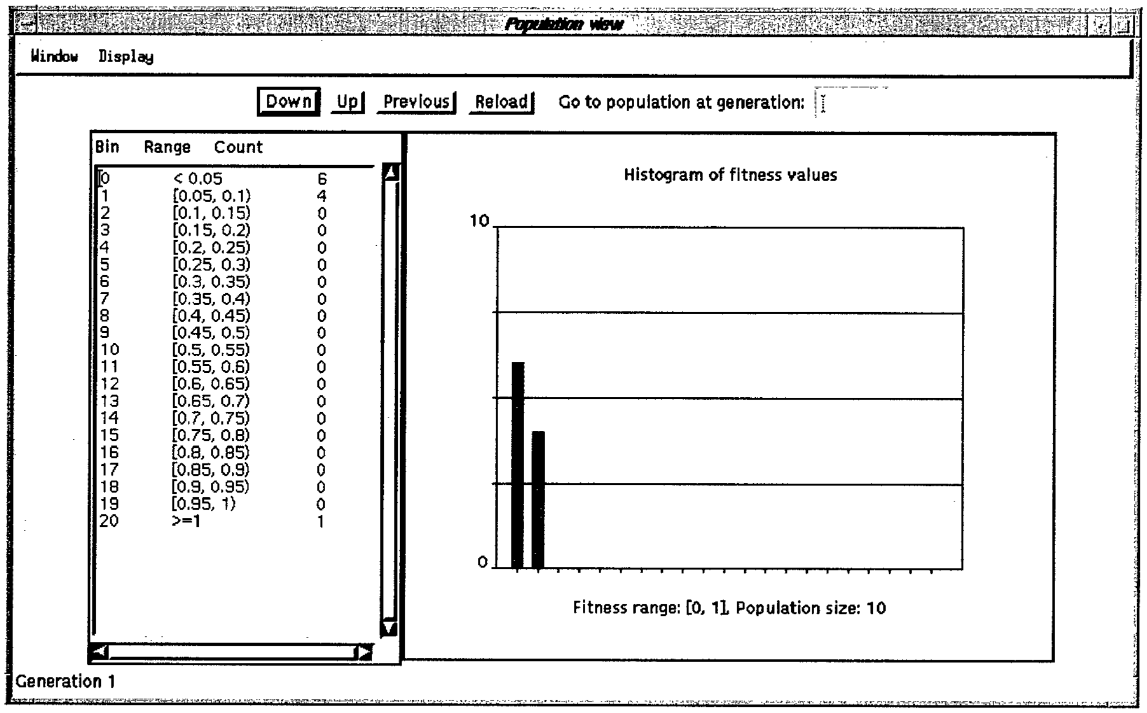
Displays the statistics for each individual of the population. Users may click on the index value an individual to bring up an **Individual** window for the selected individual.



Individual	Fitness	Raw Fitness	Genome Length	Parent 1	Parent 2	Number Mutations	Number Crossovers
0	0.043671	0.045822	352	4	4	1	0
1	0.042963	0.043751	135	2	2	0	0
2	0.013821	0.014344	273	7	7	2	0
3	0.083388	0.088723	451	8	8	0	0
4	0.057512	0.061419	477	6	6	0	0
5	0.022469	0.023755	406	9	9	1	0
6	0	0	88	1	6	0	1
7	0.058346	0.063826	644	6	1	1	1
8	0.042963	0.043751	135	2	2	0	0
9	0.083388	0.088723	451	8	8	0	0

B.7.3 Histogram

Displays a histogram of the fitnesses of the individuals in the population.

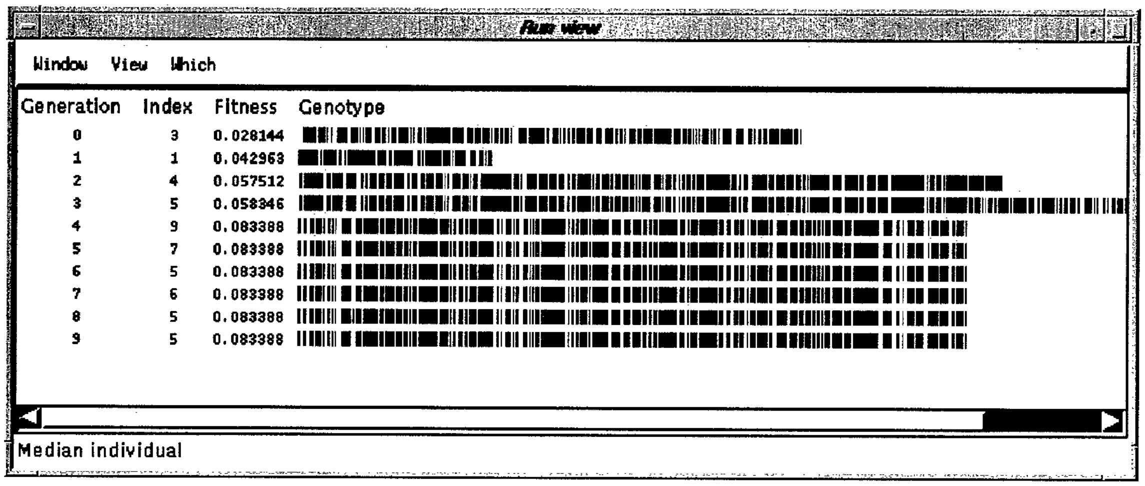

Population view
Window Display
Down Up Previous Reload
Go to population at generation:
Histogram of fitness values
10
0
Fitness range: [0, 1], Population size: 10

Bin	Range	Count
0	< 0.05	6
1	[0.05, 0.1)	4
2	[0.1, 0.15)	0
3	[0.15, 0.2)	0
4	[0.2, 0.25)	0
5	[0.25, 0.3)	0
6	[0.3, 0.35)	0
7	[0.35, 0.4)	0
8	[0.4, 0.45)	0
9	[0.45, 0.5)	0
10	[0.5, 0.55)	0
11	[0.55, 0.6)	0
12	[0.6, 0.65)	0
13	[0.65, 0.7)	0
14	[0.7, 0.75)	0
15	[0.75, 0.8)	0
16	[0.8, 0.85)	0
17	[0.85, 0.9)	0
18	[0.9, 0.95)	0
19	[0.95, 1)	0
20	>=1	1

Generation 1

B.8 Run window

The **Run** window displays data over the entire run. Current capabilities include displaying the best and median individuals for each generation of the run. Users may click on an individual to open an **Individual** window for that individual or click on a generation number to open a **Population** window for that generation.



Generation	Index	Fitness	Genotype
0	3	0.028144	
1	1	0.042963	
2	4	0.057512	
3	5	0.058346	
4	9	0.083388	
5	7	0.083388	
6	5	0.083388	
7	6	0.083388	
8	5	0.083388	
9	5	0.083388	

Median individual

B.9 Menu options

B.9.1 Window

The **Window** menu provides general window management commands. Currently the only option available here is **Close** which closes the window.

B.9.2 Display

The **Display** menu allows the user to select what is displayed in a window. The options of this menu differ for the different windows.

Individual window (See section B.6 for further detail.)

- The **Data** option displays general data for the individual.
- The **Family** option displays the current individual, its parents, and its offspring in **Gene location** format.

Population window (See section B.7 for further detail.)

- The **Individuals** option displays the individuals of the population.
- The **Statistics** option displays the statistics for the individuals of the population.
- The **Histogram** option displays a histogram of the fitnesses of the individuals of the population.

B.9.3 View

The **View** menu allows the user to select the format with which to display the genotypes of individuals.

Genotype

```
323030301130222022110012331230100112230233023032130302300012013030030023130000030012222O
```

Displays each individual as a string of bit values. For binary runs, the bit values will be either zero or one. For ACGT runs, the bit values will range from zero to three.

Zebra (Binary only)

Displays each individual as a series of black and white stripes. A black stripe indicates a zero bit; a white stripe indicates a one bit.

Neopolitan (Binary only)

Displays each individual as a series of four different colored stripes. Each stripe represents two bits. The following color coding scheme is used: 00 = black, 11 = white, 01 = magenta, 10 = orange.

Color coded (ACGT only)

Displays each individual as a series of four different colored stripes. Each stripe represents the value of a single bit. The following color coding scheme is used: 0 = blue, 1 = red, 2 = yellow, 3 = green.

Gene locations

Displays each individual with the locations of the genes indicated in color. The following color coding scheme is used: red = core protein, yellow = polymerase, blue = envelope. Overlapping genes are indicated with overlapping colors, i.e. red + blue = purple.

B.9.4 Format

The **Format** menu only appears with the **Individual** window. It allows the user to select how the gene information is displayed in the window. With the "long view", the entire reading from of each gene is displayed. Gene amino acids are displayed in upper case; all other amino acids are displayed in lower case. With the "short view", only the gene amino acids are displayed.

B.9.5 Which

The **Which** menu only appears with the **Run** window. It allows the user to select what to display for each generation. Current choices are either the best or median individual of the generation.

C Project Support

C.1 Acknowledgements

This project was supported by the Walter Reed Army Institute of Research (WRAIR) and by the Office of Naval Research.

C.2 Project Personnel

In addition to the PI, Dr. John Grefenstette, other NRL personnel supported in part by this project included:

- Kenneth De Jong, Ph.D.
- Connie Loggia Ramsey, M.S.
- William Spears, M.S.

Other NRL research personnel contributed to this project through participation in technical discussions at NRL, including Dr. Robert Daley, Dr. Ralph Hartley and especially Dr. Annie Wu, who was responsible for the VIS visualization tool. Dr. Wu was supported by a Post Doctoral Fellowship from the National Research Council.

The project team gratefully acknowledges the many stimulating discussions with Dr. Donald S. Burke of WRAIR.

D Project Presentations

D. S. Burke, "An Evolutionary Biology Approach to Emerging Diseases", Conference on Research Requirements in the Operations of DNA Technology in Infectious Diseases, Uniformed Services University of the Health Sciences, Bethesda, Maryland, 17 January 1997.

D. S. Burke, J. Grefenstette, A. Wu, C. Ramsey, and K. De Jong. "VIV: A Virtual Virus that is an Evolutionary Computation Model of Biological Viruses", Plenary Lecture, International Workshop on Molecular Epidemiology and Evolutionary Genetics of Pathogenic Microorganisms, Montpellier, France, 26 May 1997.

D. S. Burke. "Evolution of Viruses and Emerging Infectious Diseases", American Red Cross Seminar Series, Holland Laboratory, Rockville, Maryland, 25 June 1997.

J. Grefenstette. "VIV: A Virtual Virus Model", Colloquium at the Institute for Computational Sciences and Informatics, George Mason University, June 1997.

A. Wu. "VIV: The Virtual Virus Project", Workshop on Evolutionary Computation with Variable Size Representation, at the 7th International Conference on Genetic Algorithms, July, 1997.

A. Wu. "Non-coding DNA in biological systems", Workshop on Exploring Non-coding Segments and Genetics-based Encodings, at the 7th International Conference on Genetic Algorithms, July, 1997.