A Proposal: Evolution of Digital Organisms

Thomas S. Ray
University of Delaware
School of Life & Health Sciences
Newark, Delaware 19716
302-831-2281 (FAX)
302-831-2753
ray@udel.edu

Table of Contents

This is the text of a grant proposal (October 1992). It is made available because it is the best document available at this time which clearly lays out my vision of where the Tierra research project is headed, and specifically details how I plan to achieve my goals.

Project Summary

This project involves inoculating the process of evolution into an artificial medium. Self-replicating machine code algorithms ("creatures") are run on computers in which mutations occur in the form of bit flips. The result is evolution by natural selection. The project encompasses both biological and computational goals: To study the *process* of evolution in detail, through analysis of sequences of events and genetic changes. To observe the envelopes of evolutionary possibilities, both under identical and very different conditions. To understand what factors affect the shapes of phylogenetic trees. To understand what are the elements of evolvability in genetic languages. To understand what elements are required for an evolving system to spontaneously increase in diversity and complexity, as in the transition from single celled to multi-cellular life. To experiment with the control of evolution of machine code algorithms, for the purpose of breeding programs to do useful work. This work will lead to a greater understanding of the process of evolution, and to the possible forms that life may take on throughout the universe. In addition, it may provide a means (through evolution) of developing the software of the future, particularly software for massively parallel computers.

Results from Prior NSF Support

CCR-9204339, May 15, 1992 through October 31, 1993. "Computer Architectures for the Natural Evolution of Machine Code".

This project concerns development of the system for the evolution of machine codes, particularly with respect to testing new instruction sets to find a good balance of compute power and evolvability, and to explore the possibility of evolving software for parallel architectures.

Substantial effort was invested in increasing the usability, flexibility and portability of the software (known as Tierra, Spanish for Earth). In addition, several new observational tools have been developed, and old ones improved. A frontend was added to allow the user to control the software while it runs, providing facilities for altering variables during a run, and for observing various properties of the populations (e.g., frequency distributions of genotypes) of evolving programs during the run. It is now possible to bring the system down and then up again in the middle of a run without loss of information. It is also possible to fully recover from a hardware crash, and continue the run without having to start over. The memory protection options used to create the "cell membranes" around creatures now allows any combination of read, write or execute protections. Tierra now runs on most DOS machines, and on most unix machines. Tierra is supported on Sun, DEC, Silicon Graphics, Next, IBM RS6000. It has also run on Cray, the IBM 3090, and the CM5.

It is now possible to easily remap the associations of numeric opcodes to instructions. Doing so has the effect of altering the physics into which evolution will be inoculated. Specifically, it changes the connectivity through mutation. All mutations are single bit flips, thus mutation can convert each five bit instruction into only five of the thirty-one possible other instructions. Reassigning the opcode map changes these associations. A tool has been provided to make these reassignments randomly, with the option of specifying the hamming distance between pairs of instructions.

The genebank observational tool which keeps track of the genetic sequences of all programs in the population has been fully verified for accuracy and completeness. Several improvements have been made in the efficiency of the genebanker, including the use of a hash function. A phylogeny tool has been developed which allows the complete reconstruction of any evolutionary sequence. For every genome, the phylogeny tool records the genetic source of each instruction in the genome. The construction of complete phylogenetic trees will make it possible to determine how various factors (e.g. disturbance, population size, the structure of the instruction set) affect the shape of the phylogenetic tree.

A tool has been developed to scan the genebank that accumulates on disk during a run, and select genotypes according to a variety of criteria. The criteria include a lower threshold for maximum frequency achieved, time frame for appearance or the achievement of peak frequency, size of genome, ecological characteristics (mechanisms of interactions with other creatures), and several others.

Chris Stephenson of the IBM T. J. Watson Research Center has contributed a new memory allocator to Tierra, based on a cartesian tree scheme that he developed. The new allocator is more efficient than the old and provides a variety of options for controlling where offspring are placed. Free blocks are maintained in a two dimensional binary tree, in which daughter blocks can not be larger than their parents, and blocks are ordered from left to right according to their position in memory. The allocation options available are: first fit, better fit, or at a specific location. The memory allocator is used to determine the location of offspring during replication. When using the option to specify a location for the allocated memory, it can be left up to the mother creature to specify the location for her daughter. This can allow the evolution of dispersal strategies.

A virtual debugger has been created which allows the user to step through the genome of individual creatures. The contents of the cpu registers and the stack are displayed, along with the instruction pointer and the dis-assembled instruction being executed. The size and location of the mother and daughter cell membranes are also displayed. The virtual debugger includes a tracking feature which will allow you to either follow exclusively the execution of a single cell, or to swap between execution of cells as the slicer swaps.

The diversity tool has been enhanced in two ways. It optionally calculates an average of each of the diversity indices, over a user defined interval, and outputs only the cumulative averages at the end of each interval. Otherwise it may output the diversity indices after each birth and death. Outputting only the averages greatly reduces the volume of output. The other enhancement is a frequency filter which allows the diversity indices to be calculated on the basis of only size or genotype classes with at least some user defined number of individuals. The default is two individuals. This means that the calculations will not include the numerous mutants of which there are only a single individual. This prevents the diversity indices from being dominated by the large number of inviable mutants that always exist in the populations.

An option is now available to force creatures to cross-over their genomes with other creatures while reproducing. This results in a form of haploid sex. While it is not in the spirit of Tierra to force the creatures to do something, this option has been provided for those who wish to use Tierra for evolving application codes, to use in the event that crossover improves the efficiency of evolution.

Facilities have been provided to inject a genome into a running simulation. This may be used to test the viability of a particular genotype in a particular environment, or it may be used as a mechanism for creatures to migrate over the net between simulations (see the CM5 implementation below).

Working with the PI, Kurt Thearling of Thinking Machines Corporation has ported Tierra to the new Connection Machine, the CM5. The current port is a MIMD implementation based on an archipelago model. The program is parallelized by separating the simulation and genebank functions onto separate processors. The work of the genebank is split across many genebank processors by allowing each genebank processor to keep only a portion of the genebank. The CM5 nodes running the simulation each contain a complete Tierran soup. Different Tierra nodes are connected through immigration of creatures between nodes, via the network. The result is an archipelago of Tierran nodes, where each node can present creatures with different environmental variables. The nodes may be interconnected according to any desired topology. In addition, the distributed genebank is common to the entire archipelago, so that identical genotypes arising independently on different "islands" will be recognized as identical and cataloged as such.

The Tierra simulator now comes with four complete instruction sets, the original and three new ones. The three new sets came out of a series of meetings held by the Tierra group at the Santa Fe Institute over the summer of 1992. The new instruction sets are designed to be more suitable for general purpose computation. They overcome some of the obvious weaknesses of the original set. Specifically, they allow for movement of data between the soup (memory) and the cpu registers, and they provide I/O services. In addition, they include some important biological features. The template based sensory mechanisms have been extended so that they can locate matching templates beyond the closest match; they can start the search at any distance. The new instruction sets also provide facilities for inter-cellular (inter-cpu) communication, and they allow for the transmittal of non-genetic information between mother and daughter at birth, by transfering the contents of the cpu registers from mother to daughter. Also, they allow the mother cell to force the daughter cell to differentiate (express a different portion of the genome that the mother expresses), by controlling where the instruction pointer starts in the daughter cell. These last features provide the infrastructure to support the evolution of multi-cellular creatures (parallel software).

All of the work described above could be categorized as software development. In fact this has been the primary activity thus far under this grant. However, the software is reaching a plateau of effectiveness and utility, such that work is beginning to turn from development to using the software to study evolution. The first efforts in this area have been concerned with: 1) Determining how ecological factors affect competitive exclusion, as this will affect the efficiency with which new genotypes replace old in evolution. 2) Comparing multiple runs under identical and different conditions to determine the envelope of possible evolutionary outcomes. 3) Comparing the four instruction sets for their evolutionary properties. 4) Understanding the mechanisms of punctuations in evolution. These studies are currently underway.

Collaborative support for the project was provided by the Santa Fe Institute (SFI) who provided facilities and financial support for the project group members (principally the PI and Dan Pirone, a programmer, both in residence at SFI for seven months). Digital Equipment Corporation provided two DS5000 workstations for the project (donated to SFI). Thinking Machines Corporation sent an electrical engineer for two months (Kurt Thearling). Hughes Aircraft sent an electrical engineer for three weeks (Walter Tackett). The IBM T. J. Watson Research Center sent a computer scientist for three weeks (Chris Stephenson). All of these individuals overlapped for a period in the summer of 1992.

The grant became effective less than four months ago, therefore the results of this work have not been published yet. However, three progress reports were released on the computer network (internet etc.) through mailing lists and news groups. With each progress report, the new source code and its documentation was placed in a public ftp site: tierra.slhs.udel.edu. To subscribe to progress reports, send a request to tierra-request@life.slhs.udel.edu.

Project Description

This project is about inoculating the process of evolution by natural selection into an artificial medium. Self-replicating machine code algorithms (variously called: creatures, cells, individuals, and self-replicating algorithms) are run on special virtual computers designed for the purpose, in which mutations occur in the form of bit flips, and in which computations are sometimes erroneous. The result is heritable genetic variation among the offspring algorithms, and evolution by natural selection. The project encompasses both biological and computational goals, which appear distinct when viewed from the very different perspectives of the two disciplines, but which often turn out to be the same. I will discuss the goals first from the biological, and then from the computational perspective, and then show that accomplishment of many of the objectives will make significant contributions in both fields.

Biological perspective

(For those interested primarily in the computational aspects of the project, while reading the biological section it should be kept in mind that the actual objects under study are evolving machine code computer programs. The biological studies are intended to better understand the process of evolution in general, but the specific example is evolving computer software. Through a deeper understanding of the mechanisms and processes of software evolution, we will be better positioned to harness the power of evolution and manage it for useful purposes, with potential applications in the areas of artificial intelligence, massively parallel computation, and machine code optimization.)

The process of large scale biological evolution among organic organisms on Earth is too slow to be observed. Evolutionary biologists are left to attempting to infer the process and mechanisms of evolution from static observations of the diversity of living and fossilized products of evolution. This process of inference is proving highly problematic, and yields very limited and uncertain insights.

In addition to being limited to the observation of a single instant of evolutionary time (except through the sketchy fossil record), we are limited to the observation of a single instance of life, life on Earth. To a limited extent, geographic and temporal isolation have provided independent evolutionary experiments on Earth, however, ultimately all life on Earth is part of a single phylogeny, and therefore represents a single experiment. Based on a sample size of one, there is little we can say about what are the general properties of life, properties that would be found in any life form anywhere in the universe, regardless of the physical substrate in which it is embedded. Nor can we say much about what are the peculiar properties of life on Earth, which are either rare or absolutely unique to this one instance of life.

This project is about a method that can be used to overcome these limitations of evolutionary biology. The method is to create completely new living systems, in the form of self-replicating and evolving computer programs (Ray 1991a, b, c, d, 1992). It turns out that evolution occurs sufficiently rapidly in these systems that it can be studied on a macro scale. In addition, replicates can be run resulting in as many independent phylogenies as are desired.

Evolution in the digital system is embedded in a radically different physics and chemistry than the organic evolution with which we are familiar. The chemistry of carbon chains is replaced by the chemistry of chains of bits and bytes representing machine instructions. Therefore, any properties of evolution that are common to both the digital and organic systems must be among the most general of properties. A variety of fundamentally different digital systems can be created and each inoculated with evolution in order to increase the sample size of the physics within which evolution has been observed, thereby extending our knowledge of the general and peculiar properties of life.

It should be noted that the approach just presented would not be accurately described as biological modeling. In modeling, we try to capture as accurately as possible, some features of known living systems. This new approach involves, by contrast, the synthesis of new living systems for comparative purposes (among other things). The synthetic systems are never intended to be models of any known living systems. They do not represent biological viruses, bacteria, birds or plants. They are unique new life forms with their own unique properties. However, experience is showing that there are also many properties that these new life forms share with the old.

Some observers have noted that these new synthetic systems contain complexity on a level approaching that of the real world, and that what we have accomplished is the replacement as an object of study, of the complex real world about which we know very little, with complex artificial worlds about which we know even less. They ask then, what is the point? My response is that we want to understand complexity, but complexity in the real world is difficult to observe and manipulate. Artificial systems by contrast have the peculiarity that it is possible to observe, measure, and record any characteristic of the system without disturbing the system or altering its behavior (for example it is possible, in fact routine, to sequence the complete genome of every creature in the Tierra world, without disturbing the system, an impossible feat in the real world). This combined with a phenomenal ease of manipulability creates an experimenter's paradise.

Thus the inoculation of evolution into an artificial medium, the computer, allows a new approach to evolutionary biology which is at the same time synthetic and experimental. Rather than understanding life through a reductionistic process of breaking it down into its components, even to the molecular level, we study life by building it out of component parts, which in this study are based on a different physics and chemistry but analogous at the molecular level. The resultant evolving systems can then easily be subjected to experimentation.

Using the synthetic approach, we can ask questions like: "What are the minimal requirements for, or the conditions under which phenomena X will occur?" X may be something like: a) for a genetic language to be able to evolve, b) for the transition from single celled to multi-celled life to occur, c) for sexual reproduction to dominate over asexual reproduction or vice versa, d) for living systems to spontaneously increase in diversity and complexity, etc. These questions can be answered by building systems that exhibit the property of interest, then altering their elements systematically to determine what configurations will support the behavior and which will not.

The principal biological goals of the present proposal are:

1) To study the *process* of evolution in detail, through detailed analysis of the sequences of events and genetic changes in specific evolutionary runs.

2) To observe the envelopes of evolutionary possibilities, both under identical and very different conditions, and within and between the same and different physics, through the comparison of many replicates of runs.

3) To understand what factors affect the shapes of phylogenetic trees.

4) To understand what are the elements of evolvability in genetic languages that underlie evolving systems (i.e., what features of the language contribute to evolvability, which features inhibit it and what features don't matter).

5) To extend the system to include evolving (sexual) gene pools in addition to the branching asexual lineages of the present system.

6) To understand what elements are required for an evolving system to spontaneously increase in diversity and complexity, with specific reference to the transition from single celled to multi-cellular life.

Computational perspective

(The PI is an evolutionary biologist with no training in computer science or related fields. Therefore weaknesses in the following discussion are likely to be evident to specialists in those fields.)

The above discussion of the biological perspective speaks of using the evolution of digital organisms as a means of understanding the properties of evolution in general. However, the evolving entities in the Tierra system are actually machine code computer programs; thus the exercise will teach us about the properties of software evolution in particular. Use of the Tierra system can provide us with a deep knowledge of the properties of evolving computer software.

The development of computer hardware tends to greatly outpace the development of computer software. A long recognized but poorly realized method of alleviating this discrepancy is to automate the development of software. In a sense, this is one goal of the field of artificial intelligence, to make computers able to learn and adapt.

Computer scientists increasingly derive productive insights from observing the adaptive behavior of living systems, and appropriating their mechanisms into computational systems. Neural networks and genetic algorithms represent two of the main approaches to what some call adaptive computation.

The latter approach, genetic algorithms (GAs), is based on evolution. Genetic algorithms use the process of evolution to solve problems, generally problems of optimization. GAs have proven very effective in finding good solutions to otherwise nearly intractable problems. While GAs are not effective in solving every kind of problem, they have proven one of the best techniques for many classes of problems, particularly non-linear ones. Work is underway to characterize the classes of problems where GAs are and are not effective (Forrest and

Mitchell, 1992a, b; Mitchell, Forrest, and Holland 1991).

One of the greatest difficulties with GAs lies in finding an appropriate representation of the solution space of a problem, so that it can be subjected to the GA. In the GA, the "genome" consists of a bit string that represents a solution. Most GA techniques begin with random bit strings, then evaluate each one as a solution. The best solutions are then replicated with mutations and cross-over, to create the next generation of solutions. The new solutions are re-evaluated, and the process is iterated for many generations until the solutions converge.

It can be difficult however to represent the solution to a particular problem as a bit string. Furthermore, when the solution has been represented as a bit string, the form of the solution has usually been defined. The GA will then be left the work of adjusting the weights of the terms of the solution to find the optima. It would be desirable for evolution to be able to determine not only the weights of the terms, but the form of the solution itself. This is often not possible with the GA.

Computer programming languages are a more powerful tool for representing solutions to problems, therefore it would be ideal to apply the GA approach to this kind of representation. John Koza has developed an ingenious method of doing this with Lisp expressions which he calls the genetic programming paradigm (GPP). Lisp expressions can be represented as trees. Koza uses mutation to swap the nodal elements of the trees, and cross-over to swap entire branch units between trees. In this system the products maintain semantic and syntactic validity. Thus with the GPP, it is possible to evolve solutions in the form of the Lisp programming language. This methodology actually allows the form of the solution to evolve. The genetic programming paradigm has already demonstrated its ability to find good solutions to a stunning variety of problems (Koza 1990, 1991a, b).

The technique used in the present proposal could be considered a new member of the set of tools available to adaptive computation. Evolution of machine code is comparable to GPP in that solutions are represented through a computer programming language. As in GPP, the form of machine code algorithms can be altered through evolution.

Adaptive computation techniques are often tested on a set of standard test problems such as sorting a list, the traveling salesman, or the iterated prisoner's dilemma. Viewed from this perspective, I have chosen a more biologically motivated test problem, the original problem in biology: self-replication. As a test problem, I would suggest that self-replication is no more or less peculiar or appropriate than any of the other standard test problems. However, using self-replication has the advantage that one does not need to write an evaluation function to rank the validity of each solution. Self-replication self-evaluates. Self-replicating programs invent their own fitness functions as they discover completely novel way of accomplishing their tasks; in my system, often involving the exploitation of other creatures with which they share memory. This allows a level of creativity that is not found in systems where the user defines the fitness function.

When viewed simply from the point of view of optimization, evolution of machine codes has proven to be surprisingly powerful. The original self-replicating program was 80 instructions long. In some runs evolution converted this into a 22 instruction algorithm requiring only one sixth as many cpu cycles to reproduce. Under other conditions, evolution utilized "unrolling the loop" as an optimization technique. This latter solution resulted in an algorithm that was substantially more complex and intricate that the original code from which it evolved.

To my knowledge, thus far there has been only one experiment in the use of artificial selection to guide the evolution of machine code to produce "useful" code not related to self-replication (Tackett, In Press). Tackett inserted code that computed an Or function into the genome of the self-replicating ancestor. He then fed numbers to the self-replicating creature, and applied selection to the creatures on the basis of how well their responses conformed to the Xor rather than the Or. He then experimented with a carrot and a stick. The carrot is to reward creatures with cpu cycles for correct answers. The stick is to kill creatures for incorrect answers. Tackett found, as the PI has also found, that the carrot is a more effective means of control, and in fact, his creatures evolved to compute the Xor.

The PI also performed a similar but less "useful" experiment. Cpu cycles were provided to creatures only if their genomes contained an arbitrary "energy absorbing structure", in the form of a string of instructions which (when their numeric opcodes are interpreted as a base 32 numbering-lettering system) spelled out the word "chlorophill". It was imagined that "photons" were raining down on the soup at random. Wherever one hit, the chlorophill template was laid out, and the creature hit by the photon was rewarded with cpu cycles in proportion to the number of instructions in the area hit by the photon that align with the chlorophill template. For the ancestor, the best alignment throughout the entire genome was only one instruction. Through evolution, the genome became enriched in the instructions included in the template, and in chlorophill sub-strings. The test was not run for long, and in that time, the complete chlorophill string did not appear.

What we have learned is that evolution of machine codes is capable of both powerful optimization, and of restructuring the form of the solution to a problem. In addition, it is possible to guide evolution of machine codes with the use of artificial selection, so that it solves problems of our choice. However, this technique has not been tested on a wide variety of problems as has GA and GPP. In this sense the latter techniques are at a much more advanced stage of development.

Parallel computation

One area where software is falling the furthest behind hardware is in the area of parallel computation. There does not yet exist an art of writing massively parallel MIMD software. In the SIMD (single instruction multiple data) parallel architecture (e.g., the CM2), all processors must execute the same instruction in each cpu cycle. It is easy to write SIMD software, as essentially any serial algorithm can simply be broadcast to all the processors. However, in the MIMD (multiple instruction multiple data) parallel architecture (e.g., the CM5), there exists the possibility that each of thousands of processors could be executing different code, yet coordinating all their activity on the solution of a common problem. We are not yet able to write software that utilizes that potential.

In order to explain why the PI would assert that evolution is the only proven technique for producing massively parallel software (the same could be said of intelligence), an analogy will be made between a multi-cellular organism and a parallel computation. Think of each cell in the body of an organism as a processor. The program that they are running is the genome, however, no single cell expresses all of the genes of the genome. In multi-cellular organisms there are many cell types (e.g., brain cells, liver cells, skin cells, each of which categories actually includes many more specialized cell types).

One of the features that characterizes cell types is that within a type, all cells express the same subset of genes of the genome. All of the cells of a particular type are expressing the same genes, and it could be said that they are executing the same code. This is parallelism of the SIMD type, loosely speaking. Cells of different types express different genes, thus it could be said that they are executing different code, and this is parallelism of the MIMD type. Large organisms contain astronomical numbers of cells, thus this is parallelism on a truly massive scale, integrating both SIMD and MIMD parallelism into a smoothly functioning whole.

It is probably hopeless to contemplate actually writing parallel software at this level of complexity, however, evolution has proven its ability to generate such software (wetware). Therefore it seems that it is imperative to explore the possibility that evolution could be harnessed to generate such software in the computer. While it may not be possible, it seems like the best hope. The Tierran system is well suited to attacking this problem, because it is based on the evolution of machine codes, the native language of computers. By providing the instruction sets with a variety of mechanisms of inter-processor (i.e., inter-cellular) communications, evolving multi-cellular digital organisms can use evolution to explore the utility of each method, and we can learn from them what methods are effective for coordinating their activities.

The principal computational goals of the present proposal are:

1) To find what properties of a machine language and architecture provide a good balance of evolvability and computational power.

2) To experiment with the evolution of parallel software (parallel programs are the digital analogs of multi-cellular organisms).

3) To experiment with the control of the direction of evolution of machine code algorithms, by artificial selection, for the purpose of breeding programs to do useful work.

Commonality of biological and computational goals:

The first of the computational goals is largely identical to the fourth and fifth biological goals, and the second of the computational goals is largely identical to the sixth biological goal.

Instruction sets, enhanced power, enhanced evolvability:

When one defines an instruction set within which self-replicating algorithms will evolve, one determines what is and is not possible. Because evolution leads to emergent behavior, it is difficult design-in the possibilities inherent in an instruction set supporting evolution. None-the-less, it is important to understand the consequences of the design. The original instruction set used by the Tierra system supported the emergence of a rich computational ecology. However, the instruction set suffers from a variety of obvious weaknesses from both the computational and biological points of view.

It is essential to replace the original instruction set with new sets that are more rationally designed and more powerful. Biologically, there is a desire for enhanced sensory capabilities, support for variable ploidy levels to facilitate organized sexual reproduction, mechanisms to allow cell differentiation in a multi-cellular context, and a range of facilities for communication between the cells of multi-cellular digital organisms. Computationally there is a need for input/output facilities, the ability to move data between memory and the cpu, instructions that easily allow general purpose computation, and facilities to allow the coordination of parallel programs.

The enhanced capabilities just described need to be incorporated into new instruction sets without loosing the evolvability exhibited in the first. Discovering how to accomplish this is the first of the computational goals listed, and the fourth of the biological goals. Any feature of the instruction set that enhances evolvability (e.g., organized sexuality, the fifth biological goal) will contribute to achieving both the biological and computational goals.

Multi-cellular biology, parallel computation:

One of the most important events in the history of life on Earth was the Cambrian explosion of diversity, about 600 million years ago, during which complex multi-cellular life came into its own, after three billion years of dominance by single celled life forms. If something comparable can be made to occur in digital organisms, it will be possible to determine the fundamental elements required for the transition, by systematically removing each of the elements that were put in place to generate the transition, until the minimal set of requirements are determined. This may tell us something fundamental about the kinds of conditions that are required for life to make the transition from simple single celled forms to complex multi-celled forms (and for evolving software to make the transition from serial to parallel forms).

Multi-celled digital organisms will in fact be MIMD parallel programs. If it is possible to establish and maintain evolving populations of parallel programs, they would likely teach us a great deal about parallel programming techniques. Just as evolving serial programs quickly invented known optimization techniques such as unrolling the loop, evolving parallel programs will likely discover important parallel techniques, but this time they will be ahead of their human programmer counterparts. If we can, furthermore, manage the direction of evolution of digital organisms through artificial selection, then it may become possible to use them directly to develop some kinds of parallel software, in addition to learning techniques from them. It is in these domains that the second computational goal is joined with the sixth biological goal.

Proposed work:

The results from prior NSF support section describes some of the software development work that has been completed in the last several months. This work has brought the Tierra system to the point that it is ready to be used for the study of the evolution of machine codes. While development of the software system will continue, the emphasis of the work will shift now toward use and away from development of the system.

Below I repeat the primary goals listed above, and state what procedures will be used to accomplish them.

B1) To study the *process* of evolution in detail, through detailed analysis of the sequences of events and genetic changes in specific evolutionary runs.

This is done by analyzing the output saved to disk during a run, using the suite of observational and analytic tools that have been developed. Generally this begins by making plots of the size of the predominant creatures over time and various diversity indices over time. Then the genomes of all the most successful sequences over the course of the run are identified and disassembled (usually about 100 genomes are extracted). Each of these assembler programs is then studied in order to understand the nature of the algorithm, and comments and notes on the algorithm are added to the code. In come cases the algorithms can not be understood with certainty by reading the code, and it is necessary to run the algorithms in the context of their biotic environment, in order to understand the role that interaction between creatures plays in the execution of their algorithm. The phylogeny tool will make it possible to reveal "missing links", and characterize the complete chain of genetic events leading to any interesting adaptation, ecological changes, evolutionary punctuations, or major episodes of extinction or diversification.

B2) To observe the envelopes of evolutionary possibilities, both under identical and very different conditions, and within and between the same and different physics, through the comparison of many replicates of runs.

This can be accomplished by repeating the kind of analysis described for goal one above, for repeated runs under identical or different conditions. To start, eight runs under identical conditions will be analyzed (the sample size may be adjusted once the extent of variation is known). Then similar analyses will be made for substantially different "ecological" conditions such as mutation rate, "soup" size, disturbance frequencies, etc. Then comparisons will be made across the four instruction sets (or more sets as they become available). In order to be practical to make such comparisons across a broad range of conditions and with reasonable sample sizes, not every run can be analyzed at the level of detail requiring disassembly and analysis of hundreds of genomes. For the broader comparisons, such measures as diversity and turn over rates over time, rates of optimization, ecological composition (which to some extent can be determined automatically), and the shapes of phylogenetic trees can be used.

B3) To understand what factors affect the shapes of phylogenetic trees.

This can be accomplished by a comparative procedure similar to that discussed under two above. It has been observed that a number of ecological factors affect the rate or likelihood of competitive exclusion (e.g., disturbance intensity and frequency, degree of randomness of mortality, variance in the allocation of energy). The rate and likelihood of competitive exclusion must be related to the ability of new evolutionary innovations to spread through the population. These factors are likely to affect the shape of the phylogenetic tree. Once individual environmental factors are characterized with respect to their influence on competitive interactions, runs will be conducted at reasonable extremes of these factors, in order to determine if and how they influence the shapes of trees.

It has been observed that populations evolving under strong selection for efficiency tend to get smaller. This progress towards smaller sizes is sometimes continuous and gradual (gradualism), it sometimes stalls and remains flat for a period of time (stasis), and it sometimes occurs in abrupt jumps (punctuations). Of the four instruction sets studied thus far, one exhibits gradualism with occasional punctuations, two exhibit exclusively gradualism with no punctuations, and one exhibits exclusively stasis with punctuations but no gradualism. It will be interesting to see how the shapes of the phylogenetic trees vary under these different patterns of evolution.

Exploration of phylogenetic trees will require some additional software development in the areas of the visualization of the trees, and in quantification of the elements of their shapes for comparative purposes.

B4) To understand what are the elements of evolvability in genetic languages that underlie evolving systems (i.e., what features of the language contribute to evolvability, which features inhibit it and what features don't matter). C1) To find what properties of a machine language and architecture provide a good balance of evolvability and computational power.

Analysis of differences in the mechanisms and processes of evolution of the four instruction sets now available will be the first step in this process. They differ in some striking ways as described in goal three above, and they also differ in the extent to which they are able to optimize. An attempt will be made to understand what underlies these differences, by comparing the details of the mechanisms and processes of evolution in the four sets.

New instruction sets will be developed to systematically study how the elements of the language affect its evolvability: alignment, number of bits per instruction, constancy of number of bits per instruction, Huffman encoding, use of numeric operands, direct and relative addressing, syntax, hamming relationships between related instructions, smart mutation operators, and others that come to mind as the result of experimentation.

To compare the evolvability of varying architectures and instruction sets, techniques will be used similar to those presented in Ray (1991d, 1992). Sets will be compared on the basis of both rates of optimization under selection for small creatures, and rates of diversification of self-replicating algorithms.

B5) To extend the system to include evolving (sexual) gene pools in addition to the branching asexual lineages of the present system.

Sexual reproduction is the predominant mode of reproduction among higher organisms on Earth. Thus it appears possible that this type of reproductive system enhances the evolvability of organisms. Incorporation of an organized sexual process into the reproduction of digital organisms may enhance their evolvability, making them more useful for the evolution of useful software.

In a sexual system, the entity that evolves is a pool of genes; the individual organisms are just ephemeral carriers of the genes. In an asexual system, the entity that evolves is a branching lineage of individuals. In order for the Tierran system to allow the study of evolving gene pools, it must allow organized sexual reproduction. This is critical to any studies of the process of speciation, as the very concept of species is tied to sexual reproduction and the central position of the gene pool.

The infrastructure for organized sexuality via variable ploidy levels has already been coded into the Tierran system in the form of parallel tracks for genomes. Some additional details need to be worked out with respect to the control of expression of multiple copies of the genome. In addition, new creatures that make use of these facilities need to be written and tested, both as to functionality and evolvability, and in comparison to asexual creatures.

B6) To understand what elements are required for an evolving system to spontaneously increase in diversity and complexity, with specific reference to the transition from single celled to multi-cellular life. C2) To experiment with the evolution of parallel software (parallel programs are the digital analogs of multi-cellular organisms).

It is believed that the three new instruction sets all include the elements necessary to support the evolution of multi-cellular (parallel) digital organisms. What remains is to write and test such organisms. Some would say that such things should be able to evolve spontaneously, and they might. But it has been the philosophy of the Tierra project to engineer the major goals first, then try to let them evolve spontaneously later. For example, the original Tierra experiment did not start with a randomized soup, then wait for self-replicating algorithms to emerge spontaneously. The original soup was simply seeded with a self-replicating algorithm. Now that a system that supports a rich evolution is at hand, experiments are being conducted on the spontaneous origin of self-replication.

By analogy, experience will first be developed with evolving multi-cellular organisms, and their spontaneous origin will be addressed later. It will be necessary for example to study the interactions between sexual and asexual creatures living in the same soup. It will be necessary to determine under what conditions sexual organisms are able to coexist or dominate against what are likely to be more efficient asexual creatures.

With respect to the evolution of parallel software, the evolution of useful applications will not be addressed at the outset. Rather, the focus will be on gaining a familiarity with the forms taken on by “wild” multi-celled digital organisms, in order to see what kinds of inter-cellular communication mechanism prove most useful to them. This may lead to the redesign of the instruction sets to provide them with more useful communication tools.

This will result in the evolution of systems with functionalities similar to neural-nets, however, the precise form and nature of the connections will not be forced to be structured like biological neural systems. Current neural-net techniques force a digital system, the computer, to behave like an analog system, the neuron. By providing a variety of communication mechanisms to evolving multi-celled digital organisms, evolution may lead to connectionist systems more “natural” to the computational medium, which should be more efficient and evolvable that traditional neural-net systems implemented in computers. Once the properties of these parallel algorithms are known, attempts can be made to breed them to do useful work. Even without applying artificial selection, evolving wild parallel digital organisms should spontaneously increase in complexity, leading in the direction of intelligent forms.

C3) To experiment with the control of the direction of evolution of machine code algorithms, by artificial selection, for the purpose of breeding programs to do useful work.

Humans have been controlling the evolution of other species for ten thousand years. This is the basis of agriculture, and plant and animal breeding. This art can no doubt be extended to the control of the evolution of digital organisms. Already this is routine in the field of genetic algorithms, and in the genetic programming paradigm.

What is required to control evolution is to have an evaluation function that assigns fitness to each individual. With fitness assigned, we use a carrot and a stick. The stick is death, and the carrot may be energy (cpu cycles), or any other limiting resource.

There are two distinct approaches to managing evolution of digital organisms:

a) Sequences of machine codes may be treated as genetic algorithms. The algorithms are evaluated as to effectiveness, and the most fit algorithms are copied with mutations and recombinations by the GA software.

b) In an analogy to genetic engineering, application codes (genes) may be inserted into the genomes of self-replicating digital organisms, and those whose application genes perform better, will be rewarded with cpu time, or those whose application genes perform badly can be killed.

These two methods vary in the degree of control exercised by the user. In the GA system the user has total control over every detail: the fitness function, the act and mechanism of replication and the genetic operators. In the self-replicating system, some aspects of fitness are left up to the organisms (due to self-replication). It is likely that the first method will be more efficient, but the second will be more creative. This trade-off needs to be explored.

Bibliography

Forrest, S., and M. Mitchell. 1992a. Towards a stronger building-block hypothesis: Effects of relative building-block fitness on GA performance. To appear in D. Whitley (ed.), Proceedings of the Foundations of Genetic Algorithms Workshop.

——, and ——. 1992b. What makes a problem hard for a genetic algorithm? Some anomalous results and their explanations. To appear in *Machine Learning*.

Koza, J. R. 1990. Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical Report STAN-CS-90-1314, Department of Computer Science, Stanford University, CA.

——. 1991a. Evolution and co-evolution of computer programs to control independent acting agents. In J. A. Meyer and S. W. Wilson, editors, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, 366-375, Cambridge, MA, MIT Press.

——. 1991b. Evolving a computer program to generate random numbers using the genetic programming paradigm. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, 37–44, San Mateo, CA. Morgan Kaufmann.

Mitchell, M., S. Forrest, and J. H. Holland. 1991. The royal road for genetic algorithms: Fitness landscapes and GA performance. In *Proceedings of the First European Conference on Artificial Life*, Cambridge, MA. MIT Press/Bradford Books.

Ray, T. S. 1991a. Is it alive, or is it GA? *In*: Belew, R. K., and L. B. Booker [eds.], Proceedings of the 1991 International Conference on Genetic Algorithms, 527–534. San Mateo, CA: Morgan Kaufmann.

——. 1991b. An approach to the synthesis of life. *In*: Langton, C., C. Taylor, J. D. Farmer, & S. Rasmussen [eds], Artificial Life II, Santa Fe Institute Studies in the Sciences of Complexity, vol. XI, 371–408. Redwood City, CA: Addison-Wesley.

——. 1991c. Population dynamics of digital organisms. *In*: Langton, C. G. [ed.], Artificial Life II Video Proceedings. Redwood City, CA: Addison Wesley.

——. 1991d. Evolution and optimization of digital organisms. *In*: Billingsley K. R., E. Derohanes, H. Brown, III [eds.], Scientific Excellence in Supercomputing: The IBM 1990 Contest Prize Papers, Athens, GA, 30602: The Baldwin Press, The University of Georgia. Publication date: December 1991.

——. 1992. Evolution, ecology and optimization of digital organisms. Santa Fe Institute working paper.

Tackett, W. A., J. L. Gaudiot. In press. Adaptation of self-replicating digital organisms. Proceedings of IJCNN, November 1992, Bejing China. IEEE Press.