Studying Evolution with Self-Replicating Computer Programs

Tim Taylor and John Hallam
Department of Artificial Intelligence, University of Edinburgh
5 Forrest Hill, Edinburgh EH1 2QL, U.K.
{timt, john}@dai.ed.ac.uk

Abstract

A critical discussion is presented on the use of self-replicating program systems as tools for the formulation of generalised theories of evolution. Results generated by such systems must be treated with caution, but, if used properly, they can offer us unprecedented opportunities for empirical, comparative studies. A new system called Cosmos is introduced, which is based upon Ray's Tierra [15]. The major difference between Cosmos and previous systems is that individual self-replicating programs in Cosmos are modelled (in a very simplified fashion) on *cellular* organisms. Previous systems have generally used simpler self-replicators. The hope is that Cosmos may be better able to address questions concerning the sudden emergence of complex multicellular biological organisms during the Cambrian explosion. Results of initial exploratory runs are presented, which are somewhat different to those of similar runs on Tierra. These differences were expected, and indicate the sensitivity of such systems to the precise details of the language in which the self-replicating programs are written. With the strengths and weaknesses of the methodology in mind, some directions for future research with Cosmos are discussed.

1 Self-Replicating Program Systems as a Methodology for Studying Evolution

Within the last decade, computers have become powerful and affordable enough to enable a number of research groups to study the evolution of life in a new way. Rather than following the traditional approach of trying to capture properties of whole populations in mathematical models, the new approach models a large number of *individual* self-replicating entities which are competing against each other for resources required for replication. This is achieved by creating a computer which can run a large number of self-replicating programs in parallel[1].

Tom Ray pioneered this approach with his Tierra system [15, 16]. Since then, a number of other systems have also been developed, including Avida, developed by Chris Adami and Titus Brown [2], Computer Zoo, written by Jakob Skipper [18], and John Koza's system of self-replicating LISP-like programs [8].

Using such a methodology to study evolutionary systems is attractive for a number of reasons. For example, as the self-replicators are being modelled individually rather than as populations, the simulation respects the fact that a gene does not work in isolation. Rather, it is part of a large ensemble of genes which must all work together with some degree of cooperation in order for the individual organism which carries them to replicate and thus propagate the genes collectively [6]. Only through explicitly modelling individual organisms may we begin to understand the complex interactions between genes and organisms, and how these interactions affect the dynamics of the evolutionary process. In addition, by including an analogy to the process of development of multicellular organisms from single cells[2], this methodology provides a tool for investigating the interplay between such generative processes and an organism's genes—a question which is currently the subject of considerable debate (e.g. [6, 7, 14]).

In fact, systems which use self-replicating programs are not merely *simulations*. As the programs replicate *themselves* (with genetic novelty being introduced by mutations and the flawed execution of instructions), rather than being selected and copied according to an externally defined fitness function (as in genetic algorithms), they *recreate* the conditions necessary for evolution. Such systems can therefore be called *synthetic life* or *artificial life*[3], rather than just simulations.

Every computer-based self-replicating program system therefore provides a new instance of evolution. This

[1] In practice, a virtual computer is created (i.e. implemented in software), with parallelism simulated by time-slicing between the programs. Similar approaches are also being employed to investigate related subjects such as the *spontaneous emergence* of self-replicating programs (e.g. [8, 13]), although it is unclear what this work can tell us about the emergence of biological life.

[2] For example, by modelling organisms as *parallel* programs which can dynamically create additional processes.

[3] The term 'artificial life' is now used in a wide variety of situations, but we would argue that only systems based upon *self*-replicating entities are deserving of the term in the strong sense.

leads the way to a field of empirical study we have previously been unable to explore, namely *comparative evolution*. This refers both to comparisons *between* different instances of evolutionary systems (including biological evolution), and also to comparisons *within* a single system. The latter involves investigating the sensitivity of the system to initial conditions, parameters, etc. by running it many times over under slightly different conditions—a strategy which is not, of course, possible in the case of biological evolution.

When studying a population of organisms which are the end-product of an evolutionary process, we generally want to disentangle the relative contributions of three factors to the final state of the system:

1. Features due to chance events/historical accident.
2. Features due to the particular components of the system and to the laws governing their interaction.
3. Features which may be general to a wide class of evolutionary systems.

Through empirical studies of comparative evolution, the hope is that we can begin to investigate each of these factors, and move towards a truly general theory of genetic auto-adaptive systems (to use a term suggested in [1] to cover both biological evolution and self-replicating program systems).

When studying the performance of a genetic auto-adaptive system it is important to consider the relative contributions of each of the three factors. For example, runs on Tierra often result in the evolution of 'parasite' programs which cannot replicate by themselves, but utilise the code of neighbouring programs to perform this task [15]. At first glance, this is an exciting and unexpected result. However, when Tierra's mechanisms for template-driven branching are considered, where the flow of control in a program can jump just as easily to a point on a nearby program as it can to somewhere on the same program, the fact that parasites emerge becomes a little less surprising. The fact that we observe parasitic behaviour in nature and in Tierra might lead one to suppose that this may be a common feature of genetic auto-adaptive systems. However, on closer inspection it would appear that the emergence of parasites in Tierra owes much to the particular design of the language.

Parasites and symbiosis also emerged in Computer Zoo, but there too, Skipper came to the conclusion that "the concept of remote execution seems to be essential to the evolution [of parasitic behaviour]" [18].

A number of researchers have voiced their concern about the extent to which self-replicating program systems can really help us in our understanding of biological evolution. Mathematical biologist Robert May has said that, although he finds this work stimulating, he has "slight reservations about the extent to which the conclusions are perhaps inadvertently built into the program", as well as doubts about the robustness of the findings[4]. These are justified concerns, but they can be partially allayed by using carefully designed comparative studies [3]. In this way, by manipulating individual factors and noting effects on measured variables, the relative contributions of factors to the behaviour of the system can be investigated.

Indeed, several within-system comparative studies have been published for Avida (e.g. [1]) and Tierra [16][5]. However, just looking at differences *within* a system may not be enough to satisfy critics of this methodology. There will always be questions of how close an analogy is being drawn to biology, and of what stage of biological evolution the system is trying to recreate. Problems arise in these areas because there is a conflict between trying to model the physics and chemistry of the real world, and creating an efficient and "natural" representation for the logical/informational world of the computer. The complexity of the physical/chemical natural world must usually be greatly simplified in a computational model, and many aspects of this complexity are ignored completely.

The extent to which such questions matter depends upon whether one is really trying to model biological life, or rather trying to create artificial life in an appropriate form for the digital medium. To the extent that the former is true, it must be asked how important are the simplifications and omissions of the model to the performance of the system.

These issues again highlight the need for comparative studies, not only within systems, but also *between* them. The greater variety of genetic auto-adaptive systems we have access to, the more we can learn about how important the particular components of a system, and the laws governing their interactions, are to the behaviour of that system. Unfortunately, because of the impossibility of exactly modelling the complexities of physical and chemical systems, it is still the case that, at least in the near future, self-replicating computer systems will be more similar to each other than they are to biological evolution. Therefore, we may still not be able to learn much about biological evolution at this stage using such a methodology. However, this should not prevent us from exploring these systems, as gross simplifications must be made in the initial stages of any branch of scientific enquiry.

2 Motivations for Building a New Genetic Auto-Adaptive System

There are a number of reasons why the new system has been created. The first is simply because, as just men-

[4] From [4], Chapter 8.

[5] This study looked at the effects of using each of four different instruction sets. A more informative experiment would consider the effects of individual instructions within an instruction set.

tioned, the more such systems we have, the more we can learn about evolution. This is especially true if the systems are somewhat different to each other.

A second reason was to attempt to pitch the analogy at a somewhat later stage of biological evolution. One of the original aims of Tierra was

> "to parallel the second major event in the history of life, the origin of diversity [the Cambrian explosion, 600 million years ago]. Rather than attempting to create prebiotic conditions from which life may emerge, this approach involves engineering over the early history of life to design complex evolvable organisms, and then attempting to create the conditions that will set off a spontaneous evolutionary process of increasing diversity and complexity of organisms. This work represents a first step in this direction, creating an artificial world which may roughly parallel the RNA world of self-replicating molecules (still falling far short of the Cambrian explosion)" Tom Ray [15].

The system described in this paper is called Cosmos[6]. It has been designed to model (in a very simplified fashion) some of the features of cellular organisms, such as gene regulation, an evolvable mapping between genotype and phenotype, energy storage, inter-cellular communication and inter-organism communication. The hope is that Cosmos may be better able to address questions concerning the sudden emergence of complex multicellular organisms during the Cambrian explosion, in the face of selective pressures which should normally force evolving systems in the direction of smaller and simpler organisms. This is a question that has interested ecologists for a long time.

It is a commonly held belief that once evolution hits upon multicellularity, the emergence of complex organisms is an inevitable result. We may therefore wish to ask questions such as: How easy is it for evolution to hit upon multicellularity? What are the initial advantages for organisms that adopt multicellularity over those that do not? What conditions are required for multicellular organisms to emerge?

On the other hand, mathematical models of ecosystems suggest that, in general, increased complexity makes for diminished community stability [11]. As there are many cases where Nature appears to maintain ecosystem stability despite the complexity of the ecosystem, there is therefore also a need to "elucidate the devious strategies which make for stability in enduring natural systems" ([11] p.174).

The creation of the Cosmos system not only required various new features to be added to the basic Tierra design, but also required a number of existing features to be modified to fit the new analogy. The main innovations of Cosmos are described in Section 3, but the reasons for changing existing features of Tierra are as follows:

- It has already been said that programs in Tierra can directly execute the code of neighbouring programs. This could be argued to be analogous to certain processes in a (hypothetical) system of self-replicating RNA molecules. However, as Cosmos programs are supposed to be analogous to *cellular* organisms, they should not be able to directly execute the genetic code of other organisms. There is clearly still a need to allow organisms in Cosmos some method of communication and/or interaction, but this should preferably be somewhat more indirect.

- In Tierra, CPU time is the analogy for energy in a biological system. At each timeslice, every program is allowed to execute a certain number of instructions, depending only on the size of the program being executed. In a sense, the programs are getting energy 'for free', in that there is no notion of a program having to *capture* and *store* energy, and then *convert* the energy into useful work (executing an instruction). Cellular biological organisms certainly do have to concern themselves with such issues, so the idea of energy (CPU time) as a *commodity* which must be captured, stored and converted to useful work is incorporated into the design of Cosmos.

- As a consequence of Tierran programs being given energy for free, a somewhat arbitrary mechanism has to be introduced to decide which programs get killed off when the available memory in the system starts to fill up. The 'reaper queue' performs this function—programs are placed at the bottom of the queue when they are born, and programs at the top of the queue get killed off when more memory is required. Programs can move up the queue if they cause error conditions when run, and they can move down the queue if they successfully execute difficult combinations of instructions, "but, in general, the probability of death increases with age" [15]. The reaper queue effectively imposes an upper limit on the lifespan of programs. While at first glance this may seem like a sensible mechanism, there is no *a priori* reason for assuming that there should be a fixed maximum lifespan for all members of an evolving system. Indeed, in nature we see great diversity in the duration of organism life-cycles. The typical lifespan for members of a species is presumably a compromise between factors such as an individual's longevity, its fecundity, and the evolvability of the lineage. In Cosmos, the chance of an organism dying depends upon how much energy it has stored within it (as explained in Section 3).

[6] Cosmos is an acronym for "COmpetitive Self-replicating Multicellular Organisms in Software".

This mechanism imposes no fundamental limits on the lifespan of organisms.

This final point raises a more fundamental question. In any population of self-replicating entities which are competing against each other for resources required for replication (e.g. energy and materials), there are three factors which determine the rate at which any particular type of replicator will spread throughout the population [6]. These are the life-span or *longevity* of the replicator, the rate at which it replicates (its *fecundity*), and the number of errors in makes while producing copies of itself (its *copy-fidelity*). In Tierra, evolution can change the fecundity of a program by making it shorter or longer (a shorter program can be copied quicker than a longer one, all other things being equal). However, the reaper queue mechanism means that programs have minimal control over their longevity. Tierran programs also have no control over their copy-fidelity, as this is determined by global parameters of the system. The design of Tierra therefore restricts programs to evolve along the 'fecundity axis', with longevity and copy-fidelity being more or less fixed. Cosmos has been designed to allow organisms to also evolve along these other two axes.

There is an additional advantage in requiring programs to capture energy (potential CPU time) from the environment and store it for future use: each program becomes a potential resource of energy for other programs. There is therefore the potential for predator programs to evolve which prey on the energy resources of other programs, and for an exploitative coevolutionary arms race to emerge [12]. If such a process occurs, organisms on a number of different trophic levels might emerge in the system. Such conditions are undoubtedly of great importance in the evolution of complex organisms. In fact, it has even been proposed [19] that the Cambrian explosion was *caused* by the appearance of the first organisms that ate other organisms (heterotrophs).

3 Novel Features of the Cosmos System

The Cosmos system is explained in detail in [20]. It is written in an object-oriented style that allows it to be easily modified and expanded. The general design philosophy has been to make the system as flexible as possible and to try to model as many features of cellular organisms and their physical/chemical environment as possible, at least in a very abstract way, so as not to constrain the system's evolutionary potential. In addition, care was taken to ensure that all features of the system could be implemented in a computationally efficient way.

The general mode of operation is the same as Tierra, in that it simulates the parallel execution of a large number of self-replicating programs written in a low level language that has been designed to be robust under mutation. Variety between programs is introduced by two methods: a *mutation* operator, whereby any bit of any program in the system can be flipped (with a constant, low probability), and *flawed execution of instructions*, whereby an instruction, which would normally be executed once, might instead by execute twice or not at all (the rate at which this happens is again very low, but is an evolvable property of an individual program).

Cosmos uses a different instruction set to Tierra. Most Tierran instructions have equivalents in Cosmos, but additional instructions are included to provide the different functionality described below (and in Section 2).

As already mentioned, a primary motivation for building Cosmos was to address questions concerning the sudden emergence of complex multicellular organisms in the Cambrian period. In Cosmos, parallel programs are considered as the analogy of multicellular biological organisms. The same analogy has been used for studying multicellularity in Tierra [17, 21, 22]. (Using this analogy, the process of development from a fertilised egg cell to an adult organism is equivalent to the formation of a parallel computer program from an initially serial program by the dynamic creation of parallel processes as the program runs.)

Therefore, Cosmos has been designed with mechanisms to allow for parallel programs with inter-process communication and analogies for genetic regulation and energy transfer between cells. The main features which differ from some or all previous systems are as follows:

3.1 Cellularity

Each program within Cosmos is an *Organism* object. An organism contains one or more *Cell* objects. Each Cell object represents a single process, so that an Organism with one Cell is a serial program, and an Organism with multiple Cells is a parallel program. A Cell contains a bit string—the *Genome*, which gets translated to the executable code of the process. A Cell also contains a number of other objects, including: *Nucleus Working Memory* for writing a copy of the Genome for replication; *Communications Working Memory* for composing arbitrary messages; a *Regulator Store* containing promoters and repressors which dictate which sections of the Genome are translated; a buffer for receiving incoming messages; an *'Energy Token' Store*; four 16 bit registers and a stack.

When a Cell issues a `divide` command, the contents of the Nucleus Working Memory are written to the Genome of a new Cell object in a new Organism object. Most of the other structures of the new Cell are initially empty, but half of the parent Energy Token Store is transferred to the child, as is half of the contents of the Regulator Store.

The process by which a Cell dynamically creates a parallel process (another Cell) within the same Organism is exactly the same, except it is initiated by a `split` command rather than a `divide`.

Other points to mention are that when a Cell splits, it can specify a preferred location for its offspring in relation to itself (which is important for intercellular gene control and energy transfer, explained later), and, once created, a Cell can migrate to a new location within the Organism. There is also an experimental parameter which defines the energy *cost* of multicellularity (i.e. at each timeslice, a certain number of energy tokens are deducted from each Cell in a multicellular Organism, proportional to how many neighbouring Cells it touches).

It is worth highlighting a few consequences of this design. As mentioned previously, some experiments have already been conducted on evolving parallel programs in Tierra [17, 21, 22]. (This work will be referred to as *Parallel Tierra*.) Parallel Tierra uses a shared memory approach to parallelism, and, although it is theoretically capable of supporting MIMD (Multiple Instruction, Multiple Data) programs (i.e. differentiated multicellular organisms), it has so far only demonstrated the evolution of SIMD (Single Instruction, Multiple Data) programs. In contrast, Cosmos uses a distributed memory model of parallelism, and the regulator system that it employs (explained later) should promote the emergence of MIMD programs. In addition, unlike in Parallel Tierra, each cell within a multicellular organism in Cosmos actually contains a separate copy of the genome. Although this may appear to be unnecessary, it has a number of possible advantages. For example, the process of cell splitting (organism growth by the creation of parallel processes) is virtually identical to that of cell division (creating a new organism). This means that it is far easier for evolution to experiment with multicellular organisms, as little change is required from the basic self-replicating algorithm to produce an organism that grows rather than divides. Cosmos is therefore better suited for looking at the initial *emergence* of multicellular organisms from unicellular ones[7], and the conditions under which successively more complex multicellular organisms might evolve. Any satisfactory account of the evolution of multicellular organisms must proceed in a stepwise manner such as this. As Richard Dawkins notes in [5], "[a] complex developmental sequence has to have evolved from an earlier developmental sequence which was slightly less complex" (p.258), so "[t]he Darwinian must begin by seeking immediate benefits to genes promoting this kind of life cycle, at the expense of their alleles" (p.263). Another consequence of the design is that all cells within an organism can potentially divide to produce a new organism. In other words, they are all potentially germ-line cells[8]—no *a priori* assumptions are made as to which cells are germ-line and which are not.

[7] In contrast, in the work reported on Parallel Tierra, the initial ancestor program has itself been parallel rather than serial.

[8] Unlike in Parallel Tierra where only one process is capable of producing a new organism.

3.2 Communication

Cosmos uses a very flexible method for allowing programs to broadcast and receive messages to and from other programs. Basically, any cell can compose an arbitrary bit string in its Communications Working Memory, and then transmit this message to the environment. Other cells (which could belong to the same organism or a different one) can then issue a command to pick up specified types of environmental messages which are being transmitted from cells in their locality. This mechanism is an attempt to allow programs to develop arbitrary channels of communication in much the same way that biological organisms can communicate arbitrary messages using media such as light and sound.

There is a further twist to this mechanism—if certain conditions are matched for a received message, it will be treated as equivalent to the host code (i.e. it may be executed like a section of the program). In this way, genetic material may also be transferred between programs. Again, the general design philosophy has been to allow the evolutionary process some of the freedoms enjoyed by biological organisms and to prevent it from being unduly constrained. The analogy to the biological case is tenuous, but the fact that we are working with a logical/information medium, rather than a physical/chemical medium, must be respected. Whatever the details of the design, the important point is to provide that organisms with *some* forms of communication, as we are only now beginning to realise the great importance of communication in biological organisms even as simple as bacteria [9].

3.3 A $2\frac{1}{2}$D Environment

One of the problems that has been observed with the process of evolution in Tierra is that it suffers from premature convergence due to global interactions between cells [2]. Adami and Brown sought to overcome this problem in Avida by giving each of the cells a location on a two dimensional toroidal grid. Cells can only interact with other cells occupying nearby grid positions, thereby slowing down the rate of propagation of evolutionary changes throughout the total population and promoting heterogeneity (biodiversity). In Cosmos, programs live on a 2D grid, where each cell occupies a specific grid position. Each organism is flat—that is, each of its cells must be located at a different position on the grid. Within a multicellular organism, individual cells can only pass regulators and energy tokens to neighbouring cells with which they are in physical contact. Cells from different organisms can, however, share a grid position and thus compete for energy. The system is therefore $2\frac{1}{2}$D, but is still computationally easy to manage. Organisms can move around the grid if their cells execute the appropriate instructions.

As well as promoting biodiversity, this design means that the organisms live in a Euclidean space which is at least partially comparable to the 3D space of biological organisms.

3.4 Energy Tokens

At the beginning of each timeslice, a number of *energy tokens* are distributed across the environment. Each cell must issue an *et_collect* command to pick up tokens from its current location. These tokens get added to the cell's Energy Token Store. When it is that cell's turn to execute some instructions, energy tokens are deducted from its store for each instruction it executes. If the level of the store falls below a certain threshold, the cell dies.

There is a (high) limit on the total number of cells that may exist on a single Cosmos system. If this limit is reached, memory is released by killing off cells stochastically, where the chance that a cell is killed is inversely proportional to the level of its Energy Token Store. However, the total number of cells in the system can also be effectively controlled via the quantity of energy tokens that are pumped into the environment at each timeslice. By reducing this quantity, it is possible to reach a situation where this global culling is never required, because the rate at which cells are dying through lack of energy equals the rate at which new cells are being produced.

It could be argued that this mechanism is to some extent equivalent to the 'reaper queue' of Tierra—that 'illegal instructions' are just being replaced by 'amount of energy' as the factor which governs how long a cell survives. However, the current method has the advantages of not imposing a maximum age limit on cells, and of allowing the possibility of the development of trophic levels within the population of organisms, as mentioned in Section 2.

An additional feature concerning energy tokens is that, in a multicellular organism, a cell can send energy tokens to neighbouring cells with which it is in physical contact. This feature was included to allow for the possibility of the evolution of organisms which possess specialised energy collecting cells which distribute energy to the rest of the organism.

3.5 Indirect Mapping from Genotype to Program Instructions

The genome of a cell in Cosmos is literally represented as a string of bits, which gets translated into instructions using a 'genetic code' stored in the cell. In other words, in contrast to any other system of this type, there is an indirect mapping between genotype and phenotype[9].

It has been argued that the mapping from genotype to phenotype determines the *phenotypic variability* of a species, and therefore its *evolvability* [23]. Cosmos can be used to investigate such issues. For example, it is easy to test the effect of different mapping schemes on the behaviour of the system. Also, it can easily be configured so that each cell owns its own map of the genetic code, which can therefore evolve along with the rest of the cell.

3.6 Regulation of the Genome

The flow of control when reading a genome is governed by the presence of *Regulators*. These come in two forms, *promoters* and *repressors*. Both types are associated with a short bit string which determines to which parts of a genome they may bind. Promoters define the sites at which translation of the genome may begin, and repressors define sites at which translation stops. There are two ways that regulators can enter (or leave) a cell—they can either be produced (or destroyed) by the cell itself through the execution of specific commands in the instruction set, or, in the case of multicellular organisms, they can be passed from one cell to a neighbouring cell within the organism. In this way, a complex regulatory network can emerge. This mechanism was designed in an attempt to loosely model gene regulation in biological cells. It is hoped that such a system might promote the emergence of cell differentiation via gene control in multicellular organisms. Another consequence of this mechanism is that, as the genome and the regulators work at the level of individual bits, different promoters are not restricted to binding to the genome in the same reading frame. In other words, if, for example, one promoter binds to the genome five bit positions down from a second promoter, and each instruction is encoded in six bits, the first bit of the first instruction translated by the first promoter is actually the last bit of the first instruction translated by the second. The promoters are working in different reading frames, and will translate the genome into completely different programs. This can also happen in biological systems, where it has been observed that some species actually encode multiple instructions on the same section of the genome by using shifted reading frames (e.g. [10] p.144).

4 Observations from Preliminary Runs

In this section, some observations from the very first exploratory runs of Cosmos are described. The purpose of these was to quickly ascertain the basic behaviour of the evolving programs, and to check that the system worked correctly over a number of long runs, before commencing work on more carefully designed, more specific, comparative experiments.

Three long runs have been conducted, each using similar parameter settings, but with different schedules of energy token distribution across the environment. The parameter settings for the runs are listing in the Ap-

[9] Where 'phenotype' in this case refers to the executable program.

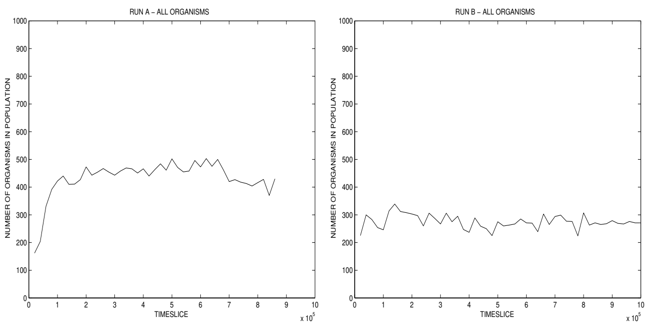

Figure 1: Number of Organisms plotted against Time in Run A (left) and Run B (right). See text for details.

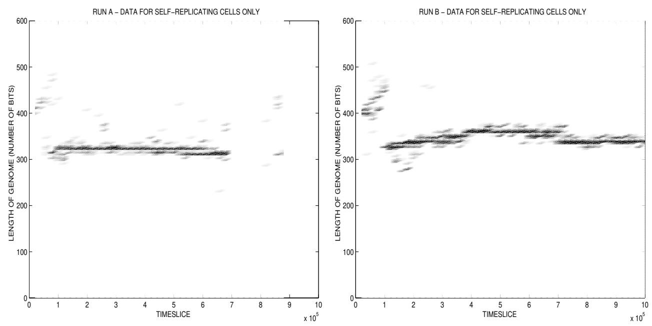

Figure 2: Length of Genomes plotted against Time in Run A (left) and Run B (right). See text for details.

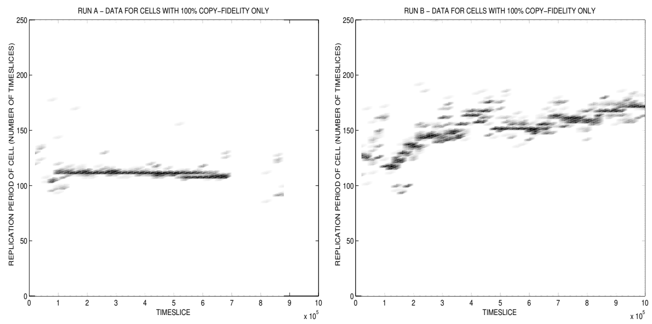

Figure 3: Cell Replication Periods plotted against Time in Run A (left) and Run B (right). See text for details.

pendix. In the light of the discussion in Section 1 it is stressed that the scientific significance of these observations, by themselves, is minimal, as they are not carefully constructed comparative experiments. Most importantly, each experiment has so far only been run once, so conclusions cannot be drawn as to the generality of the observed results across a wide range of random number seeds.

For this reason, the runs are not analysed in great detail. However, they are described primarily to demonstrate that the behaviour of the system is somewhat different to Tierra. This was expected, and emphasises the fact that an important factor governing the behaviour of these systems is the specific design of the language in which the programs are written, and the rules governing how they interact with their environment.

In these runs, a grid size of 50 x 50 was used. The grid was initially inoculated with 900 identical ancestor programs, evenly distributed across an area of 30 x 30 positions in the centre of the grid. The ancestors perform more or less the same actions as does the Tierran ancestor described in [15]—the general procedure is to first calculate (by template matching) the start and end points of the genome in memory; then to copy instructions one at a time from the beginning of the genome to the end into the Nucleus Working Memory (this section of the program will be referred to as the 'copy loop'); and finally to issue a *divide* instruction to create a new organism object with a genome constructed from the contents of the Nucleus Working Memory. At each timeslice, each cell in the population was allowed to execute 10 instructions (if it had enough Energy Tokens). Each run lasted for about 1 million timeslices[10].

Comparing the results of the runs across a number of measures, Runs B and C gave qualitatively similar results, but these were fairly different to the results of Run A. Because of the similarity between B and C, the following discussion will describe Runs A and B only. In none of these initial runs did multicellular organisms evolve in significant, sustained numbers. For this reason (i.e. the majority of organisms were single-celled), the terms 'organism' and 'cell' are used more or less interchangeably in the following discussion.

The runs only differed in one respect (apart from the random number seed)—the way in which energy tokens were distributed around the environment at the start of each timeslice[11].

In Run A the distribution was even, i.e. every grid position was given the same number of energy tokens. Ten tokens were deposited at each position at each timeslice, which could enable a cell to execute ten instructions. Therefore, as long as there is only one cell at a given position, that cell can obtain sufficient energy from the environment to survive indefinitely, without having to move around in search of more energy tokens.

In Run B, the grid was divided into five bands of 10 x 50 positions for the purposes of energy token distribution. Grid positions in the leftmost band received 8 energy tokens per timeslice, and each band to the right of this received one more token per timeslice (so the middle

[10] As there were, on average, about 300-400 cells in the population throughout the runs (Figure 1), the system therefore executed about (300 or 400)x10x1000000, or 3-4 billion individual instructions, during the run. This took approximately 100 hours of processor time on a Sun Sparc 4 workstation.

[11] Also, Runs A and C lasted for 1 million timeslices, whereas Run B, running on a slower machine, was terminated after 880,000 timeslices.

band received 10, and the rightmost band received 12). The total number of energy tokens deposited in the environment at each timeslice was the same as in Run A, but in Run B grid positions in the two leftmost bands received fewer tokens than average, and those to the right received more than average. The particular distribution used meant that 60% of the grid positions received sufficient (or more than sufficient) energy tokens to support a single cell, but 40% received insufficient energy.

Figure 1 shows how the number of cells in the system varied over time. Although the upper limit on the number of cells in the system was set at 2500, most of the 900 ancestral programs died off almost immediately in both runs. This was due to overcrowding, as each grid position can only support a single cell. When a cell divides, its offspring is placed at a random nearby grid position, so the environment can only support populations where there is some space between individual organisms (at least in the case where these organisms are immobile). In Run A, the population size stabilised at around 450 organisms (at least until timeslice 700,000), and in Run B it stabilised at around 260-270 organisms. (The fact that Run B supported about 60% of the number of organisms supported by Run A is a consequence of the different energy token distributions, mentioned previously.)

Figure 2 shows how the length of genomes in the population varies over time[12]. (The length of the initial ancestor program is 396 bits, i.e. 66 6-bit instructions.) One difference between the behaviour of Cosmos in these runs and that reported for runs on other systems [15, 16, 22, 21, 18] is the fact that in Cosmos, at each timeslice, all of the genomes are of roughly the same size—there are no parasites or symbiotes of much shorter length (as are often observed in the other systems). This result was, of course, expected, as Cosmos does not allow cells to execute the code of other cells.

There are a few more points of interest about Figure 2. In both runs, over the first 50,000 to 100,000 timeslices, there was a tendency for program length to *increase*, and there is considerable diversity in the lengths of programs in the population at any given time. This increase in program length is accompanied by a decrease in fecundity—the programs are replicating at a slower rate (Figure 3). This is a surprising result, as, recalling the discussion at the end of Section 2, one would ordinarily expect programs in such a system to evolve in the direction of *higher* fecundity (at least, this is the general behaviour observed in runs of Tierra). A closer look at how the programs changed during this period reveals that extra *et_collect* instructions were being inserted into the programs' copy loops. The *decrease* in a program's overall fitness due to the increase in program length that results from the addition of extra instructions is evidently more than compensated by the *increase* in fitness due to the collection of more energy from the environment (the more stored energy a program has, the less likely it is to die[13]).

However, at around about 100,000 timeslices (slightly earlier in Run A, later in Run C), there is an abrupt change to organisms of much shorter length. This occurs when a mutation creates a program without many of the initial instructions which are concerned with the calculation of the size of the genome, which actually turn out to be redundant due to various details of the memory addressing scheme used and the particular actions of some of the instructions. Once this transition has occurred, the length of the programs remains fairly stable throughout the rest of the run. Although this general pattern was observed in all three runs, the actual lengths of programs after the transition were slightly different in each case—in Run A the programs settled in the range of roughly 310-320 bits, in Run B it was 330-350 bits, and in Run C, 270-290 bits. It is also known that it is possible to write considerably shorter self-replicating programs (a self-replicator of length 126 bits has been hand-written by one of the authors [TJT]), yet in each of these runs there was no gradual decrease in length once the initial transition was made. This observation emphasises the fact that some fairly significant details of the results (in this case the lengths of the programs at the end of the run) depend upon chance events (in this case the particular mutation that caused the initial transition to shorter programs). The population certainly does not march inevitably to some sort of global optimum state.

One more point about Run A (left side of Figures 2 and 3) is that, roughly between timeslices 700,000 and 850,000, the population completely lacked any individual organisms that were able to make exact copies of themselves. A closer look at the programs that were around during this period shows that they generally retained most of the code required for self-replication, but with minor errors that prevented them from replicating correctly. Importantly, however, they still generally contained a loop (the copy loop inherited from their ancestors) with many *et_collect* instructions within it, so the programs generally had high energy levels and were therefore unlikely to be killed off. The total number of organisms in the population was slightly depressed during this period (Figure 1), but not by a great amount. At around timeslice 850,000 a mutation occurred which reintroduced faithful self-replicators into the population.

Figure 3 shows how the length of time between successive replications of a cell (i.e. the speed with which

[12]In this figure, at each timeslice data is only included for *self-replicating* cells, i.e. those which had made at least one faithful copy of themselves by that time. In this figure, and also in Figure 3, the darkness of the plot at any given point corresponds to the number of cells taking that ordinate value at that timeslice.

[13]The time of death of individual organisms was not recorded for these runs, so graphs of organism longevity against time (which might be expected to rise if this explanation is correct) cannot be plotted. This will be corrected in future runs.

a cell replicates) varies over time[14]. Whereas in Run A (left side of Figure 3) the replication period is fairly static after 100,000 timeslices throughout the rest of the run, in Run B (right side of Figure 3) and in Run C, there was a fairly gradual *increase* in replication period (i.e. a *decrease* in fecundity) over the run. This occurred despite the fact, as mentioned earlier, that the fecundity of organisms might be expected to increase over time in systems such as this, and also despite the fact that the *length* of the programs remained fairly constant during the run. Again, inspection of the programs over this period shows that there was a gradual accumulation of *et_collect* instructions within the copy loop. As the length of the programs remained fairly constant, it appears that most of these new *et_collect*'s came about by the mutation of existing (apparently redundant) instructions in the programs. There was a small trend for an increase in the number of instructions contained in the copy loop over time (the new instructions generally being even more *et_collect*'s), which accounts for the gradual increase in replication period.

5 Conclusions and Directions for Future Research

The results of these initial exploratory runs of the system demonstrate, if nothing else, that Cosmos behaves somewhat differently to systems such as Tierra and Avida. This was expected, because of the differences in design highlighted in Sections 2 and 3. The results also provide encouraging signs that Cosmos is capable of displaying diverse behaviours under different conditions, and that in many cases the programs do not seem to be simply evolving in the direction of increased fecundity (as is the usual observation in other systems of this type). Much of the interesting behaviour of the results reported seems to be due to the fact that energy is a commodity to be collected and used. As the organisms are responsible for energy collection, they have some control over their expected lifespan (their longevity), which is certainly also the case in the evolution of biological organisms. Programs in Cosmos also have some control over their copy-fidelity, as the rate at which flaws occur as a program runs is also an evolvable parameter of each program. Future experiments on Cosmos will look in detail at the general nature of the relationship between replicator longevity, fecundity and copy-fidelity in evolving populations.

A series of more detailed and careful experiments with Cosmos is shortly to begin. One important question to consider right at the start is how much of the behaviour of the system is due to chance events. In other words, how much do results vary when running the system a number of times under exactly the same conditions (except for a different random number seed)? It is vital to have some idea of this variability in order to know how many trials should be conducted for each set of parameter settings in future experiments. The role of chance events in determining the behaviour of the system may have been particularly influential in the runs reported in this paper, as the fairly small population sizes will have promoted genetic drift. Tests will be run to gauge the magnitude of this effect.

Experimentation will then concentrate on the investigation of a number of theories which have been proposed to explain the initial emergence of multicellular biological organisms. In addition to Stanley's theory [19] of the evolution of heterotrophs as the prime cause of the Cambrian explosion, developmental biologist Lewis Wolpert has suggested that multicellular organisms might originally have emerged in conditions where food was sparsely distributed in the environment[15]. When no food was available, a multicellular organism would be able to begin eating its own cells to survive until environmental food was available again. Cosmos may be easily configured to test such a scenario.

Experiments are also planned to investigate the sensitivity of the system to the genotype-phenotype mapping, for reasons mentioned in Section 3. At present, there are 61 instructions in the Cosmos instruction set, and these are encoded using 6 bits (giving a total of 64 different possibilities). There is therefore virtually no redundancy in the encoding, in contrast to the biological genetic code which encodes 20 amino acids with 64 possible codons. In one set of experiments, a reduced instruction set will be investigated, which consists of just 21 primary units which can be encoded on the genome. The full functionality of the existing system is maintained by allowing the primary units to form compound instructions. This is somewhat analogous to the way in which biological genomes encode just 20 amino acids, which, when decoded, are then assembled into a vast array of useful proteins.

Some exploratory runs were conducted with a much shorter ancestor than was used in the experiments reported in Section 4. The outcome of these runs was that very little evolution happened at all. It appears to be necessary to inoculate the system with an ancestor that contains a certain amount of redundancy (as was apparently contained in the longer ancestor used in Section 4). Indeed, this was also found to be true in Avida, for which it has been reported that "redundancy has emerged as a necessary requirement for successful evolution" [1]. This may be even more true of Cosmos, as it attempts to model cellular organisms at the verge of a Cambrian ex-

[14] In this figure, at each timeslice data is only included for self-replicating cells with 100% copy-fidelity (i.e. those that had only ever produced *exact* copies of themselves). This restriction is due to the way in which the replication rate figures were collected, and will be corrected in future runs.

[15] This theory, which he named 'cannibalistic altruism', was discussed during a recent talk by Wolpert at the Royal Museum of Scotland, Edinburgh, on 20 February 1997.

plosion of complexity and diversity. It could be that supplying the ancestral cells with just a self-replication algorithm in the genome is not enough. Many of the necessary genetic regulatory networks involved in the Cambrian explosion of biological organisms conceivably already existed before the Cambrian period, so that the rapid evolution of the organisms was triggered by coming across ways to regulate these networks and adjusting the degree of pleiotropy between their phenotypic effects. To facilitate the emergence of complex organisms in Cosmos, it may be that the ancestral genome not only has to be large, but must also be composed of a number of discrete functional units. Ideas such as these are discussed in a general context by Wagner and Altenberg in [23].

Throughout this paper, the issue of the importance of remote execution of code for the evolution of parasites has been raised a number of times. Experiments will be conducted on Cosmos in which cells *can* read and execute the genomes of other cells in the system. These conditions would be expected to encourage the evolution of parasites (i.e. to replicate the results observed in Tierra, Avida and Computer Zoo).

As a closing remark, the Cosmos system has been designed and developed over the course of a year or so. When re-reading Ray's original description of Tierra [15] recently, it was of interest to note that in the final section, "Extending the Model", Ray suggests a number of ways in which Tierra could be extended. These include

1. Making instructions expensive.
2. Modifying the way CPU time is allocated.
3. Separation of the genotype from the phenotype.

The incorporation of each of these features in Cosmos came about through largely independent lines of thought (Ray's suggestions having been initially overlooked), but it is satisfying to note that there is some agreement on how to extend such systems.

Acknowledgements

Tim Taylor would like to thank Tom Ray and Kurt Thearling for their comments on the system as it was being developed. He gratefully acknowledges support from EPSRC grant number 95306471. The facilities used for this work were provided by the University of Edinburgh.

References

[1] C Adami. Learning and complexity in genetic auto-adaptive systems. *Physica D*, 80(1-2):154–170, 1995.

[2] C Adami and CT Brown. Evolutionary learning in the 2D artificial life system 'Avida'. In R Brooks and P Maes, editors, *Artificial Life IV*, pages 377–381. The MIT Press, 1994.

[3] PR Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, 1995.

[4] P Coveney and R Highfield. *Frontiers of Complexity: The Search for Order in a Chaotic World*. Faber and Faber, 1995.

[5] R Dawkins. *The Extended Phenotype*. WH Freeman, Oxford, 1982.

[6] R Dawkins. *The Selfish Gene*. Oxford University Press, Oxford, 2nd edition, 1989.

[7] BC Goodwin. *How the Leopard Changed its Spots: The Evolution of Complexity*. Weidenfeld and Nicolson, London, 1994.

[8] JR Koza. Artificial life: Spontaneous emergence of self-replicating and evolutionary self-improving computer programs. In C Langton, editor, *Artificial Life III*, pages 225–262. Addison-Wesley, 1994.

[9] R Losick and D Kaiser. Why and how bacteria communicate. *Scientific American*, 276(2):52–57, February 1997.

[10] REF Matthews. *Plant Virology*. Academic Press, San Diego, CA, 3rd edition, 1991.

[11] RM May. *Stability and Complexity in Model Ecosystems*. Princeton University Press, 2nd edition, 1974.

[12] J Maynard Smith. *Evolutionary Genetics*. Oxford University Press, 1989.

[13] AN Pargellis. The spontaneous generation of digital 'life'. *Physica D*, 91:86–96, 1996.

[14] RA Raff. *The Shape of Life: Genes, Development and the Evolution of Animal Form*. University of Chicago Press, 1997.

[15] TS Ray. An approach to the synthesis of life. In Langton, Taylor, Farmer, and Rasmussen, editors, *Artificial Life II*, pages 371–408. Addison-Wesley, Redwood City, CA, 1991.

[16] TS Ray. Evolution, complexity, entropy and artificial reality. *Physica D*, 75:239–263, 1994.

[17] TS Ray. An evolutionary approach to synthetic biology: Zen and the art of creating life. *Artificial Life*, 1(2):195–226, 1994.

[18] J Skipper. The computer zoo—evolution in a box. In FJ Varela and P Bourgine, editors, *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 355–364, Cambridge, MA, 1992. MIT Press.

[19] SM Stanley. An ecological theory for the sudden origin of multicellular life in the late Precambrian. *Proc. Nat. Acad. Sci.*, 70:1486–1489, 1973.

[20] TJ Taylor. The COSMOS artificial life system. Technical report, Department of Artificial Intelligence, University of Edinburgh. In Preparation.

[21] K Thearling. Evolution, entropy and parallel computation. In W Porod, editor, *Proceedings of the Workshop on Physics and Computation (PhysComp94)*, Los Alamitos, November 1994. IEEE Press.

[22] K Thearling and TS Ray. Evolving multi-cellular artificial life. In R Brooks and P Maes, editors, *Artificial Life IV*, pages 283–288. The MIT Press, 1994.

[23] GP Wagner and L Altenberg. Complex adaptations and the evolution of evolvability. *Evolution*, 50(3):967–976, 1996.

Appendix - Parameter Settings for Runs Reported in Section 4

Size of Grid = 50 x 50, Max Cells Per Process = 2500, Max Cells Per Org = 16, Ancestor type: LA1, Inoculation scheme: 30 x 30, Overlap Type = Overlap, Distribution Type = [Runs A&B:Land, Run C:Mixed], Distribution Max Delta = [Run A:0.0, Runs B&C:0.2], Energy Sharing Type = Shared, Apply Flaws = true, Default Flaw Rate = 10, Mutation rate = 1 in 100000 per 5 timeslices, MulticellularityPenaltyFactor = 1.0, EnergyTokenStoreLowerThreshold = 1, NumOfEnergyToksPerGridPosPerSweep = 10, NumOfEnergyToksPerCollect = 10, MaxEnergyTokensPerCell = 500, MaxEnergyTokensPerGridPos = 200, NumOfInstructionsPerTimeSlice = 10.