PhD Proposal
A Study of Evolution in Self-Replicating Parallel Computer Programs

Tim Taylor
(timt@dai.ed.ac.uk)

Supervisors: John Hallam, Peter Ross

28 June 1996

1 Introduction

Our understanding of the evolution of living organisms, and attempts to derive general theories of evolution, are hampered by the fact that we only have one example of life to study—life on Earth. There are therefore many situations where we are uncertain whether a particular feature of life is general to any comparable system of evolving self-replicators, or whether it is specific to the biological organisms on our planet.

Within the field of evolutionary genetics, there are many situations where two or more plausible theories offer different predictions of how evolution may unfold in a particular case. For example, the 'Red Queen' hypothesis [Van Valen 73] suggests that, even in a constant physical environment, evolutionary change will continue indefinitely. However, alternative arguments may be constructed which lead to the opposite conclusion—that in such circumstances evolution would eventually slow down and stop when each species reaches a local selective optimum (see [Maynard Smith 89]). With only one example of evolution to go on, it is usually impossible to choose between such theories on an empirical basis.

Within the last 6–7 years, a number of researchers have started investigating the idea of creating alternative examples of evolution, in the digital medium, to assist in the formulation of generalised theories. Such work involves the creation of a computer environment in which large numbers of programs compete with each other for the resources required to make copies of themselves. This research will be described in more detail in Section 2.

The following quote from the American biologist Tom Ray, who pioneered this type of research with his `Tierra` system [Ray 91], demonstrates the utility of this work.

> "Diverse ecological communities have emerged. These digital communities have been used to experimentally examine ecological and evolutionary processes: e.g., competitive exclusion and coexistence, host/parasite density dependent population regulation, the effect of parasites in enhancing community diversity, evolutionary arms race, punctuated equilibrium, and the role of chance and historical factors in

> evolution. This evolution in a bottle may prove to be a valuable tool for the study of evolution and ecology." *Tom Ray*[1]

`Tierra` and similar programs are sometimes referred to as Artificial Life (or A-Life) systems. "A-Life" is a somewhat poorly defined term, but, broadly speaking, it describes an inter-disciplinary field of research which seeks to further our understanding of living organisms through the analysis, simulation and synthesis of processes which appear to be necessary for the emergence of life. A-Life research therefore covers experimental and theoretical work in areas such as evolution, self-organisation and complexity. As the field has developed in the last decade, new theories relating self-organisation and complexity studies to evolution have been proposed—`Tierra` and similar systems may help us to refine these theories as well as those of more traditional evolutionary genetics.

It is my intention to create a software environment which supports evolving self-replicating programs in the same spirit as `Tierra` and other existing systems. However, many of the details of my environment will be radically different to anything that has been implemented before. These differences stem from a desire to create a system with mechanisms which correspond more closely to those of biological self-replicators (although still greatly simplified). They will enable the system to address a number of questions which existing systems cannot. These include questions relating to the interactions of embryology, developmental processes and multicellularity with evolution, and also questions relating to life-cycles and life-span. The system, called `COSMOS` (standing for COmpetitive Self-replicating Multicellular Organisms in Software), is described in Section 3.

The main purpose of `COSMOS` is therefore to provide a better tool for studying evolution, ecology and embryology. In addition, there are a number of novel technical features of the system, an analysis of which will be of relevance to certain fields of computer science. These features are described later in the paper, and summarised in Section 6.

A discussion of some of the more general ideas and motivations behind my research is presented in [Taylor 96b]. The following section provides a brief introduction to the existing work in this area.

2 Review of Field

Along with the work of Tom Ray and colleagues with their `Tierra` system (e.g. [Ray 91], [Ray 94]), a number of other systems have also been developed. These include the `Avida` system of Chris Adami and Titus Brown [Adami & Brown 94], and the `Computer Zoo` system written by Jakob Skipper [Skipper 92][2].

The basic idea of these systems is that a virtual operating system is provided, complete with a simple machine language. The machine language instructions are designed to be robust under the mutation and evolution of programs written in the language. This robustness is achieved chiefly through the use of relative or template-driven[3] addressing in branching instructions

[1] Taken from a general introduction to `Tierra` written by Tom Ray and available on the WWW at `http://www.hip.atr.co.jp/~ray/tierra/whatis.html`.

[2] A recent paper [Pargellis 96] has described work on another system derived from `Tierra`. However, this research was specifically looking at the *spontaneous emergence* of self-replicating programs in a 'sea' of random instructions, rather than at the subsequent evolution observed.

[3] With template-driven branching, a branching instruction is followed by a template (a sequence of bits). The

(rather than absolute addressing), and by avoiding the use of explicit memory addresses as operands to instructions. In such an environment, the percentage of functional programs in the space of all possible programs is greatly increased, and the chances of mutating a functional program and coming up with another program that also works are much higher.

Although the details of these systems differ somewhat[4], the general mode of operation is the same. An "evolutionary run" commences with the introduction of an *ancestor* program into the otherwise empty memory. The ancestor program has been hand-written by the researcher to produce another copy of itself in the computer's memory when the program is run. A small element of stochastic behaviour is usually associated with the execution of the machine instructions, e.g. an ADD instruction which usually adds one to its operand may occasionally add zero or two instead, or a COPY instruction may sometimes mutate a byte of the data it is copying. Alternatively, the programs may be subject to point random mutations at a given low rate. In either of these ways, as a run proceeds variations of the ancestor program begin to appear. If a variation retains the ability to produce a copy of itself, then it too may be retained in the population of programs over a number of generations. As the available memory begins to fill, some sort a 'reaper' operation is performed to kill off a number of the programs.

A number of interesting results have been obtained from such evolutionary runs, for example the appearance of "parasites"—short pieces of code which run another program's copying procedure in order to copy themselves. It may be true to say that most of the interesting behaviour that has been seen to evolve has done so because facilities for these behaviours were engineered into the original language specification. For example, the fact that parasites emerge is a little less surprising when the mechanisms of template-driven branching are considered. However, what all of these studies have demonstrated is that it *is* possible to build a robust operating system in which non-trivial self-replicating computer code can evolve.

Kurt Thearling and Ray have also reported work in which they extended the standard `Tierra` system to allow programs to run parallel processes [Thearling & Ray 94], [Thearling 94]. (Throughout this paper I will refer to this work as Parallel `Tierra`). This has used a shared memory model of parallelism, and experiments so far have resulted in the evolution of only SIMD (Single Instruction, Multiple Data) programs. The evolution of more complicated forms of parallel program is one of the major issues I wish to address with `COSMOS`. More details are given in Section 3.

John Koza has proposed a system of self-replicating programs using a slightly different approach [Koza 94]. Koza's programs are boolean trees whose nodes may be standard logical functions (i.e. AND, OR, NOT) or a number of extra functions which not only produce a boolean result but also have side effects which act upon the program itself or upon other programs in the computer's memory. For example, there is an ADD function which will search the rest of the memory for a program matching a given template, and, if one is found, will have the side effect of substituting this new program in the position where the ADD instruction resided in the original program.

One result of this approach of using boolean functions with side effects is that a program may perform a boolean calculation as well as having the side effect of producing another copy

operating system will search for the nearest matching template in the rest of the code and move the instruction pointer to that position.

[4]Programs in `Avida` and `Computer Zoo` also have the notion of a location in a 2D space (in `Computer Zoo` the programs can actually move around this space). This notion was introduced mainly to provide a form of genetic isolation, and also to facilitate efficient implementation on parallel hardware.

of itself somewhere else in memory. Koza encouraged his programs to perform specific boolean calculations by imposing a fitness function on the system—a program has a good chance of proceeding to the next generation only if it is able to self-replicate *and* performs well on the target boolean function[5]. In this respect, Koza's system is very similar to `Avida`, where programs can gain more CPU time by successfully completing user-specified tasks.

While it is perfectly possible for a program on, say, the `Tierra` system to perform other functions *as well as* reproducing itself, this hasn't been observed yet, presumably because there is no obvious way in which a program's fitness may be increased by doing tasks which do not affect its ability to reproduce. In fact, we might expect any such 'additional' functionality to actually *decrease* the program's fitness, as it would have the effect of wasting CPU time which could otherwise have been utilised by the program to produce more copies of itself.

In this important sense, `Avida` and Koza's system differ from `Tierra` and `Computer Zoo`; the latter pair of systems allow programs which perform complicated functions which may not be directly related to self-reproduction, but *only* if, by however roundabout a route, such functions in some way increase that program's longevity, fecundity or copying-fidelity relative to the other programs in the system (i.e. increase the program's fitness). In contrast, in Koza's system and in `Avida` there is actually a *specified* function (or set of functions) which the programs must perform *as well as* having to be able to reproduce. In this way, there are effectively two fitness measures, the former being completely unrelated to a program's ability to self-replicate.

The reasons for Koza's and Adami et al.'s decisions to build such systems are not explicitly stated in their papers, but presumably stemmed from a desire to incorporate the potential evolutionary power of self-replicating entities within a framework where the evolving programs were actually performing a task specified by the researcher so that they were doing something *useful* as well as reproducing. This may well be a promising line of research, but the result of trying to tailor the system to perform a user-defined task is that it becomes less suited for studying general principles of evolution. It is also unclear that effectively imposing two fitness functions on a program will result in particularly efficient solutions for either task. The subject of adding extra functionality to self-replicating programs is an important issue, and is discussed further in [Taylor 96b].

In addition to the systems already described, this section would be incomplete without mention of the ideas of Douglas Hofstadter, some ten years ahead of `Tierra` [Hofstadter 79]. Hofstadter proposed a system called `Typogenetics`, in which he "tried to capture some ideas of molecular genetics in a typographical system". In this system, a strand of letters encoded a sequence of operations (such as copying, cutting etc.) which were to be performed upon the strand itself. The `Typogenetics` system was later developed and analysed by Harold Morris in his PhD dissertation [Morris 88], in which he showed that it was possible to produce several types of strand which were capable of self-replication. If the interpretation and processing of strands is handled on a computer, then we have a system of self-replicating computer programs, albeit with a somewhat different representation and interpreting process to those used in the other systems described in this section.

The research that has been described in this section can be divided into three categories depending on the type of representation employed for the self-replicators:

[5]The set of primitive computational entities from which programs may be constructed comprised not only eight functions (some of which have been described), but also three terminals, corresponding to three boolean input values. In any given evolutionary run, all the programs are tested for their performance on a specific three-argument boolean function—the *target* function.

1. Functional Representation—Koza's work
2. Procedural Representation—`Tierra`, `Avida`, `Computer Zoo`
3. Other Formal Systems—`Typogenetics`

Although functional representations do have advantages in some applications of this type (for a good discussion of these, see [Fontana 91]), they also have a number of weaknesses. By their very nature, their prime action is to perform a particular computation; any task not directly relating to the computation (such as allocating memory and copying data from one area of memory to another) has to be accomplished by allowing functions to have side-effects (which by itself means that the language is no longer purely functional). Additionally, with any functional representation one has to choose a set of terminal nodes to be used. This choice may have a profound effect on what types of program may evolve, and yet there are few guidelines for actually making this choice.

Consideration of the mechanisms for decoding DNA in a cell suggests that procedural representations are probably a better analogy (particularly parallel procedural programs). The type of formal system used in `Typogenetics` could probably also be classified as a procedural representation, albeit of a fairly unusual kind. While the `Typogenetics` representation does have some good points, it is fairly inflexible in its current form (e.g. it does not allow for interactions between different strands such as exchange of genetic material or general communications). These features could be added, but can perhaps be incorporated more naturally into a more usual procedural representation.

For these reasons I decided to use a procedural approach when developing the language in which `COSMOS` programs will be written. The language is called `REPLiCa` (Robust Evolvable Production Language for Cosmos), and, as described in Section 3, although basically a procedural language, it does contain some unusual features.

3 Description of Central Ideas

A full description of the `COSMOS` system and `REPLiCa` programming language is given in [Taylor 96a], and the reader is referred to that document for details. My primary motivation for writing `COSMOS` was to build a tool which would allow me to investigate the ways in which the path of evolution is affected when individuals must go through a process of development from single-celled zygote to multicellular adult. A related question is how multicellular organisms may have evolved from unicellular ones in the first place, and why they are evolutionarily stable over their simpler ancestors. Many features of the system were designed with these questions in mind. Other features address different topics (such as self-organisation, local interactions between cells in an organism, and co-evolution), many of which are actually related in some way to these central questions.

The most important feature of `COSMOS` which distinguishes it from existing systems is that it has been designed specifically to look at the evolution of MIMD (Multiple Instruction, Multiple Data) parallel programs. I am particularly interested in investigating the circumstances under which parallel self-replicators may evolve from and coexist with simpler, serial ancestors, and I will therefore begin initial experiments with a hand-written serial self-replicator. I consider these serial self-replicating ancestors with which the system is inoculated to be somewhat analogous

to the unicellular eukaryotic biological organisms that are thought to have been abundant on Earth prior in the Vendian period of geological time, immediately before the emergence in the fossil record of macroscopic multicellular organisms and the 'Cambrian explosion' that followed. Following this analogy, parallel programs in `COSMOS` may be compared to multicellular organisms. Instructions are included in the `REPLiCa` language to enable a program to dynamically create multiple processes, so that, by using a process of development from serial program (single-celled zygote) to parallel program (multicellular adult), evolution can explore the possibilities of parallelism.

In terms of implementational issues, three of the most important features that distinguish `COSMOS` from existing systems are the 'tag' system (involved in inter-process communications within a single program), the inter-program message passing system, and the energy token system. Inspiration for these features was drawn from biological analogies, although practical considerations limit the extent to which such analogies hold. These and other major differences between `COSMOS` and `Tierra` are described below. For each feature I indicate some of my reasons for chosing a different approach to existing systems. Full details of these features may be found in [Taylor 96a].

Before continuing, I should clarify some of the terms which I occasionally use in the following text. I sometimes use biological terms when describing `COSMOS` as they tend to be more concise and therefore contribute to the readability of the paper. While these biological terms suggest the analogy that I had in mind when designing `COSMOS`, the analogies are certainly not exact—many simplifications and modifications obviously have to be made when designing such a system. With this is mind, the following table lists the meanings I wish to attach to some biological terms in the present context.

Term	*Meaning in context of* `COSMOS`
Cell	A single process in an organism. This term encompasses the host code and any foreign code that may be present, together with associated working memory, buffers and registers.
Unicellular	An organism containing a single cell (process), i.e. a serial program.
Multicellular	An organism containing multiple cells (processes), i.e. a parallel program.
Organism	A single program, which may be unicellular or multicellular.

Table 1: Definitions of some terms used in this paper.

1. Tag System.

   The *tag system* of `COSMOS` has no equivalent in `Tierra`. It was designed specifically to allow cells in a multicellular organism to be able to influence which sections of code were being executed in neighbouring cells, thereby promoting cell specialisation and MIMD parallelism. The design of the tag system was inspired by the processes of chemical signalling between cells in biological organisms.

   A program within the `COSMOS` system is a piece of code written in the `REPLiCa` program-

ming language. REPLiCa programs are divided into sections called *tag blocks* which are each preceded by a precondition expression which must be satisfied before the tag block may be executed. These preconditions relate to the presence or absence of *tags* in the process's *tag store*. A process, or neighbouring processes in a parallel program, may add or delete tags from this store by issuing suitable instructions. As the contents of the tag store changes, different tag blocks in the code become activated and deactivated. In this way, a number of different processes in a parallel program can influence which section of code (tag block) is being executed by any particular process, and a complex regulatory network can be formed.

2. CPU Time Allocation.

   In COSMOS, each cell on a processor is allowed to execute an equal number of instructions during each time-slice. However, each time-slice costs the cell one *energy token* from its store. At the beginning of a time-slice sweep through the system, the COSMOS operating system makes a number of energy tokens available in the environment. It is the responsibility of individual programs to then collect these tokens (by issuing the appropriate instruction) and store them so that they may be exchanged for CPU time at a later date.

   If the number of energy tokens stored by a program falls below a given threshold, the program is killed off. Similarly, when the system's RAM starts filling up, programs are killed off with a probability inversely proportional to the number of energy tokens they have stored. It is therefore a prime importance that a program maintains a certain level of energy tokens in the stores of its constituent processes.

   This mechanism was designed so that COSMOS may more closely resemble the biological case where energy is a commodity which has to be collected, stored and used. It opens up the possibility of having coexisting programs which operate on a range of different timescales; one program may be very short and collect just enough energy tokens to perform self-replication, whereas another program may be much longer, and collect large numbers of energy tokens over a much longer time to enable it to perform more tasks on the way to reproducing.

   In Tierra, programs are given CPU time according to their length. A separate 'reaper queue' mechanism in employed to govern cell death so that programs do not have direct control of their lifespans. Such a reaper mechanism is redundant in COSMOS, where cell death is dictated by the amount of energy stored in the cell. This process imposes fewer constraints on the lifespan of cells and organisms, and will enable COSMOS to investigate questions relating to the interactions between lifespan and evolution which existing systems cannot address.

3. Read, Write and Execute Privileges.

   Tierran programs only have write access within their own 'cell membrane' (apart from when they are in the process of creating a daughter cell, when they also have write access to a specific additional chunk of RAM). A similar situation exists in COSMOS. However, Tierran programs have read and execute privileges to *all* areas of RAM, so that they can directly examine the code of other programs, and even execute this code. COSMOS cells, on the other hand, only have direct read and execute privileges within their own cell membrane, and must rely on the system's communication facilities to interact with other cells. For communication with other processes belonging to the same program, the

tag system is used, as described above. For communication between different programs a different mechanism is used, which is described next.

4. Exchange of Messages and Genetic Information.

   The `COSMOS` mechanisms for the direct exchange of messages and genetic information have no parallel in `Tierra`. This difference is linked to the differences in read, write and execute privileges described in the previous point.

   Any program may compose a message (which can be an arbitrary string of program instructions) and then broadcast this message marked with a particular identification number (the *messageID*). Any program may also receive all broadcast messages of any specified messageID. Messages received in this manner are copied into a special buffer owned by the receiving program, and may then be inspected by the program.

   Although this mechanism doesn't really have a parallel in biological systems, it 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. It will almost certainly be the case that programs within `COSMOS` will not evolve to make even limited use of these facilities for a very long time. However, the general design philosophy has been to make the system as flexible as possible and to try to model as many features of biological systems and their physical environments at least in a very abstract way, so as not to constrain the system's evolutionary potential.

   There is a further twist to this mechanism. Each program itself has an associated identification number (the *cellID*). If a program then receives messages which have a messageID which matches its cellID, then these messages are treated as equivalent to the host code—if the messages contain tag blocks, then their code may be executed within the host cell if their precondition expressions are satisfied. In this way, genetic material may also be transferred between programs. A number of evolutionary options are available to such host cells—they may develop various defences against such exchanges if the foreign genetic material is detrimental to its fitness, or conversely may incorporate the material into its own genome if it is beneficial. Such scenarios are discussed in [Taylor 96a]. Again, the general philosophy in the design of these mechanisms has been to allow the evolutionary process some of the freedoms enjoyed by biological organisms and to prevent it from being unduly constrained.

5. Division Process.

   This point is related to the previous two. As a `COSMOS` cell only has write access within its own cell membrane even when it is composing a copy of itself, this copy must first be composed within the parent cell. The copy is then issued *en masse* to a new memory location.

   In `Tierra` a cell first gets a new block of memory, then writes the copy into this memory, and finally 'divides', signalling that the new memory is now a new organism in its own right.

   There isn't really a great deal of difference between the two mechanisms, but an advantage of the `COSMOS` method is that it allows an organism to *reproduce* (i.e. to create a child organism) and to *grow* (i.e. create a new cell which remains a member of the multicellular organism) using exactly the same technique.

   In contrast, programs in Parallel `Tierra` grow (i.e. initiate parallel processes) by issuing a `split` command. The effect of this is to add an additional CPU to the processor structure

of the program, although all processes still read the single original copy of the program's code. This mechanism is natural for a parallel machine architecture with a shared program space, as used with Parallel `Tierra`. In `COSMOS` memory is not shared across processes in a parallel program, so that such a program must actually physically copy itself into a new process in order to run in parallel. With this type of architecture, it seems preferable that the bulk of such copying work should be performed by the cells themselves rather than by the `COSMOS` operating system.

Additionally, having very similar mechanisms for growth and reproduction of cells is arguably more analogous to the way that multicellular biological organisms may have evolved.

6. Topology of Parallel Programs.

   It has already been explained that processes in a parallel program may pass tags to each other. However, it is not the case that any process can communicate with any other process within the program in this way. Instead, each process maintains a list of neighbouring processes with which it can exchange tags. These lists therefore define a topology for the processes within a parallel program. Additionally, a process may also change the contents of this list, so that it can 'migrate' to a different position within the program.

   This feature was introduced as an attempt to model pattern formation and the physical development of biological organisms from embryo to adult. No existing systems have this type of neighbourhood mechanism, so `COSMOS` will therefore be the first tool of its kind which is able to simulate non-trivial spatial pattern formation during the developmental processes.

7. Local Competition.

   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 programs ([Adami & Brown 94]). Chris Adami and Titus Brown sought to overcome this problem in their `Avida` system by giving each of the programs a location on a two dimensional toroidal grid. Programs can only interact with other programs occupying nearby grid positions, thereby slowing down the rate of propagation of evolutionary changes throughout the total population and promoting heterogeneity.

   `COSMOS` addresses this problem with the use of *localities*. A single locality is a collection of real or virtual processors on which programs may grow, communicate and replicate freely. As far as a program is concerned, the combined RAM of all the processors in the locality is a single, continuous and homogeneous resource within which it competes with other programs. A number of different localities may exist within `COSMOS`, and there is very little interaction between them.

   The only time when information may be exchanged between localities is when a program is about to reproduce. Normally, this will result in a new child organism appearing in the same locality as the parent. However, this child will occasionally be placed into a different locality at random. In this way, an organism's descendants may end up in many different localities, although the rate of dissemination is slow.

8. Size of Instruction Set.

   The `REPLiCa` instruction set is about twice as big as that of the `Tierran` language. This increase has been necessary to accommodate the additional functionality of the system. It

remains to be seen what the effects of this will be.

9. A Serial Ancestor Program.

   In Parallel `Tierra`, the ancestral program was itself parallel. In `COSMOS` it is hoped that parallel programs may evolve from an initial serial ancestor, as described earlier. The probability of a mutation effecting such a transition is not too low, because the mechanisms for growth and reproduction are very similar. `COSMOS` may therefore be used to look at questions concerning the evolution of multicellular organisms from unicellular ones and, in particular, to investigate the advantages multicellular organisms may have in an evolutionary sense over their simpler ancestors.

10. Memory Model.

    `COSMOS` uses a distributed memory model of parallelism, for both practical and theoretical reasons. On the practical side, such a model will allow the system to run efficiently on many different types of machine, which may be stand-alone serial or parallel machines, or networked clusters. On the theoretical side, distributed memory is a closer analogy to the biological situation than is shared memory.

    Additionally, the tag system employed by `COSMOS` should promote the emergence of MIMD programs. This is because the activation and deactivation of sections of code (tag blocks) in any particular process in a parallel program is controlled by the actions of that process and a number of its neighbours. In this way, different processes within the program are unlikely to be executing the same section of code at the same time.

    Parallel `Tierra` uses a shared memory approach, and, although it is theoretically capable of supporting MIMD programs, has so far only demonstrated the evolution of SIMD programs.

11. Memory Addressing Scheme.

    `COSMOS` cells use a local addressing scheme which applies only to code and messages belonging to them. They have no knowledge of their location (or that of other cells) in the global addressing scheme of the processor. This is in contrast to `Tierra`, which uses a global addressing scheme.

    Having incorporated the other features described above, there seemed little point in letting individual programs worry about absolute addressing; the message passing systems should be sufficient to cope with communications between processes and programs, and the use of local addressing also simplifies the process of creating a system which can support programs which are subject to mutation and evolution in a robust manner.

The main purpose of `COSMOS`, as mentioned earlier, is to provide a tool for studying questions relating to evolution, ecology and embryology. A number of specific questions are suggested in [Taylor 96a], and I hope to collaborate with researchers with a more biological background in order to address further topics.

In addition, there are a number of novel technical features of the system, an analysis of which will be of relevance to certain fields of computer science. In particular, the `REPLiCa` programming language uses a novel method for determining control of flow (the tag system), which is somewhat reminiscent of a production system and also has similarities with the PROLOG programming language. My work will include an analysis of this type of language (I use the

term "production language"[6]), and, in particular, a comparison of the REPLiCa language with PROLOG and production systems. The COSMOS system also has relevance to subject of parallelisation of computer code. An analysis of the evolved parallel programs may suggest novel forms of parallelisation for other computer algorithms.

4 Work So Far

- Extensive review of existing work on self-replicating systems, developmental models, self-organisation and complexity, together with familiarisation with relevant topics in genetics, evolution, embryology etc.
- A literature review and discussion of some general ideas relating to my research were presented in [Taylor 96b].
- A concrete design for the COSMOS system has now been decided upon and presented in [Taylor 96a]. Implementational issues have also been decided (e.g. the main system will be written in C++ (using the gnu compiler), with MPI providing message passing services between physical processors).
- I have attended 3 courses at the Edinburgh Parallel Computing Centre (EPCC), and am now familiar with approaches to designing parallel programs and the use of message passing systems (specifically MPI and PVM). These courses have also familiarised me with methods of building applications which are capable of running both on the Cray-T3D parallel computer and on networks of serial computers (including Sun workstations).
- Contact has been made with a number of researchers in the field, including Tom Ray and Kurt Thearling. They seem interested in my work and will hopefully provide comments as things develop.
- I am also in the process of trying to establish contact with more researchers with a biological background, particularly at the Institute of Cell, Animal and Population Biology (ICAPB) here at Edinburgh University, and at Sussex and Oxford Universities.

5 Future Targets

5.1 Implementation and Investigation of REPLiCa language

In [Taylor 96a] I have outlined a number of features of the REPLiCa language which I would like to experiment with in the initial stages of this research. This is in order to see how the tag system works in practice before proceeding with the rest of the COSMOS system.

[6] My definition of a "production language" is somewhat vague at present. I use the term to refer to a language where the flow of control is governed in a manner similar to a production system, but where each 'rule', when 'fired', is capable of performing arbitrary computations as well as potentially causing other 'rules' to be fired. During the course of my work I hope to refine this definition and decide whether "production language" is a useful and appropriate term.

5.2 Search for Analytical Tools

I need to review the evolutionary genetics and complexity literature to look for suitable measures for assessing evolutionary runs and for quantifying evolution. Amongst other things, I need to decide upon a suitable definition of complexity for this system, so that I can meaningfully ask questions such as "Does evolution produce complexity?".

5.3 Implementation of Remaining Core System

This should be fairly straightforward after the `REPLiCa` interpreter has been written, as a detailed design has already been decided upon [Taylor 96a].

5.4 Final Design and Implementation of Monitoring Tool

The core `COSMOS` system will run as a stand-alone application. In addition, a separate monitoring tool, much like a debugger, will be written to enable a user to inspect the progress of an evolutionary run whilst it is happening. The details of this tool remain to be decided. I plan to write the tool in Java to take advantage of its powerful platform-independent window and GUI classes.

5.5 Running experiments

In [Taylor 96a] I have outlined a number of experiments to be performed. Some of these are just concerned with testing the system's sensitivity to various parameter values and to other fairly arbitrary design decisions, while others address more interesting questions about how evolution is proceeding.

As this research progresses I plan to make contact with more people who are actively working in evolutionary genetics and related fields, who may be able to suggest further questions which `COSMOS` can be used to address.

5.6 Installation of `COSMOS` on the Cray-T3D

It is my intention to write `COSMOS` in a platform-independent way right from the beginning, so that the source code may be trivially ported from a Sun environment to the Cray-T3D parallel computer at EPCC.

After obtaining some initial results from runs on networks of Sun workstations, I will apply for time on the Cray so that I can perform some more ambitious experiments.

5.7 Submission of work to conferences and journals

There are many relevant conferences and journals to which I would like to submit my work. In the timetable presented at the end of this section I have included submissions to a number of these, in the hope that if this is in writing I may be more likely to actually meet the deadlines! Obviously this list only includes conferences which have been announced at the time of writing. Even so, there are already a large number of relevant conferences being held over a fairly brief

period of time, so it is unlikely that I will actually have the time to submit to all of them (or the money to travel to them). Decisions of which to submit to will be made closer to the submission deadlines. The details of dates and locations are as follows:

- Third International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97) Norwich, UK: 2-4 April 1997 (Submissions deadline 31 Aug 1996)
- "Evolutionary Computation" Journal: Special Issue on Evolutionary Methods for Program Induction. To be published Spring 1997 (Submissions deadline 6 Sep 1996)
- 1997 IEEE International Conference on Evolutionary Computation (ICEC'97) Indianapolis, USA: 13-16 April 1997 (Submissions deadline 1 Nov 1996)
- First International Workshop on Frontiers in Evolutionary Algorithms (FEA'97) USA: 2-5 March 1997 (Submissions deadline 15 Dec 1996)
- International Work-Conference on Artificial and Natural Neural Networks (IWANN'97) Lanzarote, Canary Islands: 4-6 June 1997 (Submissions deadline 15 Jan 1997)
- International Joint Conference on Artificial Intelligence (IJCAI97) Nagoya, Japan: 23-29 August 1997 (Submissions deadline 21 Jan 1997)
- Fourth European Conference on Artificial Life (ECAL97) Brighton, UK: 28-31 July 1997 (Submissions deadline to be announced)
- Artificial Life VI (ALife6): Details to be announced.
- Artificial Life journal: (published quarterly).

5.8 Writing up

A grim task, but someone has to do it! Actually, if I manage to submit papers to most of the conferences and journals listed in the previous section, much of this work can be incorporated into the final thesis.

5.9 Proposed Timetable

The schedule on the next page is an initial estimate of the timetable for this research.

1995–1996	Q4, Q1, Q2	Literature survey. Design of `COSMOS` system.
1996	Q3	Implementation of core `COSMOS` system. Experimentation with `REPLiCa` programming language. Review of possible analytical techniques to be used. Preliminary experiments. Contact with evolutionary biologists. Submission to ICANNGA'97 and special issue of "Evolutionary Computation".
1996	Q4	Final decisions on the main analytical techniques to be used for assessing evolutionary runs. Design and implementation of monitoring system. Experiments with parameters and arbitrary features of system, and possibly more experiments with `REPLiCa` language. Start looking for new experiments to perform in collaboration with other researchers. Submission to ICEC'97, FEA'97, IWANN'97, IJCAI'97.
1997	Q1	Experiments with evolutionary questions. Refinements to system. Start of large-scale analysis. Application for time on Cray at EPCC. Continue looking for new experiments in collaboration with other researchers. Submission to ECAL'97.
1997	Q2	Installation of system on Cray. Continue looking for new experiments in collaboration with other researchers. More analysis. Start of runs on Cray?
1997	Q3, Q4	More runs on Cray. More new experiments? Submission of work to ALifeVI and to "Artificial Life" journal.
1998	Q1, Q2, Q3	Write up.

6 Likely Outcome

By the end of this research project I will have made contributions to the following areas:

1. *Theoretical issues of evolution, ecology and embryology.* `COSMOS` will be able to add to findings of existing systems, and also to tackle many questions which existing systems have not been able to. In particular, `COSMOS` will be able to address questions of how developmental processes affect the path of evolution, and in what ways multicellular organisms may have evolved incrementally in such a way that they are evolutionarily stable over simpler unicellular organisms.

2. *Automatic program induction.* Most current research on evolutionary approaches to automatic program induction uses LISP-like representations. `COSMOS` uses a very different representation, and is also one of a very small number of systems which can produce parallel programs by evolution.

3. *Production languages.* The `REPLiCa` programming language is an implementation of what I call a Production Language. My work will include an analysis of the dynamics of this type of language, and a comparison of the `REPLiCa` language with PROLOG and production systems.

4. *Parallelisation of computer code.* An analysis of parallel programs evolved in `COSMOS` may be able to suggest novel forms of parallelisation. This knowledge may be transferable to

the design of other parallel algorithms.

References

[Adami & Brown 94] 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.

[Fontana 91] W Fontana. Algorithmic chemistry. In CG Langton, C Taylor, JD Farmer, and S Rasmussen, editors, *Artificial Life II*, volume X of *SFI Studies in the Sciences of Complexity*, pages 211–254, Redwood City, CA, 1991. Addison-Wesley.

[Hofstadter 79] DR Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid*. Basic Books, New York, 1979.

[Koza 94] JR Koza. Artificial life: Spontaneous emergence of self-replicating and evolutionary self-improving computer programs. In C Langton, editor, *Artificial Life III*, volume XVII of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 225–262. Addison-Wesley, 1994.

[Maynard Smith 89] J Maynard Smith. *Evolutionary Genetics*. Oxford University Press, 1989.

[Morris 88] HC Morris. Typogenetics: A logic for artificial life. In C Langton, editor, *Artificial Life*, volume VI of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 369–395, Reading, MA, 1988. Addison-Wesley.

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

[Ray 91] 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.

[Ray 94] TS Ray. Evolution, complexity, entropy and artificial reality. *Physica D*, 75:239–263, 1994.

[Skipper 92] 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.

[Taylor 96a] TJ Taylor. The COSMOS environment and REPLiCa programming language. Departmental Working Paper No. 259, Department of Artificial Intelligence, University of Edinburgh, June 1996.

[Taylor 96b] TJ Taylor. On the incorporation of a developmental process in a system of self-replicating programs. Departmental Working Paper No. 258, Department of Artificial Intelligence, University of Edinburgh, January 1996.

[Thearling & Ray 94] 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.

[Thearling 94] 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.

[Van Valen 73] L Van Valen. A new evolutionary law. *Evolutionary Theory*, 1:1–30, 1973.