

A Model of Gene Expression and Regulation in an Artificial Cellular Organism

Paul J. Kennedy*

*Department of Software Engineering,
Faculty of Information Technology,
University of Technology, Sydney,
PO Box 123 Broadway NSW 2007 Australia*

Thomas R. Osborn

*The NTF Group, Decision Support Consultants,
Level 7, 1 York Street, Sydney NSW 2000 Australia*

Gene expression and regulation may be viewed as a parallel parsing algorithm—translation from a genomic language to a phenotype. We describe a model of gene expression and regulation based on the operon model of Jacob and Monod. Operons are groups of genes regulated in the same way. An artificial cellular metabolism expresses operons encoded on a genome in a parallel genomic language. This is accomplished using an abstract entity called a *spider*. A genetic algorithm is used to evolve the simulated cells to adapt to a simple environment. Genomes are subjected to recombination, mutation, and inversion operators. Observations from this experiment suggest four areas to explore: dynamic environments for the evolution of regulation, advantages of time lags inherent in the expression algorithm, sensitivity of our genomic language, and noncoding regions on the genome. Issues relating to the application of the expression model to evolutionary computation are discussed.

1. Introduction

Gene expression may be viewed as a parsing algorithm—translation from a one-dimensional genomic language of bases to a three-dimensional structural language of proteins. Expression of genes occurs throughout the lifetime of a cell and is an important component of the (ongoing) development of the phenotype (or cell body) from the genotype (cell DNA).

We wish to determine whether the process of gene expression is a useful notion to apply to evolutionary computation (EC). To answer this question we build a model that is able to test hypotheses associated with gene expression and regulation in EC. It is important that the gene

*Electronic mail address: paulk@it.uts.edu.au.

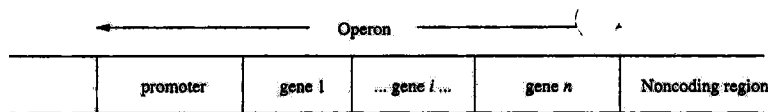


Figure 1. Schematic of operon structure. An operon consists of a promoter (regulatory) region followed by one or more genes, each coding for a protein. All genes in the operon are regulated as a unit.

expression model sits at an appropriate level. It should be an abstraction of biology so as to embody some of the flexibility and complexity of gene expression in real cells and it should also bear close relationships to EC so that it can tell us something about gene expression in the context of EC.

The model we present in this paper, therefore, sits between biology and EC. It is a model of a single-celled organism that inhabits a simple environment. Populations of cells are bred with a genetic algorithm (GA) [6] to adapt to the environment. As in a real cell, the phenotype used in our model varies with time—the genome controls the artificial cellular metabolism at all times.

A typical simulation of one of our cells contains around 185 enzyme-catalyzed reactions. These are encoded into around 700 coupled differential equations each containing between 10 and 25 terms. There are roughly 200 chemical species and 10 enzymes. This degree of simulation may seem to be large, but when compared to a real cell, our simulations are mere caricatures.

In 1961, Jacob and Monod developed the operon model of gene expression and regulation [1, 5, 7] to explain expression in prokaryotic cells. Prokaryotic cells are simple organisms of a single cell. They contain no membrane-bounded organelles such as a cell nucleus. An operon is the smallest section of the genome able to be expressed or regulated. It is a contiguous group of genes that are regulated the same way. An operon consists of a sequence of genes preceded by a regulatory region called the *promoter region* (see Figure 1). In general, each gene codes for one protein or part thereof. In our model, we encode operons onto a bit string.

Our cell model is similar in spirit to work carried out by Rosenberg [14] in 1967. His simulation was, by necessity, simpler than ours because of the limits of the computing machinery of the time. Other, more recent, cell models (e.g., [8]) also share similarities with our model of expression. See [9] for a survey of related cell models.

Recently gene expression models have been developed for EC. The grammatical evolution algorithm of O'Neill and Ryan [13] evolves code in a given language that is provided as input to the algorithm. In contrast, the model described in this paper evolves strings for a fixed

parallel genomic language. Each string (or operon) encoded on our genome acts as a separate control “thread” or program and executes in parallel with other strings. The mechanism in our model that “runs” the strings (i.e., the metabolism) is able to regulate activation of the programs (i.e., operon regulation).

In section 2 the model of a single-celled organism in which our process of gene expression operates is outlined. Following this the model of genome used and complete details of the parallel genomic language encoded on it and our method of parsing this language are given. In section 4 gene expression and regulation is described, initially in terms of a discrete model of “spider” expression and then in a continuous rephrasing more in keeping with the cell model.

After describing our model, we present results from an experiment showing that cells using the gene expression algorithm may be evolved to adapt to a simple environment. Next we discuss four questions that may be answered by the model although we do not answer them here. Finally we discuss how the model may be applied to EC.

It is not the goal of this paper to describe the cell model in more detail than is required to explain the workings of gene expression and regulation.

2. Overview of the cell model

When building a model of a cell with particular focus on gene expression and regulation we feel that it is important to consider the genotype as a string in some well defined genomic language—or some representation of a (pseudobiological) computer program on a (mostly read only) storage medium. In our model (and in biology) the genome is a highly parallel program. The phenotype may be regarded as a kind of memory and computer. In our model, different chemical species (and their relative concentrations) act as a memory and the way that the chemical species interact (*via* reactions in the metabolism) represents the computational machinery. The algorithm that generates a phenotype can be viewed as a sort of parsing algorithm and is part of the metabolism or computational machinery. It takes a program (genotype) in a parallel genomic language, parses the string from this language, and creates computational elements (enzymes) that manipulate the memory locations of the computer (i.e., the concentrations and kinds of chemical species).

The cell model is an abstraction of a biological cell. The cell is assumed to be a single “well-stirred” vessel containing no organelles. Each cell inhabits a simple environment. The environment consists of a very large well-stirred vessel consisting of a small number of simple chemical species held at fixed concentration. No reactions occur in the environment. Some of the chemicals in the environment are simply

building blocks for other chemicals or a species that can be used for expressing genes.

Our genome encodes bases which are tokens of a genomic language. When read in sequence, the bases encode sentences that specify a set of simulated protein molecules able to be generated in the cell. The simulated protein molecules act as enzymes. As in real biology, enzymes permit only a small subset of all the possible chemical reactions to occur in the cell. The entire set of chemical reactions that may occur in the artificial cell is called its *metabolism*. Indirectly, through the enzymes, the genome controls the metabolism and hence the cell. The metabolism, however, regulates the genome (by turning on or off genes) and actually produces the simulated enzyme molecules.

Five main processes are embodied in the simulated metabolism.

- Modification of chemicals by reactions catalyzed by enzymes.
- Production of proteins by gene expression and regulation.
- Degradation of proteins into their constituent monomers.
- Growth and shrinkage of the cell.
- Diffusion of chemicals through the cell membrane.

Chemical polymers are modeled as a string of digits according to the model proposed by Farmer, Kauffman, and Packard [4] and used by Bagley and Farmer [2] and Bagley, Farmer, and Fontana [3]. Each digit represents one of 10 possible artificial monomers. 10 monomers were chosen to permit combination into chemical species of considerable complexity. The chemical “shape” (i.e., the list of digits) specifies how the chemical may interact with other chemicals. The term *chemical species* means all the molecules (in the cell) that have the same (particular) shape.

The chemical reactions we model are an abstraction of polymer condensation and cleavage reactions. Polymer condensation reactions join two polymers into a longer polymer and cleavage reactions cut a polymer into two shorter molecules. Pairs of condensation and cleavage reactions may be viewed as one reversible reaction. The direction of the reaction (condensing or cleaving) depends on the relative concentrations of the long polymer and the two shorter polymers. The number of cleavage and condensation reactions that may occur for a given set of polymer strings is very large, so we model only reactions that are catalyzed by an enzyme. When catalyzing a reaction, the closer the shape of the enzyme is to the shape of reactants (i.e., the more specific the enzyme), the faster the reaction can occur.

The chemical species and the five metabolic processes listed above are encoded into a large system of coupled nonlinear ordinary differential equations.

Associated with the cell is a semipermeable cell membrane. The size of a cell is determined by the number of cell membrane molecules associated with the cell. In this model, cells are assumed to be plump spheres filled with water. The size of this sphere is determined by the surface area of the cell membrane which, in turn, is specified by the number of cell membrane molecules. Growth of the cell occurs as the result of an increase or decrease (shrinkage) in the number of cell membrane molecules and, in turn, the surface area of the cell.

Membrane molecules are not modeled explicitly as a separate chemical species, but their number may be changed under the influence of two families of chemical species. One family increases the number of membrane molecules (“builders”), whilst the other family decreases them (“breakers”). The cell controls the relative concentrations of each of these families to maintain the size of the cell.

A slow background diffusion of chemicals through the cell membrane is modeled. Chemicals that diffuse through the semipermeable membrane are called *permeants*. Proteins are assumed to be too large to pass through the cell membrane so only nonprotein chemical species are affected by diffusion. The speed of diffusion is proportional to the surface area of the cell, to the concentration gradient of the permeant molecule, and inversely proportional to the size of the molecule. The concentration gradient of a molecule is the difference between the concentration of the molecule inside the cell and in the environment. Size of a molecule is estimated as the cube of the number of monomers comprising its length.

3. Model of the genome

A genome consists of a string of binary values. Blocks of four bits are read from the string and are interpreted as tokens from a genomic language. When read in order, the tokens specify the structure of proteins to be used in the cell.

At the highest level, we interpret the genome as a list of operons separated by noncoding regions (see Figure 1). Our language dissociates genes from particular loci.

Four bit blocks are read from the genome bit string. The 16 possible coded words are divided into 10 digits (0000_2 to 1001_2) and six control codes (1010_2 to 1111_2). The 10 digits are used to specify monomers to build into proteins or as parts of keys for gene regulation. The remaining six bases encode four special codes: <start operon> (1010_2 , 1011_2), <start enzyme> (1100_2 , 1101_2), <start carrier> (1110_2), and <end operon> (1111_2). Figure 2 shows an example of how the tokens may be combined to build an operon.

Promoter regions contain a sequence of bases that specify a key used to switch the operon on or off. As in biology, we classify operons

<start operon>	Switch Data	<start enzyme>	5	3	2	<start carrier>	0	1	7	<end operon> or <start operon>
----------------	-------------	----------------	---	---	---	-----------------	---	---	---	--------------------------------

Figure 2. Example of operon structure. The fine grained structure of an operon containing two genes. Regulatory information occurs after the <start operon> code. Following this are protein/gene specifications. The first gene codes for an enzyme protein with shape “532.” The second gene codes for a carrier protein with shape “017.”

into three types depending on how they are regulated: constitutive, repressible, and inducible.

- *Constitutive operons* are always active. They express proteins continually.
- *Repressible operons* are active unless a matching key (i.e., a switch chemical) is present. Gene expression, for these operons, may be repressed under favorable conditions.
- *Inducible operons* are inactive unless an appropriate key is present. That is, these operons may be induced to express genes under the right conditions.

For repressible and inducible operons a key is associated with regulation. Any chemical species in the metabolism may act as a key providing its shape matches the key specification in the promoter region of the operon. The closer the shape of a chemical is to the key specification of an operon, the more readily that chemical will be able to change the activation of the operon.

The type of operon is determined (in our model) by the tokens found in its promoter region. Constitutive operons result when there are less than two numeric bases in the promoter region. An operon is repressible if the first base in the promoter region represents an even number and inducible if the first base is odd. We chose this strategy for determining the type of operon so as to avoid biasing the code towards either inducible or repressible operons. The remaining bases in the promoter region specify the key (or switch chemical) that may be used to regulate the operon. The strength of activation of an operon to a particular switch chemical depends on the closeness of match between the key in the promoter region and the shape of the switch chemical, that is, its specificity.

Gene specifications follow the promoter region. Each gene begins with the <start enzyme> or <start carrier> code. Bases following this code describe the monomers that will make up the shape of the protein. An enzyme is the result of a <start enzyme> gene and a carrier protein arises from a <start carrier> code. The gene

continues until the next special code. If this next special code is <start enzyme> or <start carrier>, another gene begins. If, however, it is an <end operon> or <start operon> code, then the enzyme or carrier protein results and the operon ends.

More bit patterns are used for some special codes than others. This does not cause problems for the GA because selection removes poor genomes. It does, however, stop the encoding scheme from having bit patterns that do not code for valid symbols. Such a situation would unnecessarily expand the genome search space and decrease the efficiency of the GA.

3.1 Parsing the parallel genomic language

The Backus–Naur form description of the grammar for generating valid operon strings is given in Table 1. The language dissociates genes from particular loci. This language is context-free, which means that it may not be parsed by a deterministic finite state automaton. However, it is possible to parse a genome with a *modified* deterministic finite state automaton. The finite state automata accumulates information about the parsing operation in variables. For details of the modified finite state automata see [9]. It is important to stress that we are not implying that

```

<operon> ::= <startoperon> <operonbody>
<operonbody> ::= <promoter> <genelist>
<operonbody> ::= <genelist>
<promoter> ::= <switchtype>
<promoter> ::= <switchtype> <baselist>
<switchtype> ::= <repressible> | <inducible>
<repressible> ::= 0 | 2 | 4 | 6 | 8
<inducible> ::= 1 | 3 | 5 | 7 | 9
<genelist> ::= <endofoperon>
<genelist> ::= <enzyme> <genelist>
<genelist> ::= <carrier> <genelist>
<enzyme> ::= <startenzyme> <base> <baselist>
<carrier> ::= <startcarrier> <baselist>
<baselist> ::= <base> <baselist>
<baselist> ::= <base>
<base> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<endofoperon> ::= <endoperon> | <startoperon>
<startoperon> ::= a | b
<startenzyme> ::= c | d
<startcarrier> ::= e
<endoperon> ::= f
    
```

Table 1. Backus–Naur form statement for the parallel genomic language used to encode genes in an operon. The genome contains strings in this language.

the parsing operation is accomplished with one molecule: the finite state automaton is an abstraction of a group of molecules working together.

4. Gene expression and regulation

Protein production in biological cells is a complex multistage process with stages occurring at different places in a cell. This allows the cell fine control over production of proteins [1]. However, in our model we are interested only in the basic idea that information encoded in a genome guides the creation of proteins and that this production can be regulated. We have, therefore, combined the processes of transcription and translation into one operation that is performed by an entity called a *spider*.

Spiders should be viewed as an abstraction of complex protein machinery that is able to (i) locate the start of operons; (ii) read and identify bases on the genome; and (iii) produce protein strings by traversing the genome. The analogy is of a spider moving along a surface (the genome) spinning a strand of its web (a protein chain).

A reading head is often called a “turtle” in computer science. Turtles may also draw a line or path from their movements through a two-dimensional space. We chose to introduce the new term spider because this entity produces a strand as well as reading.

Although combining transcription and translation into one process simplifies our model considerably, it also limits the kinds of control that may be exerted over the process of gene expression. However, two other forms of indirect gene regulation are also modeled: regulation of the amount of mRNA (or its analogue in our model, spiders) available to the cell, and competition in the metabolism for protein building blocks (amino acids in real cells or the monomers in the range [0, 9] for our model).

4.1 A discrete spider model

Spiders are modeled as a family of similarly shaped chemicals. Speed of initiation of transcription for a spider is proportional to the similarity of the shape of the spider chemical to an arbitrary but fixed “ideal” spider. The motivation behind this “blurriness” of spider shape and transcription ability is to encourage evolution to find spiders. A hard distinction between spider and nonspider makes discovery of spiders by evolution more difficult. The problem of spider discovery in such a scheme becomes akin to the “needle in the haystack” problem which is known to be difficult to solve because neighborhood information (in genome space) is not helpful.

We imagine a population of spider molecules crawling along the genome reading tokens (Figure 3). With each step, a spider adds a

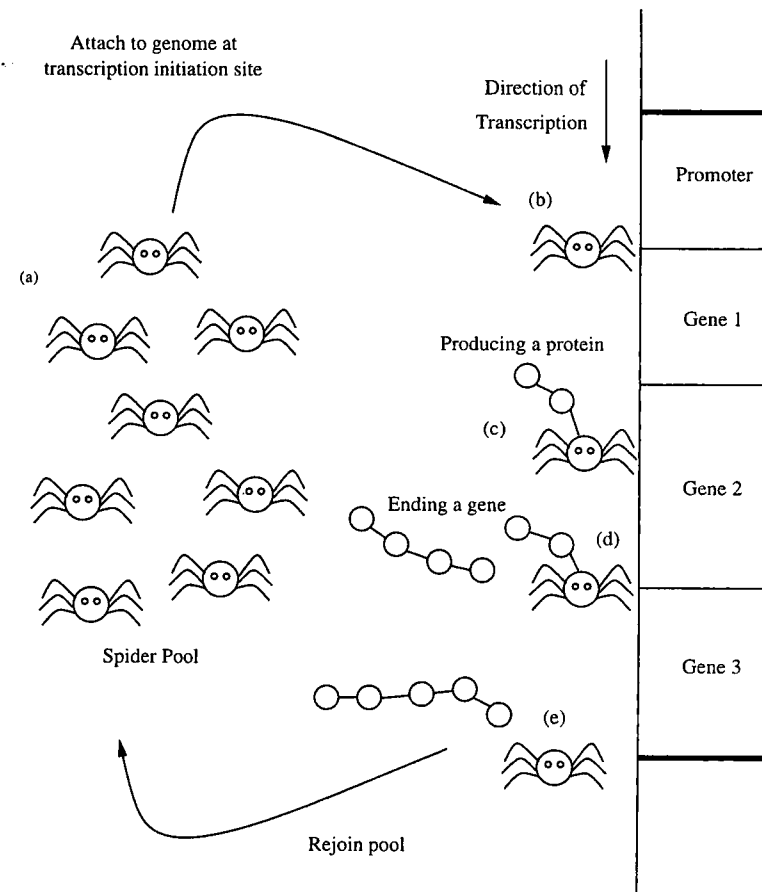


Figure 3. Overview of the discrete model of operon expression with spiders (without regulation). (a) A pool of molecules exist with each molecule having the property of being able to read genes and produce proteins. Each molecule is called a spider. Spiders in the pool are free and are not yet expressing genes. (b) A spider leaves the pool and binds to the promoter region of a random operon. Regulation of operons occurs at this stage of our model but is not shown in this diagram. (c) The spider crawls along the genome. It reads the base at its current position, appends the matching monomer to a growing protein molecule that it is “spinning” and moves to the next base. (d) When the spider encounters the end of a gene it releases the protein into the metabolic soup of the cell. As it starts the next gene in the operon it spins a new protein. (e) When the spider reaches the end of the operon it releases the protein and rejoins the pool.

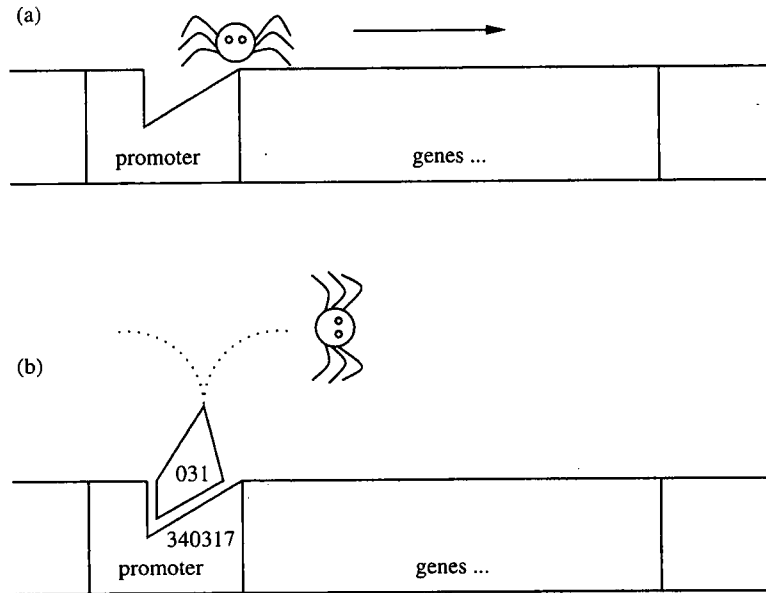


Figure 4. Regulation and activation of a repressible operon. (a) No blocker molecule is present. A spider molecule is therefore able to initiate expression of the operon. (b) A molecule able to bind to the operon promoter region is present. The trimer “031” in the switching chemical binds to the promoter region for a short time. Whilst this occurs a spider is not able to bind to the promoter region and genes may not be expressed.

monomer to a growing protein chain that it creates. A spider may only attach to the genome at the promoter region and leave it at the end of the operon. As it moves from one gene to the next, the spider finishes the protein and releases it into the metabolism. When a spider falls off the end of an operon, it randomly joins another operon. Furthermore, a spider may only attach to an operon if the operon is ready for transcription (switched on).

Constitutive operons (see section 3) are always ready for expression. Spiders may attach to them at any time.

Repressible operons are active unless a “blocker” molecule is bound to the promoter region, in which case they are turned off (see Figure 4). Spiders may only attach to repressible operons if a blocker molecule is not bound to the promoter region at that time.

Inducible operons function in the other way. They are inactive unless an activator molecule is bound to the promoter (Figure 5). Molecules bind to the promoter for only short periods. The more bases matching

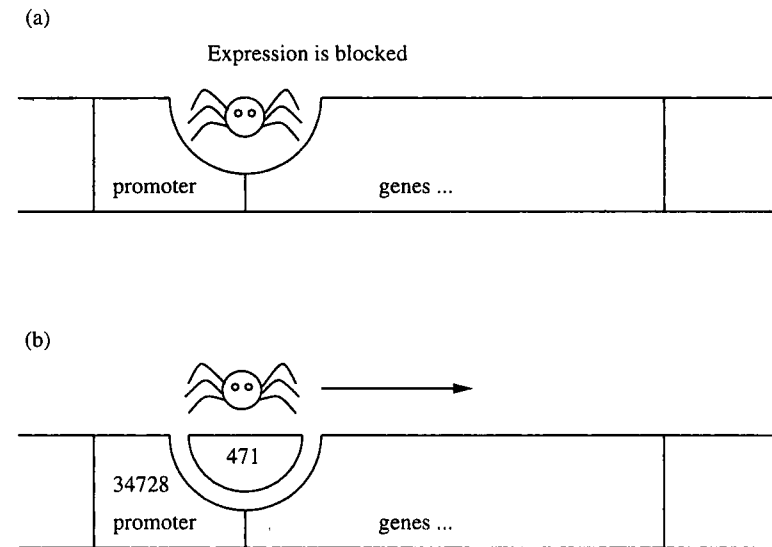


Figure 5. Regulation and activation of an inducible operon. (a) No activator molecule is present, so spider molecules are not able to start transcription. (b) A switch chemical that is able to bind to the promoter region is present. In this case, a spider molecule is able to begin transcription.

between the molecule and the promoter region, the stronger the bond, and the longer the molecule stays attached. The length of time that a switch molecule remains bound to a promoter region is given as a probability distribution (see section 4.2.1).

4.2 A continuous spider model

The gene expression scheme of section 4.1, however, is unsuitable for our cell model because it is discrete, whilst the rest of our model is encoded continuously using differential equations. We therefore recast the discrete model of molecular interactions into a continuous model of chemical reactions.

New chemical species are introduced to represent the amount of spider chemical that is bound to each token position (or base) along the interpreted genome. These “fake” chemical species do not take part in any other reactions. They exist only as a method of representing the position of spiders along the interpreted genome.

Next we define a set of irreversible reactions that represent the steps that the spider makes along the list of tokens. These reactions are irreversible because spiders may only move forward through the list of tokens. It must be noted that these reactions are not related to the

polymer condensation and cleavage reactions described in section 2. The reactions described here are operon expression reactions that build the proteins. They are not enzyme catalyzed.

As the spider moves from one token to the next, the concentration of spider at the previous token decreases a little and the concentration of spider at the new position increases by the same amount.

Each step along the list of tokens, then, is rewritten as an irreversible reaction. The two reactants of the reaction are a spider molecule bound to the current position on the token list (S) and a monomer species (M) that matches the token currently being read. The products of the reaction are the spider molecule bound to the next token in the list (S') and perhaps a protein (P):



A protein (P) is only produced if the current token is the last in a gene. In this respect, the notation in equation (1) is slightly atypical.

The product spider molecule (S') becomes the reactant (S) for the next token in the interpreted genome, thus making a biological pathway with the chain of reactions. S' will be either a spider from the pool or a spider bound to the next position along the interpreted genome. The former case applies when this reaction represents the addition of the last monomer of the last protein of an operon. The latter case represents the situation when there are more genes left in the operon.

4.2.1 Operon activation

In the previous section we described the continuous model of gene expression. We now wish to describe the other half of the model: gene regulation. Regulation depends on the type of operon: constitutive, repressible, or inducible. The regulation is modeled by changing the reaction rate of the first transcription reaction for the operon. That is, the reaction where the spider leaves the pool to start expression of the first gene.

The first reaction in each expression reaction pathway, then, has an additional parameter that gives the probability that the spider may bind to the operon at the current time. This parameter regulates the operon and works similarly to a variable reaction rate. In this case, the rate "constant" for the reaction is $G_i K_T$ where G_i is the activation of the i th operon (in the range $[0, 1]$) and K_T is the transcription initiation rate constant for the spider. For all other reactions in the expression reaction pathway a rate constant of 1 is used so as not to slow down expression. K_T is determined by the degree of resemblance between the particular spider molecule and an "ideal" spider shape.

The chain of reactions for expression of an operon is replicated for each species of spider molecule because each species of spider most probably operates with a different rate K_T . Different intermediary bound

spider species are also used. These partially expressed protein species cannot act as enzymes or ordinary chemicals, interact in any other way with the cell metabolism, or diffuse out of the cell.

Note also that gene expression in our model is noiseless: no bases are misread or the wrong monomer chosen.

The activation of an operon (G_i) depends on the availability of chemicals in the metabolism that can switch the operon on or off. This activation is modeled by calculating the probability that a spider may attach to the start of an operon.

We now describe the calculation of this probability. There is a different probability for each of the three types of operon.

The probability of a spider binding to a constitutive operon is simple to calculate. It is 1 because constitutive operons are always on.

With inducible and repressible operons the probability that a spider may bind to the promoter region depends on the probability of a switch chemical being bound.

For inducible operons, the probability of a spider binding is equal to the probability of an activator chemical being bound to the promoter region at the same time ($\hat{\rho}_n$).

In the case of repressible operons, the probability of a spider binding is one minus the probability of a blocker being bound to the promoter region (i.e., $1 - \hat{\rho}_n$).

Calculation of the probability of a spider binding to the operon, then, reduces to the problem of finding the probability of a bound switching chemical ($\hat{\rho}_n$). Determination of $\hat{\rho}_n$ is more difficult because there may be many (n) such similarly shaped species able to bind to the promoter region. We calculate $\hat{\rho}_n$ by combining the probabilities of a particular chemical species binding to the promoter in isolation (ρ_j), which is simpler to quantify.

The probability that one molecule of a particular chemical species j is bound to the promoter region (ignoring other species) is

$$\rho_j = 1 - e^{-K_j s_j} \quad (2)$$

where s_j is the concentration of species j and

$$K_j = \frac{K}{(1 - q_j) e^{-n_j b}} \quad (3)$$

q_j is the probability that switching chemical j will not immediately bind to the promoter region. This is calculated by examining all the possible ways chemical j can bind to the promoter region and counting all the positions where there are no bonds (i.e., matches) between the chemical and the promoter region. q_j is the ratio of the number of positions with no matches to the total number of positions. For example, consider a switch chemical j with shape "031" and switching information of a promoter region of "710411." There are eight positions where chemical

j may bind to the promoter region, but only three of these positions have at least one bond (match) between bases. In this case, q_j has value 0.625.

The exponential part of the denominator of equation (3) specifies the length of time chemical j will bind to the promoter region. This time follows the Boltzmann distribution and depends on the average number of bonds between the chemical and the promoter region (n_j) and the relative strength of each bond (b , typically 0.25 in our experiments).

The value of n_j is calculated using a method similar to q_j . All the configurations between the chemical species j and the promoter region are examined and the number of bonds for each configuration is determined. n_j is the ratio of total number of bonds in all configurations to the number of configurations. In the example given above with a switch chemical "031" and promoter sequence of "710411" there are a total of four bonds in the three binding configurations. The value of n_j , then, is 1.3.

K is a constant used to calibrate the concentration of a switching chemical with the probability that the chemical will be bound to the promoter region. The actual value used (1.0×10^3) is arbitrary but its general relationship with other parameters is important.

Given the probability of chemical species binding to the promoter region in isolation ρ_j , we may give the probability that a molecule of one of n competing chemical species is bound to the same promoter region as

$$\hat{\rho}_n = \frac{\rho_1 + (1 - \rho_1)I(n)}{1 + (1 - \rho_1)I(n)} \quad (4)$$

$$I(n) = \sum_{i=2}^n \frac{\rho_i}{1 - \rho_i} \quad (5)$$

where n is the number of competing switch chemical species and ρ_i is the probability that a molecule of species i will bind to the promoter region (assuming no other competition). Derivation of the above equation is presented in appendix A.

There is a variable $\hat{\rho}_n$ for each inducible or repressible operon on the genome and derivatives are added to the system of differential equations:

$$\frac{d\hat{\rho}_n}{dt} = \left(\frac{1 - \rho_1}{1 + (1 - \rho_1)I(n)} \right)^2 \sum_{i=1}^n \left(\frac{1}{(1 - \rho_i)^2} \frac{d\rho_i}{dt} \right). \quad (6)$$

Derivation of equation (6) is given in appendix A.2.

5. An experiment evolving cells to adapt to a simple environment

An experiment was developed to test the capabilities of the model. We ask the question: Is it possible to evolve cells that are able to exist

in a simple environment? A GA was used to evolve a population of cells. Each cell is simulated in isolation inside its own environment. The environment causes the cell to grow by bathing it in chemicals that make the cell membrane increase. When membrane building chemicals diffuse into the cell the number of water molecules in the cell increases (because the cell is larger) and the concentrations of chemicals in the cell decrease. This causes reactions to occur more slowly. At the same time, the surface area of the cell increases and this causes diffusion to increase.

The experiment finds pairs of genome and initial metabolic conditions that build cells which remain stable in the simple "growing" environment.

Simulation of a cell involves building the cell from its initial chemical concentrations and genome. Initial chemical concentrations of a cell are derived from the final concentration values of the mother's chemicals, the mother being a random choice between the cell's two parents. Next, the genome is parsed into a list of operons. A graph of chemical reactions in the cell is then determined by matching the operon list with the available chemical species. The reaction graph is encoded into a system of nonlinear differential equations. Additional terms are added to equations for diffusion of chemicals through the cellular membrane.

The system of differential equations is numerically integrated to determine how the cell changes over a time period. We start the numerical integration at time (t) 0 and proceed using time steps that depend on the smoothness of the differential equations at time t . The integration ends when any of three conditions occurs.

1. The time t exceeds a maximum value (1.5×10^5).
2. The number of time steps made is more than a maximum value of 2500.
3. One of the chemical concentrations in the cell moves above the (arbitrary) value 1.0×10^{-4} . This denotes cell death due to saturation by a chemical.

When chemicals are created in the cell with concentration greater than that of one molecule, new reactions become possible and the system of differential equations is updated similarly to [4].

A GA with steady-state selection (e.g., [12]) and a population of 100 individuals evolves the cell models. Simulation of the cells is distributed over a network of machines with each machine simulating only one cell at a time. At any particular time, then, individuals in the population may either have finished simulation, be waiting for a machine to become available, or be in the process of being simulated. Breeding among individuals in the population occurs only when the population contains at least 75 cells that have finished simulation. Fitness proportionate selection with the roulette wheel algorithm finds breeding pairs. Mutation is applied with the probability of a bit mutating of 0.005.

Exponentially distributed random numbers were used to find distances between successive crossover points on the genome and regions between the crossover points were recombined. This algorithm has an effect similar to two-point crossover but more crossover points are possible. The exponential distribution was parameterized so as to produce an average of four crossover points for each pair of genomes.

The inversion genetic operator [6, 9] was also applied with probability 0.15 of an inversion event happening to a genome.

Fixed length genomes containing 800 bits were used.

The objective function is a combination of six metrics in the range [0, 1]. The metrics were derived with the intention of forming a canonical set.

- m_1 : The absolute change in the volume of the cell over the simulated period relative to a target value.
- m_2 : The length of time the cell was simulated.
- m_3 : The degree of correlation between the switching regions and the chemicals controlling cell growth.
- m_4 : The complexity of the metabolic reaction graph.
- m_5 : The number of chemicals affecting cell growth.
- m_6 : A parsimonious metric of the number of differential equations in the cell.

Combined metrics were determined for each permutation of single, double, triple, quadruple, quintuple, and sextuple of metrics. These were calculated by multiplying the metrics together (e.g., $m_1m_2m_3$, $m_1m_2m_4$, $m_1m_2m_5$). The objective function consisted of the unweighted sum of these permutations. Additionally, the fitness of cells that died during the simulation is halved. Our motivation behind assigning dead cells a fitness value other than zero was to help evolution at the start of runs. Many of the metrics conflict with each other so the maximum fitness of 63 cannot be realized.

Initially we tried simpler objective functions consisting of only m_1 or m_1 and m_2 . However, these were not sufficient to evolve cells that could live in the environment. We required an objective function that helped the cell find the sorts of behavior that were required (e.g., having genome switching regions that are similar to cell growth chemicals might be a useful attribute). On the other hand, we wanted to give the GA freedom to find solutions rather than programming them in ourselves. The set of fitness metrics listed above is a compromise between these two extremes.

5.1 Results

An experiment was run evolving cells in the simple environment. The first population of cells started with random genomes and simple initial

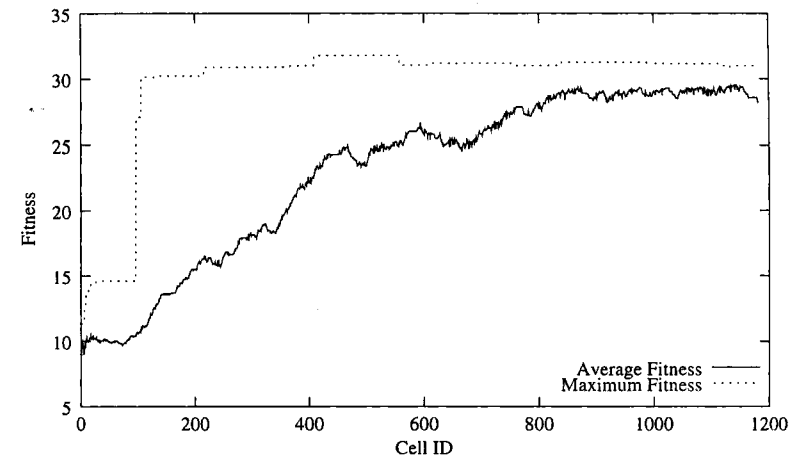


Figure 6. Graph showing the average and maximum fitness (y-axis) attained for experiment. Cells are numbered sequentially with an ID as they are generated (x-axis).

chemical ensembles. The initial chemical ensemble included a basic set of spider chemicals and simple chemicals that a cell could use as building blocks, but no enzymes. Figure 6 shows a graph of the fitness attained. When cells are created from parents they are assigned a unique identification number. Identification numbers increase for each new cell. The x-axis represents the identification number of the cell and thus shows each cell in the order they were created. Although this graph shows only one run of the experiment others were conducted and the results are representative.

Values near the maximum fitness were reached early (soon after offspring of the initial population are produced). A large diversity of genes was observed in the final population of cells. This reflects the fact that there does not appear to be one “right” solution for the experiment (with maximum fitness) but a variety of valid approaches, all having similar (high) fitness.

Genomes with different approaches to controlling a metabolism may be maintained as long as valid metabolic conditions for their use also exist. This suggests that future experiments may benefit from speciation.

It is difficult to visualize exactly how a simulated cell approaches the problem of living in its environment. This is because of the number of chemicals modeled: there are around 200 or so chemical species in each simulated cell. Many chemicals have multiple uses, they may act as a spider as well as a switching chemical, for example. Other chemicals do not contribute to the problem in an obvious way or perhaps at all. These chemical species are side effects of enzymes that perform another job or of enzymes that randomly exist on the genome. Many

chemicals exist at only very low concentration (because their constituent polymers also exist at low concentration). The GA does not produce tidy gene sequences where each enzyme does one thing and where there are no extraneous genes or chemicals. This confusion of chemicals and enzymes makes it difficult to analyze lots of cells, so here we examine two cells, in particular, in more detail.

The first cell we looked at scored the maximum fitness (31.0092) of the final population. We simulated the cell for a further time period to determine its stability. For this second time period the cell scored a fitness of 30.4144.

The cell contained five operons although only two of these had genes that encoded enzymes. The other three operons contained carrier genes. Of the two enzyme coding operons, one was constitutive and the other inducible. At the start of simulation the inducible operon was 21 percent switched on. The constitutive operon contained one gene coding for an enzyme with shape "51709" and the inducible one contained two genes: a carrier gene "02640" followed by the enzyme gene "137." All of the operons were quite short and only the one contained more than one gene.

There were 32 enzyme-catalyzed chemical reactions in the cell. However, there were 82 chemical species and two enzyme species in this cell, which implies that many of the chemical species in the cell were passed to it by the mother. Five species of chemical acted as builders although two of these were in low concentration and two in very low concentration. 15 species of chemicals could break down the membrane although 13 of these species were present in very low concentration and one other in low concentration. The cell had three chemicals it used as spiders but of these only one was present in high concentration.

The main strategy that this cell used was to build membrane breaking molecules to fight against the membrane building molecules that diffuse in. Although this appears sensible there are problems with the execution of this approach by the cell.

The enzyme "51709" that the cell used to make membrane breaking molecules is not sufficiently specific as it also catalyzes reactions that create builder molecules. Furthermore, these builder molecules are long and thus diffuse out of the cell quite slowly. The next problem with the strategy used by the cell also relates to specificity: the cell creates a chemical species that acts to both build and break down the membrane. This wastes resources in the cell. The final problem with this cell is the fact that most of the breaker molecules made by the cell are constructed out of other breaker molecules. The disadvantage here is that whilst the breaker molecules are being transformed they go out of solution and cannot act to break down the membrane.

The second cell that we examined was the one from the final population that scored the highest increase in fitness when run for the additional

time period. It scored 24.3847 the first time and 28.9469 the second time.

This cell contained nine operons, all containing one gene. We noticed that cells in earlier populations tended to have operons with more genes whereas cells in the later populations had less. Of the nine operons, three are constitutive, four are inducible, and two are repressible. There are four operons with enzyme genes: a constitutive operon encoding an enzyme with shape "07," an inducible one with enzyme "619," a repressible one for shape "53," and an inducible operon with enzyme "38." These two inducible operons were 15 and 55 percent switched on respectively and the repressible operon was 93 percent on at the start of simulation. We did not investigate how the operon activations changed throughout the lifetime of the cell. As well as the four enzymes coded on the genome, the cell inherited another enzyme "06" from the mother cell.

At the end of the first simulation there were 69 enzyme-catalyzed reactions in the cell and 107 chemical species. Of these chemical species only one was used to build the membrane and this diffuses in from the environment. There were 46 chemical species able to break down the membrane. Of these, only two had high concentration: one the cell started with (shape "30") and one species that the cell creates itself ("96730123"). The remainder were present only in very low concentration. There were five chemical species that acted as spiders, two present in high concentration and three in very low concentration.

This cell builds membrane breaking chemicals like the other cell. However, unlike the other cell, it has found enzymes that are specific. The enzymes do not create any membrane building chemicals and, interestingly, the cell has found (or exploited) a chemical (having shape "96730123") with two favorable properties: it may act as both a spider and a membrane breaking chemical. Additionally, this chemical is not built from an existing membrane breaking chemical.

It may be seen in Figure 6 that the maximum fitness of cells in the run fluctuates and even decreases at some times. This occurs because cells sometimes develop short term strategies to solve the problem posed by the environment. Such strategies work well for a short time and are consequently passed on to children. Some time later, however, the strategy starts to fail and other strategies become dominant. The reason that a strategy may stop working lies in the fact that genes alone do not govern the cell. It is the combination of genes interacting with a particular chemical ensemble that dictate how a cell acts. A set of genes that worked well in one parent may be passed to a child with a different chemical ensemble where the genes do not function. Alternately there may be a defect in the set of genes that takes a long time to manifest (e.g., the slow build up of a chemical that saturates and poisons the cell). For more details on these "myopic" cells, see [9] or [10].

6. Questions

The use of the gene expression model in the experiment outlined above encourages us to formulate four questions. We believe that our gene expression and regulation model is sufficiently detailed to investigate these questions and provide useful answers.

- Do we need a dynamic environment to evolve the regulation mechanisms?
- Does the artificial evolution take advantage of time lags inherent in the expression algorithm?
- Is our genomic language sensitive to attack from genetic operators?
- How does the GA use the noncoding regions between operons?

We discuss each of these questions in turn.

6.1 Do we need a dynamic environment to evolve the regulation mechanisms?

Although repressible and inducible operons evolved in our experiment we do not believe they were fully utilized by cells. When looking at cells in the final population (more than the two analyzed above) we did not see a strong correlation between information in the promoter switching regions and the chemicals important to the problem. Instead, we observed short random-looking strings in the switching regions. These strings are not specific to particular chemicals in the cell because the strings are, in general, quite short (comprised of one to three bases). Rather, these operons are switched by a large set of chemicals including basic “building block” chemicals that exist in high concentration in the cell and environment. The concentrations of large sets of chemicals will not change quickly over time, particularly if major constituents of this set are the high concentration “building block” chemicals which are deliberately kept constant in the cell. This means that the activation of these operons will not change much. They therefore act similar to constitutive operons that are not switched fully on. However, we need to analyze the switching regions more completely to be confident that this observation is general.

One explanation for why gene regulation has not been used is that it is not necessary for cells in this environment. This is perhaps because the environment does not present changing stimuli to the cell. An experiment to test this hypothesis requires a more dynamic environment.

6.2 Does the artificial evolution take advantage of time lags inherent in the expression algorithm?

Gene switching is not instantaneous: there is a time lag between a switch chemical binding to the promoter and a change in the amount of protein produced. This occurs because regulation affects only spiders *joining*

the operon from the pool. The size of this time lag for each gene is proportional to its distance from the promoter region and also to the number of bases preceding the gene.

The GA may conceivably take advantage of this time lag by (i) placing genes that need fast switching earlier in the operon, (ii) using shorter genes when they need to be switched quickly, or (iii) using two operons with similar promoter regions for faster gene expression and regulation (parallel processing).

As explained above, we did not observe much use of the gene regulation mechanism in our experiment. To see how the GA uses time lags we would need to see more use of gene regulation.

However, in the cells we did examine we noticed that cells in later populations seemed to favor short operons containing only one or two genes. Short operons provide a benefit to cells because the enzymes in them take less time to express (the gene expression reaction pathway is shorter) and are, therefore, available earlier. Again, we need to systematically examine more cells to test whether this observation is general.

There is an interesting tension here that the GA must resolve. Short operons and genes express faster and are useful earlier, but the short enzymes that result tend to be less specific and therefore may have more (possibly unwanted) side effects. Our cells need to find a balance.

6.3 Is our genomic language sensitive to attack from genetic operators?

The four special control codes used in our parallel genomic language cause semantic linkages within operons. The meaning of a base in an operon is not an intrinsic property, but is dependent on the preceding “special” tokens. This linkage implies that genetic operators may have a significant effect on a genome. For example, one mutation could obliterate an operon by changing a `<start operon>` code to another token.

Experiments are required to test the sensitivity of our genomic language. Factors such as the degree of redundancy in the genomic language and the density of operons on the genome would appear to be significant but further investigation with a variety of different languages is required.

Preliminary experiments varying the rate of mutation applied to the genome are reported in [9]. These experiments suggest that our encoding scheme is not unduly sensitive to high rates of mutation but this result needs to be compared with other genomic languages with differing amounts of redundancy.

6.4 How does the genetic algorithm use the noncoding regions between operons?

We are interested in investigating how the GA uses the regions between operons. We suspect that these noncoding regions may be used to store potential new genes and parts of old operons that are no longer used.

With such a scenario new genes would arise when a fortunate mutation, inversion, or recombination occurs. The ruined operons might store a garbled history of the genome over past generations. We would expect that the more redundancy that exists in the genomic language, the closer the noncoding regions are to being fully fledged operons. This would be because only small changes are required to transform a noncoding region to an operon.

Identification of the actual effect of noncoding regions and redundancy on evolution will be a focus of our future work.

7. Application to evolutionary computation

We wish to understand how our model of gene expression and regulation applies to EC and what benefits it may confer. At this point we do not have a definitive answer, but some issues are clear.

Our model has a strong biological focus: the genes encode proteins and a system of chemical reactions is required to effect gene expression. The two kinds of proteins (enzymes and carriers) produced from the genome may be viewed as two distinct *types* of parameters that are found and optimized by the GA. Our gene expression algorithm allows any number of actual instances of these types of parameters (i.e., species of proteins) to be created.

The way that these parameters interact with the cell is interesting. Rather than completely specify the cell, they modify an already existing system. A child cell is comprised of a set of chemicals inherited from the mother. As we saw in the second cell case study above, a child cell may also inherit enzymes from the mother. This ensemble is the existing chemical system that the genes will modify. Each enzyme specified on the genome modifies how this chemical system functions. No enzyme can “break” the chemical system in the sense that fitness may not be determined. The worst effect an enzyme may have on the system is to cause the cell to die. However, even in this case, fitness may be calculated. As well, regulation of operons means that the operons modify the chemical system dynamically throughout the simulation.

This is different to the way many genomes work in EC where the genes encoded on a genome completely specify a potential solution to the problem being solved. If a gene is missing or if there is more than one copy of a gene, a potential solution cannot be formed and fitness may not be calculated.

We suggest that application of a gene expression model like ours to EC requires rethinking the way that the genes apply to the potential solutions that are evolved. Genes should act as “delta” values to the system. They should change a system rather than define it. One way to do this would be to assume default values for parameters of the potential solution. Each gene would change a parameter. Different types of genes

would change different parameters. If no genes of a particular type existed on the genome then the parameter would not be changed and its default value would prevail. If there were more than one gene of a particular type then both delta values would be applied to the parameter in some well defined way.

The first step we wish to take towards applying the expression algorithm to other areas is that of returning to the discrete spider model (section 4.1) and changing the complex chemical metabolism to a simpler discrete model using finite state automata.

8. Conclusions

A gene expression and regulation algorithm was described that operates within the context of a model of a single-celled organism. There is a close relationship between the genome and metabolism. Gene expression and regulation in the model is the result of interactions between the metabolism and genome.

An experiment evolved cells to exist in a simple environment. Observations from the experiment lead us to formulate four questions that may be answered using the model.

1. Do we need a dynamic environment to evolve the regulation mechanisms?
2. Does the artificial evolution take advantage of time lags inherent in the expression algorithm?
3. Is our genomic language sensitive to attack from genetic operators?
4. How does the genetic algorithm (GA) use the noncoding regions between operons?

Some issues with the application of our gene expression and regulation model to evolutionary computation (EC) were discussed.

Appendix

A. Derivation of gene regulation formulae

A.1 Derivation of $\hat{\rho}_n$, the probability that one of n competing switch chemicals is bound to the promoter region of an operon

Initially $\hat{\rho}_n$, the probability that one of n competing chemicals is bound to a promoter region at a given time, will be derived from first principles in a recursive style. Firstly, observe that

$$\hat{\rho}_{n+1} = \frac{\sum_{i=1}^n T_i + T_{n+1}}{T} \quad (\text{A.1})$$

where T_i is the time that the i th switch chemical is bound to the promoter region and T is the total amount of time of interest. That is, the probability of one of the chemicals being bound is equal to the time they are attached, divided by the total amount of time. This can be rewritten as

$$\hat{\rho}_{n+1}T = \sum_{i=1}^n T_i + T_{n+1}. \quad (\text{A.2})$$

The probability $\hat{\rho}_n$ may also be seen as a throughput. In that case, we can say that the amount of time chemicals are bound is equivalent to the throughput multiplied by the total amount of time:

$$\sum_{i=1}^n T_i = \hat{\rho}_n(T - T_{n+1}) \quad (\text{A.3})$$

$$T = \frac{\sum_{i=1}^n T_i}{\hat{\rho}_n} + T_{n+1}. \quad (\text{A.4})$$

Similarly, looking at only the probability of the $n+1$ switch chemical binding to the promoter, we can state

$$T_{n+1} = \rho_{n+1} \left(T - \sum_{i=1}^n T_i \right) \quad (\text{A.5})$$

$$T = \sum_{i=1}^n T_i + \frac{T_{n+1}}{\rho_{n+1}}. \quad (\text{A.6})$$

Solving the simultaneous equations (A.4) and (A.6) by subtracting equation (A.6) from (A.4), we get

$$0 = \frac{\sum_{i=1}^n T_i}{\hat{\rho}_n} - \sum_{i=1}^n T_i + T_{n+1} - \frac{T_{n+1}}{\rho_{n+1}}, \quad (\text{A.7})$$

which simplifies to give

$$T_{n+1} = \frac{(\sum_{i=1}^n T_i)\rho_{n+1}(1 - \hat{\rho}_n)}{\hat{\rho}_n(1 - \rho_{n+1})}. \quad (\text{A.8})$$

Substituting equation (A.8) into (A.4) yields

$$T = \frac{\sum_{i=1}^n T_i}{\hat{\rho}_n} + \frac{(\sum_{i=1}^n T_i)\rho_{n+1}(1 - \hat{\rho}_n)}{\hat{\rho}_n(1 - \rho_{n+1})}. \quad (\text{A.9})$$

Substitute equation (A.8) into (A.2):

$$\hat{\rho}_{n+1}T = \left(\sum_{i=1}^n T_i \right) \left(1 + \frac{\rho_{n+1}(1 - \hat{\rho}_n)}{\hat{\rho}_n(1 - \rho_{n+1})} \right). \quad (\text{A.10})$$

Finally, substitute equation (A.9) into (A.10) and simplify to get

$$\hat{\rho}_n = \frac{\frac{\hat{\rho}_{n-1}}{1 - \hat{\rho}_{n-1}} + \frac{\rho_n}{1 - \rho_n}}{\frac{1}{1 - \hat{\rho}_{n-1}} + \frac{\rho_n}{1 - \rho_n}}. \quad (\text{A.11})$$

This equation is cumbersome because it is recursively defined and of a complex nature. From looking at the forms of $\hat{\rho}_2$, $\hat{\rho}_3$, and $\hat{\rho}_4$ a pattern appears which suggests that there is a simpler iterative form of equation (A.11).

Theorem 1. An iterative version of equation (A.11) exists that may be defined as

$$\hat{\rho}_n = \frac{\rho_1 + (1 - \rho_1)I(n)}{1 + (1 - \rho_1)I(n)},$$

$$I(n) = \sum_{i=2}^n \frac{\rho_i}{1 - \rho_i}. \quad (\text{A.12})$$

Proof. The proof that equation (A.12) is equivalent to equation (A.11) proceeds using proof by induction.

I. Show that equation (A.12) is valid for $n = 1$:

$$\begin{aligned} \hat{\rho}_1 &= \frac{\rho_1 + (1 - \rho_1)I(1)}{1 + (1 - \rho_1)I(1)} = \frac{\rho_1 + (1 - \rho_1)0}{1 + (1 - \rho_1)0} \\ &= \rho_1. \end{aligned} \quad (\text{A.13})$$

II. Assume that $\hat{\rho}_i$ is valid:

$$\hat{\rho}_i = \frac{\rho_1 + (1 - \rho_1)I(i)}{1 + (1 - \rho_1)I(i)}. \quad (\text{A.14})$$

III. Show that $\hat{\rho}_{i+1}$ follows from $\hat{\rho}_i$ above, using the recursive form equation (A.11):

$$\begin{aligned} \hat{\rho}_{i+1} &= \frac{\frac{\hat{\rho}_i}{1 - \hat{\rho}_i} + \frac{\rho_{i+1}}{1 - \rho_{i+1}}}{\frac{1}{1 - \hat{\rho}_i} + \frac{\rho_{i+1}}{1 - \rho_{i+1}}} \\ &= \frac{\frac{\left(\frac{\rho_1 + (1 - \rho_1)I(i)}{1 + (1 - \rho_1)I(i)} \right)}{1 - \frac{\rho_1 + (1 - \rho_1)I(i)}{1 + (1 - \rho_1)I(i)}} + \frac{\rho_{i+1}}{1 - \rho_{i+1}}}{\frac{1}{1 - \frac{\rho_1 + (1 - \rho_1)I(i)}{1 + (1 - \rho_1)I(i)}} + \frac{\rho_{i+1}}{1 - \rho_{i+1}}}. \end{aligned} \quad (\text{A.15})$$

This simplifies to give

$$\begin{aligned} \hat{\rho}_{i+1} &= \frac{\rho_1 + (1 - \rho_1) \left(I(i) + \frac{\rho_{i+1}}{1 - \rho_{i+1}} \right)}{1 + (1 - \rho_1) \left(I(i) + \frac{\rho_{i+1}}{1 - \rho_{i+1}} \right)} \\ &= \frac{\rho_1 + (1 - \rho_1)I(i+1)}{1 + (1 - \rho_1)I(i+1)}. \end{aligned} \quad (\text{A.16})$$

So, III follows and

$$\hat{\rho}_n = \frac{\rho_1 + (1 - \rho_1)I(n)}{1 + (1 - \rho_1)I(n)} \quad (\text{A.17})$$

is valid for all n . ■

■ A.2 Derivation of $d\hat{\rho}_n/dt$

The derivative of $\hat{\rho}_n$ is required for the system of differential equations driving the cell simulation. Each ρ_i in $\hat{\rho}_n$ is a function of the concentration of the switch chemical it describes:

$$\rho_i = 1 - e^{-K_i s_i}. \quad (\text{A.18})$$

We may find the derivative with respect to time using the chain rule:

$$\frac{d\rho_i}{dt} = \frac{d\rho_i}{ds_i} \frac{ds_i}{dt} = K_i e^{-K_i s_i} \frac{ds_i}{dt}. \quad (\text{A.19})$$

The value of ds_i/dt is already defined in the cell model. It is the differential equation used to determine the rate of change of the chemical with respect to time. Using the chain rule again, we find

$$\frac{d\hat{\rho}_n}{dt} = \sum_{i=1}^n \frac{\partial \hat{\rho}_n}{\partial \rho_i} \frac{d\rho_i}{dt}. \quad (\text{A.20})$$

This simplifies to give

$$\frac{d\hat{\rho}_n}{dt} = \left(\frac{1 - \rho_1}{1 + (1 - \rho_1)I(n)} \right)^2 \sum_{i=1}^n \left(\frac{1}{(1 - \rho_i)^2} \frac{d\rho_i}{dt} \right). \quad (\text{A.21})$$

References

[1] Alberts, Bray, *et al.*, *Molecular Biology of the Cell* (Garland Publishing, New York, third edition, 1994).

[2] R. J. Bagley and J. D. Farmer, "Spontaneous Emergence of a Metabolism," in Langton *et al.* [11].

[3] R. J. Bagley, J. D. Farmer, and W. Fontana, "Evolution of a Metabolism," in Langton *et al.* [11].

[4] J. D. Farmer, S. A. Kauffman, and N. H. Packard, "Autocatalytic Replication of Polymers," *Physica D*, 22 (1986) 50–67.

[5] E. J. Gardner, M. J. Simmons, and D. P. Snustad, *Principles of Genetics* (John Wiley & Sons, eighth edition, 1991).

[6] J. H. Holland, *Adaptation in Natural and Artificial Systems* (MIT Press, Cambridge, MA, first MIT press edition, 1992).

[7] F. Jacob and J. Monod, "Genetic Regulatory Mechanisms in the Synthesis of Proteins," *Journal of Molecular Biology*, 3 (1961) 318–356.

[8] N. Jacobi, "Harnessing Morphogenesis," *Cognitive Science Research Paper 423* (School of Cognitive and Computing Sciences, University of Sussex, 1995).

[9] P. J. Kennedy, *Simulation of the Evolution of Single Celled Organisms with Genome, Metabolism, and Time-Varying Phenotype*, PhD thesis (University of Technology, Sydney, 1998).

[10] P. J. Kennedy and T. R. Osborn, "Evolution of Adaptive Behaviour in a Simulated Single-celled Organism," in *SAB2000 Proceedings Supplement Book*, edited by J. A. Meyer, A. Berthoz, D. Floreano, H. Roitblat, and S. Wilson (International Society for Adaptive Behaviour, Honolulu, 2000).

[11] C. G. Langton, J. D. Farmer, and S. Rasmussen, editors. *Artificial Life II*, volume x of *SFI Studies in the Sciences of Complexity* (Addison-Wesley, 1991).

[12] M. Mitchell, *An Introduction to Genetic Algorithms* (MIT Press, Cambridge, MA, 1996).

[13] M. O'Neill and C. Ryan, "Incorporating Gene Expression Models into Evolutionary Algorithms," in *Proceedings of The 2000 Genetic and Evolutionary Computation Conference Workshop Program*, edited by A. S. Wu (Las Vegas, Nevada, 8 July 2000).

[14] R. S. Rosenberg, *Simulation of Genetic Populations with Biochemical Properties*, PhD thesis (University of Michigan, 1967).

- [38] R. S. Rosenberg, "Simulation of Genetic Populations with Biochemical Properties," *Dissertation Abstracts International*, 28(7):2732B, 1967 (University Microfilms Number 67-17,836).
- [39] P. Schuster, "The Role of Neutral Mutations in the Evolution of RNA Molecules," in *Theoretical and Computational Methods in Genome Research*, edited by S. Suhai (Plenum Press, New York, 1997).
- [40] R. E. Smith, "An Investigation of Diploid Genetic Algorithms for Adaptive Search of Nonstationary Functions," *TCGA Report Number 88001* (University of Alabama, The Clearinghouse for Genetic Algorithms, Tuscaloosa, 1988).
- [41] D. Thierens, "Estimating the Significant Non-linearities in the Genome Problem-coding," in *Proceedings of the Genetic and Evolutionary Computation Conference* (Morgan Kaufmann Publishers, 1999).
- [42] D. Thierens, "Scalability Problems of Simple Genetic Algorithms," *Evolutionary Computation*, 7(4) (1999) 331–352.
- [43] M. Vose and A. Wright, "The Simple Genetic Algorithm and the Walsh Transform: Part I, Theory," *Journal of Evolutionary Computation*, 6(3) (1998) 253–274.
- [44] M. Vose and A. Wright, "The Simple Genetic Algorithm and the Walsh Transform: Part II, The Inverse," *Journal of Evolutionary Computation*, 6(3) (1998) 253–274.
- [45] J. L. Walsh, "A Closed Set of Orthogonal Functions," *Ann. Journ. Math.*, 55, 1923.
- [46] A. Wu and R. Lindsay, "Empirical Studies of the Genetic Algorithm with Non-coding Segments," *Journal of Evolutionary Computation*, 3(2) (1995) 121–147.
- [47] A. Wu and R. Lindsay, "A Survey of Intron Research in Genetics," in *Parallel Problem Solving from Nature—PPSN IV*, edited by H. Voigt, W. Ebeling, I. Rechenberg, and H. Schwefel (Springer-Verlag, Berlin, 1996).