Preliminary Studies on the In Silico Evolution of Biochemical Networks

Anastasia Deckard[a, b] and Herbert M Sauro*[a, c]

Due to the variety and importance of roles performed by signalling networks, understanding their function and evolution is of great interest. Signalling networks allow organisms to process and react to changes in their internal and external environment. Current estimates suggest that two to three percent of all genomes code for proteins involved in signalling networks. The study of signalling networks is hindered by the complexities of the networks and difficulties in ascribing function to form. For example, a very complex dense network might comprise eighty or more densely connected proteins. In the majority of cases there is very little understanding of how these networks process signals. Unlike in electronics, where there is a broad practical and theoretical understanding of how to construct devices that can process almost any kind of signal, in biological signalling networks there is no equivalent theory. Part of the problem stems from the fact that in most cases it is unknown what particular signal processing circuits would look like in a biological form. This paper describes the evolutionary methods used to generate networks with particular signal- and computational-processing capabilities. The techniques involved are described, and the approach is illustrated by evolving computational circuits such as multiplication, radicals and logarithmic functions. The experiments also illustrate the evolution of modularity within biochemical reaction networks.

Introduction

Computation is usually associated with man-made devices, such as the ubiquitous digital computer or the lesser-known analogue computer. However there is also a vast array of computational devices to be found in living organisms in the form of protein and genetic networks. When changes are sensed in the state of their external and internal environments, biological systems utilize signalling pathways made up of interacting proteins to process the information, allowing them to coordinate and integrate a response appropriate to the actual milieu.

One of the most common examples of motifs found in biological signalling networks is the reaction cycle formed by a phosphorylating protein kinase and dephosphorylation phosphatase (Figure 1). Such cycles are commonly chained together to form "cascades", which are cross-linked with other cascades to form a dense network of interacting chemical cycles. In eukaryotes, especially in higher organisms, these signalling networks can be quite large, comprising up to eighty or more kinases and phosphatases.[1] The signalling networks receive a variety of inputs, generating one or more outputs that range from changing gene expression to modifying enzyme activities in metabolic pathways. Understanding what these networks "compute" is one of the most interesting questions in the study of signalling networks. Unfortunately, the exact decision making that occurs and how it is achieved is not well understood. When disrupted, many of these networks can result in malignancy in higher organisms, and thus their study has both academic and heath-related importance.

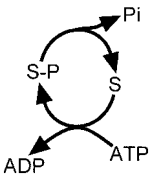

Figure 1. *The phosphorylation cycle: A commonly found network motif in biological signal-processing pathways.*

There are less than a handful of signalling networks for which we are confident that we understand the nature of the computation, two examples are *E. coli* chemotaxis and the eukaryotic mitogen-activated protein kinase (MAPK) pathway.

The *E. coli* chemotaxis network is believed to include at least three functional units: an actuator and error generator, a time-delay unit and, most remarkable of all, a numerical integrator.[2] Essentially the circuit acts as a simple but effective analogue computer. The chemotaxis network enables *E. coli* to sample the local nutrient conditions at intervals, allowing it to coordinate movement up a nutrient gradient.[3] *E. coli* must employ this sampling strategy because it is too small to detect concentration gradients directly by differential sampling across its length.

The MAPK pathway seems to perform different roles depending on its context. In certain situations, MAPK is believed to act as a switch, enabling the system to translate a continu-

[a] *A. Deckard, Dr. H. M. Sauro*
Keck Graduate Institute
535 Watson Drive, Claremont, CA 91711 (USA)
Fax: (+1) 909-607-8086
E-mail: hsauro@kgi.edu

[b] *A. Deckard*
California State University
Fullerton, CA 92834 (USA)

[c] *Dr. H. M. Sauro*
Control and Dynamical Systems 107–81
California Institute of Technology
Pasadena, CA 91125 (USA)

 DOI: 10.1002/cbic.200400178

ous graded signal into an on/off response.[4] In different situations, the MAPK may take on the role of a classic feedback amplifier or an oscillator.[5,6]

Understanding the functioning of specific networks offers insight into the roles and mechanisms of signalling networks in general. Expanding on such understanding is the central goal in the field of "Systems Biology", perhaps more aptly called "Molecular Physiology". This goal has proven elusive, as often only the structure of a network is known; in rarer cases, the computational model of the network is known. Either way, the extreme difficulty of ascribing function to networks limits understanding of the network's signal processing capacity. In the majority of cases it is unknown what control systems are employed and whether networks carry out simple or elaborate, analogue or digital signal processing.

The similarity between biological signalling networks and man-made devices is quite compelling, particularly when compared to electronic networks. For example, the properties of a phosphorylation cycle are remarkably similar to the response exhibited by an electronic transistor.[5] Other motifs that have electronic equivalents have been found in biological systems, including bistable switches, oscillators and delay units.[7] However, there is no equivalent "circuit theory" of signalling networks. In Systems Biology there is no engineering handbook for understanding signalling networks, whereas in electronics there is an extensive body of theory and a large amount of practical knowledge (however, see ref. [8]). Unfortunately, it is unknown what many of the common signal processing elements found in electronic systems would look like in a biological network. Therefore it is difficult to pinpoint the exact type of signal processing that occurs in large biological signalling networks. In addition, biological networks may contain novel signal processing capabilities that lack counterparts in man-made devices; this makes the discovery and classification of biological networks even more problematic.

Simple Boolean units constructed from chemical networks, such as AND, OR and NOT gates, have been studied theoretically for a number of years.[9–13] It is known that chemical kinetics can be used to create universal Turing machines.[14] In the study of analogue-signal processing, Wolf and Arkin,[7] and Tyson et al.,[15] in particular have discussed a variety of analogue motifs in biological networks, for example, the existence of distinct modules such as feedback loops, oscillators, switches, noise filters, amplifiers and memory elements (both analogue and digital). It is suggested that these modules form interacting building blocks capable of combining into hierarchies.[16] While interest in this field is clearly growing, it is still at a very early age of development.

Computational Systems Utilizing Biological Methods

Just as biological systems use computation, humans have been building computational devices that use biologically inspired processes. We are proposing to use evolutionary techniques[17] to build biochemical networks with particular signal-processing capabilities. There are three general types of evolutionary algorithm: genetic algorithms (GA),[18] evolutionary strategies (ES),[19] and evolutionary programming (EP).[20] All these techniques are generally based on the evolutionary processes of selection, crossover, and mutation. Kacser and Beeby[21] may have been one of the first to employ an evolutionary strategy to evolve networks. They used an evolutionary approach to try to reconstruct the general course of early enzyme evolution. Bray and Lay[22] applied a GA method to the problem of optimizing reaction rates in a model for signal transduction. Gilman and Ross[23] studied the selection of a regulatory structure that directs flux in a simple metabolic model using a GA. Genetic algorithms have also been used to determine reaction mechanisms and reaction rates for a network involving the Ce-catalyzed minimal bromate system.[24,25] In more recent work to identify bistable switches and oscillators in gene networks, François and Hakim utilized evolutionary procedures.[26] They claim to evolve networks from basic interactions between genes and proteins. The fitness of the network was defined by its dynamics, either bistability or oscillatory behaviour. Shin and Iba used the S formalism to evolve gene networks,[27] and evolutionary algorithms have also been used to design a glycolysis network that interacts with an external ATP-consuming reaction.[28] It was found in this work that evolutionary algorithms are very useful in determining the optimal stoichiometry of metabolic pathways. Efforts to define the robustness of networks against changes in their stoichiometry have also been made, and it was found that populations evolve towards clusters of networks that perform a common function.[29]

Probably the most prolific and successful application of evolutionary algorithms in designing devices has been performed by Koza,[30] who has been highly successful in using genetic programming to evolve electronic circuits, such as filters and analogue computational units, to name but two. Koza has also done some preliminary investigations into using genetic programming to study metabolic pathways.[31] Following Koza, we propose to use an evolutionary approach to evolve signal-processing biochemical reaction networks able to perform mathematical calculations.

Design by Evolution

Why use an evolutionary approach to solve a problem? An evolutionary approach is very efficient in situations in which many possible solutions exist and it is not necessary to find one perfect solution. This will be illustrated by a brief discussion of search spaces and how evolutionary techniques isolate workable solutions.

The search space is the sum of all possible unique networks that must be evaluated. Allowing for five species nodes where 0, 1, 2, …, 10 reactions can exist of one of four types (uni–uni, uni–bi, bi–uni, or bi–bi), the number of unique topologies would be approximately 10^{22}. Each of the 10^{22} topologies must be considered as a possible solution. Testing each topology in order would be like sitting next to the haystack and plucking the next thing from the pile hoping it might be a needle. This is very efficient if the first network is the best, but very ineffi-

cient if the best network happens to be the 10 000 000 000 000 000 000 000th. When the rate constants are considered part of the search space, each topology contains a near infinite search space.

Using an evolutionary approach restructures the search space and changes the way it is explored. In the search space, any given unique network is surrounded by similar unique networks. You are randomly dropped onto a network, and any single mutation moves you to an adjacent network. If you move blindly from one network to a worse one, you abandon your search in that area. If you move to a better network, you keep moving blindly in that general direction. This allows for movement towards a better solution without having to examine every solution.

Finding the single perfect solution by using evolutionary techniques is never assured. The solution depends on the starting conditions and the chance mutations that occur. The program may end up stuck in a local optimum, unable to make a large enough jump to escape to a new region that possibly contains a better solution. The solution found might be a workable solution, but it is probably one of many and not necessarily the best.

The program called Lakhesis was developed in order to find networks able to perform mathematical calculations by using evolutionary techniques. Lakhesis was created in two steps: the creation of a program able to accurately model and simulate a network of chemical reactions, and then the extension of the program to evolve the networks.

Modelling and Simulating Networks of Chemical Reactions

With Lakhesis, the reaction networks are modelled such that the molecular species are nodes with an attribute for concentration, and the reactions between the species are connections with attributes for rate constants and reaction types. All attributes in Lakhesis are numeric values, following the evolutionary-strategy approach. The networks have external input and output nodes that serve as sources and sinks in the system, enabling the networks to reach a stable steady state. Networks that go to thermodynamic equilibrium are not of interest. The external input node also serves as the signal input to the system and can be changed during a simulation. The external output nodes are always fixed. The internal floating nodes of the network serve as the readout nodes, any one of which might be capable of computing the desired objective function. An example of a typical random network is shown in Figure 2.

To compute the steady state of a network, the input and output rates for each node are computed, and the change in concentration is calculated by subtracting the total output rate from the total input rate for that node. This process is carried out for each floating node in the network to yield a set of ordinary differential equations. The differential equations are solved by using a 4th-order Runge–Kutta solver with a step size of 0.05. The solutions to the differential equations describe the changes in node concentration of the network in time.

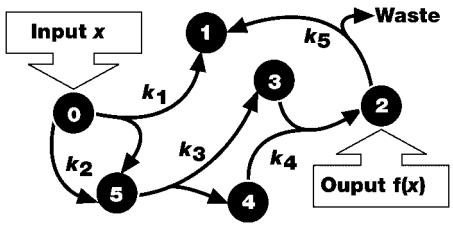

Figure 2. *A typical network constructed during the first generation of the evolution software. The concentration of node 0 is considered the input to the network (x). The concentration of node 2 is the output of the function f(x), as measured after the system reaches steady state.*

Viable networks are determined by their node's ability to sustain a steady state, in which concentrations settle to a constant value. If the change in concentration for a particular node is less than some pre-prescribed threshold value, the node is marked as temporarily being in a steady state. If it remains in a steady state over successive iterations, the node is marked as having a continuing steady state. If the network failed to reach steady state on any of its nodes at the end of an input-value run, it is immediately removed from the population.

To test the accuracy of modelling and simulating, network-simulation results from Lakhesis were compared to those from Jarnac 2.0.[32] The comparison verified that Lakhesis was indeed accurately simulating the reaction networks.

Evolving Networks of Reactions

For evolution to occur, it is necessary that genotypic variation exists among individuals in the population and that a selection process acts upon this variation to eliminate the individuals with poor phenotypes. Genotypical variation is maintained through stochastic processes, such as mutation or crossover during reproduction. In Lakhesis, selection is a deterministic process in which only the best networks are selected for reproduction. The cycle of creating variable offspring and then selecting the best of the offspring is repeated for many generations, as shown in Figure 3. This repetition of mutation and selection gently pushes the survivors towards an optimal level of adaptation.

A) Creating a genetically diverse initial population

As the raw material for evolution is genetic variability, it is necessary to have as many genetically diverse networks as possible in each generation. For the initial generation, the population size is ten times larger than subsequent generations in order to create a relatively large amount of initial diversity. The amount of genetic variability is determined by the number of unique topologies in the population, where unique topologies have different combinations of numbers of connections per node, different types of connections and the rate constants associated with each connection. To create this diversity, the networks in the initial population are created by assigning them random connections and rate constants.

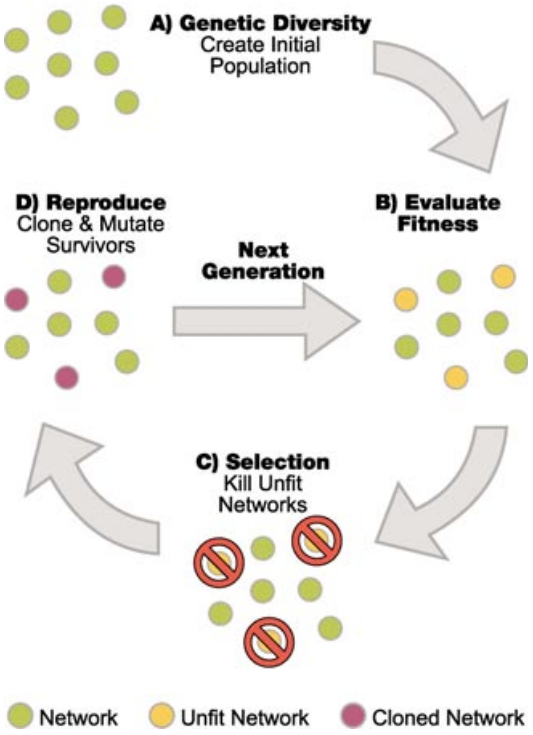

Figure 3. *Diagram of the in silico evolution A) Genetic Diversity: An initial population is created by generating random networks. B) Evaluate Fitness: The output of the network is the network's phenotype, which is compared to the expected output in order to calculate a fitness score for the network. C) Selection: Networks with poor fitness scores are removed from the population. D) Reproduction: The surviving networks are cloned or bred, and the offspring are randomly mutated to maintain genetic variability. Evaluation of fitness, selection, and reproduction are repeated for each generation.*

Every network consists of a fixed number of nodes and a maximum number of connections between the nodes, as set in the program's configuration. The number of connections for a given node is randomly generated at the time the node is created, and can be from zero to the maximum number of connections specified. Then each connection must be assigned a type (uni–uni, uni–bi, bi–uni, bi–bi) and a rate constant as shown in Figure 4. This procedure for generating connections is repeated for every connection in the network. The bimolecular reaction types are essential as they contribute nonlinear behaviour to the networks.

When the process is completed, the network has a topology that describes a set of differential equations representing the genotype of a network. If no other nodes connect to a given node, then it will not be involved in the network. If a node has connections coming into it but no connections going out, the node will serve as a waste node for the network and will not be assigned a differential equation.

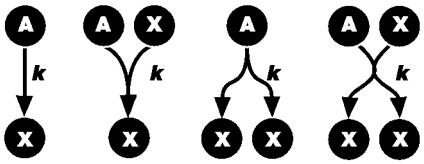

Figure 4. *To create a random connection, first a connection type is randomly selected from one of the above types for node A. The other nodes, marked as X, necessary for the connection type are randomly selected from the existing network. Finally, a random number is generated for the rate constant* k.

B) Evaluating the fitness of a network

To evaluate the fitness of a network, the network's genotype in the form of a set of ordinary differential equations is translated to a phenotype by computing the steady-state solution with a 4th-order Runge–Kutta method. The fitness of a network is based on the amount it deviates from the expected value of the objective function f(x). An input value x is set for the input node, and the time course behaviour of the network is evaluated. If the system cannot reach a steady state, it is immediately removed from the population. If the system reaches a steady state, the deviation of each node is calculated by subtracting the output of the node from the exact value of f(x). The input value x is set to a new value, and the test is repeated. For example, the training inputs for evolving a square-root network might be 2^1, 2^2, 2^3, …, 2^n, where n ranges from 6 to 11.

Each node receives a total-deviation score, which is the sum of the deviation scores that particular node received for each input value. The node that has maintained a steady state and has the least amount of deviation is marked as the best node in the network. The best node is considered the output function (f(x)) for the system, and the best node's total deviation is the fitness for that network as shown in Figure 5. The lower the deviation score, the higher the fitness of the network.

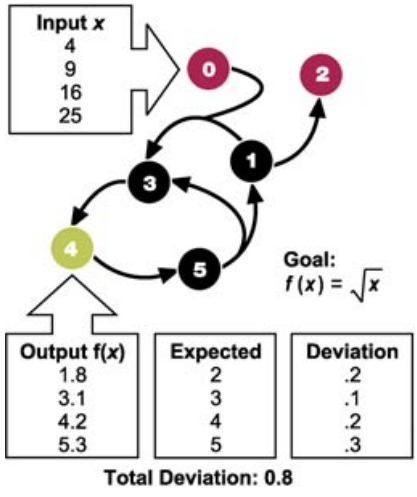

Figure 5. *Calculating deviation for a network. The network is simulated for several different inputs, and the readouts measured. All the deviation scores are summed for each of the nodes. Each node is tested, and the node with the lowest total deviation score is the output for the network.*

C) In silico selection: Survival of the least deviant

Before reproduction occurs, selection must remove the least-fit networks from the population. Lakhesis uses a selection technique called elitism, in which the networks with the best fitness scores are always guaranteed survival.[18] The number of

best fitness scores to select for the next generation is specified in the configuration of the program. For example, the ten lowest deviation scores are found in the population, and any networks that have deviation scores that are worse are removed from the population. Where a neutral mutation occurs and causes several networks to have the same deviation score and the deviation score is not selected for elimination, all networks with that phenotype remain in the population. Because of this, more networks usually remain in the population after selection than the number of top scores.

Even though Lakhesis was designed to simulate evolution, the "species" that exist in the program are not subject to the same physical and environmental constraints as their biological counterparts. For example, in Lakhesis no genetic drift occurs because individuals are never randomly removed from the population. Once a good solution appears it is never lost until it is replaced by a better solution. To make this more comfortable for the biologically inclined, the solution that is copied into the next generation can be thought of as an exact clone of the fit parent. Therefore any fluctuations in the frequencies of unique genotypes between generations are due wholly to natural selection and not to any stochastic processes.[33]

As the individuals in Lakhesis are haploid agamospecies, their evolution follows a slightly different path from that with sexual diploid species. Genetic variability is only kept through continuous mutations, and speciation can only occur at the death of intermediate clones. For these reasons, the rate of mutation and the strictness of selection are intentionally high. An exact clone of a highly fit individual is guaranteed in the next generation; this circumvents the concern that too high an error rate will lead to loss of adaptation.[33]

D) Asexual reproduction and mutations

After the unfit networks have been removed from the population, the survivors have offspring by either asexual or sexual reproduction. Currently Lakhesis uses asexual reproduction as it proved to be more efficient than sexual reproduction for smaller networks. Each of the remaining networks has enough offspring to bring the population back to capacity, so few surviving networks will have many offspring. Surviving networks remain in the population to compete in the next generation.

Lakhesis uses a base mutation rate that sets a range of mutation rates for each offspring. For example, the first offspring of a network will have a $^1/_3$ base mutation rate, and the next $^1/_4$, and the next $^1/_5$. This allows for a continuum of similarity between the offspring and the parent, in which the highly similar offspring may be a refinement of the parent's solution and the highly mutated offspring might find novel and possibly better solutions. In addition to the base mutation rates, mutations of different parts of the connections occur at differing probabilities. The probability of the mutation type is inversely proportional to the amount of disruption it will cause, as shown in Figure 6. Rate-constant mutations are the smallest possible change in the genotype and generally cause the smallest variation in the phenotype, following the principle of strong causality.[19a] These mutations occur most frequently to maximize the exploitation of the given topology's associated search space. The other mutations are very disruptive as they change the topology of the network, allowing for exploration of the search space associated with different topologies.

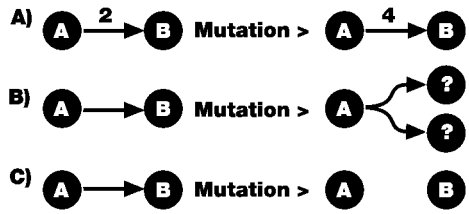

Figure 6. *Types of mutations in descending order of frequency. A) Rate-constant mutation, B) Reaction-type change, C) Reaction deletion or addition. Rate-constant mutations occur most frequently, as they are the least disruptive. Reaction deletion or addition occurs least frequently, as they are the most disruptive to the functioning of the network.*

The product of the base mutation rate for a given offspring and the rate for a given mutation type determines the probability that a given mutation type will occur. For example, given a range of base mutation rates of $^1/_3$ to $^1/_5$ and mutation rates for rate constants of $^1/_2$, connections types of $^1/_4$ and adding/removing connections of $^1/_6$:

- For the offspring with the highest mutation rate, each rate constant has a $^1/_6$ chance of mutating, each connection has a $^1/_8$ chance of mutating and, for each connection, there is a $^1/_{12}$ chance of a connection being added or removed.
- For offspring with the lowest mutation rate, each rate constant has a $^1/_{15}$ chance of mutating, each connection has a $^1/_{20}$ chance of mutating and, for each connection, there is a $^1/_{30}$ chance of a connection being added or removed.

Results

After much testing and refinement, Lakhesis has evolved networks capable of computing multiplication by constants, square roots, cube roots and natural logarithms. Multiplication by a constant is a trivial computation for a reaction network, and was only employed in the initial stages to test the software. Networks capable of computing the square root proved to be relatively easy to evolve, with more than 75% of total runs returning a viable network. Networks of the cube root and log variety were much more difficult to evolve; here less than 10% of attempted runs returned functioning networks. Figure 7 shows the step-like changes in fitness as evolution proceeds, which are typical of genetic algorithms. Each step represents a new and better solution appearing in the population.

Square-root networks

As mentioned previously, it was very common to evolve a network able to compute the square root. In fact an entire family of networks was generated. A sample of these networks is illustrated in Figure 8.

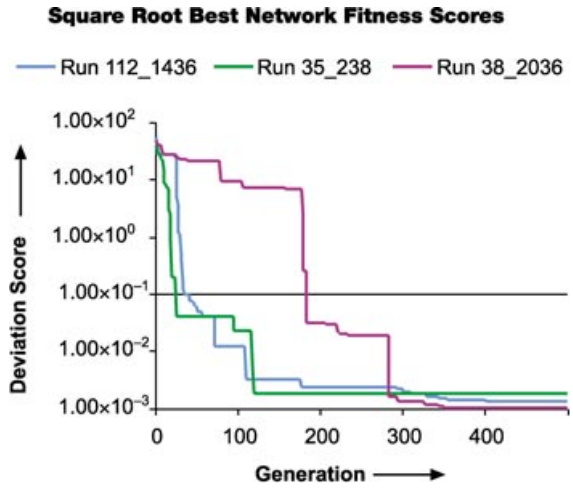

Figure 7. *Comparison of decreasing deviation scores in three selected runs evolving a network capable of computing a square root. The scores plotted are the lowest deviation score found in the given generation. Once the deviation score falls below 1, it indicates that a viable topology has been found. In most runs, a viable topology is found within the first 50 generations.*

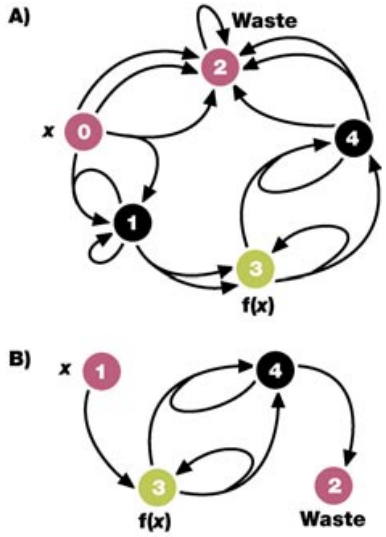

Figure 9. *Network A) illustrates a typical network as it emerges from an evolution run. Many connections are redundant, and only a core set of reactions contributes to the phenotype. Network B) illustrates the same network but with redundant connections removed and rate constants adjusted. The numbers next to the reactions are values for the rate constants. Network B) is able to compute the square root exactly.*

The networks evolved by Lakhesis are complex and highly connected. In all cases, evolved networks contained redundant connections, either reactions that carried no flux, reactions that consumed and produced the same node or reactions that connected the same nodes but in multiple ways. In order to study the networks more easily, all redundant connections were removed at the final generation. The examples shown in Figure 8 are examples of networks after redundant connections have been removed. The pruned square-root networks are concise and in many cases have an appealing symmetry. Network A shown in Figure 9 illustrates a typical network as it emerges from a simulation; B shows the same network but with redundant connections removed.

The network shown in Figure 9 is one of the most common motifs to evolve. By solving the steady-state solution from the differential equations for this system, it is easily shown that it is capable of computing the square root exactly.

Differential equations and steady-state solution for the system shown in Figure 9B:

$$\frac{dn_3}{dt} = k_1 n_1 - k_2 n_3 n_4$$

$$\frac{dn_4}{dt} = k_3 n_3 - k_4 n_4$$

$$n_3 = \sqrt{\frac{k_1 k_3 n_1}{k_2 k_4}}$$

So long as $k_1 k_3 = k_2 k_4$, n_3 is exactly the square root of the input node, n_1.

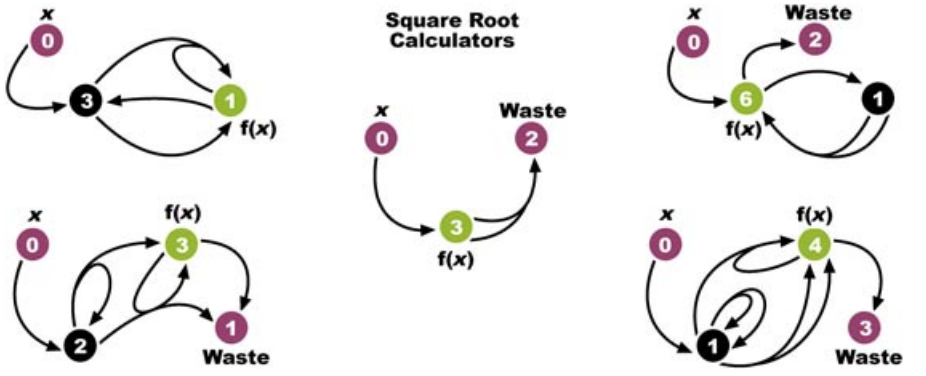

Figure 8. *Evolved square root networks. Most of these networks are able to compute the square root exactly. x designates the input node, and f(x) the readout node. Redundant connections have been removed or combined. In several networks, it may appear that mass conservation has been violated; however, there are implied waste nodes not indicated in the figures. The presence or absence of waste nodes does not affect the computation.*

Analysis of the network's evolution shows that the square-root module is a derived characteristic, as shown in Figure 10. Network A occurred in generation 185 and only has the first part of the module, 1+4→4, with two separate uni connections cycling between nodes 1 and 4. Through chance mutations, this ancestral network is eventually replaced by a network with a square-root module, shown as the final winning network B in generation 499. While both topologies are capable of computing a perfect square root, chance bestowed

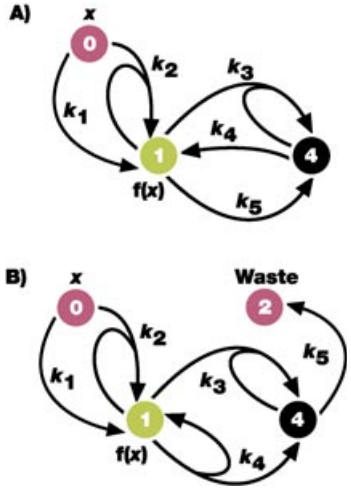

Figure 10. *Comparison of an ancestral network A, found in generation 185, to the final network B, found in generation 499. Both networks compute the square root exactly for all points tested after their rate constants have been adjusted. In both networks, mass conservation is achieved by implied loss from reactions* k_2 *and* k_3*. In network B, there is additional conservation from an explicit reaction given by* k_5*. Note the root-finding module between nodes one and four in network B.*

rate constants on the square-root lineage that yielded a more accurate solution, allowing them to replace their non-square-root competitors. In this particular example, several equally viable topologies may have popped into existence between generations 185 and 499, only to suffer extinction after being repeatedly dealt losing combinations of rate constants. This serves as a reminder that evolutionary computation cannot be depended upon to return the single best solution, and biological evolution can only seize upon the opportunities afforded by chance.

An interesting variation on the square root circuit shown in Figure 9b is given in Figure 11. In this network, an additional reaction exists between nodes 1 and 2. This network will solve a second-order polynomial equation in which the coefficients of the quadratic equation are functions of the rate constants and input node.

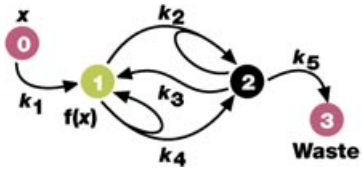

Figure 11. *The quadratic module solves the equation* $ax^2 + bx + c = 0$*. The coefficients,* a, b, *and* c *are formed from combinations of the rate constants and the input species,* n0*.*

Cube-root networks

More interesting were attempts to evolve a network capable of finding the cube-root of an input node. These proved to be much more difficult to evolve and only roughly one in ten of all simulations yielded a successful cube-root network. The unsuccessful networks were mainly networks that could replicate the training data but could not generalize beyond the training set. Figure 12 illustrates two cube-root networks.

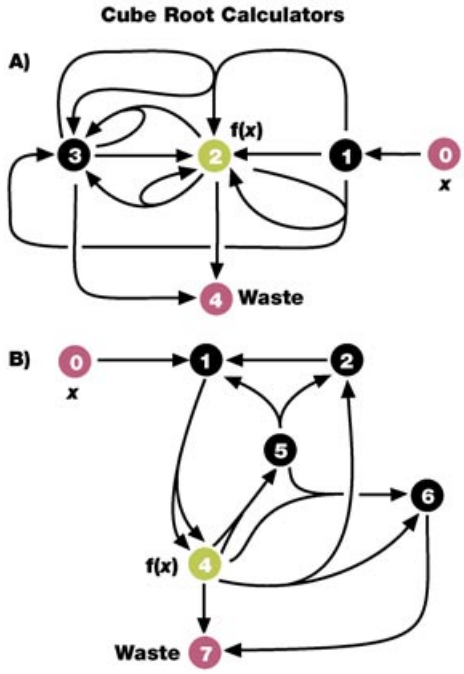

Figure 12. *Example cube root networks evolved by Lakhesis. Cube-root networks are more complex and considerably more difficult to analyze than square-root networks. The node marked with* x *is the input node and the node marked with f(x) is the output, as measured after the system reaches a steady state.*

While it is interesting that the cube-root networks appear quite different, the differences become striking when the networks are examined in detail. While it was quite simple to solve the steady-state equations and prove unambiguously that the square-root networks computed exactly, the same could not be done for the cube-root networks. Even though attempts to prove by hand that these networks could compute cube roots exactly were unsuccessful, computer simulations indicate that the networks can solve a cubic equation very accurately over at least ten orders of magnitude.

As the steady-state solutions could not be calculated by hand, Mathematica 5 was used. For network A in Figure 12, Mathematica was able to find a closed solution, but it was over ten pages of dense algebra. Network B was more amenable to algebraic analysis, and a solution derived by Mathematica was reduced to a simple quintic polynomial after some simplification. This result suggested to us that the second network was not in fact solving the cube root exactly but might have been approximating the solution using a fifth order Taylor expansion.

The first network (12A) is of much more interest and is redrawn in Figure 13 to highlight a particular part in grey. Many readers will immediately recognize this motif to be the quadratic-equation solver shown in Figure 11.

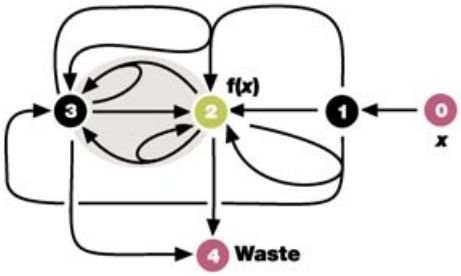

Figure 13. *Cubic solver network from Figure 12A, redrawn to highlight the modular structure. The grey panel highlights the quadratic module present in the cube-root network.*

It seems that this is an example of functional modularity that arose during the evolution of the network. While this motif is critical to the functioning of the network, it is unknown whether the motif is actually operating as a quadratic-equation solver in this cube-root network. Especially intriguing is the existence of a 16th century algorithm by Cardano for computing cube roots, that depends on solving a quadratic.[34]

Attempts to evolve networks capable of computing natural logarithms were only partially successful. The networks that evolved could not compute natural logarithms outside the given training set. This may be because log is defined as a series expansion, and the networks simply reproduced the series only as far as it matched the training set.

Discussion

This paper presents some preliminary results on the artificial evolution of biochemical networks capable of performing computations. The program Lakhesis was able to accurately simulate and evolve models of chemical networks, and was used to show that an evolutionary strategy is capable of evolving networks that can perform relatively complex computations, such as square roots and cube roots. In the case of the evolution of square-root networks, an entire family of networks that could perform the computation was discovered. More interesting was the evolution of networks capable of solving cubic equations. One could imagine that solving square or square-root problems should be relatively straightforward since squaring is already available in the form of bimolecular reactions; this was illustrated by the ease of evolving square-root networks. However, solving cube roots is more problematic, as illustrated by the much greater difficulty in evolving cube-root networks.

What was interesting and rather surprising was the appearance of a modular structure in one of the cube-root networks, where the module was capable of solving quadratic equations. This quadratic module is especially interesting in light of a 16th century algorithm by Cardano for solving cubic equations that uses the solution to a quadratic as part of the solution process.[34] It is currently unknown whether the network has stumbled upon the same 16th century algorithm, but the evidence is very suggestive.

Since there is modularity in the artificially evolved networks, it might be tempting to suggest combining the modules to generate new functionality. For example, a simple idea might be to put two square-root circuits one after the other so that the result of the first feeds into the second. The net effect of this would be a network capable of computing roots to the power of four. However, such a process is not so simple and would not actually work. The problem is well known in electronics and involves the notion of impedance. Take, for example, the process of connecting a speaker to an amplifier. The speaker, termed the load, will draw current from the amplifier. This in turn can affect the output voltage on the amplifier, which, if not curtailed, will ultimately cause the amplifier's performance to deteriorate and lead to distortion in the speaker output. Electronic engineers take great pains to make sure that different modules match so that they do not interfere with each other in this manner. In the example of the amplifier and speaker, the solution is to add negative feedback from the output of the amplifier to its input; this stabilizes the output voltage as the load draws current. In the case of our simple root-four solver, we are confronted with exactly this same problem. If one square-root circuit is attached to another, the second circuit (the load), will draw mass (current) from the first circuit. This will cause changes in the output species level (voltage) that will ultimately cause further changes in the internal dynamics of the network. As a result, the computational performance of the network will deteriorate and result in gross computational errors. Nature may cope with this by matching one signalling module to another, just as electronic engineers must. For example, such matching may exist in the MAPK pathway, where it is known that negative feedback occurs across the MAPK stack.[5] While it is generally unknown what the matching circuits might look like, we believe they must be present in order for the signalling networks to function effectively.

The preliminary results have shown it is possible to evolve networks with a desired functionality and provided a proof of concept that will allow us to develop this work further. There are several areas of study that are of particular interest and that we intend to pursue. The first area is the systematic investigation of how the various mutations rates and mutation types affect the performance of the evolution. Another is to increase the flexibility of the networks by allowing variable numbers of nodes. In addition, since the goal is to build a library of different signal-processing elements, it is highly desirable to evolve not just basic mathematical functions but a variety of networks. These networks would be of biological interest as they can be compared to real signalling networks.

In the long term, we are very interested in exploring the evolution of modularity and intend to devise a more biologically inspired genome in order to implement robust crossover. The initial version of Lakhesis attempted to use crossover; however, crossover invariably resulted in offspring that were no fitter than their parents. Part of the problem might have been the crossing of heterologous networks rather than homologous networks; however, this is an active area of research.

Finally, one observation stood out above all: the evolved networks look nothing like real biological signalling networks.

Whereas biological signalling networks are composed of repeated cyclic motifs, the artificially evolved networks lacked this kind of architecture. We believe this is due to a number of factors, one of which is the mode by which biological systems evolve. One of the main methods by which biological networks evolve is through gene duplication. For example, many of the kinases and phosphates that are found in biological signalling networks can be traced to common ancestors. It is presumed that these ancestral genes duplicated and diverged and were utilized to enlarge the original network. Therefore instead of evolving completely new architectures, gene duplication, particularly module duplication, allows networks to grow without changing the basic architecture and results in similar motifs being appended to an existing network. Our simulations had no such constraints; this allowed the networks to take on any pattern, and they were thus highly variable in architecture. We intend to look seriously at gene duplication as a means to evolve artificial networks.

Computational Methods

The software (called Lakhesis) used to carry out the evolutionary simulations was developed by using VB.NET. The program was configured to run up to 500 generations of 500 networks. For the Runge–Kutta algorithm, a time step of 0.05 was used, and the reactions were simulated for 400 time points or until steady state was reached. In the initial stages of development, output runs from Lakhesis were compared to output from Jarnac 2.0[32] in order to verify the accuracy of the solutions. In addition, all final networks and examples shown in this paper were tested in Jarnac 2.0 to verify their accuracy.

Typical runs lasted from twenty minutes to a few hours for the more demanding objective functions. All simulations were run on a 1.67 GHz machine with Windows XP. The software is freely available for download from http://www.sys-bio.org/research/lakhesis.htm.

Acknowledgements

A.D. and H.M.S. acknowledge the generous support of the Keck Graduate Institute in funding this work.

Keywords: evolution · modularity · signal transduction · simulation · systems biology

[1] B. D. Gomperts, I. M. Mramer, P. E. R. Tatham, *Signal Transduction*, Academic Press, **2002**.
[2] T. M. Yi, Y. Huang, M. I. Simon, J. Doyle, *Proc. Natl. Acad. Sci. USA* **2000**, *99*, 4649 – 4653.
[3] C. V. Rao, A. Arkin, *Annu. Rev. Biomed. Eng.* **2001**, *3*, 391 – 419.
[4] U. Bhalla, P. Ram, R. Iyengar, *Science* **2002**, *297*, 1018 – 1023.
[5] H. M. Sauro, *Curr. Proteomics* **2004**, *1*, 67 – 81.
[6] B. Kholodenko, *Eur. J. Biochem.* **2000**, *267*(6), 1583 – 1588.
[7] D. Wolf, A. Arkin, *Curr. Opin. Microbiol.* **2003**, *6*, 125 – 134.
[8] J. Hasty, D. McMillen, J. J. Collins, *Nature* **2002**, *420*, 224 – 230.
[9] M. Okamoto, T. Sakai, K. Hayashi, *Biosystems* **1987**, *21*, 1 – 11.
[10] M. Okamoto, T. Sakai, K. Hayashi, *Biotechnol. Bioeng.* **1988**, *32*, 527 – 537.
[11] "A Biochemical nand Gate and Assorted Circuits" H. M. Sauro in *Modern Trends in Biothermokinetics* (Eds.: S. Schuster, M. Rigoulet, R. Ouhabi, J.-P. Mazat), Plenum Press, New York, **1993**, pp. 133 – 140.
[12] A. Arkin, J. Ross, *Biophys. J.* **1994**, *67*, 560 – 578.
[13] A. Hjelmfelt, J. Ross, *Phys. D* **1995**, *84*, 180 – 193.
[14] M. Magnesco, *Phys. Rev. Lett.* **1997**, *78*, 1190 – 1193.
[15] J. J. Tyson K. C. Chen, B. Novak, *Curr. Opin. Cell Biol.* **2003**, *15*, 221 – 231.
[16] E. Alm, A. Arkin, *Curr. Opin. Struct. Biol.* **2003**, *13*, 193 – 202.
[17] L. Landweber, E. Winfree, *Evolution as Computation*, Springer, New York, **1998**.
[18] For genetic algorithms see: a) D. E. Goldberg, *Genetic Algorithms*, Addison Wesley Longman, Reading, **1989**; b) J. H. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, **1992**.
[19] For evolutionary strategies see: a) I. Rechenberg, *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*, Frommann–Holzboog, Stuttgart, **1973**; b) H.-P. Schwefel, *Numerical Optimization of Computer Models*, Wiley, Chichester, **1981**.
[20] For evolutionary programming see: a) D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE Press, Piscataway, **1995**; b) L. J. Fogel, A. J. Owens, M. J. Walsh, *Artificial Intelligence through Simulated Evolution*, Wiley, New York, **1966**.
[21] H. Kacser, R. Beeby, *J. Mol. Evol.* **1984**, *20*, 38 – 51.
[22] D. Bray, S. Lay, *Biophys. J.* **1994**, *66*, 972 – 977.
[23] A. Gilman, J. Ross, *Biophys. J.* **1995**, *69*, 1321 – 1333.
[24] M. Tsuchiya, J. Ross, *J. Phys. Chem. A* **2001**, *105*, 4052 – 4058.
[25] J. Ross, *Acc. Chem. Res.* **2003**, *36*, 839 – 847.
[26] P. François, V. Hakim, *Proc. Natl. Acad. Sci. USA* **2004**, *101*, 580 – 585.
[27] A. Shin, H. Iba, *Genome Inf. Ser. (Genome Information Workshop)* **2003**, *14*, 94 – 103.
[28] A. Stephani, J. C. Nuno, R. Heinrich, *J. Theor. Biol.* **1999**, *199*, 45 – 61.
[29] O. Ebenhoh, R. Heinrich, *Bull. Math. Biol.* **2003**, *65*, 323 – 357.
[30] J. R. Koza, H. F. Bennett, D. Andre, *Genetic Programming III: Automatic Programming and Automatic Circuit Synthesis*, **1999**, Morgan Kauffman.
[31] J. R. Koza, W. Mydlowec, G. Lanza, J. Yu, M. A. Keane, *Pac. Symp. Biocomput.* **2001**, *6*, 434 – 445.
[32] "Jarnac: A System for Interactive Metabolic Analysis", H. M. Sauro in *Animating the Cellular Map: Proceedings of the 9th International Meeting on BioThermoKinetics*, (Eds.: J.-H. S. Hofmeyr, J. M. Rohwer, J. L. Snoep), Stellenbosch University Press, **2000**.
[33] J. Maynard-Smith, *Evolutionary Genetics,* 2nd ed., Oxford University Press, **1998.**
[34] W. Durham, *Journey through Genius: The Great Theorems of Mathematics*, Wiley, New York, **1990**, pp. 133 – 154**.**

Received: May 30, 2004

