

Towards Evolving Electronic Circuits for Autonomous Space Applications

Jason D. Lohn

Computational Sciences Division
NASA Ames Research Center
Moffett Field, CA 94035
jlohn@ptolemy.arc.nasa.gov

Silvano P. Colombano

Computational Sciences Division
NASA Ames Research Center
Moffett Field, CA 94035
scolombano@mail.arc.nasa.gov

Gary L. Haith

Recom Technologies Corp.
NASA Ames Research Center
Moffett Field, CA 94035
haith@ptolemy.arc.nasa.gov

Dimitris Stassinopoulos

Computational Sciences Division
NASA Ames Research Center
Moffett Field, CA 94035
stassi@ptolemy.arc.nasa.gov

Abstract— The relatively new field of Evolvable Hardware studies how simulated evolution can reconfigure, adapt, and design hardware structures in an automated manner. Space applications, especially those requiring autonomy, are potential beneficiaries of evolvable hardware. For example, robotic drilling from a mobile platform requires high-bandwidth controller circuits that are difficult to design. In this paper, we present automated design techniques based on evolutionary search that could potentially be used in such applications. First, we present a method of automatically generating analog circuit designs using evolutionary search and a circuit-construction language. Our system allows circuit size (number of devices), circuit topology, and device values to be evolved. Using a parallel genetic algorithm, we present experimental results for five design tasks. Second, we investigate the use of coevolution in automated circuit design. We examine fitness evaluation by comparing the effectiveness of four fitness schedules. The results indicate that solution quality is highest with static and coevolving fitness schedules as compared to the other two dynamic schedules. We discuss these results and offer two possible explanations for the observed behavior: retention of useful information, and alignment of problem difficulty with circuit proficiency.

TABLE OF CONTENTS

1. INTRODUCTION
2. CIRCUIT REPRESENTATION
3. DESIGN TASKS
4. EXPERIMENTAL RESULTS
5. COEVOLUTION AND OTHER FITNESS SCHEDULES
6. EXPERIMENTAL SETUP
7. EXPERIMENTAL RESULTS
8. DISCUSSION

1 INTRODUCTION

Although the underlying concepts of using simulated evolution to manipulate hardware are decades old, it is only in recent years that research in this area has attracted significant interest [17, 6, 18, 19]. The nascent field of *evolvable hardware* studies how simulated evolution can reconfigure, adapt, and design hardware structures in an automated manner. The field is almost exclusively concerned with electronic circuitry, but application areas where other types of physical structures are designed or adapted by artificial evolution certainly fall within the purview of evolvable hardware (e.g., designs of trusses, antennas).

Research on using evolution to automatically create novel circuit designs can be classified into two categories: analog and digital circuitry. In the digital circuit domain, the Field-Programmable Gate Array (FPGA) [22] has played a crucial role. The FPGA is a chip that contains a large array of logic gates (e.g., AND, OR) and a user-modifiable interconnection network to connect elements together. The distinguishing feature of the FPGA is its ability to be programmed as many times as needed. Each time the FPGA is re-programmed (the act of which is commonly called reconfiguration) for new functionality, the interconnections and gate logic change. This software-changes-hardware paradox is reconciled by the fact that the physical devices on the chip never change, only the way signals are routed through the chip changes. In practice, these chips are typically programmed to perform a desired function and are rarely re-programmed. However, in an evolvable hardware setting, these chips are re-programmed over and over as evolution repeatedly tests new designs. The important point is that the hardware is nestled *inside* of the evolutionary process, allow-

ing for rapid solution testing (a common bottleneck in many evolutionary algorithms). Some of the pioneering work in this area was done by Higuchi [5] and Thompson [21].

The other circuit design domain and the focus of this paper is analog circuitry. Analog circuits are of great importance in electronic system design since the world is fundamentally analog in nature. While the amount of digital design activity far outpaces that of analog design, most digital systems require analog modules for interfacing to the external world. Techniques for analog circuit design automation began appearing about two decades ago (e.g., [20]). Efforts using techniques from evolutionary computation have appeared over the last few years. These include the use of genetic algorithms (GAs) [7] to select filter component sizes [8], to select filter topologies [3], and to design operational amplifiers using a small set of topologies [12]. Various analog filter design problems have been solved using genetic programming (e.g., [11]), and an overview of these techniques, including eight analog circuit synthesis problems, is found in [10].

In space applications, automated/evolutionary design of analog circuitry could hold many benefits, especially for controller hardware. In problems where actuator outputs need to be rapidly modulated in response to sensor feedback, analog circuits have some clear advantages over digital circuits. Digital control circuits necessitate a costly bandwidth-limiting analog to digital conversion (ADC) of the sensor signal and then a reverse conversion (DAC) from the processed digital signal to the analog actuator control output. These transformations lose information and introduce latency. In many tasks the high frequency component of the sensor signal that is lost in the analog to digital conversion is crucial to usefully controlling the actuators. For example, to implement force control of a robotic manipulator, a critical portion of the strain sensor signal is lost during conversion from analog to digital. In such cases, analog circuits are ideal because they provide a very high bandwidth sensor-to-motor signal transformation and avoid any time-consuming conversion between analog and digital signals.

Although robotic controller technology has greatly improved over the last decade, certain advanced robotic applications cannot be realized due to limitations in controller processing speed, size, and power consumption. Two such applications are robotic drilling and real-time rover-astronaut interaction. Drilling from a mobile platform is difficult because the drill bit forces and position can shift rapidly causing damage to the bit – a fast force control loop could alleviate these difficulties. Rover-astronaut interactions are severely limited by rover speed and autonomy, both of which would benefit greatly from fast control loops. To overcome the main limitation of using analog controllers, namely the difficulty designing analog circuits, the evolvable hardware techniques presented below could be applied to automatically generat-

ing controller circuit designs.

The remainder of the paper is as follows. First we discuss the genetic representation of analog circuits and describe the genetic algorithm that is used. Then we cover the design tasks and the experimental results from our evolutionary design program. A description of a coevolutionary method follows, including results in automated amplifier design. Lastly we discuss our conclusions relating to potential space-related applications of evolvable hardware.

2 CIRCUIT REPRESENTATION

In designing an effective circuit representation for use in evolutionary search, the following properties are among the most desirable. First, the representation should permit any circuit or at least a wide range of circuits to be represented. If it is known *a priori* that certain topologies are well suited to a specific design task, topological restrictions inherent in the representation may be beneficial since the search space will be reduced. Conversely, not having this limitation may bring to light novel designs that human designers have never envisioned. Second, the genotype conversion algorithm (the circuit constructing process) should run as fast as possible. Clearly if numerous traversals of the circuit graph structure are required in order to guarantee a valid circuit graph, the performance hit will be commensurate. Third, the representation should be syntactically closed so that genetic operators do not create invalid circuit graphs¹ from those that are valid. The circuit representation we present here was designed to have these properties.

Circuit designs are constructed by an automaton that is programmed via a set of low-level instructions. This automaton is programmed in a small “language” designed for building circuits. In its current incarnation, the language contains only component-placing instructions (e.g., control instructions are not included). This language has the desirable property that virtually all possible sequences of instructions result in a valid electrical circuit. This property is important because it greatly limits the “out-of-bounds” regions of the search space containing invalid circuit graphs. Thus, evolutionary search will spend nearly all its time generating valid circuit graphs. While this is a beneficial, non-trivial achievement, we do lose the ability to generate every possible circuit topology. This is not considered a drawback for the circuit types we investigated since a vast number of topologies and existing circuit designs could be encoded using this approach.

Each instruction places a circuit component and directs the movement of the automaton. The five basic instruction types are: *x*-move-to-new, *x*-cast-to-

¹Note that a graph could be a valid circuit graph, yet not make sense as an electrical circuit – for example, dissimilar voltage sources connected in parallel.

previous, x -cast-to-ground, x -cast-to-input, x -cast-to-output, where x can be replaced by R (resistor), C (capacitor), L (inductor), or transistor configuration. In a circuit design task involving only inductors and capacitors (an LC circuit), ten opcodes would be available to construct circuits (five for capacitors and five for inductors).

The meanings of each instruction are summarized in Table 1. The move-to-new instruction places one end of a component at the active node and the other at a newly created node (the “active” node is the current location of the automaton). The newly created node then becomes the active node. The cast-to instructions place one end of the component at the active node and the other at either the ground, input, output, or previously-created node. After executing a cast-to instruction, the automaton remains at the active node. The input and output nodes are the overall input and output nodes of the circuit as opposed to the input and output of the placed component. Illustrations of two instructions that place resistors are shown in Fig. 1.

Instruction	Outgoing Node	Active Node
x -move-to-new	new node	becomes new node
x -cast-to-previous	previous node	unchanged
x -cast-to-ground	ground node	unchanged
x -cast-to-input	input node	unchanged
x -cast-to-output	output node	unchanged

Table 1: Summary of opcode types used in current system. x denotes the component type: resistor, capacitor, inductor, or transistor configuration.

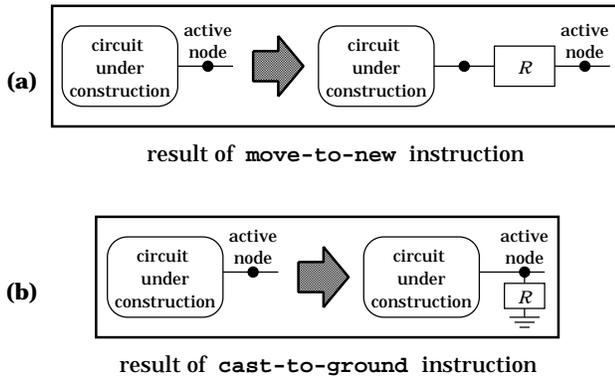


Figure 1: Effect of placing a resistor with (a) move-to-new, and (b) cast-to-ground instructions.

The circuit is constructed by the automaton inside of a template circuit. The design tasks presented here use a template having one input and one output terminal as shown in Fig. 2. An ideal voltage source v_s is connected to ground and to a source resistor R_s . The circuit’s output voltage is taken across a load resistor R_l .

The lists of instructions manipulated by the GA are variable-length lists so that the size of the circuit can be

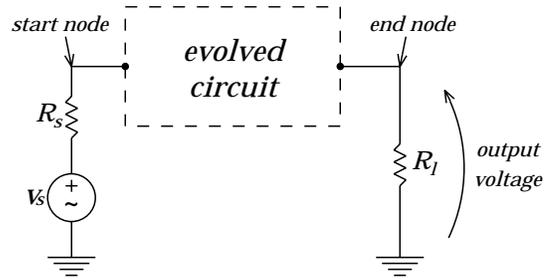


Figure 2: Template circuit: the evolved circuit is located between fixed input and output terminals. v_s is an ideal voltage source, R_s is the source resistance, R_l is the load resistance.

evolved. When the automaton reaches the last component to place in the circuit, we arbitrarily chose to have the last active node connected to the output terminal by a wire (accomplished by connection of a $1\mu\Omega$ resistor). By doing so, we eliminate unconnected branches.

As assembly language instructions are mapped to opcodes, our circuit-placing instructions are mapped to bytecodes. Instructions are represented by up to four bytecodes. For instructions that take a component value as an argument, the first byte is the instruction, and the next three represent the component value (resistance, capacitance, and inductance values). For transistors, component values are not needed. Using three bytes allows the component values to take on one of 256^3 values, a sufficiently fine-grained resolution. The raw numerical value of these bytes was then scaled into a reasonable range, depending on the type of component. Resistor values were scaled sigmoidally between 1 and 100K ohms using $1/(1 + \exp(-1.4(10x - 8)))$ so that roughly 75% of the resistor values were biased to be less than 10K ohms. Capacitor values were scaled between approximately 10 pF and 200 μ F and inductors between roughly 0.1 mH and 1.5 H.

Transistors are current amplifying and switching devices that have three terminals.² In this paper we use bipolar junction transistors as shown in Fig. 3. Using devices with three terminals makes it harder to design a circuit representation that achieves the properties that we desire. If a circuit constructing automaton were to connect one terminal of a transistor at a node, then two active nodes would result each requiring its own automaton. This could happen repeatedly resulting in an exponential growth of automata constructing the circuit in parallel. Two problems are obvious: how will the multiple constructing “threads” interconnect, and how will the dangling nodes that will likely appear at the end of the circuit constructing process be handled? To allow interconnections between constructing threads, one can introduce a spatial dimension and let the automata form interconnections as they criss-cross each other’s path. To

²Four if the substrate terminal is included, but we connect the substrate to ground and hence ignore it.

handle the dangling node problem, one can deterministically tie dangling nodes to each other, to internal nodes, or to the output node for example. Another solution is to simply prune those nodes. Although we have considered these and other solutions, a simpler way of handling transistors is also a viable alternative: treating them as having only two terminals.

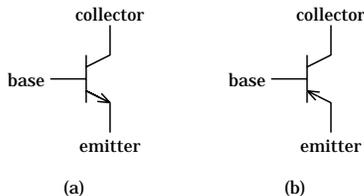


Figure 3: Bipolar junction transistor symbols: (a) npn; (b) pnp.

To work with transistors as devices with two terminals, we have the third terminal hardwired (fixed) to one of the following pre-existing circuit nodes: ground, power supply (positive or negative), input, output, the previously placed node, or even to itself. Such a scheme allows a wide variety of configurations. To understand these configurations we label the terminals in a generic way: incoming, outgoing, and fixed (see Fig. 4). The incoming terminal is the terminal that the circuit constructor will connect to the active node. The outgoing terminal will become the new active node (for move-to instructions) or it will be cast to a pre-existing circuit node. The fixed terminal is hardwired as its name implies.

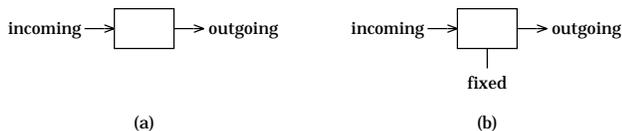


Figure 4: Labeling of terminals: (a) devices with two terminals have incoming and outgoing terminals; (b) devices with three terminals are treated as two-terminal devices by having a fixed connection at the third terminal.

To give a sense of the types of transistor configurations possible, the chart in Fig. 5 illustrates 52 configurations for an npn transistor whose base terminal is designated incoming. Each entry shows the connections that the automaton makes when executing the instruction listed in the first column. The last two columns show self-connections, some of which are frequently used by circuit designers. Similar charts can be produced for npn transistors having the collector and emitter serve as the incoming terminal, as well as the three analogous charts for pnp transistors. There are configuration redundancies so that each chart will not have exactly 52 configurations. In addition we exclude emitter-collector

self-connections since this shorts out the transistor.

This approach to representing circuits embodies the desirable properties outlined above. The encoding has syntactic closure since any combination of instructions produces a valid circuit graph, and since every instruction contained in the genome results in a circuit component, there are no non-coding genome segments. The circuit construction process is $O(n)$ since it does not require any repair operations (e.g., removal of unconnected nodes). Lastly, this approach can generate a wide range of circuit graph topologies. The topological restriction is as follows: circuit branches off of the main constructing thread cannot, in general, contain more than one node (there are some exceptions to this). The constructing thread is the sequence of components that are created by the move-to-new instructions. The constructing thread itself can be of varying lengths and can contain both series and parallel configurations. In spite of these limitations, our system allows the creation of circuits with a large variety of topologies, including numerous topologies seen in hand-designed circuits.

3 DESIGN TASKS

The design tasks considered in this paper are analog circuits for filtering and amplification applications.

A low-pass filter is a circuit that allows low frequencies to pass through it, but stops high frequencies from doing so. In other words, it is frequency selective in that it “filters out” frequencies above a specified frequency. The unshaded area in Fig. 6 depicts the region of operation for low-pass filters. Below the frequency f_p the input signal is passed to the output, potentially reduced (attenuated) by K_p decibels (dB). This region is known as the passband. Above the frequency f_s , in the region is called the stopband, the input signal is markedly decreased by K_s decibels. Between the passband and stopband the frequency response curve transitions from low to high attenuation. The parameter located in this region, f_c , is known as the cutoff frequency.

One of the design tasks concerned designing a circuit within the class of “Butterworth” filters. Butterworth filters are very common and circuits that implement them are readily found in filter design tables [9]. The attenuation (negative gain) of Butterworth filters is of the form $\sqrt{1/(1 + (f/f_c)^{2N})}$ where f is the input frequency and N is the order of the filter. The higher order a filter has, the sharper the “knee” of its gain curve, and the more complex the circuit. A plot of the attenuation for a third-order Butterworth filter is shown in Fig. 7.

The amplifier design task chosen was the inverting operational amplifier. Such a circuit has found wide application and is considered one of the workhorses of analog circuit design. Figure 8 shows the symbol and connections for an ideal inverting amplifier. This circuit generates an output voltage (v_o) that consists of the input

Instruction Type	Outgoing and Fixed Terminals Connections											
	emitter to GND	collector to GND	emitter to PS	collector to PS	emitter to INPUT	collector to INPUT	emitter to OUTPUT	collector to OUTPUT	emitter to PREV	collector to PREV	emitter to base	collector to base
MOVE-TO-NEW	new	new	new	new	new	INPUT	new	OUTPUT	new	PREV	new	new
CAST-TO-PREVIOUS	previous	previous	previous	previous	previous	INPUT	previous	OUTPUT	N.A.	N.A.	previous	previous
CAST-TO-INPUT	input	input	input	input	N.A.	N.A.	input	OUTPUT	input	PREV	input	input
CAST-TO-OUTPUT	output	output	output	output	output	INPUT	N.A.	N.A.	output	PREV	output	output
CAST-TO-GND	N.A.	N.A.	gnd	gnd	gnd	INPUT	gnd	OUTPUT	gnd	PREV	gnd	gnd

Figure 5: Transistor configurations showing the outgoing and fixed connections when the incoming terminal is the transistor’s base terminal. Terminals labeled in upper case letters denote the fixed terminal connection. In the last two columns the fixed connection is a self-connection. “PS” denotes the power supply (only the positive version is shown), and “N.A.” denotes “not applicable.” Only npn transistors are shown although analogous configurations are present for pnp transistors.

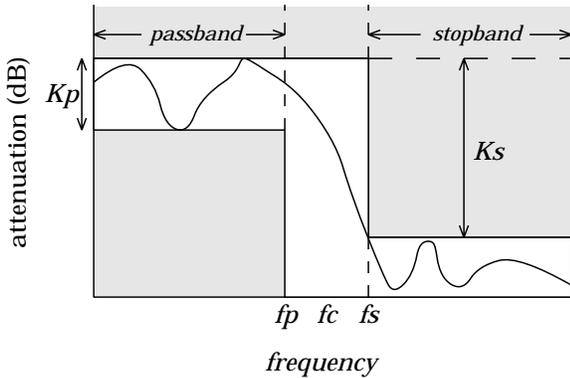


Figure 6: Low-pass filter terminology and specifications. The shaded regions represent out-of-specification areas. An example frequency response curve that meets specifications is shown.

voltage (v_i) multiplied by a gain factor, A . Voltage gain is thus equivalent to v_o/v_i . It is common to express gain values in decibels (dB) using $20 \log_{10}(A)$. Amplifiers may be either inverting or non-inverting, where an inverted output signal has a 180° phase shift compared to the input. The dc gain of the amplifier refers to the gain when only constant voltage/current sources are applied. The linearity of the gain is the degree to which the gain re-

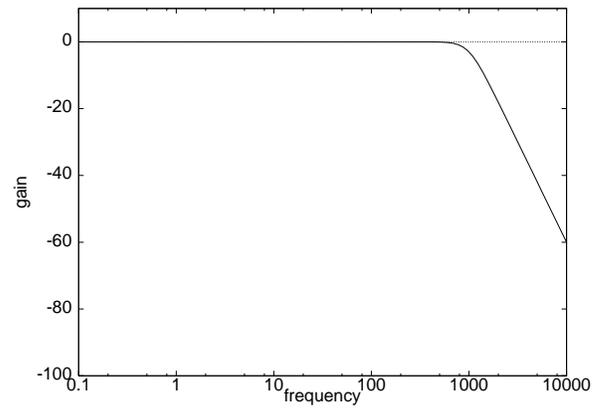


Figure 7: Gain on a logarithmic amplitude scale for a third-order Butterworth filter.

mains constant across input voltages: ideally the voltage transfer characteristic (v_o vs. v_i) should be linear. The dc component that shifts the entire signal up or down is called the dc bias of the circuit. Power dissipation is the amount of power used by the circuit and is indicative of the amounts of current flowing in the circuit. For simple amplifiers, there are publications available that catalog many designs. Since there are numerous parameters in amplifier design (e.g., input/output impedance, power

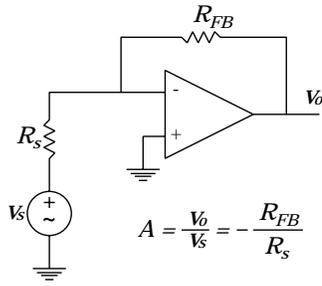


Figure 8: Ideal inverting amplifier showing how gain is set by the ratio of the feedback to source resistor.

dissipation, distortion, common-mode rejection, power supply rejection), the design task can become quite challenging and typically requires an experienced designer. For the amplifier design experiments below, we take into account four objectives: dc gain, linearity of gain, dc bias, and power dissipation.

4 EXPERIMENTAL RESULTS

Filter Design Tasks

Three filter design experiments were performed. In each experiment, 10 runs were performed and we present the circuit having the highest fitness value across all runs. The experiments increased in difficulty so that filter 3 represents a challenging design task, while filter 1 is least challenging. Table 2 lists the target specifications for each of the experiments.

Filter No.	f_p (Hz)	f_s (Hz)	K_p (dB)	K_s (dB)
1	100	4000	1.29	27.12
2	925	3200	3.01	22.00
3	1000	2000	0.01	63.50

Table 2: Target specifications for filter design tasks.

The GA parameters remained the same within a given experiment, but varied in the number of evaluations (circuit simulations): filter 1 runs had 30,000 evaluations, filter 2 had 3.6 million, and filter 3 had 1 million. These values were arrived at by experimentation and constrained by practical issues such as the availability of workstations.

For the filter experiments, fitness was calculated to promote the regression of the evolved circuit’s frequency response toward that of the target. Error values were computed as the absolute value of the difference of the individual’s output and the target output. These error values were summed across evaluation points to arrive at a fitness value.

Electronic Stethoscope Circuit – The first filter design task was set up to generate a filter suitable for use in

an electronic stethoscope. In this application, it is desired to filter out the extraneous high-frequency sounds picked up by a microphone which make it difficult to listen to (low-frequency) bodily sounds (e.g., a heart beating). As such, the frequency response specifications do not need to be extremely accurate since the human ear cannot discern frequencies that are close together. The target frequency response data was taken from an actual electronic stethoscope, which was built with a cutoff frequency of 796 Hz corresponding to an output voltage of approximately 1 volt. This circuit is relatively easy to design and so we chose it as our first design task. The cc-bot instruction set consisted of ten instructions, five for resistors and five for capacitors, which allowed for the construction of an RC low-pass filter. The evolved circuit is shown in Fig. 9(a) and its frequency response, which matches almost exactly the target is shown in Fig. 9(b). It was found in generation 3 of a 10-generation run that had a population size of 3000, an indication that this design task was relatively easy. The circuit exhibits the standard design for simple low-pass filters: a resistor (R_2) in series with the source to form a voltage divider at low frequencies (C_1 open), and a capacitor (C_1) across the output to short it at high frequencies.

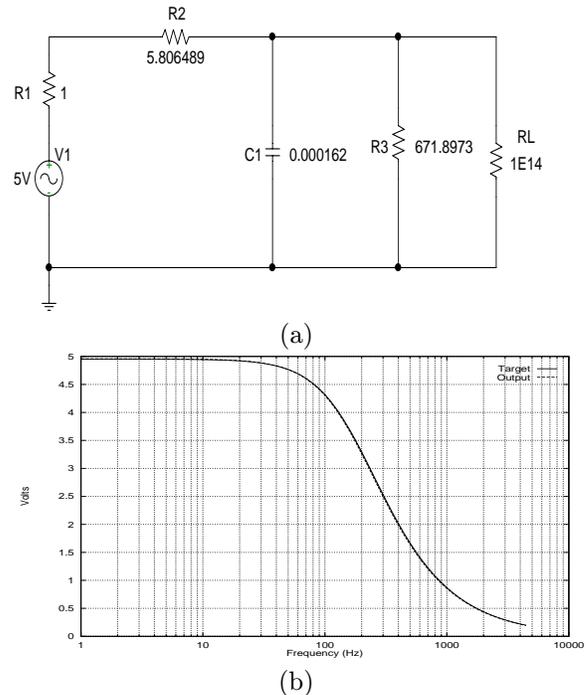


Figure 9: (a) Evolved low-pass filter for use in an electronic stethoscope (units are ohms and farads); (b) Nearly identical frequency response curves for evolved and actual electronic stethoscope circuit. The frequency axis is scaled logarithmically.

Butterworth Low-pass Filter – The second low-pass filter design task had specifications that were more difficult to

achieve than the first filter: both the passband and the stopband were longer, thus requiring the transition to be sharper. We chose a circuit that can be built using a 3rd-order Butterworth filter and having a frequency response of the form seen in Fig. 7. The specifications are listed under filter number two in Table II.

Such a filter design can be derived using a ladder topology containing two capacitors and one inductor and component values found in published tables. Because we wanted to design an LC low-pass filter, the cc-bot instruction set consisted of only capacitor and inductor instructions. The evolved circuit that meets these specifications is shown in Fig. 10 and its frequency response is shown in Fig. 11. It was found in generation 22 of a run that had a population size of 18,000.

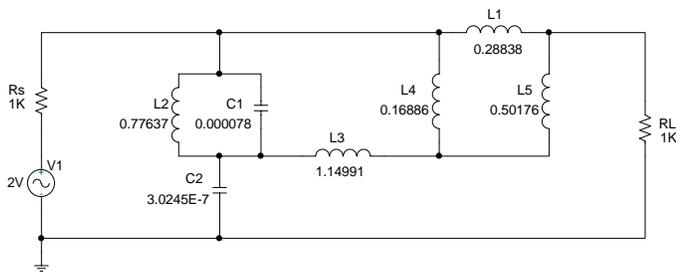


Figure 10: Evolved 3rd-order Butterworth low-pass filter (units are ohms, farads, and henries).

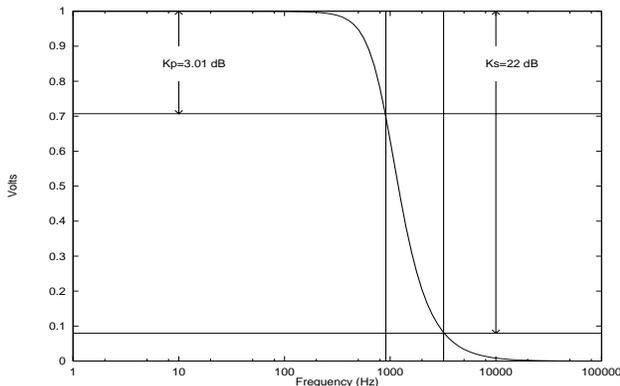


Figure 11: Frequency response curve for evolved 3rd-order Butterworth low-pass filter. Attenuation specifications are also shown. The frequency axis is scaled logarithmically.

Third Low-pass Filter – The third low-pass filter design task had specifications that were the most stringent: in addition to the passband and stopband being increased, the attenuation parameters were tightened (see Table II). These specifications are similar to the fifth-order elliptic filter described in [10]. In that work, the evolved LC circuit satisfies $K_p = 0.3$ dB and $K_s = 60$ dB. Another

evolved low-pass filter circuit [23] had the same stopband and passband frequencies, but less demanding attenuation specifications ($K_p = 1.6$ dB and $K_s = 24.8$ dB). The evolved circuit is shown in Fig. 12 and its frequency response is seen in Fig. 13. Micro-ohm resistors were added as a convergence aid for the circuit simulator, and can be ignored for analytical purposes. This circuit was found in generation 997 of a run that had a population size of 1000.

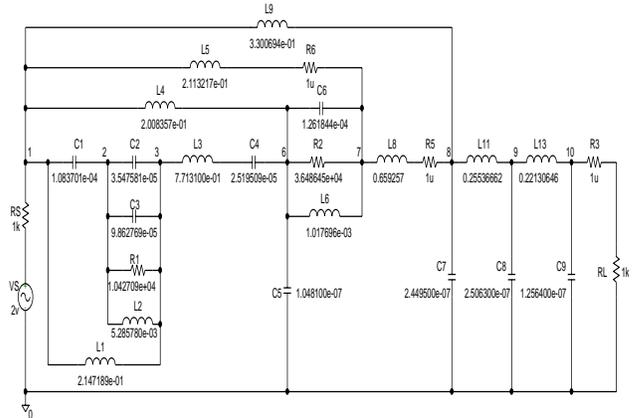


Figure 12: Evolved circuit satisfying target specifications for filter number three.

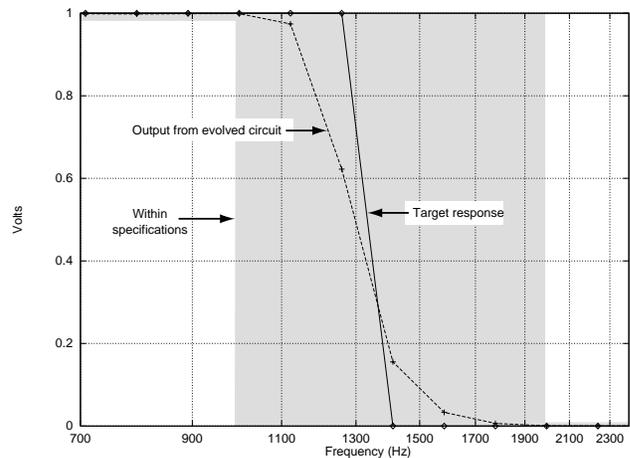


Figure 13: Frequency response for filter number three.

Amplifier Design Tasks

Two amplifier design experiments were performed. In each experiment, 10 runs were performed and we present the highest performance circuits found across all runs. The goal was to design an inverting amplifier capable of a dc voltage gain up to a maximum of either 100 dB or 120 dB, while minimizing dc bias and maximizing linearity over the dc gain. Population size was set to 1200 individuals, and each run proceeded for 5000 generations, giving a total of 6 million circuit evaluations

per run. The difference between the two sets of experiments is that in the first set, the maximum gain was set to be 120 dB, and in the second set, 100 dB was the maximum gain. The maximum gain possible is set by using feedback resistors (labeled R_{FB}). For an ideal inverting amplifier (as shown in Fig. 8), the magnitude of the gain of the amplifier is simply R_{FB}/R_S , where R_S is the source resistor. Fitness was calculated in a manner similar to the work on amplifiers in [10]. An error value is computed as the sum of the dc gain penalty (the target gain minus the observed gain), the dc bias (zero dc bias is ideal), and the degree to which the dc gain is linear.

75 dB Inverting Amplifier – In the first set of experiments the maximum voltage gain was set at 120 dB (10^6). The amplifier having the best performance had a dc gain of 74.53 dB (5324.40). Figure 14 shows the schematic for this circuit. It was found in generation 4866, and had a dc bias of 3.64 volts and a power dissipation of 0.82 watts.

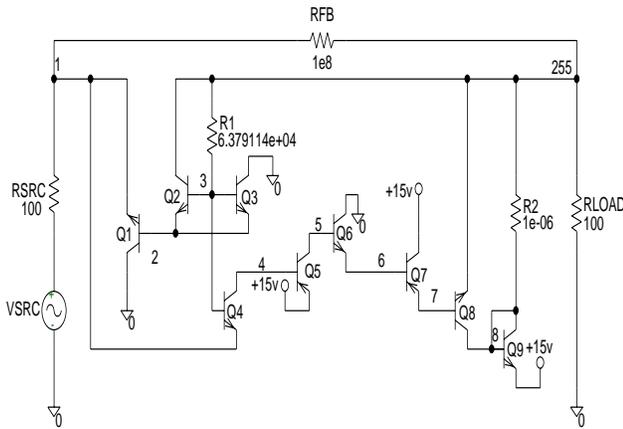


Figure 14: Circuit schematic of evolved 75 dB amplifier.

The dc behavior is best understood by examining the major current pathways in the circuit. The current through the load is the key quantity since it is converted to a voltage by the load resistor and hence forms the circuit's output. Nearly all of the dc current flowing through the load resistor originates from the power supply connected to transistor Q7's collector. Q7 is biased in such a way as to supply Q8's base with approximately 36.4 mA of current. This current is divided so that 18.1 mA flows out of Q8's emitter and 18.3 mA out of Q8's collector. Resistor R2 is a tiny resistance that was positioned in order to connect transistor Q9 to the output (the last component is forced to connect to the output terminal). Thus R2 can be ignored, and transistor Q9's 18.4 mA current flows into the output node. Currents are summed at node 255 to give the load current of 36.4 mA which flows through the load resistance to give 3.64 volts output. Because there is a negligible amount of current flowing through transistors Q1 through Q4, the

utility of these transistors is unclear. Components that are essentially non-functional, are quite commonly seen in evolutionary design applications. Figure 15(a) shows the time domain response. Amplification of a 1 kHz sine wave having a 1 microvolt amplitude can clearly be seen. Figure 15(b) shows the frequency response. The ac gain remains flat at 74.36 dB until it loses 3 dB at 7.59 kHz (its 3 dB bandwidth). Figure 16 shows the dc transfer characteristic. The dc bias of 3.64 volts can be seen at the voltage input of zero volts.

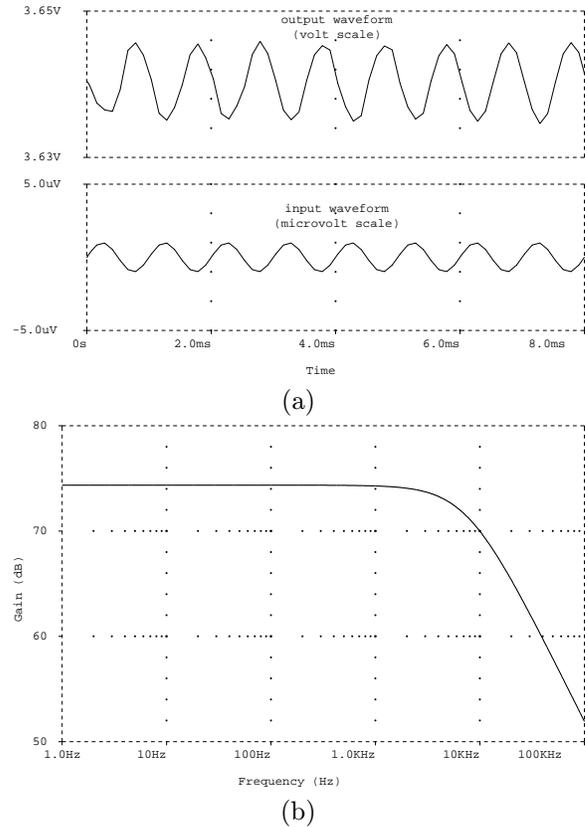


Figure 15: Small signal behavior of 75 dB evolved amplifier: (a) time domain input waveform is 1 kHz (bottom) which is inverted and amplified (top); (b) frequency response showing 3 dB bandwidth of 7.59 kHz.

85 dB Amplifier – In the second set of amplifier experiments the maximum voltage gain was set at 100 dB (10^5). The amplifier having the best performance had a dc gain of 85.41 dB (18,642.33). Figure 17 shows the schematic for this circuit. It was found in generation 3635, and had a dc bias of 5.44 volts and a power dissipation of 8.17 watts. The dc current delivered to the load is mostly supplied by the 15 volt battery attached to the collector of transistor Q7. Transistor Q7 is conducting with the sum of its base and collector currents flowing out of its emitter. Q7's base current of 13 mA is supplied by transistor Q6. As in the previous amplifier, the utility of transistors Q1 through Q3 is unclear.

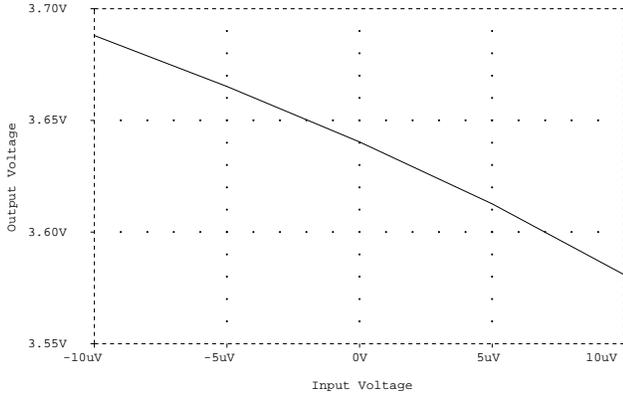


Figure 16: DC transfer characteristic of 75 dB amplifier.

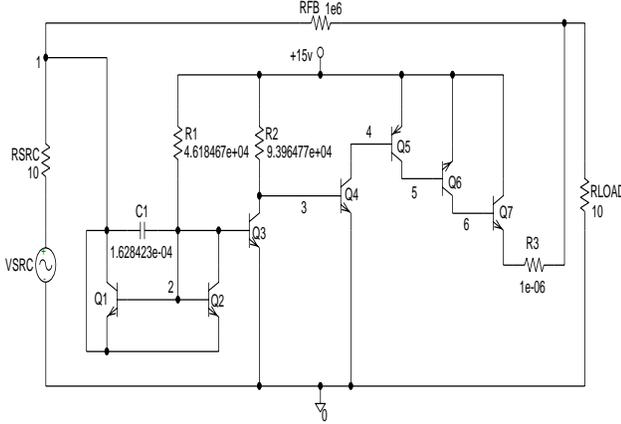


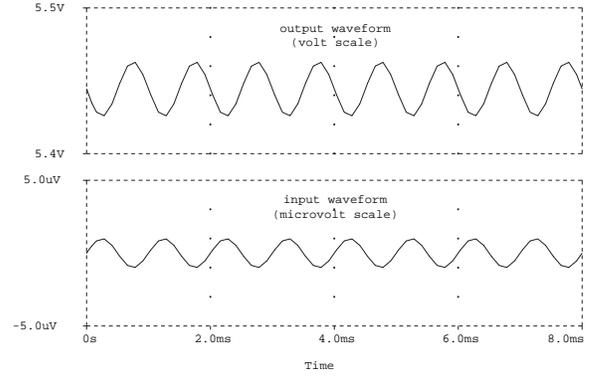
Figure 17: Circuit schematic of evolved 85 dB amplifier.

Input signal inversion and amplification are seen in Figure 18(a) which shows the time domain response to an ac input of 1 microvolt at 1 kHz. The circuit has a flat-band gain of 85.46 dB and a 3 dB bandwidth of 282.8 kHz (Figure 18(b)). The 3 dB bandwidth is significantly better than the previous amplifier. Figure 19 shows the dc transfer characteristic. The dc bias of 5.44 volts can be seen at the voltage input of zero volts. The slope, the magnitude of which is the gain, is negative since the amplifier is inverting the signal.

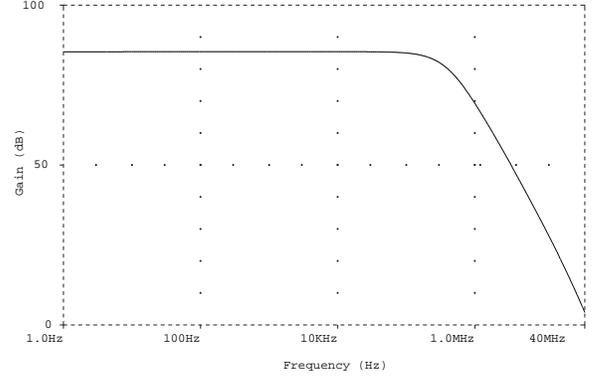
5 COEVOLUTION AND OTHER FITNESS SCHEDULES

Using the same amplifier design task described above, we now turn our attention to comparing four methods of evolutionary search. A *fitness schedule* is construct that modifies how a circuit's fitness is computed during the GA run. For example, one could use a fitness function f_1 during generation 1, f_2 during generation 2, etc. Four fitness schedules, one of which embodies coevolutionary concepts, are discussed below.

The first, called *static* for short, refers to a single fit-



(a)



(b)

Figure 18: Small signal behavior of 85 dB evolved amplifier: (a) time domain input waveform is 1 kHz (bottom) which is inverted and amplified (top); (b) frequency response showing a flatband gain of 85.46 dB.

ness function that is used to evaluate every individual in every generation of the run. The second was a fixed fitness schedule, meaning that the fitness function was modified in a pre-determined manner every k generations, for some constant k . Thus the same fitness function evaluates groups of kM individuals, where M is the population size. The third fitness schedule we call *adaptive* because it can change the fitness function dynamically based on the performance of the population. The fourth fitness schedule is coevolutionary search whereby a second of population consisting of problem difficulties (target vectors) evolve based on the performance of the circuits in the main population.

Static Fitness Schedule

The static fitness schedule is simply the standard evaluation technique in genetic algorithms [7]: a single fitness function is used to evaluate all individuals throughout the run. The fitness function used is similar to those described in [10, 13]. Briefly, it is a sum of normalized error values, where the errors are the shortfalls from the desired objectives: dc gain, dc bias, power dissipation, and the linearity of the dc gain. The gain is the slope

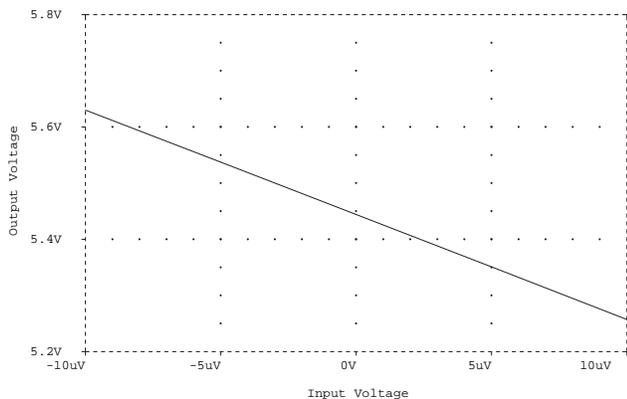


Figure 19: DC Transfer characteristic of 85 dB amplifier.

of the dc transfer characteristic (i.e., the output voltages when the input voltage is swept across five input voltages). The slope, m , is calculated by using the endpoints of the transfer characteristic. The linearity of the gain is computed as $|m - m_l| + |m - m_r|$, where m_l is the slope of the line segment formed by the two leftmost output voltages and m_r is analogous for the two rightmost output voltages. The dc bias is simply v_o when $v_i = 0$ volts, and power dissipation is the amount of power consumed during circuit operation. The gain objective was 60.0 dB, the bias and power dissipation objectives were 1.0 volt and 1.0 watt, respectively, and the linearity objective was 10.0. These values were chosen based on our previous work [14]: they represent a moderately difficult design task that we knew to be solvable.

Fixed Fitness Schedule

The fixed fitness schedule is a pre-determined schedule of fitness function modifications. As used in the experiments below, the difficulty-level of the fitness function is increased every 50 generations. With a total of 5000 generations, this allowed for a total of 100 “difficulty steps.” Each of the fitness functions used over the course of the run are of the same form as the fitness function used in the static schedule above. Writing our gain, bias, power, and linearity objectives as a *target vector*, $\langle G, B, P, L \rangle$, we specified that the difficulty level begins at $\langle 1.0, 10.0, 10.0, 1000.0 \rangle$ (easiest) and ends at $\langle 60.0, 1.0, 1.0, 10.0 \rangle$ (most difficult). The increases in difficulty are then evenly divided over the 100 steps, per objective. This is admittedly an arbitrary schedule, but that is an inherent property of a fixed schedule - it is subject to the biases of the implementor. Such biases can be advantageous if knowledge of the fitness landscape is known *a priori*, and potentially disadvantageous otherwise.

Adaptive Fitness Schedule

The adaptive fitness schedule is identical to the fixed schedule described above except in the following regard: difficulty is incremented “on-demand,” whenever the current difficulty is solved by at least one circuit in the population. As in the fixed schedule case, 100 difficulty steps are provided for. If a circuit solves the 100th fitness function before 5000 generations, it has successfully found a compliant circuit, and the run halts. On the other hand, if 5000 generations elapse and a compliant circuit is not found, the run halts at whatever difficulty level it has reached.

Coevolving Fitness Schedule

The main difference between the coevolving fitness schedule and the other dynamic schedules is the introduction of a second population consisting of target vectors (tv). The first population of circuits remains the same as in the other fitness schedules. The target vector population consists of individuals that specify problem difficulty. As described above, target vectors are denoted $\langle G, B, P, L \rangle$, representing gain, bias, power dissipation, and gain linearity, respectively. The individual targets are threshold values – a target is “solved” if a circuit’s performance equals or surpasses (either above or below, as appropriate) the threshold specified. For example, $\langle 63.0, 0.6, 0.8, 9.5 \rangle$ solves $\langle 60.0, 1.0, 1.0, 10.0 \rangle$, but $\langle 58.0, 0.6, 1.2, 18.0 \rangle$ does not. As with the other fitness schedules, the ideal target vector used was $\langle 60.0, 1.0, 1.0, 10.0 \rangle$. The gain target is satisfied if a circuit’s gain was 60.0 decibels or greater. The three remaining targets were satisfied if the circuit’s performance is less than or equal to the target values.

Target vectors are represented as a list of floating point values that are mutated individually by randomly adding or subtracting a small amount (5% of the largest legal value). Single point crossover was used, and crossover points were chosen between the values.

Fitness of individual circuits in the main population was computed as follows. Circuit i “plays” each target vector in the second population and a score, s_i , is computed:

$$s_i = \sum_{j \in \widehat{tv}_i} \frac{1}{\text{total \# circuits that solve } tv_j}$$

where \widehat{tv}_i is the set of target vector indexes such that circuit i solves tv_j . Note that the denominator in the above fraction is guaranteed to be greater than or equal to one due to the restriction on j . Then s_i is normalized linearly between its upper and lower bounds such that 0.0 is the best score and 1.0 the worst:

$$F(\text{circuit}_i) = 1.0 - s_i/M_2$$

where M_2 is the size of the target vector population. The effect of s is to reward circuits that solve the more

difficult target vectors. A target vector has the greatest difficulty level when exactly one circuit can solve it. If many circuits can solve a particular target vector, the fitness contribution in s is shared among the circuits [16].

Fitness of an individual target vector is computed as follows. Let x_j denote the number of circuits that solve tv_j , and M_1 be the circuit population size. The fitness is essentially x_j , scaled and normalized, with a tractability constraint:

$$F(tv_j) = \begin{cases} 1.0 & x_j = 0 \\ \frac{1}{(M_1-1)}(x_j - 1.0) & x_j \geq 1 \end{cases}$$

The tractability constraint gives a target vector a score of 1.0 (the “worst” score) when no circuits can solve it. This puts pressure on the target vector population to pose difficult, yet solvable problems to the circuit population.

6 EXPERIMENTAL SETUP

Using the four fitness schedules described above, 25 runs using each schedule were made resulting in a total of 100 runs. The same pseudo-random number generator seed was used across each set of four distinct fitness schedules so that the generation zero individuals would be identical. Common to each run were the following parameter settings: population size was 600, crossover rate was 80%, mutation rate was 5%. For the coevolution runs, the target vector population used the following parameters: population size was 600, crossover rate was 80%, mutation rate was 50%. Because crossover points were chosen between target vector values, this mutation rate was set high to encourage new values to appear in the population, not just those produced in generation 0.

Evolution of amplifier designs was accomplished using the system described in [13]. Briefly, circuits are represented as lists of circuit-construction instructions that program an automaton to design a circuit. Resistors, capacitors, and bipolar junction transistors were the allowed components. The method of incorporating transistors is described in [14]. Circuits were required to contain at least 10 components up to a maximum of 150.

7 EXPERIMENTAL RESULTS

To assess the quality of each fitness schedule, we examined the highest fitness circuits from each run. The performance of these circuits is quantified in corresponding *output vectors* which, like target vectors, specify gain, bias, power dissipation, and linearity values. Table 3 gives the mean values of individual objectives across output vectors for each fitness schedule. The data suggest that static and coevolving fitness schedules performed better than fixed and adaptive schedules. Another way of measuring the quality of the fitness schedules is to look

fitness schedule	gain [dB]	bias [volts]	power [watts]	linearity [unitless]
static	44.47	0.35	0.69	49.28
fixed	47.59	0.64	1.21	96.63
adaptive	54.13	1.23	1.96	340.74
coevolving	46.71	0.15	0.41	189.75

Table 3: Mean values from the performance of the best circuits found under 25 runs of each fitness schedule. The ideal target vector was $\langle 60.0, 1.0, 1.0, 10.0 \rangle$.

fitness schedule	mean	std. dev.
static	2.12*	0.67
fixed	1.48	1.12
adaptive	1.16	1.18
coevolving	2.08*	0.49

Table 4: Mean and standard deviation for the number of objectives solved for 25 runs of each fitness schedule. Means marked with asterisks (*) are not significantly different from each other, and are significantly different ($p < 0.02$) from those means without asterisks.

at the number of objectives solved in each run (assuming each of the four objectives is of equal importance). Table 4 shows the mean and standard deviation for the number of objectives solved for each schedule.

Here the relationship among the schedules is clearer: static and coevolving fitness schedules performed nearly the same and did better than the performance of the fixed and adaptive schedules. A two-tailed t-Test showed that the static and coevolving means are not significantly different from each other, and are significantly different ($p < 0.02$) from the fixed and adaptive means.

One of the motivations behind using coevolutionary search is the notion that the problem difficulty is adjusted automatically, rather than having to manually specify it. To get a sense of how coevolution accomplished this, Figure 20 shows four plots (one for each target objective), each containing 25 curves (fitted using a fourth-order polynomial). The plots show how the values of the best target vectors found in each generation fluctuated during the run. The thick curve represents the run that found a compliant circuit (i.e., it solved $\langle 60.0, 1.0, 1.0, 10.0 \rangle$).

What is most striking is the way coevolution, within the first few generations, reduced the demands for gain performance because it was the most difficult criterion to meet. Just as rapidly, the other three objectives were made more demanding because they were relatively easy to satisfy. Then as the circuit population scored better in gain, it did so at the expense of power and linearity: both power and linearity are seen peaking near generations

1000-2000.

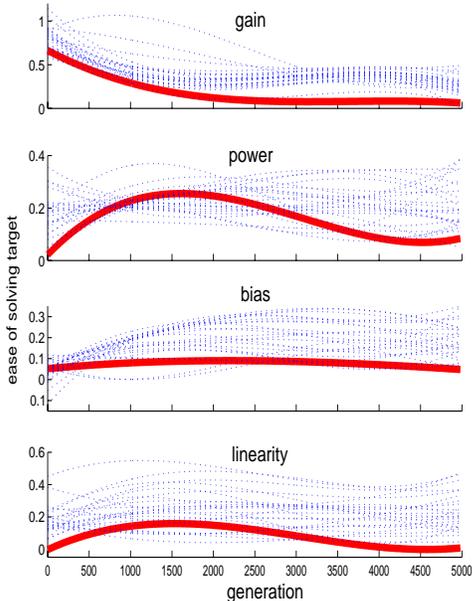


Figure 20: Highest fitness target vector values over the course of all coevolution runs. The y-axes represent the difficulty of the objective with 0.0 being the target (or most difficult) value, and 1.0 being the easiest objective value. The thick curves represent the run that found a compliant circuit. Curves were fitted using a fourth-order polynomial, and therefore sometimes appear above 1.0 and below 0.0.

From the results it is seen that static and coevolutionary fitness schedules outperformed the fixed and adaptive schedules. Although it is not completely clear why this happened, we can offer potential advantages of the static and coevolutionary schedules relative to the fixed and adaptive schedules. First, because a static fitness function induces a fitness landscape that never changes over the course of evolution there is never the possibility of getting “thrown off” a gradient (as would be the case if the fitness function changed). Second, we designed coevolution so that it would keep the level of problem difficulty near the leading edge of circuit proficiency. Developmental theory suggests (e.g., [1]) that keeping task difficulty in line with solution performance aids learning. Third, the fixed and adaptive schedules are potentially “handicapped” by the somewhat arbitrary choice of manually-crafted schedules.

8 DISCUSSION

As a step towards demanding space-related applications, we have presented encouraging results of an evolvable hardware system capable of automatically desinging analog circuits. We showed that a linear circuit representation and evolutionary search can automatically produce circuit designs of low to medium difficulty in two appli-

cations. Detailed simulations of the evolved designs suggest that all are electrically well behaved and thus suitable for physical implementation. The circuit representation method devised permits a wide range of circuits to be constructed, and results in a construction process that is unburdened with repair operations. In addition, the representation is syntactically closed, making it well suited for evolutionary search. For other applications, the instruction set can be easily extended to incorporate other devices not mentioned, such as CMOS transistors. The main limitation of our approach is the inherent restriction on circuit topologies. Such restrictions can be overcome by augmenting the instruction set, and this is one line of investigation we are pursuing. To gain performance on par with circuits designed by engineers, it will be necessary to place further constraints into the fitness functions. For example, practical amplifiers are typically judged by a dozen or so specifications. To evolve an amplifier that would perform as well would require using a multiobjective fitness function that accounts for each specification. Progress towards this goal was made in the results from coevolutionary search.

Dynamic fitness schedules can help evolutionary search because they encourage the population of circuits to follow potentially better trajectories through the solution space. Such trajectories could guide evolution in many ways, for example they could amplify weak gradients in the fitness landscape, “steer around” meta-stable solution states [16], and usefully decompose or simplify the problem by providing partial reinforcement for intermediate solutions [4]. As an illustration, an amplifier made up of a single wire has excellent performance in terms of bias, linearity and power dissipation, but has zero gain. Adding some components to the circuit might increase the gain, but only at the cost of a dip in performance on the other three criteria. Thus, if evolved with a static fitness schedule (assuming equally-weighted objectives), the single wire presents evolutionary search with a meta-stable state that is highly attractive and potentially quite difficult to escape. In contrast, the fixed fitness schedule in the present amplifier design task encourages all of the performance objectives (gain, power dissipation, bias, and linearity) to be solved in parallel by evolution. Likewise, coevolution tends to work on gain early in evolution and to scale back the requirements on bias, power and linearity until circuits are performing fairly well on gain.

In conclusion, static and coevolving fitness evaluations did relatively well in our amplifier design task. Based on our previous work in evolving amplifier designs, we suspected that the static technique would be able to solve this design task. We find it very encouraging that coevolution performed on par with static fitness schedules and intend to pursue coevolutionary search in future circuit design tasks, especially for electronic control applications.

References

- [1] J.L. Elman, *Incremental Learning, or the Importance of Starting Small*, Tech. Rept. 9101, Center for Research in Language, University of California, San Diego, CA, 1991.
- [2] G. Gielen, W. Sansen, *Symbolic Analysis for Automated Design of Analog Integrated Circuits*, Boston, MA: Kluwer, 1991.
- [3] J.B. Grimbleby, "Automatic Analogue Network Synthesis using Genetic Algorithms," *Proc. First Int. Conf. Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA)*, 1995, pp. 53-58.
- [4] G.L. Haith, S.P. Colombano, J.D. Lohn, D. Stassinopoulos, "Coevolution for Problem Simplification," *Proc. 1999 Genetic and Evolutionary Computation Conference, (GECCO-99)*, 1999, to appear.
- [5] T. Higuchi, M. Iwata, I. Kajitani, H. Iba, Y. Hirao, T. Furuya, B. Manderick, *Evolvable Hardware and Its Applications to Pattern Recognition and Fault Tolerant Systems*, (Lecture Notes in Computer Science), vol 1062, Berlin: Springer-Verlag, pp. 118-135, 1996.
- [6] T. Higuchi, M. Iwata, Eds., *Evolvable Systems: From Biology to Hardware*, Proc. of the First International Conference on Evolvable Systems, (Lecture Notes in Computer Science), vol 1259, Berlin: Springer-Verlag, 1997.
- [7] J.H. Holland, *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press, Ann Arbor, 1975.
- [8] D.H. Horrocks, Y.M.A. Khalifa, "Genetically Derived Filters using Preferred Value Components," *Proc. IEE Colloq. on Linear Analogue Circuits and Systems*, Oxford, UK, 1994.
- [9] L.P. Huelsman, *Active and Passive Analog Filter Design*, New York: McGraw-Hill, 1993.
- [10] J.R. Koza, F.H. Bennett, D. Andre, M.A. Keane, F. Dunlap, "Automated Synthesis of Analog Electrical Circuits by Means of Genetic Programming," *IEEE Trans. on Evolutionary Computation*, vol. 1, no. 2, July, 1997, pp. 109-128.
- [11] J.R. Koza, F.H. Bennett, J.D. Lohn, F. Dunlap, M.A. Keane, D. Andre, "Use of Architecture-Altering Operations to Dynamically Adapt a Three-Way Analog Source Identification Circuit to Accommodate a New Source," in *Genetic Programming 1997 Conference*, J.R. Koza, K. Deb, M. Dorigo, D.B. Fogel, M. Garzon, H. Iba, and R.L. Riolo, (eds), Morgan Kaufmann, 1997, pp. 213-221.
- [12] M.W. Kruiskamp, *Analog Design Automation using Genetic Algorithms and Polytopes*, Ph.D. Thesis, Dept. of Elect. Engr., Eindhoven University of Technology, Eindhoven, The Netherlands, 1996.
- [13] J.D. Lohn, S.P. Colombano, "Automated Analog Circuit Synthesis using a Linear Representation," *Proc. of the Second Int'l Conf on Evolvable Systems: From Biology to Hardware*, Springer-Verlag, Berlin, 1998, pp. 125-133.
- [14] J.D. Lohn, S.P. Colombano, "A Circuit Representation Technique for Automated Circuit Design," *IEEE Trans. on Evolutionary Computation*, to appear.
- [15] E.S. Ochotta, R.A. Rutenbar, L.R. Carley, "Synthesis of High-Performance Analog Circuits in AS-TRX/OBLX," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 273-294, 1996.
- [16] C.D. Rosin, R.K. Belew, *New Methods for Competitive Coevolution*, Tech. Rept. CS96-491, Department of Computer Science and Engineering, University of California, San Diego, 1996.
- [17] E. Sanchez and M. Tomassini, Eds., *Toward Evolvable Hardware: The Evolutionary Engineering Approach*, (Lecture Notes in Computer Science), vol 1062, Berlin: Springer-Verlag, 1996.
- [18] M. Sipper, D. Mange, A. Perez-Urbe, Eds., *Evolvable Systems: From Biology to Hardware*, Proc. of the Second International Conference on Evolvable Systems, (Lecture Notes in Computer Science), vol 1478, Berlin: Springer-Verlag, 1998.
- [19] M. Sipper, D. Mange, Eds., Special Issue on Evolvable Hardware *IEEE Transactions on Evolutionary Computation*, vol 3, no 3, 1999.
- [20] G.J. Sussman, R.M. Stallman, "Heuristic Techniques in Computer-Aided Circuit Analysis," *IEEE Trans. Circuits and Systems*, vol. 22, 1975.
- [21] A. Thompson, "An Evolved Circuit, Intrinsic in Silicon, Entwined with Physics," in T. Higuchi, M. Iwata, Eds., *Evolvable Systems: From Biology to Hardware*, Proc. of the First International Conference on Evolvable Systems, (Lecture Notes in Computer Science), vol 1259, Berlin: Springer-Verlag, pp. 390-405, 1997.
- [22] J. Villasenor, W. Mangione-Smith, "Configurable Computing," *Scientific American*, vol 276, no 6, pp. 67-71, June 1997.
- [23] R.S. Zebulum, M.A. Pacheco, M. Vellasco, "Comparison of Different Evolutionary Methodologies Applied to Electronic Filter Design," *1998 IEEE Int. Conf. on Evolutionary Computation*, Piscataway, NJ: IEEE Press, 1998, pp. 434-439.

Jason D. Lohn received the B.S. degree in electrical engineering from Lehigh University, Bethlehem, PA, and the M.S. and Ph.D. degrees in electrical engineering from the University of Maryland at College Park. He is a Computer Scientist at NASA Ames Research Center, Moffett Field, CA. Previously, he was a Visiting Scholar at Stanford University, a Research Assistant at the University of Maryland, and an Associate Engineer at IBM Corporation. His research interests are in automated hardware synthesis, complex adaptive systems, cellular automata, self-replicating systems, and neural computation.

Gary L. Haith received the B.A. in Anthropology from the University of California, San Diego, and the M.A. and Ph.D degrees Psychology (Computational Neuroscience) from Stanford University. He is a member of Phi Beta Kappa and a former National Merit Scholar. He is a Computer Scientist at NASA Ames Research Center, Moffett Field, CA. His research interests include robotic control, collective robotics, coevolutionary search, and machine learning.

Silvano P. Colombano received the M.A. in Physics and the Ph.D. in Biophysical Sciences from the State University of New York at Buffalo. He has spent most of his working career at NASA Ames Research Center first as a researcher in Closed Ecological Life Support Systems and later in Artificial Intelligence. He began the development work on the Astronaut Science Advisor (a.k.a. PI-in-a-box) and managed the project until its deployment on SLS-2 (Space Shuttle STS-58) in 1993. Since then he has been doing research and development work in Artificial Neural Networks, Genetic Algorithms and Artificial Life. He now leads the BIOS (Biologically Inspired Optimization Systems) group in the Computational Sciences Division at NASA Ames.

Dimitris Stassinopoulos is a Research Scientist at NASA Ames Research Center. He has pursued his research interests at Brookhaven National Laboratory, the Niels Bohr Institute, Denmark, and Oxford University, England. He received a B.A. from the University of Athens, Greece, and a Ph.D. from Boston University, both in Theoretical Physics. He has made contributions in spatially-extended nonlinear systems, fluid-dynamic modeling, machine learning, and origin of life studies.