A More Bio-plausible Approach to the Evolutionary Inference of Finite State Machines

Hooman Shayani
Department of Computer Science
University College London, UK
h.shayani@cs.ucl.ac.uk

Peter J. Bentley
Department of Computer Science
University College London, UK
p.bentley@cs.ucl.ac.uk

ABSTRACT

With resemblance of finite-state machines to some biological mechanisms in cells and numerous applications of finite automata in different fields, this paper uses analogies and metaphors to introduce an element of bio-plausibility to evolutionary grammatical inference. Inference of a finite-state machine that generalizes well over unseen input-output examples is an NP-complete problem. Heuristic algorithms exist to minimize the size of an FSM keeping it consistent with all the input-output sequences. However, their performance dramatically degrades in presence of noise in the training set. Evolutionary algorithms perform better for noisy data sets but they do not scale well and their performance drops as size or complexity of the target machine grows. Here, inspired by a biological perspective, an evolutionary algorithm with a novel representation and a new fitness function for inference of Moore finite-state machines of limited size is proposed and compared with one of the latest evolutionary techniques.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids — *Automatic synthesis*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation — *Automata*; I.2.2 [**Artificial Intelligence**]: Automatic Programming — *Program synthesis*; I.5.1 [**Pattern Recognition**]: Models — *Structural*; J.6 [**Computer Aided Engineering**]: Computer-aided design (CAD)

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

FSM, Finite-State Machine, Evolution, Evolutionary Algorithms, Genetic Algorithms, Grammatical Inference, Bioplausibility, Analogies, Metaphors

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

1. INTRODUCTION

The astonishing complexity of natural organisms, and their remarkable inherent properties like fault-tolerance, immunity, robustness, parallelism and adaptation, has made them good sources of inspiration for engineers. New, emerging paradigms such as Evolutionary Systems, Artificial Neural Networks (ANN), and Artificial Immune Systems (AIS) are all efforts to imitate nature's solutions to create intelligent, adaptive and secure systems. In the same direction, it is expected that an evolutionary developmental neural network on silicon shows desirable features like adaptation, fault-tolerance, and robustness. With that goal in mind, evolving a single digital spiking neuron led to evolving finite state machines.

Finite-state machines (FSM) [12] are used in a variety of applications. They can be found in almost any piece of sequential digital hardware in one way or another [27]. Even a Von Neumann computer could be seen as a large finite-state machine. Particularly, Moore model FSMs [21] are popular in designing high-speed synchronous digital circuits [27]. The same concept is used in software development as well.

Finite-state machines go from one state to another and emit output symbols in response to the input symbols they receive form the environment. This is very similar to the way biological organisms move in their vast multi-dimensional state space from one attractor to another and interact with their environment accordingly. However, biological organisms have very complex gene regulatory networks as their governing mechanism for these state transitions, while a finite-state machine, no matter how complex, can be simply described by a boolean transition function. As nature has evolved these gene regulatory networks to create biological organisms that live on this planet, it is possible to evolve transition functions to create finite-state machines that interact correctly with their environment.

The problem of learning a deterministic finite-state machine from examples could be also viewed as a machine learning problem and has applications in different fields such as formal language theory, syntactic and structural pattern recognition, computational linguistics, computational biology and speech recognition [7]. Different heuristic and evolutionary algorithms were used to tackle this problem [7, 5]. Although evolutionary approaches perform better than heuristic algorithms on noisy datasets [17], they do not scale well and their performance drops as size or complexity of the target machine increases [22].

Here, inspired by a biological perspective, an evolution-

ary algorithm with a novel representation and a new fitness function for inference of Moore finite-state machines of limited size is proposed and compared with one of the latest evolutionary techniques. This method was initially devised to evolve an alternative design for soma unit of a stochastic digital spiking neuron [25].

The next section reviews related literature. Definitions, notation, and the problem statement are described in section 3 before explaining the new evolutionary algorithm in section 4 . The experiment settings are elaborated in section 6 and their results are presented in section 7. Section 8 explains and justifies these results. Section 9 discusses the extended applications of metaphors and analogies in computational evolution. The paper is concluded in section 10 followed by the list of references.

2. BACKGROUND

As a class of grammatical inference problems, learning a deterministic finite-state machine from input-output sequences has been studied by many researchers [7]. It is trivial to synthesize an FSM, consistent with a given data set, simply by constructing a prefix tree acceptor [17]. However, since the synthesized machine is usually expected to generalize the training data set to unseen test data, it is desired to find the smallest consistent FSM. Gold [9] showed that inference of a minimum finite state machine from given data is an NP-complete problem. Even finding a machine with number of states polynomial on the size of the minimum machine is also NP-complete [24]. Manovit, Aporntewan and Chongstitvatana investigated the effects of length [19] and number of input-output training sequences [6] on the correctness of the synthesized machines.

A family of heuristic algorithms, called evidence-driven state merging (EDSM) devised by Lang and Price [16], can process FSMs with large number of states. Oliveira and Silva proposed a new search algorithm and reviewed the state of the art search algorithms for inference of minimum DFAs in [23]. However, these techniques are very sensitive to noise and their performance dramatically decrease in presence of noise in the data sets [17]. Recently, a number of competitions were dedicated to deterministic finite automata inference [16, 15, 10] and provided evidence that evolutionary approaches have a superior performance in extracting a deterministic FSM from noisy data compared to the heuristic methods [17, 5].

Fogel, *et al* [8] were some of the first who used an evolutionary approach to tackle this type of problem. Recently, Niparnan and Chongstitvatana [22] introduced an evolutionary algorithm for inference of Mealy machines using GA and statistical inference of the output function. Later, Lucas and Reynolds [17] used a multi-start hill-climber and a similar statistical method called Smart State Labeling for inference of language acceptors from noisy data. Bongard and Lipson also introduced a coevolutionary approach to active learning of DFAs, which can be used when the target machine or an oracle is available to label input strings [5].

As shown by [22, 17], it is possible to evolve an FSM only by evolving the transition function. This transition function can be described in a Boolean form. Evolutionary algorithms have difficulties evolving large and complex Boolean functions but a developmental process can improve the scalability of the evolution [11] as it uses an implicit mapping from genotype to phenotype. Using Fractal Proteins, Bentley [3, 4] evolved Gene Regulatory Networks, which showed a better evolvability than representations with direct genotype-phenotype mappings. Bentley's primary experiments with on-off fractal proteins (without concentration) produced machines equivalent to finite-state machines [3]. Inspired by that, it is possible to create a more bioplausible representation and a biologically more meaningful fitness function for evolutionary inference of FSMs in hope of gaining evolvability and scalability. This paper proposes a novel indirect genotype-phenotype mapping and a new progressive fitness function for evolutionary inference of Moore Model FSMs using the same output estimation technique of [17] and [22]. A Moore model FSM [21] is used here as it is widely employed in design of synchronous sequential digital circuits. In Moore machines, outputs only depend on the current state, which is in contrast with Mealy machines where outputs are functions of the inputs and the current state [27].

3. PROBLEM FORMALIZATION

A Moore finite-state machine [21] is defined here with a notation similar to [12] and [22].

3.1 Basic Definitions

Definition 1. A Moore finite-state machine is defined as a six-tuple $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where:
Q is the set of states with cardinality $u = |Q|$,
$q_0 \in Q$ is the initial state,
Σ is the input alphabet with cardinality $v = |\Sigma|$,
Δ is the output alphabet with cardinality $w = |\Delta|$,
$\delta(q, a) : Q \times \Sigma \to Q$ is the transition function and,
$\lambda(q) : Q \to \Delta$ is the output function.
A specific state, a specific input and a specific output are represented respectively as q, a, and b. It is possible to describe the transition function $\delta(q, a)$ in a state transition table or illustrate it in a state transition diagram [12].

Definition 2. An input-output sequence S of length L is an ordered set of pairs $\{(a_0, b_0), (a_1, b_1), \cdots, (a_{L-1}, b_{L-1})\}$ where $(a, b) \in \Sigma \times \Delta$. An input-output sequence set D is a set $\{S_1, S_2, \cdots, S_N\}$ of $N = |D|$ members with different lengths $L_1, L_2, \cdots, L_N$.

Notation: The final state of the Moore machine M after receiving input sequence $a_0 a_1 \cdots a_n$ is written succinctly as $\delta(q_0, a_0 a_1 \cdots a_n)$ and therefore $\delta(q_0, a_0 a_1) = \delta(\delta(q_0, a_0), a_1)$.

Definition 3. $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ is said to be consistent with input-output sequence

$$S = \{(a_0, b_0), (a_1, b_1), \cdots, (a_{L-1}, b_{L-1})\}$$

iff

$$\lambda(\delta(q_0, a_0 a_1 \cdots a_t)) = b_t \quad \text{for all } t \in \mathbb{Z} : 0 \leq t < L.$$

A Moore finite-state machine is said to be consistent with data set D iff it is consistent with all $S_j \in D$.

3.2 Problem Statement

Given the training data set D of input-output sequences randomly generated from target machine M of unknown size u and given an upper bound $2^m \leq u$, the problem is to synthesis a Moore finite-state machine M' of size u' consistent with D where $u \leq u' \leq 2^m$.

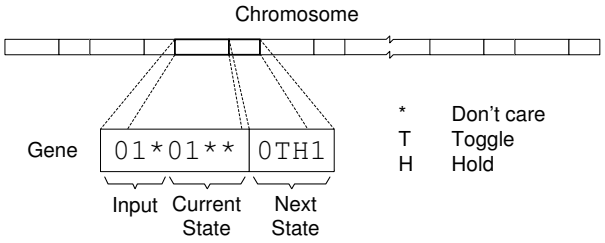

Figure 1: Chromosome structure for representing a finite state machine with 3 inputs bits and 4 state bits.

4. EVOLUTIONARY ALGORITHM

It is shown in [17] and [22] that instead of evolving both transition function $\delta(q, a)$ and output function $\lambda(q)$, only the transition function needs to be evolved and the output function can be inferred using a statistical method. The same statistical method of [22] for learning the output function (with slight modifications for a Moore model) is used here. A direct genotype-phenotype mapping was used in [17] and [22] meaning that the chromosome was constructed simply by concatenating all the next states in the right-hand column of the state transition table:

$$\delta(q_0, a_0)\delta(q_0, a_1)\cdots\delta(q_0, a_{v-1})\delta(q_1, a_0)\delta(q_1, a_1)\cdots$$

$$\delta(q_1, a_{v-1})\cdots\delta(q_{u-1}, a_0)\delta(q_{u-1}, a_1)\cdots\delta(q_{u-1}, a_{v-1}).$$

Here, an indirect genotype-phenotype mapping is used instead, with intention of creating neutral networks in the fitness landscape [26] and bringing more evolvability to the evolutionary algorithm. A fitness function based on the number of correct output symbols was used in [22]. However a new progressive fitness function based on the number of correct output symbols before emitting an incorrect symbol is used in this study.

4.1 FSM Representation

Figure 1 shows the single-chromosome structure of the genome, as an array of genes. A parameter (*chromosome size*) specifies number of genes in the chromosome. Each gene has two parts: a regulatory region, and a coding region [14]. The regulatory region has $n + m$ condition or TF (transcription factor [14]) sites. The n leftmost sites control how this gene is expressed in different input bit settings. The m rightmost sites control the gene expression in response to the state bits. Each site in the regulatory region can be '0', '1', or '*', each representing a condition. A gene can be expressed only when all conditions in the regulatory region are satisfied. A '0' at a specific site shows that this gene will not be expressed if the corresponding bit (in the input vector or current state of the FSM) is set and this condition is satisfied only when the corresponding bit is '0'. A '1' means that the gene can only be expressed if that bit in the input vector or current state is set. A '*' (representing a wild card) means that the gene does not care about this bit and its expression is not controlled by this bit (always satisfied).

The coding region of the gene comprises m sites and specifies the next state of the FSM. Each site in the coding region can be '0', '1', 'T', or 'H'. Having a '1' or '0' at a specific site means that if this gene is expressed, this bit in the next state will be set or reset respectively. A 'T' (standing for Toggle) means that the corresponding bit in the next state is the compliment of the same bit in the current state. An 'H' (representing Hold) means this bit should be copied from the current state to the next state. Only one gene from the whole genome can be expressed at a time. The first gene in the chromosome (from left) with the least number of wild cards that matches input vector and current state will be expressed. The coding region of the expressed gene determines the next state of the FSM.

An FSM with $u' \leq 2^m$ states and input alphabet of size $v' = 2^n$ can be described with this representation. For example, if we have one input ($n = 1$) and eight states ($m = 3$), the regulatory region will be four "bits" (sites) long ($n+m$) and the coding region of each gene will be three "bits" (m). Then, the following genome is one of many representations for a 3-bit binary counter with a count-enable input (counter increments when input is '1' and holds if it is '0').

0*** HHH	1*** HHT	1**1 HT0	1*** 1T0	1*11 T0T

The faded gene in the above example is always inactive due to the first gene being expressed instead. In this representation, there can be many inactive genes, which contribute to the neutral networks [26]. Neutral networks allow an evolving population to drift around equally-fit points in the search space until it reaches the neighborhood (within one mutation distance) of a fitter point. This helps the evolving population to escape from a local optimum. It is discussed in detail in section 8.

4.2 Genetic Operators

4.2.1 Crossover

Recombination of two chromosomes is performed simply by a standard 2-point uniform crossover. The crossover probability is always 1.0 during this research as other means for transferring fittest individuals to the next generation exist in the algorithm.

4.2.2 Mutation

Five types of variation (mutation) are used: normal mutation, gene randomization, gene duplication, gene swap, and cross mutation. Normal mutation can replace a site in a gene by a random symbol from {0,1,*} or {0,1,T,H} (depending on the region). If a gene randomization happens, all of the sites in a randomly selected gene is randomized. It cannot happen more than once for each chromosome in each generation. Gene duplication is when all the sites of a randomly selected gene is copied to another randomly selected gene. Gene swap, simply swaps the location of two randomly selected genes in the chromosome. Cross mutation is copying part of the regulatory region of a random gene on the coding region of another random gene. This is performed by first selecting two random loci in the chromosome, then extracting the state bits from the regulatory region of one of them, replacing wild cards with random alleles from {0,1,H,T} and overwriting the result on the coding region of the second gene. Mutations have no bias, which means equal probability for all alleles {*,0,1}, {0,1,H,T}.

4.3 Fitness Function

For fitness evaluation, each evolved FSM is initialized to q_0 (all zeros) before being simulated over each sequence S_j

in the training set D. In [22] fitness of each FSM is defined as the fraction of time during simulation that output symbols emitted by the evolving FSM are equal to the output sequences in the training set. This fitness function is called the reference fitness function throughout this paper. As will be explained later, this reference fitness function adds a needless raggedness to the fitness landscape. Here, a progressive fitness function, called "Average Lifespan", is used instead to prevent that raggedness.

4.3.1 Average Lifespan

In this method, first, the output function is inferred using the same algorithm explained in [22]. Then evolved FSM $M' = (Q', \Sigma', \Delta', \delta', \lambda', q'_0)$ is simulated on each sequence $S_j = \{(a_0^j, b_0^j), \cdots, (a_t^j, b_t^j), \cdots, (a_{L-1}^j, b_{L-1}^j)\}$ of length L_j in the training set $D = \{S_1, S_2, \cdots, S_N\}$, and its output, $\lambda'(q_t'^j)$, is compared with b_t^j at each time step t until they do not match ($q_t'^j$ is the state of the M' at the step t of the sequence S_j). The number of correct output symbols that M' emits before the first incorrect symbol on each sequence Sj is recorded as $l_j(M')$ and is called "Lifespan" of M' on S_j. Its scaled average over all sequences in the training set D is regarded as fitness of M', which can be written as:

$$F(M') = \frac{\sum_{j:S_j \in D} l_j(M')}{\sum_{j:S_j \in D} L_j} \qquad (1)$$

where

$$\lambda'(q_t'^j) = b_t^j, \qquad \text{for all } t \in \mathbf{Z}^* : 0 \leq t < l_j(M')$$

and

$$\lambda'(q_{l_j(M')}'^j) \neq b_{l_j(M')}^j, \qquad l_j(M') \leq L_j.$$

Emission of the first incorrect output symbol reveals an inconsistency and shows that M' is no longer in a correct state. As a result, the number of correct symbols emitted after this point, which is used in the reference fitness function evaluation, is irrelevant and should not contribute to the fitness value.

For example, take the FSM illustrated in figure 2a as M. Assume that the evolved FSM, M', has the same transition diagram except that in state 2, by receiving a '1' input, it goes to state 4 instead of going to state 3. During processing an input sequence, M enters state 3 by receiving a '1' and emits a '0' in the output. But M' enters state 4 and emits a '1'. From now on, some of the output symbols generated by M' may be correct (for instance if it immediately receives a '1' in the input) but they are only accidentally correct symbols, which can easily mask the effect of that incorrect transition on the fitness value and create a deceptive clue for the evolution.

4.4 Algorithm

A generic evolutionary algorithm was adopted from [3, 1]. It is based on two populations of children and adults. The adult population is maintained in order of fitness and parents are randomly (with equal probability) selected from amongst a number (*selection size*) of top adults. The child population is reproduced by these parents and evaluated with the fitness function at each generation. If a child's fitness is greater or equal to the fitness of one of the adults it is transferred to the right place in the adults population to maintain the order of the adults population and the least fit adult dies.

It is particularly important that algorithm inserts a child with a fitness equal to one of the adults at that adult's place, so that an individuals with a neutral mutation can take an old individual's place and algorithm can exploit neutral mutations [20]. Preliminary experiments showed that turning off this feature (*i.e.* using $>$ instead of $\geq$ in the search and insert algorithm) dramatically deteriorates the performance of the algorithm, which confirms the importance of the neutral networks in the fitness landscape.

These two populations can be implemented as linked lists for better performance [1]. As new individuals enter the adults population and less fit adults go out, the time that each adult is amongst the parents is proportional to its fitness. It is possible to change the selection pressure by changing relative size of two populations and the *selection size* parameter. In this algorithm, only new individuals in the children population are evaluated in each generation.

One of the effective changes that could be made in the algorithm is tuning the selection pressure. This pressure can be controlled by relative size of the child population to adult population size and number of parents (*select size*). Typical values for these parameters are *child population size*=80% and *select size*=40% of the *adult population size* [3]. In fact, putting more pressure can lead to early convergence to a suboptimal solution and low chances of escaping that, while a low pressure results in more diversity in the population but a very slow progress in evolution. Preliminary experiments showed that a slightly lower pressure gives a better performance for this problem.

5. A BIOLOGICAL PERSPECTIVE

It is possible to look at this representation from a biological standpoint [13, 14]. Each state of the FSM can be regarded as a binary abstraction of the protein concentration state of a very simple biological cell. Each state bit is a binary abstraction of the concentration of one specific protein synthesized in the cell. Input bits can be compared to inter-cellular communication proteins, hormones or environmental factors. Outputs can be regarded as physiological properties of the proteins produced by behavioral genes. Even the initial state can be compared to the maternal factors for a zygote cell [14].

The functional graph of the combinational digital circuit implementing a finite-state machine can also be regarded as a Gene Regulatory Network (GRN), which encodes the developmental and functional program of biological cells [13, 14]. For example, in a gene regulatory network, protein D is synthesized only in presence of proteins A and B, and absence of proteins C and D. It can be seen as four flip-flops labeled A, B, C, D with input of the D flip-flop connected to the logic function $F(A, B, C, D) = AB\bar{C}\bar{D}$.

We can think of the genome as the chromosome of a biological organism. An organism can live as long as it interacts correctly with its environment. Its chance for reproduction or replication is proportional to its lifespan. So we can think of its lifespan as a fitness affecting its reproduction. However, in [22], the fitness function is the average correctness of the organism interaction over the length of the input sequence. This difference led to the new fitness function detailed in section 4.3.1. The hold and toggle alleles ('H' and 'T') can be also regarded as abstraction of coding for those

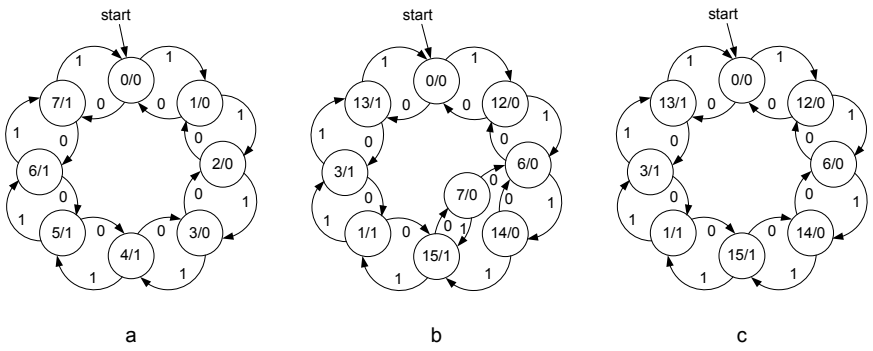

Figure 2: (a)State transition diagram of a modulo-8 up-down counter as an example of the counters used in the experiments. (b)A perfect evolved FSM targeting the FSM a. (c)A minimum size perfect solution generated by evolution beyond perfection only one generation after the FSM b. The q/b in each state means that the FSM emits output symbol b in state q. Input symbols are shown on state transitions.

proteins that interact with other proteins to decelerate their defusion rate or negate each other's effect (respectively for 'H' and 'T').

There are also some inconsistencies between this representation and what happens in nature. Each biological gene can only produce one protein (in simple organisms) and many genes can be expressed at the same time and many expressed genes can contribute to concentration of one protein. In contrast, in this representation, each gene can even produce all the proteins, only one gene is expressed at a time, and when more than one gene can be activated only the first one with the least number of wild cards is expressed.

6. EXPERIMENTS

Experiments were designed to compare the new techniques with the reference methods in terms of their evolvability and scalability. This was done by changing the complexity of the problem in a controlled fashion while keeping all other parameters fixed, repeating experiments with the new and the reference methods. Another variable, which affected the success rate of the evolutionary algorithm through preliminary experiments, was maximum size of the evolving FSMs ($u' \leq 2^m$). It can be only changed in steps of powers of two (2^m) due to the nature of the binary representation. Therefore, FSMs of maximum sizes 16, 32 and 64 were evolved.

6.1 Training Set

In many studies, randomly generated target FSMs were used as target machines for creating the training sets. However, it is difficult to generate a random FSM with a controlled level of complexity as its complexity highly depends on many other factors besides number of states. Therefore, this experiment was performed on Modulo-N up-down counters anticipating that their complexity is an increasing function of their sizes. Each modulo-N up-down counter was generated with all the N states placed in one single loop in wrapped-around binary order. A '1' in the input of the modulo-N counters made their binary state increment and a '0' decremented their binary state. States 0 to $\frac{N}{2}-1$ emit a '0' and states $\frac{N}{2}$ to N emit a '1' in the output. The state transition diagram of a modulo-8 up-down counter of this kind is shown in figure 2a as an example. Each modulo-N up-down counter was used to generate a data set consisting of 20 input-outputs sequences of length $4N$. Each input sequence was generated by a random bit stream of probability 0.7 and each output sequence was generate by feeding the input sequence to the target counter.

6.2 Experiment Setup

Experiments were performed on training sets generated by module-N up-down counters of size $N = \{2, 4, 6, 8, 10, 12, 14, 16\}$. FSMs were evolved for 25 runs. The number of runs that generated a perfect solution was recorded for three different settings of the number of state bits ($m = \{4, 5, 6\}$) using the following combinations of mappings and fitness functions:

1. Direct mapping with the reference fitness function (same as [22])
2. Indirect mapping with the reference fitness function
3. Direct mapping with the Average Lifespan fitness function
4. Indirect mapping with the Average Lifespan fitness function.

In the direct mapping representation method, the state transition table was directly coded into the chromosome, while in the indirect mapping technique the state transition table was developed by expression of genes using the new algorithm. The Smart State Labeling [17] were used for output function estimation with both the Average Lifespan and the reference fitness functions.

Parameters were tuned, fixed, and identical for all techniques throughout the experiments as reported in table 1. However, normal mutation is based on probability of mutation per site. Therefore, *normal mutation probability* was tuned each time, according to the *chromosome size*. Moreover, the *chromosome size* was set to 2^{m+n+1} where n is number of input bits and equals to one and m is the number of state bits, which is a variable during experiments.

7. RESULTS

Results for evolving 4-bit FSMs of maximum size 16 ($u' \leq 16$) are shown in figure 3. Using the Average Lifespan fitness function showed a significant improvement. However, both mappings had almost the same performance, with

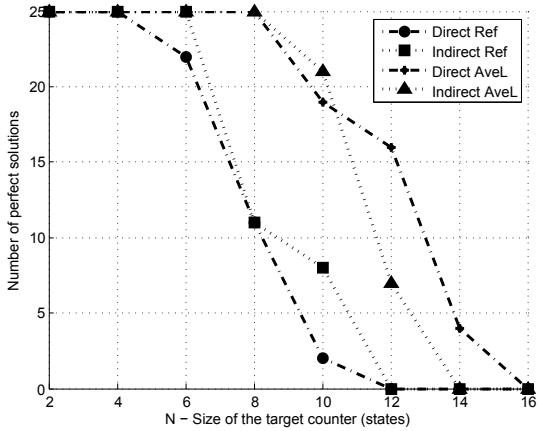

Figure 3: Number of perfect modulo-N up-down counters evolved using $u' \leq 16$ against the size of the target counter (N).

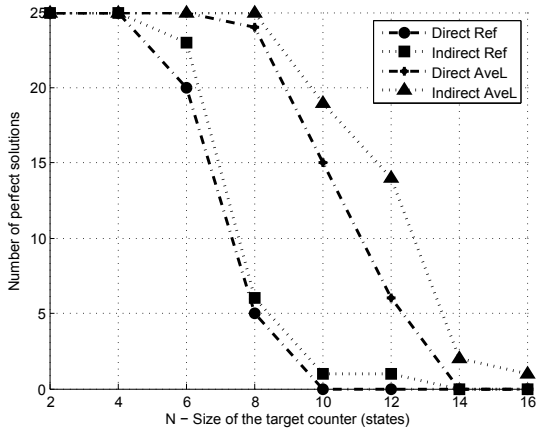

Figure 4: Number of perfect modulo-N up-down counters evolved using $u' \leq 32$ against the size of the target counter (N).

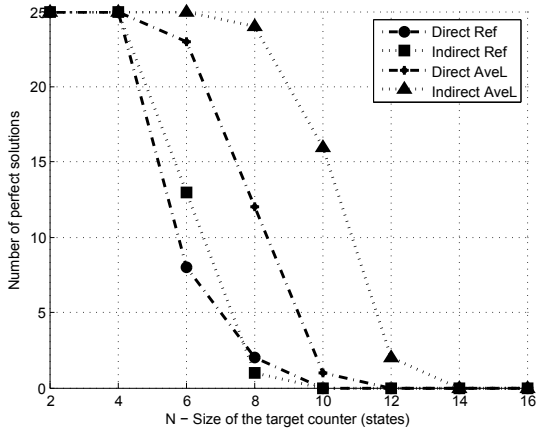

Figure 5: Number of perfect modulo-N up-down counters evolved using $u' \leq 64$ against the size of the target counter (N).

Parameter	Value
Normal mutation probability	0.003, 0.0015, 0.0008 per site for $m = 4, 5, 6$
Gene randomisation probability	0.001 per chromosome
Gene duplication probability	0.2 per chromosome
Gene swap probability	0.2 per chromosome
Cross mutation probability	0.7 per chromosome
Crossover probability	1.0
Adult population size	30
Children population size	24
Selection size	16
Chromosome size	$2^{n+m+1} = 64, 128, 256$ for $m = 4, 5, 6$
Maximum number of generations	20000 (480000 evaluations)
Number of runs	25

Table 1: Parameter settings used in the experiments

slightly higher number of perfect solutions for combination of direct mapping and Average Lifepan fitness function. For the $u' \leq 32$ case in figure 4, direct and indirect mapping still had almost equal performances using the reference fitness function. Again, using Average Lifespan fitness function, improved the successes rate. However, indirect mapping outperformed the direct mapping representation in this case. This difference was more significant in $u' \leq 64$ case, shown in figure 5. Both mapping techniques showed almost the same performance with the reference fitness function and their performances were both improved using Average Lifespan fitness function. The number of reachable states of the evolved FSMs were usually very close to the number of states in the target FSM and they generally converged to the correct number when evolution continued for a while after finding a perfect solution. Figure 2b shows a perfect evolved FSM with 9 states, which is consistent with a data set generated from the FSM shown in figure 2a. In the same run, continuing the evolution for one generation created the 8-state FSM of figure 2c.

8. ANALYSIS

The above results show a significant improvement in all cases when Average Lifespan fitness function is used. Using the Average Lifespan fitness function seems to help the evolution as explained in section 4.3.1. It may contribute to the neutrality of the fitness landscape by neutralizing those adaptive mutations that affect deeper transitions. By deeper transitions, we refer to those state transitions that are not experienced by Average Lifespan fitness function during evaluation of an FSM due to early incorrect output symbols.

Another noticeable occurrence was that the indirect mapping representation started to outperform the direct mapping technique as the maximum size of the evolving FSMs (2^m) was increased. This is the typical behavior in developmental evolutionary approaches to scale well when evolving a modular structure (like modulo-N counters in this problem) [11]. This might be because this coding technique creates a one-to-many mapping by having different redundancies in the representation and many inactive genes, which in turn contributes to neutrality in the fitness landscape [26]. While adding {*,H,T} to a default set of alleles ({0,1}) adds to the redundancy needed for neutral networks, it is also responsible for the emergent modularity and formation of motifs in the transition diagram of the evolved FSMs, which are

other typical features of a developmental evolution. Interestingly, this scalability shows itself only when the indirect mapping and the Average Lifespan fitness function are used together. This suggests a connection between them on this particular problem, which needs more investigation.

The other perceptible observation was that evolution tends to reduce the number of states in the evolving FSMs when evolution continues after finding the first perfect solution. Emergence of this property and graceful degradation in evolved solutions by "evolving beyond perfection" is investigated in more detail in [4]. As evolution tries to make the chromosome robust to adaptive mutations, it tends to use redundancy and/or code compression to create an efficient and robust genotype [4]. In this particular representation, this tendency leads to fewer number of active genes with higher number of wild cards ('*'), which consequently shows itself as more regular and usually smaller evolved FSMs.

9. DISCUSSION

Great similarities between different complex natural systems such as evolution, gene regulatory networks, development, brains, immune systems, swarms, and complex social systems may imply existence of a common set of fundamental laws governing all these systems in nature [2]. Nature-inspired computing is trying to extract these essential rules to create abstracted models for solving difficult engineering problems. However, as our understanding of biology increases, computer scientists can progress in their quest for these underlying laws by imitating nature in their models more accurately. On the other hand, biologists, neuroscientists, sociologists, economists and other scientists studying natural complex systems may also find an opportunity to use these models to gain insight into their own subjects [18]. This mutually beneficial multidisciplinary approach is not possible without a proper mapping between two interacting domains. This mapping shows itself as analogies or metaphors in the literature.

These analogies and metaphors help scientists to compare the artificial evolution with the natural evolution and locate their similarities and differences. Some of these differences point us to the essential properties of a successful evolutionary system. Most of the artificial evolutionary systems oversimplify at least two major processes in natural evolution by means of abstraction: selection and genotype-phenotype mapping.

Selection and the survival of the fittest are known to be the central drives for evolution. However there is a large difference between the selection process in nature and selection in computational evolution. Typically, individuals (solutions) are evaluated using a predefined static fitness function that determines how well this individual can solve the problem. Then, some of the fittest individuals are randomly selected and mated as parents. In nature, there is no such a thing as fitness function and there is no clear measure for fitness. Organisms develop, mature, move around and reproduce, interacting with each other and their environment, with lots of competition and symbiosis within their population and with other species. Each organism goes through a unique trajectory of its own during its lifetime and there is no accurate way of comparing two different individuals. The environment and the evolving populations are changing all the time and consequently the meaning of the fitness is also evolving through time. It looks as if evolution is free to take different approaches to the problem of survival. Although the environment dictates the basic rules of the game, evolution can tackle the problem from another angle, transforming the problem to a completely different problem. This can be seen in nature that how species start to evolve in a habitable environment and gradually adapt and migrate to more hostile environments.

The way nature develops an organism from its genome and maintains it during its lifespan is another vital process of evolution. In traditional approaches to evolutionary computing, a solution (phenotype) is usually generated directly from the genotype. However, in natural evolution it is performed through a very complex process of development governed by intricate gene regulatory networks which in turn are based on complex processes of gene expression, protein production, folding, interaction, and diffusion [14]. Each one of these processes is orders of magnitude more complex than all the evolutionary systems we have implemented in our computers. It is believed that these processes might be partly responsible for the high evolvability of the natural evolution [14]. Creating detailed abstractions of these processes may improve the evolvability of the computational evolution [3, 11] while giving developmental biologists a chance to look at development through the aperture of these abstract models to unravel some of its mysteries.

Extending these analogies can lead to a bio-plausible (or ALife) approach to multi-objective optimization. Environment can be a metaphor for the objective space, spanning from suboptimal areas to optimal areas. Species living in this environment are the solutions to the problem, which tend to gradually migrate/grow toward hostile (optimal) zones due to shortage of resources (energy, food, light, etc.) in the habitable (suboptimal) areas. The phenotype of each individual represents a specific setting of variables, developing (moving/growing in the objective space) according to its developmental program while interacting with the environment (objective space) and other individuals (solutions) by sensing the environment (objective functions), reproduction, consuming resources, etc. Constraints can also be enforced by death, injuries or other means that affect the individual's development and lifespan. Such a method seems very promising in creating speciation as individuals with similar trajectories in the environment have a higher chance of mating.

This paper shows how analogies helped us to incorporate only an element of these details into our computational model and how even such subtle changes toward bio-plausibility may cause improvements. First, it showed how using a more bio-plausible indirect mapping and a fitness function, wildly used by ALife community, affected the scalability of the evolutionary algorithm and improved its performance in a particular case. All these results substantiate the claim that incorporating abstracted biological details into evolutionary algorithms can be helpful to computational evolution.

10. CONCLUSIONS

A new representation, based on a simplified model of gene regulatory networks, and a novel fitness function called "Average Lifespan" were introduced for evolving Moore finite-state machines and were used for evolving FSMs consistent with data sets from modulo-N up/down counters of size 2 to 16 states. A Moore model was used as it is widely employed

in the design of synchronous sequential digital circuits. The average Lifespan fitness function improved the number of perfect solutions even when used with a standard representation. Results confirmed that, in this problem, using both new methods together improved the success rate of the evolutionary algorithm when higher maximum number of states were used ($u' \leq 32$ and $u' \leq 64$). This is a positive result as it is desired to use higher upper bounds for the number of states when size of the target machine is unknown. This is in contrast with the standard direct mapping representation, which showed a dramatic degradation when this upper bound was increased.

Both the indirect representation and the new fitness function are biologically more plausible than the reference methods and, together they outperformed those methods in the experiments reported here. This may be a consequence of neutrality in the fitness landscape. This performance comparison can be carried out in detail for a broader range of target FSMs in future studies. It is also of great interest to see how this new representation and fitness function perform in presence of noise in the training set. Other avenues are deeper investigation of the effects of using a reduced set of alleles and "evolution beyond perfection" [4] on the evolving FSMs.

11. REFERENCES

[1] P. Bentley. From coffee tables to hospitals: Generic evolutionary design. In P. J. Bentley, editor, *Evolutionary Design by Computers*, pages 405–423. Morgan Kaufmann, San Francisco, CA, 1999.

[2] P. Bentley. *Digital biology*. Simon & Schuster New York, 2002.

[3] P. Bentley. Fractal proteins. *Genetic Programming and Evolvable Machines*, 5(1):71–101, 2004.

[4] P. J. Bentley. Evolving beyond perfection: an investigation of the effects of long-term evolution on fractal gene regulatory networks. *Biosystems*, 76(1-3):291–301, 2004.

[5] J. Bongard and H. Lipson. Active coevolutionary learning of deterministic finite automata. *Journal of Machine Learning Research*, 6:1651–1678, 2005.

[6] P. Chongstitvatana and C. Aporntewan. Improving correctness of finite-state machine synthesis from multiple partial input/output sequences. In *Evolvable Hardware*, page 262. IEEE Computer Society, 1999.

[7] C. de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38(9):1332–1348, 2005.

[8] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, New York, 1966.

[9] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[10] J. Gomez. An Incremental Learning Algorithm for Deterministic Finite Automata Using Evolutionary Algorithms (Gecco 2004 Noisy DFA Entry). *Proc. Genetic and Evolutionary Computation Conf.*, 2004.

[11] T. G. W. Gordon and P. J. Bentley. Development brings scalability to hardware evolution. In *Proceedings of the 2005 NASA/DoD Conference on Evolvable Hardware*, pages 272–279. IEEE Press, 2005.

[12] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[13] S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *J. Theor. Biol.*, 22:437–467, 1969.

[14] S. Kumar and P. Bentley. *On growth, form and computers*. Elsevier Academic Press, 2003.

[15] K. Lang, B. Pearlmutter, and F. Coste. The gowachin server. URL http://www. irisa. fr/Gowachin/, 2005.

[16] K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the Abbadingo One DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference; 4th International Colloquium, ICGI-98*, volume 1433 of *LNCS/LNAI*, pages 1–12. Springer, 1998.

[17] S. Lucas and T. Reynolds. Learning deterministic finite automata with a smart state labeling evolutionary algorithm. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(7):1063–1074, 2005.

[18] C. M. Macal and M. J. North. Tutorial on agent-based modeling and simulation. In *Winter Simulation Conference*, pages 2–15. ACM, 2005.

[19] C. Manovit, C. Aporntewan, and P. Chongstitvatana. Synthesis of synchronous sequential logic circuits from partial input/output sequences. *Lecture Notes in Computer Science*, 1478:98s, 1998.

[20] J. F. Miller and W. Banzhaf. Evolving the program for a cell: from french flags to boolean circuits. In S. Kumar and P. J. Bentley, editors, *On Growth, Form and Computers*. Elsevier Academic Press, Oct. 2003.

[21] E. F. Moore. Gedanken-experiments on sequential machines. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, number 34 in Annals of Mathematics Studies, pages 129–153. Princeton University Press, Princeton, NJ, 1956.

[22] N. Niparnan and P. Chongstitvatana. An improved genetic algorithm for the inference of finite state machine. In *Systems, Man and Cybernetics, 2002 IEEE International Conference on*, volume 7, pages 5–, 2002.

[23] A. Oliveira and J. Silva. Efficient Algorithms for the Inference of Minimum Size DFAs. *Machine Learning*, 44(1):93–119, 2001.

[24] Pitt and Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. *JACM: Journal of the ACM*, 40, 1993.

[25] H. Shayani. Towards Evolution of a Digital Spiking Neuron Model for FPGA Implementation. Master's thesis, 2006.

[26] T. Smith, P. Husbands, P. J. Layzell, and M. O'Shea. Fitness landscapes and evolvability. *Evolutionary Computation*, 10(1):1–34, 2002.

[27] J. Wakerly. *Digital Design: Principles and Practices*. Prentice Hall, 2000.