Using Multi-objective Genetic Programming to Evolve Stochastic Logic Gate Circuits

Brian J. Ross
Department of Computer Science
Brock University
St. Catharines, ON, Canada L2S 3A1
Email: bross@brocku.com

***Abstract*—A new stochastic logic gate language is presented. Blossey *et al.*'s stochastic gene gate language is extended with a complete set of stochastic Boolean gates. Although the gates have behavioural similarities to conventional logic gates, a major difference is that they operate on quantities of products or substances that dynamically vary over time. A gene gate circuit's behaviour is characterized by a time-course plot of the substance quantities. The paper studies the Boolean gate language by using multi-objective genetic programming to evolve logic gate circuits that conform to a number of different target systems. Circuit behaviour is characterized by sets of up to 15 time course statistics, and sum of ranks is used as a many-objective scoring strategy. Results show that the language is highly compositional, just like conventional logic expressions, and that multiple circuits can exhibit similar behaviours. The new gate language uses Blossey *et al.*'s gates as a rudimentary basis within evolved circuits, with the advantage of using higher-level Boolean gates when necessary. The identification of candidate solutions can be challenging, however, and must account for noise inherent in the time course behaviours. Circuit behaviour is also highly dependent on channel rates, and future work applying the language to real-world data will need to address this sensitivity.**

I. Introduction

Boolean logic is widely used for modelling synthetic gene regulatory networks (GRN) [1], [2], [3], [4]. Part of the reason for its popularity is because it is well understood, being the foundation for computer software and hardware. It also has adequate descriptive power to model many regulatory phenomena of interest. Although Boolean logic permits modelling of causal relationships in synthetic circuits, the stochastic nature of biological behaviour is also relevant [5], [6], [7].

Automating the construction of GRNs using computational intelligence and other techniques is an active area of research. Approaches include the use of directed evolution [8], genetic algorithms [9], [8], neural networks [10], and Bayesian learning [11]. Of particular interest in this paper is the use of genetic programming (GP) for regulatory network synthesis. Banzhaf uses a linear representation with GP to generate GRNs, which model protein concentration changes over time [12]. Qian *et al.* use GP and Kalman Filtering to create differential equation models of GRNs [13]. Cai *et al.* use GP and particle swarm optimization to find GRNs [14].

This paper presents a new implementation of a stochastic Boolean gate language. There are two main contributions of this research. First, the new gate language extends and complements Blossey *et al.*'s transcriptional gene gate language [15]. We define conventional Boolean gates using the same simulation model as [15], written in the stochastic pi-calculus (SPI) [16] that they use. The language inherits from Blossey's *et al.*'s model the ability to concisely model noisy, and often chaotic, behaviours seen in stochastic systems.

A second contribution is to investigate the evolvability of stochastic gate circuits using multi-objective GP. This extends earlier work by Imada [17], [18] that used GP to evolve circuits using Blossey *et al.*'s stochastic gate language [15]. Imada characterized the noisy time-course behaviour of the target stochastic behaviours using feature-based statistics, and used these statistics within the objective function. A few of the same example systems were re-examined in [19] using many-objective ranking strategies to account for the high dimensionality of the problems. Similarly, we characterize SPI gate circuits by using statistical analyses of their time-course behaviour. We also use many-objective scoring within the GP framework as a fitness evaluation strategy. However, the added complexity of the gene gate language presents challenges when assessing final results.

The paper organization is as follows. Section II briefly reviews the stochastic pi-calculus. Section III presents the new Boolean gate language. Methods used for evaluating stochastic gate circuits are reviewed in Section IV. Experiments are described in Section V, and results are presented in Section VI. Conclusions and future research directions conclude the paper in Section VII.

II. The Stochastic π-Calculus

TABLE I. SPI syntax (excerpt)

$Proc\_Defn$ ::=	`Let Name`$(Var, \cdots, Var)$ $= Proc$	:Process definition
$Proc$ ::=	()	:null (often removed)
	$(Proc \mid \cdots \mid Proc)$	:parallel composition
	`do` $ActSeq$ `or` $\cdots$ `or` $ActSeq$	:stochastic choice
	`Name`$(Val, \cdots, Val)$	:process call
$ActSeq$ ::=	$Act \mid Act; ActSeq$	:action sequence
Act ::=	`delay@float`	:delay
	$!Channel(Val, \cdots, Val)$	:output action
	$?Channel(Val, \cdots, Val)$	:input action
Val ::=	$Channel$ | float	
Var ::=	variable label	
$Channel$ ::=	channel label	

The stochastic π-calculus (SPI) [16] is a stochastic variant of the π-calculus process algebra [20]. SPI is a rudimentary calculus that defines a core set of operators with which to build complex behaviours. The main contribution of SPI is its stochastic modelling of concurrent computations, which is based upon Gillespie's stochastic simulation of chemical reactions [21]. This enables SPI to effectively model various

978-1-4799-6926-5/15/$31.00

natural phenomena, for example, chemical and biochemical reactions, and gene regulation and transcription.

Table I shows the subset of SPI used in this work. SPI operators permit parallel execution of processes, stochastic choices of behaviours, sequences of behaviours, and process invocation. Processes communicate on channels. An inter-process handshake arises when complementary input and output actions occur on the same channel. The stochastic aspect of communication arises with the stochastic choice operation. When multiple choices of behaviour are available, each will have a probability derived for it. The probabilities arise from channel *rates*. Channels with higher rates are more active. *Delay* terms affect simulation execution, but produce no action.

During a SPI simulation, SPI terms may be duplicated due to recursion and other operations, and this will affect the rates and probabilities of active channels. If one considers channels as denoting proteins, enzymes, gene activation levels, etc., then these channel quantities can also be considered to denote *quantities* of reactive substances and products arising in the system being modelled. These dynamically changing values can be plotted over time, resulting in time-course plots of the substances within the simulation.

III. A STOCHASTIC GENE GATE LANGUAGE

A. Background: a gate language for inhibition and stimulation

TABLE II. BLOSSEY *et al.* GENE GATE LANGUAGE FROM ([15]).

```
new a@1.0:chan, b@1.0:chan, c@1.0:chan

let Tr(b:chan) =
  do !b; Tr(b)
  or delay@0.001

and Neg(a:chan,(cst:float, inh:float), b:chan) =
  do ?a; delay@inh; Neg(a,(cst,inh),b)
  or delay@cst; (Tr(b) | Neg(a,(cst,inh),b))

and Pos(a:chan, (cst:float, inh:float), b:chan) =
  do ?a; delay@inh; (Tr(b) | Pos(a,(cst, inh), b))
  or delay@cst; (Tr(b) | Pos(a,(cst, inh), b))
```

SPI is fairly rudimentary in its expressiveness. To address this, Blossey *et al.* use SPI to define a higher-level gene gate language for stochastic modelling and simulation [15]. The language definitions are shown in Table II. Channels are defined as type `chan`. They denote pathways through which communications occur, and have associated channel rates.

`Tr(b)` denotes the transcription factor. It can choose to express an output signal on channel `b`, which is denoted `!b`. This results in an increase in the quantitative availability of `!b`. Alternatively, the expression might delay using a small rate value of 0.001, in which case it forgoes expressing `b`. The `Neg` gate performs signal inhibition. The `cst` and `inh` labels are rate parameters for delays. `Neg(a,...,b)` will inhibit the production of `b` when signals of `a` are more frequent, and increase production of `b` otherwise. The `Pos` gate is similar.

There is a probabilistic race between the choices of action in the above gates, which depends upon signal rates and accumulated probabilities arising during the simulation. The dynamically changing quantities of signals results in varied, often noisy behaviours, and this makes the language useful for modelling biological phenomena [15].

B. A new Boolean gate language.

A new stochastic logic gate language is given in Table III. The SPI definitions are influenced by Blossey *et al.*'s gate language [15] (Table II). The new gates incorporate the *Tr* gate, and circuits can optionally include *Pos* and *Neg* gates. The logic gates also result in signal inhibition and excitation, but in a way that reflects conventional Boolean gates. We will examine 3 definitions.

TABLE III. SPI DEFINITION OF BOOLEAN GATE LANGUAGE

```
new a@1.0:chan, b@1.0:chan, c@1.0:chan

Let BOr(a:chan, b:chan,  (cst:float, inh:float), c:chan) =
  do ?a; delay@cst; (Tr(c) | BOr(a,b,(cst,inh),c)) or
     ?b; delay@cst; (Tr(c) | BOr(a,b,(cst,inh),c))

and BAnd(a:chan, b:chan, (cst:float, inh:float), c:chan) =
  ?a; ?b; delay@cst; (Tr(c) | BAnd(a,b,(cst,inh),c))

and Nor(a:chan, b:chan, (cst:float, inh:float), c:chan) =
  do ?a; delay@inh; Nor(a,b,(cst,inh),c) or
     ?b; delay@inh; Nor(a,b,(cst,inh),c) or
     delay@cst; (Tr(c) | Nor(a,b,(cst,inh),c))

and Nand(a:chan, b:chan, (cst:float, inh:float), c:chan) =
  do ?a; (do ?b; delay@inh; Nand(a,b,(cst,inh),c) or
             delay@cst; (Tr(c)|Nand(a,b,(cst,inh),c))) or
     ?b; (do ?a; delay@inh; Nand(a,b,(cst,inh),c) or
             delay@cst; (Tr(c)|Nand(a,b,(cst,inh),c))) or
     delay@cst; (Tr(c)|Nand(a,b,(cst,inh),c))

and Equiv(a:chan, b:chan, (cst:float, inh:float), c:chan) =
  do ?a; (do ?b; delay@cst; (Tr(c)|Equiv(a,b,(cst,inh),c))
             or delay@inh; Equiv(a,b,(cst,inh),c)) or
     ?b; (do ?a; delay@cst; (Tr(c)|Equiv(a,b,(cst,inh),c))
             or delay@inh; Equiv(a,b,(cst,inh),c)) or
     delay@cst; (Tr(c)|Equiv(a,b,(cst,inh),c))

and Xor(a:chan, b:chan, (cst:float, inh:float), c:chan) =
  do ?a; (do ?b; delay@inh; Xor(a,b,(cst,inh),c) or
             delay@cst; (Tr(c)|Xor(a,b,(cst,inh),c))) or
     ?b; (do ?a; delay@inh; Xor(a,b,(cst,inh),c) or
             delay@cst; (Tr(c)|Xor(a,b,(cst,inh),c)))

and Impl(a:chan, b:chan, (cst:float, inh:float), c:chan) =
  do ?a; (do ?b; delay@cst; (Tr(c) | Impl(a,b,(cst,inh),c))
             or delay@inh; Impl(a,b,(cst,inh),c)) or
     delay@cst; (Tr(c)|Impl(a,b,(cst,inh),c))
```

In the *BOr* gate, the input signals *a* and *b* both result in the transcription of product *c*. With *BAnd*, the reception of both *a* and *b* are required before *c* is transcribed. Since all gates use the *Tr* gate to transcribe the output signal, the transcription of *c* will be subject to degradation.

A more complex definition is *Xor*. If input *a* is seen, and *b* occurs soon after, then no transcription arises; otherwise, after a delay, *c* is transcribed. The second case mirrors this first case, but with *a* and *b* reversed. Implicit in this definition is the importance of signal timing. In the first case, *b* might eventually arrive, but if it does after the delay term has activated, then it will be ineffective. This means that channel rates, and accumulated frequencies of signals, are critical in determining the activation behaviour of gates.

Figure 1 shows some time plots for the And, Or, and Xor gates. A (blue) and B (red) are the input signals, and H (green) is the output. The behaviour of H corresponds to the truth table definitions of the gates. In the And plot, the output H activates whenever both A and B are non-zero. Note that signals do not rise and fall immediately, but require time to strengthen and degrade. In the Or plot, H is active over active portions of either A, B, or both. With Xor, H is active only when one of A and B are active, but not both simultaneously.

Figure 2(a) shows a gate circuit for a D-type transparent latch, which is a 1-bit memory flip-flop [22]. The equivalent

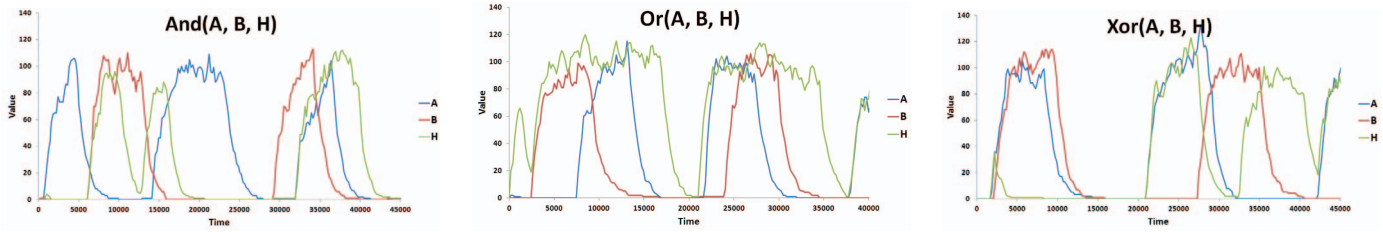
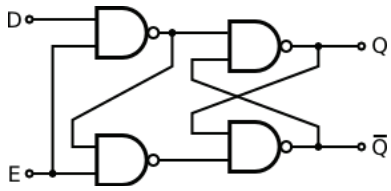
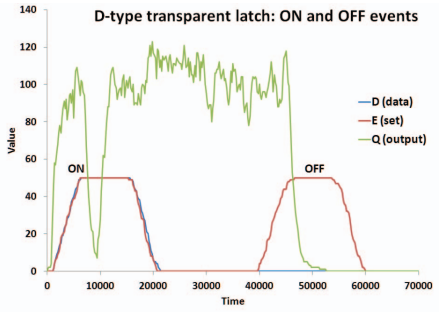


Fig. 1. Example time-course plots for And, Or, and Xor gates.



Nand(D,E,(u,t),C) | Nand(E,C,(u,t),B) |
Nand(C,$\overline{Q}$,(u,t),Q)| Nand(B,Q,(u,t),$\overline{Q}$)

(a) Boolean circuit and SPI gate expression.



(b) Time course plot.

Fig. 2. Flip-flop: D-type transparent latch. In (b), plateau labelled ON has both D and E set high, while OFF has E high only.

SPI gate expression is given. The signal lines are D (input data bit), E (input save signal), and Q (output saved memory). To use the latch, D is set to a bit value, and when stable, E is turned on to saved the high or low input on Q. Figure 2(b) shows a time course plot. The plateau labelled ON is formed by the simultaneous activation of both D and E. During this time, the Q signal shows some instability, but then settles on a stable state. After E is turned off, the Q output retains its high value. The second plateau now sets E high, while leaving the data D to low. Q then resets itself to a low value accordingly. Fritz *et al.* discuss applications of flip-flops in GRNs [1].

The above examples are interesting from a few perspectives. Firstly, their functional behaviours are analogous to their Boolean equivalents. Secondly, the time course plots are similar to what might be viewed on an oscilloscope, where the vertical "value" axis denotes voltage. Hence the SPI circuits are simulating the state transitions of electrical energy. This equivalence between biological and electrical modelling is not new. Koza *et al.* use GP to evolve bio-chemical reaction networks [23], that are modelled as electrical circuits.

TABLE IV. TIME COURSE STATISTICS

Label	Name and description
a	mean: average value
b	standard deviation: signal variance from mean
c	skew: asymmetry around the mean
d	kurtosis: flat or peaked shape relative to normal dist.
e	serial correlation: degree of fit to a white noise model
f	chaos: Lyapunov exponent; sensitivity dependence on initial values
g	periodicity: measure of cyclic patterns

IV. FITNESS EVALUATION

A. Statistical features of time course data

The modelling of time series has been actively studied [24], and it has been a topic in evolutionary computation [25], [26]. Because SPI gate expressions are non-deterministic, they do not permit direct comparison of output time plots, say between expressions and targets, which is possible in deterministic simulations [23].

We use the time series features from [17], [27]. Imada presents a suite of 17 different statistical measurements of time series, which are based on ones described in [28], [29]. When a SPI expression is simulated, the SPI interpreter generates a time-course plot of data for designated channels of interest. We take these data plots, and compute desired time-course statistics for them. The subset of statistics that we use are in Table IV; see [17] for details.

To use the time course statistics, the target system is simulated 1000 times. Time-course statistics are computed for each simulation's time plot. The mean values of these target features are then used as target values for the GP fitness function. To evaluate fitness, the evolved SPI expression is simulated, and its time-course statistics are computed. The absolute difference between the expression statistic and target statistic is computed for each feature. This vector of errors is used by the MOP strategy in the next section.

The selection of features to use for different channels is done in an *ad hoc* manner. As done in [17], the stability of particular statistics, reflected by calculating μ/σ from the 1000 interpretations of the target system, is often considered when making selections.

B. Many-objective ranking

Multi-objective optimization problems (MOP) are characterized by multiple discrete objectives that can be related to each other in complex, non-linear ways [30]. Our experiments use between 4 to 15 objectives, selected from the set in Table IV and applied to multiple channels. Problems of this size are high-dimensional (*many objective*) MOPs, and are not conducive to Pareto ranking. Therefore, we use the sum of ranks (or average rank) strategy for evaluating the many objectives into a fitness value [31], [32].

The normalized sum of ranks is computed as follows. Consider a vector of k objectives $\vec{v} =< v_1, ..., v_k > (k \geq 2)$ to be minimized. Each population member has an objective vector $\vec{v_i}(i = 1, ..., n)$. Each objective $j = 1, ..., k$ is first ranked amongst the population members, in increasing order. The least value found for objective j is assigned the rank value 1, and successive values are ranked up to a maximum max_j. After doing this for all objectives, each population member i will have a rank vector $\vec{r_i} =< r_1, ..., r_k >$. Next, for each objective $j = 1, ..., k$, a normalized rank is determined:

$$\begin{aligned} \vec{s}_n &= < \tfrac{r_1}{max_1}, ..., \tfrac{r_k}{max_k} > \\ &= < s_1, ..., s_k > \end{aligned}$$

Fitness is then computed for each individual:

$$fitness_i = \sum_{j=1}^{k} w_j s_j$$

where w_j is an optional weight value (default is 1.0).

V. Experiments

A. Target systems

The six target systems used are as follows (rate terms are omitted for brevity). (1, 2) Basic OR and NOR are:

```
(1)  And(a,e,h) | Or(b,f,h)
(2)  And(a,e,h) | Nor(b,f,h)
```

The channels a, b, e, and f are read from a repressilator circuit (described below), and are non-deterministic input values for the above expressions. The h channel is tested. Note that when two gates simultaneously transcribe h, then h's quantitative amount may double. This contrasts to the true/false output of a conventional Boolean expression. (3) The repressilator is taken from [15]. It is a well-known synthetic network which generates cyclic, circadian clock behaviour. The target expression is:

```
Neg(w,x) | Neg(x,y) | Neg(y,w)
```

All 3 channels are tested. (4, 5) The D016 and D038 circuits are also taken from [15], and are based on synthetic genetic circuits from [33]. D016 is a synthetic network that behaves similar to a Nor gate for one of the products, while D038 behaves like a logical "not if" during particular state configurations. They use additional transcription gates beyond those given in Table II, as well as a more complex parameterization of rates. The target expressions are *not* encodable in our gate language, even when supplemented with the gates in Table II. Therefore, it will be necessary to evolve behaviourally equivalent Boolean expressions. (6) The i633 target is a solution obtained for one of our early D016 runs:

```
Nand(gfp,laci,(0.1,0.001),gfp) |
  4@Neg(lci,(0.1,0.001),gfp) |
  Neg(tetr,(0.1,0.001),tetr) |
  2@Pos(gfp,(0.1,0.001),gfp) |
  Pos(gfp,(0.1,0.001),laci) |
  4@Pos(gfp,(0.1,0.001),lci) |
  Pos(lci,(0.1,0.001),gfp) |
  2@Pos(lci,(0.1,0.001),lci) |
  Pos(lci,(0.1,0.001),tetr)
```

TABLE V. Experiment summary. The Basic OR and NOR experiments use the logic gate language, while the rest use logic gates and Pos/Neg gates.

Name	Rates Fix	Rates Var	T	K	Channels tested	Channels extra	Features #: type
Basic OR	✓	-	400k	1000	1	4	4: abcd
Basic NOR	✓	-	400k	1000	1	4	4: abcd
Repressilator	-	✓	600k	1000	3	-	15: (abefg)×3
D016	✓	✓	200k	500	4	-	12: abf, (aef)×3
D038	✓	✓	100k	500	4	-	12: abf, (aef)×3
i633	✓	✓	200k	500	4	-	13: abef, (aef)×3

TABLE VI. GP Parameters: Common

Parameter	Value
Prob. crossover	90%
Prob. mutation	10%
Prob. internal crossover	85%
Prob. terminal mutation	75%
Tournament size	3
Elite migration	3

The notation `4@E` is shorthand for `E|E|E|E`. This target expression has one Nand gate, 5 Neg gates, and 11 Pos gates. Although reproducible in the Boolean language, it may be too complex for that to reasonably happen.

Table V summarizes the experiments undertaken. The columns Fix and Var refer to whether fixed or variable rate parameters were used. Fixed rates used throughout are cst=0.1 and inh=0.001. Variable rates will evolve one of 6 rate values 10^k for integer k, where $-5 \leq k \leq 0$. T is the maximum time used during SPI simulations. K is the number of data values generated per channel. Test channels are the number of channels used for fitness testing, and extra channels are those permitted in the expression, but not tested. Features refers to the statistics in Table IV. The total number of features is given, along with the set of features used. When multiple channels are tested, the feature sets might be different. For example, D016 uses features a, b, and f for one channel, and a, e and f for 3 channels.

Normalized sum of ranks is used for fitness evaluation (Section IV-B). Although unweighted sums are the default, we decided to use weighting a few times after inspecting results from early runs. In the Repressilator, we used weights of 3 for the "g" objective (frequency), and used 1 for the rest. For D016 (fix and var), we used a weight vector of (3,3,1) for the channel using objectives *abf*, and weights of 1 elsewhere.

Table VI shows the GP parameters common in all experiments, while Table VII show parameters for specific experiments, which are based on preliminary runs. A parameter sweep was not performed, and other settings could easily show improved results. Grammar-guided GP is used, and the grammar (not shown) defines variable-length lists of gate calls. To evaluate fitness, evolved gate expressions are converted into SPI-readable files, which are then interpreted with the SPI interpreter [16], [34].

TABLE VII. GP Parameters: Specific

Experiment	Popn size	Max gens	Init tree depth	Max tree depth
Basic OR	500	50	4, 5	6
Basic NOR	500	50	4, 5	6
Repressilator	500	30	4, 8	10
D016 (F,V)	1500	75	5, 7	9
D038 (F,V)	750	50	5, 7	9
i633 (F,V)	750	50	5, 7	9

B. Analyzing results

It is challenging to identify good solutions from the GP runs. This is because both the GP and target expressions are stochastic, and matching their behaviours is approximate and error prone. Expressions can have varying behaviours during different simulations. Fitness evaluation performs simulations once per expression, which may be inadequate for expressions with unstable behaviour. Stochastic time series are complex to characterize, and the particular statistical features used in experiments may not be ideal.

We processed experimental results as follows. Prior to the GP runs, each target system was simulated 1000 times, the time course statistics of the channels of interest were computed for each simulation output, and basic statistics (minimum, maximum, mean, standard deviation) were computed and saved. Solutions from each experiment were processed as follows. (1) The top 10 fittest individuals are saved from every run. Since 20 runs were done, 200 candidate solutions were saved per experiment. (2) Each solution was simulated 100 times. The time series feature vector for the channels of interest are calculated for each plot. From these, we compute: (i) Basic statistics (as above). (ii) *Z-score* match with the target expression (90% significance). (iii) Range compliance (*R-score*) with the target expression. This verifies whether the distribution of the expression's feature values resides within the target's observed range. The Z-score and R-score are heuristic aids to help identify solutions of interest. They cannot be relied upon for definitive matching. Many feature tests of our targets *do not* have normal distributions, and Z-scores will fail for them. The R-score does not consider distribution shapes.

Other analyses were also selectively used: (i) Re-ranking solutions: All 200 candidate solutions are combined, and re-ranked using normalized sum of ranks. (ii) Expression examination: Here, solution expressions can be compared to the target (in cases where the target is defined using SPI gates). (iii) Time plots: The time course behaviour can be plotted, and compared to that of the target. Each of these has advantages and disadvantages. For example, expression examination may ignore behavioural equivalence, and visual comparison of complex time plots can be error-prone.

VI. Results

A. Fitness scoring of results

TABLE VIII. Results summary. Statistics based on 200 solutions per experiment (10 solutions from each of 20 runs).

Experiment	Tot objs	Z-score hits			R-score hits		
		Avg	Best	# best	Avg	Best	# best
Basic OR	4	0.56	3	12	3.06	4	93
Basic NOR	4	0.48	3	1	3.78	4	157
Repressilator	15	1.93	13	1	8.31	15	11
D016 fix	12	1.43	5	1	9.14	11	8
D016 var	12	2.84	12	1	9.99	12	3
D038 fix	12	0.80	4	1	3.42	10	10
D038 var	12	2.18	6	2	8.76	11	12
i633 fix	13	3.99	11	1	11.99	13	61
i633 var	13	2.55	6	3	12.31	13	85

Table VIII summaries the Z-score and R-score analyses for all the runs. The total objectives column is the maximum number of hits possible for the hits. A "hit" is a Z-score or R-score match of a solution with a target (see description in Section V-B). The Avg column is the average over all 200 solutions, Best is the highest number of hits of any solution, and # best is the number of solutions with the Best score.

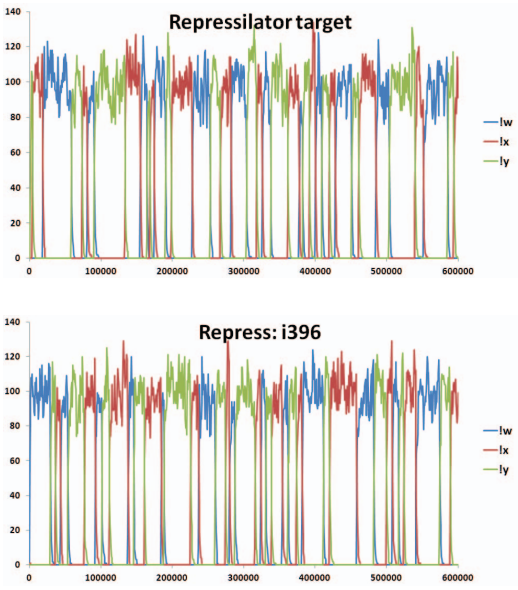


Fig. 3. Time-course plots for repressilator target, plus solution having Z-score match of 13.

Examining the table, note that the R-score is easier to satisfy than the Z-score. Matching the range of each objective over 100 simulations to the target's range is often satisfied, while Z-scores are not so easily matched. As noted earlier, Z-scores should not be relied upon. The target's objective values are often non-normally distributed, and in such cases the target will not match itself using Z-scores. In this sense, the R-score is a more accurate heuristic than the Z-score, albeit not very informative. Another insight is that variable rates have higher hit scores than the fixed rate runs in the D016 and D038 experiments. Since neither target expression is directly encodable in the Boolean gate language, variable rates added the necessary flexibility for expressions to be behaviourally closer to the targets. This effect is not evident in the i633 experiments, however, where the fixed rates have better Z-scores. This is probably because the target i633 expression is written with fixed rates.

The goal for evolved solutions is that their time-course behaviours match those of the target system. It is therefore useful to compare plots for a few solutions with their corresponding targets. Figure 3 shows plots for the repressilator target, as well as the solution with the highest Z-score hits (13 out of 15), as well as a perfect R-score of 15. Both plots are visually similar, and show round-robin oscillation of all 3 channels. The expression for this solution is:

```
Neg(x,(0.1,0.0001),y) |
  Neg(y,(0.1,0.0001),w) |
  Nor(w,w,(0.1,0.0001),x) |
  Nor(x,w,(0.0001,0.0001),y)
```

The Nor(w,w,x) gate is behaviourally (and logically) equivalent to Neg(w,x). Therefore, the first 3 terms are identical to the target expression. Although the final Nor term is extraneous, its encoded rates are small, and so it may play a minor influence.

Another plot comparison is in Figure 4. The i495 plot is the solution from the D016 Var experiment that has the highest Z-score of 12, and R-score of 12. Its expression is:

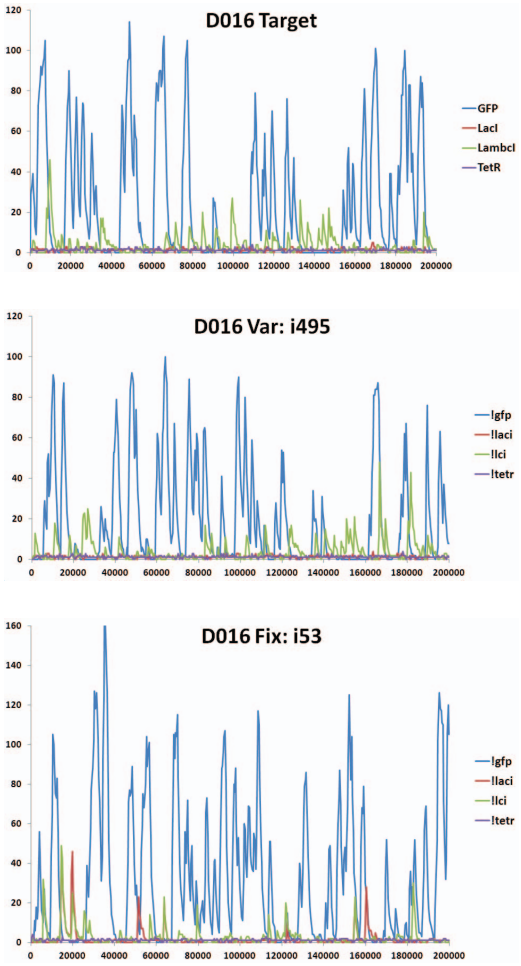


Fig. 4. Time-course plots: D016 target, i495 (var rate solution having Z-score match of 12), and i53 (fixed rate solution with no Z-scores matching).

```
BAnd(gfp,lci,(1.0E-05,0.001),lci) |
   Nor(lci,laci,(0.001,0.001),lci) |
   Neg(gfp,(0.001,0.0001),lci) |
   Neg(laci,(0.01,1.0E-05),tetr) |
   Neg(laci,(0.1,0.01),laci) |
   Neg(laci,(0.1,0.01),lci) |
   Neg(lci,(0.1,0.01),gfp) |
   Neg(tetr,(0.1,0.01),tetr)
```

The behaviours are visually similar, although the D016 target's GFP channel (blue) tends to show a higher average in this simulation instance. An example of a poor match is the i53 plot, which is a solution from the D016 fix experiment. It had a Z-score of 0, and R-score of 6.

Figure 5 shows the population fitness performance for the D016 experiments. The graphs plot the 12 raw objective values (feature statistic value errors), which is used by the sum of ranks scoring. Clearly the sum of ranks minimizes all objectives to different extents (Pareto ranking would not be able to do this with 12 objectives.) The Var runs have a higher initial error than the Fix, due to the extreme behaviours caused by wide ranges present in the channel rates. Of special note is the curve labelled "LCI avg" in the Fix plot. When using the sum of ranks scoring strategy, the performance of some objectives can be sacrificed in order to benefit the majority of

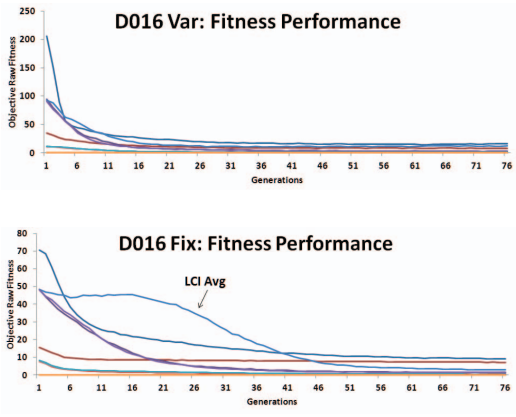


Fig. 5. Raw fitness performance plots of population for D016 runs (average over 20 runs). Each of the 12 plot lines is the error between objective values of target and GP population.

TABLE IX. GATE DISTRIBUTIONS IN D016 AND D038 SOLUTIONS. 200 SOLUTIONS PER EXPERIMENT.

	D016		D038	
Gate	Var	Fix	Var	Fix
Nor	33%	82%	42%	13%
Nand	49%	48%	54%	26%
Or	53%	-	22%	12%
And	25%	-	12%	7%
Xor	23%	46 %	36%	15%
Impl	47%	-	30%	5%
Equiv	31%	-	41%	12%
Pos	79%	100%	96%	58%
Neg	100%	100%	87%	100%

other objectives. In this case, the improvement of LCI avg noticeably lags behind the other objectives during the first half of the run, but improves significantly afterwards. If we assigned LCI avg a weight higher than the default of 1, this lag might have been avoided, but at the expense of the performance of other objectives.

B. *Examination of circuit expressions*

Solution expressions were studied with respect to gate composition, and the degree they matched the target expression. The Basic OR and NOR experiments were devised for the purpose of comparing circuit expressions. These experiments used fixed-rate Boolean gates, and omitted the Pos and Neg gates. Examining the 200 solutions for Basic OR, 190 expressions used an Or gate, and 105 used Nor. One exact match with the target was evolved. On the other hand, with the 200 Basic Nor solutions, none used Or gates, and 198 used Nor gates. No exact target matches occurred. This shows that GP is exploring expression schema appropriate to the behavioural requirements.

Target expression matching is difficult for D016 and D038, because the target is written in a different gate language than ours. A simple analysis is the gate distribution found in evolved solutions in Table IX. This table reports the percentage of solutions using each gate. The basic Pos and Neg gates are the most frequent gates. This supports the original motivation of Blossey *et al.* in proposing that these gates are fundamental in modelling gene regulation [15]. Another insight is that solutions in the variable rate language show a wider assortment of gates than fixed-rate solutions. Especially surprising is how the D016 fixed case entirely omits four gates (Or, And, Impl, Equiv) from all solutions. These same gates were also the least

TABLE X. TERM ANALYSIS FOR I633 FIX. TOTAL ARE THE NUMBER OF SOLUTIONS INCLUDING THAT TERM. EXACT ARE THE NUMBER OF SOLUTIONS WITH EXACT MATCH IN QUANTITY OF THAT TERM.

Target	Top ranked (10 tot)		Nand (8 tot)	
expression term	Total	Exact	Total	Exact
Nand(gfp,laci,gfp)	-	-	8	8
4@Neg(lci,gfp)	-	-	1	-
Neg(tetr,tetr)	3	4	6	5
2@Pos(gfp,gfp)	6	2	6	4
Pos(gfp,laci)	-	2	6	6
4@Pos(gfp,lci)	6	2	8	-
Pos(lci,gfp)	2	3	4	3
2@Pos(lci,lci)	7	2	8	6
Pos(lci,tetr)	2	3	1	1

% i633 target:	% i405	% i320	% i344
Nand(gfp,laci,gfp)	Nand(laci,gfp,gfp)	Nand(laci,gfp,gfp)	Nand(laci,gfp,gfp)
4@Neg(lci,gfp)			
Neg(tetr,tetr)	Neg(tetr,tetr)	Neg(tetr,tetr)	Neg(tetr,tetr)
2@Pos(gfp,gfp)	2@Pos(gfp,gfp)	2@Pos(gfp,gfp)	2@Pos(gfp,gfp)
Pos(gfp,laci)	Pos(gfp,laci)	Pos(gfp,laci)	Pos(gfp,laci)
4@Pos(gfp,lci)	Pos(gfp,lci)	Pos(gfp,lci)	Pos(gfp,lci)
Pos(lci,gfp)	Pos(lci,gfp)		
2@Pos(lci,lci)	2@Pos(lci,lci)	2@Pos(lci,lci)	2@Pos(lci,lci)
Pos(lci,tetr)			
Remaining terms...	Nor(tetr,tetr,tetr	Nand(gfp,lci,gfp)	Nand(gfp,lci,gfp)
	Pos(tetr,lci)	2@Neg(tetr,lci)	2@Neg(tetr,lci)
	Pos(tetr,tetr)	Nor(tetr,laci,tetr)	Nor(tetr,laci,tetr)
		2@Nor(tetr,tetr,tet	2@Nor(tetr,tetr,tetr)
		Pos(tetr,gfp)	Pos(tetr,gfp)
		Pos(tetr,tetr)	Pos(tetr,tetr)

Fig. 6. Three Nand solutions having top Z-scores. Yellow indicates exact match with target, and green is partial match.

used ones in the D038 fixed solutions. We hypothesize that the variable languages permit more variations of useful circuitry components, which also shows that rates and expressions are somewhat orthogonal in the search space. In addition, low rate values allow more "bloat terms" that have little or no effect on behaviour.

The i633 runs involve a more complex gate circuit that is replicable in our gate language. We examined the 200 solutions from the fix rate runs in two ways: (i) We re-ranked the solutions using the normalized sum of ranks, and saved the top 10 scorers. (ii) We found 8 solutions in total that used the identical Nand gate term (including channel arguments) found in the target expression. None of these Nand solutions were in the top 10 set. We then matched the terms of all these expressions with the target expression. Table X shows a summary of the matching. *Total* are the number of solutions that include this particular target term, albeit possible not the same number of instances as the target. *Exact* are the number of solutions that have the same instances as well. For example, when the target term is `2@pos(lci,lci)` and the solution term is `3@Pos(lci,lci)`, that is considered a match for Total, but not an exact match.

From this table, there is evidence that the Nand gate term has a significant influence on the rest of the expression. When solutions have the same Nand term as the target, the frequency of matches with other target terms increases, and most notably, the exact match counts. When this term is not found in an expression, there is a correspondingly lower match with the remaining expression.

Three of the i633 fix Nand solutions having the highest Z-scores are shown in Table 6. The left column is the i633 target, and each column shows a separate solution. Green indicates partial matches with the target (off in repetition factor), and yellow indicates exact matches. Many of the matches undoubtedly arise because Pos and Neg are frequently used, and there are only 16 channel argument combinations possible for each of them. On the other hand, the 8 Pos and Neg combinations residing in the target expression were common in most solutions. We surmise that many of these terms must play an important behavioural role. Although the variable rate case is not studied here, those solutions show a much wider range of terms than the fixed case.

VII. CONCLUSION

This research presents the definition of a stochastic Boolean gate language in the style of Blossey *et al.*'s stochastic gene gate language [15]. We showed that stochastic systems written in the language could be automatically synthesized using GP. As done elsewhere, we specified desired time-course behaviour with statistical features [17], [27]. Because experiments used upwards of 15 features as objectives, the many-objective sum of ranks scoring strategy was effective in evolving systems that were behaviourally similar to target systems.

The gate language was shown to be highly compositional. We discovered that Blossey *et al.*'s Pos and Neg gates more rudimentary than the Boolean gates, and GP often exploited the basic gates during evolution. It is more difficult to see higher-level causal relationships using Pos and Neg, versus the higher-level semantics afforded by Boolean gates. Note that expression behaviours *cannot* be fully understood via their logical semantics visa-viz truth tables. Full analyses requires consideration of the how time-series output changes over time, which is considerably more complicated to understand than basic truth-table equivalences. With the D016 and D038 experiments, although we tried comparing the causal relationships of channels in our solutions with the target expressions, we did not get very far. Gate expressions generate varying quantities of channels (substances) over time, in an often chaotic and unstable manner, and the mechanisms by which this happens can be unintuitive. Although stochastic gate expressions would not be straight-forward to create and understand by most people, the ones evolved with GP can be especially arcane.

There are a number of future directions for this research. A detailed study of the *functional* characteristics of gate expressions is required. It would be valuable to find biologically meaningful functional correspondences between circuit components and (say) the target expressions of D016 and D038. However, the results of our experiments here show that the effects of channel rates is critical (as is also discussed in [1]), and needs further study. Rate terms substantially alter the form of gate expressions, which shows that they introduce orthogonality into the search. The ability to evolve gate expressions for real-world data will require a better understanding of how to scale and control rate expressions. Furthermore, although Boolean logic is well understood, there is no reason to suppose that Boolean gates are the most natural means for describing phenomena such as gene regulation or metabolic pathways. It is interesting to consider SPI implementations of other gate languages, such as those in [35], as alternative modelling formalisms may be better suited to biological modelling. In any case, it is important to investigate the utility of the gate language for real-world biological systems, rather than the *in silico* examples studied exclusively in this paper.

Statistical features mitigate the problem of characterizing noisy, non-deterministic time series [28], [29]. As in [17], we

manually choose features to use in an *ad hoc* manner. Poor choices of features can negatively effect the results. Future work will consider the use of more principled feature selection strategies [36].

We will also enhance the GP implementation. When studying the D016 and D038 systems, Imada used multiple fitness cases, by "knocking out" the effects of specific channels [17]. Our GP runs would greatly benefit with this essential strategy. Our results also show the challenge of finding a suitable solution from a set of hundreds of candidates evolved over many runs. Our use of Z-scores were not guaranteed to show matches to non-normal distributions of feature values, and a non-parametric test such as Kruskal-Wallis would be worth consideration. However, this would not influence the quality of GP results, since fitness evaluation uses an absolute error measurement. More stable fitness measurements could be obtained by performing multiple simulations of gate expressions, albeit at a computational cost.

ACKNOWLEDGMENT

Thanks to Janine Imada for use of her time series feature code, Andrew Phillips for the use of his SPI machine interpreter, and to Cale Fairchild for assistance with system issues. This research is supported by NSERC DG 138467.

REFERENCES

[1] G. Fritz, N. Buchler, T. Hwa, and U. Gerland, "Designing sequentional transcription logic: a simple genetic circuit for conditional memory," *Syst Synth Biol*, vol. 1, pp. 89–98, 2007.

[2] H. Bolouri, *Computational Modeling of Gene Regulatory Networks – a Primer*. Imperial College Press, 2008.

[3] H. Ando, S. Sinha, R. Storni, and K. Aihara, "Synthetic gene networks as potential flexible parallel logic gates," *EPL*, vol. 93, 2011.

[4] C. Chaouiya and E. Remy, "Logical Modelling of Regulatory Networks, Methods and Applications," *Bull Math Biol*, vol. 75, pp. 891–895, 2013.

[5] I. Shmulevich, E. Dougherty, S. Kim, and W. Zhang, "Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks," *Bioinformatics*, vol. 18, no. 2, pp. 261–274, 2001.

[6] D. Wilkinson, *Stochastic Modelling for Systems Biology*. Chapman & Hall/CRC, 2006.

[7] A. Raj and A. van Oudenaarden, "Nature, Nurture, or Chance: Stochastic Gene Expression and Its Consequences," *Cell*, vol. 135, pp. 216–226, 2008.

[8] R. Hermsen, S. Tqans, and P. ten Wolde, "Transcriptional Regulation by Competing Factor Modules," *PLOS Computational Biology*, vol. 2, no. 12, pp. 1552–1560, December 2006.

[9] N. Kashtan and U. Alon, "Spontaneous evolution of modularity and network motifs," *PNAS*, vol. 102, no. 39, September 2005.

[10] R. Xu, D. Wunsch, and R. Frank, "Inference of Genetic Regulatory Networks with Recurrent Neural Network Models Using particle Swarm Optimization," *Computational Biology & Bioinformatics*, vol. 4, no. 4, pp. 681–692, 2007.

[11] S. Bulashevska and R. Eils, "Inferring genetic regulatory logic from expression data," *Bioinformatics*, vol. 21, no. 11, pp. 2706–2713, 2005.

[12] W. Banzhaf, "Artificial Regulatory Networks and Genetic Programming," in *Genetic Programming Theory and Practice*, R. Riolo and B. Worzel, Eds. Kluwer, 2003, pp. 43–61.

[13] L. Qian, H. Wang, and E. Dougherty, "Inference of noisy nonlinear differential equation models for gene regulatory networks using genetic programming and kalman filtering," *IEEE Transactions on Signal Processing*, vol. 56, no. 7, pp. 3327–3339, July 2008.

[14] X. Cai, P. Kodura, S. Das, and S. Welch, "Simultaneous Structure Discovery and Parameter Estimation in Gene Networks using a Multi-objective GP-PSO Hybrid Approach," *Intl Journal of Bioinformatics Research and Applications*, vol. 5, no. 3, pp. 254–268, June 2009.

[15] R. Blossey, L. Cardelli, and A. Phillips, "A Compositional Approach to the Stochastic Dynamics of Gene Networks," *Trans. in Comp. Sys. Bio (TCSB)*, vol. 3939, pp. 99–122, 2006.

[16] A. Phillips and L. Cardelli, "A Correct Abstract Machine for the Stochastic Pi-calculus," in *Proc. Bioconcur'04*, 2004.

[17] J. Imada, "Evolutionary synthesis of stochastic gene network models using feature-based search spaces," Master's thesis, Department of Computer Science, Brock University, 2009.

[18] J. Imada and B. Ross, "Evolutionary synthesis of stochastic gene network models using feature-based search spaces," *New Generation Computing*, vol. 29, no. 4, pp. 365–390, October 2011.

[19] B. Ross and J. Imada, "Using Multi-objective Genetic Programming to Synthesize Stochastic Processes," in *Genetic Programming - Theory and Practice VII*, R. Riolo, U.-M. O'Reilly, and T. McConaghy, Eds. Springer, 2010.

[20] R. Milner, *Communicating and Mobile Systems: the Pi-calculus*. Cambridge University Press, 1999.

[21] D. Gillespie, "Exact stochastic simulation of coupled chemical reactions," *J. Phys. Chem*, vol. 81, pp. 2340–2361, 1977.

[22] Wikipedia, "Flip-flop (electronics)," 2015, last accessed Mar 10, 2015. [Online]. Available: http://en.wikipedia.org/wiki/Flip-flop_%28electronics%29

[23] J. Koza, M. Keane, M. Streeter, W. Mydlowec, J. Yu, and G. Lanza, *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003.

[24] C. Chatfield, *The Analysis of Time Series: An Introduction*. Chapman and Hall/CRC, 2004.

[25] Y. Jin and J. Branke, "Evolutionary Optimization in Uncertain Environments - A Survey," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 303–317, 2005.

[26] W. Zhang, G. Yang, and Z.Wu, "Genetic Programming-based Modeling on Chaotic Time Series," in *Proc. 3rd Intl Conf. on Machine Learning and Cybernetics*. IEEE, 2004, pp. 2347–2352.

[27] J. Imada and B. Ross, "Evolutionary synthesis of stochastic gene network models using feature-based search spaces," *New Generation Computing*, 2010, in press.

[28] A. Nanopoulos, R. Alcock, and Y. Manolopoulos, "Feature-based classification of time-series data," in *Information processing and technology*, N. Mastorakis and S. Nikolopoulos, Eds. Commack, NY, USA: Nova Science Publishers, Inc., 2001, pp. 49–61.

[29] X. Wang, K. Smith, and R. Hyndman, "Characteristic-based clustering for time series data," *Data Min. Knowl. Discov.*, vol. 13, no. 3, pp. 335–364, 2006.

[30] C. C. Coello, G. Lamont, and D. V. Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2nd ed. Kluwer, 2007.

[31] P. Bentley and J. Wakefield, "Finding acceptable solutions in the pareto-optimal range using multiobjective genetic algorithms," in *Soft Computing in Engineering Design and Manufacturing*. Springer Verlag, 1997.

[32] D. Corne and J. Knowles, "Techniques for highly multiobjective optimisation: Some nondominated points are better than others," in *Proceedings GECCO 2007*. ACM Press, 2007, pp. 773–780.

[33] C. Guet, M. Elowitz, W. Hsing, and S. Leibler, "Combinatorial Synthesis of Genetic Networks," *Science*, vol. 296, no. 5572, pp. 1466–1470, 2002.

[34] A. Phillips, "Stochastic pi machine," last accessed Feb 12, 2015. [Online]. Available: http://research.microsoft.com/en-us/projects/spim/

[35] D. Wolf and A. Arkin, "Motifs, modules and games in bacteria," *Current Opinion in Microbiology*, vol. 6, pp. 125–134, 2003.

[36] H. Liu and H. Motoda, *Computational Methods of Feature Selection*. Chapman and Hall/CRC Press, 2008.