Evolving Stochastic Processes Using Feature Tests and Genetic Programming

Brian J. Ross
Brock University
Department of Computer Science
500 Glenridge Ave.
St. Catharines, ON, Canada L2S 3A1
bross@brocku.ca

Janine Imada
Brock University
Department of Computer Science
500 Glenridge Ave.
St. Catharines, ON, Canada L2S 3A1
jimada@bell.net

ABSTRACT

The synthesis of stochastic processes using genetic programming is investigated. Stochastic process behaviours take the form of time series data, in which quantities of interest vary over time in a probabilistic, and often noisy, manner. A suite of statistical feature tests are performed on time series plots from example processes, and the resulting feature values are used as targets during evolutionary search. A process algebra, the stochastic π-calculus, is used to denote processes. Investigations consider variations of GP representations for a subset of the stochastic π-calculus, for example, the use of channel unification, and various grammatical constraints. Target processes of varying complexity are studied. Results show that the use of grammatical GP with statistical feature tests can successfully synthesize stochastic processes. Success depends upon a selection of appropriate feature tests for characterizing the target behaviour, and the complexity of the target process.
Program Track: Genetic Programming

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming

General Terms

Experimentation

Keywords

Stochastic Process, Process Algebra, Time Series, Feature Tests, Genetic Programming

1. INTRODUCTION

A process is a series of actions or state changes that brings about a result. These result changes can be plotted as time series graphs. Process behaviour can be characterized by such time series. A deterministic process's behaviour is dependent upon the initial environment. Conversely, stochastic processes model behaviours in which factors that are unknown or too difficult to model can be abstracted with probabilities. Stochastic process behaviour is dependent on such probabilistic factors. Stochastic processes have been used to model various real-world phenomena, such as stock market behaviour, ecological balances of predators and prey, and protein levels during bio-molecular reactions. In particular, bio-network modelling has attracted great attention. Many important problems, such as gene regulation network modelling, use stochastic process models.

The synthesis of process models given example time plots of their behaviours has been well studied. The symbolic regression problem in GP is a basic process inference problem, if one considers the x-axis as being time [11]. Koza *et al.* use GP to reconstruct metabolic networks from example time series behaviours [12]. Kitagawa and Iba evolve metabolic networks using a GA representation of Petri nets [10]. Nonlinear differential equations can be used to model time plot curves, and have been subjects for evolutionary computation [4] [27]. Stochastic process evolution has also been studied. Leier *et al.* use set-based GP to evolve algebraic systems that denote oscillating behaviours [13]. Drennan and Beer [6] and Chu [5] use GAs to evolve stochastic models for biological networks [6]. Ross uses GP with a stochastic process algebra, but on strictly monotonic time series plots [24]. Imada uses GP with stochastic processes denoting gene regulatory networks [8].

The processing of noisy data is a much-studied fundamental issue in machine learning. Although noisy time series and stochastic time series are similar, and often amenable to the same strategies, there is a subtle difference between them. A noisy time series may in fact be generated by a deterministic process. Its output, however, may be partially obfuscated by external noise that is not to be incorporated in the underlying model. On the other hand, a stochastic process normally has stochastic factors encoded in the model; noise is not necessarily disregarded or filtered. Rodriguez-Vazquez and Fleming use GP to model oscillating chaotic systems [22]. Financial applications of noisy time series using GP include work by Schwarzel and Bylander [26], and Borrelli *et al.*[3]. Imada applies GP to the symbolic regression problem when noise is introduced [8][9]. Other work considering process evolution from noisy data includes [1] [30].

This paper investigates the use of GP to synthesize stochastic processes. We use a technique used by Imada for noisy time series as found in symbolic regression problems and gene regulation networks [8]. Target process behaviour is characterized by application of a series of statistical feature tests to the time series plots. We then evolve processes whose feature values are as close as possible to those of the target. Processes are denoted with a subset of the stochastic π-calculus, which is a process algebra for modelling stochastic processes [17][19]. It has been shown to be useful for modelling various biological and chemical systems [2][20]. Its modularity and simple semantics make it a suitable language for GP. The GP system defines the process algebra with a context-free grammar, which enables useful grammatical constraints to guide the form of solutions considered. Experiments examine the evolution of processes exhibiting a variety of stochastic behaviours.

There are many motivations for this research. The stochastic π-calculus is one of many process algebras used in the formal methods community, which is a field whose research is directed towards the formal analysis and modelling of concurrent (stochastic) systems. However, very little research exists in using machine learning towards the automatic synthesis of process algebra models. Process algebra are particularly well suited for GP, given their modular, programming language-like nature. Stochastic π-calculus models differ from conventional reaction-based models [10] [12], whose expressions denote the entire reaction that will ensue in a process. Rather, the stochastic π-calculus denotes simpler components, which interact with each another other during process execution in complex and often unpredictable ways. Resulting process executions are very dynamic and robust in nature. This makes them a challenging domain for GP.

The paper is organized as follows. Section 2 describes the stochastic π-calculus. The use of statistical features for denoting time series data is discussed in Section 3. Some specific methodologies used in the experiments, such as channel unification, grammatical constraints, and fitness strategies, are described in Section 4. Experiments and their results are given in Section 5. Conclusions are in Section 6.

2. A STOCHASTIC PROCESS ALGEBRA

$$\begin{aligned}
\mathrm{P} &::= \mathit{0} \;\|\; \mathrm{P|P} \;\|\; \mathrm{Repl}\ \pi\mathrm{:P} \;\|\; \Sigma \\
\Sigma &::= \pi.\mathrm{P} \;\|\; \mathrm{delay(t).P} \;\|\; \Sigma+\Sigma \\
\pi &::= \mathrm{?c} \;\|\; \mathrm{!c}
\end{aligned}$$

Table 1: Process Algebra Syntax

The stochastic π-calculus is a process algebra that models concurrency, stochastic behaviour, and mobility (dynamic network changes) [19]. We use a subset of the stochastic π-calculus (Table 1). We omit channel passing, renaming and restriction found in the full algebra, which makes our algebra similar in syntax to the CCS algebra [15]. Despite these omissions, arguably the most interesting feature of this process algebra is its stochastic semantics, which permits many complex behaviours to be modelled.

In Table 1, processes can be defined by a null process *0*, the concurrent operator "|", replication operator *Repl*, or choice Σ. A choice expression is one or more terms that can be stochastically selected. A term is an input (*?x*) or output (*!x*) action. A delay term, if selected, advances the clock by a stochastically-determined duration, as specified by the delay value t. Each channel c has a rate associated with it. A shorthand notation used is $K@P$, which means $P|P|...|P$ repeated K times.

Sub-process definitions with parameter passing are possible. Letting *Proc* be a channel name reserved to identify a sub-process:

Proc=P $\Rightarrow$ Repl (?Proc.P)

Calls to the sub-process are denoted "!Proc".

The stochastic semantics are based on the Gillespie algorithm [7], which is used in chemical simulations. Consider the following transition of an expression:

$$(?\mathrm{x}.P_1{+}\Sigma_1)|(!\mathrm{x}.P_2{+}\Sigma_2)|P_3 \overset{rate(x)}{\rightarrow} P_1|P_2|P_3$$

Here, ?x and !x are *active*, in that they are both able to communicate. A synchronous handshaking communication has arisen along channel x via its input and output terms, and the entire expression has transformed. The other choices of actions in the Σ_i terms have been preempted by this communication. The Gillespie algorithm selects the execution of x stochastically. Gillespie selection is identical to Roulette wheel selection. Each active channel has an area of the Roulette wheel proportional to its probabilistic strength (quantity $\times$ rate). A call to a random number generator lets the simulation environment select a channel from the wheel. Once the transition occurs, the global time counter is advanced by an amount inversely proportional to the probability of the selected action. This reflects the higher frequency of more probable actions.

During an interpretation of a stochastic π-calculus expression, the overall expression transforms dynamically as terms replicate, and choice terms appear and disappear. The changing quantities of active terms within the expression can be measured over time, which permits a time series characterization of process behaviour.

The stochastic π-calculus can denote complex systems elegantly and concisely [2][17]. Resulting time plots, however, can be unpredictable and highly sensitive to details of the expressions, as is commonly the case with nonlinear dynamic systems [28]. For example, the particular rate used for a channel can result in drastically different time series plots. Altering a single channel in an action term can immediately result in a deadlocked expression. This makes process specification challenging, both for humans and GP.

3. TIME SERIES AND STATISTICAL FEATURES

Stochastic processes generate time series which vary between separate interpretations, even when run from identical starting conditions. By examining a possibly noisy time series from a stochastic process, standardized characteristics are often calculable. By doing so, a complex time series might be accurately identified by a carefully selected set of feature characteristics.

We characterize process behaviour by analyzing the generated time series with a suite of statistical feature tests. Our system implements an ensemble of 17 uni-variate tests taken from [8] [16] [29]. Let the time series be denoted $T = (t, v_t)$, where v_t is the measured data value at time t. Experiments in this paper select from the following tests: (1) *Raw mean*: The mean of all v_t is computed. (2) *Raw standard deviation*: The standard deviation of v_t is computed. (3) *Skew*: If the data values are put into histogram bins, this measures the symmetry, or lack thereof, of the histogram. Positive values means the histogram skews to the right of the mean. Negative values means a skew left. Zero indicates a symmetric distribution around the mean. (4) *Serial correlation*: This measures the degree of fit to a white noise model. Small values indicate more noise in the data. (5) *Chaos*: This measures the sensitivity to initial values. It calculates the rate of divergence of nearby points, averaged over many measures. Negative values indicate stability, zero if steady-state, and positive if divergent, chaotic behaviour. (6) *Periodicity*: This complex computation detects cyclic activity which might vary in frequency. Trend is removed from the raw data, followed by calculation of the auto-correlation function. The periodicity is the time interval of the shortest detected cycle.

Given a target process, the statistical feature values for its output behaviour are computed. Hundreds of plots are generated, the feature values of all the plots are determined, and the means and standard deviations of these values are calculated. Next, a meaningful set of features must be chosen that competently characterizes the time series. One might be tempted to use *all* feature values. However, this is not recommended, since too many features creates a highly dimensional search space that is too difficult for evolution. Note that automatic feature selection is a research problem studied elsewhere [14]. Furthermore, the automatic identification of generalized time series is intractable.

We choose feature tests using a combination of principled and *ad hoc* selection. To help rationalize which features are worth consideration, we tabulate all the feature tests scores and compute for each a *stability* measurement: stability = μ/σ. A high stability means that the mean value does not vary a lot compared to its standard deviation, and hence may be an accurate measurement. Next, we manually select from the feature tests that are both stable, and seem sensible for the target behaviour at hand. For many processes, the feature tests to use will be obvious. For example, an oscillating time series should use periodicity.

4. METHODOLOGY

4.1 Channel unification

One issue of concern is whether it is detrimental to represent π-calculus actions within the GP tree with hard-coded channel names, for example, "!x" for channel x. Expressions can deadlock with a single change of a channel label. Also, we find that the majority of the initial population of expressions are deadlocked, and hence contribute nothing to the gene pool during evolution.

An alternative representation is to treat channels as variables to be unified. Channel unification permits channels labels to be set dynamically, according to the structure of expressions. Unlike hard-coded channels, this may reduce deadlock, and promote effective evolution. Using channel unification means that an expression is first transformed by a unification procedure. After all channel variables are unified, the modified expression can be used as usual.

A *free* channel variable is a placeholder for an actual channel. It has the form $A_{x,n}$, where A is a channel variable, with an associated channel label x and priority value n. Terms with higher priorities can take precedence over those with lower priorities, perhaps akin to dominant and recessive genes. Unifying $A_{x,n}$ with channel label z results in: unify($A_{x,n}$, z) $\Rightarrow z_n$. The z_n term is called a *ground* channel. It retains a priority value n until the expression unification procedure is complete, after which priorities are removed. A deterministic unification algorithm unifies all the channel variables in an expression. Unification happens only once per instance of channel variable in the expression, and it is permanent through the remainder of that expression's interpretation.

The *prioritized unification algorithm* uses a portion of the π-calculus interpreter, in order to find active terms (ie. those available to communicate) that *might* unify during actual interpretation. It finds the set of active channel terms, and unifies them using the following ordered set of rules:

1. Two free channel variables with the same channel label x are unified with x.
2. The free channel variable $A_{x,n}$ with the highest priority n is matched with the free variable $B_{y,m}$ with the lowest priority m. Both unify with x from A.
3. A free channel $A_{x,n}$ is unified with a ground channel x_m, having the same name as A's channel label x.
4. The ground channel x_n with highest priority n is matched with the free channel $A_{y,m}$ with the lowest priority m. A is unified with x.

If all variables have been unified, the process stops. Otherwise, the above is applied iteratively and greedily within a breadth-first π-calculus interpreter. Terms are unwound by performing handshaking of complementary terms, and putting the resulting transformed expressions in a queue. This attempts to transform expressions that are likely to occur should communication arise, irrespective of the stochastic model used in the π-calculus. The queue elements are then processed with the above rules, one after another, until all channel variables have been unified, OR a maximum queue size is reached (100 in our experiments). Any remaining free variables are unified with their composite channel labels.

A variation of the above, *free unification*, ignores priority values. The first channel term found of the type required in the rule is used.

A stochastic π-calculus interpreter is implemented in SICStus Prolog, which implements the above channel unification procedures as options. The interpreter is based on an abstract machine specification in [18].

4.2 Grammar-Guided Genetic Programming

Grammar-guided GP is used, and the process algebra is defined with a context-free grammar (CFG). It is useful for this research, as it permits the definition of grammatical constraints for the evolved process algebra expressions. The grammar can be tailored according to the the complexity of the process to evolve. The DCTG-GP system is used, in which logic-based attribute grammars are used to specify GP languages [23].

A typical CFG used in this paper is shown in Fig. 1. This grammar evolves channel rates, which is simply a list of float values. Next, two partitions of processes are defined. Each partition is set with a mask, or direction for action terms. The first partition is set to encode only input terms (Dir=in), and the other uses output terms. The direction is passed to the channel actions in *Ch* (not shown). This improves the chances of communication, by reducing

$$\begin{array}{rcl}
Expr & ::= & Rates,\ Procs(in)\ \|\ Procs(out) \\
Rates & ::= & Float,\ Float,\ ...(1\ per\ channel) \\
Procs(Dir)^{\dagger} & ::= & Proc_i = Choice\ \| \\
 & & Procs(Dir)|Procs(Dir) \\
Choice^{\dagger} & ::= & Term\ \|\ Choice + Choice \\
Term & ::= & Pi\ \|\ Pi.Pi\ \|\ Pi.Call\ \| \\
 & & Pi.Pi.Call\ \|\ Pi.(Call|Call) \\
Call & ::= & Proc\ Int \\
Pi & ::= & ?Ch\ \|\ !Ch\ \ (if\ \ Dir = in\ \ or\ \ out) \\
Ch & ::= & c\ \ (c \in \text{channels}) \\
Float & ::= & min_f \leq \mathbf{f} \leq max_f \\
Int & ::= & min_i \leq \mathbf{i} \leq max_i
\end{array}$$

Figure 1: A grammar for the stochastic π-calculus

potential deadlock. The *Procs* rule defines sub-process definitions. Process definitions are indexed with an integer value. Calls to them likewise use an integer expression in the rule *Call*. The integer field is processed to index a process defined within that partition. The *Choice* rule defines the main body of expressions. The *Term* rule shows the different kinds of expression sequences allowed. Note that the terms are *guarded*, which means that each recursive call to a sub-process must involve at least one action *Pi* beforehand. Finally, floats and integers fall within specified ranges.

Note that the *Procs* and *Choice* rules are tagged with †. These grammar rules permit Ohno-style subtree duplication, implemented with a special mutation operator. For example, a tree node labelled C may be replaced with $C+C$. A node delete operator performs the inverse transformation. *Procs* and *Choice* can have user-specified maximum sizes.

Other constraints encodable in the CFG may include mask removal, unification variables (Section 4.1), delay terms, parameter passing to processes, adding or removing partitions, terms size limits, and channel rates.

4.3 Fitness evaluation

Fitness evaluation follows a technique used by Imada for noisy symbolic regression and gene regulatory network synthesis[8][9]. The GP expression is interpreted by the stochastic π-calculus interpreter, resulting in a time series for each channel of interest. The feature values from Section 3 selected for a given target process are computed for the time series. The majority of these feature calculations are implemented in C, and are called from the Prolog-based GP system (described below). The periodicity is computed with the R system [21].

Once the feature values for the GP expression are determined, the Euclidean distances between them and the target feature values for all channels ($j \in ch$) is computed:

$$Distance = \sum_{j \in ch} \sqrt{\sum_{i \in features} \left(\frac{(v_{ij} - t_{ij})}{\sigma_{ij}} \right)^2}$$

where v_{ij} is the computed feature value, t_{ij} is the target value, and σ_{ij} is the standard deviation for that feature as exhibited by the target process. The scaling by σ_i can be replaced by the mean μ_i if desired.

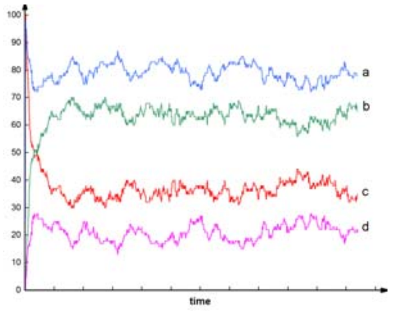
An option is to interpret an expression multiple times during fitness evaluation. In such cases, the mean of all the distances for the multiple time series is used as the fitness.

5. EXPERIMENTS

5.1 KNa2Cl

Na = !ionize1.!deionize1.Na	
K = !ionize2.!deionize2.K	
Cl = ?ionize1.Cl_minus + ?ionize2.Cl_minus	
Cl_minus = ?deionize1.Cl + ?deionize2.Cl	
KNa2Cl = 100@Na | 100@Cl | 100@K	
rate(ionize1)= 100.0	rate(ionize2) = 30.0
rate(deionize1)= 10.0	rate(deionize2) = 20.0

Table 2: KNa2Cl target process



?ionize2 (a), ?deionize1 (b), ?ionize1 (c), ?deionize2 (d)

Figure 2: KNa2Cl plot (input terms)

This example is taken from [17]. The target process models the ionization and deionization of the KNa2Cl co-transporter:

$$K + Na + 2Cl \iff K^{+} + Na^{+} + 2Cl^{-}$$

The target stochastic π-calculus expression is shown in Table 2, and an example time-series plot for the input terms is in Fig. 2. Each plot shows the measured quantity over time, of the output channels ($?ionize1$, etc.).

Although the process plot in Fig. 2 exhibits noise, we will use mean and standard deviation as the two feature tests. After interpreting the expression 500 times, feature values such as the following were computed for the input terms: (i) ?ionize1: mean (μ=37.0, σ=1.2), std dev (μ=6.2, σ=0.5); (ii) ?ionize2: mean (μ=79.0, σ=1.2), std dev (μ=4.0, σ=0.4); (iii) ?deionize1: mean (μ=62.9, σ=1.2), std dev (μ=6.3, σ=0.5); (iv) ?deionize2: mean (μ=20.9, σ=1.2), std dev (μ=3.6, σ=0.5). The output terms had similar calculations (not given). Therefore, feature evaluation uses mean and standard deviation scores for the in and out terms for the ionize1, ionize2, deionize1, and deionize2 channels. This results in a 16-term weighted sum, which is normalized by either the standard deviation or mean of the target scores.

Global parameters for all the runs are in Table 3. A few parameters requiring explanation are the following. Because the majority of random expressions in the initial population are deadlocked, the initial population is oversampled, and the worst are culled from it. Lamarckian evolution (local search) is performed on the initial population, to help "boost" its viability. One-third of the culled population will have reproduction operations applied to each member a maximum of 5 times. As soon as a change improves the

Parameter	*value*
Initial popn.	4000
Running popn.	1500
Unique popn	yes
Generations	50
Runs/experiment	20
Init max tree depth	8
Max tree depth	10
Prob. crossover	0.90
Prob. mutation	0.10
Prob. terminal mutation	0.045
Prob. node delete mut.	0.036
Prob. tree mutation	0.015
Prob. clone mut.	0.004
Tournament size	4
Lamarckian boost init. popn.	0.33 of popn,
Lamarckian evolution	0.05 of popn., every 6 gens
Elite migration	5

Table 3: Common KNa2Cl parameters

	Target	*Scale factor*	*Choice size*	*# interp.*	*I/O mask*	*Unif.*
A:	partial	σ	4	4	yes	no
B:	partial	σ	2	4	yes	no
C:	partial	μ	2	4	yes	no
D:	full	σ	2	4	yes	no
E:	full	σ	2	1	yes	no
F:	full	σ	2	1	no	no
G:	full	σ	2	1	no	yes (pri)
H:	full	σ	2	1	no	yes (free)

Table 4: KNa2Cl experiments

fitness, the new expression replaces the old one in the population. This local refinement is applied every 6 generations during the run, on the top 5% of the population. The population always contains unique trees at all times.

The KNa2Cl experiments are shown in Table 4. Experiments A, B and C target the Cl and Cl_minus expressions in Table 2, and the remaining expressions are supplied in a wrapper. Experiments D through H remove the Na and K expressions from the wrapper. Experiments G and H use priority and free channel unification respectively. The maximum choice term size is either 2 or 4. Rates are fixed.

The single expression with the best fitness score seen during a GP run is designated as the solution. The fitness for a solution expression is re-evaluated, and the resulting fitness is its test score.

5.1.1 KNa2Cl Results

The experiments that target partial expressions all obtained exact hits: A got 2, B got 6, and C got 2. Hits were determined by examining the expressions, to see if they matched the target expressions in Table 2. This suggests that the partial target was tractable for these runs, and that smaller choice expressions and σ scaling are preferred.

Experiments D through H, which target the larger process expression, did not result in hits. We believe that better results with the full expression require a more careful selection of feature tests, and evaluation of fewer channels. Nevertheless, these experiments did convey useful insights: variations in number of interpreter evaluations, I/O mask constraints and unification did not affect results.

Statistical significance tests such as the 2-sample t-test are erroneous to use for these experiments, because the fitness scores of solutions are *not* normally distributed. Nevertheless, for curiosity's sake, a 2-sample t-test assuming unequal variances was applied to the training and testing scores of all solutions from all runs (except experiment C, because it uses a different fitness scale). The result suggests that experiments A and B show superior performance to experiments D through H ($p<0.05$). This may confirm that the partial expression experiments are more tractable than the full expression versions, which is evident by the exact hits obtained. This t-test also shows that A and B are statistically equivalent, as are experiments D through H.

The Kolmogorov-Smirnov test (for non-normal distributions) was also applied. It reported no statistical difference in training and testing performance between all experiments, at least for $p<0.05$. This result is somewhat confounded by the fact that multiple hits were indeed obtained for experiments A and B, while none for D through H. More runs could strengthen the statistical significance of the observed results. Perhaps the fitness ranges of hit expressions overlap the ranges of non-hit solutions to a great extent, making true solution expressions more difficult to identify in the fitness space. A different selection of feature tests could affect this.

5.2 Repressilator gate

```
Gene(a,b) = (delay(0.1).(Protein(b).0) | Gene(a,b))
              + (?a.delay(0.0001).Gene(a,b))
Protein(b) = !b.Protein(b) + delay(0.001)
Repressilator = gene(x,y) | gene(y,z) | gene(z,x)
rate(x)=rate(y)=rate(z)= 1.0
```

Table 5: Repressilator target process

The repressilator circuit is a genetic regulatory circuit that produces oscillating behaviour [2]. Table 5 shows a stochastic π-calculus repressilator gate, as well as the wrapper circuit *Repressilator* that drives the circuit. The resulting oscillation behaviour is in Fig. 3 (a).

The goal is to evolve expressions with behaviours equivalent to that of the Gene and Protein expressions in Fig. 5. We select the following 5 features: mean, standard deviation, serial correlation, chaos, and periodicity (see Fig.3 (a) for the values). Fitnesses are scaled by σ. The rates of channels x, y, and z are set to 1.0. The features are measured for the output terms of x, y, and z, which gives a total of 15 terms for the weighted sum.

Two separate experiments were performed (20 runs each): (A) 2 sub-process expression limit; (B) 1 to 3 sub-processes limit. The GP parameters are in Table 3, except for the following changes: 30 generations; maximum tree size of 12; probability crossover is 0.85; probability mutation is 0.10; probability terminal mutation is 0.075; and probability tree mutation is 0.025. No channel unification is used.

One interpretation per expression is performed during fitness evaluation. Interpretations are limited to a maximum of 20,000 transitions, or a clock time limit of 200,000.

For efficiency purposes, interpreted streams are filtered by using every third time value in the process output.

5.2.1 Repressilator Results

	Training fitness		**Testing fitness**	
	# within	*# within*	*# within*	*# within*
Run	*target range*	$\mu \pm \sigma$	*target range*	$\mu \pm \sigma$
A	18	5	6	1
B	16	5	3	0

Table 6: Repressilator summary (20 runs)

Table 6 shows that runs generated solutions with fitnesses falling within the observed range of the target repressilator process, and some solutions were within one standard deviation of the target's mean fitness. The target fitness range was determined by running the repressilator 100 times, and tabulating the range of fitness values obtained. Testing scores are more discriminating, and show that solution behaviour naturally varies during different interpretations.

Some example plots are shown in Fig. 3. The labels a, b, and c label specific channels within the plots. Although no identical solution to the target expression arose, the following was a "near hit" from the runs in B (hand-simplified):

$$\begin{aligned} \text{Gene(a,b)} &= \text{(delay(1.0).(Protein(a,b).0) | Gene(a,b))} \\ &\quad + \text{(?b.delay(0.01).Gene(a,b))} \\ \text{Protein(a,b)} &= \text{!a.(Protein(a,b)|Protein(a,b)+ delay(0.01)} \end{aligned}$$

This expression shares similarities with the target in Table 5. It also exhibits similar feature tests values, fitness scores, and time series plot (Fig. 3 (b)). Other than using a different time scale than the target process, its time series plot is remarkably similar to the target in (a). Two other solutions' plots are shown in (c) and (d) in Fig. 3. Oscillating behaviours often arose, although the oscillations were not always cleanly alternating as in the target plot. As can be seen, the periodicity is a difficult feature to fit.

5.3 A Cyclic Process

The repressilator experiment suggests that periodicity is a challenging feature value to derive in solutions. This experiment concentrates on using periodicity and skew as the only feature tests. The target values are hand-specified, rather than calculated from a known target expression. We supply the following wrapper expression:

$$(Repl?c1)|(Repl?c3)|(Repl?c5)|P0|P1|P2$$

GP is to evolve P0, P1, and P2, using the 5 channels c1 through c5. The Repl terms are generating infinite streams of communications, and improve the chances of oscillating output. The process should have 2 observed channels: !c1 is to have a skew of 3.0, and period of 2.0; and !c3 has a skew of -3.0 and period of 6.0. Channel rates are to be evolved. Rather than use a simple floating point type for rates, we use an integer index, which is converted to a float rate using the following discrete logarithmic transformation:

$$rate = 10^{(Index \ \ modulo \ \ 7)-4}$$

This uniformly generates one of the rates: 0.0001, 0.001, 0.01. 0.1, 1.0, 10.0, 100.0. Since the ratio of channel rates is of primary importance, this scheme improves the likelihood of high channel ratios.

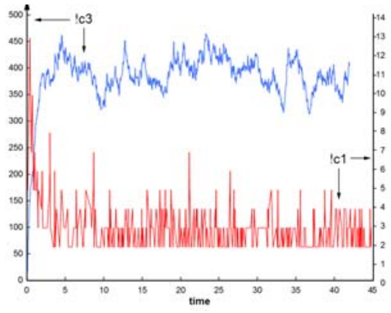
The GP parameters used are in Table 3, except for the following: initial population size is 3500; running population size is 750; maximum 30 generations; max 13 runs; maximum initial tree depth is 8; and maximum tree depth is 12. Fitness values are scaled by feature value σ's. A maximum of 3 sub-process terms and 4 choice terms are possible. No channel unification is used.

In order to give the interpretation adequate time to create cyclic behaviours, an iteration limit of 130,000 and time clock limit of 50.0 is used. With such high iterations, the resulting stream from the stochastic π-calculus interpreter can be enormous, the output stream is filtered by saving every 25th series value. Then 500 evenly-spaced time values are linearly interpolated from this stream. The resulting stream of 500 values is efficiently processed.

5.3.1 Cyclic Process Results

	!c1		!c3	
	period	*skew*	*period*	*skew*
Target	**2.0**	**3.0**	**6.0**	**-3.0**
Avg	1.9	2.3	6.5	-1.9
Best	2.1	2.9	8.1	-3.5

Table 7: Cyclic test score summary (avg 100 plots)

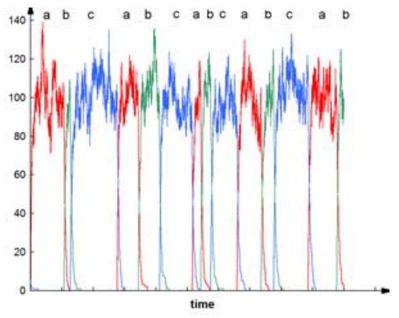


!c1 and !c3. Vertical scales are indicated.

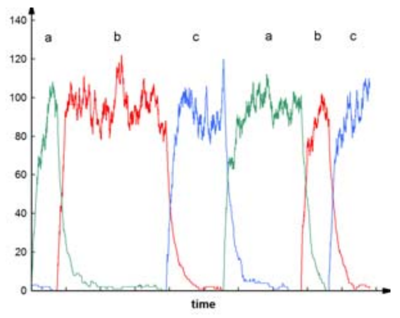
Figure 4: Cycle solution plots

A summary of the test results is in Table 7. The "Avg" row specifies the average of each feature over all 13 solutions. Hence this row does not actually correspond to any single solution. The "Best" row specifies the features of the solution with the best overall fitness score. Most solutions exhibited feature values that were reasonably close to the target features This suggests that evolution benefited from the relatively low dimensional feature space of 4 features. The selection of periodicity and skew, along with the particular wrapper expression used, seemed to define a tractable search space for this problem.

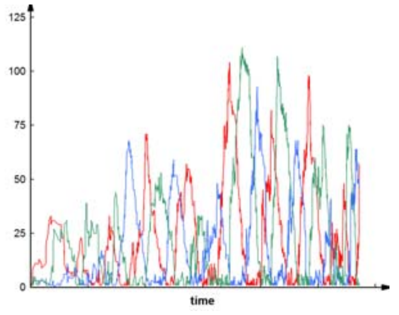
An example of the best solution's plot is in Fig. 4. The "!c3" plot (blue, left scale) has a periodicity of 8.1. The effect of the negative skew is to more data points lower than the mean, which results in the shape of plot shown. The "!c1" plot (red, right scale) is spikier, due to positive skew, and its shorter period is evident.



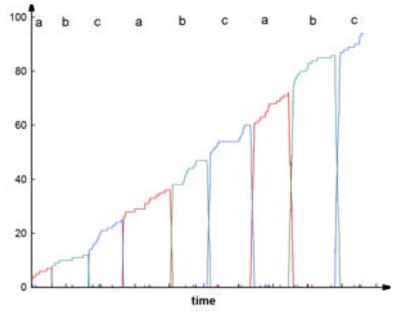
(a) Target: mean=33.6, std dev=43.9, sc=8.75, chaos=0.060,period=40915.8



(b) Solution 1: mean=33.8, std dev=42.3, sc=9.73, chaos=0.063, period=2193.8



(c) Solution 2: mean=27.8, std dev=32.3, sc=5.82, chaos=0.072, period=9218.4



(d) Solution 3: mean=30.7, std dev=49.3, sc=8.65, chaos=0.085, period=1.41

Figure 3: Repressilator plots. Feature values averaged over all 3 channels, over 100 plots.

The best solution expression is (hand simplified):

P0 = ?c2.P0 + !c3
P1 = !c2 + !c1.P1 + !c2.(P0|P1)
P2 = !c1.(P0|P1|P2) + !c1.(P0|P1|P2)
rate(c1) = 100.0, rate(c2) = rate(c3) = 1.0

Note that P2 has two identical choice terms. This has the effect of doubling the probability of selecting the "!c1" guard. Channels c4 and c5 are not used.

6. CONCLUSION

This paper is one of the first examples of the automatic synthesis of stochastic processes written in the stochastic π-calculus. Our results show that exact solutions are readily available for simple processes. More complex processes more likely result in behaviourally equivalent solutions – process expressions whose feature scores may be within the observed bounds of the target features. Note that our criteria for solution selection – the single expression with the best overall fitness in a run – is conservative, and perhaps too discriminating. A better solution may indeed exist in the population, but its fitness score may be poorer, due entirely to stochastic effects. By examining more individuals in the final population, perhaps with additional interpretations or feature tests, better results may have been reported.

A surprising result is that channel unification had no positive or negative effect when used. This suggests that the high frequency of deadlock in early generations is not a major detriment to evolution in the long term. Channel unification as implemented here did not flatten the search space.

Search spaces for stochastic systems are challenging in general, since target behaviour is inexact and often ambiguous. Unlike work in deterministic bio-network evolution in which the network model denotes reactions [12] [10], the stochastic π-calculus denotes *components* of interacting subprocesses. Although initial expressions may be simple, the overall run-time reaction network can be robust and complex, as it dynamically changes during interpretation.

This research improves on earlier work in [24] by considering true stochastic behaviours as measured by statistical feature scores, rather than monotonic "fixed" time series plots evaluated by sum-of-errors. Our fitness evaluation procedure is based on that used in [8] on noisy symbolic regression and gene regulatory network problems, which also uses normalized distances based on statistical feature scores. Research similar in spirit to this paper is [13], which uses another formal representation of bio-networks that is interpreted using the Gillespie algorithm. They also use feature tests to denote their time series plots. Main differences are the scope of feature tests implemented, as well as the semantic differences between both algebras.

The results confirm that statistical feature tests are useful for characterizing complex stochastic process behaviours. It is surprising how process behaviours can be specified with very little information. For example, mean and standard deviation were used for the partial KNa2Cl runs. This being said, the selection of feature sets for more complex processes is more difficult, and an open problem [14]. Overall success for complex processes depends upon careful selection of a reasonable number of effective features. The full KNa2Cl runs uses 16 features, and could benefit with a reduction.

Future work will consider means for scaling up the complexity of processes, larger subsets of the π-calculus, and more effective feature selection and evaluation strategies. Recent work shows that multi-objective evolution is a promising approach [25].

Acknowledgements: Research supported by NSERC Operating Grant 138467, and an NSERC PGS A award.

7. REFERENCES

[1] P. Angeline. Evolving Predictors for Chaotic Time Series. In *Proc. SPIE: Application and Science of Computational Intelligence*, volume 3390, pages 170–180, 1998.

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

[3] A. Borrelli, I. De Falco, A. Della Cioppa, M. Nicodemi, and G. Trautteura. Performance of genetic programming to extract the trend in noisy data series. *Physica A: Statistical and Theoretical Physics*, 370(1):104–108, 2006.

[4] D.-Y. Cho, K.-H. Cho, and B.-T. Zhang. Identification of biochemical networks by S-tree based genetic programming. *Bioinformatics*, 22(13):1631–1640, 2006.

[5] D. Chu. Evolving genetic regulatory networks for systems biology. In D. Srinivasan and L. Wang, editors, *Proc. CEC 2007*, pages 875–882, Singapore, 25-28 Sept. 2007. IEEE Press.

[6] B. Drennan and R. Beer. Evolution of repressilators using a biologically-motivated model of gene expression. In L. R. et al., editor, *Artificial Life X: Proc. Tenth Intl. Conf. on the Simulation and Synthesis of Living Systems*, pages 22–27. MIT Press, August 2006.

[7] D. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem*, 81:2340–2361, 1977.

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

[9] J. Imada and B. Ross. Using Feature-based Fitness Evaluation in Symbolic Regression with Added Noise. In *Proc. GECCO 2008 Late Breaking Papers*, July 2008.

[10] J. Kitagawa and H. Iba. Identifying Metabolic Pathways and Gene Regulation Networks with Evolutionary Algorithms. In G. Fogel and D. Corne, editors, *Evolutionary Computation in Bioinformatics*, pages 255–278. Morgan Kaufmann, 2003.

[11] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[12] 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.

[13] A. Leier, P. Kuo, W. Banzhaf, and K. Burrage. Evolving noisy oscillatory dynamics in genetic regulatory networks. In P. C. *et al.*, editor, *EuroGP 2006*, volume 3905 of *LNCS*, pages 290–299. Springer, 2006.

[14] H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining.* Kluwer Academic Publishers, 1998.

[15] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[16] A. Nanopoulos, R. Alcock, and Y. Manolopoulos. Feature-based classification of time-series data. In *Information processing and technology*, pages 49–61. Nova Science Publishers, Inc., Commack, NY, USA, 2001.

[17] A. Phillips. The stochastic pi machine, 2008. http://research.microsoft.com/ aphillip/spim/. Last accessed Dec 9, 2008.

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

[19] C. Priami. Stochastic pi-Calculus. *The Computer Journal*, 38(7):579–589, 1995.

[20] C. Priami, A. Regev, E. Shapiro, and W. Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80:25–31, 2001.

[21] R Development Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2007.

[22] K. Rodriguez-Vazquez and P. J. Fleming. Evolution of mathematical models of chaotic systems based on multiobjective genetic programming. *Knowledge and Information Systems*, 8(2):235–256, Aug. 2005.

[23] B. Ross. Logic-based Genetic Programming with Definite Clause Translation Grammars. *New Generation Computing*, 19(4):313–337, 2001.

[24] B. Ross. Using Genetic Programming to Synthesize Monotonic Stochastic Processes. In *Proceedings CI-2007*, July 2007.

[25] B. Ross and J. Imada. Using Multi-objective Genetic Programming to Synthesize Stochastic Processes. In *Genetic Programming - Theory and Practice*, May 2009.

[26] R. Schwaerzel and T. Bylander. Predicting currency exchange rates by genetic programming with trigonometric functions and high-order statistics. In M. Cattolico, editor, *GECCO 2006*, pages 955–956. ACM, 2006.

[27] F. Streichert, H. Planatscher, C. Spieth, H. Ulmer, and A. Zell. Comparing genetic programming and evolution strategies on inferring gene regulatory networks. In K. et al., editor, *GECCO-2004*, volume 3102 of *LNCS*, pages 471–480, Seattle, WA, 2004. Springer-Verlag.

[28] S. Strogatz. *Nonlinear Dynamics and Chaos.* Westview Press, 1994.

[29] X. Wang, K. Smith, and R. Hyndman. Characteristic-based clustering for time series data. *Data Min. Knowl. Discov.*, 13(3):335–364, 2006.

[30] 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*, pages 2347–2352. IEEE, 2004.