Evolutionary Learning and Stochastic Process Algebra

Brian J. Ross BROSS@BROCKU.CA
Brock University, Department of Computer Science, 500 Glenridge Ave., St. Catharines, ON L2S 3A1 Canada

Abstract

This extended abstract discusses research using genetic programming to evolve stochastic processes, as modelled in the stochastic pi-calculus.

1. Introduction

Process algebras are formal systems for the specification, analysis, and simulation of concurrent systems (Hoare, 1985; Milner, 1989; Milner, 1999). The stochastic pi-calculus is a process algebra that includes a model of probability in its semantics of processes (Priami, 1995). It efficiently models one probabilistic execution path or trajectory within the universe of possible outcomes. This contrasts with other systems that model the entire Markov space of probabilistic behaviours. A trajectory is a time-domain series of quantities, each of which measures the activity level over time of some channel of interest. The stochastic pi-calculus has been useful for modelling a variety of biological and chemical networks, for example, gene regulation, protein interaction, and chemical reactions (Blossey et al., 2006; Phillips et al., 2006; Priami et al., 2001). Such systems are characterized by time-series plots of measurements of elements (genes, proteins, metabolites, etc.) that dynamically change over time. The stochastic pi-calculus is a natural means for modelling these natural systems, since the elements of the natural system are directly mappable to channel terms within the algebra. The time-series trajectories seen from model simulations often reflect the behaviours of the natural entities.

Deriving a process algebra description of a known process is fairly easy to do; it is similar to using a programming language. On the other hand, specifying a suitable process algebra expression that might generate a given set of time-series plots is much more difficult. Stochastic pi-calculus process are dynamic systems, whose state changes over time. It is well-known that the behaviours of dynamic systems are challenging to intuit and understand. Hence they have been a popular subject of machine learning research, since it is posited that their complexities make them good candidates for automated analysis on the computer.

This abstract outlines research in using evolutionary computation techniques for synthesizing stochastic pi-calculus systems. Given set(s) of time-series plots of some behaviour of interest, genetic programming is used to evolve a stochastic pi-calculus expression that can generate the target behaviour. Preliminary experiments have evolved stochastic pi-calculus expressions that can generate monotonic processes (those with strictly increasing, decreasing, or constant time-series curves). Current and future research will extend the scope of processes to be handled, as well as improve the efficacy of evolution.

2. Related work

Genetic programming has been used to evolve expressions written in the CCS process algebra (Ross, 1998b). The CCS language is represented as an S-expression tree within the GP system. Interpretations of candidate expressions are performed, and the resulting trace behaviours are evaluated and scored, to obtain a fitness value. One successful experiment evolved a CCS expression that implemented a concurrent version of a parity-2 tester. For example, consider the expression:

$$(!E \mid stream)\backslash\{t, f\}$$

where *stream* is an endless stream of the form

$$f.t.f.f.t.f.t.f.t \cdots$$

A parity tester E generates either a y or n if each pair of input actions is even parity ($f.f$ or $t.t$) or odd parity ($t.f$ or $f.t$). Using GP, one even-parity checker evolved for E is:

$$E = f.(f.y + t.n) + t.(f.n + t.y)$$

Appearing in *Proc. 1st Intl Workshop on the Induction of Process Models*, ICML 2007, Corvallis, OR, 2007.

The main difficult to be addressed in such an experiment is the potential for combinatorial overflow of interpretation output, which is likely to arise with nondeterministic choice and interleaving.

A later paper examined the evolution of cyclic (iterative) CCS expressions (Ross, 1998a). One insight obtained with this latter work is that success is more likely when constrained instances of process algebra are considered.

Petri nets are related to process algebra, and likewise are used to model concurrent systems. Kitagawa and Iba use a genetic algorithm to evolve Petri nets that model metabolic pathways (Kitagawa & Iba, 2003). Target behaviours were denoted by time-series plots of substrate quantities, as generated by *in silico* simulations. Their results are impressive, as the resulting evolved networks were nearly identical to the target ones. The network model they use is deterministic, and the consequent lack of probabilistic variation or noise is an advantage for the genetic algorithm.

Koza *et al.* study the automatic synthesis of metabolic pathways with genetic programming (Koza et al., 2003). Using cellular encoding, evolved GP expressions construct candidate networks from an embryonic network. These networks in turn denote electrical circuits, and their components denote biochemical reactions. A constructed network is interpreted by an electrical circuit simulator, and electrical circuit quantities (voltages) are measured with respect to the simulator clock. These measurements define time-series plots. The plots are fitted to those of the desired target behaviour. Using this approach, they successfully evolved a variety of non-trivial metabolic pathways, to an impressive level of accuracy. Once again, their process model is deterministic.

Finally, there are vast bodies of research in the areas of time series analysis and dynamic systems modelling. Time-series models are ubiquitous in the real-world, and are used to describe diverse areas such as the stock market, Internet activity, machine vibrations, and gene expression levels. Their challenging nature has meant that they are interesting applications for machine learning research.

3. An example experiment

This experiment is an outline of one detailed in (Ross, 2007). Please see (Priami, 1995; Phillips & Cardelli, 2005) for a thorough discussion of the stochastic pi-calculus.

The goal is to use GP to evolve a stochastic pi-calculus expression that could account for example time-series behaviour of some target process. The operators from the stochastic pi-calculus considered here are sub-process definitions (!*proc*), sequential composition (:), parallel composition (#), choice (+), finite repetition (@), channel input and output (c and $\overline{c}$), and null (**0**). The GP system used is the DCTG-GP system (Ross, 2001). This is a Prolog-based system that utilizes grammar-guided genetic programming. The context-free grammar specification for the stochastic pi-calculus is in Figure 1. This grammar is used by DCTG-GP to initially construct random expressions, as well as manipulate existing expressions with crossover and mutation operators. This grammar constrains the space of candidate expressions, thus avoiding some classes of nonsensical and inefficient expressions that would arise if a more basic grammar were used.

$$
\begin{array}{rcl}
Start & ::= & ChRates,\ Proc\#Proc\#Topexpr \\
ChRates & ::= & Float,\ Float,\ ... \\
Proc & ::= & !\mathbf{proc}_i : Expr \\
Topexpr & ::= & Int@(PiCall : Guardseq) \\
 & & \mid\ Topexpr\#Topexpr \\
Expr & ::= & Choice \mid Expr\#Expr \\
Choice & ::= & Pi : Guardseq \mid Choice + Choice \\
Guardseq & ::= & \mathbf{0} \mid PiCall : Guardseq \mid Expr \\
PiCall & ::= & Pi \mid \overline{\mathbf{proc}}\ Int \\
Pi & ::= & NormCh \mid \overline{NormCh} \\
NormCh & ::= & \mathbf{c} \in \text{channels} \\
Float & ::= & min_f \leq \mathbf{f} \leq max_f \\
Int & ::= & min_i \leq \mathbf{i} \leq max_i
\end{array}
$$

Figure 1. CFG for the stochastic π-calculus

A stochastic pi-calculus interpreter is implemented in Prolog. The interpreter is based on a specification of a stochastic pi-calculus abstract machine, given in (Phillips & Cardelli, 2005). Speed is essential in any such interpreter, given the many thousands of expressions to be executed during a GP run.

To evaluate the fitness of a candidate process, the GP system generates the stochastic pi-calculus expression from the CFG tree, and then runs the expression with the pi-calculus interpreter. Time-series plots are generated by the interpreter, which represent the behaviour of the process. There will be one plot per channel of interest. Each plot is then fitted to the target mean plot, using a sum-of-errors fit. Processes which do not generate enough output, perhaps due to deadlock, are penalized. The sum-of-errors score for each channel is tallied, resulting in an overall score for

the process. The lower the score, the better; a zero score is a perfect fit. Of course, stochastic pi-calculus expressions are probabilistic, and repeated executions often result in varying output. To account for this, the GP system will perform this scoring 3 separate times per expression, and find the mean score.

Other GP parameters are not included here, and most are typical to other GP applications in the literature. One unique aspect of this application of GP is the implementation of a few mutation operators that attempt to reduce the occurrence of expression deadlock (non-activity). Also, periodic Lamarckian evolution (local search) was used to boost the fitness of expressions.

The following stochastic pi-calculus process is included as an example with the SPIM system (Phillips, 2007), and models an ethylene chemical reaction:

$$!proc_0 : (ay : \overline{proc_1} : 0 + ar : \overline{proc_1} : 0)\ \#$$
$$!proc_1 : (\overline{ar} : \overline{ep} : 0 + \overline{pr} : 0 + pr : 0)\ \#$$
$$4@(\overline{ay} : 0)\ \#$$
$$200@(\overline{proc_0} : 0)$$

Each channel is assigned a particular rate value, which determines its level of activity during interpretation. The expression was given to a stochastic pi-calculus interpreter, and time-series plots were generated for 4 channels (ar, pr, ey, ay). Because this expression is probabilistic, repeated interpretations result in different output plots. Therefore, ten separate interpretations were done, and the plots of the 4 channels were averaged, resulting in a set of mean target plots.

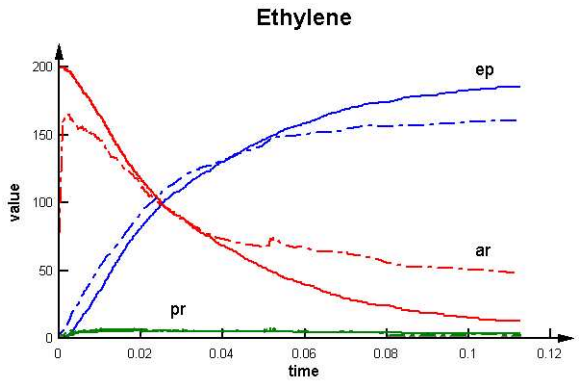

Figure 2. Ethylene: target (solid) and avg solution (dash)

Twenty runs were performed, and example plots from the 20 solutions were averaged. The plots in Figure 2 plot these averaged results (dashed) against the target plots (solid), for 3 of the target channels. One evolved solution is:

$$!proc_0 : (\overline{ay} : ep : \overline{proc_0} : \overline{pr} : \overline{proc_0} : 0 + ar : ep : 0)\ \#$$
$$!proc_1 : (ar : ep : \overline{ar} : \overline{proc_0} : 0 + ay : \overline{proc_0} : 0)\ \#$$
$$39@(\overline{proc_1} : \overline{proc_0} : 0)\ \#$$
$$5@(\overline{pr} : 0)\ \#$$
$$72@(\overline{proc_1} : 0)\ \#$$
$$27@(\overline{proc_1} : \overline{proc_1} : 0)$$

To summarize the results, it is promising that processes were synthesized that mirrored the target behaviour. However, the degree of fit leaves room for improvement. Furthermore, after examining the solution expressions, there is not a syntactic nor behavioural equivalence. Syntactic equivalence is unrealistic, unless an extremely constrained grammar were to be used. Behavioural equivalence would be sufficient. However, given the challenging nature of the stochastic environment, the most realistic expectation is to obtain acceptable equivalence to some statistically reasonable degree. Some statistical analyses showed that the generated plots are beyond the normal range expected with the solution expression. Reasons for this discrepancy are discussed in the next section.

4. Challenges and future work

Initial experiments have evolved a number of monotonic processes, using up to 4 channels for training. This research is currently being extended in a number of ways, in order to improve the quality of solutions obtained, and to generalize the scope and scale of process behaviours to be considered.

The first issue is that of adequacy of training methodology. A key reason why the experiment shown in Section 3 did not evolve a solution with a better behavioural fit to the target, was that the target behaviour was *underspecified*. There is not enough information in one set of average plots to adequately specify the underlying process mechanism. The solution to this is to borrow ideas from work in synthesizing genetic regulatory networks (Speith et al., 2004). Gene knockouts and over-expression can be used to refine the specification of the target system. Such strategies are especially necessary with laboratory data, as the results generated from genetic micro-arrays are highly underspecified.

Another issue to address is the representation of process behaviours themselves. The use of raw time-series plots to denote process behaviour is only feasible for certain constrained classes of dynamic systems, for example, monotonic processes. Processes exhibiting oscillations and noise are not effectively classifiable with simple error fitting of raw output data.

Work is commencing on improving the efficacy of genetic programming, with respect to stochastic process evolution. Improved internal representations of stochastic processes are required. Higher level modularization of stochastic pi-calculus expressions is necessary, if more sophisticated and scaled-up process behaviours are to be realized. Many oscillating behaviours require channel passing within stochastic pi-calculus expressions. Channel passing is a subtle but powerful construction within the algebra, and as implemented thus far, is too sensitive to be effectively evolved with genetic programming. It is usually the case that channel passing always disappears during runs, due to its high susceptibility towards creating deadlocking processes. Higher level expression modules that cannot be dismantled by reproduction operations, will help solve this problem.

The long-term goal of this research is the automatic synthesis of processes for biological networks, for example, gene regulatory systems. Once the above issues are addressed, it should be possible to use genetic programming as a means to evolve process networks that can be used to help ascertain the functional mechanisms of biological and chemical systems. Related work given in Section 2 has been very successful in ascertaining bio-chemical networks, albeit in a deterministic setting. The consideration of probabilistic processes is considerably more challenging than deterministic processes, since both the target behaviour and candidate solutions are nondeterministic and noise-ridden. On the other hand, a probabilistic model expands the scope of phenomena that can be considered. More realistic process models will be possible.

Acknowledgements

This research is funded by NSERC Operating Grant 138467.

References

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

Hoare, C. A. R. (1985). *Communicating Sequential Processes*. Prentice–Hall.

Kitagawa, J., & Iba, H. (2003). Identifying Metabolic Pathways and Gene Regulation Networks with Evolutionary Algorithms. In G. Fogel and D. Corne (Eds.), *Evolutionary computation in bioinformatics*, 255–278. Morgan Kaufmann.

Koza, J., Keane, M., Streeter, M., Mydlowec, W., Yu, J., & Lanza, G. (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

Milner, R. (1989). *Communication and Concurrency*. Prentice Hall.

Milner, R. (1999). *Communicating and Mobile Systems: the Pi-calculus*. Cambridge University Press.

Phillips, A. (2007). The stochastic pi machine (spim). URL: http://research.microsoft.com/ aphillip/spim/. Last accessed March 1, 2007.

Phillips, A., & Cardelli, L. (2005). A Correct Abstract Machine for the Stochastic Pi-calculus. *Proc. Bioconcur'04*. London, UK.

Phillips, A., Cardelli, L., & Castagna, G. (2006). A Graphical Representation for Biological Processes in the Stochastic pi-calculus. *Trans. in Comp. Sys. Bio (TCSB)*, *4230*, 123–152.

Priami, C. (1995). Stochastic pi-Calculus. *The Computer Journal*, *38*, 579–589.

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

Ross, B. (1998a). Pairwise Sequence Comparison and the Genetic Programming of Iterative Concurrent Programs. *Proc. Genetic Programming 1998* (pp. 338–343). Morgan Kaufmann.

Ross, B. (1998b). The Evolution of Concurrent Programs. *Applied Intelligence*, *8*, 21–32.

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

Ross, B. (2007). Using Genetic Programming to Synthesize Monotonic Stochastic Processes. *Proceedings CI-2007*. Banff, AB, Canada.

Speith, C., Streichert, F., Speer, N., & Zell, A. (2004). Iteratively inferring gene regulatory networks with virtual knockout experiments. *Proc. 2nd European Workshop on Evolutionary Bioinformatics* (pp. 102–112). Springer. LNCS 3005.