Production System Identification with Genetic Programming

Peter Denno[a,1], Charles Dickerson[b] and Jenny Harding[b]
[a]National Institute of Standards and Technology, US
[b]Loughborough University

Abstract. Modern system-identification methodologies use artificial neural nets, integer linear programming, genetic algorithms, and swarm intelligence to discover system models. Pairing genetic programming, a variation of genetic algorithms, with Petri nets seems to offer an attractive, alternative means to discover system behaviour and structure. Yet to date, very little work has examined this pairing of technologies. Petri nets provide a grey-box model of the system, which is useful for verifying system behaviour and interpreting the meaning of operational data. Genetic programming promises a simple yet robust tool to search the space of candidate systems. Genetic programming is inherently highly parallel. This paper describes early experiences with genetic programming of Petri nets to discover the best interpretation of operational data. The systems studied are serial production lines with buffers.

Keywords. System identification, Petri nets, genetic programming, smart manufacturing

1. Introduction

The ability to generate models of manufacturing systems from data is becoming increasingly useful. In earlier generations of manufacturing systems, a model of the system as a discrete-event system could be developed through inspection of the system's controller software. But, the machine-learning technology that is playing an increasing role in manufacturing control today [1] does not provide a similar presentation of system structure and mechanism. For this reason, verifying system behaviour and safety are becoming more, not less, challenging. Because of this, systems identification methodologies that suggest system structure (grey box models) are becoming increasingly useful. [2]

This paper presents a mostly-automated methodology to produce Petri net (PN) models of systems from historical, operational data describing system inputs and outputs. PNs used in this capacity provide grey-box, system models of the system that suggest the system's structure. Such structurally accurate PN models traditionally have been used for several purposes including verifying system safety, assessing system performance, and detecting deadlocks. This paper describes a method to match differing interpretations of system characterization with the appropriate Petri net structure. For example, multiple interpretations of the notions of blocking and starvation are prevalent in productions-systems engineering. Each interpretation associates accurately with some

[1] Corresponding author

Petri net structure. The goal of this work is to determine which interpretation most closely matches the given operational data.

The paper describes preliminary work towards that goal. It uses a case study involving serial production lines. [3] The system-identification algorithm used in this study is provided with two types of inputs: product mixes and machine capacities. The algorithm then generates the corresponding outputs in the form of four, steady-state, performance measures: buffer occupancy, probability of blocking, starvation and throughput. The algorithm applies genetic programming (GP), an evolutionary programming technique, to generalized stochastic Petri nets (GSPNs) to discover a Petri net that best fits the system input-output relations.

The contribution of this paper is a report of early experience using GP with GSPNs. The authors are aware of only one paper of a similar nature [4] and the case study of that paper concerns biological processes, not manufacturing processes, and discrete Petri nets, not timed Petri nets.

Section 2 of the paper discusses related work and the fundamental concepts of Petri nets and genetic programming. Section 3 describes our methodology. Section 4 concludes the work.

2. Related Work

The semantics of GSPNs are described with the help of Figure 1, which depicts a GSPN modelling a system consisting of two machines and a buffer of capacity one between them. The figure, as is typical of Petri net notation, uses 1) solid bars to represent immediate transitions and 2) hollow bars to represent exponentially timed transitions [5] 3) hollow circles to represent places, which can be populated with tokens, and 4) solid circles to represent *m1-blocked*, *buffer* and *m2-busy*. States of the system are described by the quantities of tokens in places. Figure 1 depicts a GSPN describing a system consisting of two machines that process jobs using times from exponential distributions. Between the first machine and the second is a buffer with capacity for one job. In the net on the left side of the figure, both machines *m1* and *m2* are busy; on the right side, machine *m1* is blocked.

GSPNs allow two kinds of arcs between transitions and places, both used in the figure. Activating arcs have an arrow head. Inhibitor arcs have a hollow circle head. A multiplicity is associated with arcs. The execution semantics of GSPNs is as follows. A transition fires when 1) all the input places (places at the tail of an arc) have quantities of tokens equal to or greater than the multiplicity of the arc into the transition, and no input places have quantities of tokens equal to or exceeding the multiplicity of an inhibitor arc into the transition. When the transition fires, one or more tokens, equal to the multiplicity of the input activator arcs, are removed from the input places. Token are added to the output places based on the multiplicity of the output activator arcs. The multiplicity of all arcs in the figure is 1.

Petri nets are one among several representations applied in system identification. Fu and Li [2] survey modern methods of system identification including neural nets, fuzzy logic, genetic algorithms, and swarm intelligence. Though the authors do not specifically discuss genetic programming, many of their comments regarding genetic algorithms may also apply to genetic programming. With respect using genetic algorithms, they note two benefits: quick convergence and path independence; and, one potential drawback: premature convergence to local optima.

Nobile et al. [4] describe a methodology for evolving Petri nets for purposes such as system identification. Their test case is a metabolic process, not manufacturing. In their methodology, places and transitions are partitioned into visible and hidden subsets. Each visible place and transition is permanently associated with a domain quantity. Hidden transitions and places can be subject to removal by the evolutionary operations of crossover and mutation. Nobile et al. do not specifically address timed Petri nets, and therefore, do not suggest a means of setting transition rates. The use of GSPN in the present work necessitates a different set of genetic operators than those described by Nobile et al.

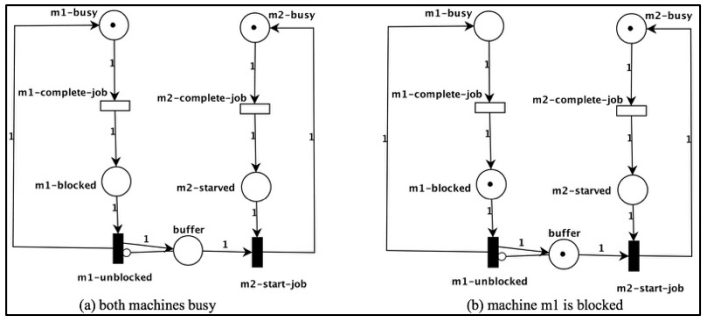

Figure 1. Two machines with a buffer for one part, block-after-service buffering convention.

Several works use Petri nets or genetic algorithms for systems identification. Tiacci [6] couples a discrete event simulator with a genetic algorithm approach to solve an assembly line balancing problem. Dotoli et al. [7] describe a method of system identification using Petri nets and integer linear programming. The work concerns the identification of discrete event systems as untimed Petri nets. In that work, the process is viewed as on-line, in the sense that it waits for events to occur and updates the Petri net after these occurrences. Basile et al. [8] also apply mixed integer linear programming. The Petri nets of this work are deterministic timed, not stochastic. The goal here is to provide a model that matches behaviour as discrete events. Rozinat [9] et al. uses process mining of event logs to create a simulation model of business processes as a coloured Petri Net. The work takes a broad perspective, involving identification of roles and merging of perspectives. Cabasino [10], a PhD thesis, describes an integer programming method of system identification and fault detection using unlabelled Petri nets. The goal of El Medhi et al. [11] is closest to the goal of the present paper. El Medhi describes an identification process for deterministic and stochastic (exponential) Petri net. Such nets can accurately represent queueing systems and machine reliability. The paper describes an integer linear programming method to synthesize a PN from measureable and non-measurable PN states.

3. Genetic Programming of Stochastic Petri Nets

Genetic algorithms are evolutionary algorithms. In a genetic algorithm, a population (sometimes called a generation) of individual solutions are scored for fitness relative to some objective. Those individuals scoring well are more likely to be promoted to the subsequent generation. Those solutions not selected are discarded. Among the promoted individuals, some are subject to modification by the application of two genetic operators:

mutation and crossover. The mutation operator modifies a single, selected individual; The crossover operator swaps elements of two individuals.

Genetic programming [12] is a form of genetic algorithm in which the individuals describe programs, typically represented as trees, and the operators are algebraic. The fitness function scores the ability of such a program to match the input/output relationships provided by training data. In applications where the best match can be represented as a mathematical function, $f: \mathcal{R} \rightarrow \mathcal{R}$, the problem closely resembles regression analysis. Indeed, the problem is a method of system identification called *symbolic regression* (SR). [13] Moreover, it can be viewed as a grey-box method if the discovered system provides a structure corresponding to a physical reality.

Our "programs" are Petri nets. [4] The elements of the programs are, of course, composed of the elements of whatever kind of PN has been selected. In this paper, we are using GSPNs because of their ability to model manufacturing systems. The manufacturing system we intend to examine in this paper concerns a 2-machine, serial production system with exponential service times and a one-place buffer between the machines. System identification in this context involves finding a Petri net that best matches the input/output relationships of the intended system. The chosen outputs are three steady-state properties: buffer occupancy, blocking of machine m1, and starvation of machine m2. These properties were used in the comparing the predicted values against the values of the intended system.

The design space of GSPNs has discrete and continuous dimensions. The discrete dimension concerns the PN's network topology. The continuous dimension concerns the real-valued rates of timed transitions and real-valued weights of immediate transitions. As is typical of generative design problems like SR, some strategy is needed to cope with the discrete/continuous dichotomy. In the design of other SR systems, that strategy uses 1) genetic programming to specify the discrete terms and 2) linear least-squares fitting to determine the optimal values for the continuous elements. Those values are the optimal coefficients of the terms in a linear function that minimizes prediction error. [13]

We have implemented a similar strategy in the design the SR system for our manufacturing example. There, the continuous terms, which are the service times and transition rates, are defined to be exponential with a mean of 1. The discrete term, the topology, evolves through the genetic program. The mutation operators used in that program are responsible for producing variations in the population.

Similar to the one in Nobile et al., [4] our GSPN design method makes a distinction between visible, and hidden, places and transitions. A *visible* place or transition is one for which system observations are provided. For this reason, the visible elements must appear in every PN. A *hidden* place or transition is one not directly associated with a system observation. Therefore, hidden elements need not be included in the PN. Given the visible/hidden dichotomy, the design of our GSPN focuses on visible elements only.

The mutation operators, as well as their arguments, in our GSPN are described in Table 1. In the application of the operators, a modified individual is promoted to the next generation only if it is found to be feasible. (Its reachability graph is calculated to determine this.) If the individual resulting from the mutation fails the feasibility test, a new set of random elements is selected and the individual is retested. If the selected mutation is not possible anywhere in the individual's structure, the individual is not promoted as a mutated form.

Table 1. Mutation operations used in the case study.

Operator	Action

add_place (t1, t2)	Two random transitions, t1, t2 (timed or immediate) are selected and a hidden place, p, and two arcs a1, and a2, are created. a1 is directed from t1 to p. a2 is directed from p to t2. The new arcs have multiplicity 1.
add_token(p)	A token is added to randomly selected place p (visible or hidden).
add_trans(p1, p2)	Two distinct places, p1 and p2 (visible or hidden) are chosen and a timed transition, t, and two arcs, a1 and a2 are created. a1 is directed from p1 to t. a2 is directed from t to p2. The new transition has rate=1.0.
add_arc(p,t)	A random place and transition (without regard to hidden/visible) are selected and randomly, either arc a is directed from p to t or t to p. The multiplicity of the arc is 1.
add_inhibitor(p,t)	A random place, p and transition, t is selected and an inhibitor arc, i, is created and directed from p to t.
remove_place(p)	A random hidden place is selected. It and all arcs to and from it are removed.
remove_token(p)	A place containing at least one token is randomly selected and its token count reduced by 1.
remove_trans(t)	A random hidden transition, t, is selected. It and all arcs to and from it are removed.
remove_arc(a)	A random arc, a, is removed.
remove_inhibitor(i)	A random inhibitor arc, i, is removed.
swap_places(v1, v2)	Two visible places are selected randomly. v1 is assigned the in-coming and out-going arcs of v2 and vice versa.

All selection actions used in the algorithm use tournament selection. Tournament selection involves choosing n individuals randomly from the population and then selecting the best among those n. Thus, when n is high, weak (low scoring) individuals are less likely to be selected. Selection is based on scoring individuals, which involves calculating the steady-state properties of the Petri net and comparing them to the target data. Since the individuals are stochastic Petri nets, this involves 1) forming the infinitesimal generator matrix **Q** for the Markov chain isomorphic to the Petri net and 2) solving the linear system:

$$\begin{aligned} \boldsymbol{\eta Q} &= \mathbf{0} \\ \boldsymbol{\eta 1}^T &= 1 \end{aligned} \tag{1}$$

where $\boldsymbol{\eta}$ is the steady-state distribution vector for the states of the Petri net. [5] For the case study, the dimension of **Q** was typically around 50 for most individuals but ranged as high as 700.

An initial population is created by producing a ring topology-PN called the Eden individual. The Eden individual contains all the visible places and transitions plus additional hidden places or transitions as needed to ensure that there are as many places as transitions. With equal numbers of places and transitions, the ring topology PN is produced by adding arcs between alternating places and transitions and closing the graph by connecting the last element used to the first. The Eden individual is repeatedly subjected to the genetic operators to create all of the individuals in the initial population.

In the case study, the visible places were labelled *m1-busy*, *m2-busy*, *buffer*, *m1-blocked,* and *m2-starved* as depicted in Figure 1. The system under study follows the usual conventions for analysis of serial production systems [3]: the first machine cannot starve and the last machine cannot block. The training data corresponded to machines with exponential service time. ($\lambda = 1.0$ for both machines.) The case study system uses block-after-service blocking convention.

Experience with the case study problem suggests that the algorithm essentially works, but that more effort will be needed to avoid convergence to local optima. As Table 2 shows, the algorithm tended to find a local optimum quickly and stick with it while the

median individual slowly improved. This was the case even when tournament selection pressure was low. The population size of the case study is 100 individuals. Bloat (the tendency in GP for individuals to become increasingly complex with little performance gain) was not a problem.

Table 2: Preliminary results from the case study

Generation	Error of Best Individual (total absolute error)	Error of Median Individual (total absolute error)
0	0.684	> 100.
1	0.684	1.80
2	0.494	1.33
8	0.470	0.855
9	0.333	0.720
10	0.333	0.512
12	0.333	0.467

4. Conclusion

The paper described early experience with a system-identification methodology that applies genetic programming to Petri nets representing discrete-event systems. The goal of the work is to discover Petri nets that best interpret the operational data. Genetic programming promises an effective method that easily parallelizes this problem. Early results suggest that the methodology is sound but that more work will be required to make it effective. We are currently undertaking that work.

References

[1] T. Wuest, D. Weimer, C. Irgens, and K.-D. Thoben, "Machine learning in manufacturing: advantages, challenges, and applications," *Prod. Manuf. Res.*, 2017.

[2] L. Fu and P. Li, "The Research Survey of System Identification Method," in *2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics*, 2013, pp. 397–401.

[3] J. Li and S. M. Meerkov, *Production System Engineering*. Springer Science+Business Media, 2009.

[4] M. S. Nobile, D. Besozzi, P. Cazzaniga, and G. Mauri, "The foundation of Evolutionary Petri nets," *CEUR Workshop Proc.*, vol. 988, pp. 60–74, 2013.

[5] M. A. Marsan, G. Balbo, G. Conte, and G. Franceschinis, "Modelling with generalised stochastic petri nets," *System*, p. 299, 1994.

[6] L. Tiacci, "Coupling a genetic algorithm approach and a discrete event simulator to design mixed-model un-paced assembly lines with parallel workstations and stochastic task times," *Int. J. Prod. Econ.*, vol. 159, pp. 319–333, 2015.

[7] M. Dotoli, M. P. Fanti, and A. M. Mangini, "Real Time Identification of Discrete Event Systems by Petri Nets," in *IFAC 2007*, 2007.

[8] F. Basile, S. Member, P. Chiacchio, S. Member, and J. Coppola, "Identification of Time Petri Net Models," *IEEE Trans. Syst. Man Cybern. Syst.*, pp. 1–15, 2016.

[9] A. Rozinat, R. S. Mans, M. Song, and W. M. P. van der Aalst, "Discovering simulation models," *Inf. Syst.*, vol. 34, no. 3, pp. 305–327, 2009.

[10] M. P. Cabasino, "Fault diagnosis and identification of discrete event systems using Petri nets," University of Cagliari, 2008.

[11] S. Ould El Mehdi, R. Bekrar, N. Messai, E. Leclercq, D. Lefebvre, and B. Riera, "Design and Identification of Stochastic and Deterministic Stochastic Petri Nets," *IEEE Trans. Syst. Man, Cybern. - Part A Syst. Humans*, vol. 42, no. 4, pp. 931–946, 2012.

[12] R. Poli, W. B. Langdon, N. F. Mcphee, and J. R. Koza, "A Field Guide to Genetic Programming," 2008.

[13] D. P. Searson, "GPTIPS 2: an open-source software platform for symbolic data mining," 2015.