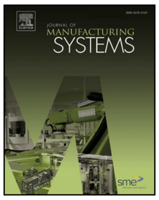
Journal of Manufacturing Systems xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Journal of Manufacturing Systems

journal homepage: www.elsevier.com/locate/jmansys



Technical Paper

Dynamic production system identification for smart manufacturing systems

Peter Denno[a,*], Charles Dickerson[b], Jennifer Anne Harding[b]

[a] *National Institute of Standards and Technology, Gaithersburg, MD, USA*
[b] *Loughborough University, Loughborough, UK*

ARTICLE INFO

Keywords:
System identification
Production systems
Genetic programming

ABSTRACT

This paper presents a methodology, called production system identification, to produce a model of a manufacturing system from logs of the system's operation. The model produced is intended to aid in making production scheduling decisions. Production system identification is similar to machine-learning methods of process mining in that they both use logs of operations. However, process mining falls short of addressing important requirements; process mining does not (1) account for infrequent exceptional events that may provide insight into system capabilities and reliability, (2) offer means to validate the model relative to an understanding of causes, and (3) updated the model as the situation on the production floor changes. The paper describes a genetic programming (GP) methodology that uses Petri nets, probabilistic neural nets, and a causal model of production system dynamics to address these shortcomings. A coloured Petri net formalism appropriate to GP is developed and used to interpret the log. Interpreted logs provide a relation between Petri net states and exceptional system states that can be learned by means of novel formulation of probabilistic neural nets (PNNs). A generalized stochastic Petri net and the PNNs are used to validate the GP-generated solutions. The methodology is evaluated with an example based on an automotive assembly system.

1. Introduction

Knowledge of process requirements, system capacities, and system reliability are the premises on which control policies are formulated. In dynamic manufacturing environments, engineering change to the product, the process, and the production equipment can cause these premises to be violated and thereby make control policies less effective. An accurate, up-to-date model of the production system is essential to production control, but a challenge to maintain.

Both the need for a production system model and the challenge of maintaining it are more intense in smart manufacturing settings. The need is more intense because a key goal of smart manufacturing is to automated decision making [1]. Decisions concerning sequencing [2], line balancing [3,4], and production system engineering [5] are sensitive to changes in process requirements, system structure, capacities, and reliability expressed in production system models. The challenge is more intense because smart manufacturing can make manufacturing more agile [1], and the changes brought on by increased agility must be reflected in the production system model. Change in process requirements is commonplace in manufacturing environments where products are evolving rapidly. Changes in system structure, capacities, and reliability are less common; but control policies are affected as much by changes in these dimensions as they are by changes in product and process.

Dynamic production system identification is a methodology that develops and updates a production system model that can provide information essential to performance analysis and control. The methodology (1) identifies a model that, like traditional statistical system identification [6] responds to stimulus accurately, (2) identifies system components, their properties, and interconnection, (3) identifies normative process for multiple job types, and (4) continually updates the model.

The production system model is a process model. Machine-learning methods of process mining typically develop such models using an analysis of frequently occurring events described in system logs. These methods fall short of addressing the challenge of dynamic production system identification in three important respects: (1) Rather than frequently occurring events, it is the infrequent, exceptional events that typically provide insight into system capacities and reliability. (2) Production system behaviour, especially machine blocking and starvation, are well-understood phenomena; an analysis of cause and effects could be used to guide search to an accurate system model. (3) Process mining lacks inherent means to update the model as the modelled system changes.

The production system model describes processes associated with International Society of Automation (ISA) Level 3 control problems [7].

* Corresponding author.
E-mail address: peter.denno@nist.gov (P. Denno).

https://doi.org/10.1016/j.jmsy.2018.04.006
Received 30 November 2017; Received in revised form 31 March 2018; Accepted 11 April 2018

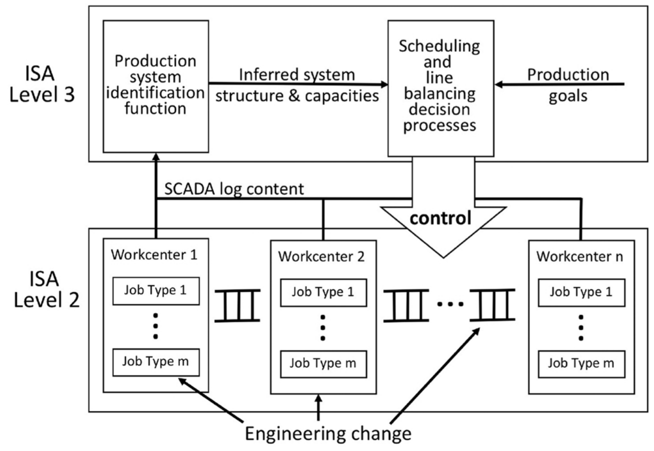


Fig. 1. Production system identification in context.

Our methodology infers the production-system structure and capacities specifically for use in line scheduling and balancing processes (see Fig. 1). In the methodology, genetic programming, default causal knowledge, and probabilistic classification of exceptional conditions are used to evolve a population of individuals each representing a candidate model. The fitness of an individual is assessed with respect to its ability to (1) reproduce the content of logs describing typical Supervisory Control and Data Acquisition (SCADA) events, (2) in comparative steady-state analyses, respond to perturbations in workstation capacity with plausible differences in buffer occupancy and state sojourn times, and (3) detect critical job-type distinctions (e.g. that one job type requires significantly more processing time at some workstation than does another job type).

The main contribution of this paper is a robust methodology for dynamic production system identification. The paper investigates the value of genetic programming (GP) of Petri nets (PNs) in meeting its goals. GP on PNs is intended to facilitate adaptation of the methodology to diverse production system architectures and logging scenarios. The paper provides novel methods to interpret logs, validate the model, and learn from exceptional events.

Section 2 of the paper describes related work. Section 3 presents a Petri net model, the Augmented Queueing Petri Net (AQPN) which provides the model of process used in GP evolution. Section 4 describes how exceptional conditions, causal validation, and model updating are handled. Section 5 describes a case study that uses the methodology. Section 6 concludes the paper with an assessment of the methodology's limitations and a discussion of future work.

2. Related work

Process mining [8,9], and advanced system identification methods [10,11] provide semi-automated means to produce process and system models for various purposes including process conformance (i.e., determining whether or not the actual process being practiced conforms to the normative process). Typically, these methods have the goal of capturing the most frequent process patterns and exhibiting robustness to noise [12].

van der Aalst et al. [13] describe a process mining algorithm known as the α-miner. The algorithm produces structured workflow nets (SWF-nets) from process logs. SWF-nets are untimed safe Petri nets constrained to avoid two forms of so-called “confusion” in the composed use of choice and synchronization in Petri nets.

Alves de Medieros [12] describes a genetic algorithm approach using SWF-nets to address some of the limitations of the α-miner. Specifically, it solves the choice/synchronisation confusion problem and addresses invisible and duplicate tasks. It is robust to noise by ignoring infrequent events.

Rozinat et al. [8] describe a methodology for constructing simulation models that involves four perspectives on process: control-flow, data, performance, and resource. The work uses coloured Petri nets. The simulation models produced do not make a distinction between normative and exceptional events.

Some relevant work associates more closely with system identification than process mining. Several of these, including [11,10,14] use integer linear programming (ILP). Ould El Mehdi et al. [11] uses ILP to produce deterministic and stochastic Petri net (DSPN) models of systems. The work is targeted to reliability analysis of repairable systems. DSPNs are of limited use in modelling production systems because an analytical solution of steady-state can only be had with DSPNs if no more than one deterministic transition is enabled in any marking [15].

Basile et al. [10] describes a mixed integer linear programming method of system identification that produces timed PNs. The underlying algorithm assumes a bijective relationship between event-log entries and PN transitions. The work does not use a coloured Petri net (CPN) model. Colours in CPNs can be used to represent differing job types, which is necessary in models of production lines.

Turner et al. [16] is the only work the authors are aware of that uses genetic programming for process mining. This short paper asserts that genetic programming provides greater flexibility in problem formulation and the possibility of mining complex and problematic event logs. The systems described do not use buffers nor does the methodology address exceptional conditions.

Compared to the work cited, our methodology emphasizes a means to establish a relationship between the information generated in production and the system's components. The identified model is not designed for use as a simulation directly but as a means to infer, organize, and update information needed when building simulations and decision support tools that need to be responsive to change.

3. Dynamic production system identification

The goal of any process modelling effort is to produce models fit for purpose [17]. Knowledge of system capacities is essential to the purpose of modelling production scheduling. For complex system engineering generally, and production system engineering particularly, capturing the most frequent process patterns will not be sufficient to create such a model. There are three interrelated reasons for this. First, the behaviour of complex systems under unforeseen circumstances cannot be predicted from the study of its response to seen circumstances. Hence models based only on frequent events (seen circumstances) are not in themselves very good simulations of the actual system. Second, a system response (e.g. blocking) can be a consequence of earlier interactions between the system and it environment. That environment might reflect exceptional circumstances. For example, while a machine is inoperative, work builds up at its input buffer. A model useful to scheduling must be capable of carrying this information forward to reflect a new state. The new state reflects exceptional circumstances and a capacity. Conversely, a model fit to data from only frequent and normative events would have no basis for doing this. Third, many analytical methods in production control require a specification that separates system description (e.g. capabilities, capacities, and system topology) from problem specification (e.g. demand, product mix). Unfortunately, state-of-the-art process-mining methods do not address these issues.

A sketch of the methodology is provided in Fig. 2. To test the methodology, a discrete event simulation system for mixed-model

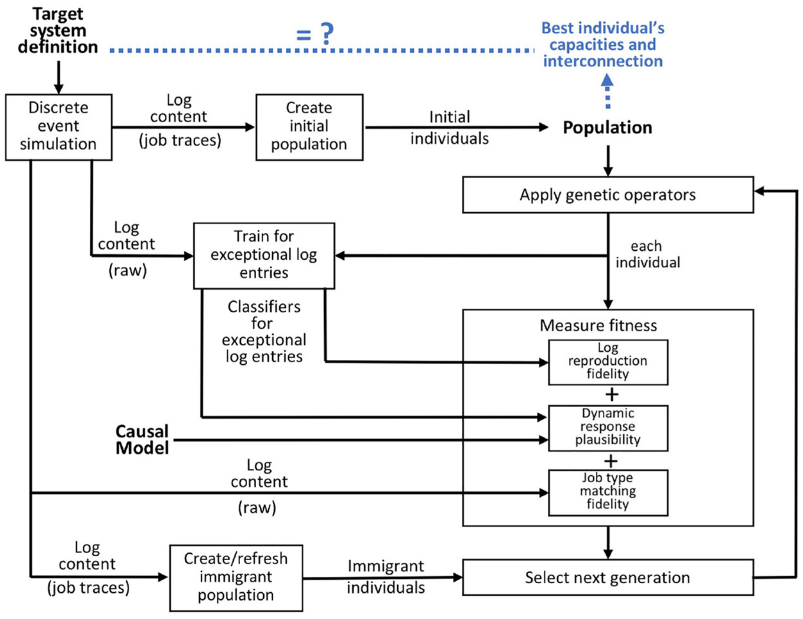

Target system definition
= ?
Best individual's capacities and interconnection
Discrete event simulation
Log content (job traces)
Create initial population
Initial individuals
Population
Apply genetic operators
Log content (raw)
Train for exceptional log entries
each individual
Classifiers for exceptional log entries
Measure fitness
Log reproduction fidelity
+
Dynamic response plausibility
+
Causal Model
Log content (raw)
Job type matching fidelity
Log content (job traces)
Create/refresh immigrant population
Immigrant individuals
Select next generation

Fig. 2. System identification methodology.

production, MJPdes [18], was developed to produce log data and performance parameters consistent with the behaviour of actual production lines. The log data is intended to resemble what can easily be provided by SCADA reporting. SCADA reporting represents activities at Level 2 of the ISA-95 hierarchy. In MJPdes, message types emitted include `job-enters-system`, `job-exits-system`, `job-starts-on-machine`, `job-moves-off-machine`, `machine-blocked`, `machine-unblocked`, `machine-starved`, and `machine-unstarved`. Associated with each of these is the time at which the event occurred. Associated with the `job-*` events are job identifier. Associated with `machine-*` events are equipment identifiers.

The remainder of Fig. 2 concerns the GP algorithm. A population of initial individuals (PN models) is created where each individual traces one job through the production system as indicated by the appearance of the job in the SCADA log. The PN representing an initial individual has one transition for each message in the log emitted about the subject job. PN's are directed bipartite graphs, so between each such transition is a place. In an initial individual, a single arc connects the places and transitions in sequence. The last place is connected to the first transition. Initial individuals thus have a simple ring topology. Similar to Nobile et al. [19], a distinction is made between *visible* transitions, which correspond to log messages, and *invisible* transitions, which do not. Initial individuals have no invisible transitions. As a consequence of their simple structure, initial individuals are not capable of expressing buffering constraints.

Successful use of GP requires that genetic operators exhibit *locality* [20]. Locality [21] is the property that small modifications to the individual's structure (i.e. genotype, the individual's PN in our case) result in proportionally small differences in the expression of behaviour (i.e. phenotype, production of log message in our case). Without locality, successive refinement is not possible and search degrades to a random generate-and-test process. Genetic operators for PNs must be carefully designed to ensure locality. Our earlier work [22] led to the conclusions that genetic operators on PNs (1) should only operate on structure within small neighborhoods, that is, places and transitions that are only a few edge hops from each other, and (2) should not be allowed to disturb the precedence order of the operations of jobs.

3.1. Augmented queueing Petri nets

Petri nets is a family of graphical formalisms to represent process causation, concurrency, choice, and synchronisation [23]. For many applications, PNs have been superseded by domain-specific simulation languages [24] and process ontologies [25]. These other representations lack characteristics important to performing the GP-based automated design tasks that are key to this work. For example, simulation languages typically do not define process formally (i.e. such that deductive reasoning can be used to ascertain the truth of statements). In simulation languages, distinctions such as those between block-before-service (BBS) and block-after-service (BAS) behaviour typically must be encoded in software. Consequently, it is not easy to access and reason about behaviors in a simulation language (see Fig. 3 for how BBS and BAS behavior is expressed in PNs). Axiomatic process ontologies do not suffer this weakness; however, typically, theorem provers are needed to infer the effects of an action. Theorem provers are inefficient for the purposes of this work. In contrast, the immediate effect of an event in a PN is limited to changes in the token counts in places directly connected to the transition representing the event. Such locality of effect is important to the log interpretation process described in Section 3.2. Finally, category theory-based modelling of PNs [26,27] may make possible functorial mappings of PNs into other forms such as analytical codes for production scheduling.

A new Petri net formalism called *Augmented Queueing Petri Nets* (AQPN) was developed for this work. In order to model the details of each individual job's movement through the production system, the formalism combines capabilities of generalised stochastic Petri nets (GSPNs), coloured Petri nets (CPNs), and Queueing Petri nets (QPNs). GSPNs [28] provide timed and immediate transitions and inhibitor arcs that are used in this work to bound buffer size. k-bounded GSPNs can be reduced to PNs that are isomorphic to continuous-time Markov chains

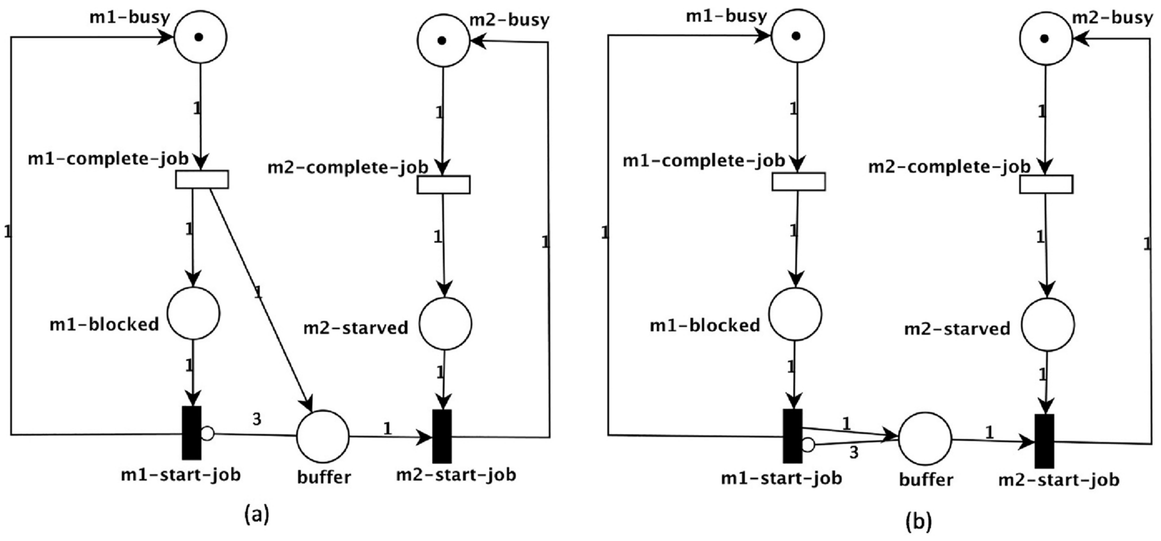


Fig. 3. 2-Machine system with a buffer of size of 3: (a) block-before-service and (b) block-after-service.

(CTMCs) for calculation of steady-state properties. CPNs [29] provide the ability, in this work, to distinguish job types and route differing job types differently. QPNs [30] provide a queueing discipline on the release of tokens on arc outbound from places. In this work, only first-in-first-out (FIFO) queues are supported.

To represent jobs entering and exiting the system, and to direct jobs along certain pathways when transitions have multiple outbound arcs, AQPNs use a simple priority scheme to allocate tokens to outbound arcs. A unique number in the set $\{1, \ldots, n\}$ called a *priority* is associated with each of the n outbound arcs from a transition. Tokens have identifiers in the set $\{1, \ldots, m\}$ representing the m jobs that have been introduced into the system during its operation. When a transition fires, tokens are removed from each incoming place according to the multiplicity of the incoming arcs and FIFO queueing. When the number of tokens entering a transition is equal to the number exiting the transition, no new tokens are created and none are destroyed. Tokens are assigned to arcs such that the token requirements (multiplicity) of the highest priority arc (lowest priority number) are satisfied first using the newest tokens (tokens with lowest job identifier). This process is repeated for each arc in priority order.

The number of tokens entering a transition may be different than the number exiting it. When the number of tokens entering a transition is less than the number exiting it, new tokens, representing new jobs, are created with new identifiers in serial order. As described above, the newest tokens are assigned to the highest priority arcs. New tokens are not assigned a colour in this process. Tokens are given a colour in log interpretation. The colour assigned is one consistent with the log message introducing the job. When the number of tokens entering the transition is more than the number exiting it, the oldest tokens are destroyed.

The assignment of priorities to arcs can be permutated with a GP mutation operator.

AQPNs are not necessarily k-bounded [28]. Synchronisation and choice can be combined in arbitrary ways, i.e. non-free-choice nets are permitted.

3.2. Interpretation of log content

Messages in the log can be classified as either ordinary or exceptional. Ordinary messages correspond to firings of PN's transitions, and in individuals that correctly model the log, a legal sequence of states of the individual's PN is consistent with the ordering of messages in log. Individuals may classify the message types differently. An individual's exceptional messages are those that are not represented by transitions. Exceptional messages may nevertheless relate to states of the PN, and where this is the case, knowledge of the relationship between the PN state and the message can sometimes reveal important information about the structure of the system. For example, knowledge of the PN state (the PN's marking) at the occurrence of an (exceptional) machine blocking message suggests the size of machine's downstream buffer. The method by which this is analysed is described in Section 4.1.

The fitness of an individual is expressed as a score based on its ability to model the log, and it validity relative to a causal model. Causal validity is discussed in Section 4.2. Regarding modelling the log, an individual's ability is a combination of the deterministic behaviour of its PN with respect to ordinary messages and the probabilistic behaviour of an associated probabilistic neural net with respect to exceptional messages.

The individual's deterministic behaviour is interpreted in terms of an abbreviated (k-bounded) version of its reachability graph. The state of a PN having n places, $\{P_1, \ldots, P_i, \ldots, P_n\}$, is completely specified by a vector M (called a *marking*), $M = [m_1, \ldots, m_i, \ldots, m_n]$ where m_i is the quantity of tokens in the ith place. The set of markings reachable from some initial marking M_0 is called the PN's *reachability set* [28]. The PN's *reachability graph* is a directed graph where nodes are elements of its reachability set and edges are firings of transitions relating a state to other states reachable from that state. A PN's reachability set may be unbounded. In the methodology, the interpretation of the log is the discovery of a relationship between ordinary messages and nodes and edges of the reachability graph. Where an interpretation is found, it provides constraints on the graph. For example, on places in the PN that are interpreted as buffers, interpretation provides lower bounds on the buffer's size. Buffer size can be expressed as inhibitor arcs in nets enhanced with GSPN modeling features. Inhibitor arcs on buffers may express a *block after service* or *block before service* discipline depending on the target transition chosen (see Fig. 3).

The interpretation process is described in Algorithm 1. In the algorithm, *setStateOfMachines* defines an initial marking where machines have some state. *kBoundGraph* defines a reachability graph where places believed to be buffers are limited to *kBound* tokens. *syncToLog* is a tree search that identifies the set of marking that are potentially consistent with the first line of the log. *fireable*? returns *true* if a transition is fireable from *pnState* to produce the message at *logLine* of the log. *fired* returns the state corresponding to firing that transition.

Algorithm 1. Interpret log.

```
$logSize \longleftarrow sizeofLog(log)$
$interp \longleftarrow \emptyset$
$kBound \longleftarrow 2$
$initialMarking \longleftarrow setStateOfMachines(PN)$
while $kBound < kBoundMax$ AND $interp = \emptyset$ do
    $rgraph \longleftarrow kBoundGraph(PN, initialMarking, kBound)$
    $startingLinks \longleftarrow syncToLog(rgraph, initialMarking, log)$
    while $startingLinks \neq \emptyset$ do
        $pnState \longleftarrow takeFrom!(startingLinks)$
        $logLine \longleftarrow 1$
        $progress? \longleftarrow true$
        while $logLine < logSize$ AND $progress? = true$ do
            if $fireable?(rgraph, pnState, log[logLine])$ then
                $pnState \longleftarrow fired(rgraph, pnState, log[logLine])$
                $interp \longleftarrow interp + pnState$
                $logLine++$
            else
                $progress? \longleftarrow false$
                $interp \longleftarrow \emptyset$
            end if
        end while
        $pnState \longleftarrow takeFrom!(startingLinks)$
    end while
    $kBound++$
end while
return $interp$
```

4. Addressing the 3 limitations

4.1. Exceptional messages

A probabilistic neural net (PNN) [31] is computed for each individual to distinguish exceptional messages (messages not modelled through execution of the individual's PN as described in the interpretation process above) from ordinary messages. The fidelity and certainty with which the PNN classifies exceptional messages provides the component of fitness associated with the individual's probabilistic behaviour.

A PNN is composed of three layers (see Fig. 4). The input layer has one node for each feature (e.g. each component of a marking). The hidden layer contains a node for each labelled training instance. The hidden layer nodes can be viewed as being grouped by the k classes of the training instances. Each node in the hidden layer is a Gaussian probability density function centred on a training instance, $\mathbf{m_i}$, as shown in Eq. (1)

$$g(\mathbf{m}, \mathbf{m_i}, \sigma_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-D(\mathbf{m}, \mathbf{m_i})}{2\sigma_i^2}\right) \tag{1}$$

where

- $\mathbf{m}$ is the input value, a marking,
- $\mathbf{m}_i$ is the ith training instance, a marking,
- σ_i is a smoothing factor, and
- $D(\mathbf{m}, \mathbf{m}_i)$ is a function determining the distance between the input value and the training instance.

The hidden layer serves to estimate the probability density function (PDF) of each class using the Parzen window technique [31] (see Eq. (2)). In this technique, the sum of PDFs from the training instances defines a PDF for each class. The value of σ determines the width of the window.

$$f_{\text{class}}(\mathbf{m}) = \frac{1}{\sqrt{2\pi\sigma^2}} \frac{1}{j} \sum_{i=1,j} \exp\left(\frac{-D(\mathbf{m}, \mathbf{m_i})}{2\sigma^2}\right) \tag{2}$$

In many applications, the square of the Euclidean distance is used as the distance function D [32]. However, in application to PN states, this is not an effective measure because the Euclidean distance between markings has an imprecise relation to transitions of system state represented by the markings. The reachability set of a PN does not form a metric space. Effective measures on which to base the distance function account for the probability between transitions. One such distance function, based on path probability, is described by Eq. (3)

$$D(\mathbf{m}_1, \mathbf{m}_2) = \sqrt{(\hat{\mathbf{m}}_1 - \hat{\mathbf{m}}_2)^2} \sum_{i \in \text{MinPath}} 1/p_i \tag{3}$$

where

- *MinPath* is the set of directed links from the minimum cost path connecting $\mathbf{m}_1$ to $\mathbf{m}_2$ in the PN's reachability graph,
- i references a link in this path,
- p_i is the probability of traversing that link, and,
- $\hat{\mathbf{m}}_1$ and $\hat{\mathbf{m}}_1$ are, respectively $\mathbf{m}_1$ and $\mathbf{m}_1$ component-wise normalized.

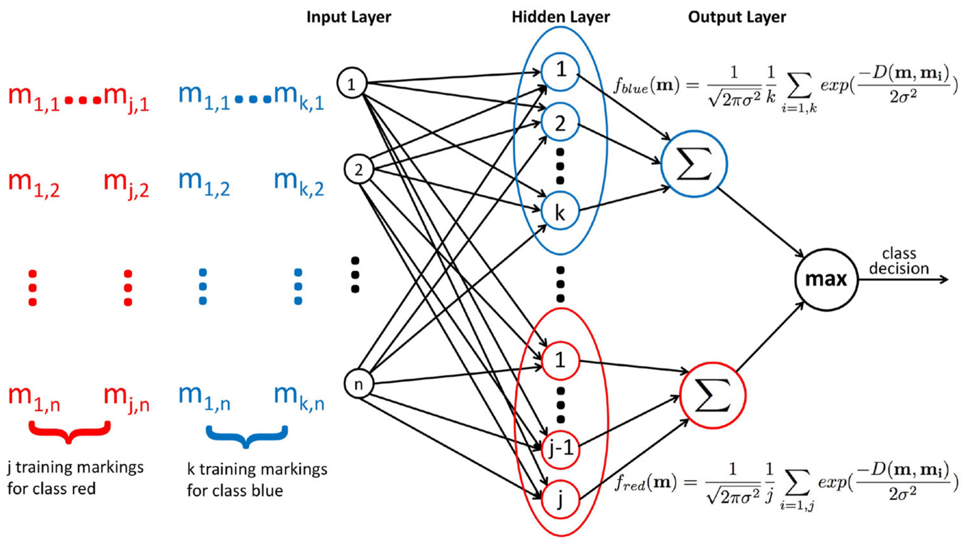

Input Layer
Hidden Layer
Output Layer
$f_{blue}(\mathbf{m}) = \frac{1}{\sqrt{2\pi\sigma^2}} \frac{1}{k} \sum_{i=1,k} exp(\frac{-D(\mathbf{m}, \mathbf{m_i})}{2\sigma^2})$
$\mathrm{m}_{1,1} \bullet\bullet\bullet \mathrm{m}_{j,1}$
$\mathrm{m}_{1,1} \bullet\bullet\bullet \mathrm{m}_{k,1}$
$\mathrm{m}_{1,2}$ $\mathrm{m}_{j,2}$ $\mathrm{m}_{1,2}$ $\mathrm{m}_{k,2}$
$\mathrm{m}_{1,n}$ $\mathrm{m}_{j,n}$ $\mathrm{m}_{1,n}$ $\mathrm{m}_{k,n}$
j training markings for class red
k training markings for class blue
1
2
n
k
j-1
j
max
class decision
$f_{red}(\mathbf{m}) = \frac{1}{\sqrt{2\pi\sigma^2}} \frac{1}{j} \sum_{i=1,j} exp(\frac{-D(\mathbf{m}, \mathbf{m_i})}{2\sigma^2})$

Fig. 4. Probabilistic neural net.

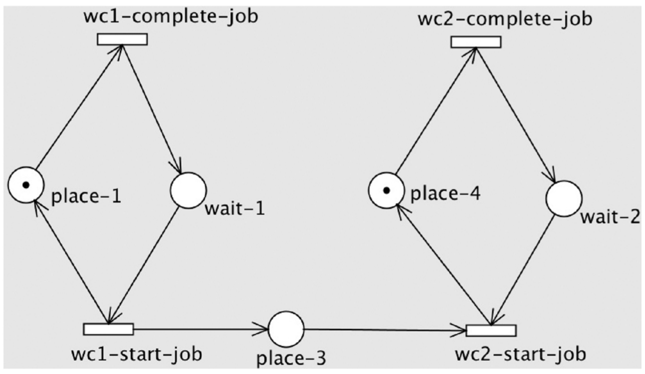


Fig. 5. An individual modelling a 2-machine system. Note that the buffer represented by *place-3* is not constrained in size. An inhibitor transition with multiplicity k from *place-3* to *wc1-start-job* would constraint the buffer size to k. The size can be inferred from analysis of exceptional messages.

Component-wise normalisation entails that, for all $i \in \{1, \ldots, n\}$ of a PN having n places, if, x_i, the ith component of a marking $x = [x_1, x_2, \ldots, x_i, \ldots, x_n]$ has a maximum value of k in the PN's reachability graph, then $\hat{x}_i = \frac{1}{k}x_i$ where $\hat{x} = [\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_i, \ldots, \hat{x}_n]$.

MinPath is found using Dijkstra’s algorithm [33], a search that finds the minimum cost path among paths between given nodes in a directed graph. In the algorithm, cost is calculated as the sum of the cost of the edges traversed. In the path-probability algorithm, the cost of traversing an edge is calculated from an interpretation of log content as $1/p$ where p is the probability of traversing the referenced states.

The relative performance of the Euclidean, path probability, and a third strategy similar to path probability but only counting edge hops (see Eq (4)), is illustrated in an example. In the example, message types from a log of 3000 messages produced from the operation of a simple 2-machine system depicted in Fig. 5 are distributed as defined in Table 1. The column *system state* represents token quantities in places. The ordering of these features shown is *wait-2*, *place-1*, *wait-1*, *place-4*, *place-3* where these symbols refer to the places shown of Fig. 5. The log reflects sparse occurrences of blocking and starvation, hence individuals are unlikely to model these messages with transitions. The table values are generated by the interpretation process described in Section 3.2. Markings associated with messages modeled as transitions (e.g. “Workcenter 1 finishes job x”) are labelled *ordinary*. Exceptional messages (i.e., *wc1-blocked*, *wc1-unblocked*, *wc2-starved*, and *wc2-unstarved*)

Table 1
Message occurrences in the example problem.

Message type	System state	Number of occurrences
m2-starved	[1 1 0 0 0]	10
	[1 1 0 0 1]	4
m2-unstarved	[0 1 0 1 0]	10
	[0 1 0 1 1]	4
ordinary	[0 0 1 1 1]	243
	[0 1 0 1 0]	155
	[1 1 0 0 2]	247
	[0 1 0 1 3]	326
	[1 1 0 0 3]	325
	[0 1 0 1 1]	393
	[1 1 0 0 0]	10
	[1 0 1 0 1]	4
	[0 0 1 1 0]	145
	[0 0 1 1 2]	326
	[1 1 0 0 1]	160
	[0 1 0 1 2]	568
	[1 0 1 0 0]	10
m1-blocked	[0 1 0 1 3]	30
m1-unblocked	[0 0 1 1 2]	30

are labelled with the specific message type and are associated with the system state at the time the message occurred, according to the interpretation.

$$D(\mathbf{m}_1, \mathbf{m}_2) = \sqrt{(\hat{\mathbf{m}}_1 - \hat{\mathbf{m}}_2)^2} \sum_{i \in \mathrm{MinPath}} 1 \tag{4}$$

As Table 2 suggests, a PNN where the distance function is purely Euclidean performs poorly on the example problem, producing incorrect results (column *Cor?*) in three states. Certainty (column *Certain.*) is calculated as a measure of the best score relative to the second best score, where score is the value of $f_{class}(m)$ for each class. If $class_1$ scores highest among all classes and $class_2$ second highest, then certainty is defined:

$$\mathrm{certainty}(\mathrm{class}_1, \boldsymbol{m}) = \frac{f_{\mathrm{class}_1}(\boldsymbol{m}) - f_{\mathrm{class}_2}(\boldsymbol{m})}{f_{\mathrm{class}_1}(\boldsymbol{m})} \tag{5}$$

Certainty is zero at the decision boundary between two class.

The network hop algorithm of Eq. (4) performs slightly better on the example than the purely Euclidean algorithm and is less certain in cases where it classifies incorrectly. The path probability algorithm of Eq. (3) classifies each marking correctly. Its worst certainty is 0.727. A value of 0.35 is used for σ for all calculations in the example.

4.2. Causal analysis

The role of causal analysis in the methodology is to validate models. As Ljung points out, validation in system identification is “the process of ensuring that the model is useful not only for the estimation data, but also for other data sets of interest” [6]. The methodology, as described so far, does not necessarily produce models that meet this requirement. For example, in a system where a particular workstation is chronically blocking, the GP algorithm is likely to promote individuals that use a PN transition to model the blocking message.[1] Such a model is not robust under circumstances where the cause of the blockage in the real system is removed (e.g. when the line is re-balanced). The GP algorithm scores such models (individuals) less fit than individuals that behave consistent with expert understanding of the domain. Fig. 6 depicts a model of causes and effects in asynchronous serial lines with buffering.

Models can be tested through two pathways: (1) Steady-state properties of an individual's PN can be computed at various workstation production rates and evaluated for response consistent with the qualitative causal model, and (2) using the same computational model as (1), the steady-state results can be applied to the individual's PNN and evaluated with respect to expectations for blocking and starvation messages.

Steady-state properties are calculated using the infinitesimal generator matrix of the Markov chain isomorphic to the PN [28]. The steady-state values provided by this method are adequate for the qualitative causal analysis performed but are not an accurate measure of actual buffer occupancies and state probabilities because the model assumes exponentially distributed transition rates whereas the actual production system is likely to exhibit deterministic processing times and unreliable machines.

The infinitesimal generator matrix $\mathbf{Q}$ is derived from the PN reachability graph and the rates between transitions. Detailed discussion of the infinitesimal generator can be found in textbooks on Markov chains or GSPNs [28]. The value of the element of matrix at row i and column j, q_{ij} is given by:

$$q_{ij} = \begin{cases} \sum_{T_k \in E_j(M_i)} r_k & i \neq j \\ -q_i, & i = j \end{cases} \tag{6}$$

[1] This is analogous to designing an automobile to get a flat tire every 100 km because that is what has happened to the automobiles that you observed. Intention cannot easily be separated from adventitious association in system identification.

Table 2
Relative performance of the Euclidean, network hop, and path probability strategies.

State	Euclidean distance			Network hop algorithm			Path probability algorithm		
	Inferred	Cor?	Certain.	Inferred	Cor?	Certain.			
[0 0 11 1]	ordinary	Yes	0.999	ordinary	Yes	0.999	ordinary	Yes	0.999
[0 1 01 0]	ordinary	Yes	0.193	ordinary	Yes	0.999	ordinary	Yes	0.969
[1 1 00 2]	m2-unstarve	No	0.708	m2-unstarve	No	0.551	ordinary	Yes	0.999
[0 1 01 3]	m1-block	Yes	0.365	m1-block	Yes	0.743	m1-block	Yes	0.898
[1 1 00 3]	m2-unstarve	No	0.050	ordinary	Yes	0.998	ordinary	Yes	0.969
[0 1 01 1]	m1-unblock	No	0.503	m1-unblock	No	0.199	ordinary	Yes	0.935
[1 1 00 0]	m2-starve	Yes	0.365	m2-starve	Yes	0.734	m2-starve	Yes	0.988
[0 0 11 0]	ordinary	Yes	0.999	ordinary	Yes	0.999	ordinary	Yes	0.999
[0 0 11 2]	ordinary	Yes	0.998	ordinary	Yes	0.999	ordinary	Yes	0.999
[1 1 00 1]	m2-unstarve	Yes	0.365	m2-unstarve	Yes	0.743	m2-unstarve	Yes	0.915
[0 1 01 2]	m1-unblock	Yes	0.365	m1-unblock	Yes	0.743	m1-unblock	Yes	0.823
[1 0 10 0]	ordinary	Yes	0.975	ordinary	Yes	0.999	ordinary	Yes	0.727

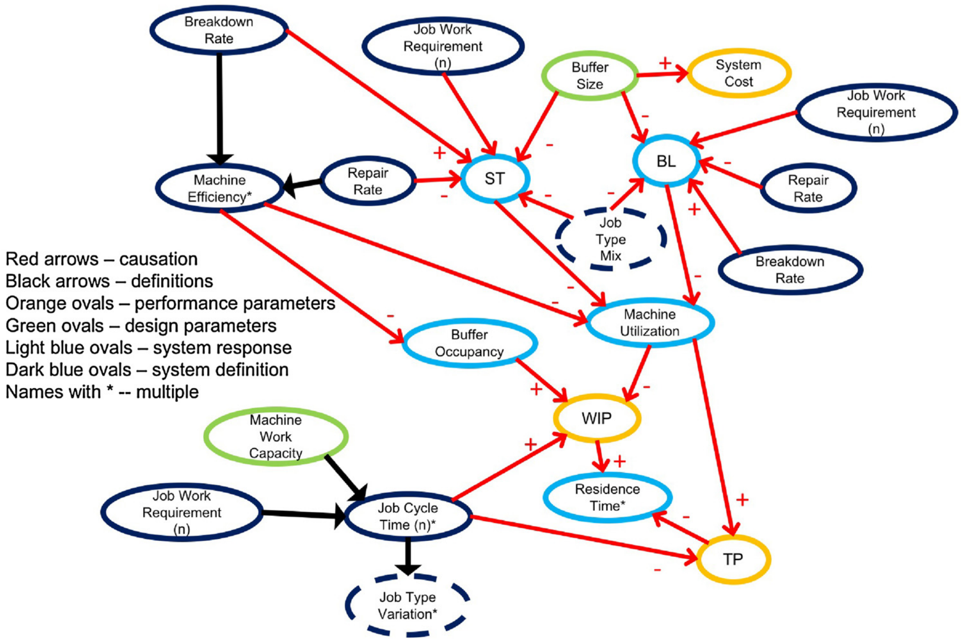


Fig. 6. Causal model of asynchronous serial lines with buffering. + on an arrow denotes that an increase in the value at the tail of the arrow causes an increase in the value at the head of the arrow.

where

$$q_i = \sum_{T_k \in E_j(M_i)} r_k \tag{7}$$

In Eq. (6), T_k is a transition with firing rate r_k. $E_j(M_i)$ is the set of transitions enabled in marking M_i that, when fired, place the PN in marking j. Using similar notation in Eq. (7), each diagonal element of the matrix is the negative of the sum of the off-diagonal terms of that row.

Q is used in the system provided by Eqs. (8) and (9). Solving this system for $\boldsymbol{\eta}$ provides the steady-state token quantities at each state, from which the token quantities at each place can be calculated.

$$\boldsymbol{\eta Q} = \mathbf{0} \tag{8}$$

$$\boldsymbol{\eta 1}^T = 1 \tag{9}$$

To perform these calculations efficiently, the software producing the infinitesimal generator was designed to accept a table of transition rates and parametrically produce the corresponding matrix.

Fig. 7 depicts a simple 3-machine system used in an example causal analysis. Table 3 provides steady-state values for occupancy of states and buffers for two versions of the system: a baseline, and a system where the efficiency of machine $M2$ is increased by 20 percent. As the table shows, states *wc1-blocked* and *wc1-busy*, in the example exhaustively cover the possible states for machine $M1$ (they sum to 1.0). Consistent with the causal model, *buffer-1* is less occupied when $M2$ is more efficient. Several other observations consistent with the causal model are described in the *Comments* column.

4.3. Model updating

As the production environment changes, so should the model of it. Model updating can be viewed as a dynamic optimization problem. In GA and GP algorithms, solution updating can be achieved by increasing

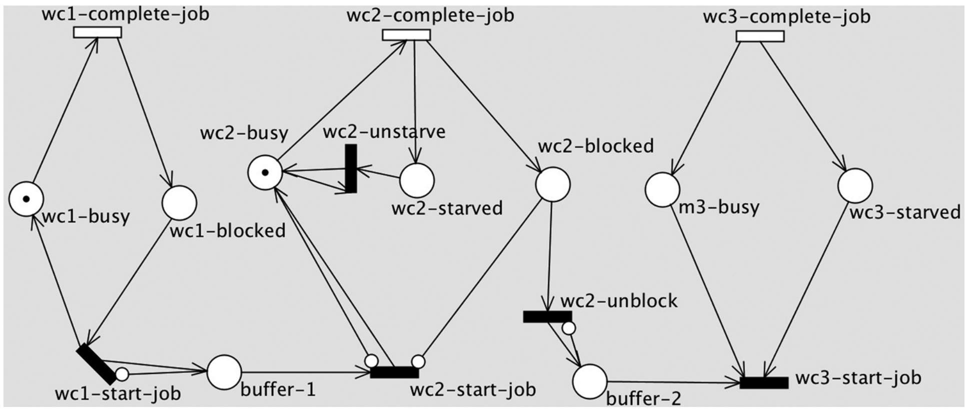


Fig. 7. 3-Machine system used for the example causal validation.

diversity as the environment changes. Many variations on three principal schemes have been used in the literature: Hypermutation schemes [34] increase the mutation rate as changes in the environment cause the population's fitness as a whole to degrade. Infusion schemes [35] allow new individuals into the population. Immigration schemes [36,37] evolve a separate, parallel and diverse population and, after some number of generations, move its best members into the main population.

In this work, an immigration scheme was used. A parallel population for immigrant individuals is updated periodically with new initial individuals, each representing a log trace from new log content. These new individuals replace poorly performing individuals of the immigration population from the last generation. The immigrant population is subjected to the same genetic operators as the main population (see Table 4). By scoring the "parallel" immigrant population separate from the main population, its individuals are protected from removal while they develop accurate model structure. The process is describe in Algorithm 2.

Table 3
Steady-state baseline and 20% increase in production rate at Machine M2.

Place	Baseline occupancy	Occupancy 20% + at M2	Comments
buffer-1	0.855	0.796	wc2 draws from buffer faster.
buffer-2	0.261	0.328	wc2 buffers faster.
wc1-blocked	0.692	0.624	Blockage at wc1 reduced…
wc1-busy	0.309	0.376	…thus wc2 busier.
wc2-blocked	0.143	0.188	Blocks more often.
wc2-busy	0.793	0.719	Blocked more, busy less.
wc2-starved	0.207	0.281	Line less balanced.
wc3-busy	0.473	0.548	wc2 supplies parts…
wc3-starved	0.527	0.452	…so wc3 starved less

Table 4
Mutation and crossover operators.

Operator	Type	Action
Add-arc	Mutation	Add an arc in a small neighborhood.
Remove-arc	Mutation	Remove an arc.
Add-trans	Mutation	Add an invisible transition.
Swap-priority	Mutation	Swap two routing priority values.
Add-machine-restart	Semantic	Add place and arc to pause workcenter.
Crossover-parallel	Crossover	Combine individuals with differing workcenters.
Crossover-colour	Crossover	Split paths by colour.

Algorithm 2. Evolution of parallel populations with immigration.

```
$p_{mute} \longleftarrow mutation\ probability$
$p_{cross} \longleftarrow crossover\ probability$
$Pop_m \longleftarrow makePNs(RandomJobTraces)$
$Pop_i \longleftarrow makePNs(RandomJobTraces)$
$gen \longleftarrow 0$
while $true$ do
    $gen \longleftarrow gen + 1$
    $Scores_i \longleftarrow evaluateFitness(Pop_i)$
    $Scores_m \longleftarrow evaluateFitness(Pop_m)$
    if $immigrate?(gen)$ then
        $Pop_m \longleftarrow Pop_m - worst(Pop_m, Scores_m) + best(Pop_i, Scores_i)$
        $Pop_i \longleftarrow Pop_i - best(Pop_i, Scores_i)$
    end if
    if $updateImmigrants?(gen)$ then
        $Pop_i \longleftarrow Pop_i - worst(Pop_i, Scores_i) + makePNs(RandomJobTraces)$
    end if
    $Pop_m \longleftarrow geneticOperatorsSelect(Pop_m, p_{mute}, p_{cross})$
    $Pop_i \longleftarrow geneticOperatorsSelect(Pop_i, p_{mute}, p_{cross})$
end while
```

In the algorithm, *immigrate*? is a Boolean function that returns true periodically to move the best immigrants into the main population. Similarly, *updateImmigrants* is a Boolean function that returns true periodically to replace some poorly performing individuals in the immigrant population with individuals created from new log content. *makePNs* makes PNs from random job traces, *evaluateFitness* measures fitness as described earlier, and *geneticOperatorsSelect* performs tournament selection on individuals and applies the genetic operators.

5. Evaluation

This section illustrates use of the methodology under conditions typical of automotive underbody assembly [38]. Underbody assembly systems are asynchronous assembly lines comprised of many workcenters. Each workcenter may be comprised of several automated units (e.g. robots) that work in concert to perform a sequence of operations. The controllers of the individual robots are capable of issuing messages. Given the appropriate genetic operators, it may be possible to apply the methodology to analyze these messages and thereby identify the workcenter-level process and its critical path. This, however, is not the objective of the work. Rather, because the methodology is intended to aid in making scheduling decisions, the focus of discussion is on effects that are abstracted from these details. In this abstracted view, one is focused on whether or not the workcenter as a whole is delayed – whether the detailed exceptional conditions reported by the robot controllers affect the execution time of the critical path. Where this is the case, the workcenter is considered "down" for a period that is the difference between its actual execution time and the normative execution time for the processes executing at the workcenter. Down periods are easily calculated from detailed execution messages, knowledge of the workcenter process ordering, and normative process execution times.

In this example, a portion of an assembly line consisting of six workcenters was modelled using the MJPdes simulation engine. Two of the six workcenters, `wc3-1` and `wc3-2` run identical operations; the system is free to randomly assign work to either of these workcenters as work becomes available.

The process of evolution is as follows. A portion of the message log consisting of 3000 messages is used to evolve a population of 50 individuals. At some later time, a second immigration population of 50 individuals is established from fresh log input. Initial individuals model single job traces as depicted in Fig. 8.

In general terms, the evolution process acts to embed in individuals properties that are inherent to a systemic view of the production process and thereby removes properties that are idiosyncratic to an individual's original job trace. For example, individuals representing a job trace such as depicted in Fig. 8 do not match logs where work can start on a downstream workcenter before other work finishes on an upstream workcenter. A genetic mutation operator, `add-machine-restart`, can introduce this systemic property. `add-machine-restart` modifies a segment of an individual's PN between where work is reported to start on a machine and where it ends on that machine. This operator adds a place after the work completed transition, and connects it to work starting transition. The connection prevents multiple jobs from starting (see Fig. 9).

Introducing parallel workcenters requires a crossover operation (mutation involve one individual; crossover, two). To model parallel workcenters, two individuals can be joined through crossover such that the new individual splits paths at a workcenter shared by the two original individuals and rejoins at another shared workcenter at least two workcenters downstream. The crossover operation `crossover-parallel` makes this modification to individuals.

The log can be analysed independent of individuals for evidence of BBS and BAS operation. The two mutation operators `bas-to-bbs` and `bbs-to-bas` switch PN structure between these two forms. These operators only have an effect where patterns of structure such as that provided by `add-machine-restart` have established a buffer place.

Priority, as described in Section 3.1, is used to specify which tokens are directed to which outbound arcs. The choice is based on token identifiers and the arc's priority number. The mutation operator `swap-arcs` randomly mutates the priority assignment of arcs outbound from

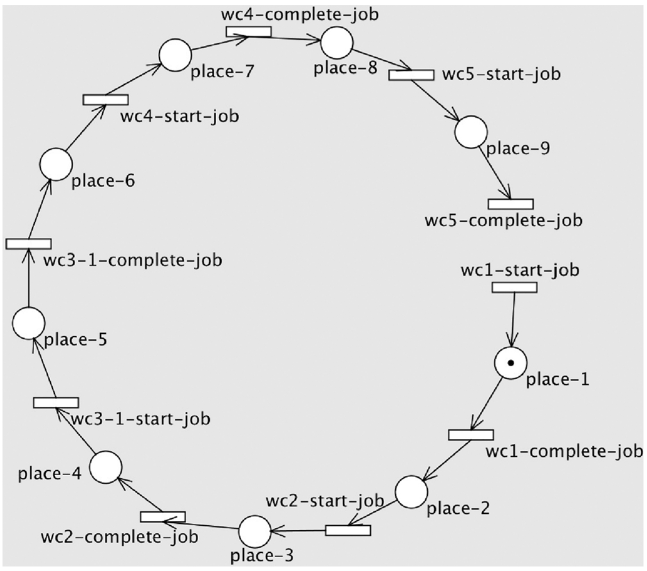


Fig. 8. An individual created directly from a job trace.

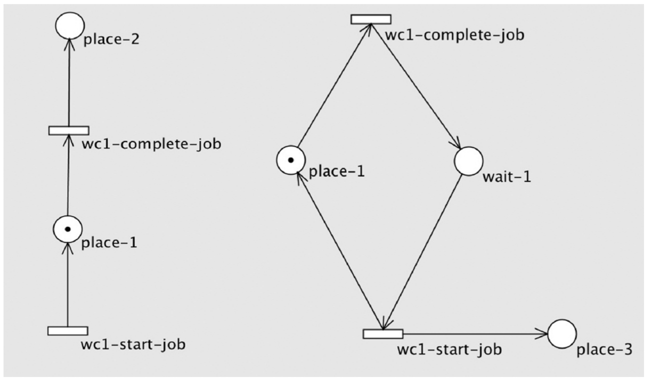


Fig. 9. The effect of genetic operator `add-machine-restart`. On the left is a segment of an individual before application of the operator. On the right is the result of applying the operator.

a transition.

An individual that roughly models the target production system, having been adapted through use of the operators just described is depicted in Fig. 10.

With each generation, the fitness function is applied to each individual to evaluate its fitness. The next generation of individuals is selected from these individuals using the tournament selection method with selection pressure of 4 (based on 50 individuals).

The evolution algorithm is a hybrid genetic programming method in the sense that individuals are updated not only through use of genetic operators, but also through inferences made in interpretation. Specifically, as described in Section 3.2, buffer sizes can be inferred from an interpretation in workcenters where blocking occurs. This is expressed with inhibitor arcs from places representing buffers. Also, transitions that express logical constraints, rather than processes, can be expressed with immediate transitions, replacing timed transitions. An individual based on the individual depicted in Fig. 10, but reflecting inferences from interpretation is shown in Fig. 11.

The example can be used to illustrate how the production system model is dynamically updated. The principal source of dynamic adaptation is the influx of high-scoring individuals from the immigrant population. Suppose that, currently, the best individual is one similar to that depicted in Fig. 11. If the parallel workcenter `wc3-2` were to become interoperative, log output would no longer report activity at the corresponding workcenter. Individuals are penalized for possessing transitions that are not exercised. None of the genetic operators can evolve individuals to eliminate multi-element substructure of the PN. However, since immigrant populations are occasionally restarted as job-trace individuals based on fresh log input, immigrants will not possess transitions involving `wc3-2`. With time, one of these individuals will immigrate to the main population.

5.1. Mixed-model production and coloured PNs

A similar production line to the one used in the example above, but emphasizing mixed-model production, is describe briefly. Suppose that instead of the two parallel workcenters, `wc3-1` and `wc3-2`, in the original problem, a single workcenter `wc3` is present. Suppose further that work consists of two job types, distinguished as `blue` and `red` that have significantly different work requirements at this workcenter. The best individuals modeling this system, depicted in Fig. 12, would be similar to that depicted in Fig. 11 however, (1) different meaning is attached to the transitions in `wc3` since these transitions refer to a single machine, (2) the two arcs from place-5 have a colour binding, red or blue, directing tokens to either execute for the red duration or the blue duration, and (3) inhibitor arcs are necessary to prevent concurrent operation of the red and blue sub-networks.

A crossover operator `crossover-colour` similar to `crossover-`

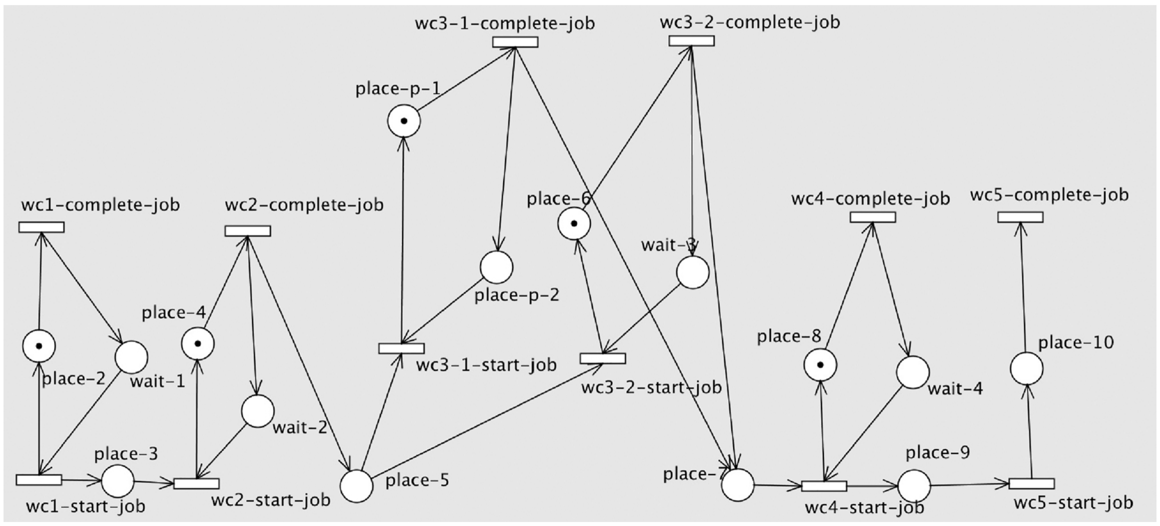


Fig. 10. An individual that exhibits asynchronous operation (and thus may interpret the log).

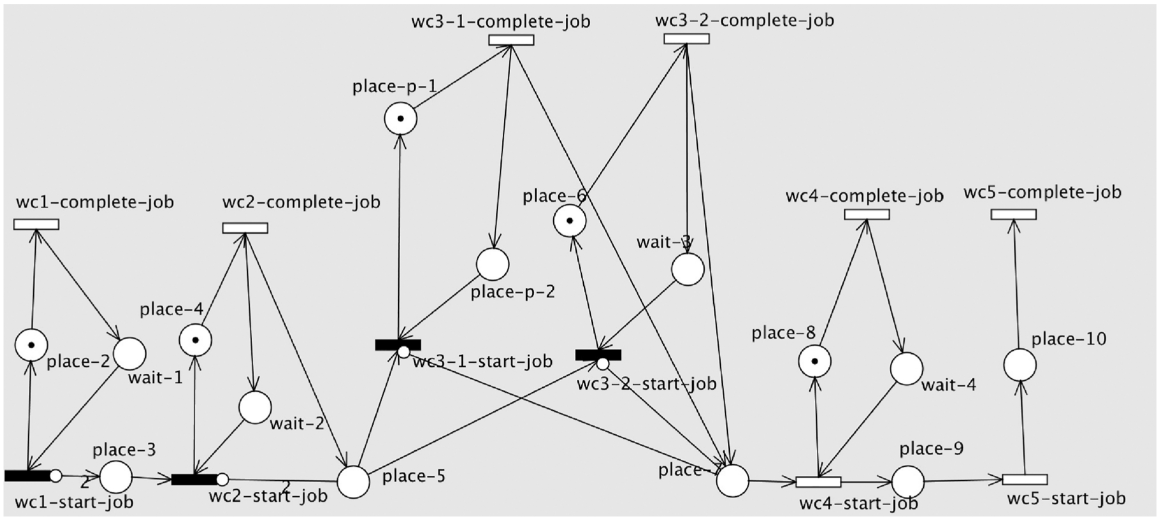


Fig. 11. The individual from Fig. 10, interpreted.

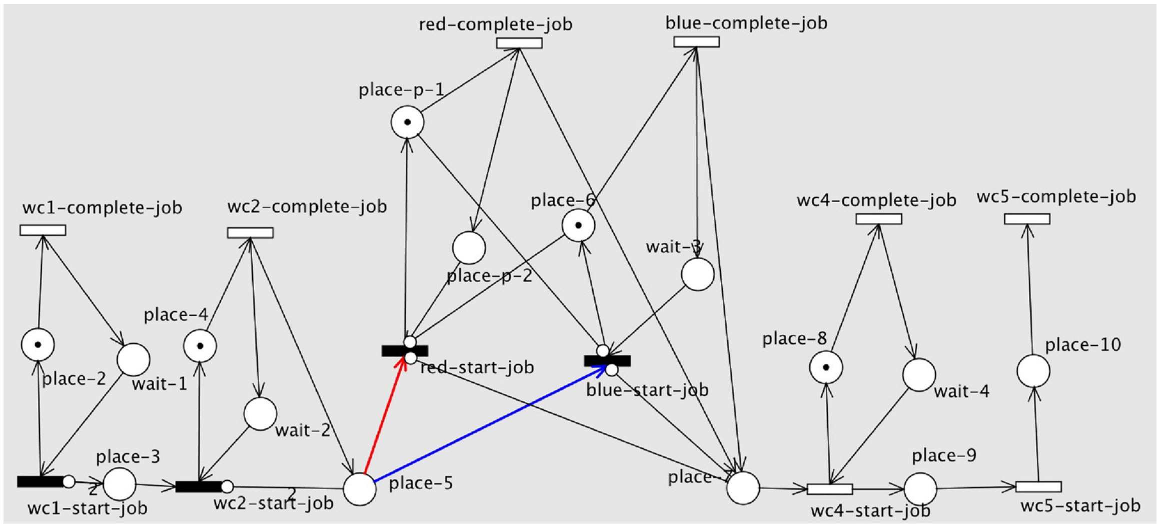


Fig. 12. The individual modeling mixed-model production.

Table 5
Fitness metrics.

Metric	Type	Penalty for failure
Interprets sequence	Interpretation	Large
Interprets interleaved	Interpretation	Medium
Uses all transitions	Interpretation	Medium
Variance	Log analysis	Medium
Causal	Causal analysis	Medium
Constrains buffers	Post-Interpretation	Small

`parallel` implements this modification of PN structure.

Table 5 summarizes fitness metrics used in the methodology. Proficiency at metric "Interprets sequences" entails the ability of the PN to match the sequence of messages in the log pertaining to each job individually. Intervening messages concerning other jobs are ignored. Job-trace individuals such as depicted in Fig. 8 can do this except where just one of $n - 1$ parallel workcenter is visited.

Proficiency at "Interprets interleaved" entails the ability of the PN to match messages that are interleaved from all workstations. This metric uses the interpretation algorithm of Section 3.2,

Proficiency at "Uses all transitions" entails a bijective relation between transitions and ordinary message types.

Proficiency at "Variance" entails a small variance in the time period between `job-starts-on-machine` and `job-ends-on-machine` messages, once workstation downtime is taken into account. A lack of proficiency in this metric suggests that the individual might score better had the job path been split along job-type (colour) distinctions, as happens when the `crossover-colour` operator is applied.

The "Causal" metric refers to causal analysis as described in Section 4.2. The metric "Constrains buffer" uses information from the interpretation to identify places representing buffers. A small penalty is applied to buffer places that do not have inhibitor arcs constraining the buffer size.

6. Conclusion

An accurate, up-to-date model of the production system is essential to production system control. Our proposed methodology addresses the three challenges to identifying and updating this model: (1) developing a method for inferring system structure from exceptional messages, (2) demonstrating that causal knowledge can be used to guide search to an accurate system model, and (3) showing that GP provides inherent means to update the model as the modelled system changes.

Using an expressive Petri net model and probabilistic and causal reasoning, we showed that it is possible to use log content to produce a model useful to production control tasks such as line balancing and job sequencing. Our methodology substantially addresses the needs of these analyses. One limitation, however, is that the methodology cannot determine the size of buffers unless that size is at some point too small to handle prevailing production conditions and exceptional (blocking) messages are generated. For many analytical purposes, however, such buffers can be treated as infinite. Incidentally, an advantage of GP over GA and numerical optimisation techniques is realised in such situations in that a GP individual can be edited by hand to reflect the buffer constraint and reinserted into the population.

Experience with the methodology demonstrates that it is generally robust. The methodology was implemented in a 5,000 line Clojure program which includes a web-based interface and PN drawing functions. The software is being made available open-source. Typical of genetic algorithms, the fitness function of several individuals can be evaluated in parallel. Problems such as the example described in Section 5 find a good solution within 2 min on 8 threads of a 4-core laptop. The speed-up from parallel execution with 8 threads is a factor of about 6.

We plan future work in three areas to extend the methodology and gain further insight. First, we intend to use the models in real-time production-control decision making. Second, we intend to explore how to more tightly integrated our methodology into smart manufacturing operational technology. Third, we plan to develop strategies to determine when model updating is best undertaken.

References

[1] Burke R, Mussomelli A, Laaper S, Hartigan M, Sniderman B. The smart factory: Responsive adaptive, connected manufacturing, Tech. rep. Deloite University Press; 2017.
[2] Scholl A. Balancing and sequencing of assembly lines. 2nd ed. Physica; 1999.
[3] Jayaswal S, Agarwal P. Balancing U-shaped assembly lines with resource dependent task times: a simulated annealing approach. J Manuf Syst 2014;33:522–34. http://dx.doi.org/10.1016/j.jmsy.2014.05.002.
[4] Akpinar S, Baykasoglu A. Modeling and solving mixed-model assembly line balancing problem with setups. Part II: A multiple colony hybrid bees algorithm. J Manuf Syst 2014;33:445–61. http://dx.doi.org/10.1016/j.jmsy.2014.04.001.
[5] Li J, Meerkov SM. Production system engineering. Springer Science + Business Media; 2009.
[6] Ljung L. Perspectives on system identification. Annu Rev Control 2010;34(1):1–12.
[7] IEC. IEC 62264-1 International Standard, Enterprise-control system integration – Part 1: Models and Terminology. 2003.
[8] Rozinat A, Mans RS, Song M, van der Aalst WMP. Discovering simulation models. Inf Syst 2009;34(3):305–27. http://dx.doi.org/10.1016/j.is.2008.09.002.
[9] Carmona J, Cortadella J. Process discovery algorithms using numerical abstract domains. IEEE Trans Knowl Data Eng 2014;26(12):3064–76.
[10] Basile F, Chiacchio P, Coppola J. Identification of time Petri net models. IEEE Trans Syst Man Cybern Syst 2016:1–15.
[11] Ould El Mehdi S, Bekrar R, Messai N, Leclercq E, Lefebvre D, Riera B. Design and identification of stochastic and deterministic stochastic Petri nets. IEEE Trans Syst Man Cybern Part A Syst Hum 2012;42(4):931–46. http://dx.doi.org/10.1109/TSMCA.2011.2173798.
[12] Alves de Medeiros AK. Genetic process mining (Ph.D. thesis). Technische Universiteit Eindhoven; 2006.
[13] Van Der Aalst W, Weijters T, Maruster L. Workflow mining: discovering process models from event logs. IEEE Trans Knowl Data Eng 2004;16(9):1128–42. http://dx.doi.org/10.1109/TKDE.2004.47.
[14] Dotoli M, Fanti MP, Mangini AM. Real time identification of discrete event systems by Petri nets. IFAC 2007 2007.
[15] Horváth A. Usability of deterministic and stochastic Petri nets in the wood industry: a case study. In: van Do T, Thi HAL, Nguyen NT, editors. Advanced computational methods for knowledge engineering Springer Nature; 2014. p. 119–27. http://dx.doi.org/10.1007/978-3-319-06569-4_9.
[16] Turner C, Tiwari A, Mehnen J. A genetic programming approach to business process mining. The genetic and evolutionary computation conference 1314 2008:1307.
[17] Yahya BN. The development of manufacturing process analysis: lesson learned from process mining. Junal Teknik Ind 2014;16(2):97–107. http://dx.doi.org/10.9744/jti.16.2.97-108.
[18] Denno P. MJPdes: a program for discrete event simulation of mixed-model production lines. 2017https://github.com/usnistgov/MJPdes.
[19] Nobile MS, Besozzi D, Cazzaniga P, Mauri G. The foundation of evolutionary Petri nets. CEUR workshop proceedings 988 2013:60–74.
[20] Whigham PA, Dick G, Maclaurin J. On the mapping of genotype to phenotype in evolutionary algorithms. 2017. http://dx.doi.org/10.1007/s10710-017-9288-x.
[21] Rothlauf F, Oetzel M. On the locality of grammatical evolution. LNCS 3905 2006:320–30.
[22] Denno P, Dickerson C, Harding J. Production system identification with genetic programming. International conference on manufacturing research. 2017.
[23] Diaz M. Petri nets: fundamental models, verification and applications. Wiley; 2009.
[24] Tina Lee Y-T, Riddick FH, Johan Ingemar Johansson B. Core manufacturing simulation data – a manufacturing simulation integration standard: overview and case studies. Int J Comput Integr Manufcturing 2011;24(8):689–709. http://dx.doi.org/10.1080/0951192X.2011.574154. URL https://www.tandfonline.com/doi/pdf/10.1080/0951192X.2011.574154?needAccess = true.
[25] International Organization for Standards. ISO 18629-1:2004 Industrial automation systems and integration – process specification language – Part 1: Overview and basic principles. 2004https://www.iso.org/standard/35431.html.
[26] Bruni R, Meseguer J, Montanari U, Sassone V. Functorial models for Petri nets. Inf Comput 2001;170(2):207–36. http://dx.doi.org/10.1006/inco.2001.3050http://eprints.ecs.soton.ac.uk/14742/.
[27] Degano P, Meseguer J, Montanari U. Axiomatizing the algebra of net computations and processes. Acta Inform 1996;33:641–67https://link.springer.com/content/pdf/10.1007%FBF03036469.pdf.
[28] Marsan MA, Balbo G, Conte G, Franceschinis G. Modelling with generalised stochastic Petri nets. System 1994:299.
[29] Jensen K, Salomaa A, Rozenberg G, Brauer W. Coloured Petri nets: basic concepts, analysis methods and practical use. Berlin: Springer; 1997.
[30] Kounev S, Spinner S, Meier P. Introduction to queueing Petri nets: modeling formalism, tool support and case studies. In: Kaeli D, Rolia J, editors. Proceedings of the 3rd ACM/SPEC international conference on performance engineering. 2012. p. 9–18.
[31] Specht DF. Probabilistic neural networks. Neural Netw. 1990;3(109).

[32] Ramakrishnan S, Emary IMME. On the application of various probabilistic neural networks in solving different pattern classification problems. World Appl Sci J 2008;4(6):772–80.
[33] Dijkstra EW. A note on two problems in connexion with graphs. Numer Math 1959;1:269–71.
[34] Cobb H, Grefenstette J. Genetic algorithms for changing environments. Parallel Probl Solv Nature 1992;2(1992):137–44http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.6501.
[35] Grefenstette JJ. Genetic algorithms for changing environments. Amsterdam: North Holland; 1992.
[36] Matei O, Pop PC, Sas JL, Chira C. An improved immigration memetic algorithm for solving the heterogeneous fixed fleet vehicle routing problem. Neurocomputing 2015;150:58–66. http://dx.doi.org/10.1016/j.neucom.2014.02.074.
[37] Yu X, Tang K, Yao X. An immigrants scheme based on environmental information for genetic algorithms in changing environments. 2008 IEEE congress on evolutionary computation (IEEE World Congress on Computational Intelligence) (3) 2008:1141–7. http://dx.doi.org/10.1109/CEC.2008.4630940.
[38] Alavian P, Denno P, Meerkov S. Multi-job production systems: definition, problems, analysis, and product-mix performance portrait of serial lines. Int J Prod Res 2017;00(4):1–8. http://dx.doi.org/10.1080/00207543.2017.1338779.