

This item was submitted to [Loughborough's Research Repository](#) by the author.
Items in Figshare are protected by copyright, with all rights reserved, unless otherwise indicated.

Dynamic production system identification for smart manufacturing systems

PLEASE CITE THE PUBLISHED VERSION

<https://doi.org/10.1016/j.jmsy.2018.04.006>

PUBLISHER

Elsevier © The Society of Manufacturing Engineers

VERSION

AM (Accepted Manuscript)

PUBLISHER STATEMENT

This work is made available according to the conditions of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0) licence. Full details of this licence are available at:
<https://creativecommons.org/licenses/by-nc-nd/4.0/>

LICENCE

CC BY-NC-ND 4.0

REPOSITORY RECORD

Denno, Peter, Charles E. Dickerson, and Jennifer A. Harding. 2019. "Dynamic Production System Identification for Smart Manufacturing Systems". figshare. <https://hdl.handle.net/2134/33111>.

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, Maryland, USA*

^b*Loughborough University, Loughborough, UK*

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.

Keywords: system identification, production systems, genetic programming

*Corresponding author

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 automate decision making.[1] Decisions concerning sequencing [2], line balancing [3], [4], and production system engineering [5] are sensitive to change 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 change 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. Change in system structure, capacities, and reliability are less common; but control policies are affected as much by change in these dimensions as they are by change 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 address-

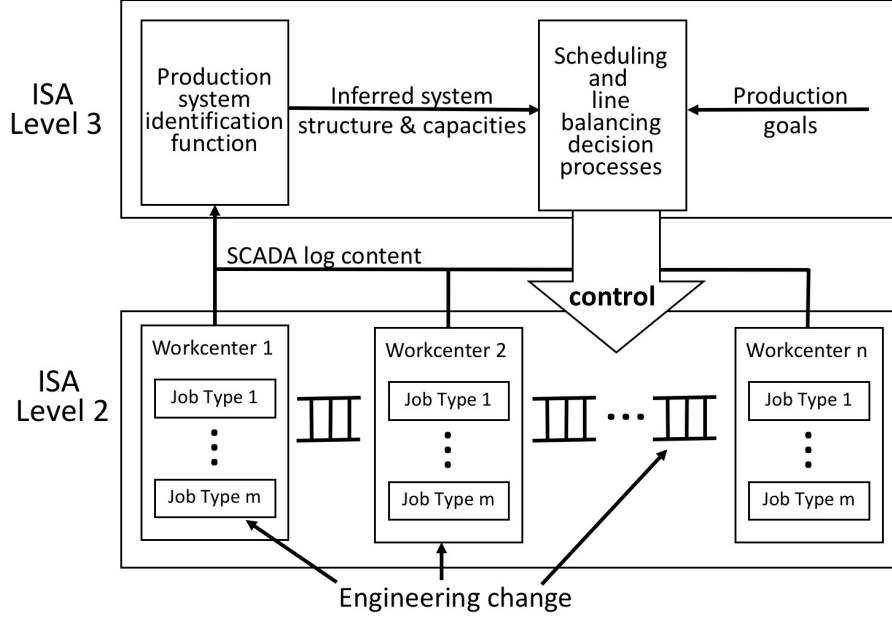
ing the challenge of dynamic production system identification in three important respects: (1) Rather than frequently occurring events, it is the infrequent, exceptional events that typically provide insight into system capacities and reliability. (2) Production system behaviour, especially machine blocking and starvation,
 35 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]. Our methodology
 40 infers the production-system structure and capacities specifically for use in line scheduling and balancing processes. (See Figure 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
 45 to its ability to (1) reproduce the content of logs describing typical Supervisory Control And Data Acquisition (SCADA) events, (2) 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
 50 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
 55 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
 60 a case study that uses the methodology. Section 6 concludes the paper with an

Figure 1: Production system identification in context



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 practised 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.

75 Alves de Medeiros [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
80 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] and [14] use integer linear
85 programming (ILP). Ould El Mehdi[11] et al. 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
90 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
95 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 ge-
netic programming for process mining. This short paper asserts that genetic
programming provides greater flexibility in problem formulation and the possi-
100 bility 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 es-
tablish a relationship between the information generated in production and the
system's components. The identified model is not designed for use as a simula-
105 tion 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 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 its 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.

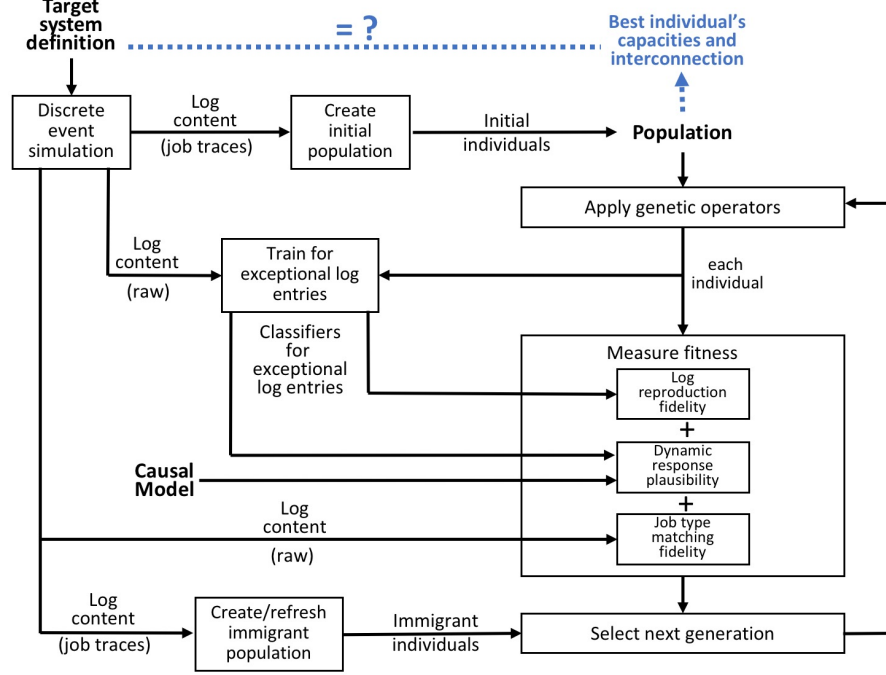
A sketch of the methodology is provided in Figure 2. To test the methodology, a discrete event simulation system for mixed-model production, MJPdes,[18] was developed to produce log data and performance parameters consistent with the behaviour of actual production lines. The log data is intended to resemble 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

135 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.

140 The remainder of Figure 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
145 bipartite graphs, so between each such transition is a place. In an initial indi-
vidual, 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
150 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 rep-
resenting structure (i.e. genotype, the individual's PN in our case) result in
155 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. Ge-
netic operators for PNs must be carefully designed to ensure locality. Our earlier
160 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.

Figure 2: System identification methodology



3.1. Augmented Queueing Petri Nets

165 Petri nets is a family of graphical formalisms to represent process causa-
tion, 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.
170 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 Figure 3 for how BBS and BAS behavior is expressed
175 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 used in this work to bound buffer size. k-bounded GSPNs can be reduced to PNs that are isomorphic to continuous-time Markov chains (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 $\{1..n\}$ called a *priority* is associated with each of the n outbound arcs from a transition. Tokens have identifiers in the set $\{1..m\}$ 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.

210 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.
215 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
220 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
225 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 re-
230 late 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
235 which this is analysed is described in Section 4.1.

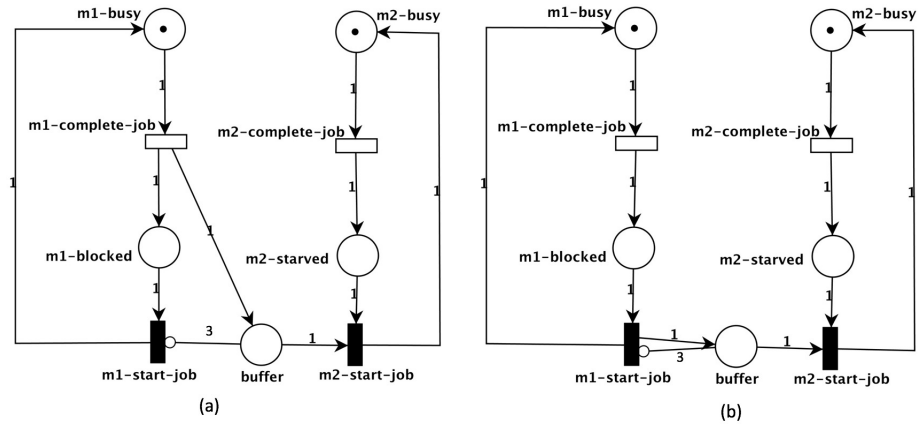
The fitness of an individual is expressed as a score based on its ability to model the log, and its validity relative to a causal model. Causal validity is discussed in Section 4.2. Regarding modelling the log, an individual's ability is

a combination of the deterministic behaviour of its PN with respect to ordinary
 240 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 abbrevi-
 ated (k -bounded) version of its reachability graph. The state of a PN having
 n places, $\{P_1, \dots, P_i, \dots, P_n\}$, is completely specified by a vector M (called
 245 a *marking*), $M = [m_1, \dots, m_i, \dots, m_n]$ where m_i is the quantity of tokens in
 the i^{th} 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
 250 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
 255 size. Buffer size can be expressed as inhibitor arcs in nets enhanced with GSPN
 modelling features. Inhibitor arcs on buffers may express a *block after service*
 or *block before service* discipline depending on the target transition chosen. (See
 Figure 3).

The interpretation process is described in Algorithm 1. In the algorithm,
 260 *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
 265 message at *logLine* of the log. *fired* returns the state corresponding to firing
 that transition.

Figure 3: 2-machine system with a buffer of size of 3: (a) block-before-service (b) block-after-service



Algorithm 1 Interpret Log

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

4. Addressing the 3 Limitations

This section discusses the three limitations of process mining mentioned earlier: handling of exceptional messages, causal validation, and dynamic model
 270 updating.

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
 275 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 Figure 4.) The input layer has one node for each feature (e.g. each component of a marking). The hidden layer
 280 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 Eqn 1

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

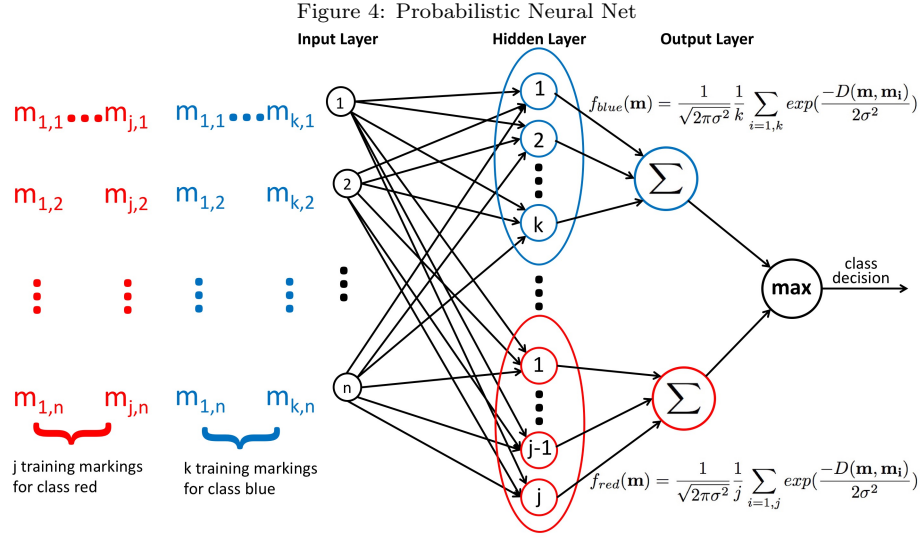
where

- 285 • \mathbf{m} is the input value, a marking,
- \mathbf{m}_i is the i th 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.

290 The hidden layer serves to estimate the probability density function (PDF) of each class using the Parzen window technique.[31] (See Eqn 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.

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



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 Eqn 3

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

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 $\hat{\mathbf{m}}_2$ are, respectively \mathbf{m}_1 and \mathbf{m}_2 component-wise normalized.

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

MinPath is found using Dijkstras 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 Eqn 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 Figure 5 are distributed as defined in Table 3. 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 Figure 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*) are labelled with the specific

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

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

Figure 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.

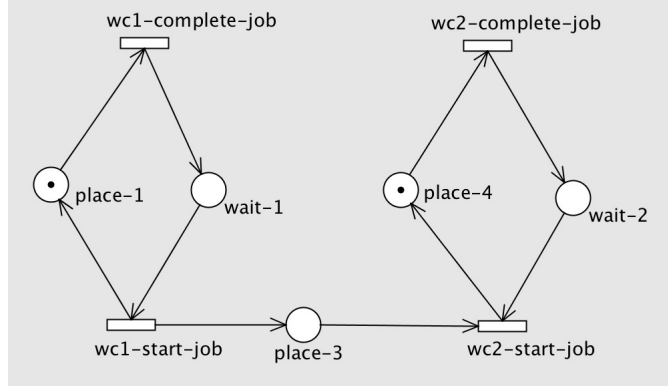


Table 1: Message occurrences in the example problem.

| 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 |

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. | Inferred | Cor? | Certain. |
| [0 0 1 1 1] | ordinary | yes | 0.999 | ordinary | yes | 0.999 | ordinary | yes | 0.999 |
| [0 1 0 1 0] | ordinary | yes | 0.193 | ordinary | yes | 0.999 | ordinary | yes | 0.969 |
| [1 1 0 0 2] | m2-unstarve | no | 0.708 | m2-unstarve | no | 0.551 | ordinary | yes | 0.999 |
| [0 1 0 1 3] | m1-block | yes | 0.365 | m1-block | yes | 0.743 | m1-block | yes | 0.898 |
| [1 1 0 0 3] | m2-unstarve | no | 0.050 | ordinary | yes | 0.998 | ordinary | yes | 0.969 |
| [0 1 0 1 1] | m1-unblock | no | 0.503 | m1-unblock | no | 0.199 | ordinary | yes | 0.935 |
| [1 1 0 0 0] | m2-starve | yes | 0.365 | m2-starve | yes | 0.734 | m2-starve | yes | 0.988 |
| [0 0 1 1 0] | ordinary | yes | 0.999 | ordinary | yes | 0.999 | ordinary | yes | 0.999 |
| [0 0 1 1 2] | ordinary | yes | 0.998 | ordinary | yes | 0.999 | ordinary | yes | 0.999 |
| [1 1 0 0 1] | m2-unstarve | yes | 0.365 | m2-unstarve | yes | 0.743 | m2-unstarve | yes | 0.915 |
| [0 1 0 1 2] | m1-unblock | yes | 0.365 | m1-unblock | yes | 0.743 | m1-unblock | yes | 0.823 |
| [1 0 1 0 0] | ordinary | yes | 0.975 | ordinary | yes | 0.999 | ordinary | yes | 0.727 |

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:

$$certainty(class_1, \mathbf{m}) = \frac{f_{class_1}(\mathbf{m}) - f_{class_2}(\mathbf{m})}{f_{class_1}(\mathbf{m})} \quad (5)$$

Certainty is zero at the decision boundary between two class.

The network hop algorithm of Eqn 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 Eqn 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.

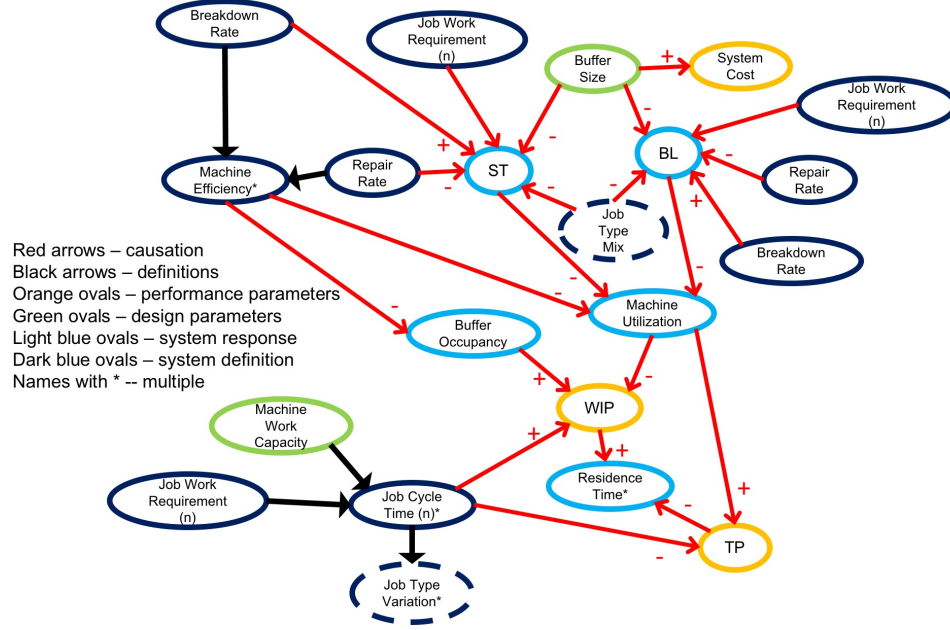
4.2. Causal Analysis

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

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

consistent with expert understanding of the domain. Figure 6 depicts a model of causes and effects in asynchronous serial lines with buffering.

Figure 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 a increase in the value at the head of the arrow.



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 responses 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
 370 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:

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

375 where

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

In Equation 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 Equation 7, each diagonal element of the matrix is the negative of the sum of the off-diagonal terms of that row.

380 \mathbf{Q} is used in the system provided by Equations 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.

$$\boldsymbol{\eta}\mathbf{Q} = \mathbf{0} \quad (8)$$

$$\boldsymbol{\eta}\mathbf{1}^T = 1 \quad (9)$$

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

Figure 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.

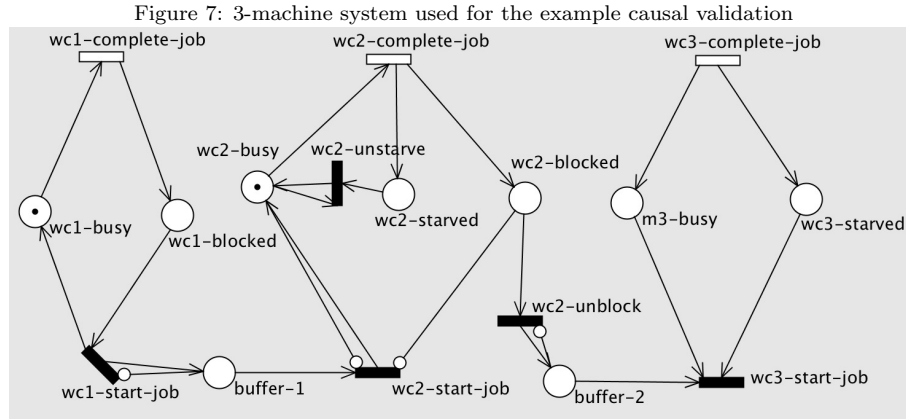


Table 3: Steady-state baseline and 20% increase in production rate at Machine M2

| 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 |

4.3. Model Updating

395 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 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
400 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
405 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 pop-
410 ulation 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 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

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

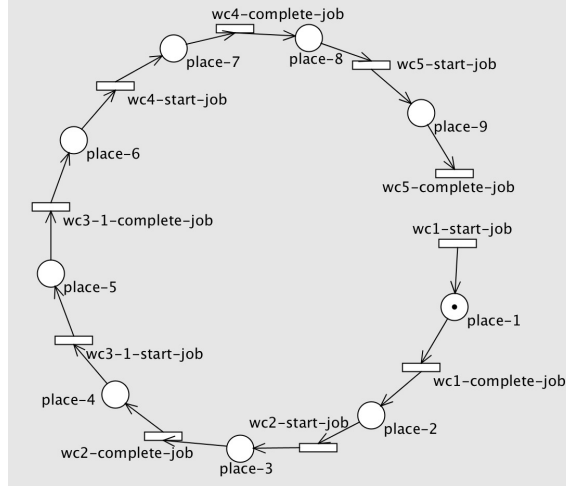
5. Evaluation

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
425 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
430 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
435 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.

440 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.

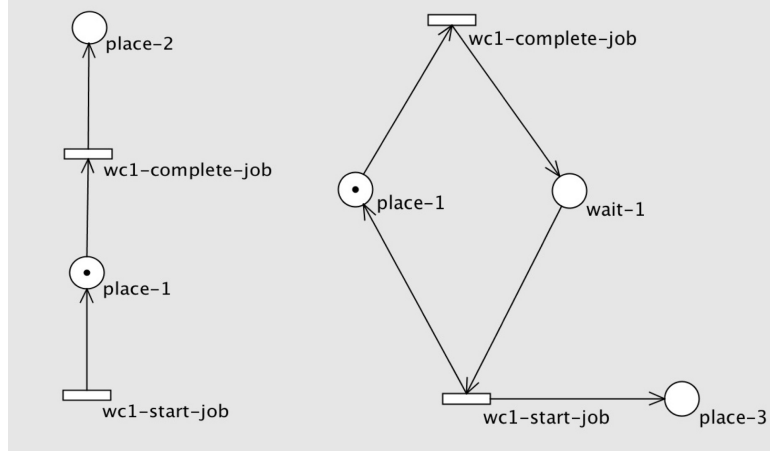
The process of evolution is as follows. A portion of the message log consisting
445 of 3000 messages is used to evolve a population of 50 individuals. At some later time, a second immigration population of 50 individuals is established from fresh log input. Initial individuals model single job traces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

Figure 8: An individual created directly from a job trace



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
450 removes properties that are idiosyncratic to an individual's original job trace. For example, individuals representing a job trace such as depicted in Figure 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.
455 **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 Figure 9.)

Figure 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.



460 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.

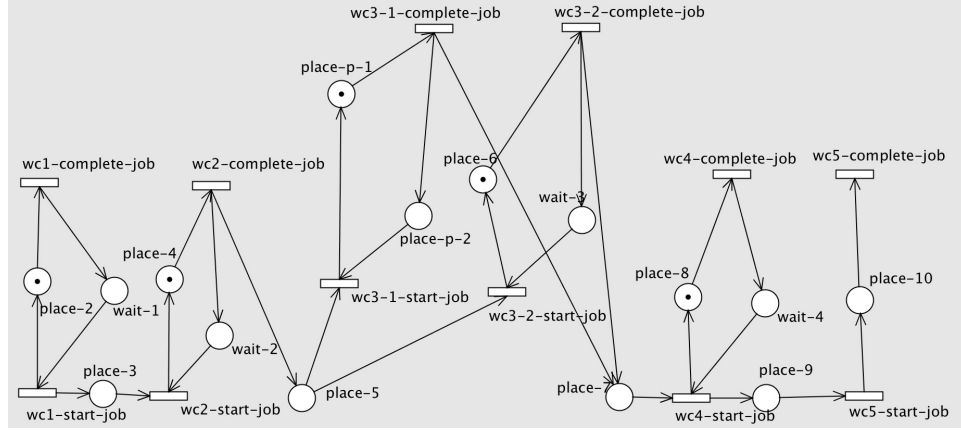
The log can be analysed independent of individuals for evidence of BBS and BAS operation. The two mutation operators **bas-to-bbs** and **bbs-to-bas** switch PN structure between these two forms. These operators only have an effect where patterns of structure such as that provided by **add-machine-restart**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

470 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 a transition.

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

10.

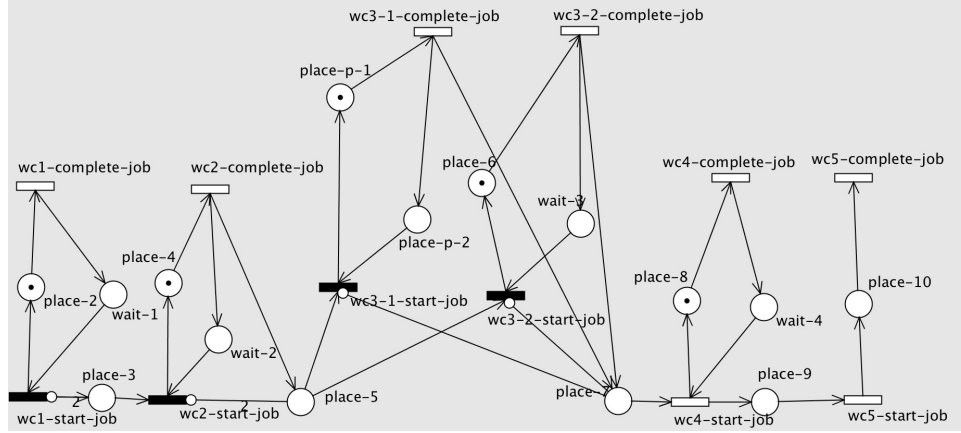
Figure 10: An individual that exhibits asynchronous operation (and thus may interpret the log)



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 Figure 10, but reflecting inferences from interpretation is shown in Figure 11.

Figure 11: The individual from Figure 10, interpreted



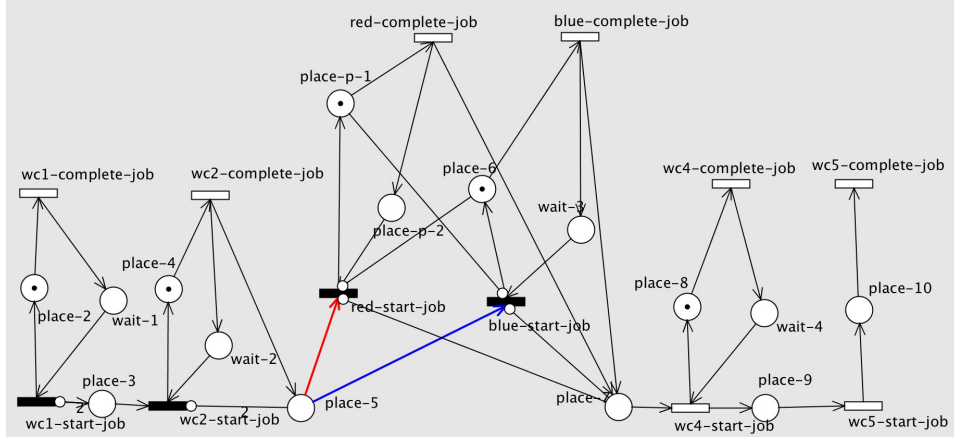
The example can be used to illustrate how the production system model is dynamically updated. The principal source of dynamic adaptation is the influx of high-scoring individuals from the immigrant population. Suppose that, currently, the best individual is one similar to that depicted in Figure 11. 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,
 510 depicted in Figure 12, would be similar to that depicted in Figure 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
 515 of the red and blue sub-networks.

Figure 12: The individual modeling mixed-model production



A crossover operator **crossover-colour** similar to **crossover-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
 520 of messages in the log pertaining to each job individually. Intervening messages
 concerning other jobs are ignored. Job-trace individuals such as depicted in
 Figure 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 inter-
 525 pretation algorithm of Section 3.2,

Proficiency at “Uses all transitions” entails a bijective relation between tran-

sitions 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
530 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
535 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.

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 |

6. Conclusion

An accurate, up-to-date model of the production system is essential to production system control. Our proposed methodology addresses the three chal-
540 lenges 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
545 modelled system changes.

Using an expressive Petri net model and probabilistic and causal reasoning, we show 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 minutes 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] R. Burke, A. Mussomelli, S. Laaper, M. Hartigan, B. Sniderman, The smart factory: Responsive adaptive, connected manufacturing, Tech. rep., Deloitte University Press (2017).

- 575 [2] A. Scholl, Balancing and Sequencing of Assembly Lines, 2nd Edition, Physica, 1999.
- [3] S. Jayaswal, P. Agarwal, Balancing U-shaped assembly lines with resource dependent task times: A Simulated Annealing approach, *Journal of Manufacturing Systems* 33 (2014) 522–534. doi:10.1016/j.jmsy.2014.05.002.
- 580 [4] S. Akpinar, A. Baykasoglu, Modeling and solving mixed-model assembly line balancing problem with setups. Part II: A multiple colony hybrid bees algorithm, *Journal of Manufacturing Systems* 33 (2014) 445–461. doi:10.1016/j.jmsy.2014.04.001.
- [5] J. Li, S. M. Meerkov, *Production System Engineering*, Springer Science+Business Media, 2009.
- 585 [6] Ljung, Perspectives on System Identification, *Annual Reviews in Control* 34 (1) (2010) 1–12.
- [7] IEC, IEC 62264-1 International Standard, Enterprise-control system integration – Part 1: Models and Terminology (2003).
- 590 [8] A. Rozinat, R. S. Mans, M. Song, W. M. P. van der Aalst, Discovering simulation models, *Information Systems* 34 (3) (2009) 305–327. doi:10.1016/j.is.2008.09.002.
- [9] J. Carmona, J. Cortadella, Process Discovery Algorithms using Numerical Abstract Domains, *IEEE Transactions on Knowledge and Data Engineering* 26 (12) (2014) 3064–3076.
- 595 [10] F. Basile, P. Chiacchio, J. Coppola, Identification of Time Petri Net Models, *IEEE Transaction on Systems, Man and Cybernetic Systems* (2016) 1–15.
- [11] S. Ould El Mehdi, R. Bekrar, N. Messai, E. Leclercq, D. Lefebvre, B. Riera, Design and Identification of Stochastic and Deterministic Stochastic Petri Nets, *IEEE Transactions on Systems, Man, and Cybernetics - Part A*:
- 600

Systems and Humans 42 (4) (2012) 931–946. doi:10.1109/TSMCA.2011.2173798.

[12] A. K. Alves de Medeiros, Genetic Process Mining, Ph.D. thesis, Technische Universiteit Eindhoven, (2006).

605 [13] W. Van Der Aalst, T. Weijters, L. Maruster, Workflow mining: Discovering process models from event logs, IEEE Transactions on Knowledge and Data Engineering 16 (9) (2004) 1128–1142. doi:10.1109/TKDE.2004.47.

[14] M. Dotoli, M. P. Fanti, A. M. Mangini, Real Time Identification of Discrete Event Systems by Petri Nets, in: IFAC 2007, 2007.

610 [15] A. Horváth, Usability of Deterministic and Stochastic Petri Nets in the Wood Industry: A Case Study, in: T. van Do, H. A. L. Thi, N. T. Nguyen (Eds.), Advanced Computational Methods for Knowledge Engineering, Springer Nature, 2014, pp. 119–127. doi:10.1007/978-3-319-06569-4_9.

[16] C. Turner, A. Tiwari, J. Mehnen, A Genetic Programming Approach to Business Process Mining, in: The Genetic and Evolutionary Computation Conference, 1314, 2008, p. 1307.
615

[17] B. N. Yahya, The Development of Manufacturing Process Analysis : Lesson Learned from Process Mining, Jurnal Teknik Industri 16 (2) (2014) 97–107. doi:10.9744/jti.16.2.97-108.

620 [18] P. Denno, MJPdes: A program for discrete event simulation of mixed-model production lines (2017).
URL <https://github.com/usnistgov/MJPdes>

[19] M. S. Nobile, D. Besozzi, P. Cazzaniga, G. Mauri, The foundation of Evolutionary Petri nets, CEUR Workshop Proceedings 988 (2013) 60–74.

625 [20] P. A. Whigham, G. Dick, J. Maclaurin, On the mapping of genotype to phenotype in evolutionary algorithms (2017). doi:10.1007/s10710-017-9288-x.

- [21] F. Rothlauf, M. Oetzel, On the Locality of Grammatical Evolution, in: LNCS 3905, 2006, pp. 320–330.
- 630 [22] P. Denno, C. Dickerson, J. Harding, Production System Identification with Genetic Programming, in: International Conference on Manufacturing Research, London, 2017.
- [23] M. Diaz, Petri Nets: Fundamental Models, Verification and Applications, Wiley, 2009.
- 635 [24] Y.-T. Tina Lee, F. H. Riddick, B. Johan Ingemar Johansson, Core Manufacturing Simulation Data a manufacturing simulation integration standard: overview and case studies, International Journal of Computer Integrated Manufacturing 24 (8) (2011) 689–709. doi:10.1080/0951192X.2011.574154.
- 640 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 (2004).
- 645 URL <https://www.iso.org/standard/35431.html>
- [26] R. Bruni, J. Meseguer, U. Montanari, V. Sassone, Functorial Models for Petri Nets, Information and Computation 170 (2) (2001) 207–236. doi:10.1006/inco.2001.3050.
- URL <http://eprints.ecs.soton.ac.uk/14742/>
- 650 [27] P. Degano, J. Meseguer, U. Montanari, Axiomatizing the algebra of net computations and processes*, Acta Informatica 33 (1996) 641–667.
- URL <https://link.springer.com/content/pdf/10.1007/%7B2FBF03036469.pdf>
- [28] M. A. Marsan, G. Balbo, G. Conte, G. Franceschinis, Modelling with generalised stochastic petri nets, System (1994) 299.
- 655

- [29] K. Jensen, A. Salomaa, G. Rozenberg, W. Brauer, Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Springer, Berlin, 1997.
- [30] S. Kounev, S. Spinner, P. Meier, Introduction to Queueing Petri Nets: Modeling Formalism, Tool Support and Case Studies, in: D. Kaeli, J. Rolia (Eds.), Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, Boston, 2012, pp. 9–18.
- [31] D. F. Specht, Probabilistic Neural Networks, Neural Networks 3 (109).
- [32] S. Ramakrishnan, I. M. M. E. Emary, On the Application of Various Probabilistic Neural Networks in Solving Different Pattern Classification Problems, World Applied Sciences Journal 4 (6) (2008) 772–780.
- [33] E. W. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik 1 (1959) 269–271.
- [34] H. Cobb, J. Grefenstette, Genetic Algorithms for Changing Environments, Parallel Problem Solving from Nature 2 (1992) (1992) 137–144. doi:10.1.1.48.6501.
- [35] J. Grefenstette, John, Genetic Algorithms for changing environments, North Holland, Amsterdam, 1992.
- [36] O. Matei, P. C. Pop, J. L. Sas, C. Chira, An improved immigration memetic algorithm for solving the heterogeneous fixed fleet vehicle routing problem, Neurocomputing 150 (2015) 58–66. doi:10.1016/j.neucom.2014.02.074.
- [37] X. Yu, K. Tang, X. Yao, 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) 1141–1147. doi:10.1109/CEC.2008.4630940.

- [38] P. Alavian, P. Denno, S. Meerkov, Multi-job production systems: definition, problems, analysis, and product-mix performance portrait of serial lines, *International Journal of Production Research* 00 (4) (2017) 1–8.
doi:<https://doi.org/10.1080/00207543.2017.1338779>.

685