Discovering Petri Net Models of Discrete-Event Processes by Computing T-Invariants

Tonatiuh Tapia-Flores, Ernesto López-Mellado, Ana Paula Estrada-Vargas, Jean-Jacques Lesage

Discovering Petri Net Models of Discrete-Event Processes by Computing T-Invariants

Tonatiuh Tapia-Flores, Ernesto López-Mellado, Ana Paula Estrada-Vargas, and Jean-Jacques Lesage

***Abstract*— This paper addresses the problem of discovering a Petri Net (PN) from a long event sequence representing the behavior of discrete-event processes. A method for building a 1-bounded PN able to execute the events sequence *S* is presented; it is based on determining causality and concurrence relations between events and computing the t-invariants. This novel method determines the structure and the initial marking of an ordinary PN, which reproduces the behavior in *S*. The algorithms derived from the method are efficient and have been implemented and tested on numerous examples of diverse complexity.**

***Note to Practitioners*—Model discovery is useful to perform reverse engineering of ill-known systems. The algorithms proposed in this paper build 1-bounded PN models, which are enough powerful to describe many discrete-event processes from industry. The efficiency of the method allows processing very large sequences. Thus, an automated modeling tool can be developed for dealing with data issued from real systems.**

***Index Terms*— Model discovery, Petri nets (PNs), t-invariants.**

I. Introduction

DISCOVERING formal models from external observation of systems behavior is an interesting and challenging approach for reverse engineering of discrete-event processes which are unknown or ill known. Although the problem is relatively recent, it deserves the attention of several research groups in the fields of discrete-event systems (DESs) and workflow management systems (WMSs).

A. Model Discovery

Pioneer works on the matter, named language learning techniques, appeared in computer sciences in the late 60s. The aim was to build fine representations (finite automata or grammars) of languages from samples of accepted word [1], [2].

In the field of DES, where the problem is usually named *identification*, several approaches have been proposed for building models representing the observed behavior of automated processes. The incremental approach proposed in [3] and [4] allows building safe interpreted Petri Net (IPN) models from a continuous stream of system's outputs. In [5], a method based on the statement and solution of an integer linear programming problem is proposed; it allows building PN from a set of sequences of events. Extensions of this method are proposed in [6] and [7]. In [8], a method for deriving finite automata from sequences of inputs and outputs is presented; it is applied to fault detection of manufacturing processes. An extension to this method that allows obtaining distributed system models is presented in [9]. In [10], input–output identification of automated manufacturing process is addressed; an IPN is obtained from a set of sequences of input–output vectors collected from the controller during the system cyclic operation. The method is extended for dealing with a long single observation of input–output vectors [11]. Later a new two stages method for dealing with observable and nonobservable parts of the PN has been proposed [12]. Other research groups have proposed methods for obtaining timed models [13]–[15]. More complete reviews on DES identification can be found in [16] and [17].

In WMS, the analogous problem is named *process mining: Discovery*; the system observation is given as a set of sequences from a finite alphabet of tasks, representing execution logs of business processes. A first proposal is reported in [18], in which a finite automaton, called conformal graph is obtained. In [19], it is proposed a probabilistic approach to find the concurrent and direct relations between tasks. The input of the method is a sequence of events that represent the activities that occurred in a WMS; the obtained model is graph similar to a PN. In [20], a mining method called alpha algorithm is presented. In this method, a workflow tasks log composed by several traces is recorded sequentially and processed yielding a subclass of PN called workflow net. Numerous publications present extensions of this algorithm, namely, [21]–[24]. In particular, in this last work, a method that computes WFN including nonfree choice constructs using invisible tasks is addressed; it allows discovering more complex models involving implicit dependencies. An alternative approach to alpha algorithm variations is based on the theory of regions [25]. In [26], a mining technique named inductive miner which is able to return fitting models in a finite time is presented. A wide literature review on process mining discovery can be found in [27].

Although there are many process discovery techniques, some of them cannot synthesize models able to replay the whole log, or that may represent nonobserved behavior. In process mining literature, a model is qualified (with respect

Manuscript received March 6, 2017; accepted March 8, 2017. This paper was recommended for publication by Associate Editor C. F. Mahulea and Editor S. Reveliotis upon evaluation of the reviewers' comments. The work of T. Tapia-Flores was supported by CONACYT, Mexico under Grant 263566. *(Corresponding author: Ernesto López-Mellado.)*

T. Tapia-Flores and E. López-Mellado are with CINVESTAV Unidad Guadalajara, 45019 Zapopan, Mexico (e-mail: ttapia@gdl.cinvestav.mx; elopez@gdl.cinvestav.mx).

A. P. Estrada-Vargas is with Oracle de México S. A. de C. V., 45110 Zapopan, Mexico (e-mail: ana.estrada@oracle.com).

J.-J. Lesage is with LURPA, ENS Cachan, University of Paris-Sud, Université Paris-Saclay, 94235 Cachan, France (e-mail: Jean-jacques.lesage@ens-cachan.fr).

This paper has supplementary downloadable multimedia material available at http://ieeexplore.ieee.org provided by the authors. The Supplementary Material contains additional examples and comparative test of software implementation of the discovery method. This material is 693 KB in size.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2017.2682060

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

to a known reference model) with four metrics called *dimensions* [28]: *fitness:* a model has perfect fitness if is able to reproduce the entire log; *precision:* it measures the quantity of exceeding behavior represented in the model; *simplicity:* it evaluates how simple is the model, and *generalization:* it determines what part of the nonobserved behavior actually corresponds to the real process.

B. Approach

This paper follows the approach presented in [12] for dealing with input–output identification. The method processes off-line the I/O-sequence w captured during the process operation and builds an IPN model that reproduces w.

Consider as an example a process handling three inputs (s, x, y) and three outputs (A, B, C), from which the following I/O sequence is captured:

$$w = \begin{array}{c} s\\ x\\ y\\ A\\ B\\ C \end{array} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \begin{bmatrix}1\\0\\0\\1\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\1\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \begin{bmatrix}0\\1\\0\\0\\1\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\\1\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \begin{bmatrix}1\\0\\0\\1\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\1\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\1\\0\\0\\1\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\\0\\1\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \begin{bmatrix}1\\0\\0\\1\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\1\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \begin{bmatrix}0\\1\\0\\0\\1\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\\1\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \begin{bmatrix}1\\0\\0\\1\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\1\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\1\\0\\0\\1\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\\0\\1\end{bmatrix}$$

The construction of an IPN model from w is performed in two stages.

The first stage processes the sequence w and obtains the observable part of the model consisting of components using observable places and transitions labeled with output symbols and expressions of input symbols, respectively (Fig. 1). This determines the set of transitions T. A transition sequence S that corresponds to the observed event sequence is also delivered. In the example, $S_1 = t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_5\ t_6\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_5$ is obtained.

The second stage builds a PN model that is able to reproduce S (perfect fitness) with a reduced excessive behavior (high precision); furthermore, the number of nodes is small (simplicity) since only the necessary places to represent the discovered structural constrains are added. The resulting PN corresponds to the process' internal behavior represented by nonobservable places. In the example, the obtained PN showed in Fig. 2 reproduces S_1 (thus w). Merging this model with the observable model and eliminating implicit places (p_{11}, p_{22}, p_{33}) yields the final IPN model shown in Fig. 3.

C. Contribution

This paper focuses on the second stage. A new method for building 1-bounded PNs from a set S of sequences S_i of transitions (tasks or events) is introduced. Supported by a novel approach, new results allowing addressing more complex behaviors such as implicit dependencies between tasks that are not observed consecutively are presented. The method is based on determining from S, causal and concurrence relations between events and the t-invariants of the PN to discover. Then obtained invariants allow, first, determining the initial structure of a PN, and later, adjusting the model when the computed t-invariants do not coincide with those of the initial model. The main features of this proposal are: the technique for computing the t-invariants from S, and the reduction of the exceeding language in the model by determining implicit dependencies; furthermore, the efficiency of the algorithms allows dealing with numerous long sequences in S. This paper extends the results presented in [29]; the results and proofs have been revised and more examples tested with the developed software are also included.

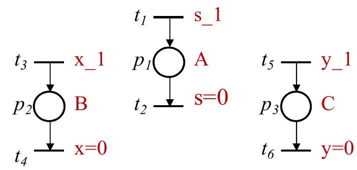


Fig. 1. First step of the identification method: IPN fragments.

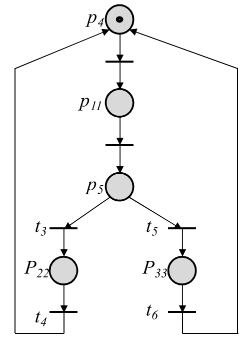


Fig. 2. Nonobservable PN.

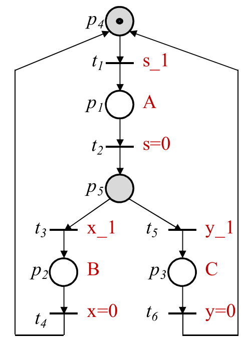


Fig. 3. IPN model integrating observable components.

In order to illustrate the aim and the features of our discovery method, consider the following tiny example. Given a single sequence $S_1 = t_1t_3t_2t_6t_4t_3t_5t_6t_1t_3t_2t_6t_1t_3t_2t_6t_4t_3t_5t_6$ involving six events $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$, the obtained model using the method presented in [12] is shown in Fig. 4(a). It is easy to see that it can reproduce S; however, it can execute also the subsequences $t_1t_3t_5$ and $t_4t_3t_2$ which are not in S. Note that this PN has four t-invariants. In contrast, the new method builds the PN model in Fig. 4(b), which reproduces S and does not contain the exceeding behavior pointed above. In this model, the transitions t_1 and t_2 are related by a place in despite that they were never observed consecutively. The identified PN has only two invariants: $[1\ 1\ 1\ 0\ 0\ 1]^T$ and $[0\ 0\ 1\ 1\ 1\ 1]^T$.

D. Outline

The paper is organized as follows. In Section II, the basic notions on PN are recalled. Section III states the addressed problem. In Section IV, basic relations computed from the tasks sequence are introduced. Section V presents a technique for determining the t-invariants. In Section VI, the PN synthesis method is described. Section VII outlines implementation

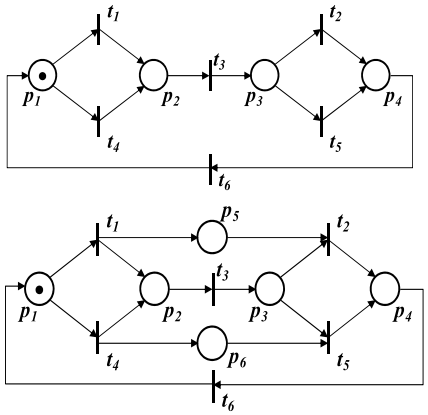


Fig. 4. PN models built from the sequence S.

and tests. Finally, in Section VIII, a discussion regarding advantages and limitations of the method with respect to related works is presented.

II. Ordinary Petri Nets

This section presents the basic concepts and notations of ordinary PN used in this paper.

Definition 1: An ordinary PN structure G is a bipartite digraph represented by the four-tuple $G = (P, T, I, O)$ where $P = \{p_1, p_2, \ldots, p_{|P|}\}$ and $T = \{t_1, t_2, \ldots, t_{|T|}\}$ are finite sets of vertices named places and transitions, respectively; $I(O) : P \times T \rightarrow \{0, 1\}$ is a function representing the arcs going from places to transitions (from transitions to places). For any node $x \in P \cup T$, ${}^\bullet x = \{y | I((y, x)) = 1\}$, and $x^\bullet = \{y | O((x, y) = 1\}$.

The incidence matrix of G is $C = C^+ - C^-$, where $C^- = [c_{ij}^-]$; $c_{ij}^- = I(p_i, t_j)$; and $C^+ = [c_{ij}^+]$; $c_{ij}^+ = O(p_i, t_j)$ are the preincidence and postincidence matrices, respectively.

A marking function $M : P \rightarrow Z^+$ represents the number of tokens residing inside each place; it is usually expressed as an $|P|$-entry vector. Z^+ is the set of nonnegative integers.

Definition 2: A PN system or PN is the pair $N = (G, M_0)$, where G is a PN structure and M_0 is an initial marking.

In a PN system, a transition t_j is *enabled* at marking M_k if $\forall p_i \in P$, $M_k(p_i) \geq I(p_i, t_j)$; an enabled transition t_j can be fired reaching a new marking M_{k+1}, which can be computed as $M_{k+1} = M_k + Cu_k$, where $u_k(i) = 0$, $i \neq j$, $u_k(j) = 1$; this equation is called the PN state equation. The reachability set of a PN is the set of all possible reachable markings from M_0 firing only enabled transitions; this set is denoted by $R(G, M_0)$.

Definition 3: A PN system is *1-bounded* or *safe* iff for any $M_i \in R(G, M_0)$ and any $p \in P, M_i(p) \leq 1$. A PN system is *live* iff for every reachable marking $M_i \in R(G, M_0)$ and $\forall t \in T$ there is a reachable marking $M_k \in R(G, M_i)$ such that t is enabled in M_k.

Definition 4: A *t-invariant Y_i of a* PN is an integer solution to the equation $CY_i = 0$ such that $Y_i \geq 0$ and $Y_i \neq 0$. The *support* of Y_i denoted as $\langle Y_i \rangle$ is the set of transitions whose corresponding entries in Y_i are strictly positive. Y is *minimal* if its support is not included in the support of other t-invariant. A *t-component* $G(Y_i)$ is a subnet of PN induced by a $\langle Y_i \rangle$: $G(Y_i) = (P_i, T_i, I_i, O_i)$, where $P_i = {}^\bullet \langle Y_i \rangle \cup \langle Y_i \rangle^\bullet$, $T_i = \langle Y_i \rangle$, $I_i = P_i \times T_i \cap I$, and $O_i = P_i \times T_i \cap O$.

In a t-invariant Y_i, if we have initial marking (M_0) that enables a $t_i \in \langle Y_i \rangle$, when t_i is fired, then M_0 can be reached again by firing only transitions in $\langle Y_i \rangle$.

III. Problem Statement and Proposed Approach

A. Petri Net Discovery

First, we formulate the problem of model discovering in the context of automated manufacturing processes and then, the assumptions made are stated.

The Problem:

Definition 5: Given a finite alphabet of events or tasks $T = \{t_1, t_2, \ldots, t_n\}$ and a set S of finite sequences $S_i = t_1 t_2 \ldots t_j \in T^*$, the PN discovery problem consists of building a 1-bounded PN structure and determining an initial marking M_0, which allow execute all the S_i from M_0. The PN must have a reduced number of nodes, and reproduces the least possible exceeding behavior. Every $t_j \in T$ appears once in the PN. The number of places is unknown.

In the context of automated manufacturing systems, S_i represents the observation of relevant input–output events sampled from a closed-loop controller during a long execution period of time, for example, a complete production process performing diverse repetitive jobs [10]. Several observations S_i from the same process may be dealt.

Furthermore, S_i cannot include consecutively two or more times the same event, since events correspond to instantaneous changes in input and/or outputs binary variables to/from the controller; events delimit the duration of operations (tasks, activities). For example, when an event is detected, let us say the rising edge of the binary input variable a (e_j); one must observe a falling edge of a (e_k) before to observe its rising edge (e_j) again. Thus, in contrast to the hypothesis made for process mining, the same event cannot appear consecutively in the sequences in S.

The delimitation of cyclic subsequences is not known *a priori*. However, if the observed behavior is provided in the form of cyclic subsequences (traces) $\sigma_i \in T^*$, a single sequence S_1 can be formed by the concatenation of σ_i using a transition t^* between each trace: $S_1 = \sigma_1 t^* \sigma_2 t^* \ldots \sigma_n t^*$ to process all the observations.

Assumptions: It is assumed that processes are well behaved, i.e., there are no faults, deadlocks, or overflows during the observation period. This is a realistic assumption since the processes whose models have to be discovered are supposed to be in operation, although the model is currently unknown or ill known. Of course, if the sequences describe halting events corresponding to the end of a production process or the sampling of some sequences is interrupted before the end of the process; this could be captured in the obtained PN as a deadlock.

Thus, we can consider that the event streams $S_i \in T^*$ are generated from the initial marking, by a 1-bounded ordinary PN without self-loops, which is deadlock-free in the repetitive component.

Given that process is unknown (black-box approach), we cannot assume completeness of S. The only constraint is

the absence of repeated events, signaled above. We expect to deal with sequences representing most of the possible behavior of the process. Longest are the sequences, better is the model approximation to the actual process behavior. However, the method leads to a PN model that reproduces the observed sequences.

B. Overview of the Method

The proposed method synthesizes a PN and an initial marking from which the sequences in S can be executed. It focuses on the computation of the causal and concurrent relations between the tasks S. This is achieved by determining the t-invariants (that are supposed to exist in systems that exhibit repetitive behavior), which also are used to find causal implicit relations between events that are not observed consecutively.

In a first stage, several binary relations between transitions are determined from all $S_i \in S$; based on these relations, the t-invariants are computed. Afterward, causal and concurrent relations are determined, and together with the discovered t-invariants, the structure of a PN model is built. Finally, the t-invariants are used again for reducing the possible exceeding language by determining causality between events not observed consecutively.

IV. Basic Concepts and Relations

We will describe the method, without loss of generality, considering a single sequence S; dealing with several S_i, which is trivially extended, is mentioned later in Section V-D. First, we introduce several relations derived directly from S. Some of the following definitions have been taken and adapted from [12].

Definition 6: The relationship between transitions that are observed consecutively in S is expressed in the relation Seq $\subseteq T \times T$ which is defined as Seq $= \{(t_j, t_{j+1}) | 1 \leq j < |S| - 1\}$; t_a Seq t_b will be frequently denoted as $t_a \prec t_b$. The relation between transitions that never occur consecutively in S is $T \times T \backslash$Seq; pairs in this relation are denoted as $t_a >< t_b$.

This relation has been used in [18] and [20] as a representation of the precedence relationship in the events on a sequence: $a < b$ means that a has been observed immediately before b.

Definition 7: Every couple of consecutive transitions $(t_a, t_b) \in$ Seq can be classed into one of the following situations.

i) *Causal Relationship:* the occurrence of t_a enables t_b, denoted as $[t_a, t_b]$. In a PN structure, this implies that there must be at least one place from t_a to t_b.

ii) *Concurrent Relationship:* if both t_a and t_b are simultaneously enabled, and t_a occurs first, its firing does not disable t_b. In a 1-bounded PN structure, this implies that it is impossible the existence of a place from t_a to t_b. In this case, t_a and t_b are said to be concurrent, denoted as $t_a || t_b$.

The notion of concurrence has been taken from the process algebra [30]; van der Aalst *et al.* [20], Wen *et al.* [21], and Wang *et al.* [22] use this notion for defining the concurrence between transitions.

Now a relation that establishes a key property named repetitive dependence is recalled [12].

Definition 8: A transition t_j is *repetitively dependent* of t_k, denoted as $t_j \prec t_k$ iff t_k is always observed between two apparitions of t_j in S. If t_j has been observed at least twice in S, then $t_j \prec t_j$. The set of transitions from which t_j is repetitively dependent is given by the function $\mathrm{Rd}(t_j) : T \rightarrow 2^T$; then $\mathrm{Rd}(t_j) = \{t_k | t_j \prec t_k\}$. If t_j was observed only once in S, then $\mathrm{Rd}(t_j) = \emptyset$.

Property 1: The transitions in a $\mathrm{Rd}(t_j)$ are included in the support of at least one t-invariant.

Proof: $\mathrm{Rd}(t_j)$ is the set of transitions that must invariantly occur to fire t_j repeatedly. Thus, the proof follows directly from *Definition* 8 and the concept of t-invariant. Any $t_k \in \mathrm{Rd}(t_j)$ may belong also to other t-components. ■

Example 1: Consider the set of tasks $T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$ and the sequence $S = \boldsymbol{t_1}\ t_2\ t_3\ t_4\ \boldsymbol{t_1}\ t_2\ t_4\ t_3\ t_5\ t_6\ t_7\ t_4\ \boldsymbol{t_1}\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_4\ \boldsymbol{t_1}\ t_2\ t_3\ t_4\ \boldsymbol{t_1}\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ \boldsymbol{t_1}\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ \boldsymbol{t_1}\ t_2\ t_4\ t_3\ t_1\ t_2\ t_3\ t_4\ \boldsymbol{t_1}\ t_2\ t_3\ t_4\ \boldsymbol{t_1}\ t_2\ t_4\ t_3\ \boldsymbol{t_1}\ t_2\ t_3\ t_4\ \boldsymbol{t_1}$. From S it is obtained Seq $= \{(t_1, t_2), (t_2, t_3), (t_3, t_4), (t_4, t_1), (t_4, t_5), (t_5, t_6), (t_6, t_7), (t_7, t_4), (t_2, t_4), (t_4, t_3), (t_3, t_1), (t_3, t_5)\}$. Furthermore, one may observe that $t_1 \prec t_2$, $t_1 \prec t_3$, $t_1 \prec t_4$, thus $\mathrm{Rd}(t_1) = \{t_1, t_2, t_3, t_4\}$. The rest of the Rd sets are $\mathrm{Rd}(t_2) = \{t_1, t_2, t_3, t_4\}$, $\mathrm{Rd}(t_3) = \{t_1, t_2, t_3\}$, $\mathrm{Rd}(t_4) = \{t_4\}$, $\mathrm{Rd}(t_5) = \{t_4, t_5, t_6, t_7\}$, $\mathrm{Rd}(t_6) = \{t_4, t_5, t_6, t_7\}$, $\mathrm{Rd}(t_7) = \{t_4, t_5, t_6, t_7\}$.

A substructure that can be straightforward derived from the sequence is the cycle involving only two transitions [31].

Definition 9: Two transitions t_a and t_b are called transitions in a *two-length cycle* (Tc) relation if S contains the subsequences $t_a t_b t_a$ or $t_b t_a t_b$. Tc denotes the set of transition pairs fulfilling this condition.

It is easy to see that simple substructures of PN can be derived straightforward from Tc. From *Example 1*, $Tc = \emptyset$.

Now, conditions for determining causal and concurrence relationships are given.

Proposition 1: Let t_a and t_b be two transitions in T; then $t_a || t_b$ if $(t_a, t_b), (t_b, t_a) \in$ Seq, i.e., t_a and t_b have been observed consecutively in S in both orders, and if t_a, t_b do not form a Tc.

Proof: It follows from *Definition* 7(ii) and from the condition that excludes the subsequence which characterizes a *Tc*. ■

Thus, the set of concurrent transition pairs deduced from S is $ConcR = \{(t_a, t_b) | t_a \prec t_b \wedge t_b \prec t_a \wedge (t_a, t_b \notin Tc)\}$. Notice that this is a symmetric relation.

Proposition 2: Let t_a, t_b be two transitions in T such that $t_a \prec t_b$; then $[t_a, t_b]$ if $t_a \prec t_b$ or $t_b \prec t_a$ or $(t_a, t_b) \in Tc$.

Proof: On the one hand, the fact that $t_a \prec t_b$ or $t_b \prec t_a$ implies that there must be a cyclic subsequence including both t_a and t_b, since they belong to a t-invariant (*Property 1*); thus since they have been observed consecutively, there exists one place between them for assuring the consecutive firing. On the other hand, the Tc relation clearly states this dependence. ■

Then, the set of transitions pairs in a causal relation in S is defined as: $CausalR = \{(t_a, t_b) | (t_a \prec t_b \wedge (t_a \prec t_b \vee t_b \prec t_a)) \vee (t_a, t_b) \in Tc\}$.

From the sequence in *Example 1*, $ConcR = \{(t_3, t_4), (t_4, t_3)\}$ and $CausalR = \{(t_1, t_2), (t_2, t_3), (t_4, t_1), (t_4, t_5), (t_5, t_6), (t_6, t_7), (t_7, t_4), (t_2, t_4), (t_3, t_1)\}$.

It is possible that several transition pairs in Seq cannot be classed as causal or as concurrent, for example (t_3, t_5). Such pairs, contained in $\text{Seq}' = ((\text{Seq}\backslash CausalR)\backslash ConcR$, will be treated later.

Remark 1: The computational complexity of finding the previous relation is $O(|S|)$ for the sequential and repetitive dependence relations; the complexity for computing causal and concurrent relations is $O(|T|^2)$.

V. Computing the T-Invariants

Based on the previous definitions and properties, a technique for determining the t-invariants is proposed. A first approximation to the supports of the t-invariants is $\text{Rd}(t_j)$ (*Property 1*). The main challenge is to discover the t-invariants whose transitions appear interleaved in S. In particular, complex situations can appear in which a t-invariant needs for its execution; the firing of other transitions not included in it when two or more t-invariants share transitions.

The technique presented herein pursues to find out the t-invariants of the net that reproduces the observed behavior given by S.

A. Extending the Repetitive Dependencies

For this purpose, it is necessary to extend the Rd sets to obtain the supports of the invariants, by using additional notions introduced below.

Definition 10: A transition t_a is *indirect repetitive dependent* of t_c denoted as $t_a \prec\prec t_c$ iff there is a transition t_b such that $(t_a \prec t_b)$ and $(t_b \prec t_c)$. Therefore, the indirect-repetitive-dependent set is $\text{IRd}(t_a) = \{t_c | t_a \prec\prec t_c\}$. The *transitive extension* of a $\text{Rd}(t_a)$ is $\text{Rd}_{\text{ex}}(t_a) = \text{Rd}(t_a) \cup \text{IRd}(t_a)$.

Notice that the $\text{IRd}(t_a)$ contains the transitions related indirectly at one step.

Property 2: All the transitions in a $\text{Rd}_{\text{ex}}(t_a)$ belong to the support of a t-invariant.

Proof: It follows from *Property* 1 and *Definition 10*; if the firing of t_b is conditioned to the firing of t_c, and $t_a \prec t_b$, then the firing of t_a is also conditioned to the firing of all the $t_c \in \text{Rd}(t_b)$, even if t_c does not always appears between two occurrences of t_a ($t_c \notin \text{Rd}(t_a)$). ■

Rd_{ex} sets approximate the supports of t-invariants; thus, it is necessary to enlarge these sets. For this purpose, relevant Rd_{ex} have to be handled.

Definition 11: A $\text{Rd}_{\text{ex}}(t_j)$ set is said to be maximal iff there is no other $\text{Rd}_{\text{ex}}(t_k)$ that includes $\text{Rd}_{\text{ex}}(t_j)$. $\text{RdM} = \{\text{RdM}_i | \text{RdM}_i \text{ is a maximal } \text{Rd}_{\text{ex}}(t_j)\}$.

Example 2: Consider the set of tasks $T = \{t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$ and the sequence $S_1 = t_6\ t_1\ t_7\ t_4\ t_6\ t_1\ t_7\ t_4\ t_6\ t_1\ t_2\ t_4\ t_6\ t_0\ t_7\ t_3\ t_6\ t_0\ t_2\ t_3\ t_6\ t_1\ t_7\ t_4\ t_6\ t_1\ t_7\ t_4\ t_6\ t_0\ t_2\ t_3\ t_6\ t_1\ t_2\ t_4\ t_6\ t_0\ t_2\ t_3\ t_6\ t_1\ t_2\ t_4\ t_6\ t_1\ t_2\ t_4\ t_6\ t_0\ t_2\ t_3\ t_6\ t_0\ t_7\ t_3\ t_6\ t_0\ t_7\ t_5\ t_4\ t_6\ t_0\ t_7\ t_3\ t_6\ t_0\ t_5\ t_2\ t_4\ t_6\ t_0\ t_5\ t_7\ t_4\ t_6\ t_1\ t_7\ t_4\ t_6\ t_1\ t_2\ t_4\ t_6\ t_0\ t_5\ t_2\ t_4\ t_6\ t_1\ t_7\ t_4\ t_6\ t_1\ t_2\ t_4\ t_6\ t_0\ t_2\ t_3\ t_6\ t_1\ t_2\ t_4\ t_6\ t_1\ t_2\ t_4\ t_6\ t_1\ t_2\ t_4\ t_6\ t_1\ t_2\ t_4\ t_6\ t_1\ t_2\ t_4\ t_6\ t_0\ t_5\ t_7\ t_4\ t_6\ t_1\ t_2\ t_4\ t_6\ t_0\ t_7\ t_3\ t_6\ t_0\ t_5\ t_2\ t_4\ t_6\ t_0\ t_2\ t_5\ t_4\ t_6\ t_1\ t_7\ t_4\ t_6\ t_1\ t_7\ t_4\ t_6\ t_0\ t_7\ t_3\ t_6\ t_1\ t_7\ t_4\ t_6\ t_0\ t_7\ t_5\ t_4\ t_6\ t_1\ t_2\ t_4\ t_6\ t_0\ t_5\ t_7\ t_4\ t_6\ t_0\ t_7\ t_3\ t_6\ t_0\ t_2\ t_5\ t_4\ t_6\ t_0\ t_5\ t_7\ t_4\ t_6\ t_0\ t_2\ t_5\ t_4\ t_6\ t_1\ t_7\ t_4\ t_6\ t_0\ t_7\ t_3\ t_6\ t_0\ t_2\ t_5, \ldots$, where $|S| = 200$.

The repetitive dependencies computed from S are: $\text{Rd}(t_0) = \{t_0, t_6\}$, $\text{Rd}(t_1) = \{t_1, t_6, t_4\}$, $\text{Rd}(t_2) = \{t_2, t_6\}$, $\text{Rd}(t_3) = \{t_3, t_0, t_6\}$, $\text{Rd}(t_4) = \{t_4, t_6\}$, $\text{Rd}(t_5) = \{t_5, t_6, t_4, t_0\}$, $\text{Rd}(t_6) = \{t_6\}$, $\text{Rd}(t_7) = \{t_7, t_6\}$; the transitive extension does not modify these sets, i.e., $\text{Ird}(t_i) = \text{Rd}(t_i)$.

TABLE I
Relations Between Tasks in *Example 2*

T_i	*CausalR* ($\bullet, t_j$)	*Seq′* ($\bullet < t_j$)	$\bullet >< t_j$
t_0	t_5	t_2, t_7	$t_1, t_3, t_4\ t_6$
t_1		t_2, t_7	$t_0, t_1, t_3, t_4, t_5, t_6$
t_2		t_3, t_4	t_1, t_0, t_7, t_6
t_3	t_6		$t_0, t_1, t_2, t_4, t_5, t_7$
t_4	t_6		$t_0, t_1, t_2, t_3, t_5, t_7$
t_5	t_4		$t_1, t_0, t_3, t_4\ t_6$
t_6	t_1, t_0		$t_7, t_2, t_3, t_4\ t_5$
t_7		t_3, t_4	t_1, t_0, t_2, t_6

The computed RdM_i are: $\text{RdM}_1 = \{t_5, t_6, t_4, t_0\}$, $\text{RdM}_2 = \{t_1, t_6, t_4\}$, $\text{RdM}_3 = \{t_3, t_6, t_0\}$, $\text{RdM}_4 = \{t_7, t_6\}$, $\text{RdM}_5 = \{t_2, t_6\}$. Other relations deduced from S are summarized in Table I.

The knowledge of transitions that belong only to one RdM_i will be useful for determining the invariants.

Definition 12: The set of transitions that belong to only one RdM_i is $T_{\text{RdM}_i} = \text{RdM}_i \bigcup_{j=1}, j \neq i^r \text{RdM}_j$, where $r = |\text{RdM}|$.

Now it is possible to enlarge these sets by merging RdM_i that share common transitions. This can be done when the RdM_i fulfill several conditions stated below.

Proposition 3: All the transitions in a $\text{RdM}_{x,y} = \text{RdM}_x \cup \text{RdM}_y$ are included in the support of a t-invariant if there exist $t_i \in \text{RdM}_x$ and $t_j \in \text{RdM}_y$ such that: 1) $(t_i, t_j) \in ConcR$ and 2) $\text{Rd}(t_i) \cap \text{Rd}(t_j) \neq \emptyset$.

Proof: Let be $t_k \in \text{Rd}(t_i) \cap \text{Rd}(t_j)$. Since $(t_i, t_j) \in ConcR$, the subsequence $t_i t_j \ldots t_k \ldots t_j t_i \ldots t_k \ldots t_i t_j \ldots t_k$ is found in S; that is, both transitions t_i and t_j appear between the occurrences of t_k. Therefore, t_i and t_j belong to a largest repetitive dependence $\text{RdM}_{x,y} = \text{RdM}_x \cup \text{RdM}_y$, which is part of the support of a t-invariant.

The next procedure obtains RdM^+, the set of extensions of RdM_i by performing the union operation between members of RdM. ■

Algorithm 1: Merging *RdMs*

Input: $RdM = \{RdM_1, RdM_2 \ldots RdM_r\}$
Output: RdM^+

1. $RdM^+ \leftarrow RdM$
2. $\forall (t_i, t_j) \in ConcR$
   If $Rd(t_i) \cap Rd(t_j) \neq \emptyset$ then
   $RdM_{i,j} \leftarrow RdM_i \cup RdM_j$
   $RdM^+ \leftarrow RdM^+ \cup \{RdM_{i,j}\}$

After applying this procedure to RdM obtained in *Example 2*, given that $(t_5 || t_7)$, $(t_2 || t_5)$, and $\text{Rd}(t_5) \cap \text{Rd}(t_7) \neq \emptyset$ and $\text{Rd}(t_5) \cap \text{Rd}(t_2) \neq \emptyset$, two new maximal sets are obtained: $\text{RdM}_{1,4} = \text{RdM}_1 \cup \text{RdM}_4$, $\text{RdM}_{1,5} = \text{RdM}_1 \cup \text{RdM}_5$.

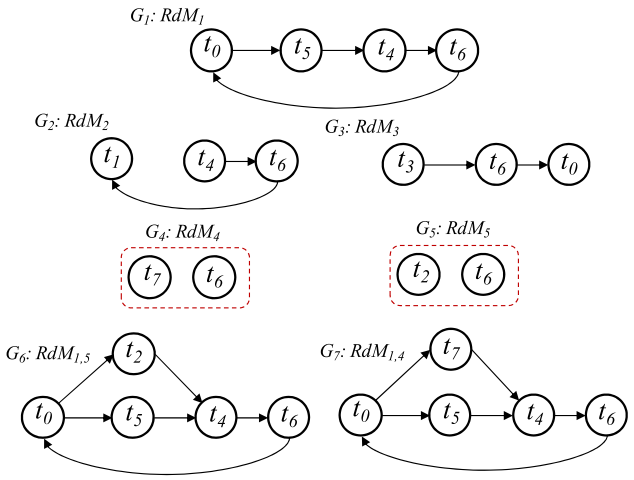


Fig. 5. CG corresponding to RDM^+ of *Example 2*.

Then, $\text{RdM}^+ = \{\text{RdM}_1, \text{RdM}_2, \text{RdM}_3, \text{RdM}_4, \text{RdM}_5, \text{RdM}_{1,4}, \text{RdM}_{1,5}\}$.

Remark 2: The computational complexity of obtaining RdM+ by applying *Algorithm* 1 is $O(|ConcR|)$.

B. Finding the Repetitive Behavior

A t-invariant induces a subgraph of the PN model, called repetitive component or t-component. In the case of a deadlock-free and 1-bounded PN, the t-component is strongly connected (Sc). We will analyze each of the RdM_i in RdM^+ through a graph representation of CausalR and the transition pairs in Seq'.

Definition 13: The Graph of causality relations between tasks, named causality graph of a RdM_i, is a digraph denoted G_i, defined as follows:

$$G_i = (V_i, E_i); \quad V_i = \{t_k | t_k \in \text{RdM}_i\}$$
$$E_i = \{(t_k, t_l) \in V_i \times V_i | (t_k, t_l) \in CausalR \cup \text{Seq}'\}$$

The set of causality graphs corresponding to RdM^+ is denoted $\text{CG} = \{G_1, G_2 \ldots G_q\}$, where G_i is the causality graph of a RdM_i. A G_i is maximal iff there is not a $G_k, \in \text{CG}$ such that $G_i \subset G_k$.

The set CG corresponding to the RdM^+ computed before for *Example* 2 is shown in Fig. 5.

Theorem 1: Let G_i be a causality graph in CG. If a maximal G_i is Sc, then the transitions represented by its vertices are the support of some minimum t-invariant of the PN that reproduces S.

Proof: The vertices of G_i correspond to a RdM_i whose transitions are included in the support of a t-invariant Y_i (*Proposition 3*). Suppose that the transitions in V_i are not the support of a t-invariant; then, there exists at least a $t_k \notin V_i$ such that $t_k \in \langle Y_i \rangle$ that must fire to allow the repetitive firing of transitions in V_i together with t_k; thus, there are not cycles containing t_k in G_i; consequently, it is not Sc or it is not maximal. ■

If the connectivity test is applied to the graphs in CG, it may occur that some G_i are not Sc. Then, it is possible to obtain larger graphs by merging G_i with common vertices, through a merging operation of graphs defined below. Notice that for the

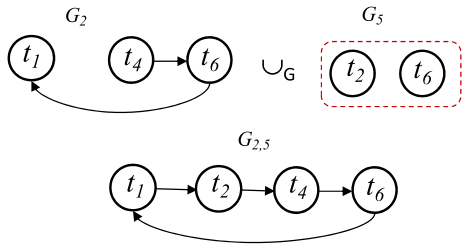


Fig. 6. $G_2 \cup_G G_5$, where $(t_1, t_2), (t_2, t_4) \in CausalR \cup \text{Seq}'$.

RdMs involved in the merging (RdM_1, RdM_4, RdM_5), RdM_1 is Sc, as well as their unions ($\text{RdM}_{1,4}$, $\text{RdM}_{1,5}$); therefore, we can discard RdM_1 as a t-invariant since we are looking for the maximal. On the other hand, RdM_2, RdM_3, RdM_4, and RdM_5 are not Sc; these RdM need to be specially treated with the procedure given below.

Definition 14: The merging operation ($\cup_G$) of two causality graphs $G_i \cup_G G_j$ produces a new graph $G_{i,j}$

$$G_{i,j} = (V_{i,j}, E); \quad V_{i,j} = \{t_k | t_k \in V_i \cup V_j\}$$
$$E = \{(t_k, t_l) \in V_{i,j} \times V_{i,j} | (t_k, t_1) \in CausalR \cup \text{Seq}'\}.$$

Fig. 6 shows the merging of the graphs G_2 and G_5. The idea is to merge iteratively graphs $G_i, G_j \in GC$ such that $V_i \cap V_j \neq \emptyset$. In each iteration, every $G_{i,j}$ produced must not include other Sc graphs. Based on this strategy, a procedure for computing all the Sc graphs from CG is presented below.

Algorithm 2: Getting the t-invariants from S

Input: $CG = \{G_1, G_2 \ldots G_q\}$
Output: $Y(S)$: Supports of t-invariants

1. $G_{Sc} \leftarrow$ *all maximal Sc* $G_i \in CG$
2. $G_{NSc} \leftarrow$ *all non-Sc* $G_i \in CG$
3. $NewNSc \leftarrow G_{\text{NSc}}$, $TempNSc \leftarrow \emptyset$, $\text{Y}(S) \leftarrow \emptyset$
4. While $(NewNSc \neq \emptyset)$
   - 4.1 $\forall G_i \in G_{NSc}$
     - 4.1.1 $\forall G_j \in NewNSc | G_i \not\subset G_j$ and $G_i \cap G_j \neq \emptyset$
       - *a)* $G_{i,j} \leftarrow G_i \cup_G G_j$
       - *b) If* $G_{i,j}$ *is Sc*
         then $G_{Sc} \leftarrow G_{Sc} \cup G_{i,j}$
         else $TempNSc \leftarrow TempNSc \cup \{G_{i,j}\}$
   - 4.2 $NewNSc \leftarrow TempNSc$; $TempNSc \leftarrow \emptyset$
5. $\forall G_i \in G_{Sc}$,
   - 5.1 $< Y_i > \leftarrow V_i$, $\text{Y}(S) \leftarrow \text{Y}(S) \cup < Y_i >$
6. Return $\text{Y}(S)$

Remark 3: This algorithm terminates when no new graph union operation can be applied (i.e., $NewNSc = \emptyset$). The computational complexity of finding the supports of t-invariants when any union $G_{i,j}$ is Sc at each iteration (worst case) is $O(2^{|G_{\text{Nsc}}|})$, where $|G_{\text{NSc}}| < |T| \ll |S|$. However, the worst case is unlikely since when $G_{i,j}$ are built at the first iteration (step 4.1.1 *a*); in practice, at least, one $G_{i,j}$ is *Sc*, then the size of NewNSc decrease; this causes that the algorithm terminates fastly in the next iterations. Furthermore, the conditions in step 4.1.1 reduce the number of graphs unions to be performed.

The worst-case scenario for the previous algorithm is the case in which the process performs a lot of choices, resulting in a large number of t-invariants in the discovered model. This behavior in the log leads the procedure to generate several single node graphs, which imply to perform a lot of unions of graphs to obtain the final t-invariants. However, this particular behavior occurs rarely in automated manufacturing systems.

Theorem 2: Algorithm 2 obtains all the supports of the minimal t-invariants of a PN model that reproduces the task sequence S.

Proof: This procedure performs exhaustively the union of graphs which are not Sc and have common vertices. In each iteration, the formed Sc graphs are no longer considered in the union operations; this reduces progressively the number of non-Sc graphs. Since it is avoided using the already obtained Sc graphs; this guarantees finding minimal Sc graphs and then the support of minimal invariants. When it is not possible to generate new Sc graphs, the procedure stops. Every V_i of G_i in G_{sc} is the support of a t-invariant. ■

The set of obtained t-invariants is $Y(S) = \{Y_i | Y_i$ is the vector corresponding to $V_i\}$.

When Algorithm 2 is applied to CG of *Example* 2, the resulting supports of t-invariants are $\langle Y_1\rangle = \{t_0, t_4, t_5, t_6, t_7\}$, $\langle Y_2\rangle = \{t_0, t_4, t_5, t_6, t_2\}$, $\langle Y_3\rangle = \{t_1, t_4, t_6, t_2\}$, $\langle Y_4\rangle = \{t_1, t_4, t_6, t_7\}$, $\langle Y_5\rangle = \{t_0, t_3, t_6, t_2\}$, $\langle Y_6\rangle = \{t_0, t_3, t_6, t_7\}$.

VI. Building the PN Model

Causal relations $[t_i, t_j]$ determine the existence of a place between transitions. Using this basic structure, named dependence, and the knowledge of t-invariants, a technique for building a PN model is now presented.

A. Merging Transitions of Dependencies

All the transitions named t_i within several dependencies must be merged into a single one following the next merging rules.

Rule 1: Two dependencies in the form $[t_i, t_j]$ and $[t_j, t_k]$ produce, straightforward, a sequential substructure including two places, which allows the firing of the sequence $t_i t_j t_k$, as illustrated in Fig. 4(a).

Rule 2: When the first transitions in two dependencies are the same ($[t_i, t_j]$ and $[t_i, t_k]$), two possible substructures can be created [Fig. 4(b)].

a) The places of the dependencies are merged into a single one iff t_j and t_k belong to different t-invariants. This is denoted as $[t_i, t_j + t_k]$. This rule applies most of the time, but a special situation could appear when $t_j || t_k$; in this case, the dependence $[t_i, t_j + t_k]$ is not created.

b) The places of the dependencies are not merged iff t_j and t_k belong to the same t-invariant. This is denoted as $[t_i, t_j || t_k]$.

Similarly, for dependencies having a common second transition ($[t_i, t_k]$ and $[t_j, t_k]$), the substructure created will be either $[t_i + t_j, t_k]$ or $[t_i || t_j, t_k]$ [Fig. 4(b)]. In both cases, the observations (t_i, t_j), (t_i, t_k), (t_i, t_k), $(t_j, t_k) \in$ Seq, deriving the dependencies, are preserved.

This merging rule is illustrated in Fig. 7. In general, a set of dependencies in the form $\{[t_i, t_j], [t_i, t_k], \ldots [t_i, t_r]\}$ may produce either $[t_i, t_j + t_k + \cdots + t_r]$ or $[t_i, t_j || t_k || \ldots || t_r]$ according to the relations between transitions whether $t_j, t_k, \ldots, t_r$ belong to different t-invariants or $t_j, t_k, \ldots, t_r$ belong to the same t-invariant, respectively.

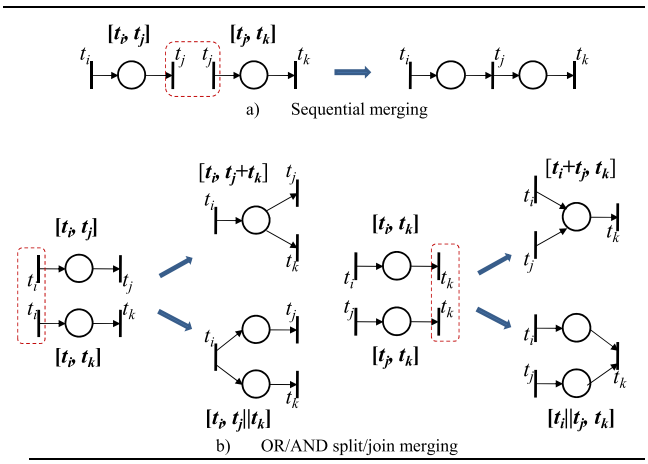


Fig. 7. Rules for merging dependencies. (a) Sequential merging. (b) OR/AND split/join merging

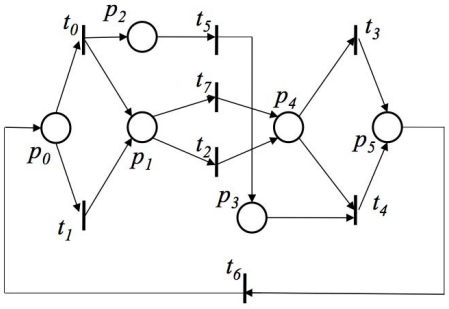


Fig. 8. N_1 built from S of *Example 2*.

Consequently, the merging can be applied to composed dependencies that coincide with one expression of transitions of type $t_i + t_j$ or $t_i || t_j$; for example, $[t_i + t_j, t_k]$ and $[t_i + t_j, t_r]$ leads to $[t_i + t_j, t_k + t_r]$ if both t_k and t_r do not belong to the same invariant.

The application of these merging rules to the dependencies derived from the pairs in $Causal\,R \cup \text{Seq}'$, leads to a PN model N_1 including all the transitions.

In *Example 2*, the application of rules 1 and 2 to the obtained relations in $Causal\,R \cup \text{Seq}'$ of Table I yields the set of composed dependencies: $[t_5, t_4]$, $[t_0, t_2 || t_5]$, $[t_0, t_5 || t_7]$, $[t_0, t_2 + t_7]$, $[t_1, t_2 + t_7]$, $[t_2, t_3 + t_4]$, $[t_7, t_3 + t_4]$, $[t_4 + t_3, t_6]$, $[t_6, t_0 + t_1]$, $[t_0 + t_1, t_2]$, $[t_0 + t_1, t_7]$, $[t_2 || t_5, t_4]$, $[t_7 || t_5, t_4]$, $[t_7 + t_2, t_3][t_7 + t_2, t_4]$. Afterward, the obtained dependencies are $\boldsymbol{p_0}$: $[t_6, t_1 + t_0]$, $\boldsymbol{p_1}$: $[t_0 + t_1, t_2 + t_7]$, $\boldsymbol{p_1}$–$\boldsymbol{p_2}$: $[t_0, (t_2 + t_7) || t_5]$, $\boldsymbol{p_3}$: $[t_5, t_4]$, $\boldsymbol{p_4}$: $[t_2 + t_7, t_4 + t_3]$, $\boldsymbol{p_5}$: $[t_4 + t_3, t_6]$. The sequential merging of substructures of the dependencies' yields the PN model N_1 shown in Fig. 8.

B. Model Adjustment

Although S is executed in N_1 most of the times, the obtained model could not fire S, or could fire S but also exceeding sequences. The PN in Fig. 8 does not reproduce S of *Example* 2, in particular, the subsequences $t_1 t_2 t_4$ and $t_1 t_7 t_4$

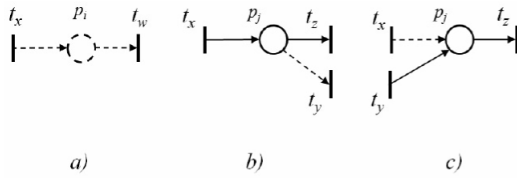


Fig. 9. Case of implicit dependencies.

cannot occur in N_1. This is because the computed t-invariants $Y(S)$ differ from those of N_1, named $J(N_1)$. If $Y(S)$ coincide with $J(N_1)$, then N_1 is the correct model; otherwise, it must be adjusted.

The mismatching between $Y(S)$ and $J(N_1)$ is due to the fact that the computed model does not include PN elements (places and arcs) which assure implicit behaviors not exhibited in S, named implicit dependencies.

Definition 15: In a 1-bounded PN, $[t_i, t_j]$ is called an *implicit* dependence, if although there is a place between the transitions, the firing of t_i does not produce a marking that enables t_j. It is necessary the firing of at least one transition before t_j.

In a PN model, implicit dependencies represent the record of the occurrence of a t_i, which is used as condition to enable a future event t_j. In general, an implicit dependence represents a constraint in the flow of tokens in the net by assuring that t_j is fired only when t_i is fired before; otherwise, the absence of such a dependence will allow the firing of exceeding sequences in the model.

After the building of N_1, it is possible to characterize two cases for which N_1 may exhibit implicit dependencies: *Case 1)* as a new place between two transitions Fig. 9(a), and *Case 2)* through an already computed place Fig. 9(b) and (c).

When $Y(S) \neq J(N_1)$, N_1 must therefore be adjusted by finding the pertinent implicit dependencies that extend N_1 into N_2, whose t-invariants agree with $Y(S)$. If $Y(S) \subset J(N_1)$ we use the *Case 1* as a strategy to solve the mismatch, in the other hand if $Y(S) \not\subset J(N_1)$, i.e., $\exists Y_i \in Y(S)$ such that $Y_i \notin J(N_1)$ we use *Case 2*. The handling of each case is described next.

Case 1: In this case, N_1 has more invariants than those computed from S; thus, it represents an exceeding behavior. A new place between two transitions t_i and t_j has to be added to N_1 in order to constrain the differed firing of t_j after the firing of t_i.

Proposition 4: A dependence $[t_i, t_j]$ must be added to N_1 if the following condition holds: i) $(t_i >< t_j)$; ii) $(t_i, t_j \in \langle Y_k \rangle)$; and $\exists T_{\mathrm{RdMx}}$ s.t. $t_i, t_j \in T_{\mathrm{RdMx}}$.

Proof: Suppose that the place p_k in the dependence $[t_i, t_j]$ must not be added; this is because of the following.

i) The place p_i $[t_i, t_j]$ already exists as result of applying *Rule* 1 or *Rule 2*, Therefore, such transitions are observed consecutively in *S,* i.e., $\neg\ (t_i >< t_j)$.
ii) It is not necessary to force t_i to appear always before t_j, then both transitions may fire independently because they belong to different t-invariants, i.e., there is not a *support of t-invariant* which contains both transitions, i.e., $\neg\ (t_i, t_j \in \langle Y_k \rangle)$ or $\neg\ (t_i, t_j \in T_{\mathrm{RdMx}})$. ■

Case 2: Let $J(N_1) = \{J_1, J_2, \ldots, J_r\}$ be the set of t-invariants of N_1, such that $\mathrm{C}J_j = 0$, where C is the incidence matrix of N_1. Consider a $Y_r \notin J(N_1)$. Let p_k be the place corresponding to the row in which $\mathrm{C}Y_r \neq 0$ (i.e., $C(p_k)$ $Y_r \neq 0$). In order to obtain the dependence $[t_i, t_j]$, other transition in N_1 must be linked through p_k to one of the transitions in ${}^\bullet p_k$ or $p_k^\bullet$ according to the following rule.

Proposition 5: Let t_i, $t_j \in \langle Y_r \rangle$ be transitions such that at least one belongs to some T_{RdMx}, and they are not observed consecutively $(t_i >< t_j)$ such that $\mathrm{C}Y_r \neq 0$, where $Y_r \in Y(S)$. Then, there exists an implicit dependence $[t_i, t_j]$ that must be added through a place p_k of N_1, such that $C(p_k)Y_r \neq 0$, in order to ensure $C(p_k)Y_r = 0$.

Proof: As stated in Proposition 4, $[t_i, t_j]$ must be added; however, there exist a place in N_1 which must be used. Given that the structure of N_1 is an ordinary PN, only two cases can occur.

i) $C(p_k)Y_r = 1$. This requires that $C(p_k, t_j) = -1$ to get $C(p_k)Y_r = 0$; thus, $t_i \in^\bullet p_k$. (The arc (p_k, t_j) must be added to obtain $[t_i, t_j]$.)
ii) $C(p_k)Y_r = -1$. This requires that $C(t_i, p_k) = 1$ to get $C(p_k)Y_r = 0$; thus $t_j \in p_k^\bullet$ (The arc (t_i, p_k) must be added to obtain $[t_i, t_j]$.)

Since $t_i \in T_{\mathrm{RdMx}}$, the added arc (p_k, t_i) only affect Y_r. Similarly, since $t_j \in T_{\mathrm{RdMx}}$, the new arc (t_j, p_k) does not affect the other t-invariants. ■

Proposition 6: Let $J(N_1)$ and $Y(S)$ be the t-invariants sets of N_1 and S, respectively, such that $J(N_1) \neq Y(S)$. N_1 can be adjusted just by the addition of implicit dependencies in *Case 1* and *Case 2,* in order to fit N_1 into the t-invariants of $Y(S)$ $(J(N_1) = Y(S))$.

Proof: Let $n_i = N_1(Y_i)$ be the induced component by $\langle Y_i \rangle$; then we need to ensure that n_i is a t-component, i.e., $\forall s_j \in n_i$, $|s_j^\bullet| = |^\bullet s_j| = 1$. If the condition is fulfilled, then Y_i is a valid t-invariant in N_1. We could reduce the excessive behavior by adding an implicit dependence *Case 1* between a couple of transitions in Y_i according to *Proposition 4*. If $|s_j^\bullet| \neq |^\bullet s_j|$, it is because some either $|s_j^\bullet| = 0$ or $|^\bullet s_j| = 0$; that is, there exist a place in n_i without output or input transition. To solve this, an implicit dependence of *Case 2 i)* or *ii)* must be added, respectively, to s_j according to *Proposition 5.* ■

Corollary 1: If all the implicit dependencies added $\forall p_k$ such that $C(p_k)Y_r \neq 0$, $\forall Y_r \in Y | \mathrm{C}Y_r \neq 0$, then the resulting net N_2 fulfills $\mathrm{CY} = 0$.

Proof: When all the corrections to N_1 are done by applying the corrections of Proposition 5, the resulting net N_2 fulfills $\mathrm{CY}(S) = 0$ and then $Y(S) = J(N_2)$. ■

Remark 4: The complexity of computing the implicit dependencies is $O(|P| \cdot |T|)$; it is related to the matrix-vector product operation $C(p_k)Y_i$. Notice that the procedure does not need to compute the t-invariants of N_1. It only operates on the computed invariants $Y(S)$ that do not agree with the computed net N_2 $(Y_r \in Y(S)$ such that $Y_r C \neq 0)$

Let us analyze N_1 in Fig. 8, obtained from S in Example 2. First, it is computed $J(N_1) = \{\langle J_1 \rangle, \langle J_2 \rangle, \langle J_3 \rangle, \langle J_4 \rangle\}$; $\langle J_1 \rangle = \{t_0, t_4, t_5, t_6, t_7\}$, $\langle J_2 \rangle = \{t_0, t_4, t_5, t_6, t_2\}$, $\langle J_3 \rangle = \{t_1, t_2, t_3, t_6\}$,

Algorithm 3: Finding Implicit Dependencies

Input: $N_1, \mathrm{Y}(S)$
Output: N_2

1. $\forall (t_i, t_j) | t_i, >< t_j \wedge \exists T_{RdMx}$ s.t. $t_i, t_j \in T_{RdMx} \wedge t_i, t_j \in <\mathrm{Y_i}>$
   add a place between (t_i, t_j)
2. *If* $\exists \mathrm{Y_i} \in \mathrm{Y}(S) | \mathrm{CY_r} \neq 0$
   a) Find a $p_k | \mathrm{C}(p_k)\mathrm{y_i} \neq 0$
   b) Add $[t_i, t_j]$ through p_k relations that fulfil
   $t_i, >< t_j \wedge t_i, t_j \in \mathrm{y_i} \wedge (t_i \in^{\bullet} p_k, t_j \in T_{RdMx})$ or
   $t_i, >< t_j \wedge t_i, t_j \in \mathrm{y_i} \wedge (t_j \in p_k^{\bullet}, t_i \in T_{RdMx})$

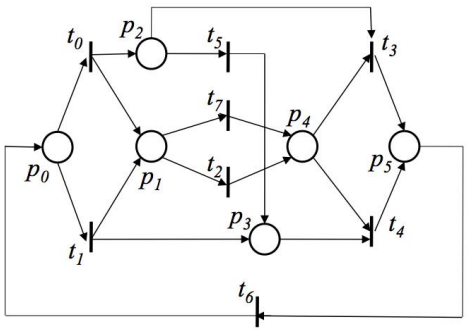


Fig. 10. Resulting PN N_2 after model adjustment.

$\langle J_4 \rangle = \{t_1, t_7, t_3, t_6\}$. There is a mismatch between both sets and since $Y(S) \not\subset J(N_1)$, the problem is handled as in *Case 2*. It can be noticed that Y_3, Y_4, Y_5, $Y_6 \notin J(N_1)$. In the analysis of Y_3, $p_k = p_3$ because it fulfills the condition $C_{N1}(p_3)Y_i \neq 0$, as shown in the next equation

$$\begin{bmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & -1 & 0 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & -1 & 0 \end{bmatrix} \bullet \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \end{bmatrix}$$

C_{N1} Y_3

The transition in $t_4 \in p_k^{\bullet}$ is chosen to find the implicit dependence $[t_i, t_4]$. The transition that fulfills the conditions $t_i >< t_4$, t_i, $t_4 \in \langle Y_3 \rangle$, $t_i \in T_{\mathrm{RdMx}}$, is t_1; therefore, the implicit dependence $[t_1, t_4]$ is added to N_1 by the corresponding arc (t_1, p_3). Similarly, Y_4, Y_5, Y_6 are treated and the implicit dependence $[t_0, t_3]$ in p_2 is found. Finally, the resulting PN model N_2, which exactly reproduces S is shown in Fig. 10.

Theorem 3: Given a sequence of transitions $S \in T^*$ obtained from a process fulfilling the assumptions stated in Section II-A, a 1-bounded PN model N_2 that reproduces S can be obtained by applying the rules 1 and 2, and performing the adjustments of *Algorithm 3*.

Proof: Causality between transitions, established by the pairs in $Causal\,R \cup \mathrm{Seq}'$, represents the precedence relationship between consecutive transitions in S that are not in ConcR. The substructure associated with a dependence $[t_i, t_j]$ guarantees the consecutive firing of these transitions; thus by applying *Rule 1,* the flow expressed in $Causal\,R \cup \mathrm{Seq}'$ is fulfilled by N_1. Furthermore, *Rule 2* determines, by the knowledge of the t-invariants, whether the flow is split or joint in choice or parallel structures. Dependencies involving transitions included in Sc causality graphs assure the construction of repetitive components in N_1. Furthermore, adjustments to N_1 provided by *Propositions* 4 and *5* allow fitting the invariants computed form the observed behavior with those of the discovered model. ■

C. Initial Marking

The initial marking must enable S; thus, the procedure for determining M_0 is simple; it suffices: 1) to place tokens in the input places of the first transition in S and 2) executing the remainder t_j in S and eventually adding tokens in some places of ${}^{\bullet}t_j$ when the reached marking is not enough for firing t_j. In the case of *Example* 2, the only place initially marked in the PN (Fig. 5) is p_5.

D. Processing Several Event Sequences

Although the presentation of the method only a single sequence is used, this discovery technique may handle several event sequences S_i corresponding to the observed behavior of the same discrete-event process. The only constraint is that all the sequences must be sampled from the starting of the process. All the observed precedence relationships in Seq_i of every S_i are gathered into the Seq relation at the beginning of the discovery procedure. The initial marking is determined for enabling all the S_i.

E. Performance of the Method

The method is divided into three main stages. First, the information derived from S, namely, relations between activities are obtained; the computational complexity of this step is $O(|S|)$, as stated in *Remark 1*.

The second stage consists of inferring the supports of the t-invariants from S; the most significant calculations are here:

1) the formation of RdM+ by *Algorithm* 1, which is $O(|ConcR|)$ (*Remark 2*);
2) iterative unions of graphs by Algorithm 2, which is in the worst-case exponential on $|G_{\mathrm{NSc}}|$, but actually faster as stated in *Remark 3*.

It is worth to notice that $|ConcR|$ and $|G_{\mathrm{NSc}}|$ are very small with respect to the length of the observation $|S|$.

Finally, in the stage that builds the PN, the step that requires more calculations is the model adjustment, which is related to the matrix product operation; thus, the complexity is $(O(|P| \cdot |T|))$, which is detailed in *Remark 4.* Nevertheless, this step is not always needed.

It is important to notice that for the matching test of $Y(S)$ performed on the first approximated PN N_1, it is not necessary to obtain the invariants $J(N_1)$; only $C_{N1}Y(S) = 0$ is tested. Furthermore, the method discovers only the invariants executed in the sequence S.

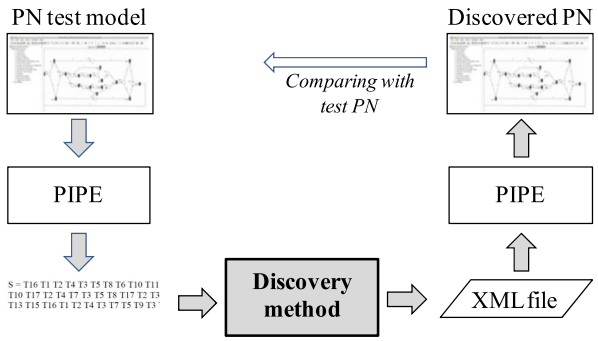


Fig. 11. Testing scheme.

Therefore, the procedure implementing the proposed discovery method can be executed approximately in polynomial time on the size of the log of sequences S, as demonstrated by the test presented in the next section.

VII. Implementation and Tests

Algorithms derived from the proposed method have been implemented as a software tool and tested on numerous examples of diverse complexity. The tests were performed using the following scheme, depicted in Fig. 11: first, a PN test model is designed, and with the help of the PN editor/simulator PIPE [32], a long sequence S is produced. Afterward, the tool processes S and yields a PN coded in XML, which is displayed using PIPE again. Using this scheme, the discovered PN can be easily compared with the test PN.

Below, we present an example regarding a less simple PN model that can be discovered using the proposed PN discovery method. The model in Fig. 12 has been obtained by processing the sequence

$S_3 = T16\,T14\,T2\,T4\,T3\,T5\,T9\,T7\,T3\,T5\,T9\,T3\,T5\,T8\,T17\,T2\,T3\,T5\,T9\,T3\,T4\,T7\,T5\,T8\,T11\,T13\,T15\,T16\,T1\,T2\,T4\,T3\,T5\,T8\,T6\,T10\,T17\,T2\,T3\,T4\,T5\,T6\,T9\,T3\,T5\,T10\,T9\,T3\,T5\,T9\,T3\,T5\,T9\,T3\,T5\,T9\,T3\,T5\,T9\,T3\,T5\,T9\,T3\,T5\,T9\,T3\,T5\,T8\,T17\,T2\,T4\,T3\,T7\,T5\,T9\,T3\,T5\,T9\,T3\,T5\,T8\,T11\,T12\,T15\,T16\,T1\,T2\,T4\,T7\,T3\,T5\,T8\,T11\,T12\,T15\,T16\,T14\,T2\,T3\,T5\,T8\,T4\,T6\,T10\,T17\,T2\,T3\,T4\,T6\,T10\,T5\,T9\,T3\,T5\,T8\,T17\,T2\,T4\,T6\,T3\,T10\,T5\,T9\,T3\,T5\,T8\,T17\,T2\,T3\,T5\,T8\,T4\,T6\,T10\,T11\,T13\,T15\,T16\,T14\,T2\,T4\,T7\,T3\,T5\,T9\,T3\,T5\,T8\,T11\,T13\,T15\,T16\,T1\,T2\,T3\,T5\,T9\,T4\,T6\,T3\,T10\,T5\,T9\,T3\,T5\,T9\,T3\,T5\,T9\,T3\,T5\,T8\,T17\,T2\,T3\,T5\,T9\,T4\,T3\,T6\,T5\,T9\,T3\,T10\,T5\,T8\,T11\,T12\,T15\,T16\,T14\,T2\,T4\,T7\,T3\,T5\,T9\,T3\,T5\,T9\,T3\,T5\,T9\,T3\,T5\,T9\,T3\,T5\,T9\,T3\,T5\,T9\,T3\,T5\,T8\,T17\,T2\,T3\,T5\,T8\,T4\,T6\,T10\,T11\,T13\,T15\,T16\,T1\,T2\,T3\,T5\,T8\,T4\,T7\,T11\,T12\,T15\,T16\,T14\,T2\,T4\,T7\,T3\,T5\,T9\,T3\,T5\,T9\,T3\,T5\,T8\,T17\,T2\,T4\,T6\,T10\,T3\,T5\,T8\,T11\,T13\,T15\,T16\,T1\,T2\,T4\,T3\,T5\,T9\,T7\,T3\,T5\,T8\,T11\,T12\,T15\,T16\,T14\,T2\,T4\,T7\,T3\,T5\,T9\,T3\,T5\,T8\,T17\,T2\,T3\,T4\,T7\,T5\,T9\,T3\,T5\,T8\,T11\,T13\,T15\,T16\,T14\,T2\,T4\,T7\,T3\,T5\,T9\,T3\,T5\,T9\,T3\,T5\,T9\,T3\,T5\,T8\,T11\,T13\,T15\,T16\,T14\,T2\,T3\,T4\,T6\,T5\,T9\,T3\,T10\,T5\,T9\,T3\,T5\,T8\,T11\,T13\,T15\,T16\,T1\,T2\,T4\,T7\,T3\,T5\,T9\,T3\,T5\,T8\,T11\,T12\,T15\,T16\,T1\,T2\,T3\,T4\,T5\,T7\,T9\,T3\,T5\,T9\,T3\,T5\,T8\,T17\,T2\,T3\,T4\,T6\,T10\ldots$, where $|S| = 1500$.

This model includes diverse structures (nested t-components evolving concurrently) which are more complex than others published in literature. As a sign of performance, the processing time for S in a laptop computer (2.4-GHz dual-core, Intel Core i5 processor, 4 GB of 1333-MHz DDR3 memory) was 3.6 s.

Thanks to software tool we developed, it has been possible to test models of diverse structures, which include cycles nested into t-components, concurrence, and implicit dependencies. This reveals the capabilities of the method for dealing with black-box model discovery.

Additional tested examples are presented in the Appendix included as Additional material associated with this paper, which can be also downloaded from [33]; they include part of the sequence processed and the model obtained.

Besides this test scheme, the method has been implemented in the ProM environment, hosted a public site located in the Technical University of Eindhoven. It is an experimental framework devoted to test process mining techniques. This implementation handles the input event logs in both formats: a single long sequence or a set of event traces. Several tests are included in the Appendix [33]; the results are compared with that of two standing process mining techniques applied to sequences used in this paper.

VIII. Concluding Remarks

A. Main Features

The black-box method herein described for discovering ordinary PN processes long sequences S_i, which represent the observed behavior of a discrete-event process during a normal operation functioning. A single very long sequence can be used for deriving the model by considering that all the possible behavior has been observed.

The discovered model is a safe ordinary PN; this subclass is well adapted to represent the behavior of a wide class of discrete-event processes, namely manufacturing systems controllers, where the tasks are represented by state variables having two values: idle (OFF) and active (ON).

The algorithms derived from the steps of the method are polynomial time on the size of S; efficient procedures have been developed and tested on numerous examples including diverse structures.

B. Related Works

This paper improves a previous technique [12], which follows a two-stage approach regarding the reduction of exceeding behavior representation of the nonobservable model. Based on some basic notions, a novel method is proposed; it is supported mainly on searching the t-invariants from the observed sequences S_i. The determined invariants are used first to build an initial model, and later to adjust it, if necessary; the final model includes implicit causal relationships between transitions that have not been observed consecutively.

As pointed out in [12], the method overcomes several limitations of the proposals in the identification problem statement for PN.

The method in [3] and subsequent extensions proposed efficient algorithms; the state equivalence is based on the observation of the same observed outputs vector, which is not very often the case for real systems.

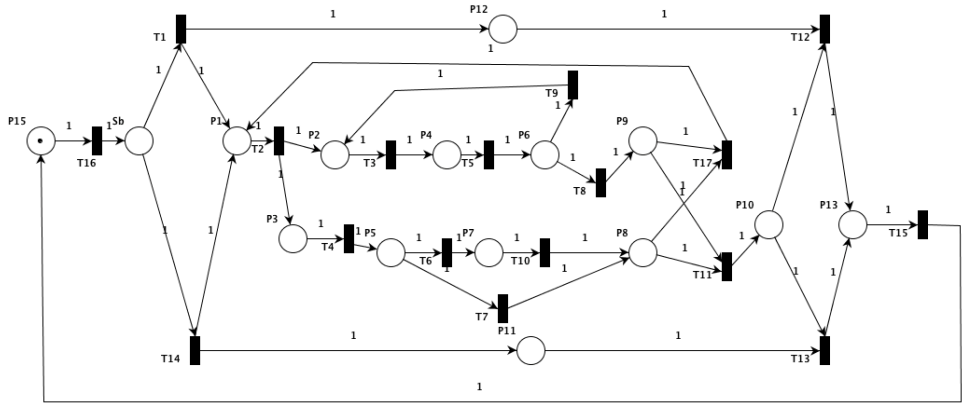


Fig. 12. Nontrivial discovered PN model from S_3.

TABLE II
SUMMARY OF MAIN FEATURES OF IDENTIFICATION AND PROCESS MINING METHODS

	Identification			**Process Mining Discovery**		
	Progressive approach [3]	Black-box approach [12]	T-inv approach [this work]	α algorithm [20]	$\alpha^{\$}$ algorithm [24]	Inductive Miner [26]
Input data	Sequence	Sequence/ traces	Sequence/ traces	traces	traces	traces
Obtained models	Ordinary PN	Ordinary PN	Ordinary PN	WFN	WFN	Process tree
Implicit dependencies	×	×	✓	×	✓	×
Invisible transitions	×	×	×	×	✓	✓
Computational complexity	Polynomial	Polynomial	Polynomial	Exponential	Exponential	Polynomial
1-length loop	×(does not apply)	×(does not apply)	×(does not apply)	×	✓	✓

The techniques based on integer linear programming [5] and their extensions yield accurate bounded PN; but, due to the high computational complexity, they are limited to deal with few short event sequences; also it is necessary to know the number of places in the PN to build.

Previous proposals on identification [16] are constrained to deal with sequences describing cyclic behavior, i.e., the initial and the final states are the same.

As indicated in the introduction, interesting proposals on process discovery are driven form the business process applications, namely the extensions of the alpha algorithm [21]–[23]. The aim of such techniques was similar but both the problem statement and the class of synthesized PN are (currently) different. Thus, comparison with our proposal is difficult; on the one hand, in our approach, the input data do not require the knowledge about the start and end of events in traces σ_j in every S_i. However, we can handle the input data as traces; a sequence S is formed by the concatenation of σ_i using a transition t^* between each trace. On the other hand, the obtained PN are not restricted to workflow nets. Although the examples used in this paper have the form of extended workflow nets, typical models include some transitions executed once after the start and before the end of the observations and the repetitive behavior issued from the middle of the sequence. A summary of the main features of such techniques is provided in Table II.

C. Limitations and Perspectives

The discovered PN executes the sequences S_i from M_0 and may eventually accept exceeding iterative subsequences. This corresponds to the inherent behavior to PN with repetitive components.

Although implementation and tests revealed accuracy and efficiency of the method when complex PN structures were addressed, a metrics for evaluating the exceeding behavior of the discovered model with respect to observed S_i is essential.

The method does not deal with sequences including repeated transitions. Although this is not required for discovering controlled industrial processes, it can be addressed in the near future.

Current research addresses the problem of discovering behaviors of process controllers that involve the use of counters and timers.

ACKNOWLEDGMENT

We would like to thank anonymous reviewers for their critical and constructive comments.

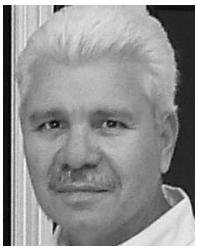
REFERENCES

[1] E. M. Gold, "Language identification in the limit," *Inf. Control*, vol. 10, no. 5, pp. 447–474, 1967.
[2] D. Angluin, "Queries and concept learning," *Mach. Learn.*, vol. 2, no. 4, pp. 319–342, 1988.

[3] M. Meda-Campaña, A. Ramirez-Treviño, and E. López-Mellado, "Asymptotic identification of discrete event systems," in *Proc. 39th IEEE Conf. Decision Control*, Sydney, NSW, Australia, Dec. 2000, pp. 2266–2271.
[4] M. Meda-Campaña and E. López-Mellado, "Identification of concurrent discrete event systems using petri nets," in *Proc. IMACS World Congr. Comput. Appl. Math.*, Dec. 2005, pp. 2266–2271.
[5] M. P. Cabasino, A. Giua, and C. Seatzu, "Identification of Petri nets from knowledge of their language," *Discrete Event Dyn. Syst.*, vol. 17, no. 4, pp. 447–474, 2007.
[6] M. P. Cabasino, A. Giua, and C. Seatzu, "Linear programming techniques for the identification of place/transition nets," in *Proc. 47th IEEE Conf. Decision Control (CDC)*, Dec. 2008, pp. 514–520.
[7] M. Dotoli, M. P. Fanti, A. M. Mangini, and W. Ukovich, "Identification of the unobservable behaviour of industrial automation systems by Petri nets," *Control Eng. Pract.*, vol. 19, no. 9, pp. 958–966, 2011.
[8] S. Klein, L. Litz, and J.-J. Lesage, "Fault detection of discrete event systems using an identification approach," in *Proc. 16th IFAC World Congr.*, Praga, Czech Republic, 2005, pp. 92–97.
[9] M. Roth, S. Schneider, J.-J. Lesage, and L. Litza, "Fault detection and isolation in manufacturing systems with an identified discrete event model," *Int. J. Syst. Sci.*, vol. 43, no. 10, pp. 1826–1841, 2012.
[10] A. P. Estrada-Vargas, E. López-Mellado, and J.-J. Lesage, "Input-output Identification of controlled discrete manufacturing systems," *Int. J. Syst. Sci.*, vol. 45, no. 3, pp. 456–471, 2014.
[11] A. P. Estrada-Vargas, E. Lóez-Mellado, and J.-J. Lesage, "A stepwise method for identification of controlled discrete manufacturing systems," *Int. J. Comput. Integr. Manuf.*, vol. 28, no. 2, pp. 187–199, 2015.
[12] A. P. Estrada-Vargas, E. López-Mellado, and J.-J. Lesage, "A Black-Box identification method for automated discrete-event systems," *IEEE Trans. Autom. Sci. Eng.*, to be pulished, doi: 10.1109/TASE.2015.2445332.
[13] S. O. E. Mehdi, R. Bekrar, N. Messai, E. Leclercq, D. Lefebvre, and B. Riera, "Design and identification of stochastic and deterministic stochastic Petri nets," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 42, no. 4, pp. 931–946, Jul. 2012.
[14] D. M. Muñoz, A. Correcher, E. García, and F. Morant, "Identification of stochastic timed discrete event systems with ST-IPN," *Math. Problems Eng.*, vol. 2014, p. 21, Jul. 2014.
[15] F. Basile, P. Chiacchio, and J. Coppola, "Real time identification of time Petri net faulty models," in *Proc. IEEE Int. Conf. Autom. Sci. Eng. (CASE)*, Gothenburg, Sweden, Aug. 2015, pp. 280–285.
[16] A. P. Estrada-Vargas, E. López-Mellado, and J.-J. Lesage, "A comparative analysis of recent identification approaches for discrete-event systems," *Math. Problems Eng.*, vol. 2010, p. 21, May 2010.
[17] M. P. Cabasino, P. Darondeau, M. P. Fanti, and C. Seatzu, "Model identification and synthesis of discrete-event systems," in *Contemporary Issues in Systems Science and Engineering*. Hoboken, NJ, USA: Wiley, 2015.
[18] R. Agrawal, D. Gunopulos, and F. Leymann, "Mining process models from workflow logs," *Advances in Database Technology—EDBT* (Lecture Notes in Computer Science), vol. 1377. Berlin, Germany: Springer, 1988, pp. 469–483.
[19] J. E. Cook, Z. Du, C. Liu, and A. L. Wolf, "Discovering models of behavior for concurrent workflows," *Comput. Ind.*, vol. 53, no. 3, pp. 297–319, Apr. 2004.
[20] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1128–1142, Sep. 2004.
[21] L. Wen, W. van der Aalst, J. Wang, and J. Sun, "Mining process models with non-free-choice constructs," *Data Mining Knowl. Discovery*, vol. 15, no. 2, pp. 1384–5810, Oct. 2007.
[22] D. Wang, G. Jidong, H. Hao, B. Luo, and L. Huang, "Discovering process models from event multiset," *Expert Syst. Appl.*, vol. 39, no. 15, pp. 1970–11978, 2012.
[23] Q. Guo, L. Wen, J. Wang, Z. Yan, and P. S. Yu, "Mining invisible tasks in non-free-choice constructs," *Bus. Process Manage.*, vol. 9253, pp. 109–125, Aug. 2015.
[24] M. Solé and C. Carmona, "Region-based foldings in process discovery," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 1, pp. 192–205, Jan. 2013.
[25] S. Leemans, D. Fahland, and W. van der Aalst, "Discovering block-structured process models from event logs—A constructive approach," in *Application and Theory of Petri Nets and Concurrency*. Berlin, Germany: Springer, 2013.
[26] W. M. P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. New York, NY, USA: Springer, 2011.
[27] J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst, "On the role of fitness, precision, generalization and simplicity in process discovery," in *On the Move to Meaningful Internet Systems: OTM*, vol. 7565. IEEE, 2012, pp. 305–322.
[28] T. Tapia-Flores, E. López-Mellado, A. P. Estrada-Vargas, and J.-J. Lesage, "Petri net discovery of discrete event processes by computing t-invariants," in *Proc. 19th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Barcelona, Spain, Sep. 2014, pp. 1–8.
[29] J. Baeten and W. Weijland, *Process Algbra*. Cambridge, U.K.: Cambridge Univ. Press, 1990.
[30] A. D. Medeiros, B. V. Dongen, and W. V. D. Aalst, *Process Mining: Extending the Alpha-Algorithm to Mine Short Loops* (Special Issue on Tools for Computer Performance Modelling and Reliability Analysis). Eindhoven, The Netherlands: Eindhoven Univ. Technology, 2004.
[31] N. J. Dingle, W. J. Knottenbelt and T. Suto, "PIPE2: A tool for the performance evaluation of generalised stochastic petri nets," *ACM SIGMETRICS Perform. Eval. Rev. (Special Issue on Tools for Computer Performance Modelling and Reliability Analysis)*, vol. 36, no. 4, pp. 34–39, Mar. 2009. [Online]. Available: http://pipe2.sourceforge.net/
[32] Accessed on Mar. 2017. [Online]. Available: http://www.gdl.cinvestav.mx/elopez/Appendix-PN-discov-T-inv.pdf

Tonatiuh Tapia-Flores received the B.Sc. degree in computational systems from the Instituto Tecnologico de Ocotlán, Ocotlán, Mexico, and the M.Sc. degree in electrical engineering from CINVESTAV Unidad Guadalajara, Zapopan, Mexico, where he is currently pursuing the Ph.D. degree in computer science.

His research interests include data mining, workflow automation, process discovery, identification of discrete-event systems, and formal modeling and analysis using Petri nets.



Ernesto López-Mellado received the B.Sc. degree in electrical engineering from the Instituto Tecnologico de Ciudad Madero, Ciudad Madero, México, in 1977, the M.Sc. degree from CINVESTAV, México City, México, in 1979, and the Docteur-Ingénieur degree in automation from the University of Toulouse, Toulouse, France, in 1986.

He is currently a Professor of Computer Science with CINVESTAV Unidad Guadalajara, Zapopan, Mexico. His research interests include discrete-event systems and distributed intelligent systems.

Ana Paula Estrada-Vargas received the B.Sc. degree in computer engineering from the Universidad de Guadalajara, Guadalajara, Mexico, in 2007, the M.Sc. degree from CINVESTAV, Zapopan, Mexico, in 2009, and the Ph.D. degree from CINVESTAV, Guadalajara, and the ENS de Cachan, Cachan, France, in 2013.

She is currently with the Oracle Semantic Technologies Team, Mexico Development Center, Guadalajara. Her research interests include the identification of discrete-event systems, formal modeling and analysis using Petri nets, and Semantic Web technologies.

Jean-Jacques Lesage received the Ph.D. degree from the Ecole Centrale de Paris, Châtenay-Malabry, France, and the "Habilitation à diriger des recherches" from the University Nancy 1, Nancy, France, in 1989 and 1994, respectively.

He was the Head of the Automated Production Research Laboratory, Ecole Normale Supérieure de Cachan, Cachan, France, for eight years. He is currently a Professor of Automatic Control with the Ecole Normale Supérieure de Cachan. His research interests include formal methods and models for synthesis, analysis and diagnosis of discrete event systems, and applications to manufacturing systems, network automated systems, energy production, and more recently, ambient-assisted living.