Discovering Process Models from Incomplete Event Logs using Conjoint Occurrence Classes

Tonatiuh Tapia-Flores, Edelma Rodríguez-Pérez and Ernesto López-Mellado

CINVESTAV Unidad Guadalajara
Av. del Bosque 1145, Col. El Bajio. 45015 Zapopan Mexico
{ttapia,erodrigue,elopez}@gdl.cinvestav.mx

Abstract. *This paper addresses the problem of discovering a sound Workflow net (WFN) from sample event traces representing the behavior of a process. A current challenging problem deals with discovering a suitable model from incomplete logs since it is not possible to ensure that a log contains all executions of a process. A method that allows building WFN from logs including a reduced number of traces is presented. It is based on the concept of invariant occurrence between activities, which yields sets of activities named conjoint occurrence classes, which are used to infer the causal and concurrent behavior not exhibited in the log. Procedures derived from the method have been implemented in the ProM framework. Experimental results show that the algorithm can discover models form logs including less traces compared to other standing discovery algorithms.*

Keywords: Process discovery, Petri nets, Incomplete logs

1 Introduction

The automated synthesis of formal models from the observation of a process execution in the form of event sequences is called process discovery [3]. Event data is provided commonly by resource planning and workflow management systems in organizations as logs of event traces, which represent the behavior of the process to be discovered. The aim is to obtain suitable models that describe clearly the causal and parallel relationships between the events in traces.

One significant problem in process discovery techniques is the fact that the log could not contain enough information to discover a process model, i.e. the log might be incomplete. This might cause that the obtained model excludes actual behavior or represents some behavior that is not in the real process. Many things can cause the incompleteness of the log; one of the most common is the presence of parallelism, due to the high number of possible parallel activities whose interleaving cannot be sampled in the traces.

The obtained model through a discovery technique must be easy to understand by representing clearly, the causal and parallel relationships between the activities. Furthermore, such a model must represent all the observed behavior (provided as input data) and the lowest amount as possible of non-observed behavior. Several metrics called quality dimensions [6] assess the features above mentioned for the discovered model with respect to the event log.

A procedure used to test whether or not a technique is able to discover models that have the same language as the real-life process by which a log was produced is called *rediscoverability*, which compares the obtained model directly with the real-life process. This procedure is summarized in Figure 1.

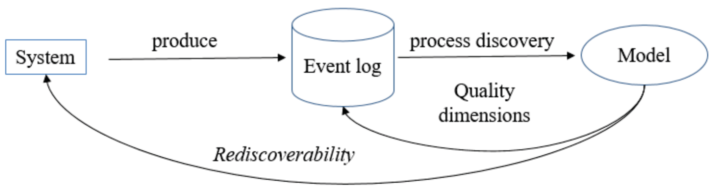


Fig. 1. Model quality evaluation

The method proposed in this paper pursues to obtain a suitable Petri net model that reveals some behavior hidden in the log, namely some causal and/or concurrent relationships. A method that allows building sound WFN from logs including a reduced number of traces is presented. The discovery technique is based on the concept of invariant occurrence between activities, which leads to obtain sets of events named conjoint occurrence classes; such classes help to infer the causal and concurrent behavior not exhibited in the log.

The paper is organized as follows. In Section 2 the related work is overviewed, In Section 3, the basic notions of PN and WFN are recalled. Section 4 states the first notions of invariance of occurrence and defines the occurrence classes. In Section 5 the synthesis of PN model from the conjoint occurrence classes is presented. Section 6 demonstrates the performance of the proposal through experiments and the results are compared with other approaches. Section 7 concludes the paper.

2 Related Work

Pioneer works on the matter, named language learning techniques, have been proposed in the field of computer sciences. The aim was to build fine representations (finite automata, grammars) of languages from samples of accepted words [12, 5]. In the field of workflow management systems, several *process discovery techniques* have been proposed. In an early proposal [4], a finite automaton, called conformal graph is obtained from sample traces execution. Later in [2] the alpha algorithm uses the log based ordering between activities to infer the a model. Numerous techniques present extensions To this algorithm, which are able to discover more complex behavior such as implicit dependencies [19]. In [8] a probabilistic approach to find the concurrent and direct relations between activities is presented. In [16] a formalism to represent the partial order between activities, which improve the visualizations of event logs with additional data is introduced. The discovery problem is addressed in an Integer linear programing approach in [20]. The inductive miner (IM) is presented in [13], which applies a divide-and-conquer approach that iteratively partitions the activities in process constructs as joint and splits. An adaptation of this algorithm capable to deal with incomplete logs called (IMin) [14] uses probabilistic behavioral relations to deal with incompleteness. Recently, a new formal model to represent process called *process tree* has been introduced in [1]. Many methods have adopted this representation; such is the case of evolutionary tree miner (ETM) [6], that uses a genetic algorithm to discover a process tree; the innovation in this technique is that it allows the user to select the preferences with respect to the quality dimensions of the final model.

In the field of Discrete Event Systems (DES), where the problem is named identification, several approaches have been proposed for building models representing the observed behavior of automated processes. The incremental approach proposed in [15] , allows building safe interpreted PN models from a continuous stream of system's outputs. Later, a technique based on project the activities sequence on sub-alphabets to discover specific patterns is present in [17]. In [11] input-output identification of automated manufacturing process is addressed; an interpreted PN is obtained from a set of sequences of input-output vectors collected from the controller during the system cyclic operation. Recently an approach based on the computing t-invariants that allow finding implicit dependencies in the PN model is presented in [18]. The literature on this research line is vast; wide reviews on DES identification can be found in [10, 7].

In both fields, the aim in general is to discover processes models from observed behavior. However, the nature of processes leads to problem statements somehow different in terms of input data.

3 Background

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

Definition 1. *An ordinary Petri Net structure G is a bipartite digraph represented by the 4-tuple $G = (P, T, I, O)$ where: $P = \{p_1, p_2, ..., p_{|P|}\}$ and $T = \{t_1, t_2, ..., t_{|T|}\}$ are finite sets of vertices named places and transitions respectively; $I : P \times T \to \{0, 1\}$ ($O : T \times P \to \{0, 1\}$) is a function representing the arcs going from places to transitions (from transitions to places).*

A marking function $M : P \to \mathbb{N}$ represents the number of tokens residing inside each place; it is usually expressed as an $|P|$-entry vector.

A Petri Net system or Petri Net (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}. The reachability set of a PN is the set of all possible reachable markings from $M0$ firing only enabled transitions;

this set is denoted by $R(G, M_0)$*. For any* $t_j \in T$, $\bullet t_j = \{p_i | I(p_i, t_j) = 1\}$*, and* $t_j \bullet = \{p_i | O(t_j, p_i) = 1\}$*, similarly for any* $p_i \in P$, $\bullet p_i = \{t_j | O(t_j, p_i) = 1\}$*, and* $p_i \bullet = \{t_j | I(p_i, t_j) = 1\}$.

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 $Mi \in R(G, M_0)$ and $t \in T$ there is a reachable marking $M_k \in R(G, M_0)$ such that t is enabled in M_k.

Definition 2. *A WorkFlow net (WFN) N is a subclass of PN owning the following properties [3]: (1) it has two special places: i and o. Place i is a source place:* $\bullet i = \emptyset$*, and place o is a sink place:* $o \bullet = \emptyset$*. (2) If a transition* t_e *is added to PN connecting place o to i, then the resulting PN (called extended WFN) is strongly connected.*

A consecutive activities sequence: $\langle x, d, e, a, b, y \rangle$ of a process executions that brings the system from initial state into the final state, corresponds to a *trace*; a *workflow log* is a multiset of traces. The set of traces that can be produced by a model N, is the language of N, is denoted by $\mathcal{L}(N)$. An example workflow net is shown in Fig.2

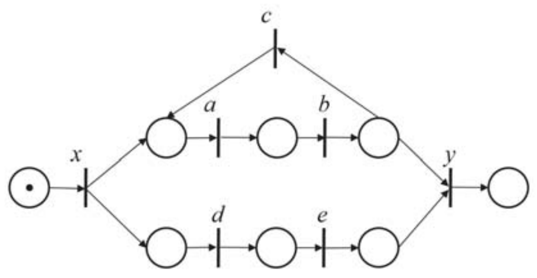


Fig. 2. A *sound* Workflow net N.

Definition 3. *A WFN N is said to be sound iff* (N, M_0) *is safe and for any marking* $M_i \in R(N, M_0)$, $o \in M_i \rightarrow M_i = [o]$*, N not contains dead transitions.*

Definition 4. *(Relations between activities). Let* λ *be a log over* T*. and let* $a, b \in T$*:*

- $a \rightarrow_\lambda b$*, iff there is a trace* $\sigma = t_1, t_2, ...t_n > \in \lambda$ *and* $1 \leq i < n-1$*, such that* $t_i = a$, $t_{i+1} = b$*,*
- $a ||_\lambda b$ *iif* $a \rightarrow_\lambda b$ *and* $b \rightarrow_\lambda a$*.*
- $first(\sigma) = t_1$*,* $last(\sigma) = t_n$*, if* $n \geq 1$

The ordering relations has been used in [2, 4] as an abstraction of the behavior described by a log or model. The previous relations for a workflow model N, are defined similarly i.e. if $\langle \ldots, a, b, \ldots \rangle \in \mathcal{L}(N)$, then $a \rightarrow_N b$.

Definition 5. *Completeness* [2], *Let N be a sound workflow net and λ a workflow log of N. λ is a complete workflow log of N iff* (1) *for any workflow log λ' of N: $\rightarrow_{\lambda'} \subseteq \rightarrow_{\lambda}$,* (2) *each activity of N is present in λ.*

4 Invariance of occurrence

This section introduces the notion of invariance of occurrence and presents a technique for deriving a partial order relationship between activities.

4.1 Trace handling

A trace is a sequence of ordered activities, such that the immediately preceding activity of a referent activity in the sequence, is associated with a position that is less by exactly one from the position of the reference activity. i.e. $< a, b, c >$ is a trace where the activity a is associated with position 1 the activity b is associated with the positions 2 and the activity c is associated with the position 3, we denote x_i to refer the *ith* activity in the trace. First, useful operators for expressing the relative positions of activities in a trace and activity precedence/subsequence relationships are introduced.

Definition 6. (**Trace handling operators**). *Let λ be a workflow log over T, let $a \in T$; if there is a trace $\sigma_k = x_1, x_2, ..., x_n$ such that $\sigma_k \in \lambda$ and $a \in \sigma_k$,*

- $\tau(x_i, \sigma_k)$ *provides the activity name of element x_i in σ_k*
- $Pos(a, \sigma_k) = \{p, q, r \ldots, s\}$ *such that* $\tau(x_p, \sigma_k) = a, ..., \tau(x_s, \sigma_k) = a$
- $Pred(x_i, \sigma_k) = \tau(x_{i-1}, \sigma_k)\ \ , i \in \{2, 3, \ldots, n\}$
- $Succ(x_i, \sigma_k) = \tau(x_{i+1}, \sigma_k)\ \ , i \in \{1, 2, \ldots, n-1\}$
- $Predset(a, \sigma_k) = \begin{cases} \emptyset \quad\quad ,if \quad 1 \in Pos(a, \sigma_k) \\ \bigcup_{i=2}^{p} Pred(x_i, \sigma_k) \cap \cdots \cap \bigcup_{i=q+2}^{r} Pred(x_i, \sigma_k) \\ \quad \cap \bigcup_{i=r+2}^{s} Pred(x_i, \sigma_k), \\ \quad\quad if Pos(a, \sigma_k) = \{p, \ldots, q, r, s\}, p <, \ldots, < r < s \end{cases}$
- $Succset(a, \sigma_k) = \begin{cases} \emptyset \quad\quad ,if \quad n \in Pos(a, \sigma_k) \\ \bigcup_{i=p}^{q-2} Succ(x_i, \sigma_k) \cap \bigcup_{i=q}^{r-2} Succ(x_i, \sigma_k) \cap \ldots \\ \quad \cap \bigcup_{i=s}^{n-1} Succ(x_i, \sigma_k), \\ \quad\quad if Pos(a, \sigma_k) = \{p, q, r, \ldots, s\}, p < q <, \ldots, < s \end{cases}$

The function $Pos(a, \sigma_k)$ provides a set of indices in which the symbol a appears in the trace σ_k, while the functions *Pred* and *Succ* provide, respectively, the predecessor and successor activity of x_i in the trace. The *Predset* is build from the activities sets appearing between the occurrence of the activity symbol, together with the beginning of the trace and the first occurrence of the activity symbol; the operator yields common symbols among these activities sets. The *Succset* is defined similarly.

Example 1. Consider a log λ with three traces $\sigma_1 = \langle x, a, b, d, e, c, a, b, y \rangle, \sigma_2 = \langle x, d, e, a, b, y \rangle, \sigma_3 = \langle x, a, b, d, c, a, e, b, y \rangle$, which has been produced using the workflow net N in Fig. 2. Note that is not a *complete* with respect to N, since $b \rightarrow_N c$, $e \rightarrow_N y$ hold in N but not in λ. Furthermore, the only knowledge about the parallel behavior generated between the cycle (a, b, c) and the sequence (d, e) is $a||_\lambda e$; all the other parallel relations are not in λ. The relations found in λ are the following: $x \rightarrow_\lambda (a, d)$, $a \rightarrow_\lambda (b, e)$, $b \rightarrow_\lambda (d, y)$, $c \rightarrow_\lambda a$, $d \rightarrow_\lambda (c, e)$, $e \rightarrow_\lambda (b, a)$.

If we apply the *Predset* and *Succset* functions over the activity symbol a and the traces in λ we get: $Predset(a, \sigma_1) = \{x\} \cap \{b, d, e, c\} = \emptyset$, $Predset(a, \sigma_2) = \{x, d, e\}$, $Predset(a, \sigma_3) = \{x\} \cap \{b, d, c\} = \emptyset$; $Succset(a, \sigma_1) = \{b, d, e, c\} \cap \{b, y\} = \{b\}$, $Predset(a, \sigma_2) = \{b, y\}$, $Predset(a, \sigma_3) = \{b, d, c\} \cap \{e, b, y\} = \{b\}$.

4.2 Conjoint Occurrence Relation

Now, we are going to introduce several notions useful to determine the sets of activities that occur together in all traces of λ.

Definition 7. *The Invariable precedence set of the activity symbol a in λ is the set of activities that always precede a in every trace σ_k it appears, i.e. $InvPred(a, \lambda) = \bigcap_{\sigma_k} Predset(a, \sigma_k)$. Similarly the Invariable subsequent set is $InvSucc(a, \lambda) = \bigcap_{\sigma_k} Succset(a, \sigma_k)$.*

Definition 8. *The Occurrence Invariable Set of the activity symbol a in λ is the set of activities that always occur when a occurs, that is, $Oi(a) = InvPred(a, \lambda) \cup InvSucc(a, \lambda) \cup \{a\}$.*

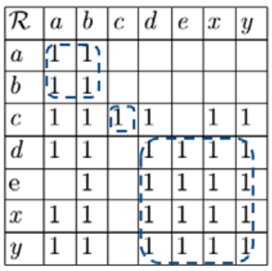
Table 1 shows the *Invariable precedence and subsequent sets* along with their respective *Occurrence Invariable sets* of each activity symbol in T, corresponding to workflow log in *Example.* 1.

Table 1. Occurrence Invariable set of activities a and d

Activity	InvPred	InvSucc	Oi
x	$\emptyset$	$\{a, b, d, e, y\}$	$\{x, a, b, d, e, y\}$
a	$\emptyset$	$\{b\}$	$\{a, b\}$
b	$\{a\}$	$\emptyset$	$\{a, b\}$
c	$\{x, a, b, d\}$	$\{a, b, y\}$	$\{x, a, b, c, d, y\}$
d	$\{x\}$	$\{a, b, d, e, y\}$	$\{x, a, b, d, e, y\}$
e	$\{x, d\}$	$\{b, y\}$	$\{x, b, d, e, y\}$
y	$\{x, a, b, d, e\}$	$\emptyset$	$\{x, a, b, d, e, y\}$

Definition 9. *Let $\mathcal{R}$ be a binary relation over T defined as $\mathcal{R} = \{(a, b) | b \in Oi(a), \forall a, b \in T\}$. Let $\mathcal{R}'$ be the coarsest equivalence relation such that $\mathcal{R}' \subseteq \mathcal{R}$. $\mathcal{R}'$ is called the Conjoint Occurrence Relation (CO_r) and the equivalence classes are called Conjoint Occurrence Classes (COc_i).*

The Conjoint Occurrence Classes of the log above are $COc_1 = \{a, b\}$, $COc_2 = \{c\}$, $COc_3 = \{x, d, e, y\}$; the Fig. 3 shows the matrix representation of the relation $\mathcal{R}$ in which the doted rectangles represent the COc_i.



$\mathcal{R}$	a	b	c	d	e	x	y
a	1	1					
b	1	1					
c	1	1	1	1		1	1
d	1	1		1	1	1	1
e		1		1	1	1	1
x	1	1		1	1	1	1
y	1	1		1	1	1	1

Fig. 3. Matrix representation of relation $\mathcal{R}$, 1 for related elements, noting otherwise.

Conjoint Occurrence Classes have interesting properties, namely *Invariance* and *Order*.

Proposition 1. (*Invariance*). *Let λ be a workflow log, COc_k a Conjoint Occurrence Class induced by $\mathcal{R}'$, and $a \in COc_k$; when $\sigma_i \in \lambda$ is executed, such that $a \in \sigma_i$ all activities in COc_k are also executed in σ_i.*

Proof. By definition, $Oi(a)$ provides all the activities that invariably occurs when a occurs. Consider $\forall a, b, c \in COc_k, a \in Oi(a)$, if $a \in Oi(b)$ then $b \in Oi(a)$ and if $a \in Oi(b) \wedge b \in Oi(c)$ then $a \in Oi(c)$, there is no activity unrelated among the rest of activities; then the set is invariable in all the traces of λ. $\square$

Proposition 2. (*Order*). *Let λ be a workflow log, COc_k a Conjoint Occurrence Class induced by $\mathcal{R}'$. All the activities in COc_k occur in the same order in all the $\sigma_i \in \lambda$.*

Proof. Assume that there are two activities $a, b \in COc_k$, in a parallel relation $(a||b)$; then when the operators *Predset* and *Succset* are applied to one activity, the other one will not appear in its Oi(), consequently they are not in the same COc_k. Similarly, suppose that there are two traces in which a and b appear in a different order; then the same reasoning is applied to conclude that such activities cannot belong to the same COc_k. $\square$

If we take activities $a, b \in T$ of Example 1, we can observe that $a \in COc_1$ and the activity a is present in $\sigma_1, \sigma_2, \sigma_3$; consequently, b is also in $\sigma_1, \sigma_2, \sigma_3$, as shown in the traces. $\sigma_1 = \langle x, \boldsymbol{a}, \boldsymbol{b}, d, e, c, a, b, y\rangle, \sigma_2 = \langle x, d, e, \boldsymbol{a}, \boldsymbol{b}, y\rangle, \sigma_3 = \langle x, \boldsymbol{a}, \boldsymbol{b}, d, c, a, e, b, y\rangle$; furthermore, they occur in the same *order*.

5 From Conjoint Occurrence Classes to Petri Net structures

Besides the *Invariance* and *Order* information provided by the Conjoint Occurrence classes, they provide much more information about the final model. Given the way in which this classes are computed and the partition of the alphabet T; it is possible to obtain hidden relations between the elements of a COc_i; below we present some propositions that help to find this hidden relations.

5.1 Finding sequential substructures

Proposition 3. *The members of a COc_k form a sequential Petri Net substructure* (transition-place-transition) *according to their Order, in which each couple of consecutive activities are causally related.*

Proof. (by contraposition). Suppose that the elements in COc_k do not have a PN sequential substructure according to their *Order*. Let a, b be any activities in COc_k such that $a < b$ (a before b); given that there is not a PN sequential substructure, at least one of the places p_i that connect a to b is missing causing that b could appear before a in some execution; therefore by *proof* of Proposition 2, these sets of elements cannot be in the same COc_k. □

Definition 10. *Each* sequential Petri Net sub-structure *can be disassembled in three parts: the **start** and the **end** activities according to the* order *and rest of the sequential substructure, which is called the **body**.*

Fig. 4 shows the sequential PN sub-structure of each COc in λ, the dotted rectangle corresponds to the class *body*.

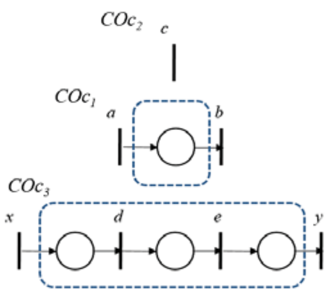


Fig. 4. The sequential substructure of the COc of the previous log.

Definition 11. *Two activities a, b belonging to a COc_i have a strong causality denoted ($a \Rightarrow b$), iff they are consecutive in the order of the COc_i*

Proposition 4. *For all places p_i in a sequential Petri Net sub-structure induced by a COc, $\bullet p_i = 1$ and $p_i \bullet = 1$; additionally their input and output activities must belong to the COc.*

Proof. Following the same reasoning as in the *proof* of Proposition 3, given that place p_i exists but there is an activity $k \in p_i\bullet$ such that $k \notin COc_i$, it could provoke an execution in which the activity a appears without the future appearance of b; therefore a, b are not in COc. In case that $k \in \bullet p_i$, this allows an execution in which the activity a appears without the previous appearance of b. □

5.2 Parallel and Causal Behavior Deducted from *COc*

The incompleteness of the log is mostly caused by the high level of parallel behavior of activities in the real process, this is a common issue in several discovery techniques based on *Log ordering relations*. The *COc* classes provide a way to deal with this problem. Since the members of a *COc* are causally related, given a property of one of its elements, this may be transmitted to the other members in the class, i.e. it is possible to deduce parallel behavior between a couple of activities, even if they are not directly consecutive in both directions.

Most of the workflow systems offers standard building blocks such as OR-split, OR-joint, AND-split, AND-joint [3], OR-splits/OR-joins correspond to places with multiple outgoing/ingoing arcs in a Petri Net, while AND-Split/AND-join corresponds to a transition with two or more output/input places, the latter blocks capture the parallel behavior of the process. Next proposition states how the *COc* and the $Oi()$ can be used to infer these particular transitions.

Proposition 5. *Let COc_i, COc_j be two different conjoint occurrence classes, such that at least an activity $a \in COc_i$ is parallel to an activity $b \in COc_j$ i.e. $(a||b)$ then.*

1) *If there exist two elements x, y corresponding to **start** and **end** activities of either COc_i or COc_j, such that $COc_i \cup COc_j \subseteq Oi(x) \cap Oi(y)$, then x, y are* And-split, And-join *activities respectively.*
2) *The elements in COc_i have a parallel relation with the elements of COc_j $(COc_i||COc_j)$, with the exception of activities corresponding to* And-split or And-join.

Proof. (1) Given that $Oi(a)$ contains the activities that occur whenever the a occurs and the And-split structure puts a token in two or more places enabling different threads, the only way in which an activity a has an And-split structure is that his $Oi(a)$ contains all the activities involved in all the parallel threads. The reasoning is similar for the AND-joint case. □

(2) Given that both COc_i and COc_j define sequential substructures in which each place has one input and output, then the only way to see the presence of an activity $a \in COc_j$ during the sequential execution of COc_i, is that the

activities in both sequential substructures are enabled by a previous And-split, consequently the activities in COc_i have a parallel behavior with the elements of COc_j $\square$

The COc gives a partitioned view of the process. Next function helps to join this partitions in order to complete the model.

Definition 12. *Let $X \subseteq T$ be a set of activities and $\sigma_i \in \lambda$; the function $\phi_X(\sigma_i)$ filters the trace by applying the natural projection of σ_i over $T \backslash X$.*

The previous function makes a filtering over the log traces, by removing all the occurrences of any activity $a \in X$. The traces preserve the order of reminder activities.

Once the parallel behavior between all the COc have been detected, it is possible to use the ϕ_X function as a strategy to find missing parallel behavior or links (causal relations) between sequential substructures, by assigning to X the parallel activities of a COc_i w.r.t a COc_j and vice versa.

In order to illustrate how the function ϕ_X operates, it is applied on the COc and previous log. Since $e||a$, then $COc_1||COc_3$. Giving X the parallel activities in COc_1 the result is: $\phi_{\{a,b\}}(\sigma_1) = \langle x, d, e, c, y\rangle$, $\phi_{\{a,b\}}(\sigma_2) = \langle x, d, e, y\rangle$ and $\phi_{\{a,b\}}(\sigma_3) = \langle x, d, c, e, y\rangle$, then by computing the *Relations between activities* for the filtered log, a new parallel behavior is discovered: $c||e$, therefore $COc_2||COc_3$.

Now being X the parallel activities in COc_3, the result is, $\phi_{\{d,e\}}(\sigma_1) = \langle x, a, b, c, a, b, y\rangle$, $\phi_{\{d,e\}}(\sigma_2) = \langle x, a, b, y\rangle$ and $\phi_{\{d,e\}}(\sigma_3) = \langle x, a, b, c, a, b, y\rangle$, then the new consecutive relation $b \rightarrow_\lambda c$ is discovered.

Notice that x, y are not part of the parallel behavior because they are the *start* and *end* in COc_3; furthermore, $COc_1 \cup COc_3 \subseteq Oi(x) \cap Oi(y)$, thus x, y corresponds to an AND-split and AND-joint respectively. The application of this procedure to the new parallel relation $(COc_2||COc_3)$ does not create new $\rightarrow_\lambda$ relations. This log filtering is performed iteratively over all the classes that have a parallel relation in addition to the new ones which result of a previous filter. The computation ends when no new consecutive relations are found.

5.3 Building the Petri Net Model

Once a hidden behavior in the log is found, it is used to connect all the sequential substructures into the final model by creating new places. **Table 2** shows the *Relations between activities* of the log λ in Example 1, after adding the found relations by filtering the log.

The rest of $(a \rightarrow_\lambda b)$ relations after removing all the parallel behavior $(||_\lambda)$ determine the existence of a place between activities a and b; some of these are already considered by the places which conform a *sequential Petri Net substructure*, such is the case of the strong causal relations $(a \Rightarrow b)$ in COc_1 and $(x \Rightarrow d)$, $(d \Rightarrow e)$, $(e \Rightarrow y)$ in COc_3 for *Example 1.* For the rest of relations, the addition of places is quite intuitive, it is defined as follows.

Table 2. Relations between activities after log filtering

T	x	a	b	c	d	e	y
x		$\rightarrow$			$\Rightarrow$		
a			$\Rightarrow$		$\parallel$	$\parallel$	
b				$\rightarrow$	$\parallel$	$\parallel$	$\rightarrow$
c		$\rightarrow$			$\parallel$	$\parallel$	
d		$\parallel$	$\parallel$	$\parallel$		$\Rightarrow$	
e		$\parallel$	$\parallel$	$\parallel$			$\Rightarrow$
y							

Definition 13. *(Merging activities in $\rightarrow_\lambda$ relation). When the left activity in two $\rightarrow_\lambda$ relations are the same ($a \rightarrow_\lambda b, a \rightarrow_\lambda c$) two possible substructures can be created.*

a) The places of $\rightarrow_\lambda$ relations are merged into a single one iff ($b \not\rightarrow_\lambda c \vee c \not\rightarrow_\lambda b$) This is denoted as $[a,\ b+c]$.

b) The places of $\rightarrow_\lambda$ relations are not merged iff ($b \rightarrow_\lambda c \vee c \rightarrow_\lambda b$) This is denoted as $[a,\ b||c]$.

Similarly, for $\rightarrow_\lambda$ relations when the right activity is common, ($b \rightarrow_\lambda a, c \rightarrow_\lambda a$), the substructure created will be either $[b||c\ ,\ a]$ or $[b+c\ ,\ a]$ accordingly. In general, a set of $\rightarrow_\lambda$ relations in the form ($a \rightarrow_\lambda b, a \rightarrow_\lambda c, \ldots, a \rightarrow_\lambda d$) may produce either $[a,\ b+c+d]$ or $[a,\ b||c||d]$. Consequently, the merging can be applied to composed relations, i.e. $[b+c\ ,\ a]$ and $[b+c\ ,\ d]$ leads $[b+c\ ,\ a+d]$.

For the activities in $first(\sigma)$ or $last(\sigma)$ we add two more places corresponding to the i and o places in the final workflow model i.e. $a \in i\bullet\,|\exists\sigma_i, a \in first(\sigma_i)$ and $a \in \bullet o|\exists\sigma_i, a \in last(\sigma_i)$

A similar way to infer the merging of places is presented in [2]. In contrast to this result our proposal reduces considerably the space search since high percentage of the places are already found previously by the sequential sub-structures

The Fig. 5 shows the Petri Net model after connecting the sequential substructures with the new relations.

All the definitions previously described can be summarized in the following algorithm that we called the *Conjoint Occurrence Miner* (*CoM*).

6 Implementation and tests

We implement the *CoM* algorithm as a plunging in the ProM Framework [9] to test our results and compare our approach with other important techniques in the literature.

The aim of these experiments is to show first, how the *CoM* algorithm deals with small logs, and then, to present how others process discovery algorithms work with the same incomplete logs.

Algorithm 1 *Conjoint Occurrence Miner* (*CoM*)

Input: Log λ.
Output: sound Workflow net N.

```
1: Compute $\rightarrow_\lambda, ||_\lambda$ relations.
2: for $\forall a \in T$ do
3:     Compute $InvPred(a, \lambda)$ and $InvSucc(a, \lambda)$
4:     Compute $Oi(a)$
5: end for
6: From $Oi$ compute the $COc$ classes $COc$
7: Build the $PN$ sequential substructures
8: Find the missing $\rightarrow$ relations
9: Build the final $PN$ as stated in (5.3)
```

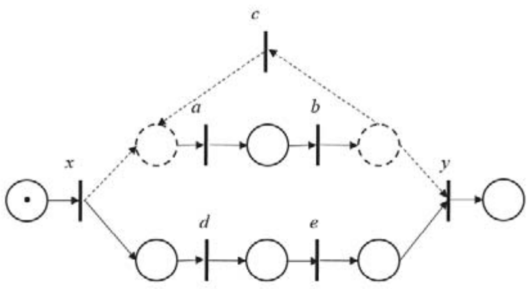


Fig. 5. Petri Net model discovered from λ

6.1 Experiments

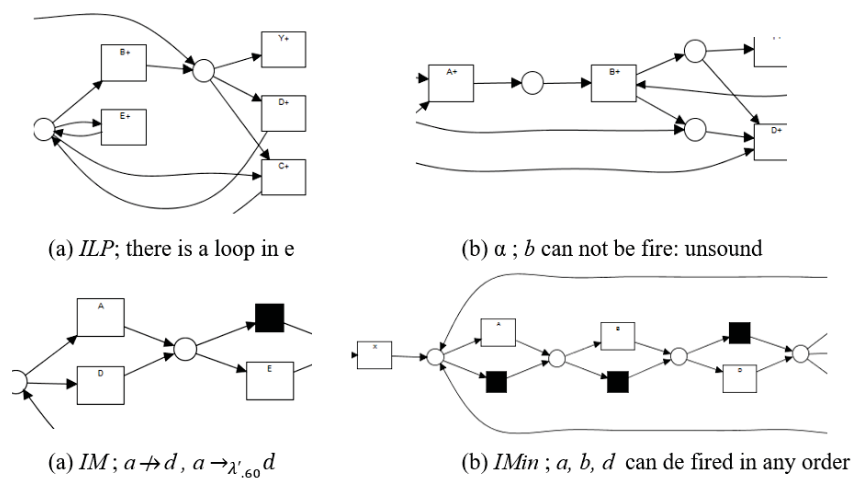
Experiment 1
We used the Workflow net N presented in *Figure* 2 to generate a complete workflow log λ' according to the $\rightarrow$ relation, then we construct 4 smaller sub-logs of λ' by removing traces from λ', which have $0.90, 0.80, 0.75$ and 0.60 of the average ratio of $\rightarrow$-pairs compared to the model N, named $\lambda'_{0.90}$, $\lambda'_{0.85}$, $\lambda'_{0.75}$ and $\lambda'_{0.60}$ respectively. In this case, $\lambda'_{0.60}$ corresponds to λ presented in *Example* 1.

For this experiment, we consider the following discovery algorithms: Integer Linear Programing miner (*ILP*)[20], α-algorithm [2], Inductive Miner (*IM*) [13], and Inductive Miner Incompleteness (*IMin*) [13], we use the ProM Framework [9] for perform the experiments; $ILP(\lambda)$, for instance, means the application of ILP algorithm to the log λ. We want to show (1) for which of the mentioned logs the algorithm return a language equivalent model and (2) if the algorithm is able to *rediscover* the original model.

Results.
(1) $\mathcal{L}(N) = \mathcal{L}(ILP(\lambda'_{0.85}))$, $\mathcal{L}(N) = \mathcal{L}(\alpha(\lambda'_{0.90}))$, $\mathcal{L}(N) = \mathcal{L}(IM(\lambda'_{0.90}))$, $\mathcal{L}(N) = \mathcal{L}(IMin(\lambda'_{0.60}))$ and $\mathcal{L}(N) = \mathcal{L}(CoM(\lambda'_{0.60}))$.

(2) ILP, α and IM require the full $\rightarrow$ relations for rediscover the original model N, whereas $IMin$ and CoM rediscover N with $\lambda'_{0.90}$, $\lambda'_{0.60}$ respectively.

The one that needs less information for rediscover the model is the CoM algorithm. Figure 6 shows fragments of the resulting models applying the others algorithms to $\lambda'_{.60}$.



(a) *ILP*; there is a loop in e

(b) α ; *b* can not be fire: unsound

(a) *IM* ; $a \not\rightarrow d$, $a \rightarrow_{\lambda'_{.60}} d$

(b) *IMin* ; *a, b, d* can de fired in any order

Fig. 6. Discovered models from $\lambda'_{.60}$

Experiment 2
In this experiment, thirty logs are used to tests the proposed method and other discovery techniques. Such logs have been obtained in the same way than in *Experiment 1*, from six diverse *sound* WFN involving form six to ten activities; the nets are shown in Figure 7.

Results. Table 3 shows the results of *Experiment* 2. Column 2: gives the percentage of logs for which the discovery technique returns a language equivalent model. Column 3: gives the percentage of logs for which the technique is able to rediscover the original model.

Table 3. Results of the Experiment 2

miner	*Language equivalent*	*Rediscoverability*
ILP	36%	23%
α	23%	16%
IM	56%	33%
IMin	100%	53%
CoM	90%	83%

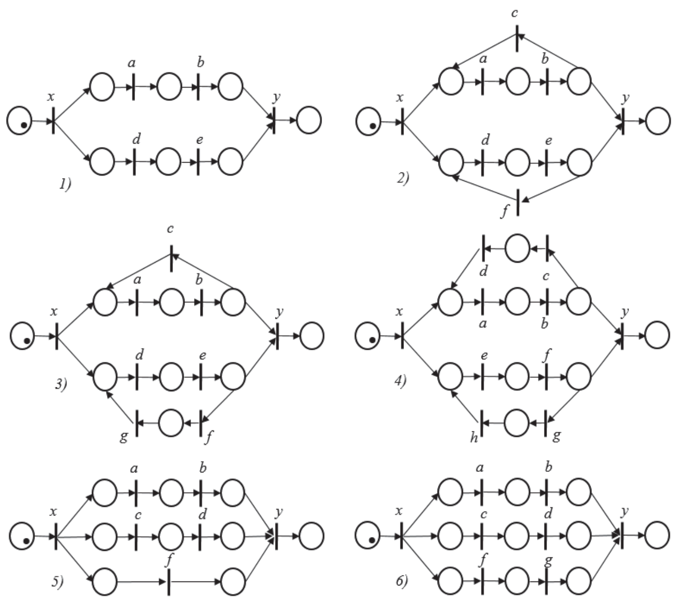


Fig. 7. Workflow nets of *Experiment 2*.

6.2 Limitations of the method

Although the method shows good results for dealing with incompleteness, there are particular behaviors that cannot be discovered using this technique; such is the case of *short loops* structures as shown in N_1 and N_2 of Figure 8. This is because it is not easy to infer the conjoint occurrence classes COc from traces in which an activity can be observed directly after itself.

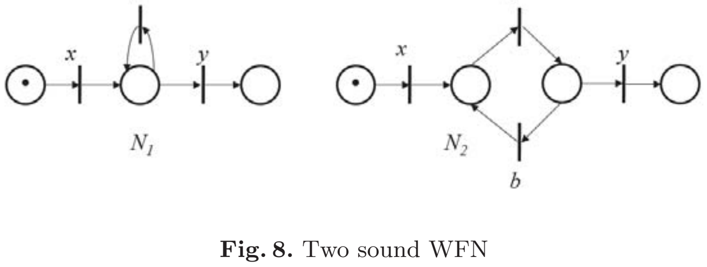


Fig. 8. Two sound WFN

7 Conclusion

In this work a novel discovery technique is presented. It is based on a new relation between activities called Conjoint Occurrence Relation, which leads to a partition of the activities alphabet into classes. Such classes are used to infer causal and parallel relations missing in the incomplete log, and consequently, Petri net substructures. Furthermore, the experimental results show that the performance algorithm is acceptable with respect to other relevant discovery techniques. Although the technique is presented as a discovery one, which can deal with logs including reduced number of traces, the proposed notions and algorithms seem to be a promising approach for dealing with the problem of *rediscoverability*. Current research addresses study the class of models and the minimum behavior exhibited in the log for ensure this property.

References

1. Aalst, W., Buijs, J., Dongen, B.: Data-Driven Process Discovery and Analysis: First International Symposium, SIMPDA 2011, Campione d'Italia, Italy, June 29 – July 1, 2011, Revised Selected Papers, chap. Towards Improving the Representational Bias of Process Mining, pp. 39–54. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), `http://dx.doi.org/10.1007/978-3-642-34044-4_3`
2. Van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. Knowledge and Data Engineering, IEEE Transactions on 16(9), 1128–1142 (2004)
3. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer Publishing Company, Incorporated, 1st edn. (2011)
4. Agrawal, R., Gunopulos, D., Leymann, F.: Mining Process Models from Workflow Logs. In: Schek, H.J., Saltor, F., Ramos, I., Alonso, G., Schek, H.J., Saltor, F., Ramos, I., Alonso, G. (eds.) EDBT. Lecture Notes in Computer Science, vol. 1377, pp. 469–483. Springer (1998), `http://dblp.uni-trier.de/rec/bibtex/conf/edbt/AgrawalGL98`
5. Angluin, D.: Queries and Concept Learning. Mach. Learn. 2(4), 319–342 (Apr 1988), `http://dx.doi.org/10.1023/a:1022821128753`
6. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity. International Journal of Cooperative Information Systems 23(01), 1440001 (2014), `http://www.worldscientific.com/doi/abs/10.1142/S0218843014400012`
7. Cabasino, M.P., Darondeau, P., Fanti, M.P., Seatzu, C.: Model identification and synthesis of discrete-event systems. Contemporary Issues in Systems Science and Engineering (2013)
8. Cook, J.E., Du, Z., Liu, C., Wolf, A.L.: Discovering models of behavior for concurrent workflows. Computers in industry 53(3), 297–319 (2004)
9. Dongen, B.F., Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., Aalst, W.M.P.: Applications and Theory of Petri Nets 2005: 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005. Proceedings, chap. The ProM Framework: A New Era in Process Mining Tool Support, pp. 444–454. Springer Berlin Heidelberg, Berlin, Heidelberg (2005), `http://dx.doi.org/10.1007/11494744_25`

10. Estrada-Vargas, A.P., López-Mellado, E., Lesage, J.J.: A comparative analysis of recent identification approaches for discrete-event systems. Mathematical Problems in Engineering 2010 (2010)
11. Estrada-Vargas, A.P., López-Mellado, E., Lesage, J.J.: Input–output identification of controlled discrete manufacturing systems. International Journal of Systems Science 45(3), 456–471 (2014)
12. Gold, M.E.: Language identification in the limit. Information and Control 10(5), 447–474 (1967), `http://www.isrl.uiuc.edu/\~{}amag/langev/paper/gold67limit.html`
13. Leemans, S.J.J., Fahland, D., Aalst, W.M.P.: Application and Theory of Petri Nets and Concurrency: 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings, chap. Discovering Block-Structured Process Models from Event Logs - A Constructive Approach, pp. 311–329. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), `http://dx.doi.org/10.1007/978-3-642-38697-8_17`
14. Leemans, S.J.J., Fahland, D., Aalst, W.M.P.: Application and Theory of Petri Nets and Concurrency: 35th International Conference, PETRI NETS 2014, Tunis, Tunisia, June 23-27, 2014. Proceedings, chap. Discovering Block-Structured Process Models from Incomplete Event Logs, pp. 91–110. Springer International Publishing, Cham (2014), `http://dx.doi.org/10.1007/978-3-319-07734-5_6`
15. Meda-Campana, M., Ramirez-Treviro, A., López-Mellado, E.: Asymptotic identification of discrete event systems. In: Decision and Control, 2000. Proceedings of the 39th IEEE Conference on. vol. 3, pp. 2266–2271. IEEE (2000)
16. Mokhov, A., Carmona, J.: Event log visualisation with conditional partial order graphs: from control flow to data. In: Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data, ATAED 2015, Brussels, Belgium, June 22-23, 2015. pp. 16–30 (2015), `http://ceur-ws.org/Vol-1371/paper02.pdf`
17. Saives, J., Faraut, G., Lesage, J.J.: Identification of discrete event systems unobservable behaviour by petri nets using language projections. In: Control Conference (ECC), 2015 European. pp. 464–471 (July 2015)
18. Tapia-Flores, T., López-Mellado, E., Estrada-Vargas, A.P., Lesage, J.J.: Petri net discovery of discrete event processes by computing t-invariants. In: Emerging Technology and Factory Automation (ETFA), 2014 IEEE. pp. 1–8 (Sept 2014)
19. Wen, L., Aalst, W.M.P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Mining and Knowledge Discovery 15(2), 145–180 (2007), `http://dx.doi.org/10.1007/s10618-007-0065-y`
20. Werf, J.M.E.M., Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Applications and Theory of Petri Nets: 29th International Conference, PETRI NETS 2008, Xi'an, China, June 23-27, 2008. Proceedings, chap. Process Discovery Using Integer Linear Programming, pp. 368–387. Springer Berlin Heidelberg, Berlin, Heidelberg (2008), `http://dx.doi.org/10.1007/978-3-540-68746-7_24`