Process Discovery Algorithms using Numerical Abstract Domains

Josep Carmona, *Member, IEEE,* Jordi Cortadella, *Member, IEEE,*

Abstract—The discovery of process models from event logs has emerged as one of the crucial problems for enabling the continuous support in the life-cycle of an information system. However, in a decade of process discovery research, the algorithms and tools that have appeared are known to have strong limitations in several dimensions. The size of the logs and the formal properties of the model discovered are the two main challenges nowadays. In this paper we propose the use of numerical abstract domains for tackling these two problems, for the particular case of the discovery of Petri nets. First, numerical abstract domains enable the discovery of general process models, requiring no knowledge (e.g., the bound of the Petri net to derive) for the discovery algorithm. Second, by using divide and conquer techniques we are able to control the size of the process discovery problems. The methods proposed in this paper have been implemented in a prototype tool and experiments are reported illustrating the significance of this fresh view of the process discovery problem.

Index Terms—Process discovery, Numerical Abstract Domains, Petri nets, Formal Methods, Concurrency.

1 Introduction

Process mining is an essential discipline for addressing challenges related to Business Process Management (BPM) and "Big Data"[1]. Informally, process mining algorithms are meant to extract knowledge from *event logs* stored by information systems, and use this knowledge for supporting the process perspective. Nowadays information systems record an overwhelming amount of data representing the footprints left by process executions.

Process mining faces three challenges relating event data (i.e., log files) and process models: *discovery* of a process model from an event log, *conformance checking* given a process model and a log, and *enhancement* of a process model with the information obtained from a log. In this paper we focus in the problem of discovery of Petri nets [2] from event logs. However, the techniques presented in this paper may be adapted for the discovery of other process formalisms. This paper is an extended version of [3].

There are several Petri net discovery algorithms in the literature [4], [5], [6], [7], [8], [9], [10], [11] that demonstrated to be of great value for undertaking small or medium-sized problem instances. However, it is well-accepted that current Petri net discovery algorithms often impose serious restrictions on the class of behaviors to be identified [4], [5], [6], or are unable to handle problems of industrial size [7], [8], [10], [11].

Abstract interpretation [12] is a generic approach for the static analysis of complex systems. The underlying notion in abstract interpretation is that of *upper approximation*: to provide an abstraction of a complex behavior with fewer details. A property about a system, such as an invariant, is in some way an abstraction: it represents all the states of the system that satisfy the property.

Intuitively, abstract interpretation defines a procedure to compute an upper approximation of the behavior of a system. This definition guarantees (a) the termination of the procedure and (b) that the result is conservative. An important decision is the choice of the kind of abstraction to be used, which is defined by a *numerical abstract domain*. For a given problem, there are typically several numerical abstract domains available. Each abstract domain provides a different trade-off between precision (proximity to the exact result) and efficiency.

There are many problems where abstract interpretation can be applied, several of them oriented towards the compile-time detection of run-time errors in software. For example, some analysis based on abstract interpretation can discover numeric invariants among the variables of a program. Several abstract domains can be used to describe the invariants: intervals [13], octagons [14], convex polyhedra [15], among others. These abstract domains provide different ways to approximate sets of values of numeric variables. This paper uses the domain of convex polyhedra for the discovery of Petri nets. However, the techniques presented in this paper can also be applied with other domains.

The algorithms for Petri net discovery presented in this paper use as input a *Parikh* representation of the log, i.e., a set of vectors denoting the number of occurrences of each log event for each intermediate state of a log trace. For example, for the trace *aba* there will be four Parikh vectors $(0,0)$ (empty prefix, initial state), $(1,0)$ (prefix *a*), $(1,1)$ (prefix *ab*) and $(2,1)$ (prefix *aba*). The set of Parikh vectors of an event log is then used to construct a convex polyhedron including these vectors (i.e., the

• *J. Carmona and J. Cortadella are with the Software Department, Universitat Politècnica de Catalunya, Barcelona (Spain).*

convex envelope). Finally, using the H-representation of the convex polyhedra domain, one may obtain a set of inequalities denoting invariants that relate the occurrence count between different log events. These inequalities can then be converted into Petri net elements.

The aforementioned technique guarantees the derivation of a *fitting* Petri net, i.e., a Petri net that can reproduce every trace in the log. Moreover, this paper shows that when all the inequalities are used, the Petri net derived is minimal in describing the log behavior, a very interesting property that relates to the well-known *precision* dimension in conformance checking [16]. Remarkably, the theory presented in this paper is the first one in deriving a general *pure* (no self-loops) Petri net without requiring any knowledge on the bounds of the model. This contrasts with the *region-based* algorithms [8], [9], [10], [11], where this information is a necessary input for the algorithm to be applied. Hence the only limitation is that no self-loops can be derived.

Moreover, due to the complexity of some of the convex polyhedra algorithms, this paper presents a technique that allows to decompose a process discovery problem into smaller instances that can be better handled. The strategy is based on clustering log events that are related. We use techniques inspired from *Principal Component Analysis* [17] to derive this information.

The paper is organized as follows: we provide a simple example in Section 2 to illustrate the approach of the paper. In Section 3, the theoretical basis of the paper is presented. Section 4 describes an algorithm to perform the discovery of invariants from an event log. For logs of medium/large size, the techniques of Section 5 should be applied in order to alleviate the complexity of the discovery problem. Experiments are reported in Section 6. In Section 7 we provide a short discussion on the relationship with the Petri net discovery techniques in the literature. Finally, conclusions are drawn in Section 8.

2 An illustrative example

Let us illustrate the theory of this paper with an example. Imagine we are given the following log:

```
 1: a b b a a b b a a b a a b a b b a a a a
 2: a a a b b b a a b a b b b a a b b a b a
 3: b a a b a a a a b a b b b b b a a b a b
 4: b a b a a b b a a b a b a a a a b a b a
 5: a a a b a b b a a b a b b a b b a a a a
 6: b a a a a b a a b b a a a b a b a a b b
 7: b a b a b a a b a a a a b b a b a a b a
 8: b a b a b a b a b a a a b b a b b a b a
 9: a b b a a a b b b a b a a a a a a a b b
10: a b b a b a a a b a a a b a a b a b b a
```

In spite of its apparent simplicity, this log represents a hard case for most of the existing techniques. This is mainly due to the fact that the log contains complex relations for the events occurrences, i.e., the *synchronic distance* [2] between a and b is non-unitary. The synchronic distance defines the degree of mutual dependence between a pair of events. Non-unitary synchronic distance between events is a common behavior in many domains, e.g., resources in *Manufacturing Systems* [18]. To express these complex relations with simple (restricted) Petri net structures will make the model to be spaghetti-like, since several ordinary elements (places, arcs) will be needed to represent a single complex structural element (e.g., a place with more than one token, or an arc with non-unitary weight).

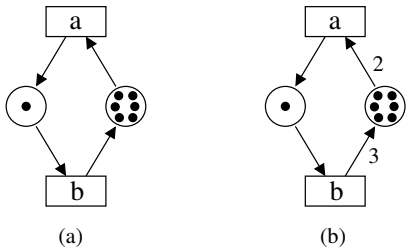

Fig. 1. (a) Petri net discovered from state-based regions with bound 7, (b) Petri net discovered by the approach of this paper.

The α-miner [4], probably the best-known Petri net miner nowadays[1], can only discover a Petri net accepting the language defined by the expression $a|b$, thus derives a model unable to reproduce anyone of the 10 traces in the log of the example[2] (i.e., an unfitting model), while the ILP miner [9] (implementing the language-based regions) does not discover any Petri net element and therefore the Petri net is simply the two transitions a and b each one without any predecessor/successor place, which generates the language $a^*||b^*$, i.e., a very imprecise model. The genetic miner [7] discovers a heuristic net that can be translated to a Petri net accepting the same language as the ILP miner, but in contrast it derives a complicated connected model with few invisible transitions.

To employ the state-based region algorithms from [10], one requires to convert the 10 traces above into an automaton that accepts the language of the log [19]. In order to use the algorithm, one must provide the bound k the Petri net should have in order for the algorithm to explore the lattice of k-bounded regions that represent Petri net places with their corresponding arcs. When setting this bound to any number less than 7 the algorithm performs as the ILP miner, i.e., no Petri net element is found. With $k = 7$ the algorithm discovers the cyclic Petri net shown in Figure 1(a). Notice that this Petri net accepts traces very different from the ones in the log, e.g., a trace starting as $aaaaaabbbbbbb\ldots$ belongs to the language of this Petri net.

Finally, for applying the approach of this paper, a Parikh representation of the log will be used. The Parikh representation of this log contains 61 Parikh vectors: $(0,0)$, $(1,0)$, $(0,1)$, $(1,1)$, $(2,0)$, ..., $(12,8)$, $(10,10)$. From

1. The α-miner is oriented towards the discovery of safe and sound *workflow* nets, thus unable to discover this complex behavior.

2. The operators *, $||$, $|$ and ; denote Kleene closure, interleaving, union and concatenation, respectively.

these vectors, a convex polyhedron is built that represents its minimal convex envelope. Importantly, the Parikh vectors are the only input required for deriving the convex polyhedron. The following are the non-trivial invariants defining this envelope:

$$6 - 2 \cdot \widehat{\sigma}(a) + 3 \cdot \widehat{\sigma}(b) \geq 0 \quad (1)$$

$$1 + \widehat{\sigma}(a) - \widehat{\sigma}(b) \geq 0 \quad (2)$$

where $\widehat{\sigma}(x)$ is a variable denoting the number of occurrences of the variable x. Invariant (1) ((2)) is transformed to the place on the right (left) of Figure 1(b). Note that the Petri net derived by our approach precisely describes the behavior of the log by using the arc weights computed.

3 Process mining, Petri nets and numerical abstract domains

3.1 Process mining notation

The behavior of a process is observed as sequences of events from a given alphabet. For convenience, we use T to denote the set of symbols that represent the alphabet of events. A trace is a word $\sigma \in T^*$ that represents a finite sequence of events. We denote λ as the empty trace. $|\sigma|_a$ represents the number of occurrences of a in σ.

A *log* $\mathcal{L}$ is a set of traces from a given alphabet. We say that $\sigma \in \mathcal{L}$ if σ is the prefix of some trace of $\mathcal{L}$.

Definition 3.1 (Parikh vector): Given an alphabet of events $T = \{t_1, \ldots, t_n\}$, the Parikh vector of a sequence of events is a function $\widehat{\ }: T^* \to \mathbb{N}^n$ defined as $\widehat{\sigma} = (|\sigma|_{t_1}, \ldots, |\sigma|_{t_n})$. For simplicity, we will also represent $|\sigma|_{t_i}$ as $\widehat{\sigma}(t_i)$.

Definition 3.2 (Parikh vectors of a log): Given a log $\mathcal{L}$, the set of Parikh vectors of $\mathcal{L}$ is defined as

$$\Pi(\mathcal{L}) = \{\widehat{\sigma} \mid \sigma \in \mathcal{L}\}.$$

The problem of process discovery requires the computation of a model M that adequately represents a log L. A model M is *overfitting* with respect to log L if it is too specific and too much driven by the information in L. On the other hand, M is an *underfitting* model for L if the behavior of M is too general and allows for things "not supported by evidence" in L. Whereas overfitting denotes lack of generalization, underfitting represents too much generalization. A good balance between overfitting and underfitting is a desired feature in any process discovery algorithm [1].

3.2 Petri nets

Definition 3.3 (Petri net [2]): A Petri net is a tuple (P, T, F, M_0) where P and T represent finite sets of places and transitions, respectively, and $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ is the weighted flow relation. A marking M is a function $M : P \to \mathbb{N}$. M_0 is a marking that defines the initial state of the Petri net.

The preset and postset of a place p are denoted as ${}^\bullet p$ and $p^\bullet$, respectively, and are defined as follows:

$${}^\bullet p = \{t \in T \mid F(t,p) > 0\}$$
$$p^\bullet = \{t \in T \mid F(p,t) > 0\}$$

A Petri net is said to be *pure* if it does not have any self-loop, i.e., $\forall p \in P : {}^\bullet p \cap p^\bullet = \emptyset$. Henceforth, we will assume that all Petri nets referred to in the paper are pure. A Petri net is *connected* if the underlying graph structure induced from F is connected.

The dynamic behavior of a Petri net is defined by its firing rules. A transition $t \in T$ is *enabled* in a marking M if $M(p) \geq F(p,t)$ for any $p \in P$. Firing an enabled transition t in a marking M leads to the marking M' defined by $M'(p) = M(p) - F(p,t) + F(t,p)$, for any $p \in P$, and is denoted by $M \xrightarrow{t} M'$. A sequence of transitions $\sigma = t_1 t_2 \ldots t_n$ is firable if there is a sequence of markings $M_1, M_2, \ldots, M_n$ such that

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \cdots \xrightarrow{t_n} M_n.$$

Given a Petri net N, $L(N)$ denotes the language of N, i.e., the set of firable sequences of transitions. The set of markings reachable from the initial marking M_0 is called the *Reachability Set* and denoted as $\mathsf{RS}(N)$. Finally, a place p is *redundant* if its removal does not change $L(N)$. Figure 1(b) contains an example of a Petri net N in which the trace $\sigma = aaaba$ belongs to $L(N)$.

3.3 The Marking Equation

Let us consider a place p with ${}^\bullet p = \{x_1, \ldots, x_k\}$, $p^\bullet = \{y_1, \ldots, y_l\}$ and all flow relations having weight 1. Let us assume that the place contains $M_0(p)$ tokens in its initial marking. Then, the following equality holds for any sequence of events σ:

$$M(p) = M_0(p) + \widehat{\sigma}(x_1) + \cdots + \widehat{\sigma}(x_k) - \widehat{\sigma}(y_1) - \cdots - \widehat{\sigma}(y_l).$$

The previous equation can be generalized for weighted flows:

$$M(p) = M_0(p) + \sum_{x_i \in {}^\bullet p} F(x_i, p) \cdot \widehat{\sigma}(x_i) - \sum_{y_i \in p^\bullet} F(p, y_i) \cdot \widehat{\sigma}(y_i).$$

If we formulate the previous equation for all places in a Petri net, we can compress it using a matrix notation:

$$M = M_0 + A \cdot \widehat{\sigma}$$

where M and M_0 are place vectors and A is the *incidence matrix* with $|P|$ rows and $|T|$ transitions that represents the flow relation of the net. The previous equation is called the *Marking Equation* of the Petri net [2].

The set of solutions for which the following inequality holds

$$M = M_0 + A \cdot \widehat{\sigma} \geq 0 \quad (3)$$

is called the *Potentially Reachable Set* ($\mathsf{PRS}(N)$). All reachable markings of a Petri net fulfill (3). However, the opposite is not always true. In general, there can

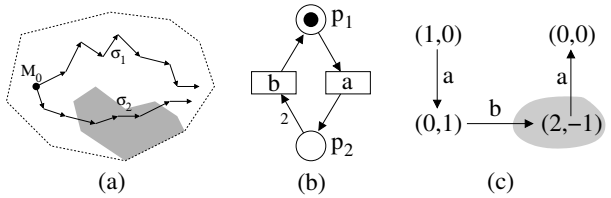

Fig. 2. Unfeasible event sequences in Petri nets.

be unreachable markings for which (3) also holds, i.e., $\mathsf{RS}(N) \subseteq \mathsf{PRS}(N)$. A detailed discussion on the relationship between $\mathsf{RS}(N)$ and $\mathsf{PRS}(N)$ can be found in [20].

Figure 2 illustrates the concept of *Potentially Reachable Set*. Figure 2(a) depicts the set of markings reachable from M_0. The shadowed area represents the set of vectors that contain at least one negative component, called *negative markings*[3]. The figure shows how a sequence σ_1 can travel across (non-negative) markings. However, the sequence σ_2 crosses some negative markings. Even though σ_2 leads to a non-negative marking, i.e., $M_0 + A \cdot \widehat{\sigma}_2 \geq 0$, the sequence is not fireable.

A simple example is shown in Fig. 2(b). A subset of potentially reachable markings is shown in Fig. 2(c). In this case, the sequences a and aba lead to non-negative markings. However, aba is not fireable because a negative marking is visited after ab.

3.4 The language of a Petri net

Given a Petri net $N = (P, T, F, M_0)$, we can define the potential language of N, denoted by $\mathsf{PL}(N)$, as

$$\mathsf{PL}(N) = \{\sigma \in T^* \mid M_0 + A\widehat{\sigma} \geq 0\}$$

where A is the incidence matrix of N. $\mathsf{PL}(N)$ contains all sequences that lead to a non-negative marking. However, not all sequences of $\mathsf{PL}(N)$ may belong to $L(N)$.

$L(N)$ is the subset of $\mathsf{PL}(N)$ that contains only those sequences that do not traverse negative markings. In other words, $L(N)$ contains all those sequences in which all prefixes also belong to $L(N)$. Thus, $L(N)$ can be recursively defined as follows:

$$L(N) = \{\lambda\} \cup \{\sigma \mid \sigma = \sigma' t, \sigma' \in L(N), t \in T, M_0 + A\widehat{\sigma} \geq 0\}.$$

The main focus of this work is the discovery of Petri nets from Parikh vectors. Given a log $\mathcal{L}$ from which we can calculate the set of Parikh vectors $\Pi(\mathcal{L})$, we will try to find A and M_0 in (3) such that the associated Petri net is a good approximation of the process behavior.

3.5 Convex polyhedra and integer lattices

An n-dimensional convex polyhedra is a convex set of points in $\mathbb{R}^n$. Convex polyhedra admit two equivalent representations: the H-representation and the V-representation [21]. The former denotes a convex polyhedron $\mathcal{P}$ as the intersection of a finite set of half-spaces, i.e.,

$$\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax + b \geq 0\} \tag{4}$$

where $A \in \mathbb{R}^{k \times n}$ and $b \in \mathbb{R}^k$ are the matrix and vector that represent k half-spaces. The V-representation uses a set of vertices and rays. The algorithms for convex polyhedra often use both representations and move from one to another depending on the most convenient representation for each case [22].

Neither the V- nor the H-representations of a convex polyhedron are unique. However, some canonical representations have been proposed [23].

The following result is important for modeling Petri nets with convex polyhedra.

Theorem 3.1: Let $\mathcal{P}$ be a convex polyhedron defined by the intersection of a finite set of half-spaces represented as in expression (4). $\mathcal{P}$ contains the origin $x = (0, \ldots, 0)$ *if and only if* $b \geq 0$.

Proof: If the origin belongs to $\mathcal{P}$, then b cannot have any negative component, otherwise one of the inequalities would not hold. Conversely, if $b \geq 0$, then the origin fulfills all the inequalities. □

Given a polyhedron $\mathcal{P}$, the set of integer points inside $\mathcal{P}$ is called the Z-polyhedron of $\mathcal{P}$. For the sake of brevity, all polyhedra mentioned in this work will be assumed to be convex.

3.6 Connecting Petri nets and convex polyhedra

Given a Petri net N, by comparing the expressions (3) and (4), we can observe that $\mathsf{PRS}(N)$ is the Z-polyhedron of a convex polyhedron that has two properties: $A \in \mathbb{Z}^{|P| \times n}$ and $M_0 \in \mathbb{N}^{|P|}$. These properties guarantee that the initial marking is not negative and only markings with integral token values are reachable.

The n-dimensional integer lattice $\mathbb{Z}^n$ is the lattice of n-tuples of integers. In our context, each lattice point represents a Parikh vector from an alphabet with n symbols. Given that we restrict ourselves to non-negative integers, we will often denote the lattice as $\mathbb{N}^n$.

A log can be represented as a set of walks in $\mathbb{N}^n$. Every step in a walk moves from one lattice point to another by only increasing one of the components of the n-tuple by one unit.

The link between logs and Petri nets is illustrated in Fig. 3. The figure at the left represents three different walks in a 2-dimensional space. The shadowed area represents a polyhedron that *covers* the points visited by the walks. The polyhedron can be represented by the intersection of two half-spaces in $\mathbb{R}^2$:

$$\begin{aligned} 1 + \widehat{\sigma}(a) - \widehat{\sigma}(b) &\geq 0 \\ 6 - 2 \cdot \widehat{\sigma}(a) + 3 \cdot \widehat{\sigma}(b) &\geq 0 \end{aligned}$$

The polyhedron can also be represented in matrix notation with a direct correspondence with the marking equation (3) of a Petri net:

$$\begin{bmatrix} 1 \\ 6 \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ -2 & 3 \end{bmatrix} \cdot \begin{bmatrix} \widehat{\sigma}(a) \\ \widehat{\sigma}(b) \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

3. We abuse the notation for markings introduced in Def. 3.3, to consider also vectors with at least one negative component.

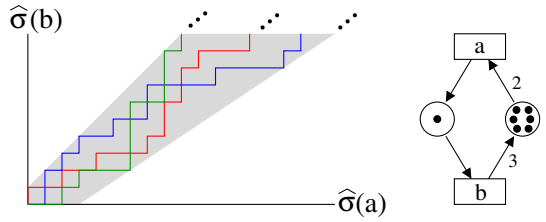

Fig. 3. Walks in the integer lattice and Petri net.

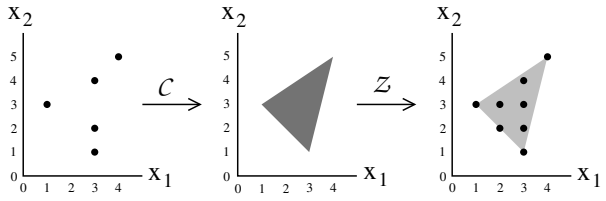

Fig. 4. Galois connection between $\mathbb{N}^n$ and $\mathbb{P}^n$.

The figure at the right represents a Petri net obtained from the interpretation of the marking equation. Each face of the polyhedron is represented by a place (row in the matrix). The set of Parikh vectors generated by the Petri net corresponds to the Z-polyhedron of the polyhedron depicted at the left.

We can observe there is a promising relationship between Parikh vectors and polyhedra. We next formalize this relationship.

3.7 Galois connection

Definition 3.4 (Galois connection [24]): Given two partially ordered sets, $(\mathcal{A}, \leq)$ and $(\mathcal{B}, \sqsubseteq)$, a *Galois connection* between them is a pair of monotone functions, $\alpha : \mathcal{A} \to \mathcal{B}$ and $\beta : \mathcal{B} \to \mathcal{A}$ such that for all $a \in \mathcal{A}$ and $b \in \mathcal{B}$: $\alpha(a) \sqsubseteq b \iff a \leq \beta(b)$.

Galois connections provide the framework for the approximation of the reachability set of a log using convex sets. The connection is established between the integer lattice $\mathbb{N}^n$ and the set of n-dimensional convex polyhedra $\mathbb{P}^n$.

The two monotone functions for the connection are the *Convex Hull* ($\mathcal{C}$) and the Z-polyhedron ($\mathcal{Z}$):

$$\mathbb{N}^n \underset{\mathcal{Z}}{\overset{\mathcal{C}}{\rightleftarrows}} \mathbb{P}^n$$

Given a set of points $S \subseteq \mathbb{N}^n$, $\mathcal{C}(S)$ is defined as the smallest convex polyhedron $\mathcal{P} \in \mathbb{P}^n$ such that $S \subseteq \mathcal{Z}(\mathcal{P})$. An example of the Galois connection is depicted in Fig. 4. The connection between the set of points and polyhedron in the figure are as follows:

$$\begin{array}{c}\{(1,3),(3,1),\\(3,2),(3,4),\\(4,5)\}\end{array} \xrightarrow{\mathcal{C}} \begin{array}{r}4x_1 - x_2 \leq 3\\ -2x_1 + 3x_2 \leq 7\\ x_1 + x_2 \geq 4\end{array} \xrightarrow{\mathcal{Z}} \begin{array}{c}\{(1,3),(2,2),(2,3),\\(3,1),(3,2),(3,3),\\(3,4),(4,5)\}\end{array}$$

The convex hull can also be defined for polyhedra. Given two polyhedra $\mathcal{P}_1$ and $\mathcal{P}_2$, the convex hull $\mathcal{P}_1 \sqcup \mathcal{P}_2$ is defined as the smallest convex polyhedron that contains $\mathcal{P}_1$ and $\mathcal{P}_2$. The monotonicity of the connections and the definition of convex hull determine the following properties:

$$\begin{aligned}\mathcal{C}(S_1 \cup S_2) &= \mathcal{C}(S_1) \sqcup \mathcal{C}(S_1)\\ \mathcal{C}(\mathcal{Z}(\mathcal{P})) &= \mathcal{P}\end{aligned}$$

This connection provides the core framework to reduce the discovery of Petri nets to the discovery of convex polyhedra that cover Parikh vectors associated to logs.

3.8 From logs to Petri nets

As it was discussed in Sect. 3.6, a log is a set of walks in $\mathbb{N}^n$. Let $\mathcal{L}$ be a log and $\mathcal{P}(\mathcal{L})$ the polyhedron obtained as the convex hull of $\Pi(\mathcal{L})$. The Galois connection presented in Sect. 3.7 guarantees that $\mathcal{P}(\mathcal{L})$ represents an over-approximation of $\Pi(\mathcal{L})$.

$\mathcal{P}(\mathcal{L})$ is the intersection of a set of half-spaces represented by $Ax + b \geq 0$, as described in (4). Since $\mathcal{P}(\mathcal{L})$ contains the origin, it also holds that $b \geq 0$ (Theorem 3.1).

If $T = \{t_1, \ldots, t_n\}$ is the alphabet of events of $\mathcal{L}$, then A is an $m \times n$ matrix, where m is the number of half-spaces. The Petri net N derived from $\mathcal{P}(\mathcal{L})$ is the tuple $N = (P, T, F, M_0)$ where

- $P = \{p_1, \ldots, p_m\}$ is a set of m places, each one corresponding to one of the half-spaces of $\mathcal{P}(\mathcal{L})$.
- $T = \{t_1, \ldots, t_n\}$ is the alphabet of events of $\mathcal{L}$.
- $\forall p_i \in P, t_j \in T$:

$$F(p_i, t_j) = \max(0, -A_{ij})$$
$$F(t_j, p_i) = \max(0, A_{ij}).$$

- $M_0 = b$ is the initial marking.

It is interesting to observe that the Petri net is guaranteed to be pure by construction. If $A_{ij} < 0$, then $F(p_i, t_j) > 0$ and $F(t_j, p_i) = 0$. Conversely, if $A_{ij} > 0$, then $F(t_j, p_i) > 0$ and $F(p_i, t_j) = 0$.

Next, an essential result of this paper follows.

Theorem 3.2: Let $\mathcal{L}$ be a log, $\mathcal{P}(\mathcal{L})$ be a polyhedron obtained from $\mathcal{L}$ and N a Petri net derived from $\mathcal{P}(\mathcal{L})$ as described above. Then, $\mathcal{L} \subseteq L(N)$.

Proof: By the construction of the Petri net N it is easy to see that any fireable sequence σ leads to a marking $M = A\widehat{\sigma} + b$ since, by construction, the marking equation of N coincides with the representation of the half-spaces of $P(\mathcal{L})$. Now, we need to prove that any sequence $\sigma \in \mathcal{L}$ is fireable in N, and does not visit any negative marking. We will prove this by induction on the length σ.

The theorem holds for $|\sigma| = 0$ since the empty sequence leads to marking $M_0 = b$, and we know that $b \geq 0$ by Theorem 3.1.

Let us now assume that any sequence $\sigma' \in \mathcal{L}$ belongs to $L(N)$, for $|\sigma'| < k$. Let us now consider a sequence $\sigma = \sigma' t \in \mathcal{L}$ with $t \in T$. By the construction of $P(\mathcal{L})$ as

the convex hull of $\Pi(\mathcal{L})$, we know that

$$M' = A\widehat{\sigma}' + b \geq 0$$
$$M = A\widehat{\sigma} + b \geq 0$$

We also know that t is enabled in M'. We can prove this by contradiction. If t would not be enabled in M', then there would be some $p_i \in {}^\bullet t$ such that $M'(p_i) < F(p_i, t)$. By the firing rules of the Petri net we also know that

$$M(p_i) = M'(p_i) - F(p_i, t) + F(t, p_i).$$

Since $M(p_i) \geq 0$ and $M'(p_i) < F(p_i, t)$, then $F(t, p_i) > 0$ and $p_i \in t^\bullet$, thus contradicting the assumption that N is a pure Petri net. □

Corollary 3.1: Let $\mathcal{P}' \supseteq \mathcal{P}(\mathcal{L})$ and N' a Petri net obtained from $\mathcal{P}'$ as described above. Then, $\mathcal{L} \subseteq L(N')$.

The following result complements Theorem 3.2 and indicates that no pure Petri net can be more precise in over-approximating the log $\mathcal{L}$.

Theorem 3.3: Let N be a Petri net derived as in Theorem 3.2. There is no pure Petri net N' such that $\mathcal{L} \subseteq L(N') \subset L(N)$.

Proof: By contradiction. Let us assume there is a pure Petri net N' such that $\mathcal{L} \subseteq L(N') \subset L(N)$. Let us consider the marking equation of N'

$$M'_0 + A'\widehat{\sigma} \geq 0$$

that also represents the intersection of a set of half-spaces, i.e., a convex polyhedron $\mathcal{P}'$. Since $\mathcal{L} \subseteq L(N')$, then $\mathcal{P}'$ also includes all $\widehat{\sigma}$ such that $\sigma \in \mathcal{L}$.

Since $L(N') \subset L(N)$, then there is some $\sigma \notin \mathcal{L}$ such that $\widehat{\sigma} \in \mathcal{P}(\mathcal{L})$ and $\widehat{\sigma} \notin \mathcal{P}'$. This contradicts the fact that $\mathcal{P}(\mathcal{L})$ is the convex hull of $\Pi(\mathcal{L})$, since the convex hull is unique. □

4 Algorithm

This section presents algorithms supporting the theoretical framework presented in Section 3. Intuitively, the main problem is to derive the convex hull of the set of points representing the Parikh vectors of a log. This problem is by no means new: there are plenty of algorithms (especially coming from computational geometry) for solving it. However, given that only efficient algorithms exist for two or three dimensions, other strategies are needed [25]. Here we use the convex polyhedra numerical abstract domain for several reasons:

- The linear inequalities representing the H-representation of a convex polyhedron can represent any pure Petri net (see Section 3.6).
- Although the convex hull construction has a computational cost, one can control it by monitoring the size of the constructed polyhedron, or by using divide and conquer techniques like the ones explained in Section 5.
- The use of *widening* [12], [26] enables the handling of large logs, at the cost of deriving underfitting models (see Section 3.1). The derivation of an underfitting model, if done in a controlled manner, may be acceptable in the scope of process mining.

Algorithm 1: IneqExtraction

Input: Parikh vectors $\widehat{\sigma_1}, \ldots, \widehat{\sigma_m}$ from a log,
f widening period,
t initial learning period,
c maximal widening application
Output: Inequality set I

```
1  begin
2      P = P_last = empty domain
3      for i ← 1 to m do
4          compute P_σ̂i
5          P = P ⊔ P_σ̂i
6          if c > 0 ∧ i > t ∧ i % f = 0 then
7              P = P∇P_last
8              P_last = P
9              c = c − 1
10         end
11     end
12     I = SelectInequalities(P)
13 end
```

4.1 The computation of the convex hull

Algorithm 1 formalizes the construction: the input of the algorithm is the set of Parikh vectors of a log ($\Pi(\mathcal{L})$), and computes a (possibly proper) subset of the inequalities from its H-representation. Any component of a Parikh vector can be seen as a constraint for the n-dimensional point that it defines. Hence, a Parikh vector $\widehat{\sigma} = (|\sigma|_{t_1}, \ldots, |\sigma|_{t_n})$ can be seen as the following polyhedron:

$$P_{\widehat{\sigma}} = \{x_1 = |\sigma|_{t_1}, \quad \ldots \quad , x_n = |\sigma|_{t_n}\}$$

For each trace σ of a log, a polyhedron $P_{\widehat{\sigma}}$ can be obtained (line 4). The polyhedron

$$P = \bigsqcup_{i \in \{1 \ldots m\}} P_{\widehat{\sigma_i}}$$

which is iteratively computed in line 5, can be obtained from the *convex-hull* operator (see Sect. 3.7) on the points represented by the polyhedra $P_{\widehat{\sigma_1}}, P_{\widehat{\sigma_2}}, \ldots, P_{\widehat{\sigma_m}}$, thus representing completely the behavior of the log.

To alleviate the complexity of the constructed polyhedron, the widening operator can be applied. Widening operators have been extensively used in order to accelerate or enforce convergence when dealing with (infinite) ascending chains of polyhedra (a detailed explanation of these issues can be found in [26]). Intuitively, a widening operator on a partial ordered set $\langle P, \sqsubseteq \rangle$ is defined as a partial function $\nabla : P \times P \to P$ satisfying:

- for all $x, y \in P$, if $x \nabla y$ is defined then $x \sqsubseteq x \nabla y$ and $y \sqsubseteq x \nabla y$
- for all increasing chains $y_o \sqsubseteq y_i \sqsubseteq \ldots$, if the increasing chain $x_o := y_o$ and $x_{i+1} := x_i \nabla y_{i+1}$ is defined for all $i \in \mathbb{N}$, then it is not strictly increasing.

Remarkably, the second condition ensures convergence in the application of widening. For instance, in Figure 7,

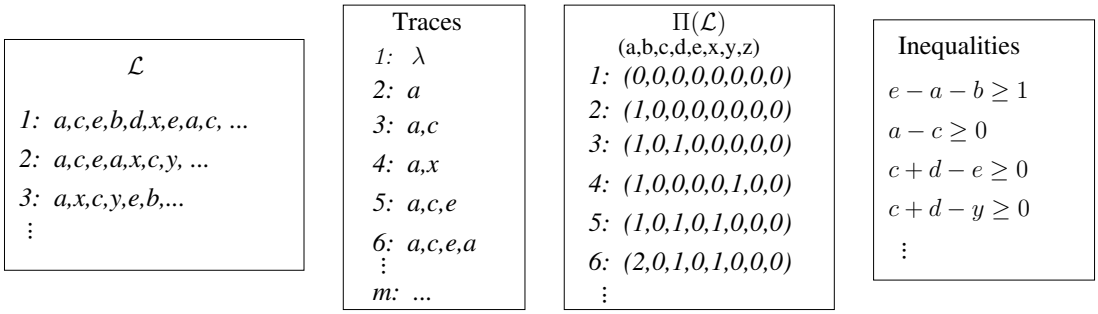

Fig. 5. From traces to invariants: (a) Initial log, (b) corresponding m traces of the log, (c) Parikh vectors associated to the traces, and (d) derived inequalities.

successive applications of the widening operator may derive the polyhedron $\mathcal{P}_i = \{a - b \leq 1\}$, which includes all the points of the log (and many more), and therefore needs not to be expanded further for representing the log. In this work, we use the *standard widening operator* from [12], although a more precise widening operator like the one explained in [26] could be also applied. In practice, this operator should be monitored in order to avoid a degradation of the derived model in some cases.

In Algorithm 1 the user can control the widening application through the parameters f, t and c (lines 6–10). The frequency at which a widening operation is applied is controlled through f. To avoid deriving a very imprecise polyhedron, it is typically required to delay the application of the widening operator until an initial admissible representation has been learned. This is the purpose of parameter t, which delays the application of widening until sufficient points have been included. The last parameter, c, sets the maximal number of applications of widening admitted.

In the last line of the algorithm, a (possibly proper) subset of the H-representation is computed (see next section for a discussion on this).

Figure 5(a) shows part of a log[4] defined on the set of events $\{a, b, c, d, e, x, y, z\}$. On Figure 5(b), the log traces are shown, and corresponding Parikh vectors are depicted in Figure 5(c). From these Parikh vectors, a unique polyhedron is derived by Algorithm 1, and the associated inequalities are extracted, some of them shown in Figure 5(d). The final Petri net is shown in Figure 6. For instance place p is obtained from the inequality $c + d - y \geq 0$.

There are two potential situations where Algorithm 1 can produce underfitting models, that are the cost for addressing two important factors: algorithmic convergence and model visualization. To enforce the convergence of the algorithm that constructs the convex hull, the widening operator has been used. As said before, this operator proved to be very effective in the iterative construction of convex hulls approximations [12], [26]. However, each time widening is applied, several points that do not appear in the log may be incorporated in the polyhedra, thus loosing the language minimality property of Theorem 3.3. In contrast, Theorem 3.2 will always hold since widening and inequalities dropping enlarges the language of the derived Petri net.

To improve the visualization of the Petri net corresponding to the convex hull computed (see Sect. 3.8), one may decide to drop some of the inequalities from the H-representation, which in turn implies that fewer places (rows of the matrix) will appear in the resulting Petri net. However, by considering only a subset of these inequalities, the polyhedron obtained may incorporate other points not appearing in the log, a problem similar to the use of widening operator. Next section contains a discussion on this selection.

4.2 Tuning the H-representation: searching a balance between overfitting and underfitting

As commented in the seminal book of Process Mining [1], one of the key challenges of process discovery techniques is to balance between *overfitting* (the derived model is too specific, allowing only the behavior observed) and *underfitting* (the derived model is too general, allowing too much unseen and unrelated behavior).

When using the complete set of inequalities from the H-representation to construct the Petri net, often one may be deriving an overfitting model. In contrast, if too many inequalities are dropped, an underfitting model may be derived. The following is a set of empirical conditions under which an inequality can be dropped without incurring in a great loss of precision:

- An inequality with a large constant b_i (e.g., $-2x_1 + 3x_2 + x_4 \leq 78$): this often represents a loose relation between the variables involved in the inequality.
- An inequality relating too many variables: if the coefficients of too many variables are non-zero, then the corresponding place in the Petri net will have too many input/output arcs, which may not correspond to the main behavior represented.
- An inequality with large coefficients (e.g., $224x_1 + 30x_2 - 11x_3 \leq 7$): it may represent a very particular

4. This log contains 100 traces of length 50 each. The reader can inspect the log by following the reference provided in [27].

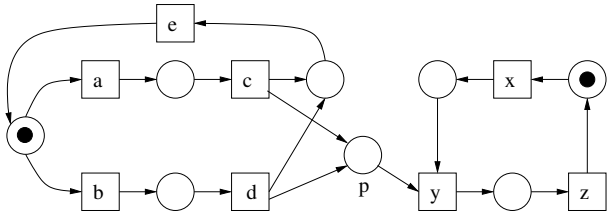

Fig. 6. Petri net derived from the inequalities shown in Figure 5(d).

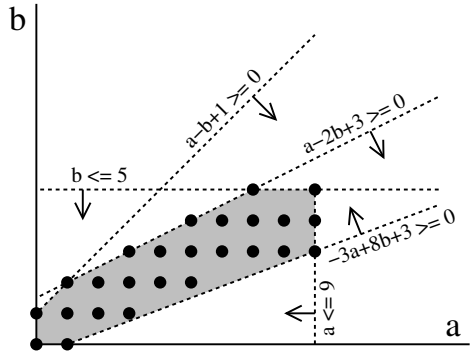

Fig. 7. An example of non-minimality of the Petri net derived.

relation between a group of variables, which may be not necessary for understanding the complete behavior.

These heuristics are included in the function SelectInequalities in line 12 of Algorithm 1. They may be turned off if one wants a minimal representation of the behavior of the log by means of a complex polyhedron. Another aspect is the criteria to determine when a coefficient is too large: this may be also heuristically determined when the whole set of inequalities in the H-representation is available.

4.3 A note on the minimality of the approach

Even when neither widening nor a proper subset of the H-representation is applied, the technique presented so far does not necessarily lead to the smallest Petri net (in number of places/arcs) such that $L(Petrinet) \supseteq L$. The following example illustrates the issue. Assume the log on events a and b contains the following Parikh vectors: $\{(0,0),(0,1),(1,0),(1,1),(1,2),(2,1),(2,2),(3,1),(3,2),(3,3),(4,2),(4,3),(5,2),(5,3),(5,4),(6,3),(6,4),(7,3),(7,4),(7,5),(8,3),(8,4),(9,3),(9,4),(9,5)\}$. The invariants computed by Algorithm 1 on these vectors are: $I = \{-3a+8b+3 \geq 0, a-b+1 \geq 0, a-2b+3 \geq 0, a \leq 9, b \leq 5\}$.

Figure 7 shows the Z-polyhedron derived from the half-spaces corresponding to each one of the inequalities. One can see that the inequality $a - b + 1 \geq 0$ is redundant: it can be safely removed while keeping the same Z-polyhedron. The existence of such redundant inequalities hampers the visualization of the Petri net, and therefore, redundant inequalities should be removed as much as possible. Techniques to simplify the representation by removing inequalities may be considered (e.g., [28]). Another possibility is to apply techniques to remove redundant places at the level of the Petri net [20].

5 High-level techniques to handle large logs: projection and sampling

The algorithm presented in the previous section cannot be applied for logs extracted from industrial/real-life applications, where either the number of events or the number of Parikh vectors in the traces are too large[5]. For these situations, other strategies are required. This section presents divide and conquer approaches that, although losing the guarantees provided by Theorems 3.2–3.3, alleviate the complexity of the algorithm presented in the previous section. These options are meant to cut the log either vertically or horizontally, by using projection or sampling, respectively. Figure 8 shows the possible strategies mentioned.

5.1 Projection

The first way of alleviating the complexity of the technique of the previous section is by reducing the number of dimensions to have in the convex polyhedron one must build to cover the log points. In this section we present a divide and conquer approach to accomplish this goal. The intuitive idea is the following:

1) Compute *clusters* of events for which there exist a high correlation between the internal events of each cluster. The clusters found need not to be a partitioning of the set of events, i.e., clusters may overlap. For example, in Figure 8, the two clusters $c_1 = \{a, b, c, d, e\}$ and $c_2 = \{x, y, z\}$ have been selected for projection.
2) Compute relations among clusters. For the example of Figure 8 there is some relation between the two clusters found, which involves variables c, d (cluster c_1) and y (cluster c_2).

In any of the two steps above, one may use only in the polyhedron construction the variables involved. Formally, projection techniques are guided to identify, among the whole set of activities T, subsets T' such that $T' \subseteq T$, $|T'| \ll |T|$ and activities in T' are correlated. Accordingly, the method described in Section 4 could be applied only for the variables in T', thus relieving the complexity of the convex-hull algorithms.

The projection algorithm presented in this section will be based on clustering activities on the set T. The clusters computed may overlap, giving rise to the derivation of relations between the variables of different clusters, which is a desired feature since this will induce model's connectivity: the transitions of related clusters will be connected by the places corresponding to the

5. The complexity of the polyhedra construction is worst-case exponential with respect to the number of dimensions [15].

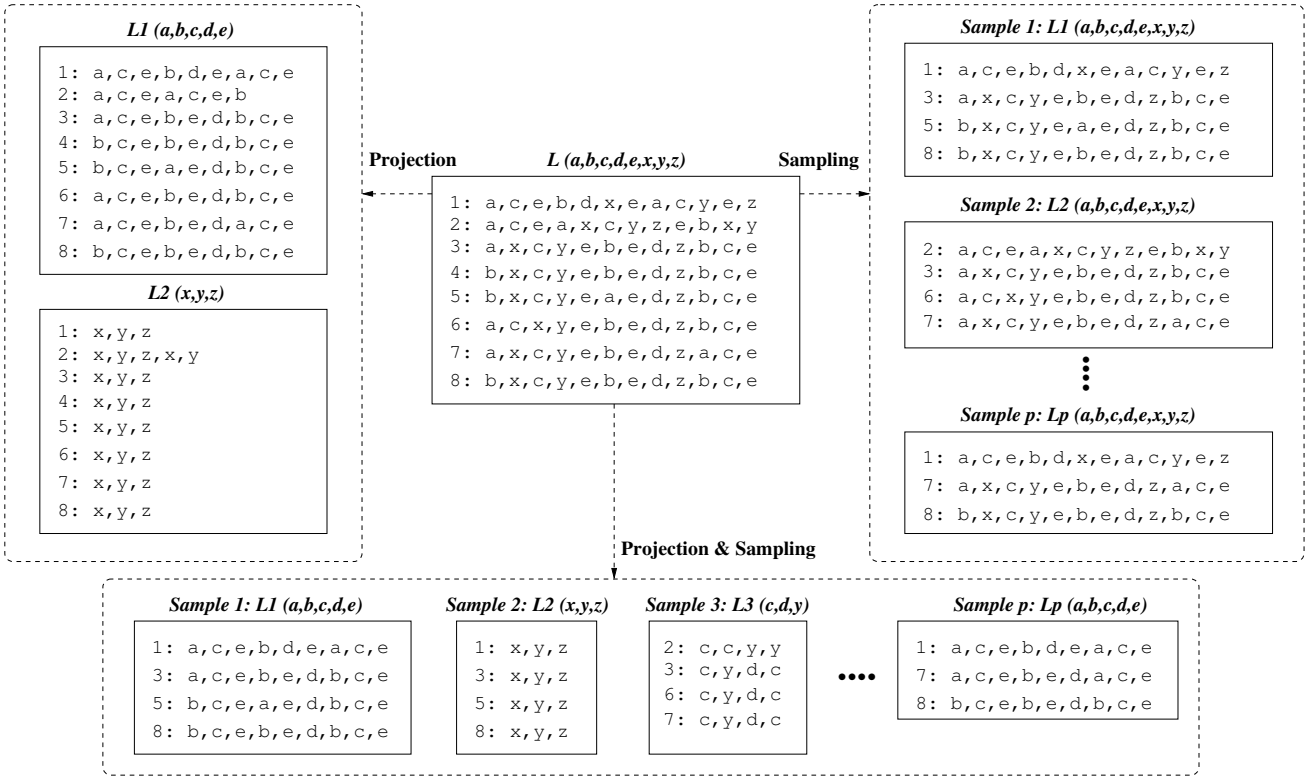

Fig. 8. Divide and conquer approaches: projection determines sets of related events and projects the log accordingly, giving rise in the figure to two projected logs. Sampling selects a maximal size (s) and a number of samples (p) and extracts from the log p samples of size at most s. Projection and sampling can be combined as it is shown in the figure.

invariants found. Models denoting processes are typically connected, and this is the reason for considering connectedness in the technique proposed.

The clustering strategy presented is based on *Principal Component Analysis (PCA)* [17], a exploratory data analysis technique that can also be applied for clustering [29]. In particular, the algorithm presented is reminiscent to the iterative techniques applied in *similarity clustering*, where clusters are extracted iteratively on the basis of the weighted similarity matrix that is updated each time a cluster is computed (the interested reader can have a detailed explanation in Chapter 6 of [30]).

Informally speaking, PCA is an exploratory data analysis technique grounded on the use of the eigenvalues/eigenvectors of the *covariance* or *correlation* matrix. This matrix is computed from the original data. We now show how to use some of the PCA steps in order to cluster related activities from a log.

Given the Parikh vectors $\widehat{\sigma_1}, \ldots, \widehat{\sigma_m}$ of a log L, the *correlation matrix* $CORR \in [-1 \ldots +1]^{|T|\times|T|}$ is

$$CORR(i,j) = \frac{\sum_{k=1}^{m}(|\sigma_k|_{t_i} - \overline{t_i})(|\sigma_k|_{t_j} - \overline{t_j})}{(m-1)s_i s_j}$$

where $\overline{t_i}$ and s_i are the mean and standard deviation of variable t_i in $\widehat{\sigma_1}, \ldots, \widehat{\sigma_m}$, respectively. This matrix measures the amount of correlation between variables t_i and t_j: when $|A(i,j)| \backsimeq 1$ then both variables are highly correlated (either directly - when $A(i,j)$ is positive, or inversely - when $A(i,j)$ is negative).

Ideally, one would like to find groups of variables that are tightly related between them and loosely related with the rest of variables of the system. For accomplishing this goal, the *eigenvalues* and *eigenvectors* of the $CORR$ matrix are used: the eigenvalues are sorted according to the variance they account for (the highest eigenvalue explains the highest variance and so on). Hence, the first eigenvalues/eigenvectors of the $CORR$ matrix carry the maximal information regarding correlation on variables.

Each eigenvalue/eigenvector can be used to find a group of related variables: given an eigenvalue λ_i, we can select a *leader* variable by looking at the corresponding eigenvector $\alpha_1 \cdot x_1 + \ldots + \alpha_n \cdot x_n$: the leader will be the variable x_i (corresponding to t_i) for which the absolute value of the coefficient α_i is maximal [17]. The row i of the CORR matrix can then be used to determine the group lead by t_i. The values $|CORR(i,j)|$ for $1 \leq j \leq |T|$ appear as points in the interval $[0 \ldots 1]$. Graphically, the distribution of points on the interval $[0 \ldots 1]$ may look like:

And the problem is to find the cluster of points closer to the right endpoint 1 (highlighted in the previous figure). The cluster of variables such that their corresponding

points are near to 1 represents variables that are in a similar high correlation with variable t_i. To compute this group, one may realize that the well-known *k-means* technique, with $k = 2$, provides a fast way to separate the points close to the left endpoint 0 from those close to the right endpoint 1. Recent advances in clustering with k-means in one dimension have provided both efficient and optimal algorithms that can be used [31].

Algorithm 2 presents the strategy. Two flags are used to control the algorithm, namely the maximal size allowed for projection (k) and the request to derive a connected model when possible (fc). The algorithm has two sequential loops: the first loop (lines 4-12) extracts groups of variables for which a high correlation exist. The second loop (lines 14-21) iteratively tries to compute inequalities that relate the variables of groups belonging to different connected components of the model corresponding to the set of inequalities computed so far.

For the first loop, the following pipeline is applied: first, the most important eigenvector of the correlation matrix and the corresponding leader variable i (the one with maximal absolute coefficient) is computed (lines 5–6). Then the absolute values of $CORR[i]$ are clustered with the 2-means algorithm to find the cluster of variables with similar high correlation (lines 7–8). In case the group found exceeds the maximal size allowed for projection, the group is truncated to force this maximal size, removing those variables which have weaker correlation within the group (line 9). Then the set of inequalities is augmented with those arising from applying Algorithm 1 on the projected Parikh vectors (line 10). To hide the relationships among the variables within a cluster and therefore induce finding a new different group in the next iteration, the correlations between the variables of the group is set to zero after computing the inequalities (line 11) (inspired from the similarity clustering techniques from [30]). The loop iterates while non-singleton groups are found.

The loop to force connectedness of the derived model uses a similar strategy: while the graph corresponding to the inequalities found so far is not connected, two unconnected components having a pair of variables with the highest correlation are found. The function Find2ClosestComps returns the variables i and j, belonging to two different components of the unconnected graph for which the correlation is maximal (line 15). Then the interval $[0 \dots 1]$ is populated both with $|CORR[i]|$ and $|CORR[j]|$, and 2-means clustering is applied to find a new group which will potentially give rise to connections between these two unconnected components (line 16). Once the group is found, the projection and the modifications to the correlation matrix is done accordingly (as in the first loop). Function *progress()* will monitor the loop to decide when its not worth to keep striving for connectedness. The possible situations where the connectedness loop is aborted are: i) the correlation matrix is totally set to zero, or ii) several trials to find inequalities between unconnected groups have fail.

Algorithm 2: ProjectionPCAMining

Input: Parikh vectors $\widehat{\sigma_1}, \dots, \widehat{\sigma_m}$
k maximal size allowed for a group,
fc flag to force a connected model
Output: Inequality set I

```
1  begin
2      I = ∅
3      CORR = CorrelationMatrix(σ̂_1, ..., σ̂_m)
4      repeat
5          v_1 = FirstEigenVector(CORR)
6          i = Leader(v_1)
7          (c_1, c_2) = TwoMeans(CORR[i])
8          G =HighCorrCluster(c_1,c_2)
9          if |G| > k then G = ForceSize(G,k)
10         I = I ∪ IneqExtraction(σ̂_1|_G, ..., σ̂_m|_G)
11         foreach i, j ∈ G do CORR[i, j] = 0
12     until |G| ≤ 1
13     if not fc then return I
14     while Unconnected(Graph(I)) and progress() do
15         (i, j) =Find2ClosestComps(Graph(I),CORR)
16         (c_1, c_2) = TwoMeans(CORR[i] ∪ CORR[j])
17         G =HighCorrCluster(c_1,c_2)
18         if |G| > k then G = ForceSize(G,k)
19         I = I ∪ IneqExtraction(σ̂_1|_G, ..., σ̂_m|_G)
20         foreach i, j ∈ G do CORR[i, j] = 0
21     end
22     return I
23 end
```

Following with the running example used in the previous section (see the resulting Petri net in Figure 6), we will show how the same Petri net can be obtained by the hierarchical approach presented in this section. The first loop of Algorithm 2 will find the two groups $g_1 = \{a, b, c, d, e\}$ and $g_2 = \{x, y, z\}$. Projecting the Parikh vectors into each group of events will give the inequalities relating only the events in the group, e.g., for group g_1 the inequalities $a - c \geq 0$, $b - d \geq 0$, $e - a - b \geq 1$ and $c + d - e \geq 0$ will be obtained. These inequalities correspond to the subnet to the left of place p in Figure 6. The right subnet corresponds to group g_2. The second loop of Algorithm 2 finds relations between the variables d, c and y and therefore will complete the missing relation $c + d - y \geq 0$, which gives rise to the place p in the figure.

5.2 Sampling

Orthogonal to the approach presented in the previous section, this section introduces a technique to avoid dealing with a large number of polyhedra and use instead a limited amount that might be enough for extracting the important relations between the events. For instance, if the log contains ten thousand traces of length a hundred, then in the worst case the techniques presented in the previous sections will need to compute the convex hull

Algorithm 3: Sampling

Input: Parikh vectors $\widehat{\sigma_1}, \ldots, \widehat{\sigma_m}$, number of samplings p, sampling size s
Output: Inequality set I

```
1  begin
2      I = ∅
3      for i ← 1 to p do
4          P = empty domain
5          for j ← 1 to s do
6              r =Random(1 ... m)
7              compute P_{σ̂_r}
8              P = P ⊔ P_{σ̂_r}
9          end
10         I_1 = inequality s(P)
11         foreach inequality i ∈ I_1 do
12             if i satisfies σ̂_1, ..., σ̂_m then I = I ∪ {i}
13         end
14     end
15 end
```

covering a million of points, a scenario that often can not be completed successfully with existing libraries for numerical abstract domains.

The general algorithm for sampling is shown in Algorithm 3. In order to avoid operations with a large number of polyhedra, one can randomly select a small set of Parikh vectors for which the convex hull will be computed (lines 5-9). Once this operation has been done, the set of inequalities *that denote properties for the Parikh vectors considered* must be verified on each one of the Parikh vectors not considered in the convex hull, and only those inequalities that are true for all the Parikh vectors will be accepted (lines 10-13). This sampling technique can be applied more than once, i.e., one can apply p samplings in order to find the relations on a set of events (external loop starting at line 3).

Sampling and the strategy presented in the previous section can be applied jointly. This will be accomplished by simply substituting the calls to Algorithm 1 in Algorithm 2 by calls to the function Sampling with a user-defined sampling size and number of samplings. In the experiments, this joint use of these strategies has enabled dealing with large specifications.

6 Experiments

The theory has been implemented in the prototype tool `aim`, which is written in `C/C++` and uses the `Apron` library for Convex Polyhedra [22][6]. For the PCA method which requires computation of eigenvalues and eigenvectors, the `ALGLIB` library [32] was used. Finally, the implementation of the optimal k-means algorithm in one dimension from [31] was incorporated into the tool. We consider in this section two versions of the tool: `aim(1)` denotes running the tool for the basic algorithm, i.e., Algorithm 1. On the other hand, `aim(p,s)` is used to denote the tool for a combination of the strategies represented by Algorithms 2 (projection) and 3 (sampling).

The experiments conducted are meant to show the capability of the algorithms of this paper for deriving interesting information (a fitting Petri net model) with very limited use of memory and in very short time. Notice that Theorems 3.2 and 3.3 provide a theoretical property on the quality of the derived models, so in this section we focus on the performance of the presented algorithms.

We have split the experiments depending on the type of logs used: the first type are synthetic logs representing well-known benchmarks in the process mining community (available within the website [33]), which will allow to illustrate the performance of Algorithms 1, 2 and 3. The second type of benchmarks are realistic logs (not synthetically created) representing typical discovery problems in a real-life scenario. For each benchmark, we report the number of events ($|T|$), the number of traces and the number of Parikh vectors obtained after removing repetitions. For each tool, the number of places discovered (P) and the number of arcs (F) is then provided, together with the CPU time (measured in a desktop computer) in seconds. For testing each tool, we limited the amount of memory and time that could be used to 1Gb and 10000 seconds respectively. For `aim(1)`, the tool was run with different parameters (c.f., Algorithm 1) depending on the log (the more complex the log, the more widening is applied), but in general satisfying $f \leq 100$, $t \geq 50$ and $c \leq 10$. For the case of `aim(p,s)`, we set $k = 10$ and $fc = true$ for all the benchmarks except for log ProdCons_3, whose corresponding model is not connected.

The synthetic logs have been used by other algorithms and therefore will be considered in this paper to perform a comparison with two other tools for the same purpose and with similar guarantees (see next section for a discussion on this). The tools are: `genet`, which implements algorithms based on the theory of regions and supports the mining of k-bounded Petri nets [10], and the `ILPMiner` [9] (within ProM), that uses the language version of the theory of regions for the same purpose. For using `genet`, an automaton representing all the traces is the input of the tool. Several algorithms exists to transform the log into an automaton [19]. For both tools we used the default parameters. We have also compared the tool with the Flexible Heuristic Miner (FHM) [34], which is widely used. Since the FHM derives *Causal nets* [1], a translation to Petri nets is used for the sake of comparison. Moreover, since the FHM incorporates invisible transitions in the derived model, we have removed some of these transitions with behavior-preserving Petri net reductions [2]. For this miner, we show in parenthesis the remaining invisible transitions for each model.

Some conclusions can be drawn from the results reported in Table 1. Regarding `aim(1)`, the use of

6. The tool is available by contacting the first author.

Log	Log Information			genet		ILPMiner		FHM		aim(1)		aim(p,s)	
	$\|T\|$	$\|\mathcal{L}\|$	$\|\Pi(\mathcal{L})\|$	P/F	Time	P/F	Time	P/F	Time	P/F	Time	P/F	Time
a12f0n00_1	12	200	18	11/25	0.1	11/25	1	12/26(0)	0	11/25	0	11/25	0
a12f0n00_5	12	1800	18	11/25	0.1	11/25	0.7	12/26(0)	0	11/25	0	11/25	0
a22f0n00_1	22	100	751	19/49	0.3	19/49	3	41/136(35)	1	15/42	0	31/77	0
a22f0n00_5	22	900	3291	19/49	0.3	19/49	23	33/98(19)	2	16/43	70	24/59	198
a32f0n00_1	32	100	1378	32/75	718	31/73	25	40/98(12)	1	23/61	10	54/127	4
a32f0n00_5	32	900	5544	31/73	1	31/73	112	40/98(12)	1	27/59	207	57/133	63
a42f0n00_1	42	100	2568	memout		44/109	154	70/198(42)	2	29/119	24	62/147	11
a42f0n00_5	42	900	15816	timeout		44/101	1557	59/146(22)	2	memout		79/196	56
ProdCons_1	8	1000	5333	7/16	14	0/0	5	7/16(0)	0	8/19	0	8/19	26
ProdCons_3	24	50	4911	timeout		0/0	182	25/65(7)	0	memout		23/54	6

TABLE 1
Petri net derivation from synthetic logs.

widening has enabled handling some of the large logs considered. When widening requires too much memory or it is too aggressive (i.e., introducing too many points not belonging to the log, see Section 4.1), one may use `aim(p,s)` to also derive valuable models in considerably shorter CPU time. Remarkably, the Petri nets derived with any of the `aim` versions have the same arcs and places of the other tools often. Often extra causalities might be obtained, denoting redundant information (unnecessary places in the model) that can be removed by a final application of well-known Petri net methods for redundant places removal [20]. This issue was already mentioned in Section 4.3. The FHM is the best miner in terms of computation time (although we do not provide the time taken to apply the invisible transition removal), but when compared with the rest of approaches, it derives significantly bigger Petri nets: the reason for this is that the translation from causal nets to Petri nets introduces plenty of *invisible* transitions that complicate considerably the model. By the application of the behavior-preserving invisible transition removal, the models have been considerably improved, but still are considerably larger than the ones derived by the rest of approaches, e.g., for the a22f0n00_1 benchmark, 35 invisible transitions remain.

The last two benchmarks of Table 1 represent the activity of a system of producers and consumers where components are synchronized through unbounded places. For ProdCons_1, the Petri net derived by any version of our tool is the one shown in Figure 6. The traces for ProdCons_3 contain the interleaving of three independent instances of Petri nets like the one in Figure 6. `genet`, FHM and the `Parikh Miner` have problems in dealing with these logs: `genet` and the FHM cannot derive the unbounded place p in ProdCons_1 and `genet` received a `timeout` for ProdCons_3, whereas the `ILPMiner` did not obtain any relation between the activities of the log[7]. The FHM could not find the In contrast, `aim(p,s)` was able to discover the exact Petri net in both logs.

7. By changing the default parameters of the `ILPMiner`, 5 places and 11 arcs are derived for ProdCons_1, but for ProdCons_3 the net is degraded (49 places, 239 arcs).

Table 2 provides similar information to the one of Table 1, but now with real-life logs. A detailed description of he DigitalCopier logs can be found in [35]. Logs BuildingPermit represent two different situations in a municipality to apply for a building permit. As in the case for synthetic benchmarks, the `aim(p,s)` tool is able to derive the process model in short time[8].

7 Related Work

In the last decade there have been proposals to solve the problem of Petri net discovery from logs. However, only the approaches grounded in the *theory of regions* [36] can be compared with the approach presented in this paper, due to the similar guarantees provided (general pure Petri nets). The rest of approaches in the literature typically impose stringent conditions on the class of Petri nets that can discover, e.g., the well-known α-algorithm [4] only considers *Structured Workflow Nets*.

Existing region-based tools for discovery can be partitioned into *language-based regions* tools [8], [9] and *state-based regions* tools [10], [11]. Informally, these approaches require as input a natural number determining the boundedness of the Petri net to derive. This is a rather strong restriction, since in order to derive the model, one must tell to the algorithm the maximal bound the model should have, which is a knowledge typically not available[9]. The theory of this paper does not need this information to find out the best possible way to represent the input log in terms of a Petri net with the necessary bounds and arc weights (c.f., Theorems 3.2 and 3.3). Additionally, the algorithms presented may also discover *unbounded* models, i.e., models representing systems for which some unbounded use of resources exist.

8 Conclusions and Future Work

In this paper we describe a novel approach for the discovery of Petri nets from event logs. When compared

8. Although the size of the problem of logs BuildingPermit is larger than some logs of Table 1, the projections derived by Algorithm 2 were smaller, thus requiring less computation time.

9. In [10], an strategy with exponential complexity is proposed to determine such bound.

Log	Log Information			aim(l)		aim(p,s)	
	$\|T\|$	#traces	#Parikh	P/F	Time	P/F	Time
DigitalCopier_0	33	750	749	30/78	16m23s	55/122	0
DigitalCopier_1	33	300	585	28/64	53	63/141	0
BuildingPermit_0	22	10000	2618	10/30	76	21/47	1
BuildingPermit_1	47	5000	87188	memout		61/131	50

TABLE 2
Petri net derivation from realistic logs.

with existing approaches in the literature, the approach overcomes the current limitations. The first restriction is the type of Petri nets current approaches can derive, which is significantly extended in this paper (general pure Petri nets). The second limitation, for those techniques in the literature with similar features like the one of this paper, is the knowledge the discovery algorithm needs in order to perform the discovery: the approaches presented in this paper require no knowledge of the model to derive, e.g., no knowledge on the bound of the Petri net to compute. Moreover, we describe divide and conquer strategies for projecting the log, guided to alleviate the complexity of the convex hull construction. Additionally, we show how sampling can help into reducing the complexity of the problem by considering only small fractions of the log.

The algorithms of this paper have been implemented into a tool, and experimental results devoted to evaluate its performance have shown significant improvements when compared with the approaches in the literature.

As future work, we plan to explore strategies for filtering noise and guiding the discovery of behavioral elements that can be identified in the Petri net structure, e.g., the resource part in a manufacturing system.

ACKNOWLEDGEMENTS

We thank A. Adriansyah and R. P. J. C. Bose for providing some of the logs. This work is supported by projects FORMALISM (TIN2007-66523) and TIN2011-22484.

REFERENCES

[1] W. M. P. van der Aalst, *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[2] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.

[3] J. Carmona and J. Cortadella, "Process mining meets abstract interpretation," in *ECML/PKDD (1)*, ser. Lecture Notes in Computer Science, J. L. Balcázar, F. Bonchi, A. Gionis, and M. Sebag, Eds., vol. 6321. Springer, 2010, pp. 184–199.

[4] W. M. P. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE TKDE*, vol. 16, no. 9, pp. 1128–1142, 2004.

[5] L. Wen, W. M. P. van der Aalst, J. Wang, and J. Sun, "Mining process models with non-free-choice constructs," *Data Min. Knowl. Discov.*, vol. 15, no. 2, pp. 145–180, 2007.

[6] L. Wen, J. Wang, W. van der Aalst, B. Huang, and J. Sun, "A novel approach for process mining based on event types," *Journal of Intelligent Information Systems*, vol. 32, pp. 163–190, 2009.

[7] W. M. P. van der Aalst, A. K. A. de Medeiros, and A. J. M. M. Weijters, "Genetic process mining," in *Applications and Theory of Petri Nets 2005, 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005, Proceedings*, ser. Lecture Notes in Computer Science, vol. 3536. Springer, 2005, pp. 48–69.

[8] R. Bergenthum, J. Desel, R. Lorenz, and S.Mauser, "Process mining based on regions of languages," in *Proc. 5th Int. Conf. on Business Process Management*, Sep. 2007, pp. 375–383.

[9] J. M. E. M. van der Werf, B. F. van Dongen, C. A. J. Hurkens, and A. Serebrenik, "Process discovery using integer linear programming," in *Petri Nets*, ser. LNCS, K. M. van Hee and R. Valk, Eds., vol. 5062. Springer, 2008, pp. 368–387.

[10] J. Carmona, J. Cortadella, and M. Kishinevsky, "New region-based algorithms for deriving bounded Petri nets," *IEEE Transactions on Computers*, vol. 59, no. 3, pp. 371–384, 2009.

[11] M. Solé and J. Carmona, "Light region-based techniques for process discovery," *Fundam. Inform.*, vol. 113, no. 3-4, pp. 343–376, 2011.

[12] P. Cousot and R. Cousot, "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints." ACM Press, 1977, pp. 238–252.

[13] ——, "Static determination of dynamic properties of programs," in *Proc. of the 2nd Int. Symposium on Programming*. Dunod, Paris, France, 1976, pp. 106–130.

[14] A. Miné, "The octagon abstract domain," in *Analysis, Slicing and Tranformation (in Working Conference on Reverse Engineering)*, ser. IEEE. IEEE CS Press, Oct. 2001, pp. 310–319. [Online]. Available: http://www.di.ens.fr/ mine/publi/article-mine-ast01.pdf

[15] P. Cousot and N. Halbwachs, "Automatic discovery of linear restraints among variables of a program." ACM Press, New York, 1978, pp. 84–97.

[16] A. Rozinat and W. M. P. van der Aalst, "Conformance checking of processes based on monitoring real behavior," *Inf. Syst.*, vol. 33, no. 1, pp. 64–95, 2008.

[17] I. T. Jolliffe, *Principal Component Analysis*. Springer, 2002.

[18] L. Recalde, M. Silva, J. Ezpeleta, and E. Teruel, "Petri nets and manufacturing systems: An examples-driven tour," in *Lectures on Concurrency and Petri Nets*, ser. Lecture Notes in Computer Science, vol. 3098. Springer, 2003, pp. 742–788.

[19] W. M. P. van der Aalst, V. Rubin, H. M. W. Verbeek, B. F. van Dongen, E. Kindler, and C. W. Günther, "Process mining: a two-step approach to balance between underfitting and overfitting," *Software and System Modeling*, vol. 9, no. 1, pp. 87–111, 2010.

[20] M. Silva, E. Teruel, and J. Colom, *Linear Algebraic and Linear Programming Techniques for the Analysis of Place/Transition Net Systems*, ser. Lecture Notes in Computer Science. Springer-Verlag, 1998, no. 1491, pp. 309–373.

[21] R. T. Rockafellar, *Convex Analysis*. Princeton University Press, 1970.

[22] B. Jeannet and A. Miné, "Apron: A library of numerical abstract domains for static analysis," in *CAV*, ser. Lecture Notes in Computer Science, A. Bouajjani and O. Maler, Eds., vol. 5643. Springer, 2009, pp. 661–667.

[23] K. Fukuda, S. Picozzi, and D. Avis, "On canonical representations of convex polyhedra," in *Proc. of the First International Congress of Mathematical Software*, 2002, pp. 350–360.

[24] O. Ore, "Galois connexions," *Transactions of the American Mathematical Society*, vol. 55, pp. 493–513, 1944.

[25] D. Avis, D. Bremner, and R. Seidel, "How good are convex hull algorithms?" *Computational Geometry*, vol. 7, no. 56, pp. 265 – 301, 1997, 11th ACM Symposium on Computational Geometry.

[26] R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella, "Precise widening operators for convex polyhedra," *Sci. Comput. Program.*, vol. 58, no. 1-2, pp. 28–56, 2005.
[27] "Example log," http://www.lsi.upc.edu/∼jcarmona/prodcons1000.tr.
[28] G. Frehse, "Phaver: Algorithmic verification of hybrid systems past hytech," in *HSCC*, ser. Lecture Notes in Computer Science, M. Morari and L. Thiele, Eds., vol. 3414. Springer, 2005, pp. 258–273.
[29] C. H. Q. Ding and X. He, "*K*-means clustering via principal component analysis," in *ICML*, ser. ACM International Conference Proceeding Series, C. E. Brodley, Ed., vol. 69. ACM, 2004.
[30] A. N. Gorban, B. Kgl, D. C. Wunsch, and A. Zinovyev, *Principal Manifolds for Data Visualization and Dimension Reduction*. Springer, 2007.
[31] H. Wang and M. Song, "Ckmeans.1d.dp: Optimal *k*-means clustering in one dimension by dynamic programming," *The R Journal*, vol. 3, no. 2, pp. 29–33, 2011.
[32] "ALGLIB library," http://www.alglib.net.
[33] "Process mining," www.processmining.org.
[34] A. J. M. M. Weijters and J. T. S. Ribeiro, "Flexible heuristics miner (FHM)," in *CIDM*, 2011, pp. 310–317.
[35] R. J. C. Bose, "Process Mining in the Large: Preprocessing, Discovery, and Diagnostics," Ph.D. dissertation, Eindhoven University of Technology, 2012.
[36] A. Ehrenfeucht and G. Rozenberg, "Partial (Set) 2-Structures. Part I, II," *Acta Informatica*, vol. 27, pp. 315–368, 1990.

Josep Carmona received the MS and PhD degrees in Computer Science from the Technical University of Catalonia, in 1999 and 2004, respectively. He is an associate professor in the Software Department of the same university. His research interests include formal methods, concurrent systems, and process and data mining. He has co-authored more than 50 research papers in conferences and journals. In 2009 he received the best paper award at the International Conference on Application of Concurrency to System Design.

Jordi Cortadella received the M.S. and Ph.D. degrees in Computer Science from the Universitat Politècnica de Catalunya, Barcelona, in 1985 and 1987, respectively. He is a Professor in the Department of Software of the same university. In 1988, he was a Visiting Scholar at the University of California, Berkeley. His research interests include formal methods and computer-aided design of VLSI systems with special emphasis on asynchronous circuits, concurrent systems and logic synthesis. He has co-authored numerous research papers and has been invited to present tutorials at various conferences.

Dr. Cortadella has served on the technical committees of several international conferences in the field of Design Automation and Concurrent Systems. He received best paper awards at the Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (2004), the Design Automation Conference (2004) and the Int. Conf. on Application of Concurrency to System Design (2009). In 2003, he was the recipient of a Distinction for the Promotion of the University Research by the Generalitat de Catalunya.