Divide-and-Conquer Strategies for Process Mining

J. Carmona[1], J. Cortadella[1], and M. Kishinevsky[2]

[1] Universitat Politècnica de Catalunya, Spain
[2] Intel Corporation, USA

Abstract. The main goal of Process Mining is to extract process models from logs of a system. Among the possible models to represent a process, Petri nets is an ideal candidate due to its graphical representation, clear semantics and expressive power. The theory of regions can be used to transform a log into a Petri net, but unfortunately the transformation requires algorithms with high complexity. This paper provides techniques to overcome this limitation. Either by using decomposition techniques, or by clustering events in the log and working on projections, the proposed techniques can be used to alleviate the complexity and make the theory of regions practical for real-life problems.

1 Introduction

Process Mining [20] is an emerging area that has arised in the last decade. The basic goal is to extract knowledge from event logs recorded in information systems. Several researchers have provided in the last years algorithms to mine formal models from logs, most of them included in the ProM framework [19].

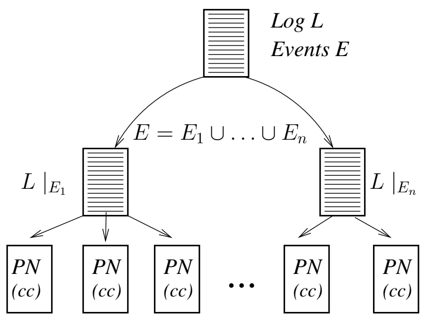
The *synthesis problem* [9] is related to process mining: it consists in building a Petri net that has a behavior equivalent to a given transition system. The problem was first addressed by Ehrenfeucht and Rozenberg [10] introducing *regions* to model the sets of states that characterize marked places. In the area of synthesis, some approaches have been studied to take the theory of regions into practice. In [2] polynomial algorithms for the synthesis of bounded nets were presented. This approach has been recently adapted for the problem of process mining in [3]. In [6], the theory of regions was applied for the synthesis of safe Petri nets with bisimilar behavior. Recently, the theory from [6] has been extended to bounded Petri nets [5].

Process mining differs from synthesis in the knowledge assumption: while in synthesis one assumes a complete description of the system, only a partial description of the system is assumed in process mining. Therefore, bisimulation is no longer a goal to achieve in process mining. Instead, obtaining approximations that succinctly represent the log under consideration are more valuable [22]. However, synthesis can be adapted for process mining in two ways: either the log is encoded as a transition system (introducing state information, as described in [18]) and state-based methods for mining [4] are applied, or language-based methods are used directly on the log [3, 21].

Due to its complexity, it is clear that the region-based approach might become impractical when dealing with large logs. In this paper, we present methods to alleviate significantly the complexity of the region-based approach. Two approaches are presented to this end:

- A decomposition approach to find a set of components (conservative Petri nets), each one describing a partial view of the log. This approach avoids the exhaustive computation of regions and instead applies local search of regions (inspired on the notion of *allocation* from Hack [12]) until a component is detected. The set of components can either be composed to form a unique Petri net or presented separately. This approach is described in Section 3.
- A *divide-and-conquer* approach to split the log into pieces, by means of projection. The method selects groups of events tightly related in the log for which the decomposition approach will be applied, projecting the log on these events. When neither the classical region-based mining nor the decomposition approach presented in this paper are able to handle a large log, this aggressive technique has proven to be very successful. The approach is presented in Section 4.

In both approaches, the goal is to offer a set of partial views of the behavior observed in the log, by means of a set of Petri nets whose parallel composition can reproduce any trace observed in the log.



Let us illustrate the idea of the divide-and-conquer approach (see Figure on the right): given a log L with set of events E, using some ordering relations of the events appearing in the log, derive a causal dependency graph of the set of the events. This graph is then cut into several pieces, each piece representing a set of events tightly related by causal dependencies (in the figure, the sets $E_1 \ldots E_n$ are found). Finding a good partitioning is a problem on its own, but several approaches can be used to this end, including *graph cut algorithms* [11, 14] or *spectral graph theory* [7]. Then the log is projected for each one of the sets of events. The decomposition method presented in this paper is then applied for each projection, obtaining a set of conservative components that covers the events in the log that were considered in the projection.

2 Basic Theory

2.1 Finite Transition Systems and Petri Nets

Definition 1 (Transition system). *A* transition system *(*TS*) is a tuple* (S, E, A, s_{in})*, where* S *is a set of* states*,* E *is an alphabet of* actions*,* $A \subseteq S \times E \times S$ *is a set of* (labelled) transitions*, and* $s_{in} \in S$ *is the* initial state.

We will use $s \xrightarrow{e} s'$ as a shortcut for $(s, e, s') \in A$, and the transitive closure of this relation will be denoted by $\xrightarrow{*}$. Let $\mathsf{TS} = (S, E, A, s_{in})$ be a transition system. We consider connected TSs that satisfy the following axioms:

– S and E are finite sets.
– Every event has an occurrence: $\forall e \in E : \exists (s, e, s') \in A$;
– Every state is reachable from the initial state: $\forall s \in S : s_{in} \xrightarrow{*} s$.

The *language* of a TS, $L(\mathsf{TS})$, is the set of traces feasible from the initial state. When $L(\mathsf{TS}_1) \subseteq L(\mathsf{TS}_2)$, we will denote TS_2 as an over-approximation of TS_1. Given a trace $\sigma \in L(\mathsf{TS})$ and a set $A \subseteq E$, $\sigma \mid_A$ is the trace resulting of removing from σ all events in $E - A$. Analogously, $\mathsf{TS} \mid_A$ is the TS that arises after contracting all transitions of events in $E - A$.

Definition 2 (Petri net [15]). *A Petri net (*PN*) is a tuple* (P, T, F, M_0) *where* P *and* T *represent finite and disjoint sets of places and transitions, respectively, and* $F \subseteq (P \times T) \cup (T \times P)$ *is the flow relation. The initial marking* $M_0 \subseteq P$ *defines the initial state of the system*[3]*.*

The sets of input and output transitions of place p in PN N are denoted by ${}^{\bullet}_{N}p$ and $p^{\bullet}_{N}$, respectively (we omit the subscript indicating the net if the context is clear). The set of all markings reachable from the initial marking m_0 is called its Reachability Set. The *Reachability Graph* of PN ($\mathsf{RG}(\mathsf{PN})$) is a transition system in which the set of states is the Reachability Set, the events are the transitions of the net and a transition (m_1, t, m_2) exists if and only if $m_1 \xrightarrow{t} m_2$. We use $L(\mathsf{PN})$ as a shortcut for $L(\mathsf{RG}(\mathsf{PN}))$.

We now define the concept of *parallel composition* of PNs [23], that will be used extensivelly along the paper.

Definition 3 (Parallel Composition).
Given PN*s* G_1 *and* G_2*, their parallel composition is denoted by* $G_1 || G_2 = (P_{||}, T_{||}, F_{||}, m_{0||})$*, where:*

$$\begin{aligned}
P_{||} &= P_1 \times \{*\} \cup \{*\} \times P_2 \\
T_{||} &= \{(t, t) \mid t \in T_1 \cap T_2 \\
&\quad \cup \{(t_1, *) \mid t_1 \in T_1 - T_2 \\
&\quad \cup \{(*, t_2) \mid t_2 \in T_2 -_1 \\
F_{||} &= \{((p_1, p_2), (t_1, t_2)) \mid (p_1, t_1) \in F_1 \ \mathit{or}\ (p_2, t_2) \in F_2\} \\
&\quad \cup \{((t_1, t_2), (p_1, p_2)) \mid (t_1, p_1) \in F_1 \ \mathit{or}\ (t_2, p_2) \in F_2\} \\
m_{0||}((p_1, p_2)) &= \begin{cases} m_{01}(p_1) \ \mathit{if}\ p_1 \in P_1 \\ m_{02}(p_2) \ \mathit{if}\ p_2 \in P_2 \end{cases}
\end{aligned}$$

[3] For the sake of clarity, we restrict the region theory of this section to the class of *elementary net systems*: 1-bounded Petri nets without loops. The theory for the general case (k-bounded weighted Petri nets) is described in [4, 5], and the theory of the rest of the paper is applicable for the general case.

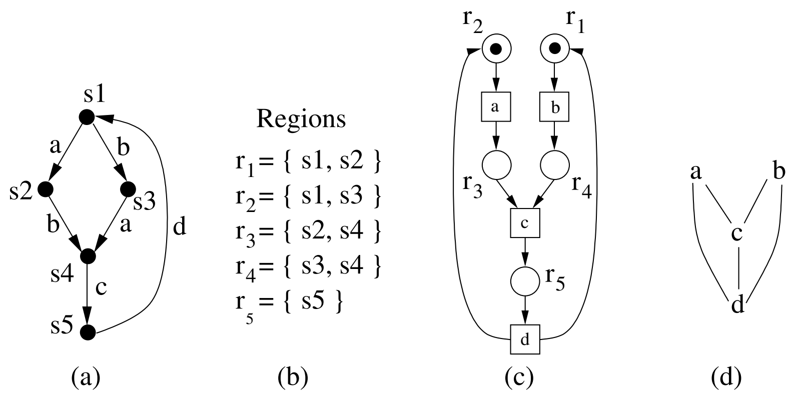


Fig. 1. (a) Transition system, (b) regions, (c) N_{TS}, (d) Causal dependency graph.

2.2 Regions and Region-based Synthesis

We now review the classical theory of regions for the synthesis of Petri nets [6,9, 10]. Let S' be a subset of the states of a TS, $S' \subseteq S$. If $s \notin S'$ and $s' \in S'$, then we say that transition $s \xrightarrow{a} s'$ *enters* S'. If $s \in S'$ and $s' \notin S'$, then transition $s \xrightarrow{a} s'$ *exits* S'. Otherwise, transition $s \xrightarrow{a} s'$ *does not cross* S'.

The notion of a *region* is central for the synthesis of PNs. Intuitively, each region is a set of states that corresponds to a place in the synthesized PN, so that every state in the region models the marking of the place.

Definition 4 (region). *A set of states $r \subseteq S$ in $\mathsf{TS} = (S, E, A, s_{in})$ is called a* region *if for each event $e \in E$, exactly one of the three predicates (*enters*,* exits *or* does not cross*) holds for all its transitions.*

Hence, a region is a subset of states in which *all* transitions labelled with the same event e have exactly the same "entry/exit" relation. This relation will become the predecessor/successor relation in the Petri net. Examples of regions are reported in Figure 1: from the TS of Figure 1(a), some regions are enumerated in Figure 1(b). For instance, for region r_2, event a is an exit event, event d is an entry event while the rest of events do not cross the region.

Each TS has two *trivial regions*: the set of all states, S, and the empty set. Further on we will always consider only non-trivial regions. The set of non-trivial regions of TS will be denoted by R_{TS}. A region r is a *pre-region* of event e if there is a transition labelled with e which exits r. A region r is a *post-region* of event e if there is a transition labelled with e which enters r. The sets of all pre-regions and post-regions of e are denoted with ${}^{\circ}e$ and e°, respectively. By definition it follows that if $r \in {}^{\circ}e$, then all transitions labelled with e exit r. Similarly, if $r \in e^{\circ}$, then all transitions labelled with e enter r.

Algorithm: PN synthesis on the set of regions R

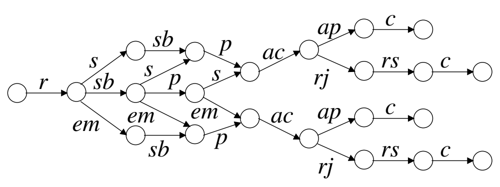
- For each event $e \in E$ generate a transition labelled with e in the PN;
- For each region $r_i \in R$ generate a place r_i;
- Place r_i contains a token in the initial marking iff the corresponding region r_i contains the initial state of the TS s_{in};
- The flow relation is as follows: $e \in r_i\bullet$ iff r_i is a pre-region of e and $e \in \bullet r_i$ iff r_i is a post-region of e, i.e.,

$$F_R \overset{def}{=} \{(r,e) | r \in R_{\mathsf{TS}} \ \wedge \ e \in E \ \wedge \ r \in {}^{\circ}e\} \\ \cup \{(e,r) | r \in R_{\mathsf{TS}} \ \wedge \ e \in E \ \wedge \ r \in e^{\circ}\}$$

Fig. 2. Algorithm for Petri net synthesis from [10].

1	`r,s,sb,p,ac,ap,c`
2	`r,sb,em,p,ac,ap,c`
3	`r,sb,p,em,ac,rj,rs,c`
4	`r,em,sb,p,ac,ap,c`
5	`r,sb,s,p,ac,rj,rs,c`
6	`r,sb,p,s,ac,ap,c`
7	`r,sb,p,em,ac,ap,c`

(a)



(b)

Fig. 3. (a) event log, (b) corresponding transition system.

The algorithm given by [10] to synthesize a PN, $N_{\mathsf{TS}} = (R, E, F_R, R_{s_{in}})$, from an *elementary transition system*[4] $\mathsf{TS} = (S, E, A, s_{in})$ and a set of regions R, is illustrated in Figure 2. An example of the application of the algorithm is shown in Figure 1. The initial TS and a set of regions is reported in Figures 1(a) and (b), respectively. The synthesized PN is show in Figure 1(c). When the TS is elementary, running algorithm of Figure 2 on the set of non-trivial regions R_{TS} derives a PN such that $L(\mathsf{PN}) = L(\mathsf{TS})$ [10].

Given an event e, $\mathsf{ER}(e)$ denote the set of states where event e is enabled (Excitation Region), and $\mathsf{SR}(e)$ the set of states reached when firing e in a state from $\mathsf{ER}(e)$ (Switching Region)[5]. These sets will be used to compute the ordering relations between events (see below).

2.3 Deriving transitions systems from logs

For a complete understanding of the approach presented in this paper, it is necessary to show how to transform a log into a TS, which is the starting point

[4] Elementary transition systems are a proper subclass of the TS considered in this paper, where additional conditions to the ones presented in Section 2.1 are required.

[5] Excitation and switching regions are not regions in the terms of Definition 4. The terms are used due to historical reasons.

of our algorithms. The theory described in [18] presents many variants for solving this problem. The basic idea to incorporate state information is to look at the pre/post history of a subtrace in the log. Figure 3 shows an example, where states are decided by looking at the set of common prefixes.

2.4 Trigger Relations and its Graph

In this section we present a relation on events, similar to the *log-based ordering relation* [20], but which is defined in the TS. It is based on the ER/SR sets.

Definition 5 (Causal Dependency Graph). *Given a* $\mathsf{TS} = (S, E, A, s_{in})$*, and two events* $a, b \in E$*:*

1. a triggers b *(*$a \rightarrow_{\mathsf{TS}} b$*) if* $\mathsf{SR}(a) \cap \mathsf{ER}(b) \neq \emptyset$ *and* $\mathsf{ER}(a) \cap \mathsf{SR}(b) = \emptyset$*, and*
2. a is concurrent to *(*$a \parallel_{\mathsf{TS}} b$*)* b *if* $\mathsf{SR}(a) \cap \mathsf{ER}(b) \neq \emptyset$ *and* $\mathsf{ER}(a) \cap \mathsf{SR}(b) \neq \emptyset$*.*

The causal dependency graph *over* TS*, denoted CDG(*TS*), is the undirected graph (E,M), with* $M \subseteq E \times E$ *such that* $(a, b) \in M$ *iff* $a \rightarrow_{\mathsf{TS}} b$ *or* $b \rightarrow_{\mathsf{TS}} a$*.*

For instance, the causal dependency graph of the transition system of Figure 1(a) is depicted in Figure 1(d).

3 Computation of Conservative Components

The goal of this section is, given a TS, derive a set of conservative components whose parallel composition contains all the traces possible in the TS. For the sake of simplicity, we will restrict the definitions for the case of conservative 1-bounded nets, known as *state machines* [15]. At the end of the section we show how to generalize the theory for arbitrary k-bounded nets.

Let us illustrate the theory of this section revisiting the example of Figure 1. From the set of regions reported $(r_1 \ldots r_5)$, there are two subsets that correspond to *partitions* of the set of states in the transition system of Figure 1(a) (depicted in Figures 4(a) and (c)). For instance, the subset r_1, r_4 and r_5 forms a partition. The main idea is: when a subset R of regions is a partition, then the synthesis algorithm from Figure 2 applied on R will derive a conservative Petri net, i.e. a Petri net where the number of tokens is preserved. Figure 4(b) and (d) show the two Petri nets corresponding to each partition, respectively.

3.1 State machines and its State-based Representation

We start by defining formally the concept of subnet and state machine component:

Definition 6 (Subnet). *A triple* $N' = (P', T', F')$ *is a subnet of a net* $N = (P, T, F)$ *if* $P' \subseteq P$*,* $T' \subseteq T$ *and* $F' = F \cap ((P' \times T') \cup (T' \times P'))$*.*

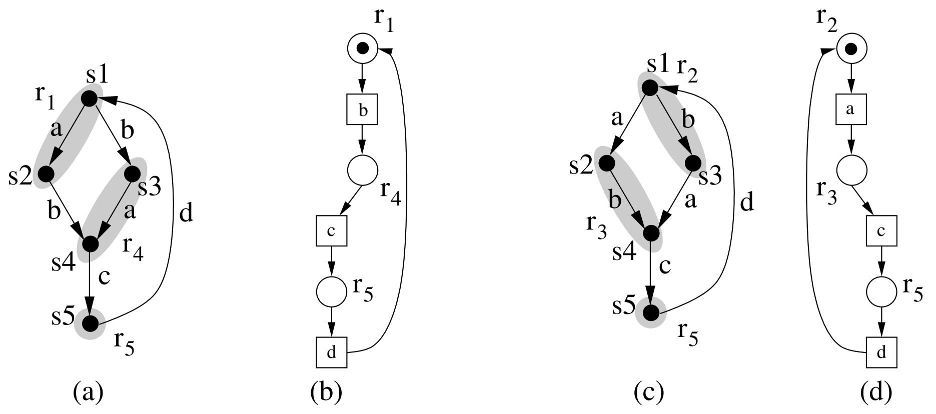


Fig. 4. Example of conservative components decomposition for the example of Figure 1: (a) Partition of the transition system on regions r_1, r_4 and r_5, and (b) the derived state machine. Partition (c) and derivation (d) for regions r_2, r_3 and r_5.

Definition 7 (State Machine Component). *A* state machine component *(SMC) $N' = (P', T', F')$ of a net N is a subnet of N such that*

1. *for every $t \in T' : |{}^\bullet_{N'}t| = |t^\bullet_{N'}| = 1$, and*
2. *for every $p \in P'$, $({}^\bullet_N p \cup p^\bullet_N) \subseteq T'$*

An SMC of a PN *(N,M_0) is a pair (N',M'_0) such that N' is a SMC of N, for every $p \in P' : M'_0(p) = M_0(p)$ and $\sum_{p\in P'} M'_0(p) = 1$.*

The following theorem states the main result of this section:

Theorem 1. *Let $\mathsf{TS} = (S, E, A, s_{in})$, and consider the net $N_{\mathsf{TS}} = (R_{\mathsf{TS}}, E, F_{R_{\mathsf{TS}}}, R_{\mathsf{TS}_{s_{in}}})$ obtained by the algorithm of Figure 2 on R_{TS}. Given a set of regions $R \subseteq R_{\mathsf{TS}}$, if R forms a partition of S, then algorithm of Figure 2 on R defines an SMC of N_{TS}.*

Proof. Assume the contrary, i.e. R is a partition of S and there is a transition e in SMC with at least two predecessor places $r_i, r_j \in R$ (the case for successor places is analogous). According to algorithm of Figure 2, $r_i \in {}^\bullet e$ iff $r_i \in {}^\circ e$, and the same applies for r_j. If $r_i \in {}^\circ e$ ($r_j \in {}^\circ e$), then every state $s \in S$ such that $(s, e, s') \in A$ satisfies $s \in r_i$ ($s \in r_j$), because otherwise r_i (r_j) will not be a pre-region of e. This implies that every state s such that $(s, e, s') \in A$ satisfies $s \in r_i \cap r_j$. But then $r_i \cap r_j \neq \emptyset$ and therefore R is not a partition, a contradiction. □

3.2 Allocation-based SMC Computation

In Hack's thesis [12], the idea of *allocation* was introduced to decompose a Free-choice Petri net into a set of safe and conservative components (S-components).

Algorithm 1: SMCComputation

```
Input: Transition system $\mathsf{TS} = (S, E, A, s_{in})$, event $ev \in E$
Output: Set of regions $R$ forming a partition of $S$
1  begin
2      $R \longleftarrow \emptyset$
3      $Evs \longleftarrow \{ev\}$
4      $r_i \longleftarrow$ PickOneRegion($\{r | r \in {}^{\circ}ev\}$)
5      $r_j \longleftarrow$PickOneRegion($\{r | r \in ev^{\circ} \wedge r \cap r_i = \emptyset\}$
6      $PendingRegs \longleftarrow \{r_i, r_j\}$
7      $Part \longleftarrow \{r_i, r_j\}$
8      repeat
9          $r \longleftarrow$ RemoveOneRegion($PendingRegs$)
10         forall $e \in E - Evs : e \in {}^{\circ}r \cup r^{\circ}$ do
11             $r_i \longleftarrow$ PickOneRegion($\{r | r \in {}^{\circ}e \wedge r \cap Part = \emptyset\}$)
12             $r_j \longleftarrow$ PickOneRegion($\{r | r \in e^{\circ} \wedge r \cap Part = \emptyset\}$)
13             if $r_i \neq \emptyset \vee r_j \neq \emptyset$ then
14                 $Evs \longleftarrow Evs \cup \{e\}$
15                 $R \longleftarrow R \cup \{r_i, r_j\}$
16                 $PendingRegs \longleftarrow PendingRegs \cup \{r_i, r_j\}$
17                 $Part \longleftarrow Part \cup \{r_i, r_j\}$
18             end
19         end
20     until $PendingRegs = \emptyset \vee Part = S$
21     if $Part \subset S$ then $R \longleftarrow \{S\}$
22 end
```

The idea is to select a-priori, among the places in the pre-set of a transition, the one that will be in the pre-set of the transition in the constructed S-component.

Following the idea of allocation from Hack's thesis, we present a method to derive an SMC from a given TS. Algorithm 1 describes the iterative process of finding regions until a partition of the states in TS is computed. Due to Theorem 1, the set of regions R forms an SMC. The idea of the algorithm is: starting from an initial event ev and two arbitrary regions in ${}^{\circ}ev$ and ev° (lines 4-5 of the algorithm), keep growing a partition by iteratively including pre-post regions of new events until the partition equals the set of states in TS or no more regions can be found (lines 8-20). In the algorithm, the set $PendingRegs$ contains all the regions to be explored, and $Part$ represents the partition constructed so far. Finally, it might be possible that no region is found as a pre/post-region of a particular event. In this situation the function PickOneRegion will assign the empty set to the corresponding regions (i.e. to r_i or r_j), and therefore the set $PendingRegs$ will not be increased. If the set of regions found are not enough as to form a partition of S, the trivial region S is returned and therefore the SMC will simply be the initial event with a self-loop place (line 21).

The general method to find a set of SMCs that cover every event of the TS is described in Algorithm 2. At each iteration i, it tries to find a new SMC SMC_i

Algorithm 2: SMCDecomposition

```
Input: Transition system $\mathsf{TS} = (S, E, A, s_{in})$
Output: Set of SMCs
        $SMC_1 = (R_1, E_1, F_1, M_{0,1}) \ldots SMC_n = (R_n, E_n, F_n, M_{0,n})$
1  begin
2      $X \longleftarrow E$
3      $i \longleftarrow 1$
4      repeat
5          $ev \longleftarrow$ RemoveOneEvent($X$)
6          $R_i \longleftarrow$ SMCComputation($\mathsf{TS}$,$ev$)
7          $E_i \longleftarrow \{e | e \in ({}^\circ r \cup r^\circ) \ \wedge \ r \in SMC_i \ \wedge \ r \subset S\} \cup \{ev\}$
8          $F_i \longleftarrow \{(r, e) | e \in r^\circ \wedge r \in R_i \wedge e \in E_i\} \cup \{(e, r) | e \in {}^\circ r \wedge r \in R_i \wedge e \in E_i\}$
9          $M_{0,i} \longleftarrow \forall r \in R_i : M_{0,i}(r) = r(s_{in})$
10         $X \longleftarrow X - E_i$
11         $i \longleftarrow i + 1$
12     until $X = \emptyset$
13 end
```

that covers one of the events still not covered by any SMC_j, for $j < i$. When a given event ev can only be found by the trivial region S, the set E_i only contains the event ev and therefore the SMC derived will be a self-loop place on event ev.

Property 1. Algorithm 1 derives an SMC, and $L(\mathsf{TS}) \subseteq L(SMC)$.

Property 2. Given the set of regions $R_1 \ldots R_n$ found by Algorithm 2, $\bigcup_{i=1\ldots n} R_i \subseteq R_{\mathsf{TS}}$.

Property 2 ensures that the set of regions needed to cover the events is at most the set of non-trivial regions. A complexity alleviation (with respect to classical synthesis methods) can be obtained when the set of regions computed by Algorithm 2 is a proper subset. Section 5 shows examples of this.

Informally, the parallel composition of n Petri nets is a Petri net where every transition with the same label in two or more components represents a synchronization point for the corresponding components [23]. See Definition 3 for a formal definition. The following theorem can be proven:

Theorem 2. *Let* $SMC_1 = (R_1, E_1, F_1, M_{0,1}) \ldots SMC_n = (R_n, E_n, F_n, M_{0,n})$ *be the set of components found by Algorithm 2 on* $\mathsf{TS} = (S, E, A, s_{in})$*. Then* $L(\mathsf{TS}) \subseteq L(SMC_1 \parallel \ldots \parallel SMC_n)$*.*

Proof. First, consider the net N_{TS} as the result of applying algorithm of Figure 2 on R_{TS}. Second, Property 2 guarantees that the set of regions found by Algorithm 2 in every SMC_i is a subset of R_{TS}. This implies that the set of predecessor (successor) places of every transition in $SMC_1 \parallel \ldots \parallel SMC_n$ is a subset (maybe proper) of the predecessor (successor) set of the transition in N_{TS}. In other words, the places and arcs in $SMC_1 \parallel \ldots \parallel SMC_n$ are a subset of places and arcs in N_{TS}. Finally, provided that the initial marking of $SMC_1 \parallel \ldots \parallel SMC_n$ is equal to the one in N_{TS} for the joint places, we have that every trace in N_{TS} is also a trace of $SMC_1 \parallel \ldots \parallel SMC_n$. This can be proven by induction on the length of a trace $\sigma \in LN_{\mathsf{TS}}$: if $|\sigma| = 1$, it represents a transition enabled in the initial marking, which will be also enabled in $SMC_1 \parallel \ldots \parallel SMC_n$ because the transition has the same or less predecessor places and therefore are marked according to Definition 3. Assume true for $|\sigma| = n$. If $|\sigma| = n + 1$, using the induction hypothesis we can fire in $SMC_1 \parallel \ldots \parallel SMC_n$ the n first transitions, and provided that places and arcs in $SMC_1 \parallel \ldots \parallel SMC_n$ are a subset of places and arcs in N_{TS}, the tokens produced to enable in N_{TS} the last transition of σ are also produced (maybe a proper subset) to enable the transition in $SMC_1 \parallel \ldots \parallel SMC_n$ after firing the first n transitions. This proves the trace inclusion $L(N_{\mathsf{TS}}) \subseteq L(SMC_1 \parallel \ldots \parallel SMC_n)$. But provided that $L(\mathsf{TS}) \subseteq L(N_{\mathsf{TS}})$ [4], then then language of the parallel composition is also a superset of $L(\mathsf{TS})$. $\square$

Algorithm 2 is nondeterministic: depending on the order of events selected, a different set of state machines can arise. This has an impact both in the quality of the overapproximation obtained and complexity of the method, measured in the number of regions necessary. In the future, more elaborated strategies can be build on top of the approach presented to address these concerns.

3.3 Covering the Causal Dependency Graph

The causal dependency graph can be used to improve the quality of the generated parallel composition: if some causal dependency between a pair of events is not transferred to an SMC with a shared place of the corresponding transitions, one can try to derive a new SMC that contains this relation.

Let $a \longrightarrow_{\mathsf{TS}} b$ be an ordering relation found in the TS. If the set

$$\{r \mid \mathsf{SR}(a) \cup \mathsf{ER}(b) \subseteq r \wedge r \in R_{\mathsf{TS}}\}$$

is not empty, then any region of this set may be used to try to find an SMC covering the ordering relation $a \longrightarrow_{\mathsf{TS}} b$. Algorithm 1 can be adapted to search for some region in this set that derives a non-trivial SMC (i.e. different from the self-loop place SMC) containing the causality relation between a and b. This is done by adapting the PickOneRegion predicates to search for regions r in the set above.

3.4 Generalization to Arbitrary Conservative Components

The theory presented in the previous sections can be lifted to the general case. Informally, in a conservative component the sum of weights in incoming arcs of a transition is equal to the sum of weights in outgoing arcs. To derive conservative components for bounds greater than one, the idea is to generalize the notion of partition, deriving the concept of *multipartition*. Let us first define the concept of P/T net to generalize Definition 2:

Definition 8 (P/T net). *A P/T 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. The initial marking $M_0 \in \mathbb{N}^{|P|}$ defines the initial state of the system.*

We can generalize the notion of state machine from Definition 7 to derive the concept of conservative component:

Definition 9 (Conservative Component). *A* conservative component *(CC) $N' = (P', T', F')$ of a net N is a connected subnet of N such that:*

1. *For every $t \in T'$, $\sum_{p \in {}^\bullet_{N'} t} F'(p, t) = \sum_{p \in t^\bullet_{N'}} F'(t, p)$*
2. *For every $p \in P'$, $({}^\bullet_N p \cup p^\bullet_N) \subseteq T'$*

A CC of a PN *(N, M_0) is a pair (N', M'_0) such that N' is a CC and for every $p \in P' : M'_0(p) = M_0(p)$.*

And assuming the notion of k-bounded region [5] where a region r assigns a cardinality $0 \ldots k$ to every state s (i.e. $0 \leq r(s) \leq k$), we can define the concept of multipartition:

Definition 10 (Multipartition). *A set $R = \{r_i\}$ of nonempty multisets forms a* multi-partition *of a set S if there is a constant $k \in \mathbb{N}^+$ such that:*

1. *$\forall s \in S : \sum_{r_i \in R} r_i(s) = k$,*
2. *$\forall i \neq j : r_i \neq r_j$.*

The case $k = 1$ corresponds to a partition.

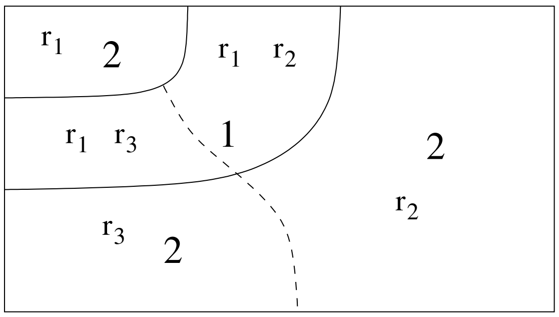


Figure on the right illustrates a multipartition for $k = 2$ in an abstract way. It shows three regions r_1, r_2 and r_3 by labelling cardinality zones with the names of the regions in the whole set of states. For instance, region r_1 has cardinality 2 for states in the top-left corner, whereas it has cardinality 1 for the set of states surrounding its cardinality-2 states. On these states, part of it intersects with cardinality-1 states of r_2, and part of it intersects with cardinality-1 states of r_3. It is easy to show that it is a multipartition since for each state the sum of the cardinalities assigned by r_1, r_2 and r_3 equals 2.

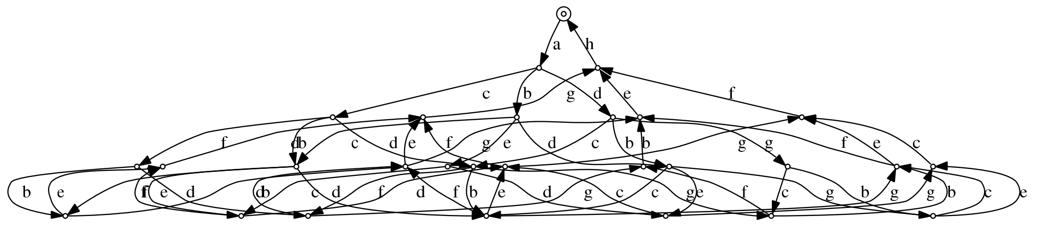


Fig. 5. Transition system.

Theorem 3. *Let* $\mathsf{TS} = (S, E, A, s_{in})$*, and consider the net* $N_{\mathsf{TS}} = (R_{\mathsf{TS}}, E, F_{R_{\mathsf{TS}}}, R_{\mathsf{TS}\,s_{in}})$ *obtained by the Algorithm of Figure 2 on* R_{TS}*. Given a set of regions* $R \subseteq R_{\mathsf{TS}}$*, if*

– *R forms a multi-partition of S, then it defines a CC of* N_{TS}*,*
– *R forms a partition of S, then it defines an SMC of* N_{TS}*.*

Proof. The proof is a generalization of the proof provided for Theorem 1. □

4 A Divide-and-Conquer Approach for Petri Net Mining

This section presents the second contribution of this paper: to face the complexity required for dealing with large TSs, an approach is presented to project the TS into tightly related events, dealing smaller TSs. Then the decomposition approach presented in the previous section can be applied on these smaller TSs.

4.1 Introductory Example

Let us illustrate the idea with the toy example from Figure 5, representing the behavior $(A; ((B; E) \parallel (C; F) \parallel (D; G)); H)$, in a TS having 28 states. In Figure 6(a), we illustrate the causal dependency graph. Our goal is to find balanced partitions of the causal dependency graph by means of cuts.

Figure 6(b,top) reports a minimal cut obtained from the graph of Figure 6(a), namely $\{C, F\}$. Notice that, provided that we are interested in conservative components that are synchronized with common events, when projecting the behavior of the initial TS into the set of events found in the cut we include the events outside of the cut which are adjacent to vertices in the cut, e.g. events A and H in the figure (these events are called *border* events). The same happens when the second cut $\{B, E\}$ is reported on the reduced graph (Figure 6(c)). From each one of the sets of events found, the TS from Figure 5 is projected onto them and a conservative component covering the events in the projection is found (this is shown in the bottom part of each cut).

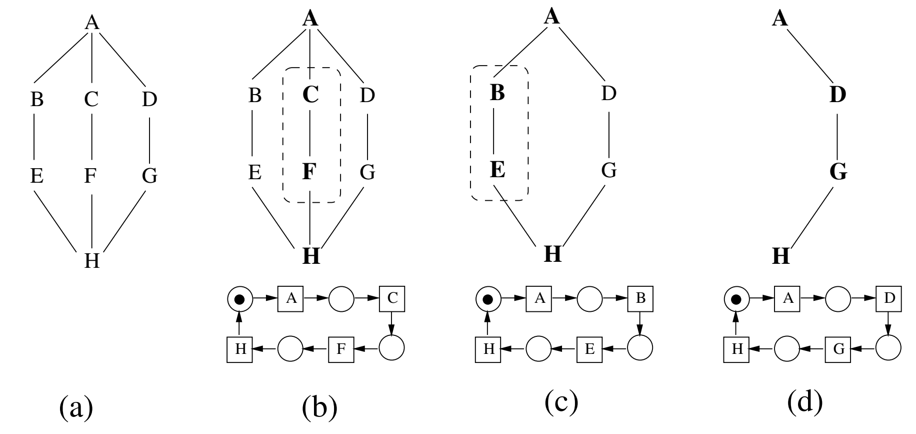


Fig. 6. (a) Causal dependency graph, (b)-(d) (Top) Consecutive cuts of the causal dependency graph, (Bottom) State machines covering each cut.

4.2 Causal Dependency Graph Partitioning

There exist several techniques for the partitioning of a graph into a set of clusters [11, 14]. In this section we show one of these techniques, that does not have to be the most efficient nor the optimal, but it was easy to implement. It consists on iteratively finding bi-partitions until some halting criterion is reached.

In order to find a balanced partition of $CDG(\mathsf{TS}) = (E, M)$ into two sets, let us use the well known RatioCut [13] metric. Given a partition $E_1 \dots E_n$ of the set E, the metric is defined as:

$$RatioCut(E_1 \dots E_n) = \sum_{i=1}^{n} \frac{cut(E_i, \overline{E_i})}{|E_i|}$$

where $\overline{E_i}$ denotes the complement of set E_i, and $cut(A, B) = |\{(i, j)|(i, j) \in M \wedge i \in A, j \in B\}|$.

If only two sets (i.e. a bi-partition) are used in the previous formula, the following optimization problem can be considered:

$$\min_{A \subset E} RatioCut(A, \overline{A})$$

i.e. finding the best bi-partition for the given graph. A way to approximate the optimal solution to this optimization problem is by using the *Fielder* vector, which is the eigenvector corresponding to the second smallest eigenvalue of the (unnormalized) Laplacian matrix $L = D - A$, where D is the *degree matrix* of the nodes in the causal dependency graph, and A its adjacency matrix [7].

More concretely, if $f \in \mathbb{R}^{|E|}$ is the Fiedler vector, then a bipartition (E_1, E_2) can be obtained as follows:

$$\begin{cases} e \in E_1 \text{ if } f_e \geq 0 \\ e \in E_2 \text{ otherwise} \end{cases}$$

Algorithm 3: DivideAndConquerMining

```
    Input: $\mathsf{TS} = (S, E, A, s_{in})$, MaxSize
    Output: Set of SMCs
            $SMC_1 = (R_1, E_1, F_1, M_{0,1}) \dots SMC_n = (R_n, E_n, F_n, M_{0,n})$
1   begin
2       Compute $\longrightarrow_{\mathsf{TS}}$, $\|_{\mathsf{TS}}$ event relations
3       $(E_1 \dots E_n) \longleftarrow$ GraphPartition(CDG($\mathsf{TS}$),MaxSize)
4       forall $E_i$ do
5           $E_i \longleftarrow$ AddBorderEvents($\mathsf{TS}$,$E_i$,$E$)
6           SMCDecomposition($\mathsf{TS} \mid_{E_i}$)
7       end
8   end
```

By iteratively finding bi-partitions, one can derive n partitions of the set of events of the causal dependency graph, as it has been done for the causal dependency graph from the example of Section 4.1. This iteration can be terminated using a halting criterion: number of events in the projection, size of the log, CPU time-limit for the region computation, among others. In our approach, provided that we are interested in the mining of conservative components, the degree of concurrency between the events and the maximal size allowed has been used to decide if further partitioning is required.

4.3 Divide-and-Conquer Approach

Algorithm 3 presents the approach. First it computes the causal and concurrent relations (see Definition 5) present in the TS (line 2). Then the causal dependency graph is partitioned into n sets (n is an output of the method, and is dependant on the MaxSize parameter). Finally, the computation of SMCs covering each projection is applied (lines 4-7). Notice that in order to avoid the derivation of independent SMCs, i.e. SMCs without common events, each set E_i is augmented with border events, i.e. events in $E - E_i$ that are adjacent in the causal dependency graph to some event in E_i (line 5). The following lemma is crucial into providing a relation between the initial TS and the set of nets derived:

Lemma 1. *Let* $\mathsf{TS} = (S, E, A, s_{in})$*,* PN_1 *and* PN_2 *such that* $L(\mathsf{TS} \mid_{E_i}) \subseteq L(\mathsf{PN}_i)$*, for* $i = 1, 2$*. Then* $L(\mathsf{TS} \mid_{E_1 \cup E_2}) \subseteq L(\mathsf{PN}_1 \parallel \mathsf{PN}_2)$*.*

Proof. (We show the case $E_1 \cap E_2 \neq \emptyset$, because the other is trivial): by induction on the length of traces in $L(\mathsf{TS} \mid_{E_1 \cup E_2})$. For traces of length one, it is easy to see that events enabled in s_{in} correspond to transitions in $\mathsf{PN}_1 \parallel \mathsf{PN}_2$ that are initially enabled. Assume it holds for traces of length at most n. Induction step: let $\sigma a \in L(\mathsf{TS} \mid_{E_1 \cup E_2})$, with $|\sigma| = n$, and assume $a \in E_1 \cap E_2$ (the other cases are particular instances of this one). By definition, $\sigma \mid_{E_1} a \in L(\mathsf{TS} \mid_{E_1})$. Hence $\sigma \mid_{E_1} a \in L(\mathsf{PN}_1)$, i.e. every event that puts a token in a place from ${}^{\bullet}a$

is fired in $\sigma \mid_{E_1}$. The same reasoning can be done for PN_2, and given that the only predecessor places of a in $\mathsf{PN}_1 \parallel \mathsf{PN}_2$ are the union of predecessor places of a in PN_1 and PN_2 (see Definition 3), and σ is possible in $\mathsf{PN}_1 \parallel \mathsf{PN}_2$ due the induction hypothesis, a is enabled in $\mathsf{PN}_1 \parallel \mathsf{PN}_2$ after firing σ. $\square$

The following theorem provides the main result of this section:

Theorem 4. *Let* $SMC_1 = (R_1, E_1, F_1, M_{0,1}) \ldots SMC_n = (R_n, E_n, F_n, M_{0,n})$ *be the set of components found by Algorithm 3 on* $\mathsf{TS} = (S, E, A, s_{in})$*. Then* $L(\mathsf{TS}) \subseteq L(SMC_1 \parallel \ldots \parallel SMC_n)$*.*

Proof. It can be shown by successive applications of Lemma 1 on sets of events until $E_1 \cup E_2 = E$. $\square$

5 Experiments

The theory described in Sections 3 and 4 has been incorporated into the tool `Genet` [4, 5]. The first experiments were conducted to test the ability to rediscover conservative components from well-structured descriptions, i.e. to apply Algorithms 1 and 2, and its corresponding generalizations (as described in Section 3.4). To this end, the TS of the following k-bounded Petri nets was used:

1. A model for n processes competing for m shared resources, where $n > m$. Figure 7(a) describes the Petri net,
2. A model for m producers and n consumers, where $m > n$. Figure 7(b) describes the Petri net.
3. A 2-bounded pipeline of n processes. Figure 7(c) describes the Petri net.

Table 1 reports the first experiment: comparing mining (-pm) versus conservative components derivation (-cc). For each benchmark, the size of the transition system considered (states and arcs), together with the number of places and transitions derived by the k-bounded mining method described in [4] is given. Finally, the number of conservative components found by Algorithm 2 and the sum of all the places found in the components is reported (the number of transitions in the conservative components derivation is equal to the number of transitions in the mining approach and is not reported). The CPU time is provided for each one of the approaches. For each example, Figure 7 provides gray boxes with the conservative components found (some of the boxes share transitions, i.e. they will synchronize on the firing of the transition in the parallel composition of the components). In conclusion, the derivation of conservative components might overcome the complexity problems of the region-based method, sometimes without the inclusion of extra behavior in the resulting Petri net.

The second experiment was to have some confidence on the quality of the approach presented in Section 3. For that end, we used the *fitness* factor, described in [17]. Fitness evaluates whether the mined net complies with the log,

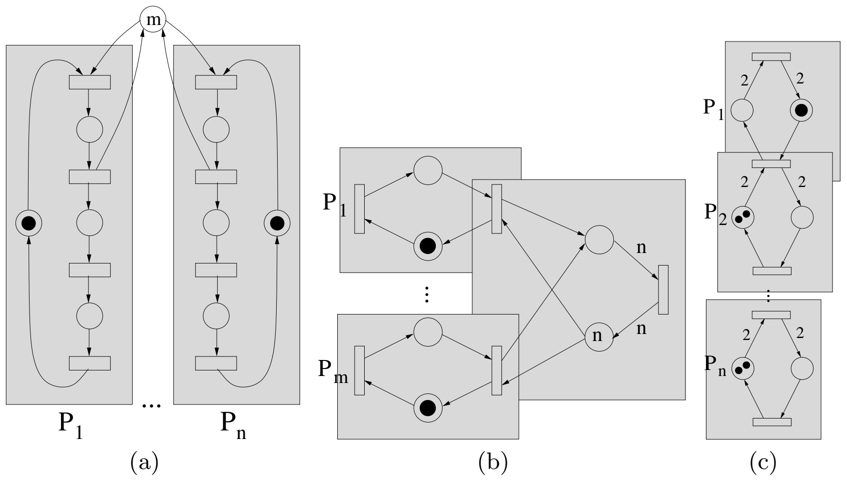


Fig. 7. Parameterized benchmarks: (a) n processes competing for m shared resources, (b) m producers and n consumers, (c) a 2-bounded pipeline of n processes. Each box represents a conservative component found.

and it is one of the main measures provided by the *Conformance checker* within ProM. Numerically, fitness ranges from 1 (good) to 0 (bad). The table on the right reports the fitness of some miners within ProM, and the fitness of the net corresponding the parallel composition of the SMCs computed by Algorithm 2. The three logs used are the illustrative logs described in [17]. We report Unb when the net obtained is unbounded and therefore no conformance checking can be done. In summary, numbers in the table are promising for our approach, and we believe they can improve if techniques like the ones presented in Section 3.3 are additionally applied.

Log	α	α++	Parikh	Heuristic	`Genet-cc`
L1	0.83	0.80	Unb	0.84	0.85
L2	0.84	0.81	Unb	0.85	0.86
L3	Unb	0.55	Unb	0.62	0.58

The third experiment was to test the divide-and-conquer mining approach described in Section 4. We have used two types of examples: logs from [1], and a real-life system modelling a complex module that controls the operation of optical lithography process for mass chip production [16]. Both types of benchmarks are difficult to mine using the region-based mining approach described in [4]. Table 2 compares the classical region-based mining and divide-and-conquer mining for these benchmarks. We report the size of the transition system, and columns $|P|$, $|[S]|$ report the number of places and size of the corresponding reachability graph of the mined Petri net. For the divide-and-conquer mining, columns $|Bis|$, k, $|CC|$, $|P|$ and $|T_U|$ report the number of bisections performed on the causal dependency graph (see Section 4.3), the bound used in the conservative component generation, the total number of conservative components found, the total number of places found and the number of events not covered by any place (the less events uncovered, the better), respectively. We use mem to report that the approach aborted due to memory problems.

			Genet-pm			Genet-cc		
benchmark	$\lvert S\rvert$	$\lvert E\rvert$	$\lvert P\rvert$	$\lvert T\rvert$	CPU	$\lvert CC\rvert$	$\lvert P\rvert$	CPU
SharedResource(5,2)	918	4320	21	20	0s	5	20	0s
SharedResource(4,3)	255	1016	17	16	0s	4	16	0s
SharedResource(6,4)	4077	24372	25	24	18s	5	24	5s
SharedResource(7,5)	16362	114408	29	28	25m	7	28	47s
ProducerConsumer(3,3)	32	92	8	7	0s	4	8	0s
ProducerConsumer(4,3)	64	240	10	9	0s	5	10	0s
ProducerConsumer(6,3)	256	1408	14	13	0s	7	14	0s
ProducerConsumer(8,3)	1024	7424	18	17	2s	9	18	0s
ProducerConsumer(8,5)	1536	11520	18	17	1h10m	9	18	25m
BoundedPipeline(6)	729	1539	12	7	6s	6	12	4s
BoundedPipeline(7)	2187	5103	14	8	48s	7	14	40s
BoundedPipeline(8)	6561	16767	16	9	12m	8	16	11m
BoundedPipeline(9)	19683	54765	18	10	1h50m	9	18	1h30m

Table 1. Synthesis versus derivation of conservative components

				Genet-pm safe			Genet-pm 2-bounded			Genet-rec k-bounded					
benchmark	$\lvert S\rvert$	$\lvert A\rvert$	$\lvert E\rvert$	$\lvert P\rvert$	$\lvert[S]\rvert$	CPU	$\lvert P\rvert$	$\lvert[S]\rvert$	CPU	$\lvert Bis\rvert$	k	$\lvert CC\rvert$	$\lvert P\rvert$	$\lvert T_U\rvert$	CPU
pn_ex_10	233	479	11	13	281	0s	16	145	4s	3	2	3	9	0	0s
a12f0n50_1	78	77	11	17	80	0s	39	63	52s	3	2	4	23	0	0s
a12f0n50_2	151	150	11	21	92	0.5s	119	96	15m	3	2	8	19	0	5s
a12f0n50_3	188	187	11	21	92	0.5s	178	102	21m	1	2	4	13	0	5s
a22f0n00_1	1209	1208	20	16	78	9m	–	–	mem	0	1	4	30	0	5s
a22f0n00_2	3380	3379	20	16	78	15m	–	–	mem	3	1	6	24	1	4s
a22f0n00_3	5334	5333	20	16	78	32m	–	–	mem	3	1	7	32	1	7s
WaferStepper	55043	289443	27	–	–	mem	–	–	mem	3	6	9	28	5	5m

Table 2. Mining versus divide-and-conquer mining.

The conclusion from Table 2 is the ability to handle large systems for the divide-and-conquer approach. For instance, the mining of 2-bounded Petri nets applied to the a12f0n50 examples needs two orders of magnitude more CPU time than the divide-and-conquer approach. Moreover, the Petri nets resulting from mining are sometimes very complex (e.g. the 2-bounded Petri net mined for the bench a12f0n50_3 contains 178 places) and will not be very useful as visualization objects. Instead, the divide-and-conquer approach provides simple views of the complex behavior considered.

The WaferStepper benchmark requires a comment: it represents a large and complex system. It needs a 6-bounded Petri net to be modelled. Of course, the region-based mining method can not deal with systems of such size for the given bound, and therefore, partitioning methods like the divide-and-conquer method presented in this paper might be the only option for systems like this.

6 Related Work

The approach presented in this paper can be seen as hierarchical method: the initial problem is partitioned into small pieces for which a solution is more likely to exist, due to the size reduction obtained. Together with the approach presented in [8], to the best of our knowledge there is no other approach for Petri net mining that uses this strategy.

The approach presented in this paper differs from the clustering approach presented in [8] in the following:

1. In this approach we give a special emphasis into the mining of conservative components, i.e. Petri nets that describe sequential and conflict dependencies between events.
2. In [8] the partition is on the set of instances (traces) of the log, i.e. the log is horizontally partitioned, whereas in our approach the separation is done on the set of events hence the log is vertically partitioned.
3. The partition approach presented in this paper is related to the Petri net derivation applied afterwards, in the sense that events tightly related by causal dependencies are likely to become in the same conservative component. In contrast, the partition approach presented in [8] uses a different principle: each trace is projected into the most relevant features (computed previously) and associated with a vector of values. Then the *k-means* algorithm is used to partition the vectorial space defined by the traces.

Nevertheless, given that both approaches are orthogonal, they can be applied together to improve the individual application of each one.

7 Conclusions and Future Work

High-level and decomposition approaches are usually required to solve large problems. This paper shows that the region-based technique for process mining can also be solved using these type of approaches. First, the decomposition approach enables the search for sequential views of the process that might be more useful than the complete process itself. Second, when the size of the log forbids the application of classical or decomposition mining, the divide-and-conquer method presented in this paper alleviates the complexity of computing regions by projecting the TS into the events that are likely to be related, thus decreasing considerably its size.

This paper shows the empirical evidence that the approach can significantly reduce the complexity of the region-based mining approaches. What remains to be studied is the quality of the derived results. We plan to evaluate different variants of the approach presented in this paper, to learn heuristics that can help into reducing the undeterminism of the algorithms presented. The goal is to obtain similar quality measures as the ones that can be obtained with the classical region-based approach for process mining.

Acknowledgements

We would like to thank E. Verbeek for providing the WaferStepper example. This work has been supported by the CICYT project FORMALISM (TIN2007-66523), and a grant by Intel Corporation.

References

1. Process mining. www.processmining.org.
2. E. Badouel, L. Bernardinello, and P. Darondeau. Polynomial algorithms for the synthesis of bounded nets. *Lecture Notes in Computer Science*, 915:364–383, 1995.
3. 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*, pages 375–383, Sept. 2007.
4. J. Carmona, J. Cortadella, and M. Kishinevsky. A region-based algorithm for discovering Petri nets from event logs. In M. Dumas, M. Reichert, and M. C. Shan, editors, *BPM*, volume 5240 of *Lecture Notes in Computer Science*, pages 358–373. Springer, 2008.
5. J. Carmona, J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. A symbolic algorithm for the synthesis of bounded Petri nets. In *29th International Conference on Application and Theory of Petri Nets and Other Models of Concurrency*, June 2008.
6. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri nets from finite transition systems. *IEEE Transactions on Computers*, 47(8):859–882, Aug. 1998.
7. D. Cvetković, P. Rowlinson, and S. Simić. *Eigenspaces of Graphs.* Cambridge University Press, 1997.
8. A. A. de Medeiros, A. Guzzo, G. Greco, W. van der Aalst, A. Weijters, B. van Dongen, and D. Sacca. Process mining based on clustering: A quest for precision. In B. B. A. ter Hofstede and H. Paik, editors, *BPM 2007 International Workshops (BPI, BPD, CBP, ProHealth, RefMod, Semantics4ws)*, volume 4928 of *Lecture Notes in Computer Science*, pages 17–29. Springer, 2008.
9. J. Desel and W. Reisig. The synthesis problem of Petri nets. *Acta Inf.*, 33(4):297–315, 1996.
10. A. Ehrenfeucht and G. Rozenberg. Partial (Set) 2-Structures. Part I, II. *Acta Informatica*, 27:315–368, 1990.
11. C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *DAC '82: Proceedings of the 19th conference on Design automation*, pages 175–181, Piscataway, NJ, USA, 1982. IEEE Press.
12. M. Hack. *Analysis of production schemata by Petri nets.* M.s. thesis, MIT, Feb. 1972.
13. L. W. Hagen and A. B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 11(9):1074–1085, 1992.
14. B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(1):291–307, 1970.
15. T. Murata. Petri Nets: Properties, analysis and applications. *Proceedings of the IEEE*, pages 541–580, Apr. 1989.

16. A. J. Pretorius. *Visualization of State Transition Graphs.* PhD thesis, Technical University of Eindhoven, 2008.
17. A. Rozinat and W. M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Inf. Syst.*, 33(1):64–95, 2008.
18. W. van der Aalst, V. Rubin, H. Verbeek, B. van Dongen, E. Kindler, and C. Günther. Process mining: A two-step approach to balance between underfitting and overfitting. Technical Report BPM-08-01, BPM Center, 2008.
19. W. M. P. van der Aalst, B. F. van Dongen, C. W. Günther, R. S. Mans, A. K. A. de Medeiros, A. Rozinat, V. Rubin, M. Song, H. M. W. E. Verbeek, and A. J. M. M. Weijters. ProM 4.0: Comprehensive support for *eal* process analysis. In J. Kleijn and A. Yakovlev, editors, *ICATPN*, volume 4546 of *Lecture Notes in Computer Science*, pages 484–494. Springer, 2007.
20. W. M. P. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–1142, 2004.
21. 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 K. M. van Hee and R. Valk, editors, *Petri Nets*, volume 5062 of *Lecture Notes in Computer Science*, pages 368–387. Springer, 2008.
22. H. Verbeek, A. Pretorius, W.M.P. van der Aalst, and J.J. van Wijk. On Petri-net synthesis and attribute-based visualization. In *Proc. Workshop on Petri Nets and Software Engineering (PNSE'07)*, pages 127–141, June 2007.
23. W. Vogler. Modular construction and partial order semantics of Petri nets. In *LNCS 625*. Springer-Verlag, 1992.