Divide-and-Conquer Strategies
for Process Mining

Josep Carmona[1], Jordi Cortadella[1], and Michael Kishinevsky[2]

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

Abstract. The 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 approach can be used to widen the applicability of classical region-based techniques.

1 Introduction

The goal of Process Mining [20] is to extract knowledge from event logs recorded in information systems. Several researchers have provided algorithms to mine formal models from logs, most of them included in the ProM framework [19].

The *synthesis problem* [10] 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 [11] introducing *regions* to model the sets of states that characterize marked places. In the area of synthesis, some techniques have been proposed to take the theory of regions into practice. In [2] polynomial algorithms for the synthesis of bounded nets were presented. These algorithms have been recently adapted for the problem of process mining in [3]. In [7], the theory of regions was applied for the synthesis of safe Petri nets with bisimilar behavior. Recently, the theory from [7] has been extended to bounded Petri nets [6].

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. 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 [5] are applied, or language-based methods are used directly on the log [3,21]. In this paper we follow the first approach.
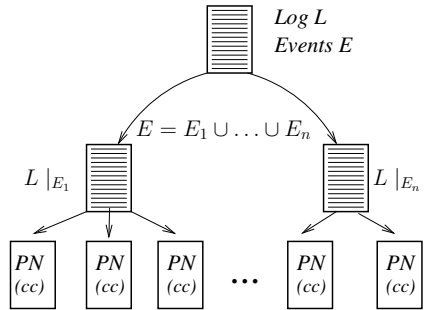
Due to its complexity, it is clear that the theory of regions might become impractical when dealing with large logs. In this paper, we present methods to

alleviate significantly the complexity of the region-based approach. Two techniques are presented to this end:

- A decomposition method 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 [13]) until a component is detected. The set of components can either be composed to form a unique Petri net or presented separately. It is described in Section 3.
- A *divide-and-conquer* method 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 technique will be applied, projecting the log on these events. When neither the classical region-based mining nor the decomposition approach are able to handle a large log, this aggressive technique has proven to be very successful. It 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* [14, 12] or *spectral graph theory* [8]. Then the log is projected for each one of the sets of events. The decomposition method of this paper is then applied for each projection, obtaining a set of Petri nets (PN) that covers the traces in the log.

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: i) S and E are finite sets, ii) every event has an occurrence and iii) every state is reachable from the initial state.

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*[1].

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 ($\mathsf{RG(PN)}$) 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 $L(\mathsf{RG(PN)})$.

2.2 Regions and Region-Based Synthesis

We now review the classical theory of regions for the synthesis of Petri nets [11, 10,7]. 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 3 (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.

---

[1] 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 [6,5], and the theory of the rest of the paper is applicable for the general case, as demonstrated in [4].
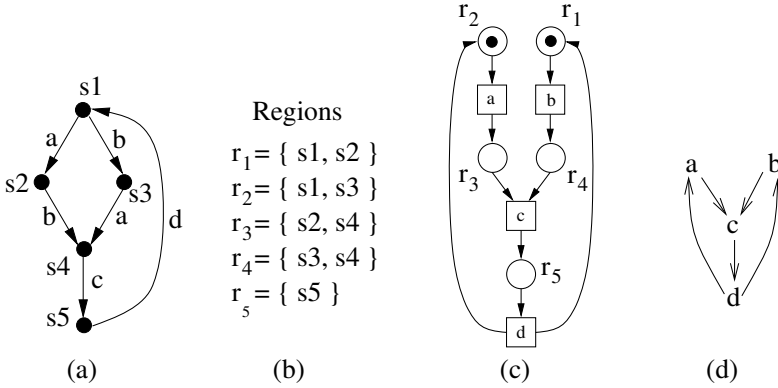
Fig. 1. (a) Transition system, (b) regions, (c) N_{TS}, (d) Causal dependency graph

---

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.,

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


Fig. 2. Algorithm for Petri net synthesis from [11]

Each TS has two *trivial regions*: the set of all states, S, and the empty set. 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.

The algorithm given by [11] to synthesize a PN, $N_{\mathsf{TS}} = (R, E, F_R, R_{s_{in}})$, from an *elementary transition system*[2] 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

---

[2] 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.

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
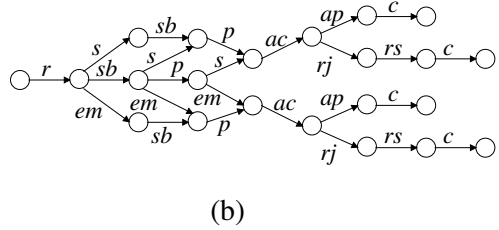
(a)   (b)

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

elementary, running algorithm of Figure 2 on the set of non-trivial regions R_{TS} derives a PN such that $L(\mathsf{PN}) = L(\mathsf{TS})$ [11].

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)[3]. 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 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 4 (Causal Dependency Graph). *Given a* 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 directed 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] Excitation and switching regions are not regions in the terms of Definition 3. The terms are used due to historical reasons.

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]. Formal proofs of the main results of this section can be found in [4].
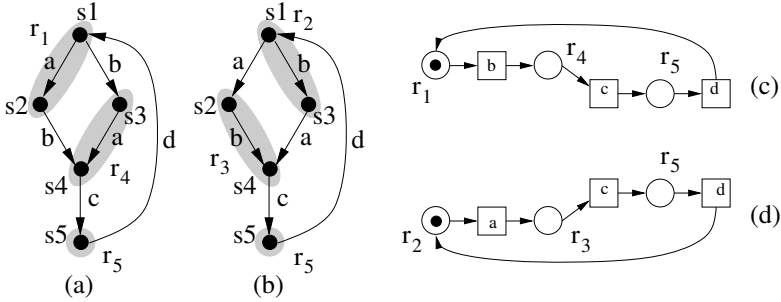

Fig. 4. Example of conservative components decomposition for the example of Figure 1: (a) Partition of the transition system on regions r_1, r_4 and r_5, and (b) for regions r_2, r_3 and r_5. The corresponding state machines are drawn in (c) and (d), respectively.

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 (b)). 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(c) and (d) show the two Petri nets corresponding to each partition, respectively.

3.1   State Machines and Its State-Based Representation

First we define formally the concepts of subnet and state machine component:

Definition 5 (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 $F' = F \cap ((P' \times T') \cup (T' \times P'))$.*

Definition 6 (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 $\sum_{p \in P'} M'_0(p) = 1$.

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 $r_i \longleftarrow \text{PickOneRegion}(\{r | r \in \,^{\circ}ev\})$
5 $r_j \longleftarrow \text{PickOneRegion}(\{r | r \in ev^{\circ} \wedge r \cap r_i = \emptyset\})$
6 $PendingRegs \longleftarrow \{r_i, r_j\}$
7 $Part \longleftarrow \{r_i, r_j\}$
8   **repeat**
9 $r \longleftarrow \text{RemoveOneRegion}(PendingRegs)$
10   **forall** $e \in E - Evs : e \in \,^{\circ}r \cup r^{\circ}$ **do**
11 $r_i \longleftarrow \text{PickOneRegion}(\{r | r \in \,^{\circ}e \wedge r \cap Part = \emptyset\})$
12 $r_j \longleftarrow \text{PickOneRegion}(\{r | r \in e^{\circ} \wedge r \cap Part = \emptyset\})$
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 following theorem states the main result of this section:

Theorem 1. *Let* 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}.

3.2 Allocation-Based SMC Computation

In Hack's thesis [13], the idea of *allocation* was introduced to decompose a Free-choice Petri net into a set of safe and conservative components (S-components). 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

---

Algorithm 2. SMCDecomposition

   Input: Transition system $\mathsf{TS} = (S, E, A, s_{in})$
   Output: $SMC_1 = (R_1, E_1, F_1, M_{0,1}) \ldots SMC_n = (R_n, E_n, F_n, M_{0,n})$
1 begin
2 $X \longleftarrow E$
3 $i \longleftarrow 1$
4   **repeat**
5 $ev \longleftarrow$ RemoveOneEvent(X)
6 $R_i \longleftarrow$ SMCComputation(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

---

regions can be found (lines 8-20). 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 that covers one of the events still not covered by any SMC_j, for $j < i$. When an event ev can only be covered by the trivial region S, then $E_i = \{ev\}$, and therefore the SMC_i 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 PNs is a PN where every transition with the same label in two or more components represents a synchronization point for the components [22]. 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* $L(\mathsf{TS}) \subseteq L(SMC_1 \parallel \ldots \parallel SMC_n)$.

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 in the complexity of the method, measured

in the number of regions needed. In the future, more elaborated strategies can be build on top of the approach presented to address these concerns[4].

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.

The theory presented in this section has been generalized to arbitrary k-bounded PNs, as it is shown in [4] (Section 3.4). Due to the lack of space, we only show the experiments on this extension in Section 5.

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

To face the complexity required for dealing with large TSs, an approach is presented to project the TS into tightly related events, obtaining smaller TSs. These smaller TSs can then be handled by computationally expensive Petri net mining methods. In this section we show how the decomposition approach of Section 3 can be applied on the TSs obtained by the projection technique to derive a set of Petri nets. These nets can be combined to form a unique Petri net that covers the traces of the initial log. Formal proofs of the main results can be found in [4].

4.1   Introductory Example

Let us illustrate the idea with the 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 depict 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 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). 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).

---

[4] Notice that the parallel composition might derive a general PN , i.e. no restriction on the class of PNs after composition is assumed in this paper.

Fig. 5. Transition system

(a)   (b)   (c)   (d)

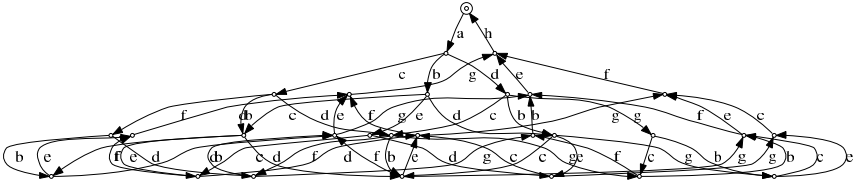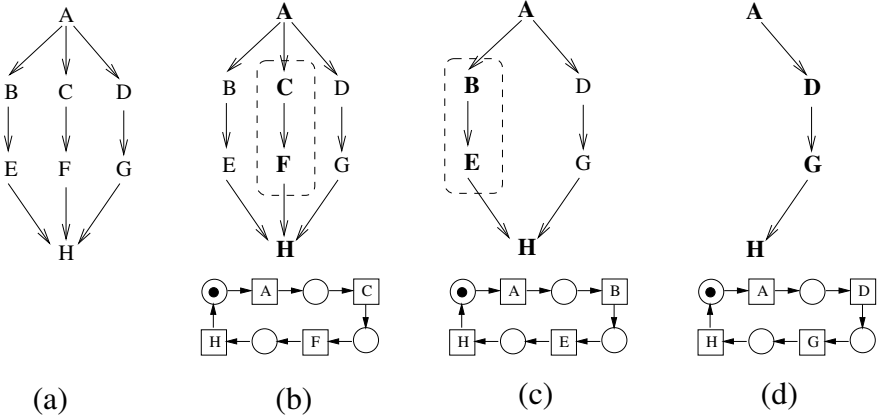**Fig. 6.** (a) Causal dependency graph, (b)-(d) (Top) Consecutive cuts of the causal dependency graph, (Bottom) State machines covering each cut

4.2 Causal Dependency Graph Partitioning

There exist several techniques for the partitioning of a graph into a set of clusters [14, 12]. 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 metric. Given a partition $E_1 \ldots E_n$ of the set E, the metric is defined as:

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

where $\overline{E_i}$ denotes the complement of set E_i, and $cut(A, B) = |\{(i,j)|(i,j) \in M \land i \in A, j \in B\}|$. If only two sets (i.e. a bipartition) are used in the previous formula, the following optimization problem can be considered:

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

Algorithm 3. DivideAndConquerMining

    Input: $\mathsf{TS} = (S, E, A, s_{in})$, MaxSize
    Output: Set of SMCs
                $SMC_1 = (R_1, E_1, F_1, M_{0,1}) \ldots SMC_n = (R_n, E_n, F_n, M_{0,n})$

1 begin
2   Compute $\longrightarrow_{\mathsf{TS}}$, $\parallel_{\mathsf{TS}}$ event relations
3 $(E_1 \ldots E_n) \longleftarrow \text{GraphPartition}(\text{CDG}(\mathsf{TS}), \text{MaxSize})$
4   **forall** E_i **do**
5 $E_i \longleftarrow \text{AddBorderEvents}(\mathsf{TS}, E_i, E)$
6 $\text{SMCDecomposition}(\mathsf{TS} \mid_{E_i})$
7   **end**
8 end

---

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 [8].

More concretely, if $f \in \mathbb{R}^{|E|}$ is the Fiedler vector, then a bipartition (E_1, E_2) can be obtained as follows: $e \in E_1$ if $f_e \geq 0$, and $e \in E_2$ otherwise.

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 4) 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 theorem provides the main result of this section (see [4] for the formal proof):

Theorem 3. *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* $L(\mathsf{TS}) \subseteq L(SMC_1 \parallel \ldots \parallel SMC_n)$.

5   Experiments

The theory described in Sections 3 and 4 has been incorporated into the tool
Genet [6,5]. The first experiments were conducted to test the ability to rediscover
conservative components from well-structured descriptions, i.e. to apply Algo-
rithms 1 and 2, and its corresponding generalizations (as described in [4]). To this
end, the TS of some k-bounded Petri nets was used (see Figures 7(a)-(c)). Table 1
reports the first experiment: comparing mining (-pm) versus conservative com-
ponents 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 [5] 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 with-
out the inclusion of extra behavior in the mined Petri net.

The second experiment was to have some confidence on the quality of the ap-
proach 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, 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
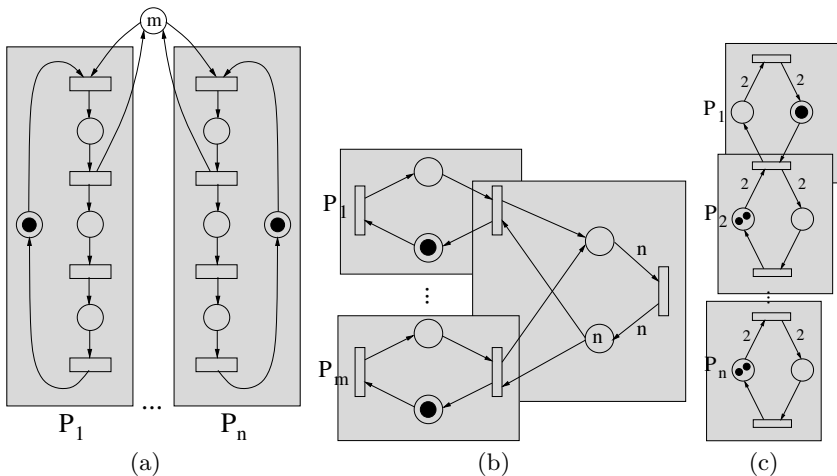

Fig. 7. Parameterized benchmarks: (a) n processes competing for m shared resources,
(b) m producers and n consumers, (c) a 2-bounded pipeline of n processes. Each box
represents a conservative component found.

the right reports the fitness of some miners
within ProM, and the fitness of the net cor-
responding the parallel composition of the
SMCs computed by Algorithm 2. The three
logs used are the illustrative logs described
in [17]. In summary, numbers in the table

Log	α	α++	DWS	Heuristic	Genet-cc
L1	0.83	0.80	0.84	0.84	0.85
L2	0.84	0.81	0.84	0.85	0.86
L3	0.63	0.55	0.62	0.62	0.58

are promising for our approach, and we believe they can improve if techniques
like the ones presented in Section 3.3 are additionally applied.

Table 1. Synthesis versus derivation of conservative components

benchmark	$|S|$	$|E|$	Genet-pm			Genet-cc		
			$|P|$	$|T|$	CPU	$|CC|$	$|P|$	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

The third experiment was to test the divide-and-conquer mining approach de-
scribed in Section 4 (-rec). We have used two types of examples: logs from [1], and
a real-life system modelling a complex module that controls the operation of opti-
cal lithography process for mass chip production [16]. Both types of benchmarks
are difficult to mine using the region-based mining approach described in [5]. Ta-
ble 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 compo-
nent 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. The conclusion from Table 2 is the
superiority to handle large systems for the divide-and-conquer approach when
compared to the classical region-based mining.

Table 2. Mining versus divide-and-conquer mining

				Genet-pm safe			Genet-pm 2-bounded			Genet-rec k-bounded				
benchmark	$|S|$	$|A|$	$|E|$	$|P|$	$|[S]|$	CPU	$|P|$	$|[S]|$	CPU	$|Bis|$	k	$|CC|$	$|P|$	$|T_U|$	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 3. Comparison for large logs from [21]

						Parikh		DWS		Genet-rec	
benchmark	# cases	# events	$|T|$	$|S|$	$|A|$	a'_B	CPU	a'_B	CPU	a'_B	CPU
a22f0n00_5	900	16952	22	676	1469	0.949	37s	0.935	4s	0.979	1s
t32f0n00_1	200	16358	32	1590	2339	0.992	7m 47s	0.863	10s	0.858	6s
a32f0n00_5	900	23195	32	2517	5907	0.933	3m	0.935	6s	1.000	9s
a42f0n00_5	900	26169	42	11170	21528	0.715	59m 29s	0.889	10s	0.962	1m 33s

Finally, we compared the divide-and-conquer technique presented in this paper with the *DWS miner*, also a clustering method presented in [9] (see Section 6 for a qualitative comparison). To this end, we used some of the largest logs that were used in [21] for a numerical analysis of the Parikh miner. We also report some other conformance measure, the *advanced behavioral appropriateness*, that gives an estimation of the degree of accuracy in which the model describes the log [17]. This measure ranges accuracy from 0 (low) to 1 (high). The results are provided in Table 3, were columns report the benchmark, number of cases, number of events, number of different tasks in the log, size of the corresponding TS[5], and for each approach we report the conformance estimation (a'_B) and the cpu time . For the benchmarks considered, one can see that the approach of this paper has similar complexity and appropriateness than the *DWS miner*.

6   Related Work

Together with the approach presented in [9], to the best of our knowledge there is no other approach for Petri net mining like the one presented in this paper. The differences are:

---

[5] Although benchmarks in Table 3 and 2 are produced from the same family of logs (being the benchmarks in Table 3 considerably larger), the settings of the FSM miner [18] used to create in each table the TS were different.

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. This sequential views can be good for visualization.
2. In [9] 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 [9] 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.

The divide-and-conquer technique presented in this paper can be used in combination with the region-based approaches for Petri net mining [5,21,3] to improve their applicability in two dimensions: firstly, to allow their application for large logs, and second, to avoid the problem of *overfitting*: in our experiments the resulting model (after the parallel composition) is often more general than the one obtained from a single application of the mining approach. The technique presented in this paper is suitable when the log contains a significant amount of different tasks, thus allowing the partition phase to be applied extensively.

7   Conclusions

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. Both approaches have been presented in combination with theoretical results that guarantee a covering of the initial log with respect to the parallel composition of the obtained nets.

Acknowledgements

References

1. Process mining, http://www.processmining.org
2. Badouel, E., Bernardinello, L., Darondeau, P.: Polynomial algorithms for the synthesis of bounded nets. In: Mosses, P.D., Schwartzbach, M.I., Nielsen, M. (eds.) CAAP 1995, FASE 1995, and TAPSOFT 1995. LNCS, vol. 915, pp. 364–383. Springer, Heidelberg (1995)
3. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: Proc. 5th Int. Conf. on Business Process Management, September 2007, pp. 375–383 (2007)
4. Carmona, J., Cortadella, J., Kishinevsky, M.: Divide-and-conquer strategies for process mining. Technical Report LSI-08-35-R, Software Department, Universitat Politécnica de Catalunya (2008)
5. Carmona, J., Cortadella, J., Kishinevsky, M.: A region-based algorithm for discovering Petri nets from event logs. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 358–373. Springer, Heidelberg (2008)
6. Carmona, J., Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: 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)
7. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving Petri nets from finite transition systems. IEEE Transactions on Computers 47(8), 859–882 (1998)
8. Cvetković, D., Rowlinson, P., Simić, S.: Eigenspaces of Graphs. Cambridge University Press, Cambridge (1997)
9. de Medeiros, A.K.A., Guzzo, A., Greco, G., van der Aalst, W.M.P., Weijters, A.J.M.M.T., van Dongen, B.F., Saccà, D.: Process mining based on clustering: A quest for precision. In: ter Hofstede, A.H.M., Benatallah, B., Paik, H.-Y. (eds.) BPM Workshops 2007. LNCS, vol. 4928, pp. 17–29. Springer, Heidelberg (2008)
10. Desel, J., Reisig, W.: The synthesis problem of Petri nets. Acta Inf. 33(4), 297–315 (1996)
11. Ehrenfeucht, A., Rozenberg, G.: Partial (Set) 2-Structures. Part I, II. Acta Informatica 27, 315–368 (1990)
12. Fiduccia, C.M., Mattheyses, R.M.: A linear-time heuristic for improving network partitions. In: DAC 1982: Proceedings of the 19th conference on Design automation, Piscataway, NJ, USA, 1982, pp. 175–181. IEEE Computer Society Press, Los Alamitos (1982)
13. Hack, M.: Analysis of production schemata by Petri nets. M.s. thesis, MIT (Feburary 1972)
14. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. The Bell system technical journal 49(1), 291–307 (1970)
15. Murata, T.: Petri Nets: Properties, analysis and applications. Proceedings of the IEEE, 541–580 (April 1989)
16. Pretorius, A.J.: Visualization of State Transition Graphs. PhD thesis, Technical University of Eindhoven (2008)
17. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. 33(1), 64–95 (2008)
18. van der Aalst, W.M.P., Rubin, V., Verbeek, H., van Dongen, B., Kindler, E., Günther, C.: Process mining: A two-step approach to balance between underfitting and overfitting. Technical Report BPM-08-01, BPM Center (2008)

19. van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., Mans, R.S., de Medeiros, A.K.A., Rozinat, A., Rubin, V., Song, M., Verbeek, H.M.W(E.), Weijters, A.J.M.M.T.: ProM 4.0: Comprehensive support for *real* process analysis. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 484–494. Springer, Heidelberg (2007)
20. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. 16(9), 1128–1142 (2004)
21. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. In: van Hee, K.M., Valk, R. (eds.) PETRI NETS 2008. LNCS, vol. 5062, pp. 368–387. Springer, Heidelberg (2008)
22. Vogler, W.: Modular Construction and Partial Order Semantics of Petri Nets. LNCS, vol. 625. Springer, Heidelberg (1992)