Process Discovery Using Localized Events*

Wil M.P. van der Aalst[1,2], Anna Kalenkova[2], Vladimir Rubin[2], and Eric Verbeek[1]

[1] Eindhoven University of Technology, P.O. Box 513, 5600 MB, Eindhoven, The Netherlands.
w.m.p.v.d.aalst@tue.nl,h.m.w.verbeek@tue.nl
[2] National Research University Higher School of Economics, Moscow, 101000, Russia,
akalenkova@hse.ru

Abstract. Process mining techniques aim to analyze and improve conformance and performance of processes using event data. Process discovery is the most prominent process-mining task: A process model is derived based on an event log. The process model should be able to capture causalities, choices, concurrency, and loops. Process discovery is very challenging because of trade-offs between fitness, simplicity, precision, and generalization. Note that event logs typically only hold example behavior and cannot be assumed to be complete (to avoid overfitting). Dozens of process discovery techniques have been proposed. These use a wide range of approaches, e.g., language- or state-based regions, genetic mining, heuristics, expectation maximization, iterative log-splitting, etc. When models or logs become too large for analysis, the event log may be automatically decomposed or traces may be clustered before discovery. Clustering and decomposition are done automatically, i.e., no additional information is used. This paper proposes a different approach where a *localized event log* is assumed. Events are localized by assigning a non-empty set of *regions* to each event. It is assumed that regions can only interact through shared events. Consider for example the mining of software systems. The events recorded typically explicitly refer to parts of the system (components, services, etc.). Currently, such information is ignored during discovery. However, references to system parts may be used to localize events. Also in other application domains, it is possible to localize events, e.g., communication events in an organization may refer to multiple departments (that may be seen as regions). This paper proposes a generic process discovery approach based on localized event logs. The approach has been implemented in *ProM* and experimental results show that location information indeed helps to improve the quality of the discovered models.

1 Introduction

Today's systems record all kinds of events, e.g., social interaction, financial transactions, user-interface activities, and the use of (mobile) devices. As more and more *event data* become available, the practical relevance of *process mining* further increases. Process mining techniques aim to discover, monitor and improve real processes by extracting knowledge from event logs [1]. The three most prominent process-mining tasks are: (i) *process discovery*: learning a process model from example behavior recorded in

* This work is supported by the Basic Research Program of the National Research University Higher School of Economics.

an event log, (ii) *conformance checking*: diagnosing and quantifying discrepancies between observed behavior and modeled behavior, and (iii) *performance analysis*: identifying bottlenecks, delays, and inefficiencies using the timestamps of events. Starting point for analysis is often an automatically discovered process model. In this paper, we focus on this first step, i.e., *learning a process model from event data*.

Input for process discovery is an *event log*. Each *event* in such a log refers to an *activity* (i.e., a well-defined step in some process) and is related to a particular *case* (i.e., a *process instance*). The events are partially ordered. Events related to a case describe one "run" of the process. Such a run is often referred to as a *trace*. It is important to note that an event log contains only example behavior.

Process discovery is challenging for a variety of reasons. Typically, only a fraction of the behavior possible can be observed and there is no explicit information on behaviors that are impossible, i.e., a sequence of activities that never occurred, may still happen in the future, but may also be impossible. Moreover, mixtures of choice, concurrency, and iteration may be difficult to uncover using merely an event log.

In this paper we propose to use "location information" present in most data sources. We assume that each event belongs to one or more *regions*. A region may be a software/hardware component, a service, a department, a team, or a geographic location. Regions can only interact through shared events just like communication involves multiple parties. We assume that events with non-overlapping sets of regions *cannot* influence each other directly. *This is comparable to the independence assumption often used in statistical analysis.*

Localized event logs combined with the independence assumption allow for a new decomposition approach. A *sublog* of the overall event log is created for every region. Then a *submodel* is created for each sublog. These submodels are merged into an *overall model*. Whereas traces at the global level are often unique showing only a fraction of the possible behavior, traces in the sublogs may have more repetitive behavior and easily cover all possible local behaviors. Therefore, location information may provide valuable information guiding decomposed discovery. This speeds up analysis and, most likely results in models better describing reality.

The idea to partition event logs is not new, see for example decomposition approaches [3, 4] and trace clustering approaches [16, 9, 27]. However, unlike existing approaches we do not try to partition cases or activities through mining. Instead, we propose to exploit location information explicitly attached to events. Such information is often available or derivable.

The approach has been implemented in *ProM* and experiments using synthetic and real-life event logs demonstrate the value of location information.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 introduces preliminaries, including process models. Process mining, in particular control-flow discovery, is introduced in Section 4. Localized event logs, i.e., logs were events have one or more associated regions, are presented in Section 5. Such logs may be used for decomposed process discovery, as shown in Section 6. The experiments presented in Section 7 (using synthetic data and data from two real-life software systems) show that localized event logs allow for significantly better models. Section 8 concludes the paper.

2 Related Work

For an introduction to process mining, we refer to [1].

Process discovery, i.e., discovering a process model from a multiset of example traces, is a very challenging problem and various discovery techniques have been proposed [5, 6, 7, 8, 10, 11, 13, 15, 18, 19, 21, 25, 28, 29]. Many of these techniques use Petri nets during the discovery process. It is impossible to provide a complete overview of all techniques here. Very different approaches are used, e.g., heuristics [13, 28], inductive logic programming [15], state-based regions [5, 11, 25], language-based regions [8, 29], and genetic algorithms [21]. Classical synthesis techniques based on regions [14] cannot be applied directly because the event log contains only example behavior. For state-based regions one first needs to create an automaton as described in [5]. Moreover, when constructing the regions, one should avoid overfitting. Language-based regions seem good candidates for discovering transition-bordered Petri nets for subnets [8, 29]. Recently, a family of inductive mining approaches has been proposed by Leemans et al. [18, 19]. These techniques can deal with incompleteness and infrequent behavior, but still provide formal guarantees (e.g., perfect fitness and rediscoverability for specific parameter settings). The approach presented in this paper can be used in conjunction with all existing process discovery approaches.

Also related is the work on decomposed process mining. In [2] two types of log decomposition are identified: *vertical decomposition* and *horizontal decomposition*. In a vertical partitioning complete cases are assigned to a group and end-to-end process models are discovered or checked. Traditional trace clustering techniques may be viewed as vertical decomposition techniques (not for scalability but for obtaining simpler models). Several authors have proposed such trace clustering techniques [16, 9, 27]. Here traces are grouped and simplified models are created per group. The approach in this paper is based on a horizontal decomposition (traces are split into subtraces) rather than a vertical decomposition. In a horizontal partitioning activities are assigned to (possibly overlapping) groups [2, 3, 4]. Cases are projected on subsets of activities, thus resulting in a sublog per group. A process fragment is discovered or checked per subgroup. The principles presented in [3, 4] are used to prove the correctness of the approach proposed in this paper.

Different divide and conquer approaches are possible [12, 3, 4]. For example, one may decompose event logs and process models based on the refined process structure tree identifying Single-Entry Single-Exit (SESE) fragments [24, 22]. This can only be done for conformance checking. Here, explicit location information is exploited to decompose *discovery* into relatively independent parts.

3 Process Models

The results presented in this paper do not depend on a particular representation. However, we use *labeled Petri nets* with designated *initial* and *final* markings to illustrate the approach. This section introduces the preliminaries needed in the remainder.

$\mathcal{B}(A)$ is the set of all multisets over some set A. For some multiset $b \in \mathcal{B}(A)$, $b(a)$ denotes the number of times element $a \in A$ appears in b. $b = [x^3, y^2, z]$ is a

multiset having 6 elements: three x elements (i.e., $b(x) = 3$), two y elements (i.e., $b(y) = 2$), and one z element (i.e., $b(z) = 1$). Operators are defined as usual, e.g. $[x^2, y] \uplus [x, y, z] = [x^3, y^2, z]$ is the union of two multisets.

$\sigma = \langle a_1, a_2, \ldots, a_n \rangle \in X^*$ denotes a sequence over X of length n. $\langle\,\rangle$ is the empty sequence and $\sigma_1 \cdot \sigma_2$ is the concatenation of two sequences. $\sigma{\restriction_Q}$ is the projection of σ on Q, e.g., $\langle a, b, c, a, b, c \rangle{\restriction_{\{a,c\}}} = \langle a, c, a, c \rangle$.

Definition 1 (Sequence Projection). *Let X be a set and $Q \subseteq X$ one of its subsets. ${\restriction_Q} \in X^* \rightarrow Q^*$ is a projection function and is defined recursively: (1) $\langle\,\rangle{\restriction_Q} = \langle\,\rangle$ and (2) for $\sigma \in X^*$ and $x \in X$: $(\langle x \rangle \cdot \sigma){\restriction_Q} = \sigma{\restriction_Q}$ if $x \notin Q$, and $(\langle x \rangle \cdot \sigma){\restriction_Q} = \langle x \rangle \cdot \sigma{\restriction_Q}$ if $x \in Q$.*

Definition 2 (Applying Functions to Sequences). *Let $f \in X \not\rightarrow Y$ be a partial function.*[3] *f may be applied to sequences of X using the following recursive definition (1) $f(\langle\,\rangle) = \langle\,\rangle$ and (2) for $\sigma \in X^*$ and $x \in X$:*

$$f(\langle x \rangle \cdot \sigma) = \begin{cases} f(\sigma) & \text{if } x \notin dom(f) \\ \langle f(x) \rangle \cdot f(\sigma) & \text{if } x \in dom(f) \end{cases}$$

Figure 1 shows a labeled Petri net composed of places $P = \{p1, p2, \ldots, p21\}$ and transitions $T = \{t1, t2, \ldots, t18\}$. The flow relation $F = \{(p1, t1), (t1, p2), (t1, p8), \ldots\}$ specifies the connections between places and transitions. A transition may have a label, e.g., transition $t1$ has label a. The label refers to the activity associated with the transition. Two transitions may have the same label, e.g., $t13$ and $t15$ correspond to the same activity. Note that transition $t4$ has no label, i.e., it does not correspond to a transition and is sometimes called "invisible".

Definition 3 (Labeled Petri Net). *A* labeled Petri net *is a tuple $N = (P, T, F, l)$ defining a finite set of places P, a finite set of transitions T (such that $P \cap T = \emptyset$), a flow relation $F \subseteq (P \times T) \cup (T \times P)$, and a labeling function $l \in T \not\rightarrow \mathcal{U}_A$ where $\mathcal{U}_A$ is some universe of activity names. A* marking *of N is a multiset of places M, i.e., $M \in \mathcal{B}(P)$.*

A labeled Petri net $N = (P, T, F, l)$ defines a directed graph with nodes $P \cup T$ and edges F. A transition $t \in dom(l)$ has a label $l(t)$ that refers to some activity. An invisible transition $t \in T \setminus dom(l)$ has no label and does not correspond to some observable activity. The state of a Petri net, called *marking*, is a multiset of places indicating how many *tokens* each place contains. The initial marking shown in Figure 1 is $[p1]$. Another marking of this Petri net is $[p3, p5, p15, p19]$.

A transition $t \in T$ is *enabled* in marking M of net N, denoted as $(N, M)[t\rangle$, if each of its input places $\bullet t$ contains at least one token. An enabled transition t may *fire*, i.e., one token is removed from each of the input places $\bullet t$ and one token is produced for each of the output places $t\bullet$. Transition $t1$ in Figure 1 is enabled in the initial marking. Firing $t1$ results in $[p2, p8, p16]$. In marking $[p3, p5, p15, p19]$ five transitions are enabled: $t3$, $t4$, $t5$, $t14$, $t17$. Firing $t4$ results in marking $[p4, p6, p15, p19]$.

[3] A partial function $f \in X \not\rightarrow Y$ has a domain $dom(f) \subseteq X$ and a range $rng(f) = \{f(x) \mid x \in dom(f)\} \subseteq Y$.

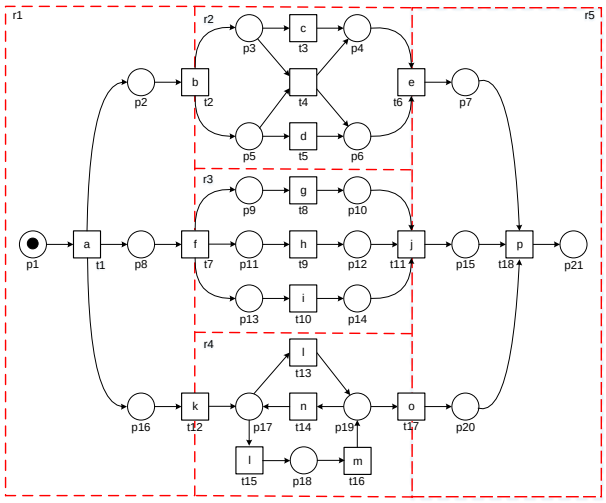

Fig. 1. Labeled Petri net with initial marking $[p1]$ and final marking $[p21]$. The dashed lines refer to regions and will be explained later.

$(N, M)[t\rangle(N, M')$ denotes that t is enabled in M and firing t results in marking M'. Let $\sigma = \langle t_1, t_2, \ldots, t_n \rangle \in T^*$ be a sequence of transitions. $(N, M)[\sigma\rangle(N, M')$ denotes that there is a set of markings $M_0, M_1, \ldots, M_n$ such that $M_0 = M$, $M_n = M'$, and $(N, M_i)[t_{i+1}\rangle(N, M_{i+1})$ for $0 \leq i < n$. A marking M' is *reachable* from M if there exists a sequence σ such that $(N, M)[\sigma\rangle(N, M')$.

In this paper we consider Petri nets with a designated initial and final markings. The behavior considered are all *complete* firing sequences from the initial marking M_{init} to the final marking M_{final}.

Definition 4 (System Net). *A system net is a triplet $SN = (N, M_{init}, M_{final})$ where $N = (P, T, F, l)$ is a labeled Petri net, $M_{init} \in \mathcal{B}(P)$ is the initial marking, and $M_{final} \in \mathcal{B}(P)$ is the final marking. $\mathcal{U}_{SN}$ is the* universe of system nets.

Given a system net SN, $\phi(SN)$ is the set of all possible *visible* traces, i.e., complete firing sequences starting in M_{init} and ending in M_{final} projected onto the set of observable activities using function l.

Definition 5 (Visible Traces). *Let $SN = (N, M_{init}, M_{final}) \in \mathcal{U}_{SN}$ be a system net with $N = (P, T, F, l)$. $\phi(SN) = \{l(\sigma) \mid (N, M_{init})[\sigma\rangle(N, M_{final})\}$ is the set of* visible *traces starting in M_{init} and ending in M_{final}.*[4]

Given a universe of activities $\mathcal{U}_A$, $\mathcal{U}_T = {\mathcal{U}_A}^*$ is the universe of visible traces. $\phi(SN) \subseteq \mathcal{U}_T$ defines the set of visible traces that can be generated by SN. Note that transitions may be invisible and that there may be multiple transitions having the same label. However, $\phi(SN)$ abstracts from such internals.

[4] Note that $l(\sigma)$ maps a firing sequence onto a trace of visible activities (see Definition 2).

In this paper, we use Petri nets to illustrate the approach. However, the results do not depend on the modeling language selected. Therefore, we define the more neutral notion of a process model. A system net SN defines a process model $PM = \phi(SN)$ if there is at least one firing sequence from the initial to the final marking.[5]

Definition 6 (Process Model). *A process model PM is a non-empty set of visible traces, i.e., $PM \subseteq \mathcal{U}_T$ and $PM \neq \emptyset$. $\mathcal{U}_{PM}$ is the universe of process models.*

In the remainder we use the following shorthand to refer to the activities appearing in a model: $\alpha(PM) = \{a \mid \exists_{\sigma \in PM}\ a \in \sigma\}$.

4 Process Mining

Starting point for any process mining technique is an event log with partially ordered events referring to cases and activities. To introduce events logs formally, we need to introduce some notations. Next to the universe of activities $\mathcal{U}_A$, the universe of visible traces $\mathcal{U}_T$, and the universe of process models $\mathcal{U}_{PM}$, we assume four additional universes:

- $\mathcal{U}_E$ is the set of all possible event identifiers,
- $\mathcal{U}_C$ is the set of all possible case identifiers,
- $\mathcal{U}_{Attr}$ is the set of all possible attribute names, and
- $\mathcal{U}_{Val}$ is the set of all possible attribute values.

Definition 7 (Event Log). $L = (E, C, act, case, attr, \prec)$ *is an* event log *if:*

- $E \subseteq \mathcal{U}_E$ *is a set of events,*
- $C \subseteq \mathcal{U}_C$ *is a set of cases,*
- $act \in E \rightarrow \mathcal{U}_A$ *maps events onto activities,*
- $case \in E \rightarrow C$ *maps events onto a set of cases,*
- $attr \in E \rightarrow (\mathcal{U}_{Attr} \not\rightarrow \mathcal{U}_{Val})$ *maps each event onto a partial function assigning values to some attributes, and*
- $\prec \subseteq E \times E$ *defines a partial order on events.*[6]

$\mathcal{U}_L$ is the set of all possible event logs.

Any $e \in E$ uniquely identifies an event. $act(e)$ is the activity executed for case $case(e)$. There may be cases without events, but every event refers to precisely one case. Event may have any number of attributes, e.g., $attr(e)(timestamp)$ = 2015-01-19T22:51:30.700+01:00 denotes the time event e occurred. Definition 7 assumes a partial order on events. In literature often a total order is assumed within a case, i.e., a case corresponds to a sequence of events. However, sometimes one is not sure about the ordering of events, e.g., multiple events have happened on the same day without

[5] Note that the labeled Petri net may deadlock or livelock before reaching M_{final}. Such traces are not considered because they cannot be related to cases in the event log. It is up to the discovery approach to ensure some notion of soundness.

[6] A partial order is a binary relation that is (1) irreflexive, i.e. $x \not\prec x$, (2) antisymmetric, i.e. $x \prec y$ implies $y \not\prec x$, and (3) transitive, i.e. if $x \prec y$ and $y \prec z$, then $x \prec z$.

an explicit order. Moreover, we may know the actual causal dependencies based on analyzing dataflow dependencies. In both cases, a partial order is more appropriate.

In the remainder we use the following shorthand to refer to the activities appearing in an event log: $\alpha(L) = \{act(e) \mid e \in E\}$.

Definition 8 (Process Discovery Technique). *A process discovery technique* $disc \in \mathcal{U}_L \rightarrow \mathcal{U}_{PM}$ *maps event logs onto process models such that for any* $L \in \mathcal{U}_L$*:* $\alpha(L) = \alpha(disc(L))$.

A process discovery technique produces a process model for an event log. Here we only require that the set of activities in the event log $\alpha(L)$ matches the set of activities in the model $\alpha(disc(L))$. As discussed in Section 2, many discovery techniques have been proposed in literature. These may be viewed as specific instances of $disc$.

Process discovery is challenging because event logs are often far from complete and there are at least four competing quality dimensions: (1) *fitness*, (2) *simplicity*, (3) *precision*, and (4) *generalization* [1]. A model with good *fitness* allows for most of the behavior seen in the event log. A model has a perfect fitness if all traces in the log can be replayed by the model from beginning to end. The *simplest* model that may explain the behavior seen in the log is the best model. This principle is known as Occam's Razor. Fitness and simplicity alone are not sufficient to judge the quality of a discovered process model. For example, it is very easy to construct an extremely simple Petri net that is able to replay all traces in an event log (but also any other event log referring to the same set of activities).[7] Similarly, it is undesirable to have a model that only allows for the exact behavior seen in the event log. Remember that the log contains only example behavior and that many traces that are possible may not have been observed yet. A model is *precise* if it does not allow for "too much" behavior. A model that is not precise is "underfitting", i.e., the model allows for behaviors very different from what was seen in the log. At the same time, the model should generalize and not restrict behavior to just the examples seen in the log. A model that does not *generalize* is "overfitting". Overfitting means that an overly specific model is generated whereas it is obvious that the log only holds example behavior (i.e., the model explains the particular sample log, but there is a high probability that the model is unable to explain the next batch of cases).

Here we do not quantify the four quality dimensions and restrict ourselves to simple fitness notions such as perfect fitness and the fraction of perfectly fitting cases.

Definition 9 (Fitness). *Let* $L = (E, C, act, case, attr, \prec) \in \mathcal{U}_L$ *be an event log and* $PM \in \mathcal{U}_{PM}$ *a process model.*

- *A case* $c \in C$ *is perfectly fitting* PM *(notation* $PM \rightsquigarrow c$*) if and only if there exists a trace* $\sigma = \langle a_1, a_2, \ldots, a_n \rangle \in PM$ *and a bijection* $f \in \{1, 2, \ldots n\} \rightarrow \{e \in E \mid case(e) = c\}$ *such that* $a_i = act(f(i))$ *for* $1 \leq i \leq n$ *and* $f(j) \not\prec f(i)$ *for any* $1 \leq i \leq j \leq n$*.*[8]

[7] System net $SN = ((P, T, F, l), M_{init}, M_{final})$ with $P = \emptyset$, $T = \alpha(L)$, $F = \emptyset$, l the identity function, $M_{init} = [\,]$, and $M_{final} = [\,]$ can replay any case in L.

[8] A function $f \in X \rightarrow Y$ is bijective if there is a one-to-one correspondence between the elements of X and Y, i.e., function f is total, surjective and injective.

- *$fit(L, PM) = \{c \in C \mid PM \rightsquigarrow c\}$ is the set of perfectly fitting cases.*
- *$nofit(L, PM) = C \setminus fit(L, PM)$ is the set of non-fitting cases,*
- *$fitness(L, PM) = \frac{|fit(L,PM)|}{|C|}$ is the fraction of traces in the event log perfectly fitting the model, and*
- *L is perfectly fitting PM if $nofit(L, PM) = \emptyset$.*

Note that we use interleaving semantics for process models while events are partially ordered (to capture uncertainty or causalities). Event log L is perfectly fitting model PM if for any observed case c there is model trace that could explain the set of events observed for c. When making a *trade-off* between fitness, simplicity, precision, and generalization, we may end up with a model not ensuring perfect fitness (e.g., deliberately leaving out exceptional behavior).

5 Localized Event Logs

As mentioned in the introduction, we assume *localized* event logs, i.e., each event e has a non-empty set of regions $loc(e)$. If event e occurs exclusively inside region r (i.e., no interaction between regions), then $loc(e) = \{r\}$. If event e describes some form of interaction between two regions $r1$ and $r2$, then $loc(e) = \{r1, r2\}$. Any form of interaction (from communicating humans to function calls and service invocations) involves multiple entities (e.g., components, services, or departments), here called *regions*.

Definition 10 (Localized Event Log). *A localized event log $L_L = (L, R, loc)$ is composed of an event log $L = (E, C, act, case, attr, \prec) \in \mathcal{U}_L$, a set of locations (called regions) R, and a location function $loc \in E \to \mathcal{P}_{NE}(R)$.*[9]

Given an event e, $loc(e)$ defines the set of regions involved. As mentioned before, regions can only interact through shared events.

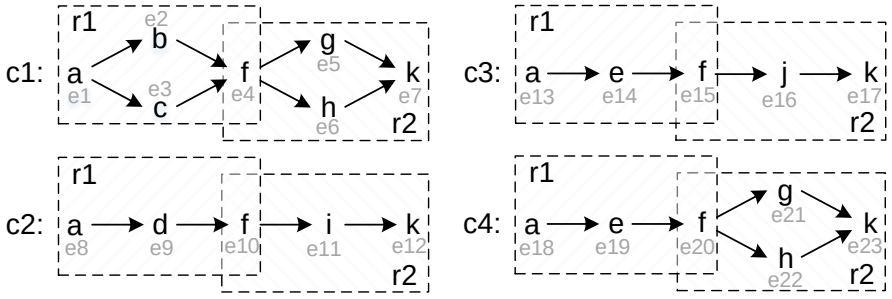

Fig. 2. Localized event log with 4 cases and 23 events.

Figure 2 visualizes a small event log with $E = \{e1, e2, \ldots, e23\}$ (23 events), $C = \{c1, c2, c3, c4\}$ (4 cases), and $R = \{r1, r2\}$ (2 regions). Functions act and $case$ are

[9] $\mathcal{P}_{NE}(X) = \{Y \subseteq X \mid Y \neq \emptyset\}$, i.e., all non-empty subsets of X.

also depicted in Figure 2: $act(e1) = a$, $case(e1) = c1$, $act(e2) = b$, $case(e2) = c1$, $act(e8) = a$, $case(e8) = c2$, etc. $\prec$ is only partially shown in Figure 2. Ordering relations of events in different cases are not depicted and only the transitive reduction of the ordering relations within a case is shown. Consider for example case $c1$. First activity a is executed (event $e1$) followed by both b (event $e2$) and c (event $e3$), then f (event $e4$) is executed followed by both g (event $e5$) and h (event $e6$). Case $c1$ concludes with the execution of activity k (event $e7$). We abstract from attributes here (i.e., $attr$ is not shown), e.g., each event e may have an associated timestamp $attr(e)(timestamp)$ and resource $attr(e)(resource)$. The location function loc is depicted using the shaded rectangles: $loc(e1) = \{r1\}$, $loc(e2) = \{r1\}$, $loc(e4) = \{r1, r2\}$, $loc(e5) = \{r2\}$, $loc(e20) = \{r1, r2\}$, $loc(e23) = \{r2\}$, etc. Note that all f events belong to both regions.

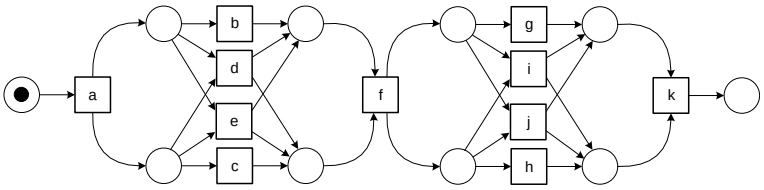

Fig. 3. Process model represented by a system net (the initial marking is shown; the final marking only marks the sink place).

Classical discovery approaches consider all events to be potentially related. *However, based on the regions involved we may conclude that events are unrelated thus significantly simplifying process discovery.* Consider again the localized event log of Figure 2. Based on the four cases, one could conclude that d is always followed by i and that j is always preceded by e. However, we have seen only four cases and the next case may reveal new behavior. Process discovery should be able to deal with incompleteness. For non-trivial processes, typically most traces are globally unique, i.e., there is no other case following exactly the same path from start to finish. If there are many unique traces, one cannot assume global completeness. However, we may assume events to be *unrelated* unless they are in the same region. Interaction between regions is possible only through shared events. Using this assumption, we could discover the process shown in Figure 3 using only the four cases of Figure 2. Without using such an assumption, we may end up with the process model shown in Figure 4. This model allows for the behavior exhibited by the four cases in Figure 2 and nothing more. In this overfitting model, e may be followed by g and h, or e may be followed by j, but e may not be followed by i. However, using the notion of regions in the localized event log, we know that the choice made in region $r1$ is unrelated to the choice made in region $r2$.

To illustrate the value of localized events consider the system net shown in Figure 5 (the final marking just marks place end). There are n concurrent parts each composed

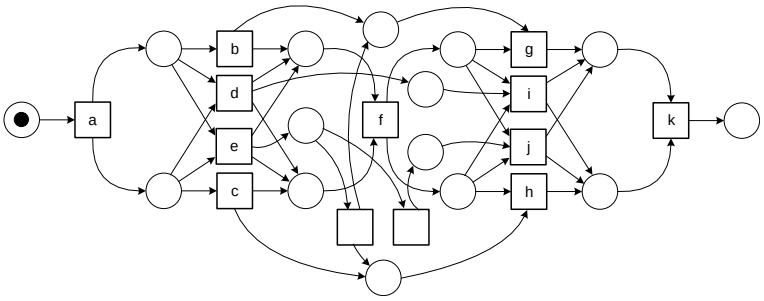

Fig. 4. Overfitting process model not taking into account the regions. Due to incompleteness, dependencies between $\{b, c, d, e\}$ and $\{g, h, i, j\}$ are derived that do not exist.

of k parallel activities. The model allows for:

$$pst_{all} = \frac{(n(k+2))!}{((k+2)!)^n}(k!)^n$$

possible (sequential) traces.[10] Note that we only consider sequential traces here. We may also consider the number of "directly follows" relations:

$$df_{all} = n + n(k+1)(k + (n-1)(k+2)) + n(1 + (n-1)(k+2))$$

where a directly follows relation is a pair of activities such that one activity is directly followed in a sequential trace.[11] The directly follows relation is interesting because it is used by many process discovery algorithms to uncover causal relationships.

Let us now consider one of the concurrent parts (say ri with $i \in \{1, \ldots, n\}$). The submodel allows for $pst_i = k!$ possible (sequential) traces of length $k + 4$ (including as, ais, aie and ae). The corresponding number of directly follows relations is $df_i = k^2 + k + 2$.[12]

Table 1 shows the effects of parameters n and k (there are n concurrent parts each composed of k parallel activities). If $n = 10$ and $k = 10$, then there are 4.17×10^{177}

[10] Each of the n concurrent parts allows for $k! = k \times (k-1) \times \ldots \times 1$ sequential traces of length $k + 2$ (abstracting from the fixed first activity as and the last activity ae which are invariable, but including ais and aie)). These n traces of length $k + 2$ can be interleaved in $\frac{(n(k+2))!}{((k+2)!)^n}$ ways and there are $(k!)^n$ unique collections of such n traces.

[11] Activity as can be directly followed by n activities ($a1s \ldots ans$). Each ais activity (with $i \in \{1, \ldots, n\}$) can be directly followed by $k + (n-1)(k+2)$ activities. Each aij activity (with $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, k\}$) can also be directly followed by $k + (n-1)(k+2)$ activities. Each aie activity (with $i \in \{1, \ldots, n\}$) can be directly followed by $1 + (n-1)(k+2)$ activities. Activity ae is never followed by another activity.

[12] Activity as can only be directly followed by ais in the submodel corresponding to ri. Activity ais can be directly followed by k activities. Each aij activity (with $j \in \{1, \ldots, k\}$) can be followed by k activities (aie and aij' with $j' \neq j$). Activity aie can only be directly followed by ae.

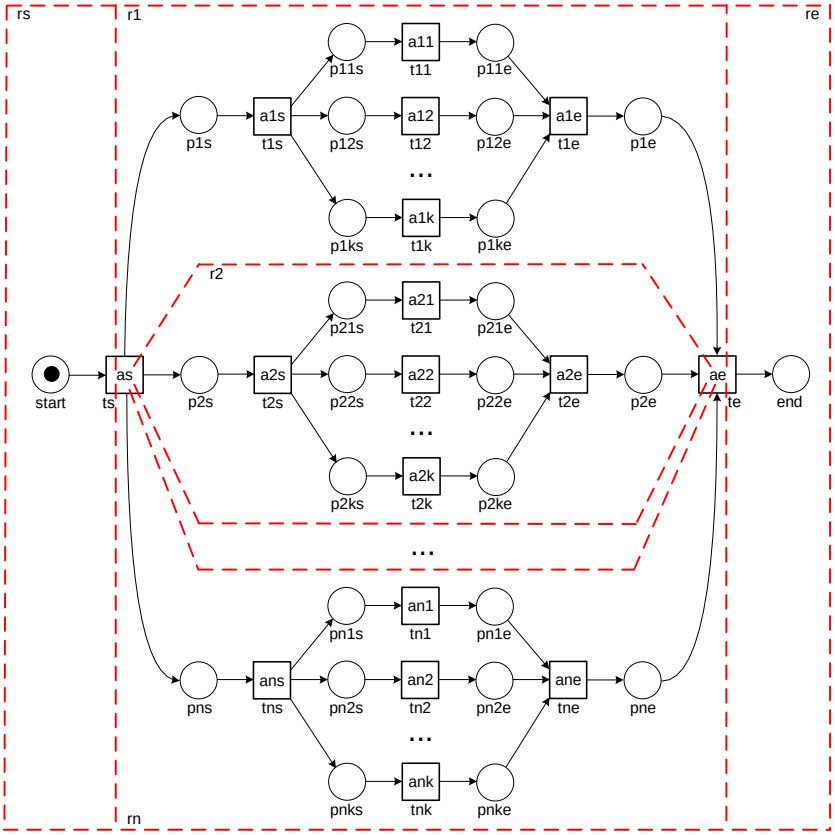

Fig. 5. A process composed of $n + 2$ subprocesses marked $rs, r1, r2, \ldots, rn, re$. Each of the n subprocesses in the middle has k parallel activities. For larger values of n and k this process is difficult to discover due to the many possible interleavings.

unique traces. Clearly, it is highly unlikely (understatement) to see all of these possibilities. Per concurrent part, there are 3628800 unique traces, still a lot but nevertheless a spectacular reduction (factor 1.15×10^{171}). Process discovery algorithms do not rely on seeing all possible traces to avoid overfitting. For example, if there are loops there may be infinitely many possible behaviors (see for example the lower part of Figure 1). Therefore, many discovery algorithms use notions such as the directly follows relation. If $n = 10$ and $k = 10$, then the directly follows relation has 14080 elements. This reduces to 1120 if it suffices to see only the local directly follows relationships, i.e., *less than* 8 *percent of the overall direct successions need to be observed to discover the "correct" model!*

Figure 5 is a rather extreme example. However, it nicely shows that the same model can be discovered using smaller, less complete event logs by exploiting localization

Table 1. Effects of n and k values in Figure 5 on the number of traces or direct successions that need to be observed for complete coverage.

parameters	n	1	1	5	5	1	10	10
	k	1	5	1	5	10	1	10
overall process	number of unique traces	1	120	1.68E+8	7.91E+31	3628800	4.39E+24	4.17E+177
	number of directly follows relationships	4	32	200	1140	112	850	14080
single fragment	number of unique traces	1	120	1	120	3628800	1	3628800
	number of directly follows relationships	4	32	4	32	112	4	112
combined fragments	minimal number of global traces needed to cover all locally unique traces	1	120	1	120	3628800	1	3628800
	total number of local directly follows relationships	4	32	20	160	112	40	1120

information in event logs. Compare this to statistics where assumptions about independence are used in predictions or when computing confidence intervals.

Definition 10 allows for two events that refer to the same activity but different regions. For process discovery, we would like to relate activities to a fixed number of regions. Hence, we aim at event logs that are *stable*.

Definition 11 (Stable). *A localized event log $L_L = (L, R, loc)$ with $L = (E, C, act, case, attr, \prec)$ is stable if for all $e_1, e_2 \in E$ with $act(e_1) = act(e_2)$: $loc(e_1) = loc(e_2)$.*

The localized event log of Figure 2 is stable, e.g., f events always refer to $r1$ and $r2$. A localized event log that is not stable can be "stabilized" by refining function $act \in E \to \mathcal{U}_A$. For example, function act can be replaced by act' where $act'(e) = (act(e), loc(e))$ for $e \in E$. The new function distinguishes activities having distinct sets of regions involved.

6 Decomposed Process Discovery

A localized event log can be transformed into a collection of sublogs, i.e., one event log per region. The sublogs are used to discover submodels. Finally, the submodels can be merged into a single overall process model. To create sublogs, we define a projection operator.

Definition 12 (Projection). *Let $L = (E, C, act, case, attr, \prec)$ be an event log and $X \subseteq E$ a subset of events. $L{\restriction_X} = (X, C, act{\restriction_X}, case{\restriction_X}, attr{\restriction_X}, \prec')$ with $\prec' = (\prec \cap (X \times X))$.*[13]

Definition 13 (Decomposed Discovery). *Let $L_L = (L, R, loc)$ be a localized event log with $L = (E, C, act, case, attr, \prec)$ and $A = \alpha(L)$, and let $disc \in \mathcal{U}_L \to \mathcal{U}_{PM}$ be a process discovery technique. For any region $r \in R$, we define the following shorthands:*

- *$E_r = \{e \in E \mid r \in loc(e)\}$ are the events of region r,*
- *$L_r = L{\restriction_{E_r}}$ is the sublog of region r,*

[13] $f{\restriction_X}$ is function f with the domain restricted to X, i.e., $dom(f{\restriction_X}) = X \cap dom(f)$.

– $A_r = \{act(e) \mid e \in E_r\}$ *are the activities of region* r*, and*
– $PM_r = disc(L_r)$ *is the process model discovered for region* r*.*

$PM_R = \{\sigma \in A^* \mid \forall_{r \in R}\, \sigma \restriction_{A_r} \in PM_r\}$ *is the overall process model constructed by merging the individual models.*

Note that the smaller process models are merged by weaving the region-based subsequences.

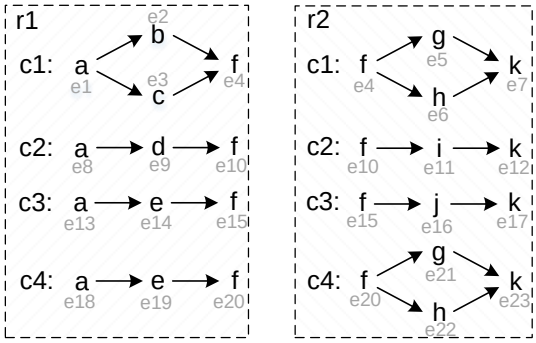

Fig. 6. Two projected event logs based on the overall event log of Figure 2: one sublog for each region.

Figure 6 illustrates how event logs can be projected onto the different regions. Now a model can be discovered for each region and the models can be merged as defined next.

PM_R merges the subprocesses discovered for the $|R|$ sublogs. Activity sequence σ is a visible trace of PM_R if and only if $\sigma \restriction_{A_r} \in PM_r$ (i.e., the projected sequence is a visible trace of the corresponding submodel) for each region $r \in R$. Like the rest of the paper, Definition 13 is not Petri net specific. However, the merging of the submodels into one overall model corresponds to the following union operator for system nets.

Definition 14 (Union of Nets). *Let* $SN^1 = (N^1, M^1_{init}, M^1_{final}) \in \mathcal{U}_{SN}$ *with* $N^1 = (P^1, T^1, F^1, l^1)$ *and* $SN^2 = (N^2, M^2_{init}, M^2_{final}) \in \mathcal{U}_{SN}$ *with* $N^2 = (P^2, T^2, F^2, l^2)$ *be two system nets with* $P^1 \cap P^2 = \emptyset$*.*

– $P^3 = P^1 \cup P^2$ *is the resulting set of places,*
– $A_S = rng(l^1) \cap rng(l^2)$ *is the set of shared activities (appearing in both regions),*
– $T^1_S = \{t \in dom(l^1) \mid l^1(t) \in A_S\}$ *and* $T^2_S = \{t \in dom(l^2) \mid l^2(t) \in A_S\}$ *are the transitions corresponding to shared activities,*
– $T^3 = \{(t_1, t_2) \in T^1_S \times T^2_S \mid l^1(t_1) = l^2(t_2)\} \cup \{(t_1, \gg) \mid t_1 \in T^1 \setminus T^1_S\} \cup \{(\gg, t_2) \mid t_2 \in T^2 \setminus T^2_S\}$ *is the resulting set of transitions,*[14]
– $dom(l^3) = \{(t_1, t_2) \in T^3 \mid t_1 \in dom(l^1) \vee t_2 \in dom(l^2)\}$*,* $l^3((t_1, t_2)) = l^1(t_1)$ *if* $t_1 \in dom(l^1)$ *and* $l^3((t_1, t_2)) = l^2(t_2)$ *if* $t_2 \in dom(l^2)$*,*
– $F^3 = \{(p, (t_1, x)) \in P^1 \times T^3 \mid (p, t_1) \in F^1\} \cup \{((t_1, x), p) \in T^3 \times P^1 \mid (t_1, p) \in F^1\} \cup \{(p, (x, t_2)) \in P^2 \times T^3 \mid (p, t_2) \in F^2\} \cup \{((x, t_2), p) \in T^3 \times P^2 \mid (t_2, p) \in F^2\}$*,*
– $N^1 \cup N^2 = (P^3, T^3, F^3, l^3)$ *is the union of* N^1 *and* N^2*, and*

[14] Next to synchronizing transitions of the form (t_1, t_2), there are transitions of the form $(t_1, \gg)$ or $(\gg, t_2)$ that do no synchronize as these are local to one of the nets.

– $SN^1 \cup SN^2 = (N^1 \cup N^2, M^1_{init} \uplus M^2_{init}, M^1_{final} \uplus M^2_{final})$ *is the union of system nets* SN^1 *and* SN^2.

The above definition takes the union of two system nets, but this can be extended to any number of system nets. The following lemma shows that such union based on merging transitions indeed implements the composition used in Definition 13.

Lemma 1. *Let* $SN^1, SN^2, \ldots, SN^n$ *be* n *system nets with non-overlapping sets of places.* $\phi(\bigcup_{1 \leq i \leq n} SN^i) = \{\sigma \in A^* \mid \forall_{1 \leq i \leq n}\ \sigma \restriction_{rng(l^i)} \in \phi(SN^i)\}$ *with* $A = \bigcup_{1 \leq i \leq n} rng(l^i)$ *as the set of activities.*

Proof. Assume $n = 2$, $SN^1 \cup SN^2 = (N^1 \cup N^2, M^1_{init} \uplus M^2_{init}, M^1_{final} \uplus M^2_{final})$, $N^1 = (P^1, T^1, F^1, l^1)$, $N^2 = (P^2, T^2, F^2, l^2)$, and $N^1 \cup N^2 = (P^3, T^3, F^3, l^3)$. The proof can be generalized for any number of system nets $n \geq 1$.

Let $\sigma \in \phi(SN^1 \cup SN^2)$, we need to show that $\sigma \restriction_{rng(l^1)} \in \phi(SN^1)$ and $\sigma \restriction_{rng(l^2)} \in \phi(SN^2)$. SN^1 can be seen as a projection of $SN^1 \cup SN^2$, i.e., places in P^2 are removed, places in P^1 are kept, transitions of the type $(\gg, t_2)$ are removed, and transitions of the type (t_1, t_2) or $(t_1, \gg)$ renamed to t_1. The firing sequence corresponding to σ in $SN^1 \cup SN^2$ corresponds to a firing sequence in SN^1 after renaming and removing transitions of the type $(\gg, t_2)$ from the sequence. This firing sequence is indeed possible because removing places from P^2 can never lead to blocking transitions. Hence, $\sigma \restriction_{rng(l^1)} \in \phi(SN^1)$. Similarly: $\sigma \restriction_{rng(l^2)} \in \phi(SN^2)$.

Let $\sigma \in A^*$ be such that $\sigma \restriction_{rng(l^1)} \in \phi(SN^1)$ and $\sigma \restriction_{rng(l^2)} \in \phi(SN^2)$, we need to show that $\sigma \in \phi(SN^1 \cup SN^2)$. $\sigma \restriction_{rng(l^1)} \in \phi(SN^1)$ defines a full firing sequence $\sigma_1 \in (T^1)^*$ with $l^1(\sigma_1) = \sigma \restriction_{rng(l^1)}$, i.e., a sequence of transitions starting in M^1_{init} and ending in M^1_{final}. Similarly, $\sigma \restriction_{rng(l^2)} \in \phi(SN^2)$ defines a full firing sequence $\sigma_2 \in (T^2)^*$ with $l^2(\sigma_2) = \sigma \restriction_{rng(l^2)}$. Note that $l^1(\sigma_1) \restriction_{A_S} = l^2(\sigma_2) \restriction_{A_S} = \sigma \restriction_{A_S}$.

There exists a $\sigma_3 \in (T^3)^*$ such that $l^3(\sigma_3) = \sigma$, $f_1(\sigma_3) = \sigma_1$ and $f_2(\sigma_3) = \sigma_2$ with $dom(f_1) = \{(t_1, t_2) \in T^3 \mid t_1 \neq \gg\}$, $f_1(t_1, t_2) = t_1$, and $dom(f_2) = \{(t_1, t_2) \in T^3 \mid t_2 \neq \gg\}$, $f_2(t_1, t_2) = t_2$. Such a sequence exists because in σ both system nets agree on shared activities A_S and for any t_1 and t_2 with $l^1(t_1) = l^2(t_2) \in A_S$: $(t_1, t_2) \in T^3$ (i.e., all combinations have been included). Now, it is easy to see that σ_3 is indeed a firing sequence possible in $SN^1 \cup SN^2$: it starts in $M^1_{init} \uplus M^2_{init}$ and ends in $M^1_{final} \uplus M^2_{final}$. Since $l^3(\sigma_3) = \sigma$, $\sigma \in \phi(SN^1 \cup SN^2)$. □

The lemma is related to classical results on net composition [20]. Also see [3, 4] for other properties preserved by the union of two system nets in relation to an event log.

Theorem 1 (Decomposed Discovery). *Let* $L_L = (L, R, loc)$ *be a stable localized event log and let* $disc \in \mathcal{U}_L \rightarrow \mathcal{U}_{PM}$ *be a process discovery technique. Let* PM_R, PM_r, *and* L_r *be as defined in Definition 13.*

– $fit(L, PM_R) \leq \bigcap_{r \in R} fit(L_r, PM_r)$,
– $fitness(L, PM_R) \leq \frac{|\bigcap_{r \in R} fit(L_r, PM_r)|}{|C|}$,
– $fit(L, PM_R) = \bigcap_{r \in R} fit(L_r, PM_r)$ *if* $\prec$ *defines a strict total order,*[15]

[15] A strict order is a partial order that is also trichotomous (exactly one of $x \prec y$, $y \prec x$ or $x = y$ holds).

– $\mathit{fitness}(L, PM_R) = \frac{|\bigcap_{r \in R} \mathit{fit}(L_r, PM_r)|}{|C|}$ *if* $\prec$ *defines a strict total order.*

Proof. The second and fourth statement follow directly from the first and third statement respectively. To prove the first statement we need to show that for any $c \in \mathit{fit}(L, PM_R)$ and $r \in R$: $c \in \mathit{fit}(L_r, PM_r)$. Because $PM \rightsquigarrow c$ there is a trace $\sigma_R = \langle a_1, a_2, \ldots, a_n \rangle \in PM_R$ and a bijection $f \in \{1, 2, \ldots n\} \rightarrow \{e \in E \mid case(e) = c\}$ such that $a_i = act(f(i))$ for $1 \leq i \leq n$ and $f(j) \not\prec f(i)$ for any $1 \leq i \leq j \leq n$. Let $\sigma_r = \sigma_R{\restriction_{A_r}}$. Clearly, $\sigma_r \in PM_r$ due to the construction of PM_R (see Definition 13). c is not just an case in L but also a case in L_r (see Definition 12). Due to stability, the set of c events projected away matches the elements projected away in $\sigma_r = \sigma_R{\restriction_{A_r}}$. Hence, a smaller bijection can be created relating σ_r to the A_r events in c. Therefore, $c \in \mathit{fit}(L_r, PM_r)$.

The reverse does not necessarily hold if $\prec$ is just a partial order and not a total order. The partial order could be linearized differently in the region-based submodels. To prove the third statement we additionally need to show that for any $c \in L$ such that $c \in \mathit{fit}(L_r, PM_r)$ for all $r \in R$: $c \in \mathit{fit}(L, PM_R)$. Since $\prec$ is now a strict total order, there is one $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ describing the sequence of activities (not events) in case c. Let $\sigma_r = \sigma{\restriction_{A_r}}$. For all $r \in R$: $\sigma_r \in PM_r$ because $c \in \mathit{fit}(L_r, PM_r)$ and L_L is stable. Since $PM_R = \{\sigma \in A^* \mid \forall_{r \in R}\, \sigma{\restriction_{A_r}} \in PM_r\}$, we conclude that $\sigma \in PM_R$ and $c \in \mathit{fit}(L, PM_R)$. □

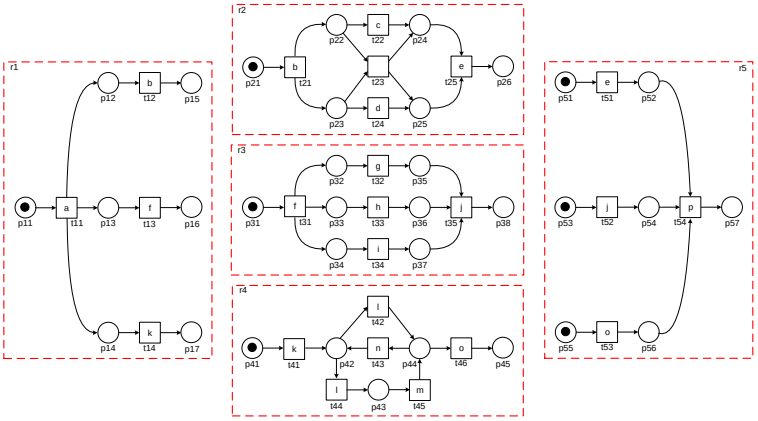

Fig. 7. Five discovered system nets: one for each region. The initial markings are indicated. The final markings are the states with all sink places marked with one token (not indicated explicitly).

Figure 7 shows the basic idea. Suppose that we take an event log created by simulating Figure 1 such that the event log is *locally* complete with respect to the directly follows relation. Now project the overall event log onto the five regions and discover a process model per region. In this case, discovery techniques may discover the five system nets shown in Figure 7. It is easy to see that these submodels indeed describe

the corresponding sublogs well. The five system nets in Figure 7 may be merged using Definition 14. In this case we do not get Figure 7 immediately. However, after removing some of the redundant places (i.e., hanging places whose removal does not change the behavior), we get the original system net (modulo renaming of places).

The composition of an overall model from submodels used in Definition 13 (and the specific Petri-net realization in Definition 14), assumes *synchronous* communication. *Asynchronous* communication can be supported by introducing special "channel regions", these are regions with a *send* and *receive* activity. This corresponds to the system net $SN_a = ((\{p_{buffer}\}, \{t_{send}, t_{receive}\}, \{(t_{send}, p_{buffer}), (p_{buffer}, t_{receive})\}, l), [\,], [\,])$ with $l(t_{send}) = a_{send}$ and $l(t_{receive}) = a_{receive}$. The corresponding process $PM_a = \phi(SN_a)$ is a simple buffer and may be viewed as a region. Hence, results like the property expressed in Theorem 1 can also be applied in the asynchronous setting.

7 Experimental Results

The decomposition discovery approach was implemented as a plugin for *ProM* (`www.processmining.org`) – an open source framework aimed to develop and test process mining algorithms. The plugin takes a localized event log as input (in localized event logs regions are specified as additional event attributes) and produces a system net as a result. This plugin was added to the package called *LocalizedLogs* available in the *Nightly Build* of *ProM*. The *DivideAndConquer* package [26] is used to handle the sublogs and to merge the resulting models.

7.1 Synthetic Event Data

Consider the reference model of a booking process depicted in Figure 8. Figure 9 shows an event log, generated by this model. This event log is not complete with respect to the directly follows relation, e.g., in the small event log the *select hotel* activity never directly followed the *register* activity.

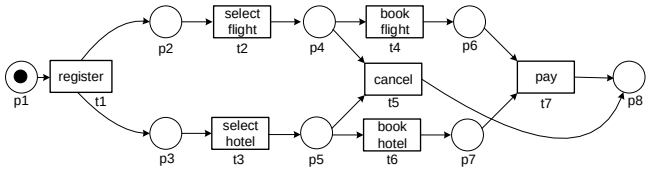

Fig. 8. A system net of a booking process with the initial and final markings $[p_1]$ and $[p_8]$ respectively.

All the known discovery methods, including those that deal with incomplete logs, will not rediscover the initial model, because they cannot exploit localization information and demand some form of global completeness. The inductive mining approach [19],

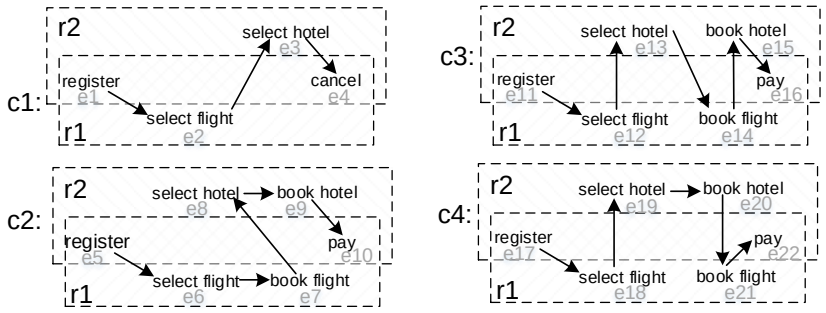

Fig. 9. A localized event log generated by the system net presented in Figure 8. There are two regions: one concerned with flights ($r1$) and one concerned with hotels ($r2$).

which is able to mine models from incomplete event logs, will discover the process model presented in Figure 10. The model is overfitting the event log with respect to the accidental ordering of two selection activities. Moreover, two loops are created. However, if we apply the approach proposed in this paper, we discover the initial system net (Figure 8) using the same discovery technique (after removing redundant hanging places, as described). This is possible because the event log in Figure 9 is complete per region.

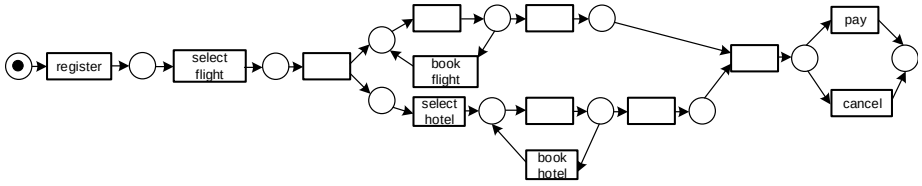

Fig. 10. The process model discovered by the inductive miner *without* exploiting localization information. Note that causalities between unrelated parts are inferred due to the incomplete event log.

7.2 Real-Life Event Data from Software

Using the approach proposed we have analyzed event logs of two real-life software systems: a *booking flight system* and a *banking system*.

The user of a booking flight system fills three different web forms to insert personal, insurance and payment information. The user may complete the web forms in any order. Thus, due to the event log complexity and incompleteness the direct application of the well-known discovery algorithms quickly results incomprehensible process models that contain misleading cycles and non-existing dependencies between activities. The

overall event log was enriched with three regions corresponding to the web forms, i.e., an attribute was added for this purpose. These regions naturally follow from the system design. Hence, it was easy to produce a localized event log. Shared activity labels correspond to common window operations, such as load and unload, and data verification. By applying the approach presented in this paper, we could obtain the model depicted in Figure 11.

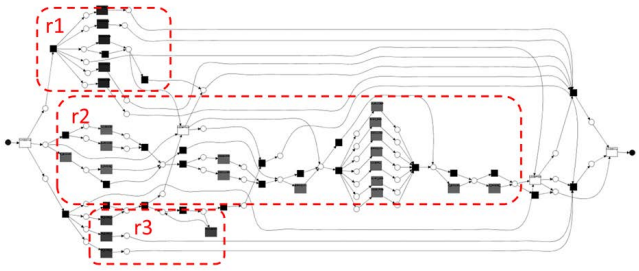

Fig. 11. A model of a booking flight system. Shared activities are highlighted in white, although it is not explicitly shown, they belong to all the regions.

The inductive mining approach was utilized as an underlying algorithm. The model obtained by directly applying the inductive miner contains 1809 connections between transitions, because of a global cycle, connecting almost all the transitions with each other, while the model constructed using regions contains only 177 connections.[16] Relations derived between different regions other than through overlapping activities are artifacts of the incompleteness of the event log.

The other software system under consideration is a banking system. This banking system handles requests and provides the user with the information about customer services. The banking system has a hierarchical structure and is represented by different program layers. Namely, it includes facade, services, data and common data access layers. Each request is received on the facade layer and then redirected to the next layer of the hierarchy. To treat layers as regions the event log was enriched with additional events, denoting request/response communications between layers and belonging to both communicating regions. The localized event log can be used to create the model. Again, the resulting model is simpler and our approach succeeds in handling incompleteness better than traditional approaches: the model contains 1986 connections between transitions instead of 19115, presented in the model obtained by applying the inductive miner directly on the event log. This multilayer model was represented as a model of interacting processes (or layers). A plugin for *ProM*, which constructs a BPMN [23] model of interacting processes from a set of system nets and a correspond-

[16] A pair (t_1, t_2) is a "connection" between visible transitions t_1 and t_2 (i.e., $t_1, t_2 \in dom(l)$) if and only if there exists a non-trivial path from t_1 to t_2, which does not go through other visible transitions.

ing event log, was developed as well. This plugin is based on the BPMN-supporting plugins, described in [17]. It converts each system net to a BPMN process within a pool, each request or response activity is converted to a message event, and each pair of corresponding message events is connected by a message flow. Note that for this plugin each shared event should have an additional attribute to determine its type (send or receive event). The automatically generated BPMN model of the multilayer banking system is presented in Figure 12.

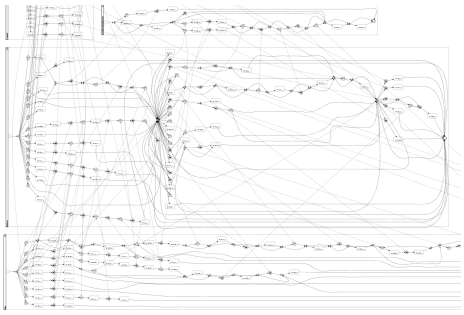

Fig. 12. A BPMN model discovered for a multilayer banking system.

Thus, the decomposition discovery approach allows not only to improve the quality of the models discovered, but also assists in creating hierarchical models exploiting higher-level process notations like BPMN.

For models constructed from the real-life event logs using various discovery approaches: heuristic [13, 28], inductive [18, 19], and ILP (language-based regions) [8, 29] miners, quality metrics, such as *fitness*, *precision* and *generalization* were obtained. Table 2 contains quality characteristics[17] of process models constructed directly from the event log, using the discovery approach specified, and the characteristics of corresponding process models constructed using localization information. Table 2 shows that the models constructed from the localized logs allow for more traces to fit and are more general, while the models constructed directly from the event logs tend to be more precise, but less fitting.

Table 2. Quality of process models discovered from the real-life event logs

Event logs	Discovery algorithms	Fitness	Trace fitness	Precision	Generalization
Booking system	Heuristic miner	0.00 / 0.13	0.64 / 0.75	0.55 / 0.32	0.89 / 0.90
	Inductive miner	0.23 / 1.00	0.85/ 1.00	0.22 / 0.16	0.98 / 1.00
	ILP miner	1.00 / 1.00	1.00/ 1.00	0.36 / 0.25	1.00 / 1.00
Banking system[17]	Inductive miner	0.25 / 1.00	0.84 / 1.00	0.14/ 0.06	0.97 / 1.00
	ILP miner	0.54 / 1.00	0.64 / 1.00	0.44 / 0.16	0.68 / 1.00

[17] *Fitness* is the fraction of perfectly fitting cases. *Trace fitness* is the measure of discrepancy between a log and a model. *Precision* is the fraction of additional cases, obtained during replay, which are not represented in the log. *Generalization* is the fraction of states visited during replay, which are covered by the model.

[17] Characteristics for the models constructed by the heuristic miner cannot be obtained in a reasonable amount of time.

8 Conclusion

In this paper we presented a novel process discovery approach exploiting localization information, i.e., events refer to one or more regions. Such information is available in most application domains. In this paper, we illustrated this using event data from software systems. Such systems have an explicit architecture and events may be related to this architecture. Hence, it is easy to create localized event logs. Experiments show that such reasonably chosen information can be used to produce much better process models. Whereas conventional approaches require some global form of completeness, our approach only needs local completeness (within a region). Therefore, the resulting models are simpler, more general and allow more cases to fit. Moreover, localization information may be exploited to create hierarchical models.

References

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

[2] W.M.P. van der Aalst. Distributed Process Discovery and Conformance Checking. In *International Conference on Fundamental Approaches to Software Engineering 2012*, volume 7212 of *Lecture Notes in Computer Science*, pages 1–25. Springer-Verlag, Berlin, 2012.

[3] W.M.P. van der Aalst. A General Divide and Conquer Approach for Process Mining. In *Federated Conference on Computer Science and Information Systems (FedCSIS 2013)*, pages 1–10. IEEE Computer Society, 2013.

[4] W.M.P. van der Aalst. Decomposing Petri Nets for Process Mining: A Generic Approach. *Distributed and Parallel Databases*, 31(4):471–507, 2013.

[5] W.M.P. van der Aalst, V. Rubin, H.M.W. Verbeek, B.F. van Dongen, E. Kindler, and C.W. Günther. Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting. *Software and Systems Modeling*, 9(1):87–111, 2010.

[6] W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

[7] R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, volume 1377 of *Lecture Notes in Computer Science*, pages 469–483. Springer-Verlag, Berlin, 1998.

[8] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process Mining Based on Regions of Languages. In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 375–383. Springer-Verlag, Berlin, 2007.

[9] R.P. J. C. Bose and W.M.P. van der Aalst. Trace Clustering Based on Conserved Patterns: Towards Achieving Better Process Models. In *BPM 2009 Workshops, Proceedings of the Fifth Workshop on Business Process Intelligence (BPI'09)*, volume 43 of *Lecture Notes in Business Information Processing*, pages 170–181. Springer-Verlag, Berlin, 2010.

[10] J. Carmona and J. Cortadella. Process Mining Meets Abstract Interpretation. In J.L. Balcazar, editor, *ECML/PKDD 210*, volume 6321 of *Lecture Notes in Artificial Intelligence*, pages 184–199. Springer-Verlag, Berlin, 2010.

[11] J. Carmona, J. Cortadella, and M. Kishinevsky. A Region-Based Algorithm for Discovering Petri Nets from Event Logs. In *Business Process Management 2008*, pages 358–373, 2008.

[12] J. Carmona, J. Cortadella, and M. Kishinevsky. Divide-and-Conquer Strategies for Process Mining. In *Business Process Management (BPM 2009)*, volume 5701 of *Lecture Notes in Computer Science*, pages 327–343. Springer-Verlag, Berlin, 2009.

[13] J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.

[14] P. Darondeau. Unbounded Petri Net Synthesis. In *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 413–438. Springer-Verlag, Berlin, 2004.

[15] S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens. Robust Process Discovery with Artificial Negative Events. *Journal of Machine Learning Research*, 10:1305–1340, 2009.

[16] G. Greco, A. Guzzo, L. Pontieri, and D. Saccà. Discovering Expressive Process Models by Clustering Log Traces. *IEEE Transaction on Knowledge and Data Engineering*, 18(8):1010–1027, 2006.

[17] A. Kalenkova, M. de Leoni, and W.M.P. van der Aalst. Discovering, Analyzing and Enhancing BPMN Models Using ProM. In *Business Process Management Demo Sessions (BPMD 2014)*, volume 1295 of *CEUR Workshop Proceedings*, pages 36–40, 2014.

[18] S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour. In *International Workshop on Business Process Intelligence (BPI 2014)*, volume 171 of *Lecture Notes in Business Information Processing*, pages 66–78. Springer-Verlag, Berlin, 2014.

[19] S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Discovering Block-structured Process Models from Incomplete Event Logs. In *Applications and Theory of Petri Nets 2014*, volume 8489 of *Lecture Notes in Computer Science*, pages 91–110. Springer-Verlag, Berlin, 2014.

[20] A. Mazurkiewicz. Semantics of Concurrent Systems: A Modular Fixed-Point Trace Approach. In G. Rozenberg, editor, *Advances in Petri Nets 1984*, volume 188 of *Lecture Notes in Computer Science*, pages 353–375. Springer-Verlag, Berlin, 1984.

[21] A.K. Alves de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Genetic Process Mining: An Experimental Evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007.

[22] J. Munoz-Gama, J. Carmona, and W.M.P. van der Aalst. Single-Entry Single-Exit Decomposed Conformance Checking. *Information Systems*, 46:102–122, December 2014.

[23] OMG. Business Process Model and Notation (BPMN). Object Management Group, formal/2011-01-03, 2011.

[24] A. Polyvyanyy, J. Vanhatalo, and H. Völzer. Simplified Computation and Generalization of the Refined Process Structure Tree. In *WS-FM 2010*, volume 6551 of *Lecture Notes in Computer Science*, pages 25–41. Springer-Verlag, Berlin, 2011.

[25] M. Sole and J. Carmona. Process Mining from a Basis of Regions. In *Applications and Theory of Petri Nets 2010*, volume 6128 of *Lecture Notes in Computer Science*, pages 226–245. Springer-Verlag, Berlin, 2010.

[26] E. Verbeek. Decomposed process mining with DivideAndConquer. In *Proceedings of the BPM Demo Sessions 2014*, pages 1–5, 2014.

[27] J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens. Leveraging Process Discovery With Trace Clustering and Text Mining for Intelligent Analysis of Incident Management Processes. In *IEEE Congress on Evolutionary Computation (CEC 2012)*, pages 1–8, 2012.

[28] A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.

[29] J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik. Process Discovery using Integer Linear Programming. *Fundamenta Informaticae*, 94:387–412, 2010.