

Process Mining in the Large: A Tutorial

Wil M.P. van der Aalst^{1,2,3}(✉)

¹ Department of Mathematics and Computer Science,
Eindhoven University of Technology, Eindhoven, The Netherlands

² Business Process Management Discipline, Queensland University of Technology,
Brisbane, Australia

³ International Laboratory of Process-Aware Information Systems,
National Research University Higher School of Economics, Moscow, Russia
`w.m.p.v.d.aalst@tue.nl`

Abstract. Recently, process mining emerged as a new scientific discipline on the interface between process models and event data. On the one hand, conventional Business Process Management (BPM) and Workflow Management (WfM) approaches and tools are mostly model-driven with little consideration for event data. On the other hand, Data Mining (DM), Business Intelligence (BI), and Machine Learning (ML) focus on data without considering end-to-end process models. Process mining aims to bridge the gap between BPM and WfM on the one hand and DM, BI, and ML on the other hand. Here, the challenge is to turn torrents of event data (“Big Data”) into valuable insights related to process performance and compliance. Fortunately, process mining results can be used to identify and understand bottlenecks, inefficiencies, deviations, and risks. This tutorial paper introduces basic process mining techniques that can be used for process discovery and conformance checking. Moreover, some very general decomposition results are discussed. These allow for the decomposition and distribution of process discovery and conformance checking problems, thus enabling process mining in the large.

Keywords: Process mining · Big Data · Process discovery · Conformance checking

1 Introduction

Like most IT-related phenomena, also the growth of event data complies with Moore’s Law. Similar to the number of transistors on chips, the capacity of hard disks, and the computing power of computers, the digital universe is growing exponentially and roughly doubling every 2 years [55,64]. Although this is not a new phenomenon, suddenly many organizations realize that increasing amounts of “Big Data” (in the broadest sense of the word) need to be used intelligently in order to compete with other organizations in terms of efficiency, speed and service. However, the goal is not to collect as much data as possible. The real challenge is to turn event data into valuable insights. Only *process mining* techniques directly relate event data to end-to-end business processes [2]. Existing

business process modeling approaches generating piles of process models are typically disconnected from the real processes and information systems. Data-oriented analysis techniques (e.g., data mining and machines learning) typically focus on simple classification, clustering, regression, or rule-learning problems (Fig. 1).

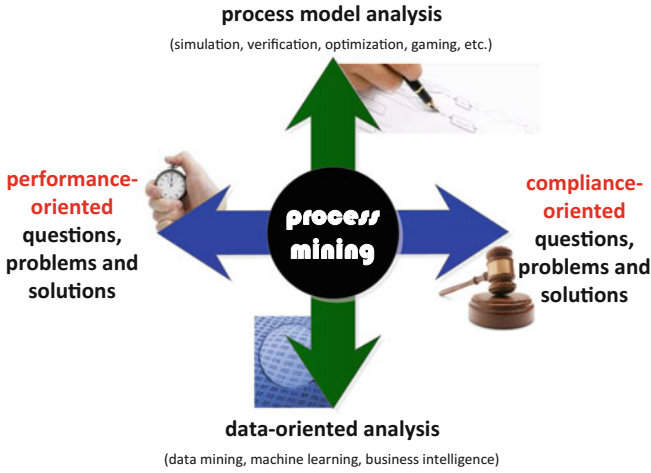


Fig. 1. Process mining provides the missing link between on the one hand process model analysis and data-oriented analysis and on the other hand performance and conformance.

Process mining aims to *discover, monitor and improve real processes by extracting knowledge from event logs* readily available in today’s information systems [2]. Starting point for any process mining task 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 belonging to a case are *ordered* and can be seen as one “run” of the process. The sequence of activities executed for a case is called a *trace*. Hence, an event log can be viewed as a multiset of *traces* (multiple cases may have the same trace). It is important to note that an event log contains only *example behavior*, i.e., we cannot assume that all possible traces have been observed. In fact, an event log often contains only a fraction of the possible behavior [2].

The growing interest in process mining is illustrated by the *Process Mining Manifesto* [57] recently released by the *IEEE Task Force on Process Mining*. This manifesto is supported by 53 organizations and 77 process mining experts contributed to it.

The process mining spectrum is quite broad and includes techniques like process discovery, conformance checking, model repair, role discovery, bottleneck analysis, predicting the remaining flow time, and recommending next steps. In this paper, we focus on the following two main process mining problems:

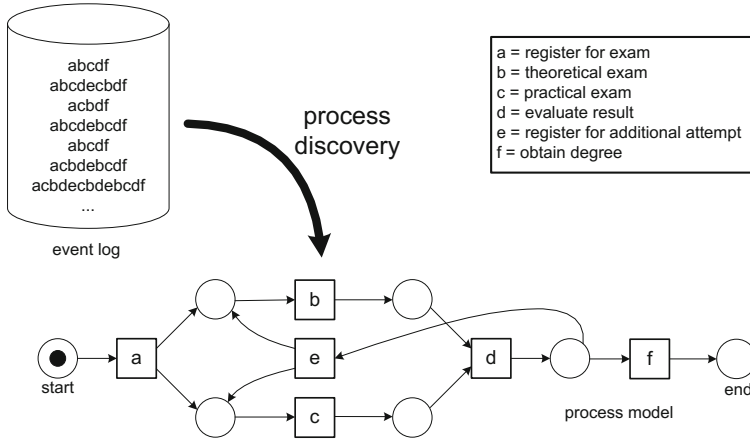


Fig. 2. Starting point for *process discovery* is an event log consisting of traces. In this example each trace describes activities related to an exam candidate. Based on the observed behavior a process model is inferred that is able to reproduce the event log. For example, both the traces in the event log and the runs of the process model always start with *a* (register for exam) and end with *f* (obtain degree). Moreover, *a* is always directly followed by *b* or *c*, *b* and *c* always happen together (in any order), *d* can only occur after both *b* and *c* have happened, *d* is always directly followed by *e* or *f*, etc. There are various process discovery techniques to automatically learn a process model from raw event data.

- *Process discovery problem*: Given an event log consisting of a collection of traces (i.e., sequences of events), construct a Petri net that “adequately” describes the observed behavior (see Fig. 2).¹
- *Conformance checking problem*: Given an event log and a Petri net, diagnose the differences between the observed behavior (i.e., traces in the event log) and the modeled behavior (i.e., firing sequences of the Petri net). Figure 3 shows examples of deviations discovered through conformance checking.

Both problems are formulated in terms of Petri nets. However, any other process notation can be used, e.g., BPMN models, BPEL specifications, UML activity diagrams, Statecharts, C-nets, and heuristic nets.

The incredible growth of event data is also posing new challenges [84]. As event logs grow, process mining techniques need to become more efficient and highly scalable. Dozens of process discovery [2, 19, 21, 26, 28, 32–34, 50, 52, 65, 85, 92, 93] and conformance checking [9, 22, 23, 25, 31, 35, 52, 68, 69, 79, 90] approaches have been proposed in literature. Despite the growing maturity of these approaches, the quality and efficiency of existing techniques leave much to be desired. State-of-the-art techniques still have problems dealing with large and/or

¹ As will be shown later, there are different ways of measuring the quality of a process discovery result. The term “adequately” is just an informal notion that will be detailed later.

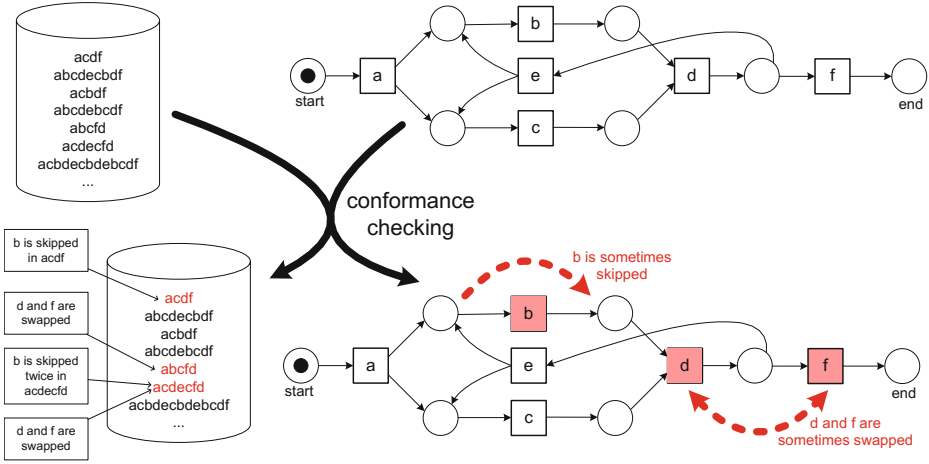


Fig. 3. *Conformance checking* starts with an event log and a process model. Ideally, events in the event log correspond to occurrences of activities in the model. By replaying the traces on the model one can find differences between log and model. The first trace shown cannot be replayed because activity *b* is missing (although no theoretical exam was made the result was evaluated). The fifth trace cannot be replayed because *d* and *f* are swapped, i.e., the candidate obtained a degree before the formal decision was made). The sixth trace has both problems. Conformance checking results can be diagnosed using a log-based view (bottom left) or a model-based view (bottom right).

complex event logs and process models. Consider for example Philips Healthcare, a provider of medical systems that are often connected to the Internet to enable logging, maintenance, and remote diagnostics. More than 1500 Cardio Vascular (CV) systems (i.e., X-ray machines) are remotely monitored by Philips. On average each CV system produces 15,000 events per day, resulting in 22.5 million events per day for just their CV systems. The events are stored for many years and have many attributes. The error logs of ASML’s lithography systems have similar characteristics and also contain about 15,000 events per machine per day. These numbers illustrate the fact that today’s organizations are already storing terabytes of event data. Process mining techniques aiming at very precise results (e.g., guarantees with respect to the accuracy of the model or diagnostics), quickly become intractable when dealing with such real-life event logs. Earlier applications of process mining in organizations such as Philips and ASML, show that there are various challenges with respect to performance (response times), capacity (storage space), and interpretation (discovered process models may be composed of thousands of activities) [29]. Therefore, we also describe the generic divide and conquer approach presented in [7]:

- For *conformance checking*, we decompose the process model into smaller partly overlapping model fragments. If the decomposition is done properly, then any trace that fits into the overall model also fits all of the smaller model fragments

and vice versa. Hence, metrics such as the fraction of fitting cases can be computed by only analyzing the smaller model fragments.

- To decompose *process discovery*, we split the set of activities into a collection of partly overlapping activity sets. For each activity set, we project the log onto the relevant events and discover a model fragment. The different fragments are glued together to create an overall process model. Again it is guaranteed that all traces in the event log that fit into the overall model also fit into the model fragments and vice versa.

This explains the title of this tutorial: “Process Mining in the Large”.

The remainder of this paper is organized as follows. Section 2 provides an overview of the process mining spectrum. Some basic notions are introduced in Sect. 3. Section 4 presents two process discovery algorithms: the α -algorithm (Sect. 4.1) and region-based process discovery (Sect. 4.2). Section 5 introduces two conformance checking techniques. Moreover, the different quality dimensions are discussed and the importance of *aligning* observed and modeled behavior is explained. Section 6 presents a very generic decomposition result showing that most process discovery and conformance checking can be split into many smaller problems. Section 7 concludes the paper.

2 Process Mining Spectrum

Figure 4 shows the *process mining framework* described in [2]. The top of the diagram shows an external “world” consisting of business processes, people, and organizations supported by some information system. The information system records information about this “world” in such a way that events logs can be extracted. The term *provenance* used in Fig. 4 emphasizes the systematic, reliable, and trustworthy recording of events. The term provenance originates from scientific computing, where it refers to the data that is needed to be able to reproduce an experiment [39, 61]. *Business process provenance* aims to systematically collect the information needed to reconstruct what has actually happened in a process or organization [37]. When organizations base their decisions on event data it is essential to make sure that these describe history well. Moreover, from an auditing point of view it is necessary to ensure that event logs cannot be tampered with. Business process provenance refers to the set of activities needed to ensure that history, as captured in event logs, “cannot be rewritten or obscured” such that it can serve as a reliable basis for process improvement and auditing.

As shown in Fig. 4, event data can be partitioned into “*pre mortem*” and “*post mortem*” event logs. “Post mortem” event data refer to information about cases that have completed, i.e., these data can be used for process improvement and auditing, but not for influencing the cases they refer to. “Pre mortem” event data refer to cases that have not yet completed. If a case is still running, i.e., the case is still “alive” (pre mortem), then it may be possible that information in the event log about this case (i.e., current data) can be exploited to ensure the correct or efficient handling of this case.

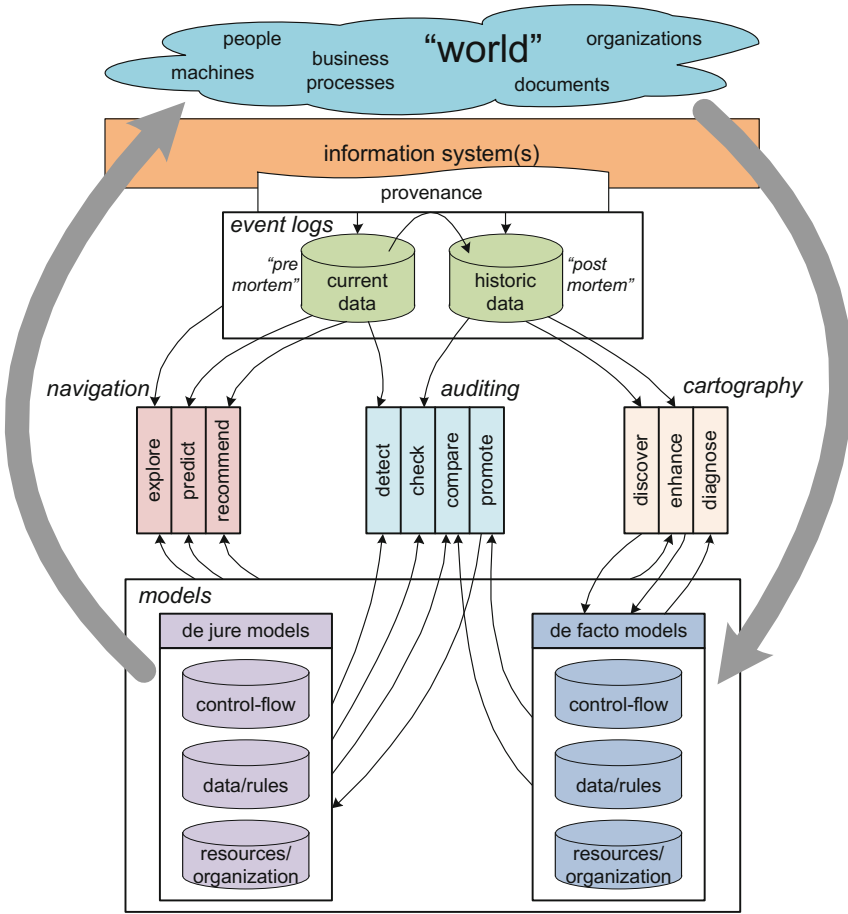


Fig. 4. Overview of the process mining spectrum [2].

“Post mortem” event data are most relevant for *off-line process mining*, e.g., discovering the control-flow of a process based on one year of event data. For *online process mining* mixtures of “pre mortem” (current) and “post mortem” (historic) data are needed. For example, historic information can be used to learn a predictive model. Subsequently, information about a running case is combined with the predictive model to provide an estimate for the remaining flow time of the case.

The process mining framework described in [2] also distinguishes between two types of models: “*de jure models*” and “*de facto models*”. A *de jure model* is *normative*, i.e., it specifies how things should be done or handled. For example, a process model used to configure a BPM system is normative and forces people to work in a particular way. A *de facto model* is *descriptive* and its goal is not to steer or control reality. Instead, de facto models aim to capture reality.

As shown in Fig. 4 both de jure and de facto models may cover multiple perspectives including the *control-flow perspective* (“How?”), the *organizational perspective* (“Who?”), and the *case perspective* (“What?”). The control-flow perspective describes the ordering of activities. The organizational perspective describes resources (worker, machines, customers, services, etc.) and organizational entities (roles, departments, positions, etc.). The case perspective describes data and rules.

In the middle of Fig. 4 ten process mining related activities are depicted. These ten activities are grouped into three categories: *cartography*, *auditing*, and *navigation*. The activities in the cartography category aim at making “process maps”. The activities in the auditing category all involve a de jure model that is confronted with reality in the form of event data or a de facto model. The activities in the navigation category aim at improving a process while it is running.

Activity *discover* in Fig. 4 aims to learn a process model from examples stored in some event log. Lion’s share of process mining research has been devoted to this activity [2, 47]. A discovery technique takes an event log and produces a model without using any additional a-priori information. An example is the α -algorithm [21] that takes an event log and produces a Petri net explaining the behavior recorded in the log. If the event log contains information about resources, one can also discover resource-related models, e.g., a social network showing how people work together in an organization.

Since the mid-nineties several groups have been working on techniques for process discovery [18, 21, 26, 34, 38, 44, 45, 92]. In [12] an overview is given of the early work in this domain. The idea to apply process mining in the context of workflow management systems was introduced in [26]. In parallel, Datta [38] looked at the discovery of business process models. Cook et al. investigated similar issues in the context of software engineering processes [34]. Herbst [54] was one of the first to tackle more complicated processes, e.g., processes containing duplicate tasks.

Most of the classical approaches have problems dealing with concurrency. The α -algorithm [21] is an example of a simple technique that takes concurrency as a starting point. However, this simple algorithm has problems dealing with complicated routing constructs and noise (like most of the other approaches described in literature). Process discovery is very challenging because techniques need to balance four criteria: *fitness* (the discovered model should allow for the behavior seen in the event log), *precision* (the discovered model should not allow for behavior completely unrelated to what was seen in the event log), *generalization* (the discovered model should generalize the example behavior seen in the event log), and *simplicity* (the discovered model should be as simple as possible). This makes process discovery a challenging and highly relevant research topic.

Activity *enhance* in Fig. 4 corresponds any effort where event data are used to repair a model (e.g., to better reflect reality) or to extend a model (e.g., to show bottlenecks). When existing process models (either discovered or hand-made) can be related to events logs, it is possible to enhance these models. The connection can be used to repair models [49] or to extend them [78, 80–82].

Activity *diagnose* in Fig. 4 does not directly use event logs and focuses on classical model-based process analysis, e.g., simulation or verification.

Activity *detect* compares de jure models with current “pre mortem” data (events of running process instances) with the goal to detect deviations at runtime. The moment a predefined rule is violated, an alert is generated [17, 62, 63].

Activity *check* in Fig. 4 analyzes conformance-related questions using event data. Historic “post mortem” data can be cross-checked with de jure models. The goal of this activity is to pinpoint deviations and quantify the level of compliance. Various conformance checking techniques have been proposed in literature [9, 22, 23, 31, 35, 52, 68, 69, 79, 90]. For example, in [79] the fitness of a model is computed by comparing the number of missing and remaining tokens with the number of consumed and produced tokens during replay. The more sophisticated technique described in [9, 22, 23] creates a so-called *alignment* which relates a trace in the event log to an execution sequence of the model that is as similar as possible. Ideally, the alignment consists of steps where log and model agree on the activity to be executed. Steps where just the model “makes a move” or just the log “makes a move” have a predefined penalty. This way the computation of fitness can be turned into an optimization problem: for each trace in the event log an alignment with the lowest costs is selected. The resulting alignments can be used for all kinds of analysis since any trace in the event log is related to an execution sequence of the model. For example, timestamps in the model can be used to compute bottlenecks and extend the model with performance information (see activity *enhance* in Fig. 4).

Activity *compare* highlights differences and commonalities between a de jure model and a de facto model. Traditional equivalence notions such as trace equivalence, bisimilarity, and branching bisimilarity [51, 67] can only be used to determine equivalence using a predefined equivalence notion, e.g., these techniques cannot be used to distinguish between very similar and highly dissimilar processes. Other notions such as graph-edit distance tend to focus on the syntax rather than the behavior of models. Therefore, recent BPM research explored various alternative similarity notions [42, 43, 58, 59, 66, 91]. Also note the *Greatest Common Divisor* (GCD) and *Least Common Multiple* (LCM) notions defined for process models in [11]. The GCD captures the common parts of two or more models. The LCM embeds all input models. We refer to [42] for a survey and empirical evaluation of some similarity notions.

Activity *promote* takes (parts of) de facto models and converts these into (parts of) de jure models, i.e., models used to control or support processes are improved based on models learned from event data. By promoting proven “best practices” to de jure models, existing processes can be improved.

The activities in the cartography and auditing categories in Fig. 4 can be viewed as “backward-looking”. The last three activities forming the navigation category are “forward-looking” and are sometimes referred to as *operational support* [2]. For example, process mining techniques can be used to make predictions about the future of a particular case and guide the user in selecting suitable actions. When comparing this with a car navigation system from TomTom or

Garmin, this corresponds to functionalities such as predicting the arrival time and guiding the driver using spoken instructions.

Activity *explore* in Fig. 4 visualizes running cases and compares these cases with similar cases that were handled earlier. The combination of event data and models can be used to explore business processes at run-time and, if needed, trigger appropriate actions.

By combining information about running cases with models (discovered or hand-made), it is possible to make predictions about the future, e.g., predicting the remaining flow time or the probability of success. Figure 4 shows that activity *predict* uses current data and models (often learned over historic data). Various techniques have been proposed in BPM literature [20, 46, 76]. Note that already a decade ago Staffware provided a so-called “prediction engine” using simulation [86].

Activity *recommend* in Fig. 4 aims to provide functionality similar to the guidance given by car navigation systems. The information used for predicting the future can also be used to recommend suitable actions (e.g. to minimize costs or time) [17, 83]. Given a set of possible next steps, the most promising step is recommended. For each possible step, simply assume that the step is made and predict the resulting performance (e.g., remaining flow time). The resulting predictions can be compared and used to rank the possible next steps.

The ten activities in Fig. 4 illustrate that process mining extends far beyond process discovery. The increasing availability and growing volume of event data suggest that the importance of process mining will continue to grow in the coming years.

It is impossible to cover the whole process mining spectrum in this tutorial paper. The reader is referred to [2, 6] for a more complete overview.

3 Preliminaries

This section introduces basic concepts related to Petri nets and event logs.

3.1 Multisets, Functions, and Sequences

Multisets are used to represent the state of a Petri net and to describe event logs where the same trace may appear multiple times.

$\mathcal{B}(A)$ 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 . Some examples: $B_1 = []$, $B_2 = [x, x, y]$, $B_3 = [x, y, z]$, $B_4 = [x, x, y, x, y, z]$, $B_5 = [x^3, y^2, z]$ are multisets over $A = \{x, y, z\}$. B_1 is the empty multiset, B_2 and B_3 both consist of three elements, and $B_4 = B_5$, i.e., the ordering of elements is irrelevant and a more compact notation may be used for repeating elements.

The standard set operators can be extended to multisets, e.g., $x \in B_2$, $B_2 \uplus B_3 = B_4$, $B_5 \setminus B_2 = B_3$, $|B_5| = 6$, etc. $\{a \in B\}$ denotes the set with all elements a for which $B(a) \geq 1$. $[f(a) \mid a \in B]$ denotes the multiset where element $f(a)$ appears $\sum_{x \in B \mid f(x)=f(a)} B(x)$ times.

A relation $R \subseteq X \times Y$ is a set of pairs. $\pi_1(R) = \{x \mid (x, y) \in R\}$ is the domain of R , $\pi_2(R) = \{y \mid (x, y) \in R\}$ is the range of R , and $\omega(R) = \pi_1(R) \cup \pi_2(R)$ are the elements of R . For example, $\omega(\{(a, b), (b, c)\}) = \{a, b, c\}$.

To the (total) function $f \in X \rightarrow Y$ maps elements from the set X onto elements of the set Y , i.e., $f(x) \in Y$ for any $x \in X$. $f \in X \not\rightarrow Y$ is a partial function with domain $\text{dom}(f) \subseteq X$ and range $\text{rng}(f) = \{f(x) \mid x \in X\} \subseteq Y$. $f \in X \rightarrow Y$ is a total function, i.e., $\text{dom}(f) = X$. A partial function $f \in X \not\rightarrow Y$ is injective if $f(x_1) = f(x_2)$ implies $x_1 = x_2$ for all $x_1, x_2 \in \text{dom}(f)$.

Definition 1 (Function Projection). Let $f \in X \not\rightarrow Y$ be a (partial) function and $Q \subseteq X$. $f \upharpoonright_Q$ is the function projected on Q : $\text{dom}(f \upharpoonright_Q) = \text{dom}(f) \cap Q$ and $f \upharpoonright_Q(x) = f(x)$ for $x \in \text{dom}(f \upharpoonright_Q)$.

The projection can also be used for multisets, e.g., $[x^3, y, z^2] \upharpoonright_{\{x, y\}} = [x^3, y]$.

$\sigma = \langle a_1, a_2, \dots, a_n \rangle \in X^*$ denotes a sequence over X of length n . $\langle \rangle$ is the empty sequence. $\text{mults}^k(\sigma) = [a_1, a_2, \dots, a_k]$ is the multiset composed of the first k elements of σ . $\text{mults}(\sigma) = \text{mults}^{|\sigma|}(\sigma)$ converts a sequence into a multiset. $\text{mults}^2(\langle a, a, b, a, b \rangle) = [a^2]$ and $\text{mults}(\langle a, a, b, a, b \rangle) = [a^3, b^2]$.

Sequences are used to represent paths in a graph and traces in an event log. $\sigma_1 \cdot \sigma_2$ is the concatenation of two sequences and $\sigma \upharpoonright_Q$ is the projection of σ on Q .

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

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

So $\langle y, z, y \rangle \upharpoonright_{\{x, y\}} = \langle y, y \rangle$. Functions can also be applied to sequences: if $\text{dom}(f) = \{x, y\}$, then $f(\langle y, z, y \rangle) = \langle f(y), f(y) \rangle$.

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

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

Summation is defined over multisets and sequences, e.g., $\sum_{x \in \langle a, a, b, a, b \rangle} f(x) = \sum_{x \in [a^3, b^2]} f(x) = 3f(a) + 2f(b)$.

3.2 Petri Nets

We use Petri nets as the process modeling language used to introduce process mining (in the large). However, as mentioned in Sect. 1 the results presented in the paper can be adapted for various other process modeling notations (BPMN

models, BPEL specifications, UML activity diagrams, Statecharts, C-nets, heuristic nets, etc.). This does not imply that these notations are equivalent. There are differences in *expensiveness* (e.g., classical Petri nets are not Turing complete, but most extension of Petri nets are) and *suitability* (cf. research on the workflow patterns [15]). Translations are often “lossy”, i.e., the model after translation may allow for more or less behavior. However, in practice this is not a problem as the basic concepts are often the same. There is also a trade-off between accuracy and simplicity. For example, inclusive OR-joins are not directly supported by Petri nets, because an OR-join may need to synchronize a variable (at design-time unknown) number of inputs. Using a rather involved translation it is possible to model this in terms of classical Petri nets using so-called “true” and “false” tokens [15]. This only works if there are no arbitrary unstructured loops. See for example the many translations proposed for the mapping from BPEL to Petri nets [56, 60, 72]. There also exists a naïve much simpler translation that includes the original behavior (but also more) [1, 10]. Using Single-Entry Single-Exit (SESE) components and the refined process structure tree (RPST) [74, 87] it is often possible to convert an unstructured graph-based model into a structured model. Also see the approaches to convert Petri nets and BPMN models into BPEL [16, 73].

The above examples illustrate that many conversions are possible depending on the desired outcome (accuracy versus simplicity). It is also important to stress that the representational bias used during process discovery may be different from the representational bias used to present the result to end-users. For example, one may use Petri nets during discovery and convert the final result to BPMN.

In this paper we would like to steer away from notational issues and conversions and restrict ourselves to Petri nets as a representation for process models. By using Petri nets we minimize the notational overhead allowing us the focus on the key ideas.

Definition 4 (Petri Net). *A Petri net is a tuple $N = (P, T, F)$ with P the set of places, T the set of transitions, $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ the flow relation.*

Figure 5 shows a Petri net $N = (P, T, F)$ with $P = \{start, c1, \dots, c9, end\}$, $T = \{t1, t2, \dots, t11\}$, and $F = \{(start, t1), (t1, c1), (t1, c2), \dots, (t11, end)\}$. The state of a Petri net, called *marking*, is a multiset of places indicating how many *tokens* each place contains. $[start]$ is the initial marking shown in Fig. 5. Another potential marking is $[c1^{10}, c2^5, c4^5]$. This is the state with ten tokens in $c1$, five tokens in $c2$, and five tokens in $c4$.

Definition 5 (Marking). *Let $N = (P, T, F)$ be a Petri net. A marking M is a multiset of places, i.e., $M \in \mathcal{B}(P)$.*

A Petri net $N = (P, T, F)$ defines a directed graph with nodes $P \cup T$ and edges F . For any $x \in P \cup T$, $\overset{N}{\bullet}x = \{y \mid (y, x) \in F\}$ denotes the set of input nodes and

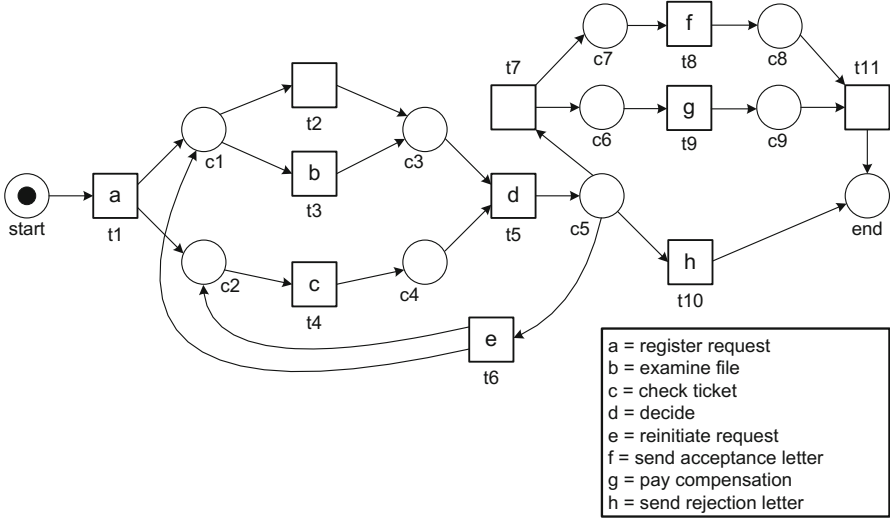


Fig. 5. A labeled Petri net.

$x \bullet^N = \{y \mid (x, y) \in F\}$ denotes the set of output nodes. We drop the superscript N if it is clear from the context.

A transition $t \in T$ is *enabled* in marking M of net N , denoted as $(N, M)[t]$, if each of its input places $\bullet t$ contains at least one token. Consider the Petri net N in Fig. 5 with $M = [c3, c4]$: $(N, M)[t5]$ because both input places of $t5$ are marked.

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 $t \bullet$. Formally: $M' = (M \setminus \bullet t) \uplus t \bullet$ is the marking resulting from firing enabled transition t in marking M of Petri net N . $(N, M)[t](N, M')$ denotes that t is enabled in M and firing t results in marking M' . For example, $(N, [start])[t1](N, [c1, c2])$ and $(N, [c3, c4])[t5](N, [c5])$ for the net in Fig. 5.

Let $\sigma = \langle t_1, t_2, \dots, t_n \rangle \in T^*$ be a sequence of transitions. $(N, M)[\sigma](N, M')$ denotes that there is a set of markings M_0, M_1, \dots, M_n such that $M_0 = M$, $M_n = M'$, and $(N, M_i)[t_{i+1}](N, M_{i+1})$ for $0 \leq i < n$. A marking M' is *reachable* from M if there exists a σ such that $(N, M)[\sigma](N, M')$. For example, $(N, [start])[\sigma](N, [end])$ with $\sigma = \langle t1, t3, t4, t5, t10 \rangle$ for the net in Fig. 5.

Definition 6 (Labeled Petri Net). A labeled Petri net $N = (P, T, F, l)$ is a Petri net (P, T, F) with labeling function $l \in T \rightarrow \mathcal{U}_A$ where \mathcal{U}_A is some universe of activity labels. Let $\sigma_v = \langle a_1, a_2, \dots, a_n \rangle \in \mathcal{U}_A^*$ be a sequence of activities. $(N, M)[\sigma_v \triangleright (N, M')$ if and only if there is a sequence $\sigma \in T^*$ such that $(N, M)[\sigma](N, M')$ and $l(\sigma) = \sigma_v$ (cf. Definition 3).

If $t \notin \text{dom}(l)$, it is called invisible. An occurrence of visible transition $t \in \text{dom}(l)$ corresponds to observable activity $l(t)$. The Petri net in Fig. 5 is labeled. The

labeling function is defined as follows: $dom(l) = \{t1, t3, t4, t5, t6, t8, t9, t10\}$, $l(t1) = a$ (a is a shorthand for “register request”), $l(t3) = b$ (“examine file”), $l(t4) = c$ (“check ticket”), $l(t5) = d$ (“decide”), $l(t6) = e$ (“reinitiate request”), $l(t8) = f$ (“send acceptance letter”), $l(t9) = g$ (“pay compensation”), and $l(t10) = h$ (“send rejection letter”). Unlabeled transitions correspond to so-called “silent actions”, i.e., transitions $t2$, $t7$, and $t11$ are unobservable.

Given the Petri net N in Fig. 5: $(N, [start])[σ_v \triangleright (N, [end])$ for $σ_v = \langle a, c, d, f, g \rangle$ because $(N, [start])[σ](N, [end])$ with $σ = \langle t1, t2, t4, t5, t7, t8, t9, t11 \rangle$ and $l(σ) = σ_v$.

In the context of process mining, we always consider processes that start in an initial state and end in a well-defined end state. For example, given the net in Fig. 5 we are interested in so-called *complete* firing sequences starting in $M_{init} = [start]$ and ending in $M_{final} = [end]$. Therefore, we define the notion of a *system net*.

Definition 7 (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.

Definition 8 (System Net Notations). Let $SN = (N, M_{init}, M_{final}) \in \mathcal{U}_{SN}$ be a system net with $N = (P, T, F, l)$.

- $T_v(SN) = dom(l)$ is the set of visible transitions in SN ,
- $A_v(SN) = rng(l)$ is the set of corresponding observable activities in SN ,
- $T_v^u(SN) = \{t \in T_v(SN) \mid \forall t' \in T_v(SN) \ l(t) = l(t') \Rightarrow t = t'\}$ is the set of unique visible transitions in SN (i.e., there are no other transitions having the same visible label), and
- $A_v^u(SN) = \{l(t) \mid t \in T_v^u(SN)\}$ is the set of corresponding unique observable activities in SN .

Given a system net, $\phi(SN)$ 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.

Definition 9 (Traces). Let $SN = (N, M_{init}, M_{final}) \in \mathcal{U}_{SN}$ be a system net. $\phi(SN) = \{\sigma_v \mid (N, M_{init})[\sigma_v \triangleright (N, M_{final})]\}$ is the set of visible traces starting in M_{init} and ending in M_{final} . $\phi_f(SN) = \{\sigma \mid (N, M_{init})[\sigma](N, M_{final})\}$ is the corresponding set of complete firing sequences.

For Fig. 5: $\phi(SN) = \{\langle a, c, d, f, g \rangle, \langle a, c, b, d, f, g \rangle, \langle a, c, d, h \rangle, \langle a, b, c, d, e, c, d, h \rangle, \dots\}$ and $\phi_f(SN) = \{\langle t1, t2, t4, t5, t7, t8, t9, t11 \rangle, \langle t1, t3, t4, t5, t10 \rangle, \dots\}$. Because of the loop involving transition $t6$ there are infinitely many visible traces and complete firing sequences.

Traditionally, the bulk of Petri net research focused on *model-based analysis*. Moreover, the largest proportion of model-based analysis techniques is limited to *functional properties*. Generic techniques such as model checking can be used to check whether a Petri net has particular properties, e.g., free of deadlocks. Petri-net-specific notions such as traps, siphons, place invariants, transition invariants,

and coverability graphs are often used to verify desired functional properties, e.g., liveness or safety properties [77]. Consider for example the notion of *soundness* defined for *WorkFlow nets* (WF-nets) [13]. The Petri net shown in Fig. 5 is a WF-net because there is a unique source place *start*, a unique sink place *end*, and all nodes are on a path from *start* to *end*. A WF-net is sound if and only if the following three requirements are satisfied: (1) *option to complete*: it is always still possible (i.e., from any reachable marking) to reach the state which just marks place *end*, (2) *proper completion*: if place *end* is marked all other places are empty, and (3) *no dead transitions*: it should be possible to execute an arbitrary transition by following the appropriate route through the WF-net. The WF-net in Fig. 5 is sound and as a result cases cannot get stuck before reaching the end (termination is always possible) and all parts of the process can be activated (no dead segments). Obviously, soundness is important in the context of business processes and process mining. Fortunately, there exist nice theorems connecting soundness to classical Petri-net properties. For example, a WF-net is sound if and only if the corresponding short-circuited Petri net is live and bounded. Hence, proven techniques and tools can be used to verify soundness.

Although the results in this paper are more general and not limited of WF-nets, all examples in this paper use indeed WF-nets. As indicated most of Petri net literature and tools focuses on model-based analysis thereby ignoring actual observed process behavior. Yet, the confrontation between modeled and observed behavior is essential for understanding and improving real-life processes and systems.

3.3 Event Log

As indicated earlier, *event logs* serve as the starting point for process mining. An event log is a multiset of *traces*. Each trace describes the life-cycle of a particular *case* (i.e., a *process instance*) in terms of the *activities* executed.

Definition 10 (Trace, Event Log). Let $A \subseteq \mathcal{U}_A$ be a set of activities. A trace $\sigma \in A^*$ is a sequence of activities. $L \in \mathcal{B}(A^*)$ is an event log, i.e., a multiset of traces.

An event log is a *multiset* of traces because there can be multiple cases having the same trace. In this simple definition of an event log, an event refers to just an *activity*. Often event logs store additional information about events. For example, many process mining techniques use extra information such as the *resource* (i.e., person or device) executing or initiating the activity, the *timestamp* of the event, or *data elements* recorded with the event (e.g., the size of an order). In this paper, we abstract from such information. However, the results presented can easily be extended to event logs containing additional information.

An example log is $L_1 = [\langle a, c, d, f, g \rangle^{10}, \langle a, c, d, h \rangle^5, \langle a, b, c, d, e, c, d, g, f \rangle^5]$. L_1 contains information about 20 cases, e.g., 10 cases followed trace $\langle a, c, d, f, g \rangle$. There are $10 \times 5 + 5 \times 4 + 5 \times 9 = 115$ events in total.

The projection function \downarrow_X (cf. Definition 2) is generalized to event logs, i.e., for some event log $L \in \mathcal{B}(A^*)$ and set $X \subseteq A$: $L \downarrow_X = [\sigma \downarrow_X \mid \sigma \in L]$. For example, $L_1 \downarrow_{\{a,g,h\}} = [\langle a, g \rangle^{15}, \langle a, h \rangle^5]$. We will refer to these projected event logs as *sublogs*.

4 Process Discovery

Process discovery is one of the most challenging process mining tasks. In this paper we consider the basic setting where we want to learn a system net $SN = (N, M_{init}, M_{final}) \in \mathcal{U}_{SN}$ from an event log $L \in \mathcal{B}(A^*)$. We will present two process discovery techniques: the α -algorithm and an approach based on language-based regions. These techniques have many limitations (e.g., unable to deal with noise), but they serve as a good starting point for better understanding this challenging topic.

4.1 Alpha Algorithm

First we describe the α -algorithm [21]. This was the first process discovery technique able to discover concurrency. Moreover, unlike most other techniques, the α -algorithm was proven to be correct for a clearly defined class of processes [21]. Nevertheless, we would like to stress that the basic algorithm has many limitations including the inability to deal with noise, particular loops, and non-free-choice behavior. Yet, it provides a good introduction into the topic. The α -algorithm is simple and many of its ideas have been embedded in more complex and robust techniques. We will use the algorithm as a baseline for discussing the challenges related to process discovery.

The α -algorithm *scans the event log for particular patterns*. For example, if activity a is followed by b but b is never followed by a , then it is assumed that there is a causal dependency between a and b .

Definition 11 (Log-based ordering relations). *Let $L \in \mathcal{B}(A^*)$ be an event log over A , i.e., $L \in \mathcal{B}(A^*)$. Let $a, b \in A$:*

- $a >_L b$ if and only if there is a trace $\sigma = \langle t_1, t_2, t_3, \dots, t_n \rangle$ and $i \in \{1, \dots, n-1\}$ such that $\sigma \in L$ and $t_i = a$ and $t_{i+1} = b$;
- $a \rightarrow_L b$ if and only if $a >_L b$ and $b \not>_L a$;
- $a \#_L b$ if and only if $a \not>_L b$ and $b \not>_L a$; and
- $a \parallel_L b$ if and only if $a >_L b$ and $b >_L a$.

Consider for instance event log $L_2 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^2, \langle a, e, d \rangle]$. For this event log the following log-based ordering relations can be found.

$$\begin{aligned}
 >_{L_2} &= \{(a, b), (a, c), (a, e), (b, c), (c, b), (b, d), (c, d), (e, d)\} \\
 \rightarrow_{L_2} &= \{(a, b), (a, c), (a, e), (b, d), (c, d), (e, d)\} \\
 \#_{L_2} &= \{(a, a), (a, d), (b, b), (b, e), (c, c), (c, e), (d, a), (d, d), (e, b), (e, c), (e, e)\} \\
 \parallel_{L_2} &= \{(b, c), (c, b)\}
 \end{aligned}$$

Relation $>_{L_2}$ contains all pairs of activities in a “directly follows” relation. $c >_{L_2} d$ because d directly follows c in trace $\langle a, b, c, d \rangle$. However, $d \not>_{L_2} c$ because c never directly follows d in any trace in the log. \rightarrow_{L_2} contains all pairs of activities in a “causality” relation, e.g., $c \rightarrow_{L_2} d$ because sometimes d directly follows c and never the other way around ($c >_{L_2} d$ and $d \not>_{L_2} c$). $b \parallel_{L_2} c$ because $b >_{L_2} c$ and $c >_{L_2} b$, i.e., sometimes c follows b and sometimes the other way around. $b \#_{L_2} c$ because $b \not>_{L_2} c$ and $c \not>_{L_2} b$.

For any log L over A and $x, y \in A$: $x \rightarrow_L y$, $y \rightarrow_L x$, $x \#_L y$, or $x \parallel_L y$, i.e., precisely one of these relations holds for any pair of activities. The log-based ordering relations can be used to *discover patterns* in the corresponding process model as is illustrated in Fig. 6. If a and b are in sequence, the log will show $a \rightarrow_L b$. If after a there is a choice between b and c , the log will show $a \rightarrow_L b$, $a \rightarrow_L c$, and $b \#_L c$ because a can be followed by b and c , but b will not be followed by c and vice versa. The logical counterpart of this so-called XOR-split pattern is the XOR-join pattern as shown in Fig. 6(b-c). If $a \rightarrow_L c$, $b \rightarrow_L c$, and $a \#_L b$, then this suggests that after the occurrence of either a or b , c should happen. Figure 6(d-e) shows the so-called AND-split and AND-join patterns. If $a \rightarrow_L b$, $a \rightarrow_L c$, and $b \parallel_L c$, then it appears that after a both b and c can be executed in parallel (AND-split pattern). If $a \rightarrow_L c$, $b \rightarrow_L c$, and $a \parallel_L b$, then the log suggests that c needs to synchronize a and b (AND-join pattern).

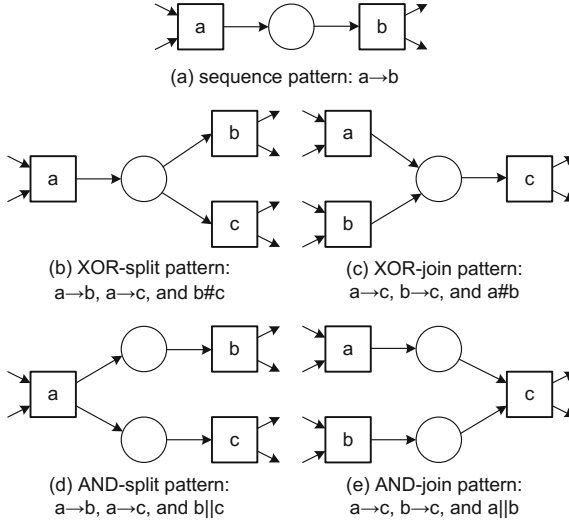


Fig. 6. Typical process patterns and the footprints they leave in the event log

Figure 6 only shows simple patterns and does not present the additional conditions needed to extract the patterns. However, it provides some initial insights useful when reading the formal definition of the α -algorithm [21].

Definition 12. (α -algorithm). Let $L \in \mathcal{B}(A^*)$ be an event log over A . $\alpha(L)$ produces a system net and is defined as follows:

1. $T_L = \{t \in A \mid \exists \sigma \in L \ t \in \sigma\}$,
2. $T_I = \{t \in A \mid \exists \sigma \in L \ t = \text{first}(\sigma)\}$,
3. $T_O = \{t \in A \mid \exists \sigma \in L \ t = \text{last}(\sigma)\}$,
4. $X_L = \{(A, B) \mid A \subseteq T_L \wedge A \neq \emptyset \wedge B \subseteq T_L \wedge B \neq \emptyset \wedge \forall a \in A \forall b \in B \ a \rightarrow_L b \wedge \forall a_1, a_2 \in A \ a_1 \#_L a_2 \wedge \forall b_1, b_2 \in B \ b_1 \#_L b_2\}$,
5. $Y_L = \{(A, B) \in X_L \mid \forall (A', B') \in X_L \ A \subseteq A' \wedge B \subseteq B' \implies (A, B) = (A', B')\}$,
6. $P_L = \{p_{(A, B)} \mid (A, B) \in Y_L\} \cup \{i_L, o_L\}$,
7. $F_L = \{(a, p_{(A, B)}) \mid (A, B) \in Y_L \wedge a \in A\} \cup \{(p_{(A, B)}, b) \mid (A, B) \in Y_L \wedge b \in B\} \cup \{(i_L, t) \mid t \in T_I\} \cup \{(t, o_L) \mid t \in T_O\}$,
8. $l_L \in T_L \rightarrow A$ with $l(t) = t$ for $t \in T_L$, and
9. $\alpha(L) = (N, M_{\text{init}}, M_{\text{final}})$ with $N = (P_L, T_L, F_L, l_L)$, $M_{\text{init}} = [i_L]$, $M_{\text{final}} = [o_L]$.

In Step 1 it is checked which activities do appear in the log (T_L). These are the observed activities and correspond to the transitions of the generated system net. T_I is the set of start activities, i.e., all activities that appear first in some trace (Step 2). T_O is the set of end activities, i.e., all activities that appear last in some trace (Step 3). Steps 4 and 5 form the core of the α -algorithm. The challenge is to determine the places of the Petri net and their connections. We aim at constructing places named $p_{(A, B)}$ such that A is the set of input transitions ($\bullet p_{(A, B)} = A$) and B is the set of output transitions ($p_{(A, B)} \bullet = B$) of $p_{(A, B)}$.

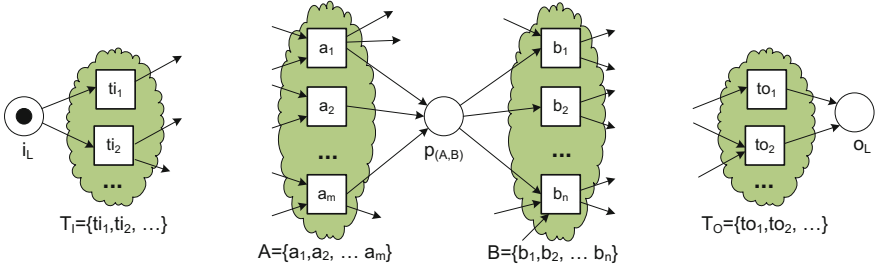


Fig. 7. Place $p_{(A, B)}$ connects the transitions in set A to the transitions in set B , i_L is the input place of all start transition T_I , and o_L is the output place of all end transition T_O .

The basic motivation for finding $p_{(A, B)}$ is illustrated by Fig. 7. All elements of A should have causal dependencies with all elements of B , i.e., for all $(a, b) \in A \times B$: $a \rightarrow_L b$. Moreover, the elements of A should never follow one another, i.e., for all $a_1, a_2 \in A$: $a_1 \#_L a_2$. A similar requirement holds for B . Let us consider $L_2 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^2, \langle a, e, d \rangle]$ again. Clearly, $A = \{a\}$ and $B = \{b, e\}$ meet the requirements stated in Step 4. Also $A' = \{a\}$ and $B' = \{b\}$ meet the

same requirements. X_L is the set of all such pairs that meet the requirements just mentioned. In this case:

$$X_{L_2} = \{(\{a\}, \{b\}), (\{a\}, \{c\}), (\{a\}, \{e\}), (\{a\}, \{b, e\}), (\{a\}, \{c, e\}), (\{b\}, \{d\}), (\{c\}, \{d\}), (\{e\}, \{d\}), (\{b, e\}, \{d\}), (\{c, e\}, \{d\})\}$$

If one would insert a place for any element in X_{L_2} , there would be too many places. Therefore, only the “maximal pairs” (A, B) should be included. Note that for any pair $(A, B) \in X_L$, non-empty set $A' \subseteq A$, and non-empty set $B' \subseteq B$, it is implied that $(A', B') \in X_L$. In Step 5, all non-maximal pairs are removed, thus yielding:

$$Y_{L_2} = \{(\{a\}, \{b, e\}), (\{a\}, \{c, e\}), (\{b, e\}, \{d\}), (\{c, e\}, \{d\})\}$$

Every element of $(A, B) \in Y_L$ corresponds to a place $p_{(A,B)}$ connecting transitions A to transitions B . In addition P_L also contains a unique source place i_L and a unique sink place o_L (cf. Step 6). In Step 7 the arcs of the Petri net are generated. All start transitions in T_I have i_L as an input place and all end transitions T_O have o_L as output place. All places $p_{(A,B)}$ have A as input nodes and B as output nodes. Figure 8 shows the resulting system net. Since transition identifiers and labels coincide ($l(t) = t$ for $t \in T_L$) we only show the labels. For any event log L , $\alpha(L) = (N, M_{init}, M_{final})$ with $N = (P_L, T_L, F_L, l_L)$, $M_{init} = [i_L]$, $M_{final} = [o_L]$ aims to describe the behavior recorded in L .

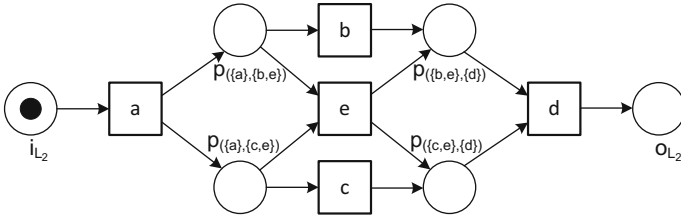


Fig. 8. System net $\alpha(L_2) = (N, [i_{L_2}], [o_{L_2}])$ for event log $L_2 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^2, \langle a, e, d \rangle]$.

Next, we consider the following three events logs $L_{3a} = [\langle a, c, d \rangle^{88}, \langle a, c, e \rangle^{82}, \langle b, c, d \rangle^{83}, \langle b, c, e \rangle^{87}]$, $L_{3b} = [\langle a, c, d \rangle^{88}, \langle b, c, e \rangle^{87}]$, $L_{3c} = [\langle a, c, d \rangle^{88}, \langle a, c, e \rangle^2, \langle b, c, d \rangle^3, \langle b, c, e \rangle^{87}]$. $\alpha(L_{3a}) = SN_{3a}$, i.e., the system net depicted in Fig. 9 without places p_3 and p_4 (modulo renaming of places). It is easy to check that all traces in L_{3a} are allowed by the discovered model SN_{3a} and that all firing sequences of the SN_{3a} appear in the event log. Now consider L_{3b} . Surprisingly, $\alpha(L_{3b}) = \alpha(L_{3a}) = SN_{3a}$ (modulo renaming of places). Note that event logs L_{3a} and L_{3b} are identical with respect to the “directly follows” relation, i.e., $>_{L_{3a}} = >_{L_{3b}}$. The α -algorithm is unable to discover SN_{3b} because the dependencies between on the one hand a and d and on the other hand c and e are

non-local: a , d , c and e never directly follow one another. Still, $\alpha(L_{3a})$ allows for all behavior in L_{3b} (and more). Sometimes it is not so clear which model is preferable. Consider for example L_{3c} where two traces are infrequent. SN_{3a} allows for all behavior in L_{3c} , including the infrequent ones. However, SN_{3b} is more precise as it only shows the “highways” in L_{3c} . Often people are interested in the “80/20 model”, i.e., the process model that can describe 80 % of the behavior seen in the log. This model is typically relatively simple because the remaining 20 % of the log often account for 80 % of the variability in the process. Hence, people may prefer SN_{3b} over SN_{3a} for L_{3c} .

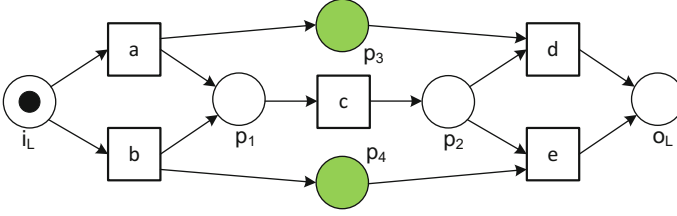


Fig. 9. $SN_{3a} = (N_{3a}, [i_L], [o_L])$ is the system net depicted *without* places p_3 and p_4 . $SN_{3b} = (N_{3b}, [i_L], [o_L])$ is the same net but now *including* places p_3 and p_4 . (Only the transition labels are shown.)

If we assume that all transitions in Fig. 5 have a visible label and we have an event log L that is complete with respect to the “directly follows” relation (i.e., $x >_L y$ if and only if y can be directly followed by x in the model), then the α -algorithm is able to rediscover the original model. If $t7$ is invisible (not recorded in event log), then a more compact, but correct, model is derived by the α -algorithm. If $t2$ or $t11$ is invisible, the α -algorithm fails to discover a correct model, e.g., skipping activity b ($t2$) does not leave a trail in the event log and requires a more sophisticated discovery technique.

4.2 Region-Based Process Discovery

In the context of Petri nets, researchers have been looking at the so-called *synthesis problem*, i.e., constructing a system model its desired behavior. State-based regions can be used to construct a Petri net from a transition system [36, 48]. Language-based regions can be used to construct a Petri net from a prefix-closed language. Synthesis approaches using language-based regions can be applied directly to event logs. To apply state-based regions, one first needs to create a transition system as shown in [19]. Here, we restrict ourselves to an informal introduction to language-based regions.

Suppose, we have an event log $L \in \mathcal{B}(A^*)$. For this log one could construct a system net SN without any places and just transitions being continuously enabled. Given a set of transitions with labels A this system net is able to reproduce any event log $L \in \mathcal{B}(A^*)$. Such a Petri net is called the “flower model”

and adding places to this model can only limit the behavior. Language-based regions aim at finding places such that behavior is restricted properly, i.e., allow for the observed and likely behavior [27, 28, 32, 93].

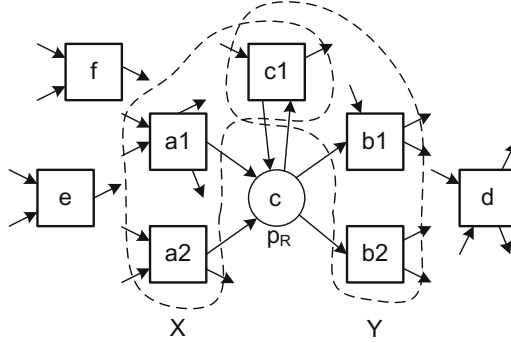


Fig. 10. Region $R = (X, Y, c)$ corresponding to place p_R : $X = \{a1, a2, c1\} = \bullet p_R$, $Y = \{b1, b2, c1\} = p_R \bullet$, and c is the initial marking of p_R

Consider for example place p_R in Fig. 10. Removing place p_R will not remove any behavior. However, adding p_R may remove behavior possible in the Petri net without this place. The behavior gets restricted when a place is empty while one of its output transitions wants to consume a token from it. For example, $b1$ is blocked if p_R is unmarked while all other input places of $b1$ are marked. Suppose now that we have a multiset of traces L . If these traces are possible in the net with place p_R , then they are also possible in the net without p_R . The reverse does not always hold. This triggers the question whether p_R can be added without disabling any of the traces in L . This is what regions are all about.

Definition 13 (Language-Based Region). Let $L \in \mathcal{B}(A^*)$ be an event log. $R = (X, Y, c)$ is a region of L if and only if:

- $X \subseteq A$ is the set of input transitions of R ;
- $Y \subseteq A$ is the set of output transitions of R ;
- $c \in \{0, 1\}$ is the initial marking of R ; and
- for any $\sigma \in L$, $k \in \{1, \dots, |\sigma|\}$:

$$c + \sum_{t \in X} \text{mults}^{k-1}(\sigma)(t) - \sum_{t \in Y} \text{mults}^k(\sigma)(t) \geq 0.$$

$R = (X, Y, c)$ is a region of L if and only if inserting a place p_R with $\bullet p_R = X$, $p_R \bullet = Y$, and initially c tokens does not disable the execution of any of the traces in L . To check this, Definition 13 inspects all events in the event log. Let $\sigma \in L$ be a trace in the log. $a = \sigma(k)$ is the k -th event in this trace. This event should

not be disabled by place p_R . Therefore, we calculate the number of tokens $M(p_R)$ that are in this place just before the occurrence of the k -th event.

$$M(p_R) = c + \sum_{t \in X} \text{mults}^{k-1}(\sigma)(t) - \sum_{t \in Y} \text{mults}^{k-1}(\sigma)(t)$$

$\text{mults}^{k-1}(\sigma)$ is the multiset of events that occurred before the occurrence of the k -th event. $\sum_{t \in X} \text{mults}^{k-1}(\sigma)(t)$ counts the number of tokens produced for place p_R , $\sum_{t \in Y} \text{mults}^{k-1}(\sigma)(t)$ counts the number of tokens consumed from this place, and c is the initial number of tokens in p_R . Therefore, $M(p_R)$ is indeed the number of tokens in p_R just before the occurrence of the k -th event. This number should be positive. In fact, there should be at least one token in p_R if $a \in Y$. In other words, $M(p_R)$ minus the number of tokens consumed from p_R by the k -th event should be non-negative. Hence:

$$M(p_R) - \sum_{t \in Y} [a](t) = c + \sum_{t \in X} \text{mults}^{k-1}(\sigma)(t) - \sum_{t \in Y} \text{mults}^k(\sigma)(t) \geq 0.$$

This shows that a region R , according to Definition 13, indeed corresponds to a so-called *feasible place* p_R , i.e., a place that can be added without disabling any of the traces in the event log.

The requirement stated in Definition 13 can also be formulated in terms of an inequation system. To illustrate this we use the example log $L_{3b} = [\langle a, c, d \rangle^{88}, \langle b, c, e \rangle^{87}]$ for which the α -algorithm was unable to find a suitable model. There are five activities. For each activity t we introduce two variables: x_t and y_t . $x_t = 1$ if transition t produces a token for p_R and $x_t = 0$ if not. $y_t = 1$ if transition t consumes a token from p_R and $y_t = 0$ if not. A potential region $R = (X, Y, c)$ corresponds to an assignment for all of these variables: $x_t = 1$ if $t \in X$, $x_t = 0$ if $t \notin X$, $y_t = 1$ if $t \in Y$, $y_t = 0$ if $t \notin Y$. The requirement stated in Definition 13 can now be reformulated in terms of the variables $x_a, x_b, x_c, x_d, x_e, y_a, y_b, y_c, y_d, y_e$, and c for event log L_{3b} :

$$\begin{aligned} c - y_a &\geq 0 \\ c + x_a - (y_a + y_c) &\geq 0 \\ c + x_a + x_c - (y_a + y_c + y_d) &\geq 0 \\ c - y_b &\geq 0 \\ c + x_b - (y_b + y_c) &\geq 0 \\ c + x_b + x_c - (y_b + y_c + y_e) &\geq 0 \\ c, x_a, \dots, x_e, y_a, \dots, y_e &\in \{0, 1\} \end{aligned}$$

Note that these inequations are based on all non-empty prefixes of $\langle a, c, d \rangle$ and $\langle b, c, e \rangle$. Any solution of this linear inequation system corresponds to a region. Some example solutions are:

$$\begin{aligned} R_1 &= (\emptyset, \{a, b\}, 1) \\ c = y_a = y_b &= 1, \quad x_a = x_b = x_c = x_d = x_e = y_c = y_d = y_e = 0 \end{aligned}$$

$$R_2 = (\{a, b\}, \{c\}, 0)$$

$$x_a = x_b = y_c = 1, \quad c = x_c = x_d = x_e = y_a = y_b = y_d = y_e = 0$$

$$R_3 = (\{c\}, \{d, e\}, 0)$$

$$x_c = y_d = y_e = 1, \quad c = x_a = x_b = x_d = x_e = y_a = y_b = y_c = 0$$

$$R_4 = (\{d, e\}, \emptyset, 0)$$

$$x_d = x_e = 1, \quad c = x_a = x_b = x_c = y_a = y_b = y_c = y_d = y_e = 0$$

$$R_5 = (\{a\}, \{d\}, 0)$$

$$x_a = y_d = 1, \quad c = x_b = x_c = x_d = x_e = y_a = y_b = y_c = y_e = 0$$

$$R_6 = (\{b\}, \{e\}, 0)$$

$$x_b = y_e = 1, \quad c = x_a = x_c = x_d = x_e = y_a = y_b = y_c = y_d = 0$$

Consider for example $R_6 = (\{b\}, \{e\}, 0)$. This corresponds to the solution $x_b = y_e = 1$ and $c = x_a = x_c = x_d = x_e = y_a = y_b = y_c = y_d = 0$. If we fill out the values in the inequation system, we can see that this is indeed a solution. If we construct a Petri net based on these six regions, we obtain SN_{3b} , i.e., the system net depicted in Fig. 9 including places p_3 and p_4 (modulo renaming of places).

Suppose that the trace $\langle a, c, e \rangle$ is added to event log L_{3b} . This results in three additional inequations:

$$\begin{aligned} c - y_a &\geq 0 \\ c + x_a - (y_a + y_c) &\geq 0 \\ c + x_a + x_c - (y_a + y_c + y_e) &\geq 0 \end{aligned}$$

Only the last inequation is new. Because of this inequation, $x_b = y_e = 1$ and $c = x_a = x_c = x_d = x_e = y_a = y_b = y_c = y_d = 0$ is no longer a solution. Hence, $R_6 = (\{b\}, \{e\}, 0)$ is not a region anymore and place p_4 needs to be removed from the system net shown in Fig. 9. After removing this place, the resulting system net indeed allows for $\langle a, c, e \rangle$.

One of the problems of directly applying language-based regions is that the linear inequation system has many solutions. Few of these solutions correspond to sensible places. For example, $x_a = x_b = y_d = y_e = 1$ and $c = x_c = x_d = x_e = y_a = y_b = y_c = 0$ also defines a region: $R_7 = (\{a, b\}, \{d, e\}, 0)$. However, adding this place to Fig. 9 would only clutter the diagram. Another example is $c = x_a = x_b = y_c = 1$ and $x_c = x_d = x_e = y_a = y_b = y_d = y_e = 0$, i.e., region $R_8 = (\{a, b\}, \{c\}, 1)$. This region is a weaker variant of R_2 as the place is initially marked.

Another problem is that classical techniques for language-based regions aim at a Petri net that does not allow for any behavior not seen in the log [28]. This means that the log is considered to be complete. This is very unrealistic and results in models that are complex and overfitting. To address these problems dedicated techniques have been proposed. For instance, in [93] it is shown how to avoid overfitting and how to ensure that the resulting model has

desirable properties (WF-net, free-choice, etc.). Nevertheless, pure region-based techniques tend to have problems handling noise and incompleteness.

4.3 Other Process Discovery Approaches

The α -algorithm and the region-based approach just presented have many limitations. However, there are dozens of more advanced process discovery approaches. For example, consider *genetic process mining* techniques [30,65]. The idea of genetic process mining is to use evolution (“survival of the fittest”) when searching for a process model. Like in any genetic algorithm there are four main steps: (a) initialization, (b) selection, (c) reproduction, and (d) termination. In the *initialization step* the initial population is created. This is the first generation of individuals to be used. Here an individual is a process model (e.g., a Petri net, transition system, Markov chain or process tree). Using the activity names appearing in the log, process models are created randomly. In a generation there may be hundreds or thousands of individuals (e.g., candidate Petri nets). In the *selection step*, the fitness of each individual is computed. A fitness function determines the quality of the individual in relation to the log.² Tournaments among individuals and elitism are used to ensure that genetic material of the best process models has the highest probability of being used for the next generation: *survival of the fittest*. In the *reproduction phase* the selected parent individuals are used to create new offspring. Here two genetic operators are used: *crossover* (creating child models that share parts of the genetic material of their parents) and *mutation* (e.g., randomly adding or deleting causal dependencies). Through reproduction and elitism a new generation is created. For the models in the new generation fitness is computed. Again the best individuals move on to the next round (elitism) or are used to produce new offspring. This is repeated and the expectation is that the “quality” of each generation gets better and better. The evolution process terminates when a satisfactory solution is found, i.e., a model having at least the desired fitness.

Next to genetic process mining techniques [30,65] there are many other discovery techniques. For example, *heuristic* [92] and *fuzzy* [53] mining techniques are particularly suitable for practical applications, but are outside the scope of this tutorial paper (see [2] for a more comprehensive overview).

5 Conformance Checking

Conformance checking techniques investigate how well an event log $L \in \mathcal{B}(A^*)$ and a system net $SN = (N, M_{init}, M_{final})$ fit together. Note that SN may have been discovered through process mining or may have been made by hand. In any case, it is interesting to compare the observed example behavior in L with the potential behavior of SN .

² Note that “fitness” in genetic mining has a different meaning than the (replay) fitness at other places in this paper. Genetic fitness corresponds to the more general notion of conformance including replay fitness, simplicity, precision, and generalization.

5.1 Quality Dimensions

Conformance checking can be done for various reasons. First of all, it may be used to audit processes to see whether reality conforms to some normative or descriptive model [14, 41]. Deviations may point to:

- *fraud* (deliberate non-conforming behavior),
- *inefficiencies* (carelessness or sloppiness causing unnecessary delays or costs),
- *exceptions* (selected cases are handled in an ad-hoc manner because of special circumstances not covered by the model),
- *poorly designed procedures* (to get the work done people need to deviate from the model continuously), or
- *outdated procedures* (the process description does not match reality anymore because the process evolved over time).

Second, conformance checking can be used to evaluate the performance of a process discovery technique. In fact, genetic process mining algorithms use conformance checking to select the candidate models used to create the next generation of models [30, 65].

There are four quality dimensions for comparing model and log: (1) *replay fitness*, (2) *simplicity*, (3) *precision*, and (4) *generalization* [2]. A model with good *replay 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. If there are two models explaining the behavior seen in the log, we generally prefer the *simplest* 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 (“flower model”) that is able to replay all traces in an event log (but also any other event log referring to the same set of activities). 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 seen yet. A model is *precise* if it does not allow for “too much” behavior. Clearly, the “flower model” lacks precision. A model that is not precise is “underfitting”. Underfitting is the problem that the model over-generalizes the example behavior in the log (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” [8, 9]. Overfitting is the problem that a very 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). Process discovery techniques typically have problems finding the appropriate balance between precision and generalization because the event log only contains “positive examples”, i.e., the event log does not indicate what could *not* happen.

In the remainder, we will focus on fitness. However, replay fitness is the starting point to the other quality dimensions [8, 9, 30].

5.2 Token-Based Replay

A simple fitness metric is the fraction of perfectly fitting traces. For example, the system net shown in Fig. 8 has a fitness of 0.8 for event log $L_4 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^3, \langle a, e, d \rangle^2, \langle a, d \rangle, \langle a, e, e, d \rangle]$ because 8 of the 10 traces fit perfectly. Such a naïve fitness metric is less suitable for more realistic processes because it cannot distinguish between “almost fitting” traces and traces that are completely unrelated to the model. Therefore, we also need a more refined fitness notion defined *at the level of events* rather than full traces. Rather than aborting the replay of a trace once we encounter a problem we can also continue replaying the trace on the model and record all situations where a transition is forced to fire without being enabled, i.e., we count all missing tokens. Moreover, we record the tokens that remain at the end.

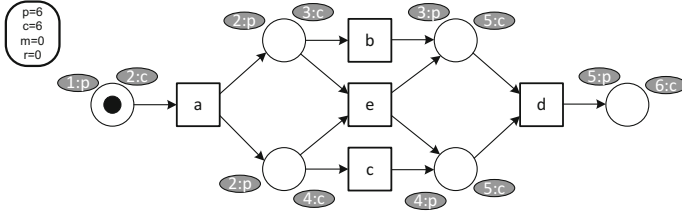


Fig. 11. Replaying trace $\sigma_1 = \langle a, b, c, d \rangle$ on the system net shown in Fig. 8: $fitness(\sigma_1) = \frac{1}{2}(1 - \frac{0}{6}) + \frac{1}{2}(1 - \frac{0}{6}) = 1$. (Place and transition identifiers are not shown, only the transition labels are depicted.)

To explain the idea, we first replay $\sigma_1 = \langle a, b, c, d \rangle$ on the system net shown in Fig. 8. We use four counters: p (produced tokens), c (consumed tokens), m (missing tokens), and r (remaining tokens). Initially, $p = c = 0$ and all places are empty. Then the environment produces a token to create the initial marking. Therefore, the p counter is incremented: $p = 1$ (Step 1 in Fig. 11). Now we need to fire transition a first. This is possible. Since a consumes one token and produces two tokens, the c counter is incremented by 1 and the p counter is incremented by 2 (Step 2 in Fig. 11). Therefore, $p = 3$ and $c = 1$ after firing transition a . Then we replay the second event (b). Firing transition b results in $p = 4$ and $c = 2$ (Step 3 in Fig. 11). After replaying the third event (i.e. c) $p = 5$ and $c = 3$. They we replay d . Since d consumes two tokens and produces one, the result is $p = 6$ and $c = 5$ (Step 5 in Fig. 11). At the end, the environment consumes a token from the sink place (Step 6 in Fig. 11). Hence the final result is $p = c = 6$ and $m = r = 0$. Clearly, there are no problems when replaying the σ_1 , i.e., there are no missing or remaining tokens ($m = r = 0$).

The fitness of trace σ is defined as follows:

$$fitness(\sigma) = \frac{1}{2} \left(1 - \frac{m}{c} \right) + \frac{1}{2} \left(1 - \frac{r}{p} \right)$$

The first part computes the fraction of missing tokens relative to the number of consumed tokens. $1 - \frac{m}{c} = 1$ if there are no missing tokens ($m = 0$) and $1 - \frac{m}{c} = 0$ if all tokens to be consumed were missing ($m = c$). Similarly, $1 - \frac{r}{p} = 1$ if there are no remaining tokens and $1 - \frac{r}{p} = 0$ if none of the produced tokens was actually consumed. We use an equal penalty for missing and remaining tokens. By definition: $0 \leq \text{fitness}(\sigma) \leq 1$. In our example, $\text{fitness}(\sigma_1) = \frac{1}{2}(1 - \frac{0}{6}) + \frac{1}{2}(1 - \frac{0}{6}) = 1$ because there are no missing or remaining tokens.

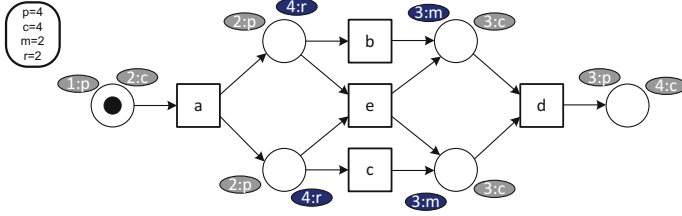


Fig. 12. Replaying trace $\sigma_2 = \langle a, d \rangle$ on the system net shown in Fig. 8: $\text{fitness}(\sigma_2) = \frac{1}{2}(1 - \frac{2}{4}) + \frac{1}{2}(1 - \frac{2}{4}) = 0.5$.

Let us now consider a trace that cannot be replayed properly. Figure 12 shows the process of replaying $\sigma_2 = \langle a, d \rangle$. Initially, $p = c = 0$ and all places are empty. Then the environment produces a token for the initial marking and the p counter is updated: $p = 1$. The first event (a) can be replayed (Step 2 in Fig. 12). After firing a , we have $p = 3$, $c = 1$, $m = 0$, and $r = 0$. Now we try to replay the second event. This is not possible, because transition d is not enabled. To fire d , we need to add a token to each of the input places of d and record the two missing tokens (Step 3 in Fig. 12). The m counter is incremented. The p and c counter are updated as usual. Therefore, after firing d , we have $p = 4$, $c = 3$, $m = 2$, and $r = 0$. At the end, the environment consumes a token from the sink place (Step 4 in Fig. 12). Moreover, we note the two remaining tokens on the output places of a . Hence the final result is $p = c = 4$ and $m = r = 2$. Figure 12 shows diagnostic information that helps to understand the nature of non-conformance. There was a situation in which d occurred but could not happen according to the model (m -tags) and there was a situation in which b and c or e were supposed to happen but did not occur according to the log (r -tags). Moreover, we can compute the fitness of trace σ_2 based on the values of p , c , m , and r : $\text{fitness}(\sigma_2) = \frac{1}{2}(1 - \frac{2}{4}) + \frac{1}{2}(1 - \frac{2}{4}) = 0.5$.

Figures 11 and 12 illustrate how to analyze the fitness of a single case. The same approach can be used to analyze the fitness of a log consisting of many cases. Simply take the sums of all produced, consumed, missing, and remaining tokens, and apply the same formula. Let p_σ denote the number of produced tokens when replaying σ on N . c_σ , m_σ , r_σ are defined in a similar fashion, e.g., m_σ is the number of missing tokens when replaying σ . Now we can define the

fitness of an event log L on a given system net:

$$\text{fitness}(L) = \frac{1}{2} \left(1 - \frac{\sum_{\sigma \in L} L(\sigma) \times m_{\sigma}}{\sum_{\sigma \in L} L(\sigma) \times c_{\sigma}} \right) + \frac{1}{2} \left(1 - \frac{\sum_{\sigma \in L} L(\sigma) \times r_{\sigma}}{\sum_{\sigma \in L} L(\sigma) \times p_{\sigma}} \right)$$

By replaying the entire event log, we can now compute the fitness of event log $L_4 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^3, \langle a, e, d \rangle^2, \langle a, d \rangle, \langle a, e, e, d \rangle]$ for the system net shown in Fig. 8. The total number of produced tokens is $p = 3 \cdot 6 + 3 \cdot 6 + 2 \cdot 6 + 1 \cdot 4 + 1 \cdot 8 = 60$. There are also $c = 60$ consumed tokens. The number of missing tokens is $m = 3 \cdot 0 + 3 \cdot 0 + 2 \cdot 0 + 1 \cdot 2 + 1 \cdot 2 = 4$. There are also $r = 4$ remaining tokens. Hence, $\text{fitness}(L_4) = \frac{1}{2} \left(1 - \frac{4}{60} \right) + \frac{1}{2} \left(1 - \frac{4}{60} \right) = 0.933$.

Typically, the event-based fitness is higher than the naïve case-based fitness. This is also the case here. The system net in Fig. 8 can only replay 80 % of the cases from start to end. However, about 93 % of the individual events can be replayed. For more information on token-based replay we refer to [2, 79].

An event log can be split into two sublogs: *one event log containing only fitting cases and one event log containing only non-fitting cases*. Each of the event logs can be used for further analysis. For example, one could construct a process model for the event log containing only deviating cases. Also other data and process mining techniques can be used, e.g., one can use classification techniques to further investigate non-conformance.

5.3 Aligning Observed and Modeled Behavior

There are various ways to quantify fitness [2, 9, 22, 52, 65, 68, 69, 79]. The simple procedure of counting missing, remaining, produced, and consumed tokens has several limitations. For example, in case of multiple transitions with the same label or transitions that are invisible, there are all kinds of complications. Which path to take if multiple transitions with the same label are enabled? Moreover, in case of poor fitness the Petri net is flooded with tokens thus resulting in optimistic estimates (many transitions are enabled). The notion of *cost-based alignments* [9, 22] provides a more robust and flexible approach for conformance checking.

To measure fitness, we *align* traces in the event log to traces of the process model. Consider the following three alignments for the traces in $L_1 = [\langle a, c, d, f, g \rangle^{10}, \langle a, c, d, h \rangle^5, \langle a, b, c, d, e, c, d, g, f \rangle^5]$ and the system net in Fig. 5:

$$\begin{aligned} \gamma_1 &= \begin{array}{c|c|c|c|c|c|c|c|c|c|} a & c & \gg & d & \gg & f & g & \gg & & \\ \hline a & c & \tau & d & \tau & f & g & \tau & & \\ \hline t1 & t4 & t2 & t5 & t7 & t8 & t9 & t11 & & \end{array} & \gamma_2 &= \begin{array}{c|c|c|c|c|c|} a & c & \gg & d & h & \\ \hline a & c & \tau & d & h & \\ \hline t1 & t4 & t2 & t5 & t10 & \end{array} \\ \\ \gamma_3 &= \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|} a & b & c & d & e & c & \gg & d & \gg & g & f & \gg & \\ \hline a & b & c & d & e & c & \tau & d & \tau & g & f & \tau & \\ \hline t1 & t3 & t4 & t5 & t6 & t4 & t2 & t5 & t7 & t9 & t8 & t11 & \end{array} \end{aligned}$$

The top row of each alignment corresponds to “moves in the log” and the bottom two rows correspond to “moves in the model”. Moves in the model are represented by the transition and its label. This is needed because there could be

multiple transitions with the same label. In alignment γ_1 the first column refers to a “move in both”, i.e., both the event log and the process model make an a move. If a move in the model cannot be mimicked by a move in the log, then a “ \gg ” (“no move”) appears in the top row. This situation is referred to as a “move in model”. For example, in the third position of γ_1 the log cannot mimic the invisible transition $t2$. The τ above $t2$ indicates that $t2 \notin \text{dom}(l)$. In the remainder, we write $l(t) = \tau$ if $t \notin \text{dom}(l)$. Note that all “no moves” (i.e., the seven \gg symbols) in $\gamma_1 - \gamma_3$ are “caused” by invisible transitions.

Let us now consider some example alignments for the deviating event log $L'_1 = [\langle a, c, d, f \rangle^{10}, \langle a, c, d, c, h \rangle^5, \langle a, b, d, e, c, d, g, f, h \rangle^5]$ and system net SN in Fig. 5:

$$\gamma_4 = \begin{array}{c|c|c|c|c|c|c|c|c|c|} a & c & \gg & d & \gg & f & \gg & \gg & & \\ \hline a & c & \tau & d & \tau & f & g & \tau & & \\ \hline t1 & t4 & t2 & t5 & t7 & t8 & t9 & t11 & & \end{array} \quad \gamma_5 = \begin{array}{c|c|c|c|c|c|c|} a & c & \gg & d & c & h & \\ \hline a & c & \tau & d & \gg & h & \\ \hline t1 & t4 & t2 & t5 & & t10 & \end{array}$$

$$\gamma_6 = \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|} a & b & \gg & d & e & c & \gg & d & \gg & g & f & \gg & h & \\ \hline a & b & c & d & e & c & \tau & d & \tau & g & f & \tau & \gg & \\ \hline t1 & t3 & t4 & t5 & t6 & t4 & t2 & t5 & t7 & t9 & t8 & t11 & & \end{array}$$

Alignment γ_4 shows a “ \gg ” (“no move”) in the top row that does not correspond to an invisible transition. The model makes a g move (occurrence of transition $t9$) that is not in the log. Alignment γ_6 has a similar move in the third position: the model makes a c move (occurrence of transition $t4$) that is not in the log. If a move in the log cannot be mimicked by a move in the model, then a “ \gg ” (“no move”) appears in the bottom row. This situation is referred to as a “move in log”. For example, in γ_5 the c move in the log is not mimicked by a move in the model and in γ_6 the h move in the log is not mimicked by a move in the model. Note that the “no moves” not corresponding to invisible transitions point to deviations between model and log.

A *move* is a pair $(x, (y, t))$ where the first element refers to the log and the second element refers to the model. For example, $(a, (a, t1))$ means that both log and model make an “ a move” and the move in the model is caused by the occurrence of transition $t1$. $(\gg, (g, t9))$ means that the occurrence of transition $t9$ with label g is not mimicked by corresponding move of the log. (c, \gg) means that the log makes an “ c move” not followed by the model.

Definition 14 (Legal Moves). Let $L \in \mathcal{B}(A^*)$ be an event log and let $SN = (N, M_{init}, M_{final}) \in \mathcal{U}_{SN}$ be a system net with $N = (P, T, F, l)$. $A_{LM} = \{(x, (x, t)) \mid x \in A \wedge t \in T \wedge l(t) = x\} \cup \{(\gg, (x, t)) \mid t \in T \wedge l(t) = x\} \cup \{(x, \gg) \mid x \in A\}$ is the set of legal moves.

An alignment is a sequence of legal moves such that after removing all \gg symbols, the top row corresponds to a trace in the log and the bottom row corresponds to a firing sequence starting in M_{init} and ending M_{final} . Hence, the middle row corresponds to a visible path when ignoring the τ steps.

Definition 15 (Alignment). Let $\sigma_L \in L$ be a log trace and $\sigma_M \in \phi_f(SN)$ a complete firing sequence of system net SN . An alignment of σ_L and σ_M is a sequence $\gamma \in A_{LM}^*$ such that the projection on the first element (ignoring \gg) yields σ_L and the projection on the last element (ignoring \gg and transition labels) yields σ_M .

γ_1 – γ_3 are examples of alignments for the traces in L_1 and their corresponding firing sequences in the system net of Fig. 5. γ_4 – γ_6 are examples of alignments for the traces in L'_1 and complete firing sequences of the same system net. The projection of γ_6 on the first element (ignoring \gg) yields $\sigma_L = \langle a, b, d, e, c, d, g, f, h \rangle$ which is indeed a trace in L'_1 . The projection of γ_6 on the last element (ignoring \gg and transition labels) yields $\sigma_M = \langle t1, t3, t4, t5, t6, t4, t2, t5, t7, t9, t8, t11 \rangle$ which is indeed a complete firing sequence. The projection of γ_6 on the middle element (i.e., transition labels while ignoring \gg and τ) yields $\langle a, b, c, d, e, c, d, g, f \rangle$ which is indeed a visible trace of the system net of Fig. 5.

Given a log trace and a process model there may be many (if not infinitely many) alignments. Consider the following two alignments for $\langle a, c, d, f \rangle \in L'_1$:

$$\gamma_4 = \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline a & c & \gg & d & \gg & f & \gg & \gg & \\ \hline a & c & \tau & d & \tau & f & g & \tau & \\ \hline t1 & t4 & t2 & t5 & t7 & t8 & t9 & t11 & \\ \hline \end{array} \quad \gamma'_4 = \begin{array}{|c|c|c|c|c|c|c|c|} \hline a & c & \gg & d & \gg & f & \gg & \\ \hline a & c & b & d & \tau & \gg & h & \\ \hline t1 & t4 & t3 & t5 & t7 & & t10 & \\ \hline \end{array}$$

γ_4 seems to be better alignment than γ'_4 because it has only one deviation (move in model only; $(\gg, (g, t9))$) whereas γ'_4 has three deviations: $(\gg, (b, t3))$, (f, \gg) , and $(\gg, (h, t11))$. To select the most appropriate one we associate *costs* to undesirable moves and select an alignment with the lowest total costs. To quantify the costs of misalignments we introduce a cost function δ .

Definition 16 (Cost of Alignment). Cost function $\delta \in A_{LM} \rightarrow \mathbb{N}$ assigns costs to legal moves. The cost of an alignment $\gamma \in A_{LM}^*$ is the sum of all costs: $\delta(\gamma) = \sum_{(x,y) \in \gamma} \delta(x, y)$.

Moves where log and model agree have no costs, i.e., $\delta(x, (x, t)) = 0$ for all $x \in A$. Moves in model only have no costs if the transition is invisible, i.e., $\delta(\gg, (\tau, t)) = 0$ if $l(t) = \tau$. $\delta(\gg, (x, t)) > 0$ is the cost when the model makes an “ x move” without a corresponding move of the log (assuming $l(t) = x \neq \tau$). $\delta(x, \gg) > 0$ is the cost for an “ x move” in just the log. These costs may depend on the nature of the activity, e.g., skipping a payment may be more severe than sending too many letters. However, in this paper we often use a standard cost function δ_S that assigns unit costs: $\delta_S(x, (x, t)) = 0$, $\delta_S(\gg, (\tau, t)) = 0$, and $\delta_S(\gg, (x, t)) = \delta_S(x, \gg) = 1$ for all $x \in A$. For example, $\delta_S(\gamma_1) = \delta_S(\gamma_2) = \delta_S(\gamma_3) = 0$, $\delta_S(\gamma_4) = 1$, $\delta_S(\gamma_5) = 1$, and $\delta_S(\gamma_6) = 2$ (simply count the number of \gg symbols not corresponding to invisible transitions). Now we can compare the two alignments for $\langle a, c, d, f \rangle \in L'_1$: $\delta_S(\gamma_4) = 1$ and $\delta_S(\gamma'_4) = 3$. Hence, we conclude that γ_4 is “better” than γ'_4 .

Definition 17 (Optimal Alignment). Let $L \in \mathcal{B}(A^*)$ be an event log with $A \subseteq \mathcal{U}_A$ and let $SN \in \mathcal{U}_{SN}$ be a system net with $\phi(SN) \neq \emptyset$.

- For $\sigma_L \in L$, we define: $\Gamma_{\sigma_L, SN} = \{\gamma \in A_{LM}^* \mid \exists_{\sigma_M \in \phi_f(SN)} \gamma \text{ is an alignment of } \sigma_L \text{ and } \sigma_M\}$.
- An alignment $\gamma \in \Gamma_{\sigma_L, SN}$ is optimal for trace $\sigma_L \in L$ and system net SN if for any $\gamma' \in \Gamma_{\sigma_L, SN}$: $\delta(\gamma') \geq \delta(\gamma)$.
- $\lambda_{SN} \in A^* \rightarrow A_{LM}^*$ is a deterministic mapping that assigns any log trace σ_L to an optimal alignment, i.e., $\lambda_{SN}(\sigma_L) \in \Gamma_{\sigma_L, SN}$ and $\lambda_{SN}(\sigma_L)$ is optimal.
- $costs(L, SN, \delta) = \sum_{\sigma_L \in L} \delta(\lambda_{SN}(\sigma_L))$ are the misalignment costs of the whole event log.

γ_1 – γ_6 are optimal alignments for the corresponding six possible traces in event logs L_1 and L'_1 and the system net in Fig. 5. γ'_4 is not an optimal alignment for $\langle a, c, d, f \rangle$. $costs(L_1, SN, \delta_S) = 10 \times \delta_S(\gamma_1) + 5 \times \delta_S(\gamma_2) + 5 \times \delta_S(\gamma_3) = 10 \times 0 + 5 \times 0 + 5 \times 0 = 0$. Hence, L_1 is perfectly fitting system net SN . $costs(L'_1, SN, \delta_S) = 10 \times \delta_S(\gamma_4) + 5 \times \delta_S(\gamma_5) + 5 \times \delta_S(\gamma_6) = 10 \times 1 + 5 \times 1 + 5 \times 2 = 25$.

It is possible to convert misalignment costs into a fitness value between 0 (poor fitness, i.e., maximal costs) and 1 (perfect fitness, zero costs). We refer to [9, 22] for details.

Only perfectly fitting traces have costs 0 (assuming $\phi(SN) \neq \emptyset$). Hence, Event log L is perfectly fitting system net SN if and only if $costs(L, SN, \delta) = 0$.

Once an optimal alignment has been established for every trace in the event log, these alignments can also be used as a basis to quantify other conformance notations such as precision and generalization [9]. For example, precision can be computed by counting “escaping edges” as shown in [68, 69]. Recent results show that such computations should be based on alignments [24]. The same holds for generalization [9]. Therefore, we focus on alignments when decomposing conformance checking problems in Sect. 6.

5.4 Beyond Conformance Checking

The importance of alignments cannot be overstated. Alignments relate observed behavior with modeled behavior. This is not only important for conformance checking, but also for enriching and repairing models. For example, timestamps in the event log can be used to analyze bottlenecks in the process model. In fact, partial alignments can also be used to predict problems and to recommend appropriate actions. This is illustrated by Fig. 13. See [2, 9] for concrete examples.

6 Decomposing Process Mining Problems

The torrents of event data available are an important enabler for process mining. However, the incredible growth of event data also provides computational challenges. For example, conformance checking can be time consuming as potentially many different traces need to be aligned with a model that may allow for an exponential (or even infinite) number of traces. Event logs may contain millions of events. Finding the best alignment may require solving many optimization problems [22] or repeated state-space explorations [79]. In worst case

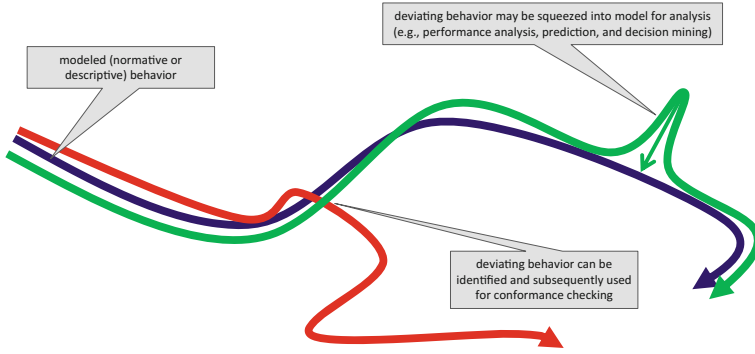


Fig. 13. The essence of process mining: relating modeled and observed behavior.

a state-space exploration of the model is needed per event. When using genetic process mining, one needs to check the fitness of every individual model in every generation [30, 65]. As a result, thousands or even millions of conformance checks need to be done. For each conformance check, the whole event log needs to be traversed. Given these challenges, we are interested in reducing the time needed for conformance checking by decomposing the associated Petri net and event log. See [3, 4, 7] for an overview of various decomposition approaches. For example, in [4] we discuss the *vertical partitioning* and *horizontal partitioning* of event logs.

Event logs are composed of cases. There may be thousands or even millions of cases. In case of vertical partitioning these can be distributed over the nodes in the network, i.e., each case is assigned to one computing node. All nodes work on a subset of the whole log and in the end the results need to be merged.

Cases are composed of multiple events. We can also partition cases, i.e., part of a case is analyzed on one node whereas another part of the same case is analyzed on another node. This corresponds to a horizontal partitioning of the event log. In principle, each node needs to consider all cases. However, the attention of one computing node is limited to a particular subset of events per case.

Even when only one computing node is available, it may still be beneficial to decompose process mining problems. Due to the exponential nature of most conformance checking techniques, the time needed to solve “many smaller problems” is less than the time needed to solve “one big problem”. In the remainder, we only consider the so-called horizontal partitioning of the event log.

6.1 Decomposing Conformance Checking

To decompose conformance checking problems we split a process model into model fragments. In terms of Petri nets: the overall system net SN is decomposed into a collection of subnets $\{SN^1, SN^2, \dots, SN^n\}$ such that the union of these subnets yields the original system net. The union of two system nets is defined as follows.

Definition 18 (Union of Nets). Let $SN^1 = (N^1, M_{init}^1, M_{final}^1) \in \mathcal{U}_{SN}$ with $N^1 = (P^1, T^1, F^1, l^1)$ and $SN^2 = (N^2, M_{init}^2, M_{final}^2) \in \mathcal{U}_{SN}$ with $N^2 = (P^2, T^2, F^2, l^2)$ be two system nets.

- $l^3 \in (T^1 \cup T^2) \not\vdash \mathcal{U}_A$ with $\text{dom}(l^3) = \text{dom}(l^1) \cup \text{dom}(l^2)$, $l^3(t) = l^1(t)$ if $t \in \text{dom}(l^1)$, and $l^3(t) = l^2(t)$ if $t \in \text{dom}(l^2) \setminus \text{dom}(l^1)$ is the union of l^1 and l^2 ,
- $N^1 \cup N^2 = (P^1 \cup P^2, T^1 \cup T^2, F^1 \cup F^2, l^3)$ is the union of N^1 and N^2 , and
- $SN^1 \cup SN^2 = (N^1 \cup N^2, M_{init}^1 \uplus M_{init}^2, M_{final}^1 \uplus M_{final}^2)$ is the union of system nets SN^1 and SN^2 .

Using Definition 18, we can check whether the union of a collection of subnets $\{SN^1, SN^2, \dots, SN^n\}$ indeed corresponds to the overall system net SN . It suffices to check whether $SN = \bigcup_{1 \leq i \leq n} SN^i = SN^1 \cup SN^2 \cup \dots \cup SN^n$. A decomposition $\{SN^1, SN^2, \dots, SN^n\}$ is *valid* if the subnets “agree” on the original labeling function (i.e., the same transition always has the same label), each place resides in just one subnet, and also each invisible transition resides in just one subnet. Moreover, if there are multiple transitions with the same label, they should reside in the same subnet. Only unique visible transitions (i.e., $T_v^u(SN)$, cf. Definition 8) can be shared among different subnets.

Definition 19 (Valid Decomposition). Let $SN \in \mathcal{U}_{SN}$ be a system net with labeling function l . $D = \{SN^1, SN^2, \dots, SN^n\} \subseteq \mathcal{U}_{SN}$ is a *valid decomposition* if and only if

- $SN^i = (N^i, M_{init}^i, M_{final}^i)$ is a system net with $N^i = (P^i, T^i, F^i, l^i)$ for all $1 \leq i \leq n$,
- $l^i = l \upharpoonright_{T^i}$ for all $1 \leq i \leq n$,
- $P^i \cap P^j = \emptyset$ for $1 \leq i < j \leq n$,
- $T^i \cap T^j \subseteq T_v^u(SN)$ for $1 \leq i < j \leq n$, and
- $SN = \bigcup_{1 \leq i \leq n} SN^i$.

$\mathcal{D}(SN)$ is the set of all valid decompositions of SN .

Every system net has a trivial decomposition consisting of only one subnet, i.e., $\{SN\} \in \mathcal{D}(SN)$. However, we are often interested in a *maximal decomposition* where the individual subnets are as small as possible. Figure 14 shows the maximal decomposition for the system net shown in Fig. 5.

In [7] it is shown that a unique maximal valid decomposition always exists. Moreover, it is possible to decompose nets based on the notion of *passages* [3] or using Single-Entry Single-Exit (SESE) components [70]. In the remainder, we assume a valid decomposition without making any further assumptions.

Next, we show that conformance checking can be done by locally inspecting the subnets using correspondingly projected event logs. To illustrate this, consider the following alignment for trace $\langle a, b, c, d, e, c, d, g, f \rangle$ and the system net in Fig. 5:

$$\gamma_3 = \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ \hline a & b & c & d & e & c & \gg & d & \gg & g & f & \gg \\ \hline a & b & c & d & e & c & \tau & d & \tau & g & f & \tau \\ \hline t1 & t3 & t4 & t5 & t6 & t4 & t2 & t5 & t7 & t9 & t8 & t11 \end{array}$$

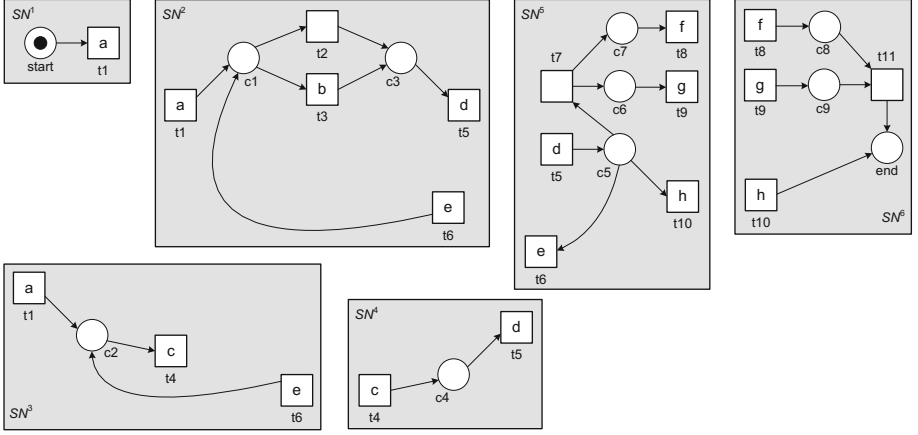


Fig. 14. Maximal decomposition of the system net shown in Fig. 5 with $M_{init} = [start]$ and $M_{final} = [end]$. The initial and final markings are as follows: $M_{init}^1 = [start]$ and $M_{init}^i = []$ for $2 \leq i \leq 6$, $M_{final}^i = []$ for $1 \leq i \leq 5$, and $M_{final}^6 = [end]$.

For convenience, the moves have been numbered. Now consider the following six alignments:

$$\begin{aligned}
 \gamma_3^1 &= \begin{array}{|c|} \hline 1 \\ \hline a \\ \hline a \\ \hline t1 \\ \hline \end{array} &
 \gamma_3^2 &= \begin{array}{|c|c|c|c|c|c|} \hline 1 & 2 & 4 & 5 & 7 & 8 \\ \hline a & b & d & e & \gg & d \\ \hline a & b & d & e & \tau & d \\ \hline t1 & t3 & t5 & t6 & t2 & t5 \\ \hline \end{array} &
 \gamma_3^3 &= \begin{array}{|c|c|c|c|} \hline 1 & 3 & 5 & 6 \\ \hline a & c & e & c \\ \hline a & c & e & c \\ \hline t1 & t4 & t6 & t4 \\ \hline \end{array} \\
 \gamma_3^4 &= \begin{array}{|c|c|c|c|} \hline 3 & 4 & 6 & 8 \\ \hline c & d & c & d \\ \hline c & d & c & d \\ \hline t4 & t5 & t4 & t5 \\ \hline \end{array} &
 \gamma_3^5 &= \begin{array}{|c|c|c|c|c|c|} \hline 4 & 5 & 8 & 9 & 10 & 11 \\ \hline d & e & d & \gg & g & f \\ \hline d & e & d & \tau & g & f \\ \hline t5 & t6 & t5 & t7 & t9 & t8 \\ \hline \end{array} &
 \gamma_3^6 &= \begin{array}{|c|c|c|} \hline 10 & 11 & 12 \\ \hline g & f & \gg \\ \hline g & f & \tau \\ \hline t9 & t8 & t11 \\ \hline \end{array}
 \end{aligned}$$

Each alignment corresponds to one of the six subnets SN^1, SN^2, \dots, SN^6 in Fig. 14. The numbers are used to relate the different alignments. For example γ_3^6 is an alignment for trace $\langle a, b, c, d, e, c, d, g, f \rangle$ and subnets SN^6 in Fig. 14. As the numbers 10, 11 and 12 indicate, γ_3^6 corresponds to the last three moves of γ_3 .

To create sublogs for the different model fragments, we use the projection function introduced in Sect. 3. Consider for example the overall log $L_1 = [\langle a, c, d, f, g \rangle^{10}, \langle a, c, d, h \rangle^5, \langle a, b, c, d, e, c, d, g, f \rangle^5]$. $L_1^1 = L_1 \upharpoonright_{\{a\}} = [\langle a \rangle^{20}]$, $L_1^2 = L_1 \upharpoonright_{\{a, b, d, e\}} = [\langle a, d \rangle^{15}, \langle a, b, d, e, d \rangle^5]$, $L_1^3 = L_1 \upharpoonright_{\{a, c, e\}} = [\langle a, c \rangle^{15}, \langle a, c, e, c \rangle^5]$, etc. are the sublogs corresponding to the subnets in Fig. 14.

The following theorem shows that any trace that fits the overall process model can be decomposed into smaller traces that fit the individual model fragments. Moreover, if the smaller traces fit the individual model fragments, then they can be composed into an overall trace that fits into the overall process model. This result is the basis for decomposing a wide range of process mining problems.

Theorem 1 (Conformance Checking Can be Decomposed). *Let $L \in \mathcal{B}(A^*)$ be an event log with $A \subseteq \mathcal{U}_A$ and let $SN \in \mathcal{U}_{SN}$ be a system net. For any valid decomposition $D = \{SN^1, SN^2, \dots, SN^n\} \in \mathcal{D}(SN)$: L is perfectly fitting system net SN if and only if for all $1 \leq i \leq n$: $L \upharpoonright_{A_v(SN^i)}$ is perfectly fitting SN^i .*

Proof. See [7]. □

Theorem 1 shows that any trace in the log fits the overall model if and only if it fits each of the subnets.

Let us now consider trace $\langle a, b, d, e, c, d, g, f, h \rangle$ which is not perfectly fitting the system net in Fig. 5. An optimal alignment is:

$$\gamma_6 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ \hline a & b & \gg & d & e & c & \gg & d & \gg & g & f & \gg & h \\ \hline a & b & c & d & e & c & \tau & d & \tau & g & f & \tau & \gg \\ \hline t1 & t3 & t4 & t5 & t6 & t4 & t2 & t5 & t7 & t9 & t8 & t11 & \\ \hline \end{array}$$

The alignment shows the two problems: the model needs to execute c whereas this event is not in the event log (position 3) and the event log contains g , f , and h whereas the model needs to choose between either g and f or h (position 13). The cost of this optimal alignment is 2. Optimal alignment γ_6 for the overall model can be decomposed into alignments $\gamma_6^1 - \gamma_6^6$ for the six subnets:

$$\begin{array}{l} \gamma_6^1 = \begin{array}{|c|} \hline 1 \\ \hline a \\ \hline a \\ \hline t1 \\ \hline \end{array} \quad \gamma_6^2 = \begin{array}{|c|c|c|c|c|c|} \hline 1 & 2 & 4 & 5 & 7 & 8 \\ \hline a & b & d & e & \gg & d \\ \hline a & b & d & e & \tau & d \\ \hline t1 & t3 & t5 & t6 & t2 & t5 \\ \hline \end{array} \quad \gamma_6^3 = \begin{array}{|c|c|c|c|} \hline 1 & 3 & 5 & 6 \\ \hline a & \gg & e & c \\ \hline a & c & e & c \\ \hline t1 & t4 & t6 & t4 \\ \hline \end{array} \\ \\ \gamma_6^4 = \begin{array}{|c|c|c|c|} \hline 3 & 4 & 6 & 8 \\ \hline \gg & d & c & d \\ \hline c & d & c & d \\ \hline t4 & t5 & t4 & t5 \\ \hline \end{array} \quad \gamma_6^5 = \begin{array}{|c|c|c|c|c|c|c|} \hline 4 & 5 & 8 & 9 & 10 & 11 & 13 \\ \hline d & e & d & \gg & g & f & h \\ \hline d & e & d & \tau & g & f & \gg \\ \hline t5 & t6 & t5 & t7 & t9 & t8 & \\ \hline \end{array} \quad \gamma_6^6 = \begin{array}{|c|c|c|c|} \hline 10 & 11 & 12 & 13 \\ \hline g & f & \gg & h \\ \hline g & f & \tau & \gg \\ \hline t9 & t8 & t11 & \\ \hline \end{array} \end{array}$$

Alignments γ_6^1 and γ_6^2 have costs 0. Alignments γ_6^3 and γ_6^4 have costs 1 (move in model involving c). Alignments γ_6^5 and γ_6^6 have costs 1 (move in log involving h). If we would add up all costs, we would get costs 4 whereas the costs of optimal alignment γ_6 is 2. However, we would like to compute an upper bound for the degree of fitness in a distributed manner. Therefore, we introduce an adapted cost function δ_Q .

Definition 20 (Adapted Cost Function). *Let $D = \{SN^1, SN^2, \dots, SN^n\} \in \mathcal{D}(SN)$ be a valid decomposition of some system net SN and $\delta \in A_{LM} \rightarrow \mathbb{N}$ a cost function (cf. Definition 16). $c_Q(a, (a, t)) = c_Q(\gg, (a, t)) = c_Q(a, \gg) = |\{1 \leq i \leq n \mid a \in A_i\}|$ counts the number of subnets having a as an observable activity. The adapted cost function δ_Q is defined as follows: $\delta_Q(x, y) = \frac{\delta(x, y)}{c_Q(x, y)}$ for $(x, y) \in A_{LM}$ and $c_Q(x, y) \neq 0$.*

An observable activity may appear in multiple subnets. Therefore, we divide its costs by the number of subnets in which it appears: $\delta_Q(x, y) = \frac{\delta(x, y)}{c_Q(x, y)}$. This way we avoid counting misalignments of the same activity multiple times. For our example, $c_Q(\gg, (c, t4)) = |\{3, 4\}| = 2$ and $c_Q(h, \gg) = |\{5, 6\}| = 2$. Assuming the standard cost function δ_S this implies $\delta_Q(\gg, (c, t4)) = \frac{1}{2}$ and $\delta_Q(h, \gg) = \frac{1}{2}$. Hence the aggregated costs of $\gamma_6^1 - \gamma_6^6$ are 2, i.e., identical to the costs of the overall optimal alignment.

Theorem 2 (Lower Bound for Misalignment Costs). *Let $L \in \mathcal{B}(A^*)$ be an event log with $A \subseteq \mathcal{U}_A$, $SN \in \mathcal{U}_{SN}$ be a system net, and δ a cost function. For any valid decomposition $D = \{SN^1, SN^2, \dots, SN^n\} \in \mathcal{D}(SN)$:*

$$\text{costs}(L, SN, \delta) \geq \sum_{1 \leq i \leq n} \text{costs}(L \upharpoonright_{A_v(SN^i)}, SN^i, \delta_Q)$$

Proof. See [7]. □

The sum of the costs associated to all selected optimal local alignments (using δ_Q) can never exceed the cost of an optimal overall alignment using δ . Hence, it can be used as an optimistic estimate, i.e., computing an upper bound for the overall fitness and a lower bound for the overall costs. More important, the fitness values of the different subnets provide valuable local diagnostics. *The subnets with the highest costs are the most problematic parts of the model. The alignments for these “problem spots” help to understand the main problems without having to look at very long overall alignments.*

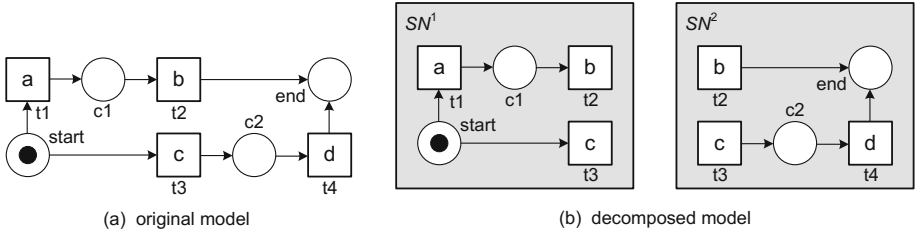


Fig. 15. Example showing that the total misalignment costs may be higher than the costs associated to the two optimal local alignments.

Theorem 2 only provides a lower bound for the misalignment costs. The total misalignment costs may be higher than the costs associated to the optimal local alignments. To understand this, consider the following optimal alignments for trace $\langle a, b, c, d \rangle$ and the (decomposed) process model shown in Fig. 15:

$$\gamma = \begin{array}{|c|c|c|c|} \hline a & b & c & d \\ \hline a & b & \gg & \gg \\ \hline t1 & t2 & & \\ \hline \end{array} \quad \gamma' = \begin{array}{|c|c|c|c|} \hline a & b & c & d \\ \hline \gg & \gg & c & d \\ \hline & & t3 & t4 \\ \hline \end{array} \quad \gamma_1 = \begin{array}{|c|c|c|} \hline a & b & c \\ \hline a & b & \gg \\ \hline t1 & t2 & \\ \hline \end{array} \quad \gamma_2 = \begin{array}{|c|c|c|} \hline b & c & d \\ \hline \gg & c & d \\ \hline & t3 & t4 \\ \hline \end{array}$$

There are two optimal alignments (γ and γ') for $\langle a, b, c, d \rangle$ and the overall system net shown in Fig. 15(a). Both optimal alignments (γ and γ') have two moves in log only (i.e., these events cannot be mimicked by the model). Hence, $\delta_S(\gamma) = \delta_S(\gamma') = 2$. Now consider the decomposition shown in Fig. 15(b). The cost of the optimal alignment γ_1 for subnet SN^1 is $\delta_Q(\gamma_1) = 0 + 0 + \delta_Q(c, \gg) = \frac{\delta_S(c, \gg)}{c_Q(c, \gg)} = \frac{1}{2} = 0.5$. The cost of the optimal alignment γ_2 for subnet SN^2 is $\delta_Q(\gamma_2) = \delta_Q(b, \gg) + 0 + 0 = \frac{\delta_S(b, \gg)}{c_Q(b, \gg)} = \frac{1}{2} = 0.5$. Since $\delta_Q(\gamma_1) + \delta_Q(\gamma_2) = 1$ and $\delta_S(\gamma) = 2$, we can observe the misalignment costs for γ are indeed higher than the costs associated to the two optimal local alignments (γ_1 and γ_2). This is caused by the fact that the two optimal local alignments don't agree on the moves with respect to activities b and c . γ_1 suggests a move in both for activity b and a move in model for c . γ_2 makes a different choice and suggests a move in both for activity c and a move in model for d . Therefore, γ_1 and γ_2 cannot be stitched back into an overall alignment with costs 1. Practical experiences show that the difference between $costs(L, SN, \delta)$ and $sum_{1 \leq i \leq n} costs(L \upharpoonright_{A_v(SN^i)}, SN^i, \delta_Q)$ increases when the fragments are getting smaller. Hence, there is tradeoff between the accuracy of the lower bound and the degree of decomposition. If the subnets are chosen large enough, accuracy tends to be quite good.

Theorem 2 uses a rather sophisticated definition of fitness. We can also simply count the *fraction of fitting traces*. In this case the problem can be decomposed easily using the notion of valid decomposition.

Corollary 1 (Fraction of Perfectly Fitting Traces). *Let $L \in \mathcal{B}(A^*)$ be an event log with $A \subseteq \mathcal{U}_A$ and let $SN \in \mathcal{U}_{SN}$ be a system net. For any valid decomposition $D = \{SN^1, SN^2, \dots, SN^n\} \in \mathcal{D}(SN)$:*

$$\frac{|\{\sigma \in L \mid \sigma \in \phi(SN)\}|}{|L|} = \frac{|\{\sigma \in L \mid \forall 1 \leq i \leq n \ \sigma \upharpoonright_{A_v(SN^i)} \in \phi(SN^i)\}|}{|L|}$$

The corollary follows directly from the construction used in Theorem 1. A trace is fitting the overall model if and only if it fits all subnets. As Corollary 1 suggests, traces in the event log can be marked as fitting or non-fitting *per subnet*. These results can be merged easily and used to compute the *fraction of traces fitting the overall model*. Note that Corollary 1 holds for any decomposition of SN .

6.2 Decomposing Process Discovery

Process discovery, i.e., discovering a process model from event data, is highly related to conformance checking. This can be observed when considering genetic process mining algorithms that basically “guess” models and recombine parts of models that have good fitness to discover even better models [30, 65]. The fitness computation in genetic process mining is in essence a conformance check.

Using Theorem 1 we can distribute *any* process discovery algorithm by (1) *decomposing the overall event log into smaller sublogs*, (2) *discovering a model fragment for each sublog*, and (3) *merging the discovered models into one overall process model*.

Given an event log L containing events referring to a set of activities A , we decompose discovery by distributing the activities over multiple sets A^1, A^2, \dots, A^n . The same activity may appear in multiple sets as long as $A = A^1 \cup A^2 \cup \dots \cup A^n$. For each activity set A_i , we discover a system net \overline{SN}^i by applying a discovery algorithm to sublog $L \upharpoonright_{A_i}$, i.e., the overall event log projected onto a subset of activities. Subsequently, the discovered system nets are massaged to avoid name clashes and to make sure that transitions with a visible label are merged properly. By combining the resulting system nets we obtain an overall system net $SN = SN^1 \cup SN^2 \cup \dots \cup SN^n$ describing the behavior in the overall event log L [7].

The quality of the system net obtained by merging the process models discovered for the sublogs highly depends on the way the activities are decomposed and the characteristics of the discovery algorithm used per sublog. However, the system nets discovered for the sublogs are always a valid decomposition of the overall model. This implies that we can apply Theorem 1 (Conformance Checking Can be Decomposed), Theorem 2 (Lower Bound for Misalignment Costs), and Corollary 1 (Fraction of Perfectly Fitting Traces). If the discovery algorithm is able to create a perfectly fitting model for each sublog, then the overall model is also perfectly fitting. Moreover, if the discovery algorithm has problems finding a perfectly fitting model for a particular sublog, then the overall model will also exhibit these problems. For example, the fraction of traces fitting the overall model equals the fraction of traces fitting all individual models.

6.3 Decomposition Strategies

The decomposition results for conformance checking and process discovery are not tailored towards a particular type of decomposition. Recently, several papers have been published on different ways of decomposing process mining problems. In [3, 88, 89] it is shown that so-called “passages” can be used to decompose both process discovery and conformance checking problems. In [70, 71] it is shown that so-called SESE (Single-Exit-Single-Entry) components obtained through the Refined Process Structure Tree (RPST) [74, 87] can be used to decompose conformance checking problems. These papers use a particular decomposition strategy. However, as shown in [7], there are many ways to decompose process mining problems.

The above papers are all using Petri nets as a representation. However, as shown in [5] the essence of the decomposition results is not limited to Petri nets at all.

Experimental results shows that significant speed-ups are possible through decomposition [70, 88]. Process mining algorithms are typically linear in the number of cases and exponential in the average length of traces or the number of unique activities. Through a vertical partitioning [4] many process mining algorithms can be decomposed trivially. Consider for example conformance checking problems. These are solved per case. Hence, by distributing the cases over different computing nodes it is easy to realize a linear speedup. Discovery algorithms

often use some variant of the “directly follows” relation ($>_L$) discussed in the context of the α -algorithm. Obviously a vertical partitioning (e.g. using a Map-Reduce [40, 75] programming style) can be used to create such a relation in a distributed fashion.

In this paper we focused on a horizontal partitioning of the event log because process mining algorithms tend to be exponential in the average length of traces or the number of unique activities. Hence, the potential gains of horizontal partitioning are much larger. Just like the state space of a Petri net may grow exponentially in the number of transitions, the search space may grow rapidly as the number of different activities increases. Hence, the time needed to solve “many smaller problems” is often less than the time needed to solve “one big problem”, even when this is done sequentially. In fact, horizontal partitioning may lead to super linear speedups. Consider for example conformance checking approaches that use state-space analysis (e.g., in [79] the shortest path enabling a transition is computed) or optimization over all possible alignments (e.g., in [22] the A^* algorithm is used to find the best alignment). These techniques do *not* scale linearly in the number of activities. Therefore, decomposition is often useful even if the checks per subnet are done on a single computer. Moreover, decomposing conformance checking is not just interesting from a performance point of view: decompositions can also be used to pinpoint the most problematic parts of the process and provide localized diagnostics [70]. Decompositions are not just interesting for conformance diagnostics; also performance-related diagnostics (e.g., bottleneck analysis) benefit from a hierarchical structuring of the whole process.

7 Conclusion

The torrents of event data available in most organizations enable *evidence-based Business Process Management* (ebBPM). We predict that there will be a remarkable shift from pure model-driven or questionnaire-driven approaches to data-driven process analysis as we are able to monitor and reconstruct the real business processes using event data. At the same time, we expect that machine learning and data mining approaches will become more *process-centric*. Thus far, the machine learning and data mining communities have not been focusing on end-to-end processes that also exhibit concurrency. Hence, it is time to move beyond decision trees, clustering, and (association) rules.

Process mining can be used to diagnose the actual processes. This is valuable because in many organizations most stakeholders lack a correct, objective, and accurate view on important operational processes. Process mining can subsequently be used to improve such processes. Conformance checking can be used for auditing and compliance. By replaying the event log on a process model it is possible to quantify and visualize deviations. Similar techniques can be used to detect bottlenecks and build predictive models. Given the applicability of process mining, we hope that this tutorial encourages the reader to start using process mining today. The book [2] provides a comprehensive introduction into the process mining field. Moreover, the open source process mining

tool *ProM* can be downloaded from www.processmining.org. Many of the ideas developed in the context of *ProM* have been embedded in commercial tools such as Fluxicon's Disco (www.fluxicon.com), Perceptive Process Mining (www.perceptivesoftware.com), Celonis (www.celonis.de), and QPR ProcessAnalyzer (www.qpr.com). This illustrates the practical relevance of process mining.

In the last part of this tutorial we discussed some very general decomposition results. Clearly, highly scalable analysis approaches are needed to deal with the ever-growing amounts of event data. This requires additional research efforts. Moreover, we refer to the *Process Mining Manifesto* by the IEEE Task Force on Process Mining [57] for additional challenges in this exciting new research field.

Acknowledgements. This work was supported by the Basic Research Program of the National Research University Higher School of Economics (HSE).

References

1. van der Aalst, W.M.P.: Formalization and verification of event-driven process chains. *Inf. Softw. Technol.* **41**(10), 639–650 (1999)
2. van der Aalst, W.M.P.: *Process Mining: Discovery Conformance and Enhancement of Business Processes*. Springer, Berlin (2011)
3. van der Aalst, W.M.P.: Decomposing process mining problems using passages. In: Haddad, S., Pomello, L. (eds.) *PETRI NETS 2012*. LNCS, vol. 7347, pp. 72–91. Springer, Heidelberg (2012)
4. van der Aalst, W.M.P.: Distributed process discovery and conformance checking. In: de Lara, J., Zisman, A. (eds.) *FASE 2012*. LNCS, vol. 7212, pp. 1–25. Springer, Heidelberg (2012)
5. van der Aalst, W.M.P.: A general divide and conquer approach for process mining. In: Ganzha, M., Maciaszek, L., Paprzycki, M. (eds.) *Federated Conference on Computer Science and Information Systems (FedCSIS 2013)*, pp. 1–10. IEEE Computer Society (2013)
6. van der Aalst, W.M.P.: Business process management: a comprehensive survey. *ISRN Softw. Eng.* 1–37 (2013). doi:[10.1155/2013/507984](https://doi.org/10.1155/2013/507984)
7. van der Aalst, W.M.P.: Decomposing Petri nets for process mining: a generic approach. *Distrib. Parallel Databases* **31**(4), 471–507 (2013)
8. van der Aalst, W.M.P.: Mediating between modeled and observed behavior: the quest for the “Right” process. In: *IEEE International Conference on Research Challenges in Information Science (RCIS 2013)*, pp. 31–43. IEEE Computing Society (2013)
9. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. *WIREs Data Min. Knowl. Disc.* **2**(2), 182–192 (2012)
10. van der Aalst, W., Adriansyah, A., van Dongen, B.: Causal nets: a modeling language tailored towards process discovery. In: Katoen, J.-P., König, B. (eds.) *CONCUR 2011*. LNCS, vol. 6901, pp. 28–42. Springer, Heidelberg (2011)
11. van der Aalst, W.M.P., Basten, T.: Identifying commonalities and differences in object life cycles using behavioral inheritance. In: Colom, J.-M., Koutny, M. (eds.) *ICATPN 2001*. LNCS, vol. 2075, pp. 32–52. Springer, Heidelberg (2001)

12. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: a survey of issues and approaches. *Data Knowl. Eng.* **47**(2), 237–267 (2003)
13. van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H.M.W., Voorhoeve, M., Wynn, M.T.: Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects Comput.* **23**(3), 333–363 (2011)
14. van der Aalst, W.M.P., van Hee, K.M., van der Werf, J.M., Verdonk, M.: Auditing 2.0: using process mining to support tomorrow’s auditor. *IEEE Comput.* **43**(3), 90–93 (2010)
15. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. *Distribut. Parallel Databases* **14**(1), 5–51 (2003)
16. van der Aalst, W.M.P., Lassen, K.B.: Translating unstructured workflow processes to readable BPEL: theory and implementation. *Inf. Softw. Technol.* **50**(3), 131–159 (2008)
17. van der Aalst, W.M.P., Pesic, M., Song, M.: Beyond process mining: from the past to present and future. In: Pernici, B. (ed.) *CAISE 2010. LNCS*, vol. 6051, pp. 38–52. Springer, Heidelberg (2010)
18. van der Aalst, W.M.P., Reijers, H.A., Weijters, A.J.M.M., van Dongen, B.F., Alves de Medeiros, A.K., Song, M., Verbeek, H.M.W.: Business process mining: an industrial application. *Inf. Syst.* **32**(5), 713–732 (2007)
19. van der Aalst, W.M.P., Rubin, V., Verbeek, H.M.W., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. *Softw. Syst. Model.* **9**(1), 87–111 (2010)
20. van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. *Inf. Syst.* **36**(2), 450–475 (2011)
21. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow mining: discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.* **16**(9), 1128–1142 (2004)
22. Adriansyah, A., van Dongen, B., van der Aalst, W.M.P.: Conformance checking using cost-based fitness analysis. In: Chi, C.H., Johnson, P. (eds.) *IEEE International Enterprise Computing Conference (EDOC 2011)*, pp. 55–64. IEEE Computer Society (2011)
23. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Towards robust conformance checking. In: zur Muehlen, M., Su, J. (eds.) *BPM 2010 Workshops. LNBIP*, vol. 66, pp. 122–133. Springer, Heidelberg (2011)
24. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.P.: Alignment based precision checking. In: La Rosa, M., Soffer, P. (eds.) *BPM Workshops 2012. LNBIP*, vol. 132, pp. 137–149. Springer, Heidelberg (2013)
25. Adriansyah, A., Sidorova, N., van Dongen, B.F.: Cost-based fitness in conformance checking. In: *International Conference on Application of Concurrency to System Design (ACSD 2011)*, pp. 57–66. IEEE Computer Society (2011)
26. Agrawal, R., Gunopulos, D., Leymann, F.: Mining process models from workflow logs. In: Schek, H.-J., Saltorè, F., Ramos, I., Alonso, G. (eds.) *EDBT 1998. LNCS*, vol. 1377, pp. 469–483. Springer, Heidelberg (1998)
27. Badouel, E., Darondeau, P.: Theory of regions. In: Reisig, W., Rozenberg, G. (eds.) *APN 1998. LNCS*, vol. 1491, pp. 529–586. Springer, Heidelberg (1998)
28. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) *BPM 2007. LNCS*, vol. 4714, pp. 375–383. Springer, Heidelberg (2007)
29. Chandra Bose, R.P.J.C.: Process mining in the large: preprocessing, discovery, and diagnostics. Ph.D. thesis, Eindhoven University of Technology (2012)

30. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: On the role of fitness, precision, generalization and simplicity in process discovery. In: Meersman, R., et al. (eds.) OTM 2012, Part I. LNCS, vol. 7565, pp. 305–322. Springer, Heidelberg (2012)
31. Calders, T., Guenther, C., Pechenizkiy, M., Rozinat, A.: Using minimum description length for process mining. In: ACM Symposium on Applied Computing (SAC 2009), pp. 1451–1455. ACM Press (2009)
32. Carmona, J., Cortadella, J.: Process mining meets abstract interpretation. In: Balcázar, J., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML PKDD 2010, Part I. LNCS, vol. 6321, pp. 184–199. Springer, Heidelberg (2010)
33. 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)
34. Cook, J.E., Wolf, A.L.: Discovering models of software processes from event-based data. *ACM Trans. Software Eng. Methodol.* **7**(3), 215–249 (1998)
35. Cook, J.E., Wolf, A.L.: Software process validation: quantitatively measuring the correspondence of a process to a model. *ACM Trans. Software Eng. Methodol.* **8**(2), 147–176 (1999)
36. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving Petri nets from finite transition systems. *IEEE Trans. Comput.* **47**(8), 859–882 (1998)
37. Curbera, F., Doganata, Y., Martens, A., Mukhi, N.K., Slominski, A.: Business provenance - a technology to increase traceability of end-to-end operations. In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part I. LNCS, vol. 5331, pp. 100–119. Springer, Heidelberg (2008)
38. Datta, A.: Automating the discovery of As-Is business process models: probabilistic and algorithmic approaches. *Inf. Syst. Res.* **9**(3), 275–301 (1998)
39. Davidson, S., Cohen-Boulakia, S., Eyal, A., Ludaescher, B., McPhillips, T., Bowers, S., Anand, M., Freire, J.: Provenance in scientific workflow systems. *Data Eng. Bull.* **30**(4), 44–50 (2007)
40. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. *Commun. ACM* **51**(1), 107–113 (2008)
41. Depaire, B., Swinnen, J., Jans, M., Vanhoof, K.: A process deviation analysis framework. In: La Rosa, M., Soffer, P. (eds.) BPM 2012 Workshops. LNBIP, vol. 132, pp. 701–706. Springer, Heidelberg (2013)
42. Dijkman, R., Dumas, M., van Dongen, B., Käärik, R., Mendling, J.: Similarity of business process models: metrics and evaluation. *Inf. Syst.* **36**(2), 498–516 (2011)
43. Dijkman, R., Dumas, M., García-Bañuelos, L.: Graph matching algorithms for business process model similarity search. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 48–63. Springer, Heidelberg (2009)
44. van Dongen, B.F., van der Aalst, W.M.P.: Multi-phase process mining: building instance graphs. In: Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T.-W. (eds.) ER 2004. LNCS, vol. 3288, pp. 362–376. Springer, Heidelberg (2004)
45. van Dongen, B.F., van der Aalst, W.M.P.: Multi-phase mining: aggregating instances graphs into EPCs and Petri nets. In: Marinescu, D. (ed.) Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management, pp. 35–58. Florida International University, Miami (2005)
46. van Dongen, B.F., Crooy, R.A., van der Aalst, W.M.P.: Cycle time prediction: When will this case finally be finished? In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part I. LNCS, vol. 5331, pp. 319–336. Springer, Heidelberg (2008)

47. van Dongen, B.F., Alves de Medeiros, A.K., Wen, L.: Process mining: overview and outlook of Petri net discovery algorithms. In: Jensen, K., van der Aalst, W.M.P. (eds.) *ToPNoC II*. LNCS, vol. 5460, pp. 225–242. Springer, Heidelberg (2009)
48. Ehrenfeucht, A., Rozenberg, G.: Partial (Set) 2-Structures - Part 1 and Part 2. *Acta Inf.* **27**(4), 315–368 (1989)
49. Fahland, D., van der Aalst, W.M.P.: Repairing process models to reflect reality. In: Barros, A., Gal, A., Kindler, E. (eds.) *BPM 2012*. LNCS, vol. 7481, pp. 229–245. Springer, Heidelberg (2012)
50. Gaaloul, W., Gaaloul, K., Bhiri, S., Haller, A., Hauswirth, M.: Log-based transactional workflow mining. *Distrib. Parallel Databases* **25**(3), 193–240 (2009)
51. van Glabbeek, R.J., Weijland, W.P.: Branching time and abstraction in bisimulation semantics. *J. ACM* **43**(3), 555–600 (1996)
52. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events. *J. Mach. Learn. Res.* **10**, 1305–1340 (2009)
53. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) *BPM 2007*. LNCS, vol. 4714, pp. 328–343. Springer, Heidelberg (2007)
54. Herbst, J.: A machine learning approach to workflow management. In: Lopez de Mantaras, R., Plaza, E. (eds.) *ECML 2000*. LNCS (LNAI), vol. 1810, pp. 183–194. Springer, Heidelberg (2000)
55. Hilbert, M., Lopez, P.: The world's technological capacity to store, communicate, and compute information. *Science* **332**(6025), 60–65 (2011)
56. Hinz, S., Schmidt, K., Stahl, C.: Transforming BPEL to Petri nets. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) *BPM 2005*. LNCS, vol. 3649, pp. 220–235. Springer, Heidelberg (2005)
57. van der Aalst, W., et al.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) *BPM 2011 Workshops, Part I*. LNBIP, vol. 99, pp. 169–194. Springer, Heidelberg (2012)
58. Jin, T., Wang, J., Wen, L.: Efficient retrieval of similar business process models based on structure. In: Meersman, R., et al. (eds.) *OTM 2011, Part I*. LNCS, vol. 7044, pp. 56–63. Springer, Heidelberg (2011)
59. Jin, T., Wang, J., Wen, L.: Efficient retrieval of similar workflow models based on behavior. In: Sheng, Q.Z., Wang, G., Jensen, C.S., Xu, G. (eds.) *APWeb 2012*. LNCS, vol. 7235, pp. 677–684. Springer, Heidelberg (2012)
60. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting BPEL processes. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) *BPM 2006*. LNCS, vol. 4102, pp. 17–32. Springer, Heidelberg (2006)
61. Ludaescher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger-Frank, E., Jones, M., Lee, E., Tao, J., Zhao, Y.: Scientific workflow management and the Kepler system. *Concurrency Comput. Pract. Experience* **18**(10), 1039–1065 (2006)
62. Maggi, F.M., Montali, M., van der Aalst, W.M.P.: An operational decision support framework for monitoring business constraints. In: de Lara, J., Zisman, A. (eds.) *FASE 2012*. LNCS, vol. 7212, pp. 146–162. Springer, Heidelberg (2012)
63. Maggi, F.M., Westergaard, M., Montali, M., van der Aalst, W.M.P.: Runtime verification of LTL-based declarative process models. In: Khurshid, S., Sen, K. (eds.) *RV 2011*. LNCS, vol. 7186, pp. 131–146. Springer, Heidelberg (2012)
64. Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., Byers, A.: Big data: the next frontier for innovation, competition, and productivity. McKinsey Global Institute (2011)

65. Alves de Medeiros, A.K., Weijters, A.J.M.M., van der Aalst, W.M.P.: Genetic process mining: an experimental evaluation. *Data Min. Knowl. Disc.* **14**(2), 245–304 (2007)
66. Mendling, J., van Dongen, B.F., van der Aalst, W.M.P.: On the degree of behavioral similarity between business process models. In: Nuettgens, M., Rump, F.J., Gadatsch, A. (eds.) *Proceedings of Sixth Workshop on Event-Driven Process Chains (WI-EPK 2007)*, St. Augustin, November 2007, pp. 39–58. Gesellschaft für Informatik, Bonn (2007)
67. Milner, R.: *Communication and Concurrency*. Prentice-Hall Inc., Upper Saddle River (1989)
68. Munoz-Gama, J., Carmona, J.: A fresh look at precision in process conformance. In: Hull, R., Mendling, J., Tai, S. (eds.) *BPM 2010. LNCS*, vol. 6336, pp. 211–226. Springer, Heidelberg (2010)
69. Munoz-Gama, J., Carmona, J.: Enhancing precision in process conformance: stability, confidence and severity. In: Chawla, N., King, I., Sperduti, A. (eds.) *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011)*, Paris, France, April 2011, pp. 184–191. IEEE (2011)
70. Munoz-Gama, J., Carmona, J., van der Aalst, W.M.P.: Conformance checking in the large: partitioning and topology. In: Daniel, F., Wang, J., Weber, B. (eds.) *BPM 2013. LNCS*, vol. 8094, pp. 130–145. Springer, Heidelberg (2013)
71. Munoz-Gama, J., Carmona, J., van der Aalst, W.M.P.: Hierarchical conformance checking of process models based on event logs. In: Colom, J.-M., Desel, J. (eds.) *PETRI NETS 2013. LNCS*, vol. 7927, pp. 291–310. Springer, Heidelberg (2013)
72. Ouyang, C., van der Aalst, W.M.P., Breutel, S., Dumas, M., ter Hofstede, A.H.M., Verbeek, H.M.W.: Formal semantics and analysis of control flow in WS-BPEL. *Sci. Comput. Program.* **67**(2–3), 162–198 (2007)
73. Ouyang, C., Dumas, M., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Pattern-based translation of BPMN process models to BPEL web services. *Int. J. Web Serv. Res.* **5**(1), 42–62 (2007)
74. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified computation and generalization of the refined process structure tree. In: Bravetti, M., Bultan, T. (eds.) *WS-FM 2010. LNCS*, vol. 6551, pp. 25–41. Springer, Heidelberg (2011)
75. Rajaraman, A., Ullman, J.D.: *Mining of Massive Datasets*. Cambridge University Press, Cambridge (2011)
76. Reijers, H.A.: Case prediction in BPM systems: a research challenge. *J. Korean Inst. Ind. Eng.* **33**, 1–10 (2006)
77. Reisig, W.: *Petri Nets: Modeling Techniques, Analysis, Methods, Case Studies*. Springer, Heidelberg (2013)
78. Rozinat, A., van der Aalst, W.M.P.: Decision mining in ProM. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) *BPM 2006. LNCS*, vol. 4102, pp. 420–425. Springer, Heidelberg (2006)
79. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. *Inf. Syst.* **33**(1), 64–95 (2008)
80. Rozinat, A., Mans, R.S., Song, M., van der Aalst, W.M.P.: Discovering colored Petri nets from event logs. *Int. J. Softw. Tools Technol. Transfer* **10**(1), 57–74 (2008)
81. Rozinat, A., Mans, R.S., Song, M., van der Aalst, W.M.P.: Discovering simulation models. *Inf. Syst.* **34**(3), 305–327 (2009)
82. Rozinat, A., Wynn, M., van der Aalst, W.M.P., ter Hofstede, A.H.M., Fidge, C.: Workflow simulation for operational decision support. *Data Knowl. Eng.* **68**(9), 834–850 (2009)

83. Schonenberg, H., Weber, B., van Dongen, B.F., van der Aalst, W.M.P.: Supporting flexible processes through recommendations based on history. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 51–66. Springer, Heidelberg (2008)
84. Sheth, A.: A new landscape for distributed and parallel data management. *Distrib. Parallel Databases* **30**(2), 101–103 (2012)
85. Solé, M., Carmona, J.: Process mining from a basis of state regions. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 226–245. Springer, Heidelberg (2010)
86. Staffware. Staffware Process Suite Version 2 - White Paper. Staffware PLC, Maidenhead, UK (2003)
87. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. *Data Knowl. Eng.* **68**(9), 793–818 (2009)
88. Verbeek, H.M.W., van der Aalst, W.M.P.: An experimental evaluation of passage-based process discovery. In: La Rosa, M., Soffer, P. (eds.) BPM 2012 Workshops. LNBIP, vol. 132, pp. 205–210. Springer, Heidelberg (2013)
89. Verbeek, H.M.W., van der Aalst, W.M.P.: Decomposing replay problems: a case study. BPM Center Report BPM-13-09. www.bpmcenter.org (2013)
90. De Weerd, J., De Backer, M., Vanthienen, J., Baesens, B.: A robust F-measure for evaluating discovered process models. In: Chawla, N., King, I., Sperduti, A. (eds.) IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011), Paris, France, pp. 148–155. IEEE (2011)
91. Weidlich, M., Dijkman, R.M., Weske, M.: Behavior equivalence and compatibility of business process models with complex correspondences. *Comput. J.* **55**(11), 1398–1418 (2012)
92. Weijters, A.J.M.M., van der Aalst, W.M.P.: Rediscovering workflow models from event-based data using little thumb. *Integr. Comput. Aided Eng.* **10**(2), 151–162 (2003)
93. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. *Fundamenta Inf.* **94**, 387–412 (2010)

Business Intelligence

Third European Summer School, eBISS 2013, Dagstuhl

Castle, Germany, July 7-12, 2013, Tutorial Lectures

Zimányi, E. (Ed.)

2014, IX, 243 p. 95 illus., Softcover

ISBN: 978-3-319-05460-5