Process Discovery: Capturing the Invisible

Wil M. P. van der Aalst

Department of Mathematics and Computer Science,
Technische Universiteit Eindhoven, The Netherlands.
W.M.P.v.d.Aalst@tue.nl

Abstract. Processes are everywhere. Organizations have business processes to manufacture products, provide services, purchase goods, handle applications, etc. Also in our daily lives we are involved in a variety of processes, for example when we use our car or when we book a trip via the Internet. Although such operational processes are omnipresent, they are at the same time intangible. Unlike a product or a piece of data, processes are less concrete because of their dynamic nature. However, more and more information about these processes is captured in the form of event logs. Contemporary systems ranging from copiers and medical devices to enterprise information systems and cloud infrastructures record events. These events can be used to make processes visible. Using process mining techniques it is possible to discover processes. This provides the insights necessary to manage, control, and improve processes. Process mining has been successfully applied in a variety of domains ranging from healthcare and e-business to high-tech systems and auditing. Despite these successes, there are still many challenges as process discovery shows that the real processes are more "spaghetti-like" than people like to think. It is still very difficult to capture the complex reality in a suitable model. Given the nature of these challenges, techniques originating from Computational Intelligence may assist in the discovery of complex processes.

1 Introduction

Process mining is a relative young research discipline that sits between machine learning and data mining on the one hand and process modeling and analysis on the other hand [3]. Process mining includes the automated discovery of processes from event logs. Based on observed events (e.g., activities being executed or messages being exchanged) a process model is constructed. Since most processes are highly concurrent and activities may have complex dependencies, simple techniques such as sequence mining [8] are unable to capture the underlying process adequately.

Process mining aims at the analysis of operational processes that are repeatable. If there are only few instances following the same process, it is typically impossible to reconstruct a reasonable process model. Examples of operational processes are the treatment of patients having a particular type of cancer, handling customer orders by a furniture factory, booking business trips within a

consultancy firm, opening new bank accounts, handling insurance claims, repairing high-end copiers, etc. These processes are instantiated for specific cases. Examples of cases (i.e., process instances) are patient treatments, customer orders, business trips, insurance claims, etc. Although both organizations and individuals are constantly involved in a multitude of processes, little attention is given to these processes. Of course people talk about processes and parts of these processes are supported by IT systems. Some organizations even document their processes using a variety of notations. However, a detailed analysis of what actually happens is typically missing.

Unlike products, processes are less tangible. Processes may only exist in the minds of people and it is difficult to "materialize processes". One could argue that "information" is similar to "processes" in this respect. In fact, some people like to distinguish between data, information, and knowledge. Nevertheless, in many cases it is possible to "print" information elements and view these as products. For example, it is possible to print all information in an electronic patient record. For processes this is more difficult. Processes may emerge from human behavior or may be partly controlled by procedures and/or information systems. Many processes are documented in the form of normative or descriptive models. However, these models are human-made artifacts that do not necessarily say much about the actual process. As a result, there may be a disconnect between model and reality. Moreover, the people involved in a process are typically unable to understand the corresponding process model and cannot easily see what is actually going on (especially if the products are services or data).

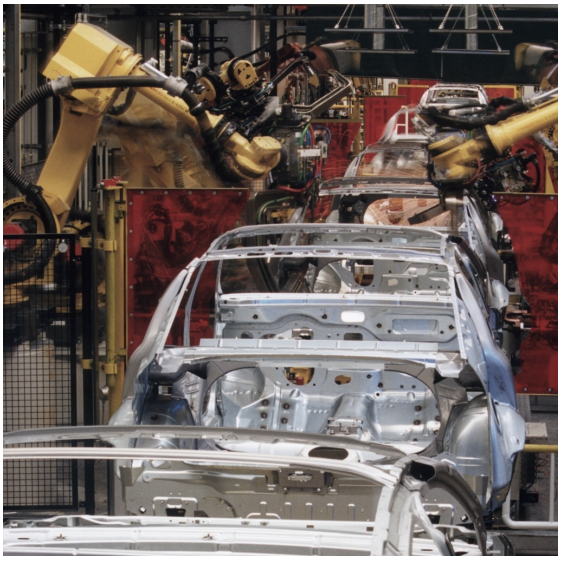

Fig. 1. One of the few of today's processes that can easily be observed

To illustrate this consider the production line shown in Figure 1. This is one of the few processes that can easily be observed. The products are physical entities rather than information or services. Moreover, the routing of work is partly defined by the physical layout of the production line. Most of today's processes do not have such a physical representation. Think for example of a hospital, a municipality, or an insurance company where processes are much more dynamic and driven by people rather than machines. In such less tangible processes there is a strong desire to visualize the actual flow of work.

Fortunately, more and more information about processes is being recorded in the form of event logs. Today's information systems log enormous amounts of events. Classical workflow management systems (e.g. FileNet), ERP systems (e.g. SAP), case handling systems (e.g. FLOWer), PDM systems (e.g. Windchill), CRM systems (e.g. Microsoft Dynamics CRM), middleware (e.g., IBM's WebSphere), hospital information systems (e.g., Chipsoft), etc. provide very detailed information about the activities that have been executed [3]. Moreover, more and more devices are connected to the Internet today, thus allowing for unprecedented streams of data. This "data explosion" is an important enabler for process mining.

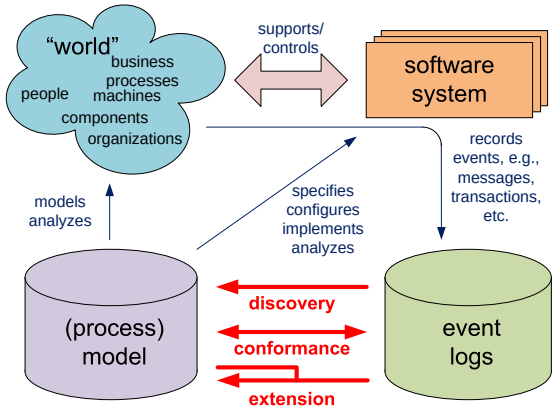

Fig. 2. Three types of process mining: (1) Discovery, (2) Conformance, and (3) Extension

The idea of process mining is to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs. We consider three basic types of process mining (Figure 2):

– *Discovery*: There is no a-priori model, i.e., based on an event log some model is constructed. For example, using the α-algorithm [6] a process model can be discovered based on low-level events.

– *Conformance*: There is an a-priori model. This model is used to check if reality, as recorded in the log, conforms to the model and vice versa. For example, there may be a process model indicating that purchase orders of more than one million Euro require two checks. Another example is the checking of the four-eyes principle. Conformance checking may be used to detect deviations, to locate and explain these deviations, and to measure the severity of these deviations. An example, is the conformance checking algorithm described in [28].
– *Extension*: There is an a-priori model. This model is extended with a new aspect or perspective, i.e., the goal is not to check conformance but to enrich the model. An example is the extension of a process model with performance data, i.e., some a-priori process model is used on which bottlenecks are projected. Another example is the decision mining algorithm described in [27] that extends a given process model with conditions for each decision.

Orthogonal to the three types of mining, there are at least three perspectives. The *control-flow perspective* focuses on the control-flow, i.e., the ordering of activities. The goal of mining this perspective is to find a good characterization of all possible paths, e.g., expressed in terms of a Petri net or some other notation (e.g., EPCs, BPMN, UML ADs, etc.). The *organizational perspective* focuses on information about resources hidden in the log, i.e., which performers are involved and how are they related. The goal is to either structure the organization by classifying people in terms of roles and organizational units or to show the social network. The *case perspective* focuses on properties of cases. Cases can be characterized by their path in the process or by the originators working on a case. However, cases can also be characterized by the values of the corresponding data elements. For example, if a case represents a replenishment order, it may be interesting to know the supplier or the number of products ordered.

This article will focus on process discovery, i.e., distilling a model from an event log describing the control-flow. First, we present the basic problem. Then we present a concrete algorithm. The algorithm is able to capture concurrency, but has problems dealing with less structured models. Subsequently, we list the challenges in process discovery. Finally, we describe some typical applications and an outlook on process mining as a grand challenge for computational intelligence.

2 Process Discovery

In this section, we describe the goal of process discovery. In order to do this, we present a particular format for logging events and a particular process modeling language (Petri nets). Based on this we sketch various process discovery approaches.

2.1 Event Logs

The goal of process mining is to extract knowledge about a particular (operational) process from event logs, i.e., process mining describes a family of *a-posteriori* analysis techniques exploiting the information recorded in audit trails,

transaction logs, databases, etc. Typically, these approaches assume that it is possible to *sequentially record events* such that each event refers to an *activity* (i.e., a well-defined step in the process) and is related to a particular *case* (i.e., a process instance). Furthermore, some mining techniques use additional information such as the performer or *originator* of the event (i.e., the person / resource executing or initiating the activity), the *timestamp* of the event, or *data elements* recorded with the event (e.g., the size of an order).

To clarify the notion of an event log consider Table 1 which shows a fragment of some event log. Only two traces are shown, both containing four events. Each event has a unique id and several properties. For example event 35654423 belongs to case $x123$ and is an instance of activity a that occurred on December 30th at 11.02, was executed by John, and costed 300 euros. The second trace (case $x128$) starts with event 35655526 and also refers to an instance of activity a. The

Table 1. A fragment of some event log.

case id	event id	properties				
		timestamp	activity	resource	cost	...
x123	35654423	30-12-2008:11.02	a	John	300	...
x123	35654424	30-12-2008:11.06	b	John	400	...
x123	35654425	30-12-2008:11.12	c	John	100	...
x123	35654426	30-12-2008:11.18	d	John	400	...
x128	35655526	30-12-2008:16.10	a	Ann	300	...
x128	35655527	30-12-2008:16.14	c	John	450	...
x128	35655528	30-12-2008:16.26	b	Pete	350	...
x128	35655529	30-12-2008:16.36	d	Ann	300	...
...	...	...	...	...	...	...

information depicted in Table 1 is the typical event data that can be extracted from today's systems.

Systems store events in very different ways. Process-aware information systems (e.g., workflow management systems) provide dedicated audit trails. In other systems, this information is typically scattered over several tables. For example, in a hospital events related to a particular patient may be stored in different tables and even different systems. For many applications of process mining, one needs to extract event data from different sources, merge these data, and convert the result into a suitable format. We advocate the use of the so-called MXML (Mining XML) format that can be read directly by ProM ([2]). Tools such as our ProM Import Framework allow developers to quickly implement plug-ins that can be used to extract information from a variety of systems and convert this into MXML. MXML is able to store the information shown in Table 1. Most of this information is optional, i.e., if it is there, it can be used for process mining, but it is not necessary for control-flow discovery.

Since we focus on control-flow discovery, we only consider the activity column in Table 1. This means that an event is linked to a case (process instance) and an activity, and no further attributes are needed. Events are ordered (per case), but do not need to have explicit timestamps. This allows us to use the following very simple definition of an event log.

Definition 1 (Event, Trace, Event log). *Let A be a set of activities. $\sigma \in A^*$ is a* trace*, i.e., a sequence of events. $L \in \mathbb{B}(A^*)$ is an* event log*, i.e., a multi-set of traces.*

The first four events in Table 1 form a trace $\langle a, b, c, d \rangle$. This trace represents the path followed by case $x123$. The second case ($x128$) can be represented by the trace $\langle a, c, b, d \rangle$. Note that there may be multiple cases that have the same trace. Therefore, an event log is defined as a *multi-set* of traces.

A *multi-set* (also referred to as *bag*) is like a set where each element may occur multiple times. For example, $[horse, cow^5, duck^2]$ is the multi-set with eight elements: one horse, five cows and two ducks. $\mathbb{B}(X)$ is the set of multi-sets (bags) over X. We assume the usual operators on multi-sets, e.g., $X \cup Y$ is the union of X and Y, $X \setminus Y$ is the difference between X and Y, $x \in X$ tests if x appears in X, and $X \leq Y$ evaluates to true if X is contained in Y. For example, $[horse, cow^2] \cup [horse^2, duck^2] = [horse^3, cow^2, duck^2]$, $[horse^3, cow^4] \setminus [cow^2] = [horse^3, cow^2]$, $[horse, cow^2] \leq [horse^2, cow^3]$, and $[horse^3, cow^1] \not\leq [horse^2, cow^2]$. Note that sets can be considered as bags having only one instance of every element. Hence, we can mix sets and bags, e.g., $\{horse, cow\} \cup [horse^2, cow^3] = [horse^3, cow^4]$.

In the remainder, we will use the following example log: $L_1 = [\langle a, b, c, d \rangle^5, \langle a, c, b, d \rangle^8, \langle a, e, d \rangle^9]$. L_1 contains information about 22 cases; five cases following trace $\langle a, b, c, d \rangle$, eight cases following trace $\langle a, c, b, d \rangle$, and nine cases following trace $\langle a, e, d \rangle$. Note that such a simple representation can be extracted from sources such as Table 1, MXML, or any other format that links events to cases and activities.

2.2 Petri Nets

The goal of process discovery is to distil a process model from some event log. Here we use *Petri nets* [26] to represent such models. In fact, we extract a subclass of Petri nets known as *workflow nets* (WF-nets) [1].

Definition 2. *An* Petri net *is a tuple (P, T, F) where:*

1. *P is a finite set of* places*,*
2. *T is a finite set of* transitions *such that $P \cap T = \emptyset$, and*
3. *$F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the* flow relation*.*

An example Petri net is shown in Figure 3. This Petri net has six places represented by circles and four transitions represented by squares. Places may contain tokens. For example, in Figure 3 both $p1$ and $p6$ contain one token, $p3$ contains two tokens, and the other places are empty. The state, also called

marking, is the distribution of tokens over places. A *marked* Petri net is a pair (N, M), where $N = (P, T, F)$ is a Petri net and where $M \in \mathbb{B}(P)$ is a bag over P denoting the *marking* of the net. The initial marking of the Petri net shown in Figure 3 is $[p1, p3^2, p6]$. The set of all marked Petri nets is denoted $\mathcal{N}$.

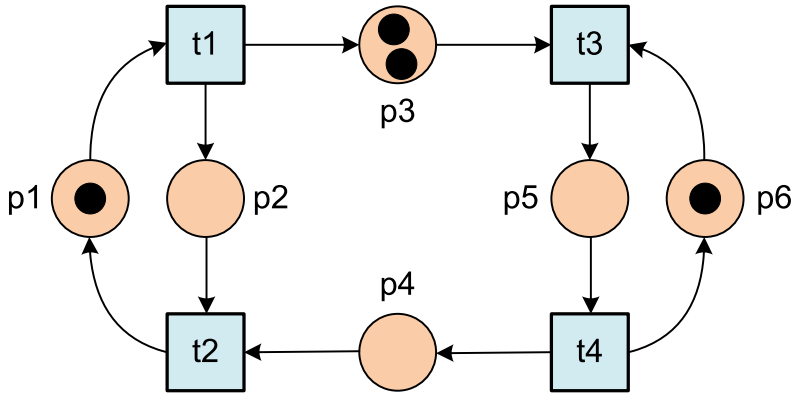

Fig. 3. A Petri net with six places ($p1$, $p2$, $p3$, $p4$, $p5$, and $p6$) and four transitions ($t1$, $t2$, $t3$, and $t4$)

Let $N = (P, T, F)$ be a Petri net. Elements of $P \cup T$ are called *nodes*. A node x is an *input node* of another node y iff there is a directed arc from x to y (i.e., $(x, y) \in F$). Node x is an *output node* of y iff $(y, x) \in F$. For any $x \in P \cup T$, $\bullet x = \{y \mid (y, x) \in F\}$ and $x\bullet = \{y \mid (x, y) \in F\}$. In Figure 3, $\bullet t3 = \{p3, p6\}$ and $t3\bullet = \{p5\}$.

The dynamic behavior of such a marked Petri net is defined by the so-called *firing rule*. A transition is *enabled* if each of its input places contains a token. An enabled transition can *fire* thereby consuming one token from each input place and producing one token for each output place.

Definition 3 (Firing rule). *Let (N, M) be a marked Petri net with $N = (P, T, F)$. Transition $t \in T$ is* enabled*, denoted $(N, M)[t\rangle$, iff $\bullet t \leq M$. The* firing rule *$\_\,[\_\rangle\,\_ \subseteq \mathcal{N} \times T \times \mathcal{N}$ is the smallest relation satisfying for any $(N, M) \in \mathcal{N}$ and any $t \in T$, $(N, M)[t\rangle \Rightarrow (N, M)\ [t\rangle\ (N, (M \setminus \bullet t) \cup t\bullet)$.*

In the marking shown in Figure 3, both $t1$ and $t3$ are enabled. The other two transitions are not enabled because at least one of the input places is empty. If $t1$ fires, one token is consumed (from $p1$) and two tokens are produced (one for $p2$ and one for $p3$). Formally, $(N, [p1, p3^2, p6])\ [t1\rangle\ (N, [p2, p3^3, p6])$. So the resulting marking is $[p2, p3^3, p6]$. If $t3$ fires in the initial state, two tokens are consumed (one from $p3$ and one from $p6$) and one token is produced (for $p5$). Formally, $(N, [p1, p3^2, p6])\ [t3\rangle\ (N, [p1, p3, p5])$.

Let (N, M_0) with $N = (P, T, F)$ be a marked P/T net. A sequence $\sigma \in T^*$ is called a *firing sequence* of (N, M_0) iff, for some natural number $n \in \mathbb{N}$, there exist markings $M_1, \ldots, M_n$ and transitions $t_1, \ldots, t_n \in T$ such that $\sigma = \langle t_1 \ldots t_n \rangle$ and, for all i with $0 \leq i < n$, $(N, M_i)[t_{i+1}\rangle$ and $(N, M_i)\,[t_{i+1}\rangle\,(N, M_{i+1})$.

Let (N, M_0) be the marked Petri net shown in Figure 3, i.e., $M_0 = [p1, p3^2, p6]$. The empty sequence $\sigma = \langle\ \rangle$ is enabled in (N, M_0). The sequence $\sigma = \langle t1, t3\rangle$ is also enabled and results in marking $[p2, p3^2, p5]$. Another possible firing sequence is $\sigma = \langle t3, t4, t3, t1, t4, t3, t2, t1\rangle$. A marking M is *reachable* from the initial marking M_0 iff there exists a sequence of enabled transitions whose firing leads from M_0 to M. The set of reachable markings of (N, M_0) is denoted $[N, M_0\rangle$.

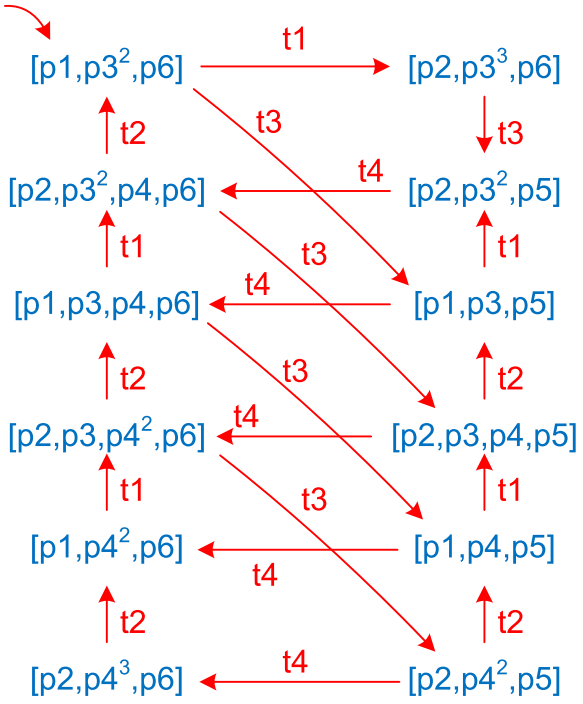

Fig. 4. The reachability graph of the marked Petri net shown in Figure 3

For the marked Petri net shown in Figure 3 there are 12 reachable states. These states can be computed using the so-called *reachability graph* shown in Figure 4. All nodes correspond to reachable markings and each arc corresponds to the firing of a particular transition. Any path in the reachability graph corresponds to a possible firing sequence. For example, using Figure 4 is is easy to see that $\langle t3, t4, t3, t1, t4, t3, t2, t1\rangle$ is indeed possible and results in $[p2, p3, p4, p5]$. A

marked net may be unbounded, i.e., have an infinite number or reachable states. In this case, the reachability graph is infinitely large, but one can still construct the so-called coverability graph [26].

2.3 Workflow Nets

For process discovery, we look at processes that are instantiated multiple times, i.e., the same process is executed for multiple cases. For example, the process of handling insurance claims may be executed for thousands or even millions of claims. Such processes have a clear starting point and a clear ending point. Therefore, the following subclass of Petri nets (WF-nets) is most relevant for process discovery.

Definition 4 (Workflow nets). *Let $N = (P, T, F)$ be a Petri net and $\bar{t}$ a fresh identifier not in $P \cup T$. N is a* workflow net *(WF-net) iff:*

1. *object creation: P contains an input place i (also called source place) such that $\bullet i = \emptyset$,*
2. *object completion: P contains an output place o (also called sink place) such that $o\bullet = \emptyset$,*
3. *connectedness: $\bar{N} = (P, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\})$ is strongly connected, i.e., there is a directed path between any pair of nodes in $\bar{N}$.*

Clearly, Figure 3 is not a WF-net because a source and sink place are missing. Figure 5 shows an example of a WF-net: $\bullet start = \emptyset$, $end\bullet = \emptyset$, and every node is on a path from *start* to *end*.

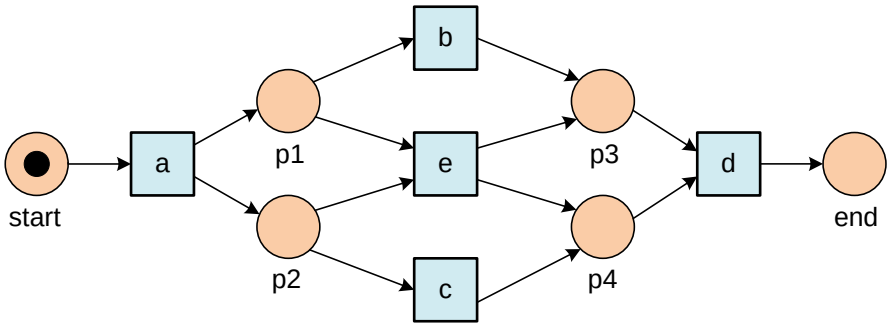

Fig. 5. A workflow net with source place $i = start$ and sink place $o = end$

Not every WF-net represents a correct process. For example, a process represented by a WF-net may exhibit errors such as deadlocks, tasks which can never become active, livelocks, garbage being left in the process after termination, etc. Therefore, we define the following correctness criterion.

Definition 5 (Soundness). *Let $N = (P, T, F)$ be a WF-net with input place i and output place o. N is* sound *iff:*

1. *safeness: $(N, [i])$ is safe, i.e., places cannot hold multiple tokens at the same time,*
2. *proper completion: for any marking $M \in [N, [i]\rangle$, $o \in M$ implies $M = [o]$,*
3. *option to complete: for any marking $M \in [N, [i]\rangle$, $[o] \in [N, M\rangle$, and*
4. *absence of dead tasks: $(N, [i])$ contains no dead transitions (i.e., for any $t \in T$, there is a firing sequence enabling t).*

The WF-net shown in Figure 5 is sound. Soundness can be verified using standard Petri-net-based analysis techniques. In fact soundness corresponds to liveness and safeness of the corresponding short-circuited net [1]. This way efficient algorithms and tools can be applied. An example of a tool tailored towards the analysis of WF-nets is Woflan [29]. This functionality is also embedded in our process mining tool ProM [2].

2.4 Problem Definition and Approaches

After introducing events logs and WF-nets, we can define the the main goal of process discovery.

Definition 6 (Process discovery). *Let L be an event log over A, i.e., $L \in \mathbb{B}(A^*)$. A* process discovery algorithm *is a function γ that maps any log L onto a Petri net $\gamma(L) = (N, M)$. Ideally, N is a sound WF-net and all traces in L correspond to possible firing sequence of (N, M).*

The goal is to find a process model that can "replay" all cases recorded in the log, i.e., all traces in the log are possible firing sequences of the discovered WF-net. Assume that $L_1 = [\langle a, b, c, d\rangle^5, \langle a, c, b, d\rangle^8, \langle a, e, d\rangle^9]$. In this case the WF-net shown in Figure 5 is a good solution. All traces in L_1 correspond to firing sequences of the WF-net and vice versa. Note that it may be possible that some of the firing sequences of the discovered WF-net do not appear in the log. This is acceptable as one cannot assume that all possible sequences have been observed. For example, if there is a loop, the number of possible firing sequences is infinite. Even if the model is acyclic, the number of possible sequences may be enormous due to choices and parallelism. Later in this article, we will discuss the quality of discovered models in more detail.

Since the mid-nineties several groups have been working on techniques for process mining [6, 3, 7, 11, 13, 15, 16, 30], i.e., discovering process models based on observed events. In [5] 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 [7]. In parallel, Datta [13] looked at the discovery of business process models. Cook et al. investigated similar issues in the context of software engineering processes [11]. Herbst [22] 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 [6] 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). In [15, 16] a more robust but less precise approach is presented.

Recently, people started using the "theory of regions" to process discovery. There are two approaches: state-based regions and language-based regions. State-based regions can be used to convert a transition system into a Petri net [4, 12]. Language-based regions add places as long as it is still possible to replay the log [10, 32]

More from a theoretical point of view, the process discovery problem is related to the work discussed in [9, 19, 20, 25]. In these papers the limits of inductive inference are explored. For example, in [20] it is shown that the computational problem of finding a minimum finite-state acceptor compatible with given data is NP-hard. Several of the more generic concepts discussed in these papers can be translated to the domain of process mining. It is possible to interpret the problem described in this paper as an inductive inference problem specified in terms of rules, a hypothesis space, examples, and criteria for successful inference. The comparison with literature in this domain raises interesting questions for process mining, e.g., how to deal with negative examples (i.e., suppose that besides log L there is a log L' of traces that are not possible, e.g., added by a domain expert). However, despite the relations with the work described in [9, 19, 20, 25] there are also many differences, e.g., we are mining at the net level rather than sequential or lower level representations (e.g., Markov chains, finite state machines, or regular expressions), tackle concurrency, and do not assume negative examples or complete logs.

The above approaches assume that there is no noise or infrequent behavior. For approaches dealing with these problems we refer to the work done by Christian Günther [21], Ton Weijters [30], and Ana Karla Alves de Medeiros [23].

3 The Alpha Algorithm

After introducing the process discovery problem and providing an overview of approaches described in literature, we focus on the α-algorithm [6]. The α-algorithm is not intended as a practical mining technique as it has problems with noise, infrequent/incomplete behavior, and complex routing constructs. Nevertheless, it provides a good introduction into the topic. The α-algorithm is very simple and many of its ideas have been embedded in more complex and robust techniques. Moreover, it was the first algorithm to really address the discovery of concurrency.

3.1 Basic Idea

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. To reflect this dependency, the corresponding Petri net should have a place connecting a to b. We distinguish four log-based ordering relations that aim to capture relevant patterns in the log.

Definition 7 (Log-based ordering relations). *Let L be an event log over A, i.e., $L \in \mathbb{B}(A^*)$. Let $a, b \in A$:*

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

Consider for example $L_1 = [\langle a, b, c, d \rangle^5, \langle a, c, b, d \rangle^8, \langle a, e, d \rangle^9]$. $c >_{L_1} d$ because d directly follows c in trace $\langle a, b, c, d \rangle$. However, $d \not>_{L_1} c$ because c never directly follows d in any trace in the log. $>_{L_1} = \{(a, b), (a, c), (a, e), (b, c), (c, b), (b, d), (c, d), (e, d)\}$ contains all pairs of activities in a "directly follows" relation. $c \rightarrow_{L_1} d$ because sometimes d directly follows c and never the other way around ($c >_{L_1} d$ and $d \not>_{L_1} c$). $\rightarrow_{L_1} = \{(a, b), (a, c), (a, e), (b, d), (c, d), (e, d)\}$ contains all pairs of activities in a "causality" relation. $b \|_{L_1} c$ because $b >_{L_1} c$ and $c >_{L_1} b$, i.e, sometimes c follows b and sometimes the other way around. $\|_{L_1} = \{(b, c), (c, b)\}$. $b \#_{L_1} e$ because $b \not>_{L_1} e$ and $e \not>_{L_1} b$. $\#_{L_1} = \{(a, a), (a, d), (b, b), (b, e), (c, c), (c, e), (d, a), (d, d), (e, b), (e, c), (e, e)\}$. Note that for any log L over A and $x, y \in A$: $x \rightarrow_L y$, $y \rightarrow_L x$, $x \#_L y$, or $x \|_L y$.

The log-based ordering relations can be used to discover patterns in the corresponding process model as is illustrated in Figure 6. If a and b are in sequence, the log will show $a >_L b$. 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 Figure 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 $b \|_L c$, 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 $a \|_L b$, then it appears that c needs to synchronize a and b (AND-join pattern).

Figure 6 only shows simple patterns and does not present the additional conditions needed to extract the patterns. However, the figure nicely illustrates the basic idea.

Consider for example WF-net N_2 depicted in Figure 7 and the log event log $L_2 = [\langle a, b, c, d, e, f, b, d, c, e, g \rangle, \langle a, b, d, c, e, g \rangle, \langle a, b, c, d, e, f, b, c, d, e, f, b, d, c, e, g \rangle]$. The α-algorithm constructs WF-net N_2 based on L_2. Note that the patterns in the model indeed match the log-based ordering relations extracted from the

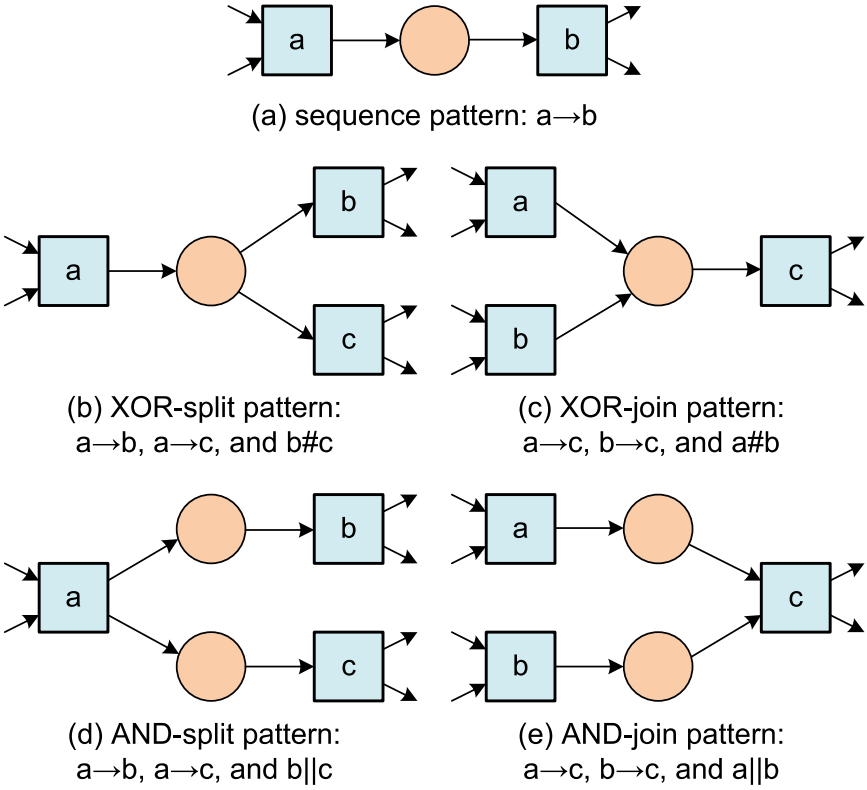

Fig. 6. Typical process patterns and the footprints they leave in the event log

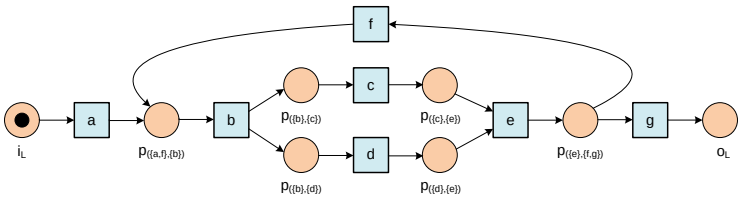

Fig. 7. WF-net N_2 derived from $L_2 = [\langle a, b, c, d, e, f, b, d, c, e, g\rangle, \langle a, b, d, c, e, g\rangle, \langle a, b, c, d, e, f, b, c, d, e, f, b, d, c, e, g\rangle]$

event log. Consider for example the process fragment involving b, c, d, and e. Obviously, this fragment can be constructed based on $b \rightarrow_{L_2} c$, $b \rightarrow_{L_2} d$, $c \|_{L_2} d$, $c \rightarrow_{L_2} e$, and $d \rightarrow_{L_2} e$. The choice following e is revealed by $e \rightarrow_{L_2} f$, $e \rightarrow_{L_2} g$, and $f \#_{L_2} g$. Etc.

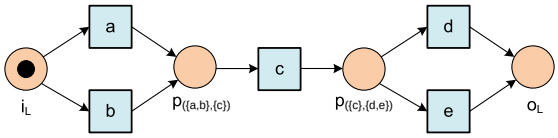

Fig. 8. WF-net N_3 derived from $L_3 = [\langle a, c, d\rangle^{45}, \langle b, c, d\rangle^{42}, \langle a, c, e\rangle^{38}, \langle b, c, e\rangle^{22}]$

Another example is shown in Figure 8. WF-net N_3 can be derived from $L_3 = [\langle a, c, d\rangle^{45}, \langle b, c, d\rangle^{42}, \langle a, c, e\rangle^{38}, \langle b, c, e\rangle^{22}]$. Note that here there are two start and two end activities. These can be found easily by looking for the first and last activities in traces.

3.2 Algorithm

After showing the basic idea and some examples, we describe the α-algorithm.

Definition 8 (α-algorithm). *Let L be an event log over T. $\alpha(L)$ is defined as follows.*

1. $T_L = \{t \in T \mid \exists_{\sigma \in L}\ t \in \sigma\}$,
2. $T_I = \{t \in T \mid \exists_{\sigma \in L}\ t = \mathit{first}(\sigma)\}$,
3. $T_O = \{t \in T \mid \exists_{\sigma \in L}\ t = \mathit{last}(\sigma)\}$,
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. $Y_L = \{(A, B) \in X_L \mid \forall_{(A', B') \in X_L} A \subseteq A' \ \wedge B \subseteq B' \Longrightarrow (A, B) = (A', B')\}$,
6. $P_L = \{p_{(A,B)} \mid (A, B) \in Y_L\} \cup \{i_L, o_L\}$,
7. $F_L = \{(a, p_{(A,B)}) \mid (A, B) \in Y_L \wedge a \in A\} \cup \{(p_{(A,B)}, b) \mid (A, B) \in Y_L \wedge b \in B\} \cup \{(i_L, t) \mid t \in T_I\} \cup \{(t, o_L) \mid t \in T_O\}$, *and*
8. $\alpha(L) = (P_L, T_L, F_L)$.

L is an event log over some set T of activities. In Step 1 it is checked which activities do appear in the log (T_L). These will correspond to the transitions of the generated WF-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 find the places of the WF-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 ($p_{(A,B)} \bullet = B$).

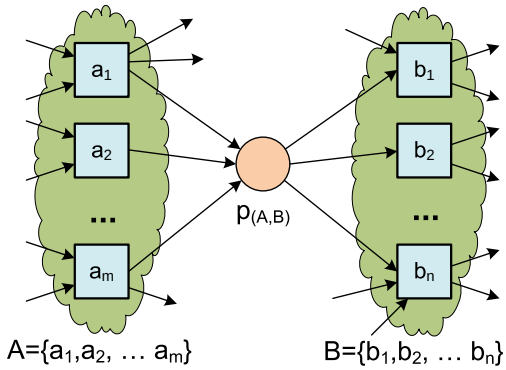

Fig. 9. Place $p_{(A,B)}$ connects the transitions in set A to the transitions in set B

The basic idea for finding $p_{(A,B)}$ is shown in Figure 9. All elements of A should have causal dependencies with all elements of B, i.e., for any $(a,b) \in A \times B$: $a \rightarrow_L b$. Moreover, the elements of A should never follow any of the other elements, i.e., for any $a_1, a_2 \in A$: $a_1 \#_L a_2$. A similar requirement holds for B.

Let us consider $L_1 = [\langle a,b,c,d\rangle^5, \langle a,c,b,d\rangle^8, \langle a,e,d\rangle^9]$. Clearly $A = \{a\}$ and $B = \{b,e\}$ meet the requirements stated in Step 4. Also note that $A' = \{a\}$ 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, $X_{L_1} = \{(\{a\},\{b\}), (\{a\},\{c\}), (\{a\},\{e\}), (\{a\},\{b,e\}), (\{a\},\{c,e\}), (\{b\},\{d\}), (\{c\},\{d\}), (\{e\},\{d\}), (\{b,e\},\{d\}), (\{c,e\},\{d\})\}$. If one would insert a place for any element in X_{L_1} 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. So $Y_{L_1} = \{(\{a\},\{b,e\}), (\{a\},\{c,e\}), (\{b,e\},\{d\}), (\{c,e\},\{d\})\}$.

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 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. The result is a Petri net $\alpha(L) = (P_L, T_L, F_L)$ that describes the behavior seen in event log L.

Thus far we presented three logs and three WF-nets. Clearly $\alpha(L_2) = N_2$, and $\alpha(L_3) = N_3$. In figures 7 and 8 the places are named based on the sets Y_{L_2} and Y_{L_3}. Moreover, $\alpha(L_1) = N_1$ modulo renaming of places (because different place names are used in Figure 5). These examples show that the α-algorithm is indeed able to discover WF-nets based event logs.

Figure 10 shows another example. WF-net N_4 can be derived from $L_4 = [\langle a,b,e,f\rangle^2, \langle a,b,e,c,d,b,f\rangle^3, \langle a,b,c,e,d,b,f\rangle^2, \langle a,b,c,d,e,b,f\rangle^4, \langle a,e,b,c,d,b,f\rangle^3]$, i.e., $\alpha(L_4) = N_4$.

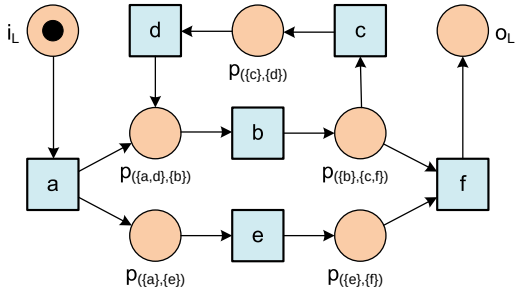

Fig. 10. WF-net N_4 derived from $L_4 = [\langle a, b, e, f\rangle^2, \langle a, b, e, c, d, b, f\rangle^3, \langle a, b, c, e, d, b, f\rangle^2, \langle a, b, c, d, e, b, f\rangle^4, \langle a, e, b, c, d, b, f\rangle^3]$

3.3 Limitations

In [6] it was shown that the α-algorithm is able to discover a large class of WF-nets if one assumes that the log is *complete* with respect to the log-based ordering relation $>_L$. This assumption implies that, for any event log L, $a >_L b$ if a can be directly followed by b. We revisit the notion of completeness later in this article.

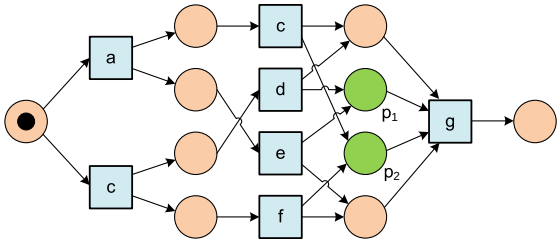

Fig. 11. WF-net N_5 derived from $L_5 = [\langle a, c, e, g\rangle^2, \langle a, e, c, g\rangle^3, \langle b, d, f, g\rangle^2, \langle b, f, d, g\rangle^4]$

Even if we assume that the log is complete, the α-algorithm has some problems. There are many different WF-nets that have the same possible behavior, i.e., two models can be structurally different but trace equivalent. Consider for example $L_5 = [\langle a, c, e, g\rangle^2, \langle a, e, c, g\rangle^3, \langle b, d, f, g\rangle^2, \langle b, f, d, g\rangle^4]$. $\alpha(L_5)$ is shown in Figure 11. Although the model is able to generate the observed behavior, the resulting WF-net is needlessly complex. Two of the input places of g are redundant, i.e., they can be removed without changing the behavior. The places denoted as p_1 and p_2 are so-called implicit places and can be removed without allowing for more traces. In fact, Figure 11 shows only one of many possible trace equivalent WF-nets.

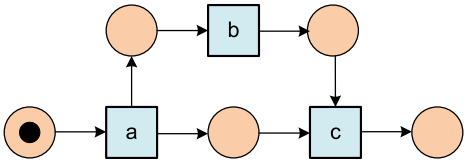

Fig. 12. Incorrect WF-net N_6 derived from $L_6 = [\langle a,c\rangle^2, \langle a,b,c\rangle^3, \langle a,b,b,c\rangle^2, \langle a,b,b,b,b,c\rangle^1]$

The original α-algorithm has problems dealing with short loops, i.e., loops of length 1 or 2. This is illustrated by WF-net N_6 in Figure 12 which shows the result of applying the basic algorithm to $L_6 = [\langle a,c\rangle^2, \langle a,b,c\rangle^3, \langle a,b,b,c\rangle^2, \langle a,b,b,b,b,c\rangle^1]$. It is easy to see that the model does not allow for $\langle a,c\rangle$ and $\langle a,b,b,c\rangle$. In fact, in N_6, transition b needs to be executed precisely once and there is an implicit place connecting a and c. This problem can be addressed easily as shown in [24]. Using an improved version of the α-algorithm one can discover the WF-net shown in Figure 13.

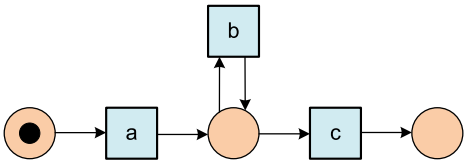

Fig. 13. WF-net N_7 having a so-called "short-loop"

A more difficult problem is the discovery of so-called non-local dependencies resulting from non-free choice process constructs. An example is shown in Figure 14. This net would be a good candidate after observing for example $L_8 = [\langle a,c,d\rangle^{45}, \langle b,c,e\rangle^{42}]$. However, the α-algorithm will derive the WF-net without the place labeled p_1 and p_2. Hence, $\alpha(L_8) = N_3$ shown in Figure 8 although the traces $\langle a,c,e\rangle$ and $\langle b,c,d\rangle$ do not appear in L_8. Such problems can be (partially) resolved using refined versions of the α-algorithm such as the one presented in [31].

The above examples show that the α-algorithm is able to discover a large class of models. The basic 8-line algorithm has some limitations when it comes to particular process patterns (e.g., short-loops and non-local dependencies). Some of these problems can be solved using various refinements. However, several more fundamental problems remain as shown next.

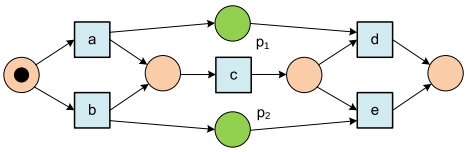

Fig. 14. WF-net N_8 having a non-local dependency

4 Challenges in Process Discovery

The α-algorithm was the first process discovery algorithm to adequately capture concurrency. Today there are much better algorithms that overcome the weaknesses of the α-algorithm. These are either variants of the α-algorithm or completely different approaches (e.g., based on regions or genetic algorithms). However, there are still several challenges. Below we list some of these challenges.

The first problem is that many algorithms have problems with *complex control-flow constructs*. For example, the choice between the concurrent execution of b and c or the execution of just e shown in Figure 5 cannot be handled by many algorithms. Most algorithms do *not* allow for so-called "non-free-choice constructs" where concurrency and choice meet. The concept of free-choice nets is well-defined in the Petri net domain [14]. However, in reality processes tend to be non-free-choice. In the example of Figure 5, the α-algorithm is able to deal with the non-free-choice construct. However, it is easy to think of a non-free-choice process that cannot be discovered by the α-algorithm (see for example N_8 in Figure 14). The non-free-choice construct is just one of many constructs that existing process mining algorithms have problems with. Other examples are arbitrary nested loops, cancelation, unbalanced splits and joins, partial synchronization, etc. In this context it is important to note that *process mining is, by definition, restricted by the expressive power of the target language*, i.e., if a simple or highly informal language is used, process mining is destined to produce less relevant or over-simplified results.

The second problem is the fact that most algorithms have problems with *duplicates*. The same activity may appear at different places in the process or different activities may be recorded in an indistinguishable manner. Consider for example Figure 5 and assume that activities a and d are both recorded as x (or, equivalently, assume that a and d are both replaced by activity x). Hence, an event log could be $L'_1 = [\langle x, b, c, x\rangle^5, \langle x, c, b, x\rangle^8, \langle x, e, x\rangle^9]$. Most algorithms will try to map the first and the second x onto the same activity. In some cases this make sense, e.g., to create loops. However, if the two occurrences of x (i.e., a and d) really play a different role in the process, then algorithms that are unable to separate them will run into all kinds of problems, e.g., the model becomes more difficult or incorrect. Since the duplicate activities have the same "footprint" in the log, most algorithms map these different activities onto a single activity thus making the model incorrect or counter-intuitive.

The third problem is that many algorithms have a tendency to generate *inconsistent models*. Note that here we do not refer to the relation between the log and the model but to the internal consistency of the model by itself. For example, the α-algorithm may yield models that have deadlocks or livelocks when the log shows certain types of behavior. When using Petri nets as a model to represent processes, an obvious choice is to require the model to be *sound* (i.e., free of deadlocks and other anomalies). Unfortunately, this can typically only be achieved by severely limiting the expressiveness of the modeling languages (e.g., require it to be block structured).

The fourth and last problem described here is probably the most important problem: Existing algorithms have *problems balancing between "overfitting" and "underfitting"*. Overfitting is the problem that a very specific model is generated while it is obvious that the log only holds example behavior, i.e., the model explains the particular sample log but a next sample log of the same process may produce a completely different process model. Underfitting is the problem that the model over-generalizes the example behavior in the log, i.e., the model allows for very different behaviors from what was seen in the log. The problem of balancing between "overfitting" and "underfitting" is related to the notion of completeness discussed next.

5 Completeness: Between Overfitting and Underfitting

When it comes to process mining the notion of *completeness* is very important. Like in any data mining or machine learning context one cannot assume to have seen all possibilities in the "training material" (i.e., the event log at hand). For WF-net N_1 in Figure 5 and event log $L_1 = [\langle a, b, c, d\rangle^5, \langle a, c, b, d\rangle^8, \langle a, e, d\rangle^9]$, the set of possible traces found in the log is exactly the same as the set of possible traces in the model. In general, this is not the case. For example, the trace $\langle a, b, e, c, d\rangle$ may be possible but did not (yet) occur in the log. Process models typically allow for an exponential or even infinite number of different traces (in case of loops). Moreover, some traces may have a lower probability than others. Therefore, it is unrealistic to assume that every possible trace is present in the event log.

The α-algorithm assumes a relatively weak notion of completeness to avoid this problem. Although N_2 has infinitely possible firing sequences, a small log like $L_2 = [\langle a, b, c, d, e, f, b, d, c, e, g\rangle, \langle a, b, d, c, e, g\rangle, \langle a, b, c, d, e, f, b, c, d, e, f, b, d, c, e, g\rangle]$ can be used to construct N_2. The α-algorithm uses a local completeness notion based on $>_L$, i.e., if there are two activities a and b, and a can be directly followed by b, then this should be observed at least once in the log.

To illustrate the relevance of completeness, consider a process where in principle 10 tasks which can be executed in parallel and the corresponding log contains information about 10000 cases. The total number of possible interleavings is 10! = 3628800. Hence, it is impossible that each interleaving is present in the log as there are fewer cases than potential traces. Even if there are 3628800 cases in the log, it is extremely unlikely that all possible variations are present. To motivate

this consider the following analogy. In a group of 365 people it is very unlikely that everyone has a different birthdate (probability $365!/365^{365} \approx 0$). Similarly, it is unlikely that all possible traces will occur for any process of some complexity. It is even worse as some sequences are less probable than others. Weaker completeness notions are therefore needed. Note that for the process where 10 tasks which can be executed in parallel, local completeness can reduce the required number of observations dramatically. For example, for the α-algorithm only $10(10-1) = 90$ rather than 3628800 different observations are needed to construct the model.

Completeness is closely linked to the notions of *overfitting* and *underfitting* mentioned earlier. It is also linked to Occam's Razor, a principle attributed to the 14th-century English logician William of Ockham. The principle states that "one should not increase, beyond what is necessary, the number of entities required to explain anything", i.e., one should look for the "simplest model" that can explain what is in the log. Using this principle different algorithms assume different notions of completeness.

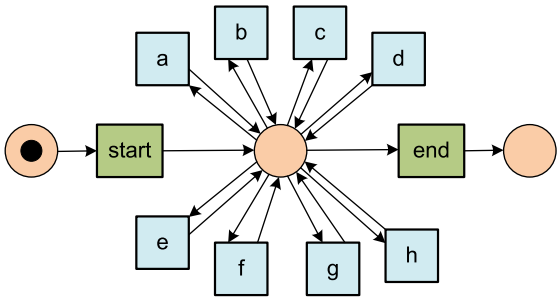

Fig. 15. The so-called "flower Petri net" allowing for any log containing activities $\{a, b, \ldots h\}$

Process mining algorithms needs to strike a balance between "overfitting" and "underfitting". A model is overfitting if it does not generalize and only allows for the exact behavior recorded in the log. This means that the corresponding mining technique assumes a very strong notion of completeness: "If the sequence is not in the event log, it is not possible.". An underfitting model over-generalizes the things seen in the log, i.e., it allows for more behavior even when there are no indications in the log that suggest this additional behavior. An example is shown in Figure 15. This so-called "flower Petri net" allows for any sequence starting with *start* and ending with *end* and containing any ordering of activities in between. Clearly, this model allows for event log L_1 (without the added *start* and *end* activities) but also many more, e.g., all other example logs mentioned in this article and many more.

Let us now consider another example showing that it is difficult to balance between being too general and too specific. Consider WF-net N_3 shown in Figure 8 and N_8 shown in Figure 14. Both nets can produce log $L_8 = [\langle a, c, d\rangle^{45}, \langle b, c, e\rangle^{42}]$ and only N_3 can produce $L_3 = [\langle a, c, d\rangle^{45}, \langle b, c, d\rangle^{42}, \langle a, c, e\rangle^{38}, \langle b, c, e\rangle^{22}]$. Clearly, N_3 is the logical choice for L_3. Moreover, although both nets can produce L_8, it is obvious that N_8 is a better model for L_8 as none of the 87 cases required the two additional alternatives. However, now consider $L_9 = [\langle a, c, d\rangle^{99}, \langle b, c, d\rangle^{1}, \langle a, c, e\rangle^{2}, \langle b, c, e\rangle^{98}]$. One can argue that N_3 is a better model for L_9 as all traces can be reproduced. However, 197 out of 200 traces can be explained by the more precise model N_8. If the three traces are seen as deviations, the main behavior is captured by N_8 and not N_3. Such considerations show that there is a delicate balance and that it is non-trivial to compare logs and process models. Hence, it is difficult, if not impossible, to select the "best" model.

In [28] notions such as *fitness* and *appropriateness* have been quantified. An event log and Petri net "fit" if the Petri net can generate each trace in the log.[1] In other words: the Petri net should be able to "parse" (i.e., reproduce) every activity sequence observed. In [28] it is shown that it is possible to quantify fitness as a measure between 0 and 1. The intuitive meaning is that a fitness close to 1 means that all observed events can be explained by the model. However, the precise meaning is more involved since tokens can remain in the net and not all transactions in the model need to be logged [28]. Unfortunately, a good fitness alone does not imply that the model is indeed suitable, e.g., it is easy to construct Petri nets that are able to reproduce any event log (cf. the "flower model" in Figure 15). Although such Petri nets have a fitness of 1, they do not provide meaningful information. Therefore, in [28] a second dimension is introduced: *appropriateness*. Appropriateness tries to answer the following question: "Does the model describe the observed process in a concise way?". This notion can be evaluated from both a *structural* and a *behavioral* perspective. In [28] it is shown that a "good" process model should somehow be "minimal in structure" to clearly reflect the described behavior, referred to as *structural appropriateness*, and "minimal in behavior" in order to represent as closely as possible what actually takes place, which is called *behavioral appropriateness*. The ProM conformance checker [28] supports both the notion of fitness and various notions of appropriateness, i.e., for a given log and a given model it computes the different metrics.

Although there are different ways to quantify notions such as fitness and appropriateness, it is difficult to agree on the definition of an "optimal model". What is optimal seems to depend on the intended purpose and even given a clear metric there may be many models having the same score. *Since there is not "one size fits all", it is important to have algorithms that can be tuned to specific applications.*

[1] It is important not to confuse *fitness* with *overfitting* and *underfitting*. A model that is overfitting or underfitting may have a fitness of 1.

Why is process mining a difficult problem? There are obvious reasons that also apply to many other machine learning problems. Examples are noise and a large search space. However, there are also some specific problems:

- there are *no negative examples* (i.e., a log shows what has happened but does not show what could not happen),
- due to concurrency, loops, and choices the *search space has a complex structure* and the log typically only contains a *fraction* of all possible behaviors, and
- there is *no clear relation between the size of a model and its behavior* (i.e., a smaller model may generate more or less behavior while classical MDL methods typically assume some monotonicity property).

6 Applications

Despite the challenges mentioned, there are mature techniques available today. ProM supports the major process mining techniques described in literature. Robust techniques such as the Fuzzy Miner [21] and the Heuristic Miner [30] can analyze complex event logs efficiently and effectively. These techniques have been implemented in ProM which can be downloaded from `www.processmining.org`.

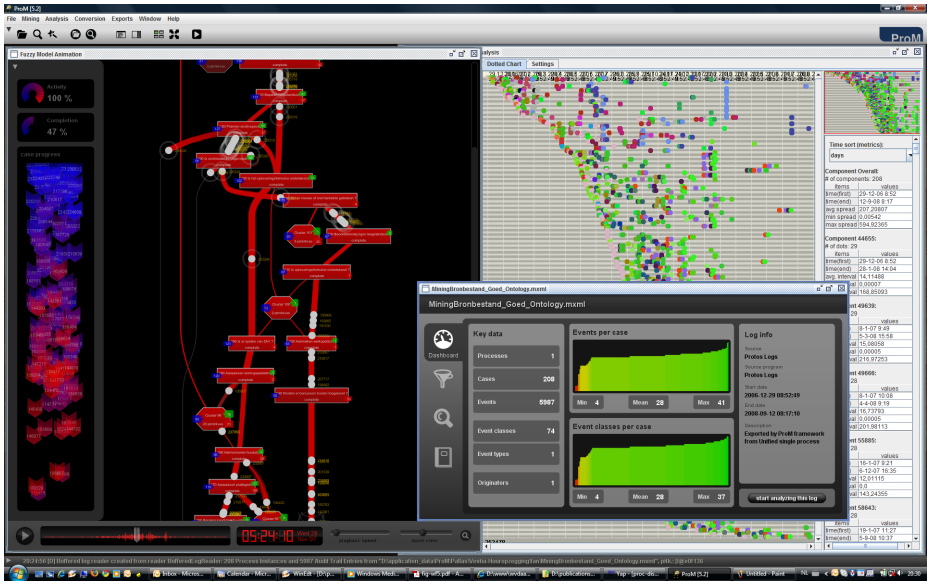

Fig. 16. Screenshot of ProM while analyzing an event log of a housing agency

In this paper, we do not elaborate on more advanced process mining techniques and their implementations in ProM. Figure 16 shows a screenshot of ProM while analyzing a log containing 208 cases that generated 5987 events. The left part shows an animation of the discovered fuzzy model, i.e., the recorded events are replayed on top of the discovered process model. This gives insights in bottlenecks, deviations, etc. The right part depicts a so-called dotted chart showing a helicopter view of all events in the log.

Most other tools in this area only implement one specific technique and focus on a single perspective and technique. There are also a few commercial tools. Most notable are *Futura Reflect* by Futura Process Intelligence and *BPM|one* by Pallas Athena. These two tools support process discovery based on genetic mining [23] and should be seen as a re-implementation of plug-ins already present in ProM (making it faster and more user-friendly). *Comprehend* by Open Connect, *Interstage Automated Business Process Discovery and Visualization* by Fujitsu, *Process Discovery Focus* by Iontas, and *Enterprise Visualization Suite* by BusinesScape are some examples of commercial tools that offer some form of process discovery. Typically such tools can only discover sequential models and are, in terms of functionality, comparable to a handful of the 250 plug-ins of ProM. Another example is the *ARIS Process Performance Manager* (ARIS PPM) by IDS Scheer which has adopted the idea to extract social networks from event logs from ProM. Until recently, ARIS PPM did not support control-flow discovery in the sense described above. However, it was possible to link events to process fragments that are then glued together as they occur. The drawback of this approach is that the model still needs to be made manually. Despite the limitations mentioned, these examples show that there is a growing interest from software vendors (and their customers) in process mining.

We have applied process mining in a wide variety of organizations:

- Municipalities (e.g., Alkmaar, Heusden, Harderwijk, etc.)
- Government agencies (e.g., Rijkswaterstaat, Centraal Justitieel Incasso Bureau, Justice department)
- Insurance related agencies (e.g., UWV)
- Banks (e.g., ING Bank)
- Hospitals (e.g., AMC hospital, Catharina hospital)
- Multinationals (e.g., DSM, Deloitte)
- High-tech system manufacturers and their customers (e.g., Philips Healthcare, ASML, Thales)
- Media companies (e.g. Winkwaves)

Based on these experiences, we highlight three application domains where process mining can be extremely valuable.

6.1 Hospital information systems

For many years hospitals have been working on Electronic Patient Records (EPR), i.e., information about the health history of a patient, including all past

and present health conditions, illnesses and treatments, is managed by the information system. Although there are still many problems that need to be resolved (mainly of a non-technical nature), today's hospital information systems already contain a wealth of event data. For example, by Dutch law all hospitals need to record the diagnostic and treatment steps at the level of individual patients in order to receive payment. This so-called "Diagnose Behandeling Combinatie" (DBC) forces Dutch hospitals to record all kinds of events. Clearly, there is a need for process discovery, conformance checking, and process improvement (e.g., reducing flow times) in the health-care domain.

6.2 Municipal information systems

Municipalities are continuously confronted with new regulations that need to be implemented. Although these processes are typically not "controlled" by some process engine, the processes are documented and the activities in these processes are recorded. Conformance checking is highly relevant as compliance is important in public administration. Moreover, many regulations have legal deadlines, so conformance checking with temporal constraints and time prediction is relevant. For example, if it is predicted that a case will not be dealt with before the legal deadline, appropriate actions can be triggered. Moreover, there are many less structured processes where discovery can be very useful.

6.3 High-tech deployed systems

Increasingly high-tech deployed systems such as high-end copiers, complex medical equipment, lithography systems, automated production systems, etc. record events which allow for the remote monitoring of these systems. For the manufacturers of such systems it is interesting to use process mining, e.g., are the systems used in the way they should be used and do systems show signs of deviating behavior. The "CUSTOMerCARE Remote Services Network" of Philips Healthcare (PH) is a nice example of an enabling technology for process mining. Through this network PH is recording events from systems all over the world (e.g., X-ray machines in hospitals). Process mining can be used to see if maintenance is needed and gives good insight in whether the medical equipment is used as intended.

7 Process Mining: A Grand Challenge for Computational Intelligence

Computational Intelligence (CI) is sometimes defined as "a branch of computer science studying problems for which there are no effective computational algorithms" [17]. The techniques used are often inspired by nature and typical examples are neural networks, fuzzy sets, rough sets, swarm intelligence, evolutionary computing, and genetic algorithms [18]. Clearly, there are also relationships with

artificial intelligence, reenforcement learning, data mining, machine learning, and pattern recognition.

In this paper, we presented an overview of process mining and zoomed in on the challenge of process discovery by presenting a simple mining algorithm. The α-algorithm was presented because of its simplicity and because it forms the basis for many other discovery algorithms. However, there are also other process discovery techniques that use approaches more common in CI. The genetic miner [23] uses genetic algorithms and the fuzzy miner [21] uses a variety of quantitative measures to decide whether there may be causal dependency or not. Given the nature of the problem, it seems that also other typical CI techniques such as neural networks, rough sets, and reenforcement learning could be used for process discovery (and other forms of process mining). In fact, the problem described in this paper can be seen as a grand challenge for CI researchers given its many intricate difficulties.

As indicated in the introduction, processes are less tangible than products or data. Processes tend to be elusive; they emerge and dissolve and many actors may influence parts of it. Yet most individuals and organizations are involved in a multitude of processes and many of these processes are essential for their existence. Although processes are considered important and are frequently discussed, there is little attention for studying them in detail. It seems that lion's share of attention goes to their high-level design and management at an aggregate level, while neglecting the analysis of the actual process events. Until recently, it was difficult to systematically record events. However, the "data explosion" in all domains where IT systems are used, makes process mining techniques viable.

In many branches of science researchers study reality, e.g., physicists study matter and its motion through spacetime, chemists study the composition, structure, and properties of matter, and biologists study evolution, organisms, cells, and genes. In these branches of science it is common to discover reality by observation or to test a model using real-life observations. Computer science is mainly a design science, i.e., often some new (typically virtual) reality is created. However, for building information systems that work well in reality, it is vital to understand the processes they are supposed to support. This illustrates the importance of process mining. Therefore, the IEEE recently launched the *IEEE Task Force on Process Mining* chaired by the author of this article. This Task Force has been established in the context of the Data Mining Technical Committee (DMTC) of the Computational Intelligence Society (CIS) of the Institute of Electrical and Electronic Engineers, Inc. (IEEE). The goal of this Task Force is to promote the research, development, education and understanding of process mining. More concretely, its purpose is to make end-users, developers, consultants, and researchers aware of the state-of-the-art in process mining, promote the use of process mining techniques and tools, stimulate new applications, play a role in standardization efforts for logging event data, and organize tutorials, special sessions, workshops, panels. The interested reader is invited to contribute to the work of the IEEE Task Force on Process Mining.

Acknowledgment

The author would like to thank the many people involved in the development of ProM and the organizations sponsoring our work on process mining.

References

1. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
2. W.M.P. van der Aalst, B.F. van Dongen, C.W. Günther, R.S. Mans, A.K. Alves de Medeiros, A. Rozinat, V. Rubin, M. Song, H.M.W. Verbeek, and A.J.M.M. Weijters. ProM 4.0: Comprehensive Support for Real Process Analysis. In J. Kleijn and A. Yakovlev, editors, *Application and Theory of Petri Nets and Other Models of Concurrency (ICATPN 2007)*, volume 4546 of *Lecture Notes in Computer Science*, pages 484–494. Springer-Verlag, Berlin, 2007.
3. W.M.P. van der Aalst, H.A. Reijers, A.J.M.M. Weijters, B.F. van Dongen, A.K. Alves de Medeiros, M. Song, and H.M.W. Verbeek. Business Process Mining: An Industrial Application. *Information Systems*, 32(5):713–732, 2007.
4. W.M.P. van der Aalst, V. Rubin, B.F. van Dongen, E. Kindler, and C.W. Günther. Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting. *Software and Systems Modeling*, 2009.
5. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
6. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
7. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.
8. R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proceedings of the 11th International Conference on Data Engineering (ICDE'95)*, pages 3–14. IEEE Computer Society, 1995.
9. D. Angluin and C.H. Smith. Inductive Inference: Theory and Methods. *Computing Surveys*, 15(3):237–269, 1983.
10. R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process Mining Based on Regions of Languages. In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 375–383. Springer-Verlag, Berlin, 2007.
11. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
12. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri Nets from Finite Transition Systems. *IEEE Transactions on Computers*, 47(8):859–882, August 1998.
13. A. Datta. Automating the Discovery of As-Is Business Process Models: Probabilistic and Algorithmic Approaches. *Information Systems Research*, 9(3):275–301, 1998.

14. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.
15. B.F. van Dongen and W.M.P. van der Aalst. Multi-Phase Process Mining: Building Instance Graphs. In P. Atzeni, W. Chu, H. Lu, S. Zhou, and T.W. Ling, editors, *International Conference on Conceptual Modeling (ER 2004)*, volume 3288 of *Lecture Notes in Computer Science*, pages 362–376. Springer-Verlag, Berlin, 2004.
16. B.F. van Dongen and W.M.P. van der Aalst. Multi-Phase Mining: Aggregating Instances Graphs into EPCs and Petri Nets. In D. Marinescu, editor, *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management*, pages 35–58. Florida International University, Miami, Florida, USA, 2005.
17. W. Duch. What Is Computational Intelligence and Where Is It Going? In W. Duch and J. Mandziuk, editors, *Challenges for Computational Intelligence*, volume 1420 of *Studies in Computational Intelligence*, pages 1–13. Springer-Verlag, Berlin, 2007.
18. A.P. Engelbrecht. *Computational Intelligence: An Introduction.* Wiley & Sons, 2007.
19. E.M. Gold. Language Identfication in the Limit. *Information and Control*, 10(5):447–474, 1967.
20. E.M. Gold. Complexity of Automaton Identification from Given Data. *Information and Control*, 37(3):302–320, 1978.
21. C.W. Günther and W.M.P. van der Aalst. Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics. In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 328–343. Springer-Verlag, Berlin, 2007.
22. J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.
23. A.K. Alves de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Genetic Process Mining: An Experimental Evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007.
24. A.K.A. de Medeiros, W.M.P. van der Aalst, and A.J.M.M. Weijters. Workflow Mining: Current Status and Future Directions. In R. Meersman, Z. Tari, and D.C. Schmidt, editors, *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, volume 2888 of *Lecture Notes in Computer Science*, pages 389–406. Springer-Verlag, Berlin, 2003.
25. L. Pitt. Inductive Inference, DFAs, and Computational Complexity. In K.P. Jantke, editor, *Proceedings of International Workshop on Analogical and Inductive Inference (AII)*, volume 397 of *Lecture Notes in Computer Science*, pages 18–44. Springer-Verlag, Berlin, 1989.
26. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
27. A. Rozinat and W.M.P. van der Aalst. Decision Mining in ProM. In S. Dustdar, J.L. Fiadeiro, and A. Sheth, editors, *International Conference on Business Process Management (BPM 2006)*, volume 4102 of *Lecture Notes in Computer Science*, pages 420–425. Springer-Verlag, Berlin, 2006.
28. A. Rozinat and W.M.P. van der Aalst. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, 33(1):64–95, 2008.
29. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.

30. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.
31. L. Wen, W.M.P. van der Aalst, J. Wang, and J. Sun. Mining process models with non-free-choice constructs. *Data Mining and Knowledge Discovery*, 15(2):145–180, 2007.
32. J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik. Process Discovery using Integer Linear Programming. In K. van Hee and R. Valk, editors, *Proceedings of the 29th International Conference on Applications and Theory of Petri Nets (Petri Nets 2008)*, volume 5062 of *Lecture Notes in Computer Science*, pages 368–387. Springer-Verlag, Berlin, 2008.