Time Prediction Based on Process Mining

W.M.P. van der Aalst, M.H. Schonenberg, and M. Song

Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{w.m.p.v.d.aalst}@tue.nl

Abstract. Process mining allows for the automated discovery of process models from event logs. These models provide insights and enable various types of model-based analysis. This paper demonstrates that the discovered process models can be extended with information to *predict the completion time* of running instances. There are many scenarios where it is useful to have reliable time predictions. For example, when a customer phones her insurance company for information about her insurance claim, she can be given an estimate for the remaining processing time. In order to do this, we provide a configurable approach to construct a process model, augment this model with time information learned from earlier instances, and use this to predict e.g. the completion time. To provide meaningful time predictions we use a configurable set of abstractions that allow for a good balance between "overfitting" and "underfitting". The approach has been implemented in ProM and through several experiments using real-life event logs we demonstrate its applicability.

1 Introduction

More and more information about processes is recorded by information systems in the form of so-called "event logs". A wide variety of *Process-Aware Information Systems* (PAISs) [12] is recording excellent data on actual events as they are taking place. ERP (Enterprise Resource Planning), WFM (WorkFlow Management), BPM (Business Process Management), CRM (Customer Relationship Management), SCM (Supply Chain Management), and PDM (Product Data Management) systems are examples of such systems. Despite the omnipresence and richness of these event logs, most software vendors use this information for answering only relatively simple questions *under the assumption that the process is fixed and known*, e.g., the calculation of simple performance metrics like utilization and flow time for a well-known process. However, in many domains processes are evolving and people typically have an oversimplified and incorrect view of the actual business processes. Therefore, *process mining* techniques attempt to extract non-trivial and useful information from event logs. One aspect of process mining is *control-flow discovery*, i.e., automatically constructing a process model (e.g., a Petri net or BPMN model) describing the causal dependencies between activities [3,5–7,9,31,32]. The basic idea of control-flow discovery is very simple: given an event log containing a set of traces, automatically construct a suitable process model "describing the behavior" seen in the log. Such discovered processes have proven to be very useful for the understanding, redesign, and continuous improvement of business processes [2].

As process mining techniques are getting more mature, there is the desire to *use the discovered models in an operational setting*. This means that the PAIS is using results from process mining at *runtime*. An example is the *recommendation service* presented in [26]. This service, which is implemented in the process mining tool ProM, gives advice on the best possible next activity. Unlike the fixed routing in a workflow management system which is strictly enforced, the recommendation service merely gives an advice what to do next. Another example is the *prediction service* presented in [11]. This service is using non-parametric regression to predict the completion time of partially executed process instances. Using the same regression technique it is also estimated whether activities will be executed in the future and, if so, the time it takes to reach them.

The goal of the approach presented in this paper is similar to the prediction service described in [11]. However, we want to overcome some of the limitations of this earlier approach and therefore use a completely new approach to time prediction. Unlike existing approaches, the problem is not reduced to a simple heuristic (e.g., always estimating half of the average flow time or the average flow time minus the already elapsed time) or a regression model. Instead an *annotated transition system* is generated that represents an *abstraction of the process with time annotations*.

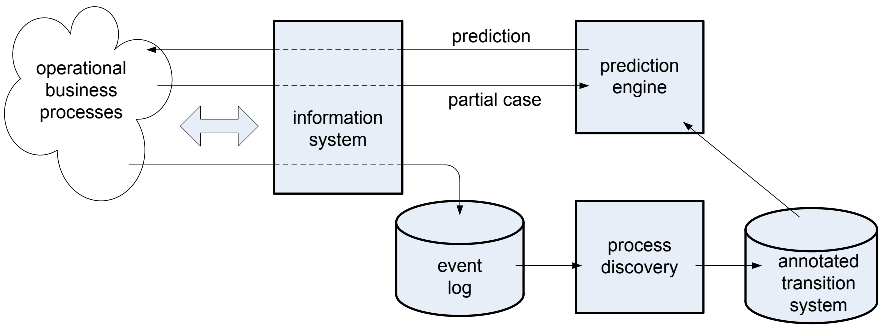


Fig. 1. Overview of our approach.

Figure 1 shows an overview of the approach. There is an information system supporting some operational process. Events that take place in this process are recorded (e.g., starting some activity). This recorded information is used to derive a process model. In the process mining field, many process discovery algorithms have been proposed. Here we use a variant of the approach described in [3], i.e., using various abstractions a transition system is derived. This transition system is annotated with information about *elapsed times* (e.g., the average time to reach a particular state), *sojourn times* (e.g., the average time spent in a particular state), and *remaining times* (e.g., the average time to reach the end from this state). The annotated transition system can be used to predict the remaining flow time of all or some of the running cases (i.e., process instances). In our approach, we heavily rely on the transition system generation and corresponding abstractions presented in [3]. This allows us to find a balance between a transition system that is too specific (overfitting) and one that is too general (underfitting) with respect to the log and thus provide better predictions. As will be demonstrated, our approach performs much better than simple heuristics and also outperforms regressions models in terms of efficiency and precision.

The approach can be easily extended to predict other features of a case. For example, the transition system can also be annotated with information about the occurrence of particular event or the time until a particular event. This can lead to predictions such as: "there is a 90% probability that your claim will be rejected", "there is 90% probability that this activity will be conducted in the next three days", and "the expected time until the submission of some intermediate report is 11 days". Although these additional features can easily be added, we focus here on predicting the completion of a case.

The approach presented in this paper has been implemented in ProM, cf. www.processmining.org. ProM serves as a testbed for our process mining research [1] and has been applied in various domains, e.g., hospitals (AMC and Catherina hospitals), banks (ING), several municipalities (Heusden, Alkmaar, etc.), high-tech system manufacturers (ASML and Philips Medical Systems), software repositories for open-source projects, etc. To show the applicability of our results, we present an experimental evaluation using a simulation model and two real-life case studies. In these case studies, we use our techniques to predict the completion time of cases in two different Dutch municipalities.

The remainder of this paper is organized as follows. Related work is discussed in Section 2. Section 3 introduces process mining and shows that a transition system can be derived after (potentially) making some abstractions. Section 4 presents the main idea of time prediction based on annotated transition systems. The implementation in ProM is described in Section 5. Section 6 discusses various ways of defining and measuring the quality of predictions. The approach has been validated and tested using several (real-life) examples. The results of our evaluation are given in Section 7. Section 8 concludes the paper.

2 Related Work

Since the mid-nineties several groups have been working on techniques for process mining [3,5–7,9,15,31,32], i.e., discovering process models based on observed events. In [4] 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 [6]. In parallel, Datta [9] looked at the discovery of business process models. Cook et al. investigated similar issues in the context of software engineering processes [7]. Herbst [16] 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 [5] was the first technique taking 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 the context of the ProM framework [1] more robust techniques have been developed. The heuristics miner [31] and the fuzzy miner [15] can deal with incomplete, unbalanced, and/or noisy events logs. The two-phase approach presented in [3] allows for various abstractions to obtain more useful models. The first step in [3] is used as a basis for the work in this paper and will be elaborated later. It is impossible to give a complete review of process mining techniques here, see www.processmining.org for more pointers to literature.

The approaches mentioned above focus on control-flow discovery. However, when event logs contain time information, the discovered models can be extended with timing information. For example, in [29] it is shown how timed automata can be derived. In [23] it is shown how any Petri net discovered by ProM can be enriched with timing and resource information.

The above approaches all focus on discovering process models based on historic information and do not support users at run-time. The recommendation service of ProM learns based on historic information and uses this to guide the user in selecting the next work-item [26]. This is related to the use of case-based reasoning in workflow systems [30]. Most related to our work is the prediction service presented in [11]. This service predicts the completion time of cases by using non-parametric regression [8,11]. In [10] different techniques are compared using a case study including various heuristics and the prediction service presented in [11]. In [25] it is shown that the interaction between cases and the availability of resources are important factors when predicting the remaining time until completion. The topic of time prediction was also discussed in [20], but no concrete prediction technique was proposed. Instead the problem of having cross-trained resources (i.e., resources that are more flexible to perform also other tasks) on performance prediction was highlighted. Eder et al. [13,14] also looked into time management in workflow systems. However, the focus of their work is more on scheduling and escalation and like in [10,20] assuming that the workflow is known beforehand and stable. Also related is the prediction engine of Staffware [25,28] which is using simulation to complete audit trails with expected information about future events. This particular approach is rather unreliable since it is based on one run through the system using a copy of the actual engine. Hence, no probabilities are taken into account and there is no means of "learning" to make better predictions over time. A more refined approach focusing on the transient behavior (called "short-term simulation") is presented in [24].

The approach presented in this paper differs from existing approaches in various ways. First of all, an explicit process model (i.e., the annotated transition system) is constructed (unlike heuristics or simple regression models) and this model is used for predictions. Second, the degree of abstraction can be adjusted based on the questions at hand and the volume of historic information.

Note that the process model can become more fine-grained once more data is available. Third, the approach allows for better diagnostics (unlike for example the non-parametric regression approach [11]). Finally, the approach has a better performance compared to approaches based on simulation or regression (both in terms of quality of prediction and computation time).

3 Transition System Generation Using Abstractions

This section describes the basis for our prediction approach. We first show how to construct a transition system based on an event log. In Section 4, we annotate this transition system and use it to predict e.g. completion times.

3.1 Preliminaries

To explain the different strategies for constructing transition systems from event logs, we need the following notations.

$f \in A \rightarrow B$ is a function with domain A and range B. $f \in A \not\rightarrow B$ is a partial function, i.e., the domain of f may be a subset of A.

A *multi-set* (also referred to as *bag*) is like a set where each element may occur multiple times. For example, $[a, b^2, c^3, d, d, e]$ is the multi-set with nine elements: one a, two b's, three c's, two d's, and one e.

$\mathbb{B}(A) = A \rightarrow \mathbb{N}$ is the set of multi-sets (bags) over a finite domain A, i.e., $X \in \mathbb{B}(A)$ is a multi-set, where for each $a \in A$, $X(a)$ denotes the number of times a is included in the multi-set. For example, if $X = [a, b^2, c^3, d]$, then $X(b) = 2$ and $X(e) = 0$. The sum of two multi-sets $(X + Y)$, the difference $(X - Y)$, the presence of an element in a multi-set $(x \in X)$, and the notion of subset $(X \leq Y)$ are defined in a straightforward way. For example, $[a, b^2, c^3, d] + [c^3, d, e^2, f^3] = [a, b^2, c^6, d^2, e^2, f^3]$. Moreover, we also apply these operators to sets, where we assume that a set is a multi-set in which every element occurs exactly once. The operators are also robust with respect to the domains of the multi-sets, i.e., even if X and Y are defined on different domains, $X + Y$, $X - Y$, and $X \leq Y$ are defined properly by extending the domain where needed. $\mid X \mid = \sum_{a \in A} X(a)$ is the cardinality of some multi-set X over A. $set(X)$ transforms a bag X into a set, i.e., $set(X) = \{a \in X \mid X(a) > 0\}$.

$\mathcal{P}(A)$ is the powerset of A, i.e., $\mathcal{P}(A) = \{X \mid X \subseteq A\}$.

For a given set A, A^* is the set of all finite sequences over A. A finite sequence over A of length n is a mapping $\sigma \in \{1, \ldots, n\} \rightarrow A$. Such a sequence is represented by a string, i.e., $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ where $a_i = \sigma(i)$ for $1 \leq i \leq n$. $hd^k(\sigma) = \langle a_1, a_2, \ldots, a_{k\ min\ n} \rangle$, i.e., the sequence consisting of the first k elements (if possible). Note that $hd^0(\sigma)$ is the empty sequence and for $k \geq n$: $hd^k(\sigma) = \sigma$. $tl^k(\sigma) = \langle a_{(n-k+1)\ max\ 1}, a_{k+2}, \ldots, a_n \rangle$, i.e., the sequence composed of the last k elements (if possible). Note that $tl^0(\sigma)$ is the empty sequence and for $k \geq n$: $tl^k(\sigma) = \sigma$. $\sigma \uparrow X$ is the projection of σ onto some subset $X \subseteq A$, e.g., $\langle a, b, c, a, b, c, d \rangle \uparrow \{a, b\} = \langle a, b, a, b \rangle$ and $\langle d, a, a, a, a, a, a, d \rangle \uparrow \{d\} = \langle d, d \rangle$.

For any sequence σ over A, the Parikh vector $par(\sigma)$ maps every element a of A onto the number of occurrences of a in σ, i.e., $par(\sigma) \in \mathbb{B}(A)$ where for any $a \in A$: $par(\sigma)(a) = \mid \sigma \uparrow \{a\} \mid$. Later, we will use the Parikh vector to count the number of times an activity occurs in a log trace.

3.2 Event logs

The goal of process mining is to extract knowledge about a particular (business) 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). Today, many information systems record such information. To explain the kind of input needed for process mining and our prediction approach, we first define the concept of an *event*.

Definition 1 (Event, Property). *Let $\mathcal{E}$ be the event universe, i.e., the set of all possible events identifiers, and $\mathcal{T}$ the time domain. We assume that events are characterized by various* properties*, e.g., an event has a timestamp, corresponds to an activity, is executed by a particular person, has associated costs, etc. We do not impose a specific set of properties, however, given the focus of this paper, we assume that one of these properties is the timestamp of the event, i.e., there is a function $prop_T \in \mathcal{E} \rightarrow \mathcal{T}$ assigning timestamps to events. As a shorthand, we denote $\overline{e} = prop_T(e)$, i.e., the time of an event $e \in \mathcal{E}$.*

So an event e is described by some unique identifier and can have several properties. In this paper, we focus on the timestamp of an event $\overline{e} = prop_T(e)$. However, other properties such as the person executing the event ($prop_{resource}(e)$), the name of the corresponding activity ($prop_A(e)$), the cost of an event ($prop_{cost}(e)$), etc. can be used both for process discovery and prediction techniques presented in thus paper.

An *event log* is simply a set of events. Each event in the log is linked to a particular trace and globally unique, i.e., the same event cannot occur twice in a log. Note that a trace in a log represents a particular process instance also referred to as "case" (e.g., a customer order, the treatment of a patient, or an insurance claim). Moreover, time should be non-decreasing within each trace in the log.

Definition 2 (Trace, Event log). *A* trace *is a finite sequence of events $\sigma \in \mathcal{E}^*$ such that each event appears only once and time is non-decreasing, i.e., for $1 \leq i < j \leq \mid \sigma \mid$: $\sigma(i) \neq \sigma(j)$ and $\overline{\sigma(i)} \leq \overline{\sigma(j)}$. $\mathcal{C}$ is the set of all possible traces (including partial traces). An* event log *is a set of traces $L \subseteq \mathcal{C}$ such that each event appears at most once in the entire log, i.e., for any $\sigma_1, \sigma_2 \in L$: $set(par(\sigma_1)) \cap set(par(\sigma_2)) = \emptyset$ or $\sigma_1 = \sigma_2$.*

Note that $\overline{\sigma(i)} \leq \overline{\sigma(j)}$ means that time is non-descending (i.e., $prop_T(\sigma(i)) \leq prop_T(\sigma(j))$ if i occurs before j). In the expression "$set(par(\sigma_1)) \cap set(par(\sigma_2))$" traces are converted into multi-sets using *par* which in turn are converted into sets using *set*. This is done to ensure that events are globally unique and do not appear in multiple traces.

Table 1 shows a fragment of some event log. Only two traces are shown, both containing 4 events. Each event has a unique id and several properties. For example event 35654423 is an instance of activity A that occurred on December 30th at 11.10, was executed by John, and costed 300 euros. The second trace starts with event 35655526 and also refers to an instance of activity A.

Table 1. A fragment of an event log.

event id	properties				
	timestamp	activity	resource	cost	...
35654423	30-12-2008:11.10	A	John	300	...
35654424	30-12-2008:15.21	B	John	400	...
35654425	30-12-2008:15.35	C	John	100	...
35654426	30-12-2008:15.55	D	John	400	...
35655526	29-12-2008:16.15	A	Ann	300	...
35655527	30-12-2008:16.05	C	John	450	...
35655528	30-12-2008:16.25	B	Pete	350	...
35655529	31-12-2008:10.55	D	Ann	300	...
...	...	...	...	...	...

3.3 Constructing a Transition System

One of the goals of process mining is to extract process models from logs such as the one depicted in Table 1. In this paper, we partly use the approach presented in [3]. We first define a transition system. A *transition system* is a triplet (S, E, T) where S is the *state space* (i.e., possible states of the process), E is the set of *event labels* (i.e., transition labels), and $T \subseteq S \times E \times S$ is the *transition relation* describing how the system can move from one state to another. A transition $(s_1, e, s_2) \in T$ describes that the process can move from state s_1 to s_2 by an event labeled e. This is often denoted as $s_1 \xrightarrow{e} s_2$. A transition system has some initial state and set of final states. The set of behaviors possible according to a transition system is given by all "walks" from the initial state to some final state. Hence a trace is possible according to the transition system if it corresponds to such a "walk" in the transition system. So the goal is to come up with a transition system that given an event log, characterizes the observed behaviors well. Typically, one aims at a model that allows for most of the behavior in the log and not “too much” additional behavior. Before discussing the delicate balance between “overfitting” en “underfitting”, we first provide the general idea of extracting a transition system from an event log.

It is natural to assume that at any point in time a process instance is in some state and that this state depends on its history. Hence, any prefix of a trace in the log should be mapped onto some state. The *state representation function* takes care of this.

Definition 3 (State representation). *A state representation function l^{state} is a function that, given a (partial) trace σ produces some representation. Formally, $l^{state} \in \mathcal{C} \rightarrow \mathcal{R}$ where $\mathcal{C}$ is the set of possible traces and $\mathcal{R}$ is the set of possible (state) representations (e.g., sequences, sets, or bags over one or more event properties).*

Assume that σ is the partial trace consisting of the first two events in Table 1. One possibility is that function l^{state} maps σ on the last activity, i.e., the activity name property of the last event in σ. So $l^{state}(\sigma) = B$. Another possibility is that function l^{state} maps σ onto the set of resources that have worked on the process instance, i.e., $l^{state}(\sigma) = \{John\}$. Note that these two examples use abstractions, e.g., only the last activity is relevant or only the people that have worked on the case are relevant. It is also possible to have no abstractions, i.e., $l^{state}(\sigma) = \sigma$. However, as we will see later, it is impossible to make predictions without some form of abstraction.

Just like we need to label the states in the transition system, we also need to label events. Note that any event e in the log extends a partial trace σ_1 into a longer trace $\sigma_2 = \sigma_1; \langle e \rangle$, i.e., σ_1 concatenated with the sequence just containing event e. In the transition system there should be a transition connecting state $l^{state}(\sigma_1)$ to state $l^{state}(\sigma_2)$. This transition has an event label $l^{event}(e)$, based on some *event representation function l^{event}*.

Definition 4 (Event representation). *An event representation function l^{event} is a function that, given an event e produces some representation. Formally, $l^{event} \in \mathcal{E} \rightarrow \mathcal{R}$ where $\mathcal{E}$ is the set of possible events and $\mathcal{R}$ is the set of possible (event) representations (e.g., the corresponding activity name).*

Based on particular l^{state} and l^{event} functions, we can define the transition system. The approach is very simple. The states in the transition system correspond to prefixes in the log mapped to the desired representation using a particular state representation function l^{state}. Moreover, the transition relation is computed by “replaying” the traces on the transition system while using the event representation function l^{event}.

Definition 5 (Transition system). *Let $L \subseteq \mathcal{C}$ be an event log. Given a state representation function l^{state} and an event representation function l^{event}, we define a* labeled transition system *$TS = (S, E, T)$ where $S = \{l^{state}(hd^k(\sigma)) \mid \sigma \in L \wedge 0 \leq k \leq \mid \sigma \mid\}$ is the state space*[1]*, $E = \{l^{event}(\sigma(k)) \mid \sigma \in L \wedge$*

[1] Recall that $hd^k(\sigma)$ is a prefix of σ, i.e., the sequence consisting of the first k elements of σ (cf. Section 3.2).

$1 \leq k \leq| \sigma |\}$ *is the set of events labels, and* $T \subseteq S \times E \times S$ *with* $T = \{(l^{state}(hd^k(\sigma)), l^{event}(\sigma(k+1)), l^{state}(hd^{k+1}(\sigma))) \mid \sigma \in L \wedge 0 \leq k <| \sigma |\}$ *is the transition relation.* $S^{start} = \{l^{state}(\langle\rangle)\}$ *is the singleton set of initial states.* $S^{end} = \{l^{state}(\sigma) \mid \sigma \in L\}$ *is the set of final states.*

The set of states of the transition system is determined by the range of function l^{state} when applied to the log data. The transitions in the transition system have a label based on function l^{event}. The naive algorithm for constructing a transition system is straightforward: for every trace σ, iterating over k (i.e. $0 \leqslant k \leqslant| \sigma |$), we create a new state $l^{state}(hd^k(\sigma))$ if it does not exist yet. Then the traces are scanned for transitions $l^{state}(hd^k(\sigma)) \xrightarrow{l^{event}(\sigma(k+1))} l^{state}(hd^{k+1}(\sigma))$ and these are added if they do not exist already. So given concrete functions l^{state} and l^{event}, it is possible to automatically build a transition system.

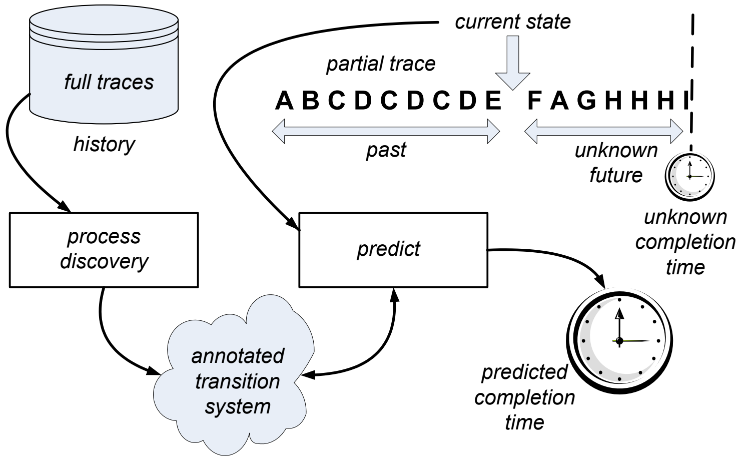


Fig. 2. Illustrating the approach using an example trace with a known past and an unknown future.

Figure 2 shows an example of a trace represented as a sequence of activity names $\langle A, B, C, D, C, D, C, D, E, F, A, G, H, H, H, I\rangle$. After the prefix $\langle A, B, C, D, C, D, C, D, E\rangle$ is executed it is in principle unknown that the "future" is $\langle F, A, G, H, H, H, I\rangle$ and that the process instance ends at a particular time. However, based on the event log consisting of full traces executed in the past, one can construct a transition system as sketched in Figure 2. Moreover, based on the prefix $\langle A, B, C, D, C, D, C, D, E\rangle$ and l^{state}, one can determine the current state in the transition system. Based on this a prediction is made as will be elaborated later.

3.4 Abstraction

Based on a log L and the l^{state} and l^{event} functions, a transition system is defined. The key element is to select the right l^{state} and l^{event} functions. In the extreme case, $l^{state}(\sigma) = \sigma$ and $l^{event}(e) = e$ for any prefix σ and event e. In this case, there are no abstractions and all states (except the initial state) are visited only once when replaying the event log. Hence, every new process instance will be unique and one cannot learn form earlier instances. Clearly, this is undesirable and some degree of abstraction is needed. Therefore, we consider an example where some form of abstraction is used. Assume we have an event log and we are only interested in activity names. Suppose that $prop_A \in \mathcal{E} \rightarrow \mathcal{A}$ is a function mapping events onto the corresponding activity names. Figure 3 refers to such a log without showing the complete log, i.e., just a few traces are shown. Also note that per event only the activity name is shown and other properties and the event's id are not shown. The first line in the log shown in Figure 3 could refer to the first event in Table 1. Since we are only interested in activity names, it makes sense to choose the event representation function such that $l^{event}(e) = prop_A(e)$. Now assume that we let states only depend on the occurrences of earlier activities, i.e., the order is not important. Moreover, we do not distinguish between the single execution of an activity and multiple executions of this activity. This corresponds to the state representation function $l_1^{state}(\sigma) = \{prop_A(e) \mid e \in \sigma\}$ (for $\sigma \in L$ or some prefix). Alternatively, one could argue that the order is important and choose the state representation function $l_2^{state}(\sigma) = \langle prop_A(\sigma(1)), prop_A(\sigma(2)), \ldots, prop_A(\sigma(|\ \sigma\ |))\rangle$ (for $\sigma \in L$). Figure 3 shows the result of applying these two state representation functions. Note that the transition system in Figure 3(b) is more precise than Figure 3(a), i.e., a weaker abstraction is used. For example, l_2^{state} distinguishes between activity sequences $\langle A, B, C\rangle$ and $\langle A, C, B\rangle$ while l_1^{state} does not.

Note that both transition systems provided in Figure 3 can replay the log, i.e., any trace corresponds to a walk in the transition system starting in a initial state and ending in a final state. This shows that the transition system indeed reflects the behavior seen in the log. In Figure 3, initial and final states are denoted using "small dangling arcs" going into respectively out of the respective nodes.

When building a transition system, we aim at a balance between "overfitting" and "underfitting". Therefore, we elaborate on these two notions. Let L be a log and TS be a transition system.

– *TS is overfitting* L if TS does not generalize and is sensitive to particularities in L. In an extreme case, TS could merely be a representation of the log without any inference. A mining algorithm is producing overfitting models if the removal or addition of a small percentage of the process instances in L would lead to a remarkably different model. In a complex process with many possible paths, most process instances will follow a path not taken by other instances in the same period. Therefore, it is undesirable to construct a model that allows only for the paths that happened to be present in the log as this is only a fraction of all possible paths. If one knows that only

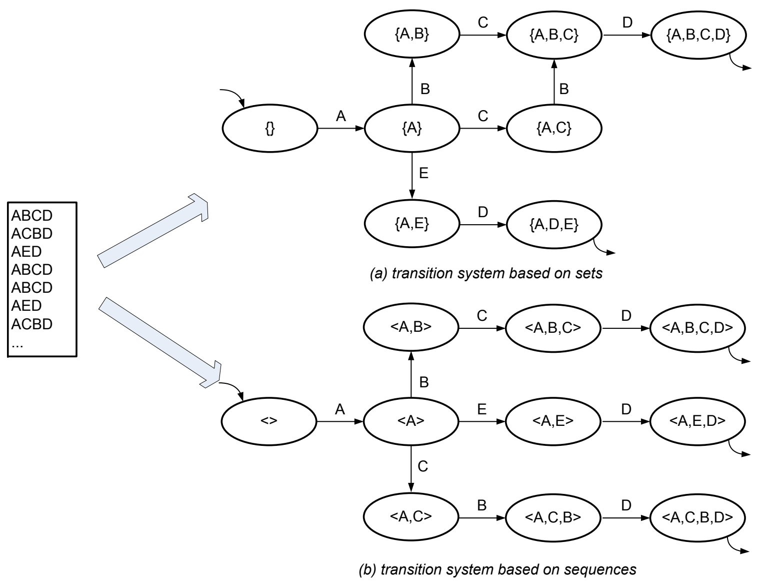


Fig. 3. Two example transition systems extracted from the same log using (a) $l_1^{state}(\sigma) = \{prop_A(e) \mid e \in \sigma\}$ and (b) $l_2^{state}(\sigma) = \langle prop_A(\sigma(1)), prop_A(\sigma(2)), \ldots, prop_A(\sigma(\mid \sigma \mid))\rangle$ as state representation functions.

a fraction of the possible event sequences are in the log, the only way to avoid overfitting is to generalize and have a model TS that allows for more behavior than recorded in L.

– *TS is underfitting* L if TS allows for "too much behavior" that is not supported by L. This is also referred to as "overgeneralization". It is very easy to construct a model that allows for the behavior seen in the log but also completely different behavior. For example, assume a log L consisting of 1000 cases. For each case A is followed by B and there are no cases where B is followed by A. Obviously, one could derive a causal dependency between A and B. However, one could also create a model TS where A and B are in parallel (i.e., all interleavings are included). The latter would not be "wrong" in the sense that the behavior seen in the log is possible according to the model. However, it is very unlikely and therefore one could argue that this TS is underfitting L.

Note that the notions of overfitting and underfitting are orthogonal to "non-fitting". A model is non-fitting if the observed traces are not possible according to the model. An overfitting log that merely encodes all observed traces is still fitting in the sense that the log could have been produced by the model. An underfitting log that allows for any trace over a given alphabet is also still fitting because it does not exclude the observed behavior. There are various ways of quantifying these notions [21]. This is, however, outside the scope of this paper.

3.5 Examples

To conclude this section, we give some examples of possible abstractions and illustrate the importance of such abstractions using a real-life example.

Let us first consider *abstractions with respect to event names.* As indicated earlier, the event representation function $l^{event}(e) = e$ does not provide any abstraction and will result in arcs in the transition system that are taken only once when relaying a log. A typical abstraction is $l^{event}(e) = prop_A(e)$, i.e., mapping the event onto the name of the activity. This was used in Figure 3. Of course other properties or combinations of properties can be used, e.g., $l^{event}(e) = prop_{cost}(e)$ or $l^{event}(e) = (prop_A(e), prop_{resource}(e))$. Note that it is also possible to use a rather rigorous abstraction such as $l^{event}(e) = null$. Now all events are labeled *null* and it becomes impossible to distinguish activities, etc.

Abstractions with respect to states are more involved because they are not based on a single event but on a sequence of events. Below, we informally discuss some of the abstraction mechanisms for states.

Abstraction 1: Maximal horizon The basis of the state calculation can be the complete prefix of some partial prefix. In the latter case, only a subset of the trace is considered. For example, instead of taking the complete prefix $\langle A, B, C, D, C, D, C, D, E\rangle$ shown in Figure 2, only the last four ($h = 4$) events could be considered: $\langle D, C, D, E\rangle$. In a partial prefix, only the h most recent events are considered as input for the state calculation. Taking a complete prefix corresponds to $h = \infty$.

Abstraction 2: Filter The second abstraction is to filter the (partial) prefix, i.e., certain events are simply not considered when calculating the current state. For example filtering could be used to project the horizon onto a set of activities X, i.e., only events that correspond to some activity in X are considered in l^{state}. For example, if $X = \{C, D\}$, then the prefix $\langle A, B, C, D, C, D, C, D, E\rangle$ shown in Figure 2 is reduced to $\langle C, D, C, D, C, D\rangle$. Note that the filtering is applied to the sequence resulting from the horizon. The occurrence of some activity $a \in X$ is considered relevant for the state of a case. If $a \notin X$, then the occurrence of a is still relevant for the process (i.e., it may appear on the arcs in the transition system) but is assumed to be irrelevant for determining the state. If a is not relevant at all, it should be filtered out before and should not appear in L.

Note that the above two abstractions can be swapped. However, the order influences the result.

Abstraction 3: Sequence, bag, or set The first two abstractions yield a sequence. The third abstraction mechanism optionally removes the order or frequency from the resulting trace. For the current state it may be less interesting to know when some activity a occurred and how many times a occurred, i.e., only the fact that it occurs within the scope determined by the first three abstractions is relevant. In other cases, it may be relevant to know how many times

a occurred or it may be essential to know whether a occurred before b or not. This suggests that there are three ways of representing knowledge about the past:

– *sequence*, i.e., the order of activities is recorded in the state,
– *multi-set of activities*, i.e., the number of times each activity is executed ignoring their order, and
– *set of activities*, i.e., the mere presence of activities.

Consider again the prefix $\langle A, B, C, D, C, D, C, D, E\rangle$. The above three possibilities result in $\langle A, B, C, D, C, D, C, D, E\rangle$ (sequence), $[A, B, C^3, D^3, E]$ (multi-set), or $\{A, B, D, E\}$ (set).

The notations introduced in Section 3.1 can be used to formalize the abstractions mentioned above. For example, let $\alpha(\sigma) = \langle prop_A(\sigma(1)), prop_A(\sigma(2)), \ldots, prop_A(\sigma(\mid \sigma \mid))\rangle$ be a prefix based on activity names. Then state representation $l_1^{state}(\sigma) = set(par(\alpha(\sigma)))$ uses the set abstraction without filtering and an infinite horizon. $l_2^{state}(\sigma) = \alpha(\sigma)$ uses the sequence abstraction without filtering and an infinite horizon. $l_3^{state}(\sigma) = par(tl^4(\alpha(\sigma)))$ uses the multi-set abstraction without filtering and finite horizon ($h = 4$). $l_4^{state}(\sigma) = par(\alpha(\sigma) \uparrow \{a, b, c\})$ uses the multi-set abstraction after filtering. $l_5^{state}(\sigma) = set(par(tl^{10}(\alpha(\sigma)) \uparrow \{a, b, c\}))$ uses various abstractions, i.e., the state based on a prefix is determined by which of the activities a, b, and/or c occurred in the last 10 events.

So far, we only used rather academic examples to illustrate issues related to abstraction and the need to balance between overfitting and underfitting. However, it is important to realize that these issues are of the utmost importance when applying process mining in a real-life setting. We have been applying process mining in a wide variety of organizations and were often confronted with spaghetti-like models when applying classical process mining approaches. These models where typically the result of overfitting, i.e., the models were a correct reflection of reality, but not very useful.

To illustrate this we show some results based on an event log of a municipality of about 40.000 citizens in the south of The Netherlands. The event log is based on the process "Bezwaar WOZ". Later we will use the same log in one of the case studies used to evaluate our prediction approach.

The "Bezwaar WOZ" process handles objections (i.e., appeals) against the real-estate property valuation or the real-estate property tax. We used an event log with data on 1982 objections handled by the municipality. The log contains 12726 events. Because the actual activity names are not relevant for our discussion here (and because of reasons of confidentiality), we anonymized the process and replaced names by letters. (The resulting models are not intended to be readable anyway.)

Figure 4 shows two transition systems generated using our approach. The larger model was obtained by applying assuming that the state of a case is determined by the set of activities that have taken place, i.e., $l^{state}(\sigma) = set(par(\alpha(\sigma)))$. Figure 4(a) shows the whole model and the selected part in a bit more detail. This model is able to reproduce the event log, i.e., all observed traces can be reproduced and the model does not allow for any traces not present in the original

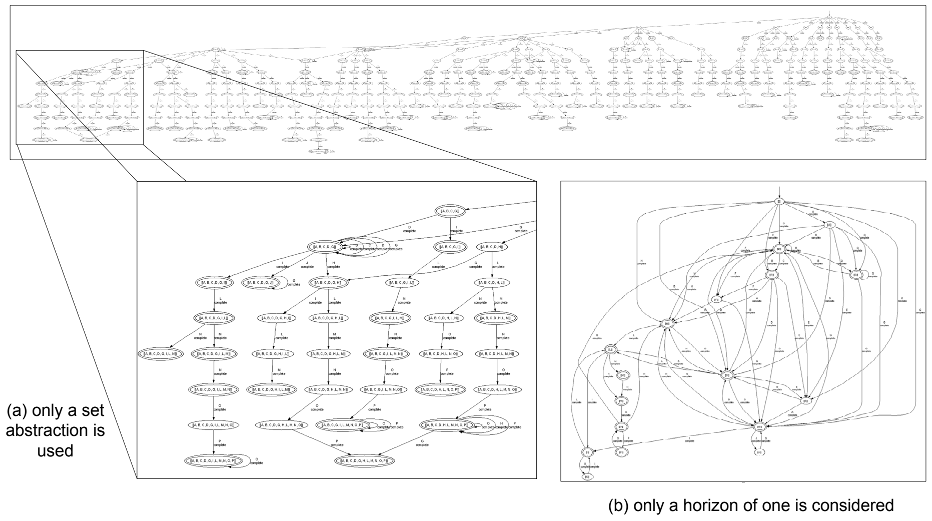

(a) only a set abstraction is used
(b) only a horizon of one is considered

Fig. 4. Two models discovered using an event log of a Dutch municipality. Although both models are based on the same log and provide information on the same set of activities, they are very different. The larger model is clearly overfitting, difficult to interpret, and, therefore, not very useful. The smaller model is obtained after applying a more rigorous abstraction (setting the horizon $h = 1$). This more simple model provides better insights.

event log. So the model is definitely "correct" but not very useful as it does not give much insight into the Municipality's appeal process. The second (smaller) transition system (cf. Figure 4(b)) was obtained based the same log but now using a more rigorous abstraction. It uses the abstraction that the state of a case is determined by only the last activity that has taken place (if any), i.e., $l^{state}(\sigma) = tl^1(\alpha(\sigma))$. This simpler model is also able to reproduce the event log, i.e., all observed traces can be generated by the net. However, the model also allows for many traces not present in the original log. Note that the two transition systems in Figure 4 are not supposed to be readable and are only shown to illustrate the effect of abstraction.

It should be noted that both models in Figure 4 provide information on identical sets of activities, i.e., the scope is not changed. Both models are able to reproduce the initial log and no noise or infrequent behavior have been removed for the smaller model. Therefore, Figure 4 nicely illustrates the relevance of abstraction. By using an appropriate state representation function l^{state} and an appropriate event representation function l^{event}, one can balance between underfitting and overfitting.

4 Time Prediction

In the previous section, we showed how a transition system can be generated on the basis of an event log. In this section, we show how this transition system can be annotated and used for prediction purposes. In Figure 2, we already showed an overview of the overall approach. Based on the event log, an annotated transition system is generated. Whenever we want to predict the completion time of some process instance, we take its partial trace (i.e., the sequence of events executed thus far) and use the state representation function l^{state} to map the partial trace onto a state in the transition system. Here we can *learn from the information collected for earlier process instances that visited the same state.* Using this information, a prediction is made, e.g., based on the average time to completion for earlier process instances in a similar state.

In this section, we first show how to construct the annotated transition system. Then, we show how it can be used for predictions.

4.1 Constructing an Annotated Transition System

The goal is to attach predictive information to the states of the transition system, e.g., "In this state the average time until completion is 6.5 days". In order to do so, we *annotate* the states with *measurements*. For example, we scan the history and for each situation were an instance was in a state s, we annotate the state with the remaining time until completion. This way *states are annotated with multi-sets of measurements* that are used as a basis for predictions.

Table 2. An example log. Each line corresponds to a trace represented as a sequence of activities with timestamps.

1	$\langle A^{00}, B^{06}, C^{12}, D^{18}\rangle$
2	$\langle A^{10}, C^{14}, B^{26}, D^{36}\rangle$
3	$\langle A^{12}, E^{22}, D^{56}\rangle$
4	$\langle A^{15}, B^{19}, C^{22}, D^{28}\rangle$
5	$\langle A^{18}, B^{22}, C^{26}, D^{32}\rangle$
6	$\langle A^{19}, E^{28}, D^{59}\rangle$
7	$\langle A^{20}, C^{25}, B^{36}, D^{44}\rangle$

As a running example, we use the event log shown in Table 2. Each line corresponds to a process instance, e.g., the first trace $\langle A^{00}, B^{06}, C^{12}, D^{18}\rangle$ refers to a process instance where activity A was executed at time 0, activity B was executed at time 6, activity C was executed at time 12, and activity D was executed at time 18[2]. For simplicity, execution durations are not considered in

[2] Note that we use a more compact representation here, e.g., we assume that the only two relevant properties are (a) the activity name and (b) the timestamp, and we do not attach unique id's to events. As discussed in Section 3.2, events can have many more properties and are uniquely identifiable.

this example and we assume that each instance starts with the execution of its first event.

Suppose we use a state representation function l^{state} that represents partial traces by the set of activities that have been executed. Now consider all prefixes of the first trace $\langle A^{00}, B^{06}, C^{12}, D^{18} \rangle$. The empty prefix $\langle \rangle$ maps onto state $\emptyset$ and has a remaining time of 18 time units.[3] Therefore, we add 18 to the annotation of state $\emptyset$. The prefix $\langle A^{00} \rangle$ maps onto state $\{A\}$ and also has a remaining time of 18. Therefore, we add 18 to the annotation of state $\{A\}$. The prefix $\langle A^{00}, B^{06} \rangle$ maps onto state $\{A, B\}$ and has a remaining time of $18 - 6 = 12$ time units. Therefore, we add 12 to the annotation of state $\{A, B\}$. The prefix $\langle A^{00}, B^{06}, C^{12} \rangle$ results in the addition of annotation "6" to state $\{A, B, C\}$ and finally prefix $\langle A^{00}, B^{06}, C^{12}, D^{18} \rangle$ results in the addition of annotation "0" to state $\{A, B, C, D\}$. This process is repeated for all other traces. Consider for example the third trace $\langle A^{12}, E^{22}, D^{56} \rangle$. The empty prefix $\langle \rangle$ maps onto state $\emptyset$ and has a remaining time of $56 - 12 = 44$ time units. Therefore, we add 44 to the annotation of state $\emptyset$. The prefix $\langle A^{12} \rangle$ maps onto state $\{A\}$ and also has a remaining time of 44. Therefore, we add 44 to the annotation of state $\{A\}$. The prefix $\langle A^{12}, E^{22} \rangle$ maps onto state $\{A, E\}$ and has a remaining time of $56 - 22 = 34$ time units. Therefore, we add 34 to the annotation of state $\{A, E\}$. Etc. After we have followed this procedure for the whole log in Table 2, then state $\emptyset$ is annotated with a bag containing seven elements: $[18, 26, 44, 13, 14, 40, 24]$. State $\{A, E\}$ is annotated with a bag containing two elements: $[34, 31]$. State $\{A, B, C\}$ is annotated with a bag containing five elements: $[6, 10, 6, 6, 8]$. State $\{A, B, C, D\}$ is also annotated with a bag containing five elements: $[0, 0, 0, 0, 0]$.

The following definition formalizes the way of annotating states described before. Each full trace is split into a prefix σ_1 ("the part that already took place") and the postfix σ_2 ("the part that still needs to happen"). Based on this a measurement $l^{measure}(\sigma_1, \sigma_2)$ is generated that can be attached to the corresponding state. Note that it is crucial that we have full historic information, i.e., although we only consider σ_1 to determine the corresponding state $l^{state}(\sigma_1)$, we can use both the "past" σ_1 and "future" σ_2 to generate a measurement.

Definition 6 (Measurement). *A measurement function $l^{measure}$ is a function that, given a prefix trace σ_1 and a postfix trace σ_2 produces some measurement $l^{measure}(\sigma_1, \sigma_2)$, e.g., the remaining time until completion. Formally, $l^{measure} \in (\mathcal{C} \times \mathcal{C}) \rightarrow \mathcal{M}$ where $\mathcal{C}$ is the set of possible traces and $\mathcal{M}$ is the set of possible measurement values (e.g., some time duration).*

In principle, different measurement functions can be used. Thus far, we focused on the remaining time until completion. If this is the thing we want to predict, the following measurement function should be used:

[3] This is the best guess we can make as the first event is at time 0 and the last one is at time 18.

$$l^{measure}_{remaining}(\sigma_1, \sigma_2) = \begin{cases} 0 & \text{if } \sigma_2 = \langle\rangle \\ max_T(\sigma_2) - min_T(\sigma_2) & \text{if } \sigma_1 = \langle\rangle \text{ and } \sigma_2 \neq \langle\rangle \\ max_T(\sigma_2) - max_T(\sigma_1) & \text{if } \sigma_1 \neq \langle\rangle \text{ and } \sigma_2 \neq \langle\rangle \end{cases}$$

where $max_T(\sigma) = max\{\overline{e} \mid e \in \sigma\}$ and $min_T(\sigma) = min\{\overline{e} \mid e \in \sigma\}$. To illustrate $l^{measure}_{remaining}$ consider the first trace in Table 2 after executing the first two activities. The full trace ($\langle A^{00}, B^{06}, C^{12}, D^{18}\rangle$) is split into $\sigma_1 = \langle A^{00}, B^{06}\rangle$ and $\sigma_2 = \langle C^{12}, D^{18}\rangle$. For this particular situation $l^{measure}_{remaining}(\sigma_1, \sigma_2) = max_T(\langle C^{12}, D^{18}\rangle) - max_T(\langle A^{00}, B^{06}\rangle) = 18 - 6 = 12$.

As mentioned before, different measurement functions can be used. For example the function that considers the time that has already elapsed:

$$l^{measure}_{elapsed}(\sigma_1, \sigma_2) = \begin{cases} 0 & \text{if } \sigma_1 = \langle\rangle \\ max_T(\sigma_1) - min_T(\sigma_1) & \text{if } \sigma_1 \neq \langle\rangle \end{cases}$$

Note that this function has no predictive value. It is merely added to show another example. The total time can be measured as follows:

$$l^{measure}_{total}(\sigma_1, \sigma_2) = \begin{cases} 0 & \text{if } \sigma_1;\ \sigma_2 = \langle\rangle \\ max_T(\sigma_1;\ \sigma_2) - min_T(\sigma_1;\ \sigma_2) & \text{if } \sigma_1;\ \sigma_2 \neq \langle\rangle \end{cases}$$

It is easy to see that $l^{measure}_{total}(\sigma_1, \sigma_2) = l^{measure}_{elapsed}(\sigma_1, \sigma_2) + l^{measure}_{remaining}(\sigma_1, \sigma_2)$. One can also measure the time spent in a particular state:

$$l^{measure}_{sojourn}(\sigma_1, \sigma_2) = \begin{cases} 0 & \text{if } \sigma_1 = \langle\rangle \text{ or } \sigma_2 = \langle\rangle \\ min_T(\sigma_2) - max_T(\sigma_1) & \text{if } \sigma_1 \neq \langle\rangle \text{ and } \sigma_2 \neq \langle\rangle \end{cases}$$

Note that $l^{measure}_{sojourn}(\sigma_1, \sigma_2)$ is included in $l^{measure}_{elapsed}(\sigma_1, \sigma_2)$.

The functions listed above are all related to the duration of a process instance. It is also possible to provide completely other measurement functions using the principle of knowing both the past σ_1 and the future σ_2 of earlier instances.

- One can define a function that determines whether some activity x will take place in the future. Such as measurement function can be used to predict the probability of certain desirable or undesirable effects.
- One can define a function that measures the time until a particular activity x is executed. This is useful if the process has so-called milestones and one is interested in the moment the milestone will be reached.
- One can have functions associated to costs, e.g., "What will the total cost of a case be?".
- One can have functions related to service level agreements, e.g., "Will the case be finished in four weeks?".
- One can have functions related to resources, e.g., "Will a particular resource be used for this case?".
- Etc.

The above examples show that our approach allows for all kinds of measurements and that there are many interesting application scenarios. However, in the remainder we focus on $l^{measure}_{remaining}$, i.e., the time until completion.

Using a log, a transition system, and a measurement function $l^{measure}$, we can annotate the transition system as described before.

Definition 7 (Annotated transition system). *Let $L \subseteq \mathcal{C}$ be an event log and $TS = (S, E, T)$ a transition system obtained based on a state representation function l^{state} and an event representation function l^{event}. For a particular measurement function $l^{measure} \in (\mathcal{C} \times \mathcal{C}) \rightarrow \mathcal{M}$, we construct an annotation $A \in S \rightarrow \mathbb{B}(\mathcal{M})$ where for any $s \in S$:*[4]

$$A(s) = \sum_{\sigma \in L} \sum_{\substack{0 \leq k \leq |\sigma| \\ s = l^{state}(hd^k(\sigma))}} \left[l^{measure}(hd^k(\sigma), tl^{|\sigma|-k}(\sigma)) \right]$$

(S, E, T, A) is an annotated transition system parameterized by L, l^{state}, l^{event}, and $l^{measure}$.

Function A attaches a bag of measurements to each state, i.e., $A(s)$ is a multi-set. Note that the two $\sum$ operators range over all prefixes that correspond to a particular state s. Then for each prefix that maps onto s, a measurement $l^{measure}(hd^k(\sigma), tl^{|\sigma|-k}(\sigma))$ is added to corresponding multi-set.

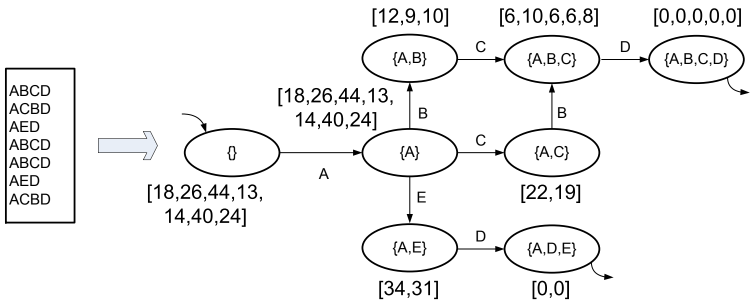


Fig. 5. The annotated transition system based on the log shown in Table 2 using $l^{measure}_{remaining}$.

Figure 5 shows an annotated transition system. It is based on the log shown in Table 2. Some example annotations are: $A(\emptyset) = [18, 26, 44, 13, 14, 40, 24]$,

[4] Note that $\left[l^{measure}(hd^k(\sigma), tl^{|\sigma|-k}(\sigma)) \right]$ denotes a multi-set containing precisely one element (a particular measurement). Using the two $\sum$ operators multi-sets are joined into larger multi-sets to collect all relevant measurements for a particular state s.

$A(\{A,E\}) = [34,31]$, $A(\{A,B,C\}) = [6,10,6,6,8]$, and $A(\{A,B,C,D\}) = [0,0,0,0,0]$. Note that the annotated transition system is completely determined by choosing L, l^{state}, l^{event}, and $l^{measure}$. The annotated transition system Figure 5, is completely parameterized by (a) the log shown in Table 2, (b) $l^{state}(\sigma) = set(par(\alpha(\sigma)))$, (c) $l^{event}(e) = prop_A(e)$, and (d) $l^{measure}(\sigma_1,\sigma_2) = l^{measure}_{remaining}(\sigma_1,\sigma_2)$.

4.2 Predictions

Next we show how an annotated transition system (like the one in Figure 5) can be used for making predictions. Consider a new case N that did not yet finish. The partial trace observed so far is $\sigma_N = \langle A^{85}, E^{95}\rangle$ (using the shorthand notation also used in Table 2), i.e., activity A occurred at time 85 and E occurred at time 95. $l^{state}(\sigma_N) = \{A,E\}$. For this state we have two earlier measurements: 31 and 34 ($A(l^{state}(\sigma_N)) = [34,31]$). Hence, the best prediction for the remaining time until completions seems to be the average of 31 and 34, i.e., 32.5. Therefore, it can be predicted that the completion time will be $95 + 32.5 = 127.5$. This example shows that we need a prediction function that converts a multi-set of measurements into a single value.

Definition 8 (Prediction function). *A prediction function is a function that, given a bag a measurements produces some prediction, e.g., the average. Formally, $predict \in \mathbb{B}(\mathcal{M}) \rightarrow \mathcal{M}$, i.e., for some bag of measurements b, $predict(b)$ returns some prediction.*

An obvious choice for *predict* is simply taking the average. However, before defining prediction functions we stress that the bag $b = A(s)$ for a particular state s should be seen as a *sample*. In statistics, a sample is a (hopefully) representative subset of an unknown population. Let us assume that $b = [b_1, b_2, \ldots, b_n]$, i.e., the n measurements linked to state s are taken as a sample. The sample mean is defined as follows: $\overline{b} = \frac{\sum_{i=1}^{n} b_i}{n}$. The sample mean is a good estimator of the population mean, i.e., $\overline{b}$ is the best guess for real expected value. Based on this we define $predict_{average}(b) = \overline{b}$ as a prediction function.

If the sample $b = [b_1, b_2, \ldots, b_n]$ is taken as a population, the sample variance is $s_n^2 = \frac{\sum_{i=1}^{n}(b_i - \overline{b})^2}{n}$. If s_n^2 is small, then the values in the sample are closer together. If s_n^2 is large, then the values in the sample are further apart. $s^2 = \frac{\sum_{i=1}^{n}(b_i - \overline{b})^2}{n-1}$ is an unbiased estimator of the population variance, i.e., it is the best guess for the real variance knowing that we have only seen a subset of the whole population. Note that s_n^2 and s^2 converge for larger values of n.

The square root of the variance, called the *standard deviation*, has the same unit as the original variable and, for this reason, is easier to interpret. Therefore, $predict_{stdev}(b) = \sqrt{s^2}$ is another example of a prediction function.

Other prediction functions can be used for the measurements, for example $predict_{min}(b) = min\{b_1, b_2, \ldots, b_n\}$ or $predict_{max}(b) = max\{b_1, b_2, \ldots, b_n\}$.

Note that these are not really estimators for the whole population, but only for the sample. Nevertheless, these functions provide interesting information, especially if many measurements are linked to the states. Based on a particular prediction function *predict*, we can define the notion of a *prediction*.

Definition 9 (Prediction). *Let $L \subseteq \mathcal{C}$ be an event log and (S, E, T, A) an annotated transition system parameterized by L, l^{state}, l^{event}, and $l^{measure}$. Moreover, let $predict \in \mathbb{B}(\mathcal{M}) \rightarrow \mathcal{M}$ be a prediction function. For any partial trace σ_N, the* predicted value *is $predict(A(l^{state}(\sigma_N)))$ if $l^{state}(\sigma_N) \in S$.*

As the above definition shows, a prediction for a running instance is made by looking up the state corresponding to the partial trace σ_N. For this state there have been a number of measurements and using an appropriate prediction function a prediction is made. Note that it is only possible to make a meaningful prediction if the calculated state is in the annotated transition system (i.e. $l^{state}(\sigma_N) \in S$) and there are a reasonable number of measurements for this state.

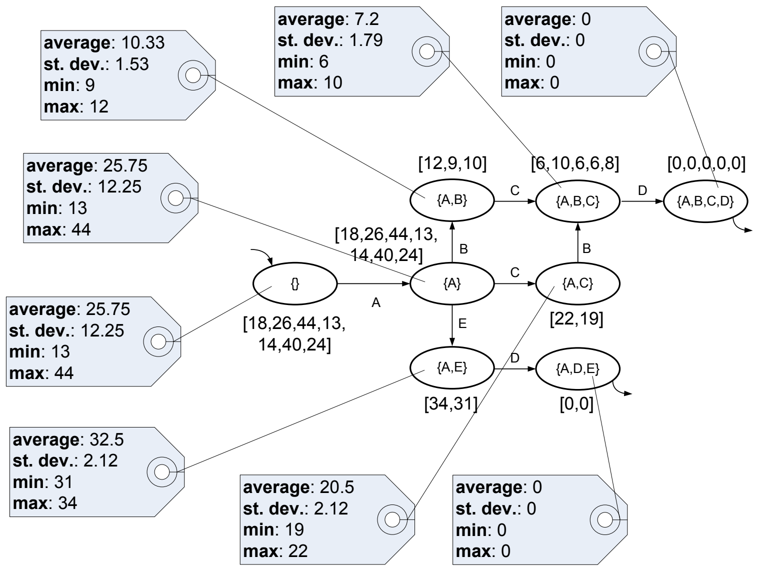


Fig. 6. The annotated transition system showing per state: the average, standard deviation, minimum, and maximum remaining time until completion.

Figure 6 shows per state predictions based on $predict_{average}$, $predict_{min}$, $predict_{max}$, $predict_{stdev}$. For example, process instances that have completed activities A, B, and C (but not yet D) have an expected remaining processing time of 7.2. At the start, when no information is available yet, the predicted remaining processing is 25.75.

5 Implementation

The work presented in this paper is supported by various plug-ins of our process mining tool *ProM* [1, 2, 21]. The first version of ProM was released in 2004. The initial goal of ProM was to unify process mining efforts at Eindhoven University of Technology and other cooperating groups [4]. Traditionally, most analysis tools focusing on processes are restricted to *model-based analysis*, i.e., a model is used as the starting point of analysis. Such analysis is only useful if the model reflects reality. Process mining techniques use *event logs* as input, i.e., information recorded by systems ranging from information systems to embedded systems. Hence the starting point is not (only) some model, but the "observed reality". Since predictions are also based on observations, like other process mining techniques, ProM is a good basis to implement the ideas presented in this paper.

ProM is open source and uses a plug-able architecture, e.g., people can add new process mining techniques by adding plug-ins without spending any efforts on the loading and filtering of event logs and the visualization of the resulting models. One of the first plug-ins was the plug-in implementing the α-algorithm [5], i.e., a technique to automatically derive Petri nets from event logs. Currently, ProM provides 274 plug-ins.[5] To facilitate a basic understanding of the scope and architecture of ProM, we briefly describe the six types of plug-ins supported by ProM:

- *Mining plug-ins* implement some mining algorithm, e.g., the α-miner to discover a Petri net, the FSM miner to discover a transition system, or the social network miner to discover a social network.
- *Export plug-ins* implement some "save as" functionality for specific objects in ProM. For example, there are plug-ins to save Petri nets, EPCs, social networks, YAWL models, spreadsheets, etc. often also in different formats (PNML, CPN Tools, EPML, AML, etc.).
- *Import plug-ins* implement an "open" functionality for specific objects, e.g., load instance-EPCs from ARIS PPM or BPEL models from WebSphere.
- *Analysis plug-ins* which typically implement some property analysis on some mining result. For example, for Petri nets there is a plug-in which constructs place invariants, transition invariants, and a coverability graph. However, there are also analysis plug-ins to compare a log and a model (i.e., conformance checking) or a log and an LTL formula. Moreover, there are analysis plug-ins related to performance measurement (e.g., projecting waiting times onto a Petri net).
- *Conversion plug-ins* implement conversions between different data formats, e.g., from EPCs to Petri nets or from Petri nets to BPEL.
- *Log filter plug-ins* implement different ways of "massaging" the log before applying process mining techniques. For example, there are plug-ins to select

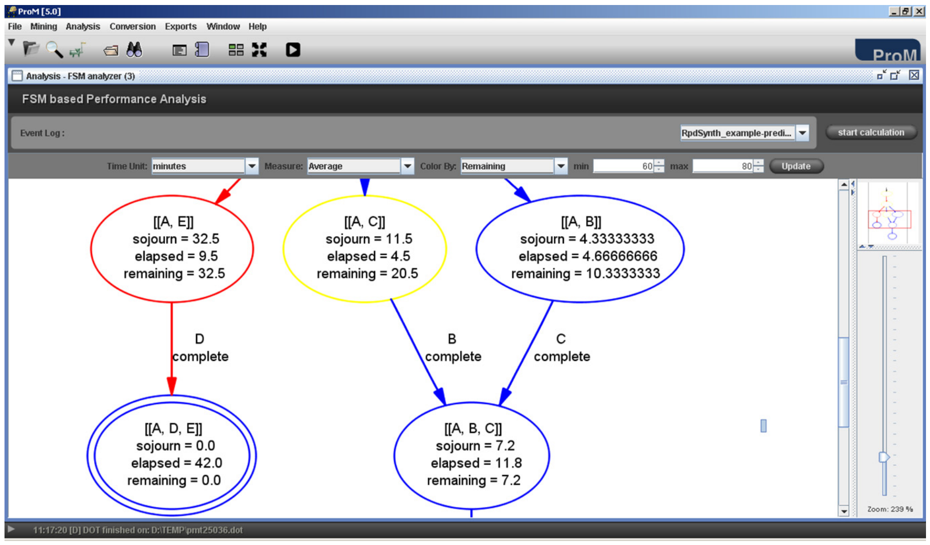
[5] This paper uses the ProM Nightly Build of 9-1-2009. ProM can be downloaded from www.processmining.org.

different parts of the log, to abstract from infrequent behavior, clean the log by removing incomplete cases, etc.

In this paper we do not elaborate on the architecture and implementation framework for plug-ins (for this we refer to earlier papers [1,2,21]). Instead we focus on the new prediction functionality.

First of all, ProM provides the so-called *FSM Miner* plug-in [3] to extract a transition system from an event log. The FSM Miner is an example of mining plug-in in the classification given above. The basic functionality of this plug-in is to create a transition system based on an event log, i.e., it implements Definition 5 and supports the abstractions presented in Section 3.5. In fact, even more abstractions and perspectives are supported, e.g., also data elements, resources, transactional information, etc. can be used to build a transition system. As shown in [3], the FSM Miner can be used for various purposes. For example, using the Theory of Regions, the transition system can be converted into a Petri net or some other higher-level representation. However, the FSM Miner does not provide any functionality directly related to prediction.

For this paper, we implemented another plug-in called the *FSM Analyzer* which takes a transition system and an event log as input. The output of the FSM Analyzer is a transition system extended with information useful for predictions. For example, the FSM Analyzer can produce results such as shown Figure 6. The FSM Analyzer is an example of an analysis plug-in in the classification given earlier. Figure 7 shows a screenshot of the FSM Analyzer.



Event Log : RpdSynth_example-predi...

Time Unit: minutes Measure: Average Color By: Remaining min 60 max 80

[[A, E]]
sojourn = 32.5
elapsed = 9.5
remaining = 32.5

[[A, C]]
sojourn = 11.5
elapsed = 4.5
remaining = 20.5

[[A, B]]
sojourn = 4.33333333
elapsed = 4.66666666
remaining = 10.3333333

[[A, D, E]]
sojourn = 0.0
elapsed = 42.0
remaining = 0.0

[[A, B, C]]
sojourn = 7.2
elapsed = 11.8
remaining = 7.2

```
11:17:20 [D] DOT finished on: D:\TEMP\pmt25036.dot
```

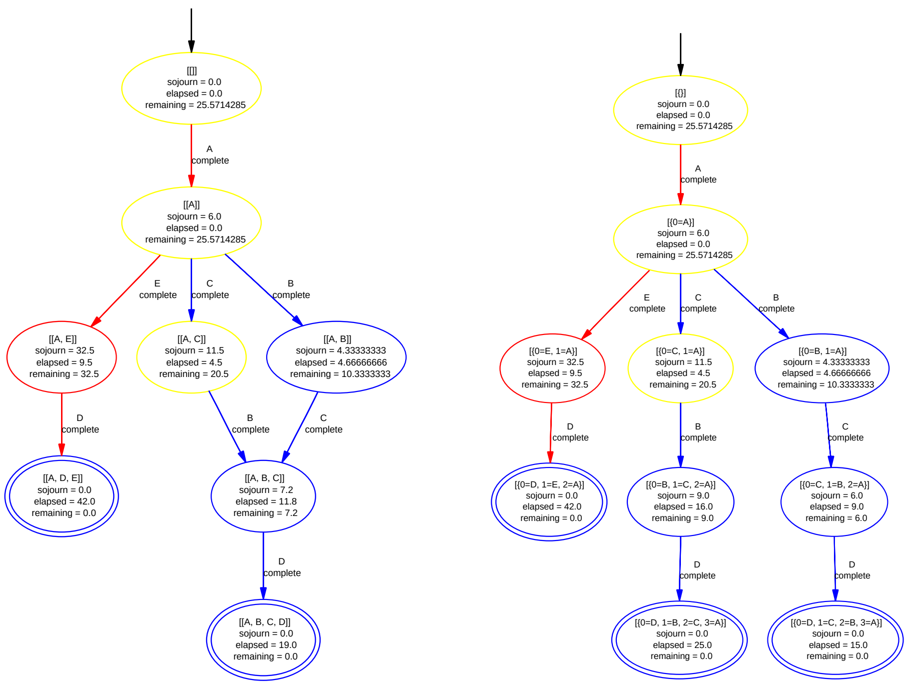
Fig. 7. Screenshot of the FSM Analyzer showing some results for the log described in Table 2.

The FSM Analyzer visualizes the transition system and by default shows information about *elapsed times* (i.e., the average time to reach a particular state based on $l^{measure}_{elapsed}$), *sojourn times* (i.e., the average time spent in a particular state based on $l^{measure}_{sojourn}$), and *remaining times* (i.e., the average time to reach the end from this state $l^{measure}_{remaining}$). The user can select the time unit, the measurement function, and the coloring function. The measurement function is the implementation of *predict* (cf. Definition 8) and currently average, variance, minimum, maximum, frequency, sum, standard deviation, and median are supported. The coloring function determines the coloring of arcs and nodes. In Figure 7 the colors are based on the remaining time until completion. The nodes indicated in red represent states from which the predicted time until completion is long. The nodes indicated in blue represent states from which the predicted time until completion is short. The nodes that are yellow correspond to states which fall in-between these two categories. A similar coloring is applied to arcs, e.g., if it typically takes a long time to move from one node to another, then the corresponding arc is colored red. The coloring can be customized by setting thresholds, etc.

The result of the FSM Analyzer can be used to make predictions at run-time. For this we use the same approach as the recommendation service [26] and the earlier prediction service [11]. The basic idea is that some run-time environment supplies a partial trace and waits for a recommendation or prediction as described in Definition 9. Note that this only requires a lookup in the result provided by the FSM Analyzer. To realize this, the link between DECLARE [19] and PROM described in [26] can be used. DECLARE is a workflow management system providing more flexibility than the traditional systems by supporting various declarative languages (DecSerFlow, ConDec, etc.) rather than some fixed procedural language [19]. In [26] it is shown how PROM can be used to recommend the next step based on the analysis of historic data. Although the recommendation service described in [26] is a very specific application using predictions, a similar interface can be used for the broad spectrum of predictions based on the output of the FSM Analyzer (also see [8]).

In this paper, we do not elaborate on the link between PROM and DECLARE. Instead we refer to [8,26]. Moreover, we would like to *stress that the idea and implementation are not specific for* DECLARE. Any workflow management system can be taken and extended with prediction functionality based on the FSM Analyzer. The only functionality required is that the workflow management system stores event logs and is able to supply a partial trace when it needs a prediction.

To further illustrate the functionality of the FSM Analyzer, we revisit our running example. Figure 8 shows two transition systems derived from Table 2 using PROM's FSM Miner. These are the same transition systems as shown in Figure 3, but now automatically generated by PROM. Figure 8(a) is based on the set abstraction and Figure 8(b) uses the sequence abstraction. Both only consider the activity names and no other properties. The FSM Analyzer has been applied to both transition systems resulting in the two diagrams depicted



(a) Predictions based on the set abstraction

(b) Predictions based on the sequence abstraction

Fig. 8. Two transition systems with prediction information based on the same event log.

in Figure 8. Now let us assume that we have a new process instance N for which we want to make a prediction. The partial trace observed so far is $\sigma_N = \langle A^{20}, B^{30}, C^{40} \rangle$ (using the shorthand notation also used in Table 2). Based on Figure 8(a) the predicted remaining time is 7.2 while based on Figure 8(b) the predicted remaining time is 6.0. So using the set abstraction, the predicted end time is 47.2, while using the sequence abstraction, the predicted end time is 46.0. This example nicely illustrates that the prediction depends on the abstractions selected when generating the transition system. Both predictions are "correct", however, Figure 8(a) assumes that the order of past activities does not matter, while Figure 8(b) assumes that the order is relevant. Note that the predictions after just executing $\langle A^{20} \rangle$ or $\langle A^{20}, B^{30} \rangle$ are identical for both transition systems because A is always first no matter which abstraction is chosen.

The fact that different predictions are possible based on the abstraction selected, illustrates the need to assess the quality of predictions.

6 Quality of Predictions

As shown thus far, our approach allows for the prediction of various things including the remaining time until completion. Depending on the abstraction chosen, different predictions are possible. Moreover, under certain circumstances one prediction may be less reliable than another. Therefore, this section elaborates on the quality of predictions.

Intuitively, the quality of the prediction depends on the number of measurements per state. If there are only few observations, the predictive value is limited. Moreover, if the individual measurements are very different, then the predictive value seems less than in the situation were all measurements are similar. As indicated in Section 4.2, the bag of measurements $A(s)$ associated to some state s can be seen as a random sample. Let $A(s) = b = [b_1, b_2, \ldots, b_n]$, i.e., n observations have been made: $b_1, b_2, \ldots, b_n$. Now we need to predict the next observation b_{n+1} based on $b_1, b_2, \ldots, b_n$. Let b^r_{n+1} be the real value and b^p_{n+1} be the predicted value. Clearly the goal is to minimize the error $\mid b^r_{n+1} - b^p_{n+1} \mid$.

We will look at the quality of predictions from different angles. First, we try to compute a confidence interval for the true average. Second, we compute a confidence interval for the real value assuming that we know the true average. Finally, we discuss cross-validation as a means to assess the quality of a prediction in an experimental setting.

6.1 Predicting the Average Right

Let $A(s) = b = [b_1, b_2, \ldots, b_n]$ be the bag of measurements and let us assume that these are all sampled from some random variable. We assume that these samples are independent, i.e., b_i does not depend on b_j, and the next realization b^r_{n+1} does not depend on earlier samples. Based on $b_1, b_2, \ldots, b_n$, the best estimator for b^r_{n+1} is $\overline{b}$, i.e., the sample mean. However, if there are only a few measurements, then $\overline{b}$ may be very different than the true expected value μ of b^r_{n+1}. Therefore, we first try to establish a *confidence interval* for the *true expected value* of b^r_{n+1}. There are various ways of calculating such a confidence interval. Here, we sketch two of the most basic approaches.

First, let us assume that we have many measurements (say $n \geq 30$). Because $\overline{b}$ is the average of a large number of independent measures, we can assume that $\overline{b}$ is approximately normally distributed (cf. law of large numbers). From this fact, we deduce the probability that the true expected value μ lies within a so-called confidence interval. Given the sample mean $\overline{b}$ and the sample standard deviation s, the real value μ conforms with confidence $(1 - \alpha)$ to the following equation: $\overline{b} - \frac{s}{\sqrt{n}}\, z(\frac{\alpha}{2}) < \mu < \overline{b} + \frac{s}{\sqrt{n}}\, z(\frac{\alpha}{2})$ where $z(\frac{\alpha}{2})$ is defined as follows. If Z is a standard normally distributed random variable, then $\mathbb{P}[Z > z(x)] = x$. The value α represents the unreliability, that is the chance that μ does not conform to the equation. Some example values for the z function: $z(0.001) = 3.090$, $z(0.005) = 2.576$, $z(0.01) = 2.326$, $z(0.05) = 1.645$, and $z(0.1) = 1.282$. The interval $\left[\overline{b} - \frac{s}{\sqrt{n}}\ z(\frac{\alpha}{2}), \overline{b} + \frac{s}{\sqrt{n}}\ z(\frac{\alpha}{2})\right]$ is also called the $(1-\alpha)$-confidence interval for the estimated value μ.

If there are fewer measurements, then the above approach cannot be used because it depends on the law of large numbers. Therefore, we mention a second approach that can still be applied if there are just a few observations. However, for this approach it needs to be assumed that the original distribution is normally distributed. In this case the *Student's t-distribution* can be used to calculate the confidence intervals. Let $[b_1, b_2, \dots, b_n]$ be again the bag of measurements with a sample mean $\overline{b}$, sample deviation s. For confidence $(1-\alpha)$ the following confidence interval can be deduced: $\left[\overline{b} - \frac{s}{\sqrt{n}}\ t_{n-1}(\frac{\alpha}{2}), \overline{b} + \frac{s}{\sqrt{n}}\ t_{n-1}(\frac{\alpha}{2})\right]$ where $t_v(x)$ is the critical value of a Student's t-distribution, also called t-distribution, with v degrees of freedom.

In our running example, there are only a few observations. Therefore, we apply the second approach. As Figure 6 shows there are only 5 measurements linked to state $\{A, B, C\}$: $A(\{A, B, C\}) = b = [6, 10, 6, 6, 8]$. Hence, $n = 5$, $\overline{b} = 7.2$ and $s = 1.79$. Let us assume that we are interested in a 90% confidence interval, i.e., $\alpha = 0.1$. By simply applying the formula, we get $\left[\overline{b} - \frac{s}{\sqrt{n}}\ t_{n-1}(\frac{\alpha}{2}), \overline{b} + \frac{s}{\sqrt{n}}\ t_{n-1}(\frac{\alpha}{2})\right]$ $= \left[7.2 - \frac{1.79}{\sqrt{5}}\ t_{5-1}(0.05), 7.2 + \frac{1.79}{\sqrt{5}}\ t_{5-1}(0.05)\right]$. Since $t_4(0.05) = 2.13$, the 90% confidence interval for the completion time in state $\{A, B, C\}$ is $[5.5, 8.9]$. So, assuming a normal distribution, the expected flow time for partial cases in state $\{A, B, C\}$ is with 90% confidence between 5.5 and 8.9. This simple example shows that by making some basic assumptions, the quality of the predicted average can be quantified.

6.2 Estimating the Quality of a Prediction

We just showed that under certain assumptions, we can establish a confidence interval for the true expected value of b^r_{n+1}. However, even if the confidence interval is very narrow, the error $\mid b^r_{n+1} - b^p_{n+1} \mid$ may still be substantial.

Suppose that $b_1, b_2, \dots, b_n$ are mutually independent and sampled from a probability distribution with mean μ and variance σ^2 and that the next realization will be sampled from the same probability distribution. As n becomes larger, the confidence interval will become more narrow with upper and lower bounds close to μ. However, the expected average error $\mid b^r_{n+1} - b^p_{n+1} \mid$ remains roughly σ.

Note that in general we do not know μ and σ^2. However, when we have sufficient measurements, we can approximate them as described in Section 6.1. Moreover, let us assume that the $b_1, b_2, \dots$ are mutually independent and sampled from a normal distribution with mean μ and variance σ^2. Under these assumptions we can state that with probability 0.683 the next measurement will be between $[\mu - \sigma, \mu + \sigma]$, with probability 0.954 the next measurement will be between $[\mu - 2\sigma, \mu + 2\sigma]$, and with probability 0.997 the next measurement will be between $[\mu - 3\sigma, \mu + 3\sigma]$. This illustrates that for a known distribution, it is possible to not only supply a prediction but also provide information about the quality of the prediction.

6.3 Cross-Validation

In the previous subsections, we made assumptions about the underlying probability distribution. For example, we assumed that the measurements are independent and that the shape of the distribution is known (e.g., normal distribution). However, it is possible to measure the quality in a more direct manner. One way would be to simply take the log, derive the annotated transition system and predictions and then in a second run compare the predicted values with the real values. The term *Mean Squared Error* (MSE) is used to quantify the difference between the predicted and real values. Suppose that for a particular state s with $A(s) = b = [b_1, b_2, \ldots, b_n]$, we predict $predict(b)$. The MSE for this state is $MSE = \frac{\sum_{i=1}^{n}(b_i - predict(b))^2}{n}$. We can also take the Root Mean Squared Error (RMSE), by taking the square root of MSE.

Although the principle described above is appealing, there is one problem. The same data set is used both for making the predictions and evaluating the quality of the predictions. This is undesirable. For example, if $n = 1$, then $MSE = 0$ by definition. This is why *cross-validation* is needed. Cross-validation is the statistical practice of partitioning a sample data set into two subsets such that the analysis is initially performed on one subset (the "training set") while the other subset is used for validation (the "test set").

The most basic form of cross-validation, *holdout validation*, is to randomly select samples to form the test set, and the remaining observations are retained as the training data. More involved is *K-fold cross-validation* where the whole data set is partitioned into K sets of equal size. Of the K sets, one set is selected as test set, while the union of the other $K - 1$ sets is used as training set. The cross-validation process is then repeated K times (the folds), with each of the K sets used exactly once as the validation data. The K results from the folds then can be averaged (or otherwise combined) to produce a single estimation. The advantage of this method is that all observations are used for both training and validation, and that it is possible to get insight into the reliability of the validation itself. A commonly used value for K is 10.

Note that for cross validation, there is no need to make assumptions about the underlying distribution. ProM supports this kind of cross validation through the so called *FSM Evaluator*. This plug-in takes (a) a transition system with predictions based on one log and (b) another event log, and then calculates the MSE and other statistics for each state. We will use this plug-in in the next section to evaluate our approach.

7 Experiments

In this section, we demonstrate the applicability of our approach using a synthetic logs obtained via simulation and two real-life case studies. For the case studies we use two different processes within two different Dutch municipalities.

7.1 Synthetic Example

For our first experiment we use an event log of a reviewing process. This log was obtained through simulation. We first use this synthetic log to allow for a controlled experiment where it is clear what the desired outcome should be. This way we can validate our approach.

Using CPN Tools [17] we have modeled the reviewing process in terms of a so-called colored Petri net. From ProM it is possible to generate CPN models [22] and from CPN Tools it is possible to generate event logs for ProM [18]. Hence, there is a bidirectional coupling of CPN Tools and ProM.

The stimulated reviewing process starts with inviting reviewers. Then the reviewers can return their reviews. However, some reviewers may not return their review resulting in a time-out event. After a while the reviews are collected and a decision is made. Based on the decision, the paper is accepted or rejected. For internal activities such as inviting reviewers, making decisions, and accepting/rejecting papers, we record a start and complete event. For external activities such as reviewing, we can only see the completion. The CPN model also contains information about timing and resources, e.g., the various internal activities are done by particular people having distinct roles. The external activities are done by a large pool of reviewers.

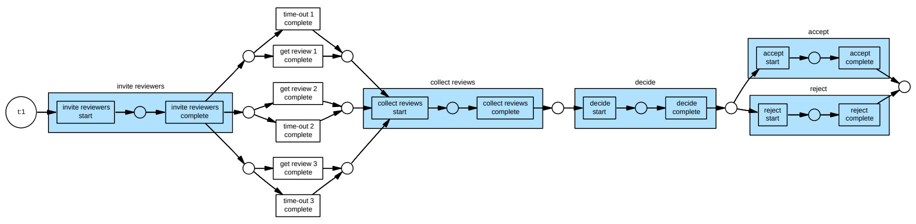
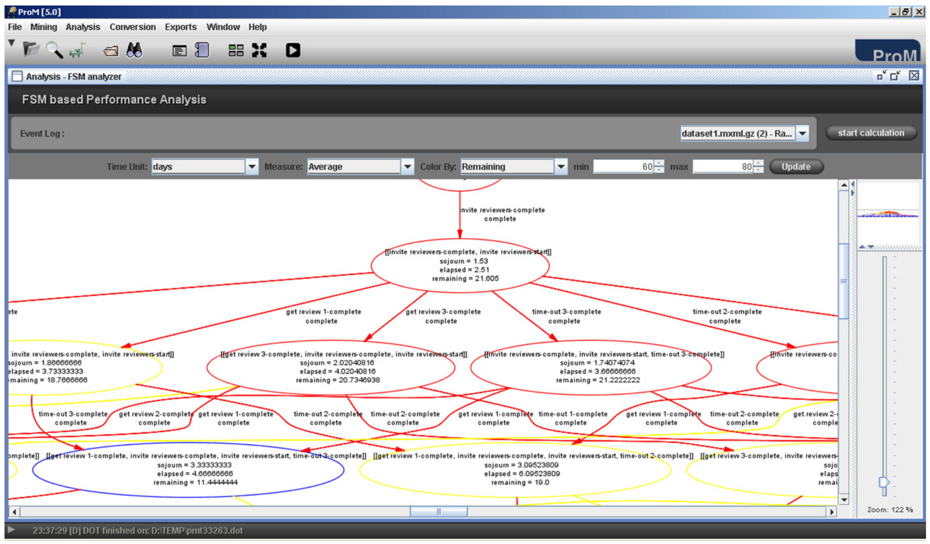


Fig. 9. Petri net discovered by ProM using the α-algorithm.

Based on the CPN model described above, we generated two event logs: a training set L_1 and a test set L_2. Both event logs have information about 200 process instances, i.e., submitted papers. To provide some insight into the process we use the α-algorithm [5] to discover a Petri net based on L_1 (i.e., the event log holding the training set). The discovered model indeed matches the model constructed using CPN Tools (cf. Figure 9). Note that the model indicates start and complete events by grouping two transitions in a single box.

It is important to note that we only show the Petri net representation of the process to provide the reader with some insight into the process. Our approach does not need a process model in terms of a Petri net or any other notation and it is not necessary to use process discovery algorithms such as the α-algorithm.

Based on L_1, we can use the FSM Miner of ProM to extract a transition system using a particular abstraction. Using the FSM Analyzer we can extend



Event Log : dataset1.mxml.gz (2) - Ra...

Time Unit: days Measure: Average Color By: Remaining min 60 max 80

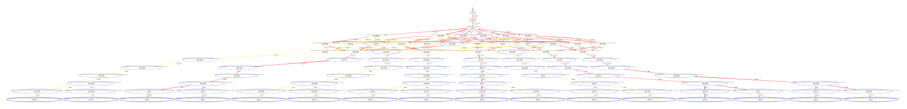
23:37:29 [D] DOT finished on: D:\TEMP\pmt33263.dot

Fig. 10. Screenshot of ProM's FSM Analyzer while making a prediction model based on an event log of 200 cases.

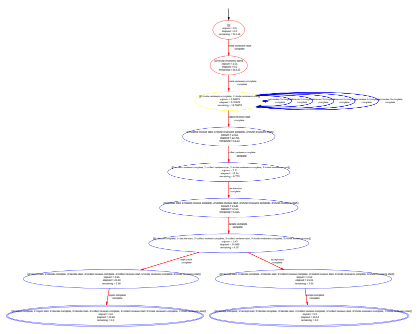
this transition system with predictive information as shown before. Figure 10 shows a screenshot of an annotated transition system based on L_1 while using a set abstraction based on all activities. For example, Figure 10 shows the state [[*invite reviewers-complete, invite reviewers-start*]].[6] As indicated in Figure 10, the predicted time until completion is 21.605 days, i.e., after completing the invitations it takes on average three weeks to complete the reviewing process. Figure 10 also shows information about the average/expected sojourn time for this state (1.53 days) and the average/expected elapsed time (2.51 days). For state {*collect reviews-start, get review 1-complete, get review 2-complete, get review 3-complete, invite reviewers-complete, invite reviewers-start*} the predicted time until completion is 11.87 days (not visible in Figure 10). This means that for cases where all reviews are sent back in time, it takes on average 12 days to make the decision, inform the author, and complete the reviewing process. For the empty state, the predicted remaining time until completion is 24.12 days. Note that this is the average total time to handle reviews from begin to end.

The screenshot shown in Figure 10 is based on the set abstraction. However, as described in Section 3.5 various abstractions are possible. Figure 11 shows four transition systems obtained using four different abstractions. Figure 11(a) shows the set abstraction already used in Figure 10. Figure 11(b) shows an abstraction where the six "time out" and "get review" activities are ignored, but the order

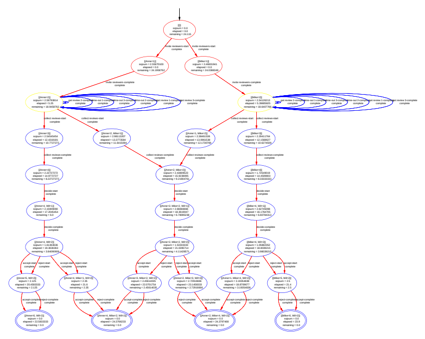
[6] This is the way the FSM Analyzer represents the set containing two events related to the start and completion of activity *invite reviewers*, i.e., [[*invite reviewers-complete, invite reviewers-start*]] denotes {*invite reviewers-complete, invite reviewers-start*}.



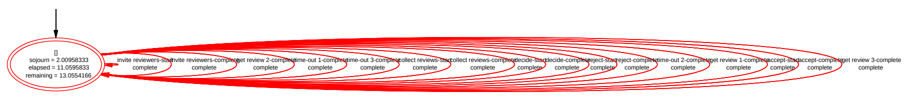
(a) Set abstraction based on all activities



(b) Sequence abstraction based on activities while filtering out the "time out" and "get review" activities



(c) Bag abstraction based on the three key resources



(d) Complete abstraction: $l^{state}(\sigma) = \emptyset$

Fig. 11. Four transition systems with prediction information based on the same event log but using different abstractions. Note that the labels are not intended to be readable. The goal is to merely show the size and shape of each transition system.

of the remaining activities is taken into account. This results in a simpler model. The transition system can be based on all kinds of properties of events, i.e., the data and resource perspectives can also be used. Figure 11(c) only looks at the resource perspective. In this particular transition system, the state of a paper is determined by which resources have worked on it. Here only the three key resources working in the editorial office are taken into account. Moreover, since a bag abstraction is used, the order is not important, but the frequency is. This results in a completely different transition systems with nodes such as $[Anne^2, Mike^2, Wil^1]$ indicating that Anne has performed two events, Mike has performed two events, and Wil has performed one event. For this state the predicted time until completion is 6.78 days. Figure 11(d) uses a full abstraction, i.e., $l^{state}(\sigma) = \emptyset$ meaning that all prefixes are mapped onto the same state. This is an interesting abstraction as it *serves as a benchmark*. This benchmark abstraction assumes that one knows nothing and simply takes the average over all events in the past. When the transition system shown in Figure 11(d) is used, the prediction for the remaining time until completion is always 13.06 days. This value is obtained by simply taking the average of all remaining times in the whole log L_1. Note that 13.06 is more than half of the average total time (which is 24.12 days). In the beginning of the process there are more events than at the end. Therefore, the average remaining time until completion is more than half of the average total time.

The four transition systems shown in Figure 11 can be used to make concrete predictions for individual cases. Suppose that the set abstraction is used (i.e., Figure 11(a)), then initially the predicted remaining time is 24.12 days. After the reviewers have been invited (i.e., the prefix is mapped onto state {*invite reviewers-complete, invite reviewers-start*}), then the predicted remaining time is reduced to 21.6 days, etc.

As discussed in Section 6, it is important to be able to asses the quality of a prediction. ProM's FSM Analyzer provides metrics such as standard deviation, etc. to determine the quality of a prediction. The standard deviation in the initial state of Figure 11(a) is 10.3. The standard deviation in state {*collect reviews-start, get review 1-complete, get review 2-complete, get review 3-complete, invite reviewers-complete, invite reviewers-start*} is 8.3. The standard deviation steadily decreases as the process progresses. This means that towards the end of the process the predictions become more reliable.

To truly evaluate the quality of the predictions and to compare the four transition systems shown in Figure 11, we use cross validation. To keep things simple, we do not perform a K-fold validation and merely test L_2 (the test set) on the model learned using L_1 (the training set). Hence, we take the four models depicted in Figure 11 and compare the predicted times with the real times for the papers in L_2. Earlier we already mentioned the terms *Mean Squared Error* ($MSE = \frac{\sum_{i=1}^{n}(b_i - \overline{b})^2}{n}$, with $\overline{b} = predict(b)$) and *Root Mean Squared Error* ($RMSE = \sqrt{MSE}$). For our evaluation we also use the *Mean Absolute Error* ($MAE = \frac{1}{n}\sum_{i=1}^{n} \mid b_i - \overline{b} \mid$) and the *Mean Absolute Percentage Error* ($MAPE = \frac{1}{n}\sum_{i=1}^{n} \frac{|b_i - \overline{b}|}{b_i}$).

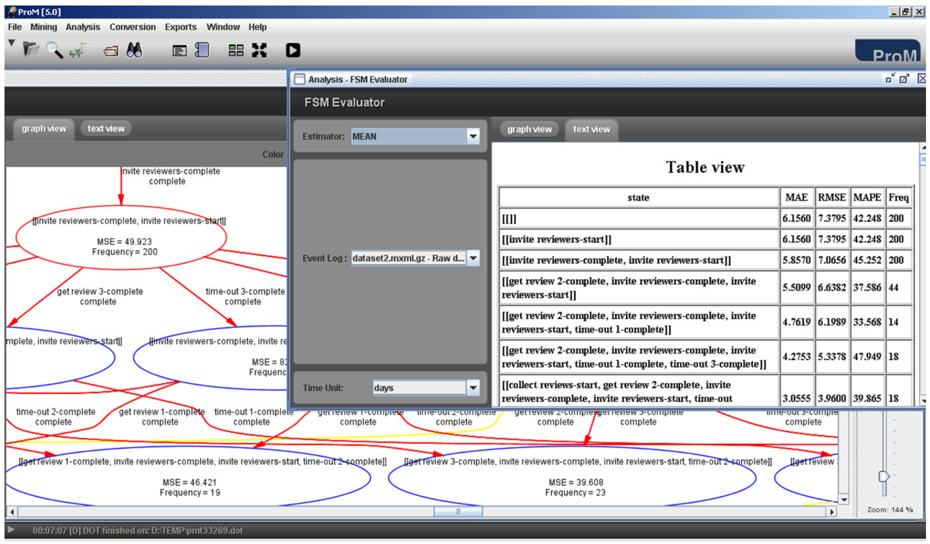

ProM [5.0]
File Mining Analysis Conversion Exports Window Help
Analysis - FSM Evaluator
FSM Evaluator
Estimator: MEAN
Event Log: dataset2.mxml.gz - Raw d...
Time Unit: days
graph view
text view
[[invite reviewers-complete, invite reviewers-start]]
MSE = 49.923
Frequency = 200
[[get review 1-complete, invite reviewers-complete, invite reviewers-start, time-out 2-complete]]
MSE = 46.421
Frequency = 19
[[get review 3-complete, invite reviewers-complete, invite reviewers-start, time-out 2-complete]]
MSE = 39.608
Frequency = 23
Zoom: 144 %
00:07:07 [D] DOT finished on: D:\TEMP\pmt33269.dot

Table view

state	MAE	RMSE	MAPE	Freq
[[]]	6.1560	7.3795	42.248	200
[[invite reviewers-start]]	6.1560	7.3795	42.248	200
[[invite reviewers-complete, invite reviewers-start]]	5.8570	7.0656	45.252	200
[[get review 2-complete, invite reviewers-complete, invite reviewers-start]]	5.5099	6.6382	37.586	44
[[get review 2-complete, invite reviewers-complete, invite reviewers-start, time-out 1-complete]]	4.7619	6.1989	33.568	14
[[get review 2-complete, invite reviewers-complete, invite reviewers-start, time-out 1-complete, time-out 3-complete]]	4.2753	5.3378	47.949	18
[[collect reviews-start, get review 2-complete, invite reviewers-complete, invite reviewers-start, time-out	3.0555	3.9600	39.865	18

Fig. 12. Screenshot of ProM's FSM Evaluator while analyzing the quality of the prediction on the test set L_2.

Figure 12 shows ProM's FSM Evaluator while cross validating the results using the second event log L_2. For state {*invite reviewers-complete, invite reviewers-start*}, MSE equals 49.923 (see graph on left-hand-side of Figure 12). The other metrics for this state are shown in tabular form (see right-hand-side of figure): MAE=5.857, RMSE=7.0656, and MAPE=45.252. Recall that the predicted time until completion from this state is 21.605 days, hence an average error of 5.857 days is quite acceptable. Figure 12 also shows that this state was evaluated 200 times. Let us now consider a later state: {*collect reviews-start, get review 1-complete, get review 2-complete, get review 3-complete, invite reviewers-complete, invite reviewers-start*}. For this state we obtain the following quality metrics: MSE=19.134, MAE=3.888, RMSE=4.3742, and MAPE=49.676. As can be expected the average error decreases when a paper is further in the process. The relative error (MAPE) increases slightly because the absolute times get smaller. The various quality metrics can also be given for the whole process, i.e., all states. In order to do this, each state is weighted by the number of times it is visited. For the set abstraction based on all activities (i.e., the transition system shown in Figure 11(a)), the overall values are MAE=4.079, RMSE=4.917, and MAPE=56.21.

Table 3 compares the four transition systems shown in Figure 11 using cross validation. All four models are learned using L_1 and evaluated using L_2. It shows that the first three models perform comparable, e.g., the average error is around 3 days. However, the fourth model does not perform as good. The average error is much higher: more than 7 days. The average time it takes to complete the

Table 3. Some results

Abstraction	MAE	RMSE	MAPE
Set abstraction based on all activities (cf. Figure 11(a))	4.079	4.917	56.21
Sequence abstraction based on activities while filtering out the "time out" and "get review" activities (cf. Figure 11(b))	4.119	4.984	56.68
Bag abstraction based on the three key resources (cf. Figure 11(c))	4.139	4.997	56.89
Complete abstraction: $l^{state}(\sigma) = \emptyset$ (cf. Figure 11(d))	7.239	8.635	120.87
Simple heuristic: half of average total flow time (12 days)	7.053	8.479	110.98

process is about 24 days. Hence, a naive approach would be to always guess $24/2 = 12$ days. If this is done, then the performance is comparable to the complete abstraction shown in Figure 11(d) (which always guesses 13.06 days). The last row in Table 3 shows the results for this heuristic.

Therefore, we can conclude that the findings in Table 3 support our approach and show that a proper abstraction easily outperforms simple heuristics. Please note that our approach does not require an a-priori model; the predictions are just learned from the training set L_1.

7.2 Case Study I: WMO Process of a Municipality

In the previous subsection, it was shown that the average error using our prediction method is much smaller than using a simple heuristic. However, this analysis was based on event logs generated by simulation. A valid question is whether we can achieve the same performance for real-life logs. Therefore, we also tested our approach using an event log from the Dutch municipality of Harderwijk. Harderwijk is a municipality and a small city in the eastern part of the Netherlands.

The event log of Harderwijk used for cross-validation contains information about 796 cases. Each case corresponds to a request to the municipality in the context of the so-called "Wet Maatschappelijke Ondersteuning" (WMO). WMO is a Dutch law regulating how municipalities should support their citizens. The purpose of WMO is to support elderly people, people having a chronic illness, handicapped citizens, etc. Through WMO citizens can request wheelchairs, household help, etc. The particular log chosen for this evaluation only concerns citizens asking for household help.

The 796 requests for household help, triggered 5187 events that were recorded over a period of 1.5 years (2007-2008). In this process 8 different types of activities were used and the average time it takes to handle a request from begin to end is 117 days with some cases taking longer than one year. See Figure 13 for the distribution of total flow times for all cases. The x-axis shows the 796

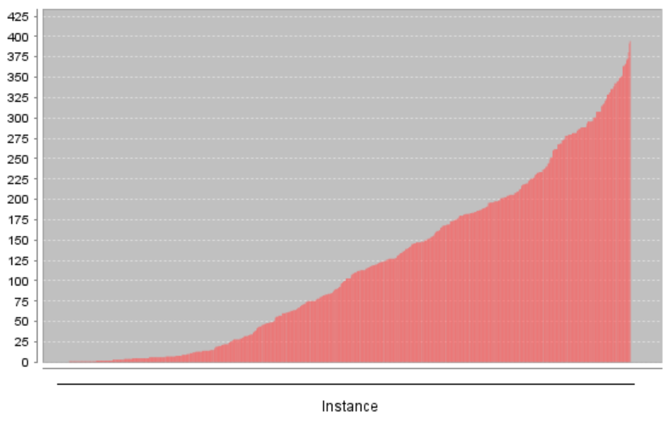


Fig. 13. The distribution of the total flow time of cases extracted using ProM. Note that some cases almost take 400 days.

cases and the y-axis shows the duration in days. Given the long flow times, it is interesting to predict the overall time.

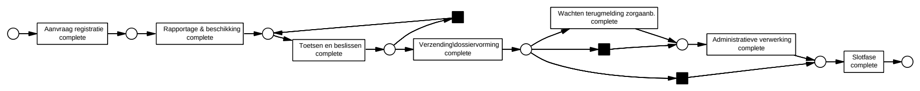
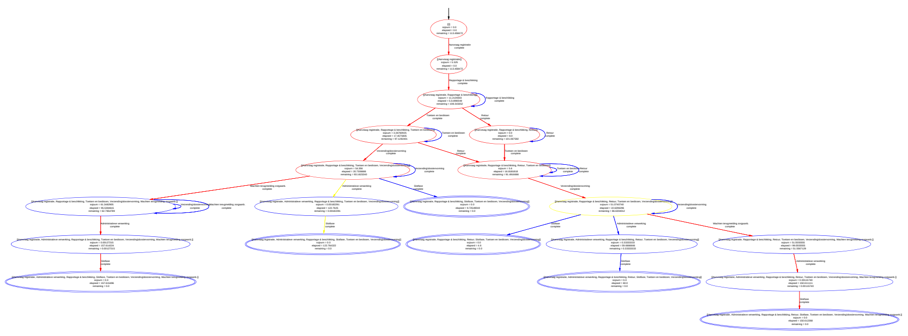

Aanvraag registratie complete
Rapportage & beschikking complete
Toetsen en beslissen complete
Verzending\dossiervorming complete
Wachten terugmelding zorgaanb. complete
Administratieve verwerking complete
Slotfase complete

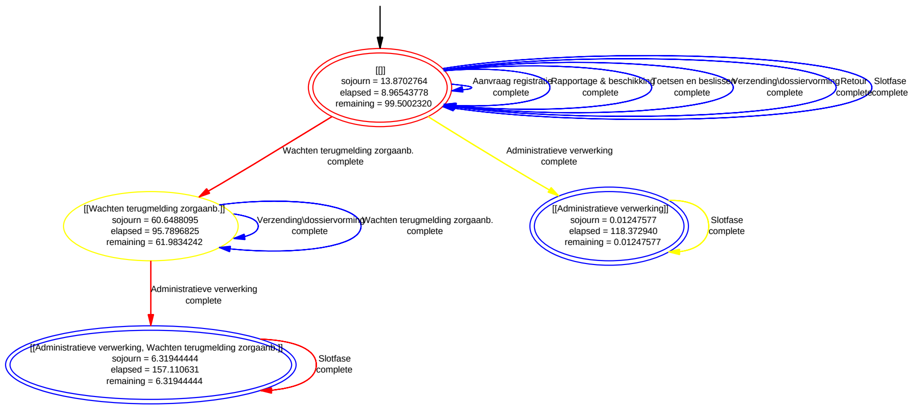
Fig. 14. A simplified representation of the process obtained by ProM.

To provide some insight into the structure of the process, we have used the heuristics miner of ProM to extract a Petri net from the event log (cf. Figure 14). Note that in this model infrequent activities have been removed and only the main flows are shown. The black transitions in the Petri net represent the skipping of activities. In this paper, we do not discuss the WMO process in detail, instead we focus on the quality of our predictions.

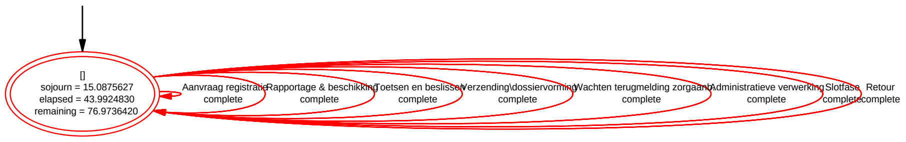
Figure 15 shows three transition systems extended with prediction information. These are all based on only half of the event log, i.e., the original log with 796 requests for household help is split into a log L_1 with 400 requests and a log L_2 with 396 requests. Event log L_1 is used as the training set and log L_2 is used as a test set. Hence, the three transition systems shown in Figure 15 are based on L_1. Figure 15(a) was obtained using the set abstraction based on all activities, i.e., a state is determined by the set of activities that have been conducted. Figure 15(b) uses the same abstraction, but now only uses two activities: "Administratieve verwerking" and "Wachten terugmelden zorgaanbieder".



(a) Set abstraction based on all activities



(b) Set abstraction based on just two activities ("Administratieve verwerking" and "Wachten terugmelden zorgaanbieder")



(c) Complete abstraction

Fig. 15. Three transition systems with prediction information based on the same event log but using different abstractions.

These two activities have been chosen because they are expected to be relevant for the time until completion. Figure 15(c) completely abstracts from all information and this extreme abstraction is again used as a benchmark.

Let us consider some example predictions based on Figure 15(a). For the initial state $\emptyset$, the predicted remaining time until completion is 113 days.[7] The first activity is always "Aanvraag registratie" and the second activity is always "Rapportage & beschikking". On average 5 days are spent in the state in-between these two activities, so after "Aanvraag registratie" the predicted remaining time until completion is 108 days. Note that after these first two steps, the process is less structured. (Recall that Figure 14 abstracts from less frequent paths/activities.)

Table 4. Some results for the municipality's WMO process

Abstraction	MAE	RMSE	MAPE
Set abstraction based on all activities (cf. Figure 15(a))	63.805	74.947	680.43
Set abstraction based on just two activities (cf. Figure 15(b))	67.329	79.034	648.80
Complete abstraction (cf. Figure 15(c))	83.469	98.158	85444.9
Simple heuristic: half of average total flow time (56.73 days)	80.823	101.342	62969.0

Let us now compare the three prediction models shown in Figure 15. We use L_2 to evaluate and compare the quality of the three models. The first two models perform better than the benchmark model. Set abstractions based on all activities or the two key activities lead to an average error of roughly 65 days. If we completely abstract from the history of a case, the average error is more than 80 days. If one takes half of the average flow time for each prediction, the average error is similar as for the model in Figure 15(c) (see last row in Table 4). Hence, also for this real log, we can conclude that simple abstractions outperform simple heuristics.

Moreover, it is interesting to note that for some states the mean average error is very small. For example, after executing "Aanvraag registratie", "Administratieve verwerking", "Rapportage & beschikking", "Toetsen en beslissen", "Verzending dossiervorming", "Wachten terugmelding zorgaanb." the mean average error of predictions based on the annotated transition system in Figure 15(c) is only 0.00163.

Note that the average error over all activities is still substantial for all models. This is due to the huge variations in flow time. Note that some cases take more than one year to be handled while other just take a few days. Hence, one cannot expect a better performance for this process with so little information. We expect that in most processes, the variation is smaller thus allowing for bet-

[7] Note that this number is based on L_1 while the average of 117 was based on $L_1 + L_2$.

ter predictions. Moreover, in most processes more data is available that can be exploited by our prediction techniques.

7.3 Case Study II: WOZ Process of Another Municipality

As a second case study, we revisit the process mentioned in Section 3.5. This case study is based on a log from another municipality and another process that deals with objections (i.e., appeals) against the real-estate property valuation or the real-estate property tax. This municipality is located in the south of the Netherlands and has a similar size as the municipality involved in the first case study (more than 40.000 inhabitants). The municipality is using eiStream workflow (formerly known as Eastman Software and today named Global 360).

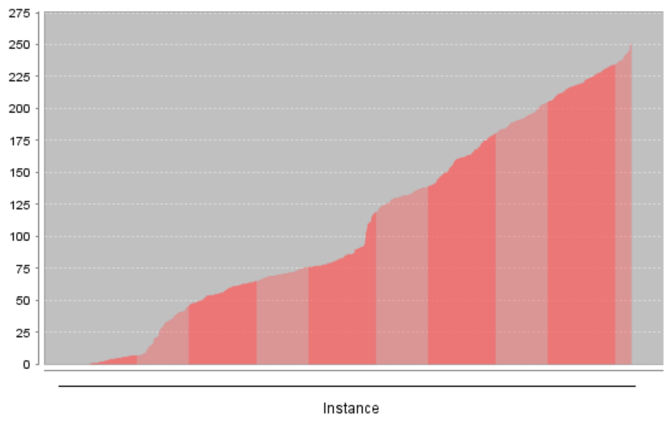
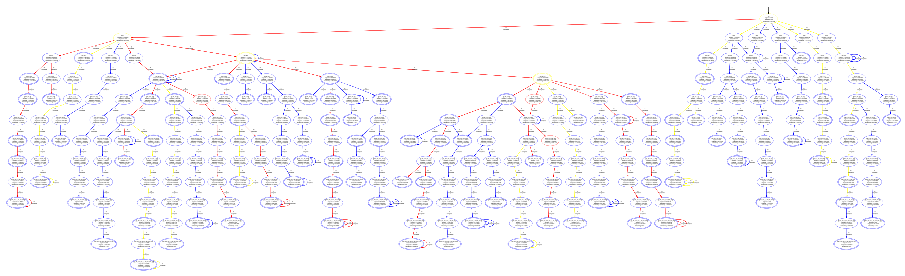


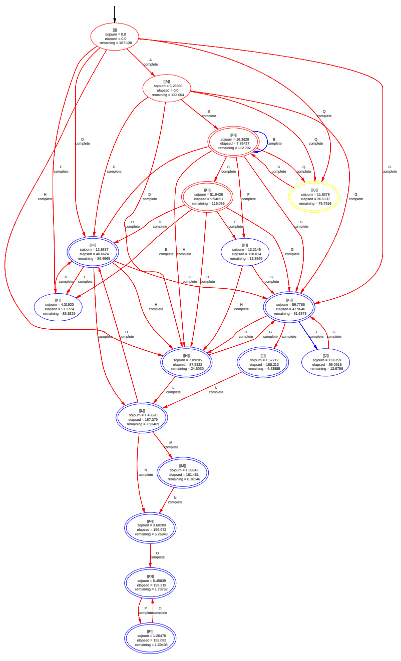
Fig. 16. The distribution of the total flow time of cases extracted using ProM. Note that some cases almost take 250 days.

This process considered in this case study is called "Bezwaar WOZ", where WOZ ("Waardering Onroerende Zaken") refers to the particular law describing regulations related to real-estate property valuation by municipalities. We used an event log with data on 1882 objections handled by the municipality. The log contains 11985 events and the average total flow time is 107 days while some cases take more than 200 days. Figure 16 shows the distribution of total flow times. The x-axis shows the 1882 cases and the y-axis shows the duration in days. Note that some cases take a very short time while others take much longer, thus making it difficult to predict the remaining time for cases in the system. As before we split the log into a training set (log L_1) and a test set (log L_2). Log L_1 contains 982 cases and log L_2 contains 900 cases.

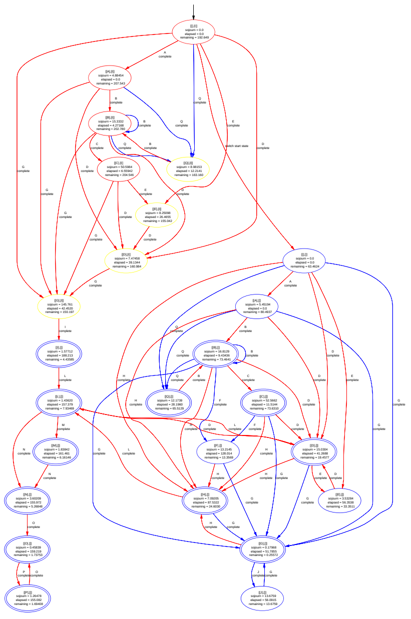
Figure 17 shows four transition systems extended with prediction information based on log L_1. The first transition system (Figure 17(a)) uses the set



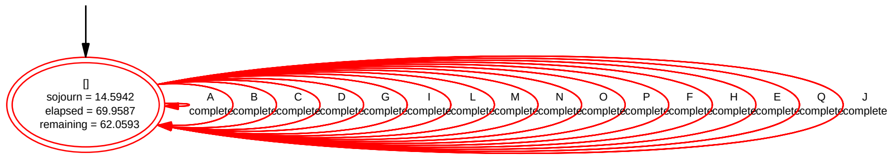
(a) Set abstraction based on all activities



(b) Set abstraction with a horizon of 1



(c) Set abstraction with a horizon of 1 and knowledge about the case



(d) Complete abstraction

Fig. 17. Four transition systems with prediction information based on the same event log but using different abstractions.

abstraction over all activities. The second one (Figure 17(a)) uses the same abstraction but now with horizon 1, i.e., only the last activity is considered. Note that these first two transition systems correspond to the ones already shown in Figure 4. The third transition system uses knowledge about the property of the case, i.e., based on this knowledge it is known whether a particular task (labeled "I") needs to be executed. Figure 17(c) shows the resulting transition system. Note that it extends Figure 17(b) with additional knowledge about the occurrence of task "I". The last abstraction (Figure 17(d)) is again the extreme case where no historic information is used. This is again used as a benchmark.

Table 5. Some results for the municipality's WOZ process

Abstraction	MAE	RMSE	MAPE
Set abstraction based on all activities (cf. Figure 17(a))	41.648	47.513	1505.07
Set abstraction based on last activity (cf. Figure 17(b))	43.080	49.666	1818.49
Set abstraction based on last activity and additional information related to the occurrence of "I" (cf. Figure 17(c))	17.129	23.550	900.07
Complete abstraction (cf. Figure 17(d))	63.391	74.965	7169.55
Simple heuristic: half of average total flow time (53.57 days)	61.750	75.505	6188.04

Table 5 shows the results for the four annotated transition systems shown in Figure 17. The last row shows again the results for simply predicting half of the average flow time. Note that the transition systems are constructed based on L_1 while the error rates in Table 5 are based on L_2. Again simple abstractions such as shown in figure 17(a) and 17(b) outperform simple heuristics. Note that the the complete abstraction and simple heuristic have a MAE of more than 60 days while set abstractions based on all activities or just the last activity have a MAE of just above 40 days. The third row of Table 5 shows the spectacular performance of the annotated transition system shown in Figure 17(c). The MAE drops to 17 days. This illustrates that additional information can be very valuable. Note that the average total flow time is about 107 days. Moreover, as shown in Figure 16 there is a huge variation in flow times. Hence, it is quite remarkable that we can predict the remaining time until completion so accurately.

8 Conclusion

In this paper, we presented a new method for predicting the "future of a running instance". Given a running case, our prediction approach allows answering questions like "When will this case be finished?", "How long does it take before activity A is completed?", "How likely is it that activity B will be performed in the next two days?", etc. In this paper, we mainly focused on the remaining time until completion. However, our approach can easily be used for other types of predictions. The basic idea of the approach is to build an annotated transition system. This transition system is learned from past executions using an appropriate abstraction mechanism.

The approach is fully implemented in ProM. Using the FSM Miner a transition system is learned. The FSM Analyzer extends this transition system with predictive information. This information can be used to make predictions at run-time. Currently, we only support a link between the workflow management system Declare and our prediction tools in ProM. However, it is easy to extend the same toolset to other workflow management systems. To support experiments and to cross validate results, our FSM evaluator can be used to measure standard quality metrics such as MSE, RMSE, MAE, and MAPE.

In this paper, we evaluated our approach using a synthetic event log and two real-life event logs. Both experiments show that our approach outperforms simple heuristics. This is quite remarkable as we do not use a-priori knowledge. Note that we learn the model from past executions. The model can be used for predictions, but also has a value by itself as it shows where in the process the bottlenecks are.

As future work we plan to work more on the automation of abstractions, i.e., now the user still needs to select an abstraction based on the characteristics of the process and log. For example, when there are fewer process instances, then a more abstract transition system is desirable. It seems possible to automatically refine or aggregate states based on the quality metrics already gathered. Such approaches are not only interesting for predictions but also for process discovery, i.e., the parts of the process where there is more certainty are depicted in more detail and the more "fuzzy" parts of the process are simplified. For predictions all measures are considered to be of equal importance, while there could be correlations between activities. For example, after the occurrence of some activity it is very unlikely that some other activity will occur. In this case it does not make sense to (fully) use measures from traces that contain that latter activity for predicting an instance where the former activity has already been exectued. Correlations can be mined from the log [27] and be used to select or to weigh the state prediction measures. An assumption in our approach is the independency of the samples, i.e., the bag of measurements. However, the interaction between cases and the availability of resources are important factors when predicting the remaining time until completion [25]. These interactions should be incorporated into the predictions.

9 Acknowledgements

This research is supported by EIT, NWO-EW, SUPER, and the Technology Foundation STW. We would like to thank the many people involved in the development of ProM. In particular we would like to thank Eric Verbeek for implementing the FSM Miner, and Boudewijn van Dongen en Ronald Crooy for their work on the regression-based prediction service.

References

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. R. Crooy. Predictions in Information Systems: A process mining perspective. Master's thesis, Eindhoven University of Technology, Eindhoven, 2008.
9. A. Datta. Automating the Discovery of As-Is Business Process Models: Probabilistic and Algorithmic Approaches. *Information Systems Research*, 9(3):275–301, 1998.
10. S.N. den Hertog. Case prediction in BPM systems: Research to the predictability of the remaining time of individual cases. Master's thesis, Eindhoven University of Technology, Eindhoven, 2008.
11. B.F. van Dongen, R.A. Crooy, and W.M.P. van der Aalst. Cycle Time Prediction: When Will This Case Finally Be Finished? In R. Meersman and Z. Tari, editors, *Proceedings of the 16th International Conference on Cooperative Information Systems, CoopIS 2008, OTM 2008, Part I*, volume 5331 of *Lecture Notes in Computer Science*, pages 319–336. Springer-Verlag, Berlin, 2008.
12. M. Dumas, W.M.P. van der Aalst, and A.H.M. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software through Process Technology*. Wiley & Sons, 2005.
13. J. Eder, E. Panagos, and M. Rabinovich. Time Constraints in Workflow Systems. In M. Jarke and A. Oberweis, editors, *Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE '99)*, volume 1626 of *Lecture Notes in Computer Science*, pages 286–300. Springer-Verlag, Berlin, 1999.

14. J. Eder and H. Pichler. Probabilistic Calculation of Execution Intervals for Workflows. In *Proceedings of the 12th International Symposium on Temporal Representation and Reasoning*, pages 183–185, Washington, DC, USA, 2005. IEEE Computer Society.
15. 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.
16. 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.
17. K. Jensen, L.M. Kristensen, and L. Wells. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *International Journal on Software Tools for Technology Transfer*, 9(3-4):213–254, 2007.
18. A.K. Alves de Medeiros and C.W. Günther. Process Mining: Using CPN Tools to Create Test Logs for Mining Algorithms. In K. Jensen, editor, *Proceedings of the Sixth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2005)*, volume 576 of *DAIMI*, pages 177–190, Aarhus, Denmark, October 2005. University of Aarhus.
19. M. Pesic, H. Schonenberg, and W.M.P. van der Aalst. DECLARE: Full Support for Loosely-Structured Processes. In M. Spies and M.B. Blake, editors, *Proceedings of the Eleventh IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, pages 287–298. IEEE Computer Society, 2007.
20. H. A. Reijers. Case Prediction in BPM Systems: A Research Challenge. *Journal of the Korean Institute of Industrial Engineers*, 33:1–10, 2006.
21. 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.
22. A. Rozinat, R.S. Mans, M. Song, and W.M.P. van der Aalst. Discovering Colored Petri Nets From Event Logs. *International Journal on Software Tools for Technology Transfer*, 10(1):57–74, 2008.
23. A. Rozinat, R.S. Mans, M. Song, and W.M.P. van der Aalst. Discovering Simulation Models. *Information Systems*, 34(3):305–327, 2009.
24. A. Rozinat, M.T. Wynn, W.M.P. van der Aalst, A.H.M. ter Hofstede, and C. Fidge. Workflow Simulation for Operational Decision Support Using Design, Historic and State Information. In M. Dumas, M. Reichert, and M.C. Shan, editors, *International Conference on Business Process Management (BPM 2008)*, volume 5240 of *Lecture Notes in Computer Science*, pages 196–211. Springer-Verlag, Berlin, 2008.
25. B. Schellekens. Cycle Time Prediction in Staffware. Master's thesis, Eindhoven University of Technology, Eindhoven, 2009.
26. H. Schonenberg, B. Weber, B.F. van Dongen, and W.M.P. van der Aalst. Supporting Flexible Processes Through Recommendations Based on History. In M. Dumas, M. Reichert, and M.C. Shan, editors, *International Conference on Business Process Management (BPM 2008)*, volume 5240 of *Lecture Notes in Computer Science*, pages 51–66. Springer-Verlag, Berlin, 2008.
27. M.H. Schonenberg, N. Sidorova, W.M.P. van der Aalst, and K. vans Hee. History-dependent stochastic petri nets. In *Perspectives of Systems Informatics, 7th International Andrei Ershov Memorial Conference, PSI 2009, Novosibirsk, Russia, June 15-19, 2009.*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, June 2009.

28. Staffware. *Staffware Process Suite Version 2 – White Paper.* Staffware PLC, Maidenhead, UK, 2003.
29. S.E. Verwer, M.M. de Weerdt, and C. Witteveen. Efficiently learning timed models from observations. In L. Wehenkel, P. Geurts, and R. Maree, editors, *Benelearn*, pages 75–76. University of Liege, 2008.
30. B. Weber, W. Wild, and R. Breu. CBRFlow: Enabling Adaptive Workflow Management Through Conversational Case-Based Reasoning. In *Advances in Case-Based Reasoning*, volume 3155 of *Lecture Notes in Computer Science*, pages 434–448. Springer-Verlag, Berlin, 2004.
31. 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.
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.