Using Hidden Markov Models to Evaluate the Quality of Discovered Process Models

A. Rozinat[1,2], M. Veloso[2], and W.M.P. van der Aalst[1]

[1] Information Systems Group, Eindhoven University of Technology, NL-5600 MB, Eindhoven, The Netherlands. {a.rozinat,w.m.p.v.d.aalst}@tue.nl
[2] Computer Science Department, Carnegie Mellon University, Pittsburgh PA 15213-3890, USA. veloso@cmu.edu

Abstract. Hidden Markov Models (HMMs) are a stochastic signal modeling formalism that is actively used in the machine learning community for a wide range of applications such as speech and activity recognition. Efficient techniques exist to learn HMM models from a given data set, and to estimate the *data likelihood* with respect to a given HMM (i.e., "How probable is it that these data were produced by this HMM?"). The latter enables the evaluation and selection of suitable models. In the domain of *process mining* the evaluation of models (i.e., "How can we measure the quality of a mined process model?") is still subject to ongoing research. Because the types of models used in process mining are typically on a higher level of abstraction (i.e., they are more expressive as they, for example, allow to capture concurrency), the problem of model evaluation is challenging. In this paper, we investigate the possibilities and limitations of using HMMs for evaluating process models and compare the resulting quality measurements to the metrics typically used in the process mining domain.

1 Introduction

Process mining deals with the discovery of *process models* (i.e., structures that model behavior) from event-based data. The goal is to construct a process model which reflects the behavior that has been observed in some kind of *event log*. An event log is a set of finite event sequences, whereas each event sequence corresponds to one particular materialization of the process. Process modeling languages, such as Petri nets [13], can then be used to capture the causal relationships of the steps, or activities, in the process. While many different process mining approaches have been proposed over the last decade, no standard measure is available to evaluate the quality of such a learned model [29]. Quality measures are needed because a learned model cannot always explain all the data, and there are multiple models for the same data ("Which one is the best?"). These problems are due to the fact that a log typically does not contain negative examples and that there may be syntactically different models having the same (or very similar) behavior. This paper deals with the topic of evaluating process models and we use Petri nets as a typical representation of the class of graph-based process modeling languages (EPCs, BPMN, UML Activity Diagrams etc.).

Hidden Markov Models (HMMs) [26] are a stochastic signal modeling formalism. HMMs are actively used in the machine learning community for a wide range of applications such as speech and activity recognition [26, 19]. Efficient techniques exist to learn HMM models from a given data set, and to estimate the *data likelihood* with respect to a given HMM (i.e., "How probable is it that these data were produced by this HMM?"). The latter enables to evaluate, and choose between, alternative models.

In this paper, we investigate the applicability of HMMs for the purpose of evaluating process models. Because the types of models used in process mining are typically on a higher level of abstraction (i.e., they are more expressive as they, for example, allow to capture concurrency), the problem of model evaluation is challenging and HMMs can only be used by posing certain restrictions. Within these limits, in this paper we will apply HMM-based quality measurements and compare them to the metrics typically used in the process mining domain. We will see that typical process mining evaluation techniques—precisely because they need to deal with concurrency—have a bias when applied to simpler, sequential models. Furthermore, while focusing on evaluation, we also show the differences in representational power by comparing HMMs and Petri nets as a modeling technique, and we validate our analysis approach based on data of significant complexity. In summary, the contributions of this paper are as follows:

- We define an efficient mapping from Simple Petri nets (Labeled Petri nets without parallelism) to HMMs, based on which we provide metrics for the quality dimensions *fitness* and *precision*.
- We demonstrate that if this mapping is applied to Petri nets with concurrency, the metrics—due to the simplification—yield optimistic fitness and pessimistic precision results.
- We leverage the probabilistic nature of HMMs and present a framework to generate logs for a given model with varying, yet determined levels of *noise*. We then use this framework to evaluate a number of fitness metrics.

The remainder of the paper is organized as follows. First, we explain the problem domain of process model evaluation in more detail (Section 2). Then, we introduce the HMM and Petri net formalisms and scope the setting of our work (Section 3). Afterwards, we explain our mapping from Sequential Petri nets to HMMs based on a simple example (Section 4). Then, we define a number of HMM-based quality measurements (Section 5) and illustrate the differences in representational power of HMMs and Petri nets based on the presented mapping (Section 6). Next, we describe the experimental setup (Section 7) and present our results (Section 8). Finally, we conclude the paper (Section 9).

2 Process Model Evaluation

Process mining techniques focus on discovering behavioral aspects from log data. Since the mid-nineties several groups have been concentrating on the discovery of *process models* from event-based data. Process models are structures—usually

directed graphs—that model behavior. The idea of applying process mining in the context of workflow management was first introduced by Agrawal et al. in [5]. Over the last decade many process mining approaches have been proposed [10, 20, 4, 33, 34, 14, 11, 18, 2, 31]. In [3] van der Aalst et al. provide an overview about the early work in this domain. While all these approaches aim at the discovery of a "good" process model, often targeting particular challenges (e.g., the mining of loops, duplicate tasks, or in the presence of noise), they have their limitations and many different event logs and quality measurements are used. Hence, no commonly agreed upon measure is available.

In [28, 29] we motivate the need for a concrete framework that evaluates the quality of a process model, and thus enables (a) process mining researchers to compare the performance of their algorithms, and (b) end users to estimate the validity of their process mining results ("How much does this model reflect reality?"), and to choose between alternative models ("Which model is the best"?). It is important to understand that there is never only one single model for a given event log, but multiple models are possible due to two main reasons.

1. *There are syntactically different models having the same (or very similar) behavior.* Furthermore, there are numerous different modeling formalisms that can be employed. These models could be evaluated with respect to their complexity, size, understandability etc. We do not focus on this aspect in this paper.
2. *For a given input, an infinite number of models can be constructed.* Resulting models might not always accommodate all the traces in the event log, and they might allow for behavior not represented by any trace in the log. Both scenarios could be desirable, and different criteria can be used to construct and evaluate these models.

To illustrate the second aspect, consider Figure 1 which depicts four different process models that could be constructed based on the event log in the center. The event log contains five different traces with frequencies ranging from 1207 to 23 instances per trace. For example, the sequence $ABDEI$ (i.e., A followed by B, etc.) occurred 1207 times. While the model in Figure 1(c) does not accommodate all these five traces but only the first one (lack of *fitness*), the model in Figure 1(d) allows for much more behavior than just the traces in the event log (lack of *precision*).

Cook and Wolf [10] approach the discovery of Finite State Machine (FSM) models for software processes as a grammar inference problem, and, reflecting on the "goodness" of a model, they cite Gold [16] who showed that both *positive and negative* samples are required to construct 'accurate' (the FSM accepting all legal sentences and rejecting all illegal sentences of the language) and 'minimal' (the FSM containing the minimum number of states necessary) models. Furthermore, the samples must be *complete* (i.e., cover all possible inputs). However, the event logs used for process discovery cannot be assumed to be complete, and they normally do not contain negative examples. Note that the five traces in the event log in Figure 1(a) are positive examples, but no negative, or forbidden, traces are given. Furthermore, in some situations it can be the case that the

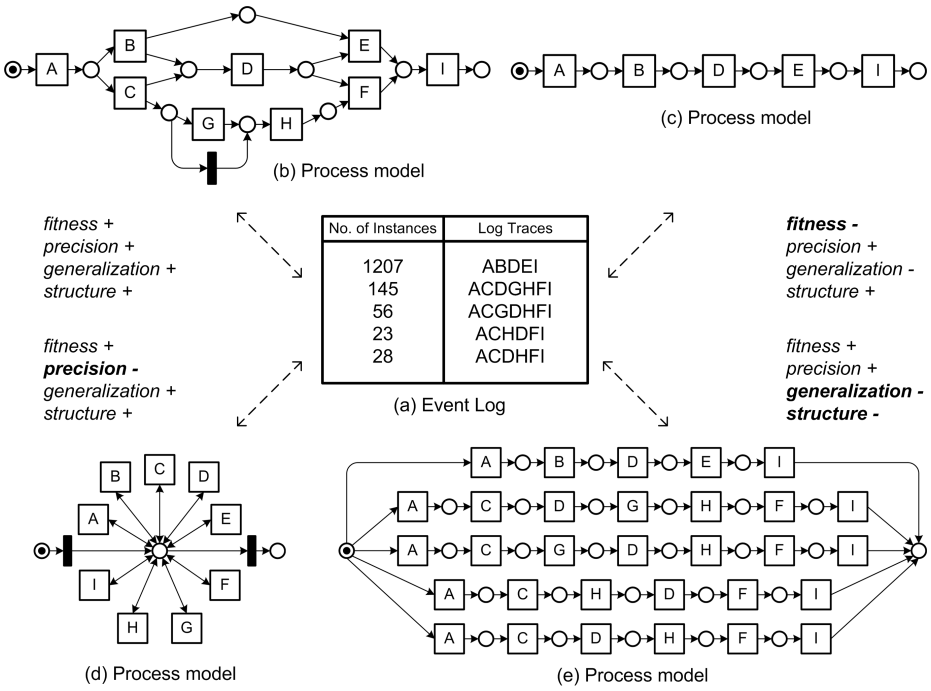


Fig. 1. Process model evaluation can place in different dimensions [28].

positive examples are in fact distorted (noise), or contain exceptions that should not be included in the model. Therefore, process discovery algorithms have to face the following problems.

Dealing with Incompleteness If the log would be complete, it would be easy to assume that every sequence not present in the log is a negative example, and thus should not be possible according to the discovered model. Unfortunately, total completeness is an unrealistic assumption as the number of possible interleavings of concurrent activities increases exponentially[3]. Thus, *generalization* beyond the observed sequences to accommodate concurrent or combined behavior is often desirable.

Further Abstraction Besides generalizing to deal with incompleteness, further abstraction may be necessary to obtain meaningful process models. For example, in the presence of overly complex processes it often does not make sense to show a very detailed ("spaghetti-like") model. Furthermore, one might want to deal with noise, or show the main flow of a process and thus ignore possible exceptions. In theses case, abstraction can lead to models with a decreased *precision* and a decreased *fitness*.

[3] Already 5 concurrent activities can generate 5! = 120 possible traces, and 10 concurrent activities can result in 10! = 3628800 differently ordered sequences.

So, we can see that—while on the first glance it seems logical to aim at models with perfect fitness and precision—this is not always desirable. Instead, algorithms strive to find "the right degree of abstraction", depending on the assumed circumstances and the purpose of the discovered model. As a consequence, process model evaluation needs to take these goals into account, and may have an unwanted bias if applied in the wrong context. For example, the model depicted in Figure 1(c) might be considered a good abstraction of the 80% most frequent behavior, but would be a relatively poor model if the goal is to obtain a complete picture of the overall process.

Although it is vital to develop practical methods that are able to take the desired abstractions made by process discovery algorithms into account when evaluating the resulting models, in reality there are often also simpler processes (or parts of processes) that only exhibit sequential routing, alternative behavior, and loops (but no parallelism). Examples of such processes can be found in administrative procedures employed in municipalities, insurance companies etc.

In the remainder of this paper we want to focus on such simple processes with the assumption that the models should be as accurate, i.e., *fitting* and *precise*, as possible for a given event log. We will define new evaluation metrics and compare them to existing evaluation methods used in the process mining field. It is important to note that here we do not seek for the all-encompassing standard measure (for this, posing restrictions on concurrency would not be a good idea), but rather aim at providing efficient base line metrics against which other evaluation metrics can be compared to see how they perform in these simple situations. Ultimately, the goal should be to give better support for what is the "right" quality metric for the situation at hand.

3 Preliminaries

We want to evaluate a given process model in terms of a Petri net with respect to a given event log. To do this, we will map the Petri net onto a Hidden Markov Model. However, in this section we first introduce the notion of an event log and the two modeling formalisms in more detail.

3.1 Event logs

Control flow discovery algorithms typically ignore time stamps and additional data that are commonly present in real-life logs and focus on the actual steps that take place. A process instance can then be seen as a sequence of such steps and an event log can be simplified to a set of different log traces and their frequencies. Figure 1(a) depicts such a simplified event log. We define an event log as follows.

Definition 1 (Event Log) *An event log is a tuple* (L, E, m)*, where:*

- L *is a finite set of* labels *that can be observed,*

– $E \subseteq L^*$ *is a finite set of* event sequences *(L^* is the set of all sequences of arbitrary length over alphabet L), and*
– $m : E \to \mathbb{N}_{\geq 1}$ *is a* frequency function *for event sequences.*

For simplicity, we write event sequences by juxtaposing the function values (e.g., *ABDEI* in Figure 1(a)). Furthermore, note that—rather than a set of event sequences—we consider a bag (or multiset), which can contain duplicate members. The multiplicity of the bag members is captured by the frequency function (e.g., $m(\sigma) = 1207$ for sequence $\sigma = ABDEI$).

3.2 HMMs

A Hidden Markov Model (HMM) [26] is an extension of a discrete markov process. A discrete, first order, Markov chain consists of a set of states and a set of state transition probabilities, whereas the probabilistic description is truncated to just the current and the predecessor state. This means, associated with each state we have a set of probabilities describing the likelihood of changing to a particular other state (possibly going back to the same state) that are independent of any previous states, and thus sum up to 1.

The difference between *Hidden* Markov Models and plain Markov Models is that the actual *states* of the process cannot be observed (they are *hidden*). Instead, there is a set of observable elements, whereas one *observation* can be produced by more than one state. Therefore, seeing a particular observation element in isolation is not enough to suggest the current state of the process. We define the Hidden Markov Model as follows.

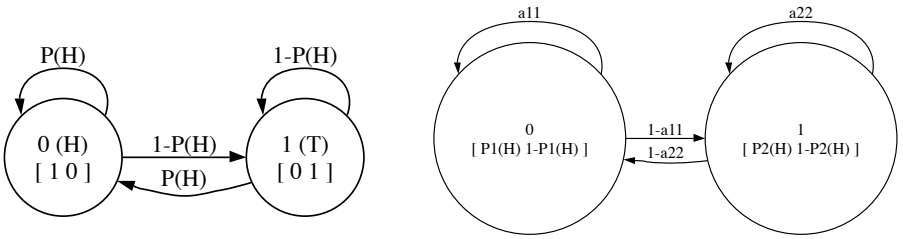
Definition 2 (Hidden Markov Model) *A Hidden Markov Model is a tuple (N, L, A, B, π), where:*

– N *is a finite set of* states*,*
– L *is a finite set of* observations*,*
– $A : (N \times N) \to [0, 1]$ *is a* state transition matrix*, such that:* $\forall_{s_1 \in N} \sum_{s_2 \in N} A(s_1, s_2) = 1$*,*
– $B : (N \times L) \to [0, 1]$ *are the* observation probabilities*, such that:* $\forall_{s \in N} \sum_{o \in L} B(s, o) = 1$*, and*
– $\pi : N \to [0, 1]$ *is the* initial state distribution*, such that:* $\sum_{s \in N} \pi(s) = 1$

So, in each of the states all of the observation elements could be produced, and the likelihood of observation o to be produced in a particular state s is captured by the observation probability $B(s, o)$. Note that we want to use HMMs to represent processes that have been observed by recording an event log, and therefore link the observations that can be produced by the HMM to the observable log elements by using the same identifier L.

Consider the following coin-tossing example taken from [26]. A certain sequence of coin tossing outcomes has been observed. That is, the observation

elements are heads (H) and tails (T). In each state, we can determine the probabilities for how likely each of the observation elements is to be produced. The HMM depicted in Figure 3.2(a) models one biased coin by producing only heads in state *0* and only tails in state *1*. It is, therefore, a degenerate HMM, which corresponds to an ordinary Markov chain, where the states are observable. Consider now Figure 3.2(b) which depicts an HMM that can produce both heads and tails in each state, with different probabilities. This representation is more powerful as it can model the situation of two biased coins that are tossed in an arbitrary order, which is useful as we do not know whether the observed sequence was produced by a single or multiple coins (only the outcome is visible—the underlying source is *hidden*). Similarly, we could construct an HMM with three states, which would be capable of modeling three biased coins, etc., and we could also represent the HMM from Figure 3.2(a) by one state only. Obviously, the more states an HMM has, the more representational power it embeds.



(a) HMM modeling a single biased coin. The state sequence for the given observation sequence would be *00110100110...*

(b) HMM modeling the merged outcome of two biased coins. A possible state sequence for the given observation sequence could be *10011101101...*

Fig. 2. Two possible HMMs that could be used to model the observation sequence of heads (H) and tails (T): *HHTTHTHHTTH...* Within the square brackets the probabilities for the observations [H T] are given for each state, respectively.

There are three fundamental problems for HMMs, for which formalized solutions are provided in [26]. These problems are:

1. "Given observation sequence $\sigma = o_1, o_2, ..., o_n$, how to compute the probability of σ, given the model λ, i.e., $Pr(\sigma|\lambda)$?"
2. "Given σ, how do we choose a corresponding state sequence $s_1, s_2, ..., s_n$ which best explains the observations?"
3. "How do we adjust the model parameters to maximize the fit with the observation sequences?"

The first problem can be solved efficiently using the *Forward-Backward Procedure* [6, 8]. The solution for the second problem depends on the optimality

criterion (i.e., what exactly means "best"?), and the most commonly used approach is to determine the single state sequence that is most likely to produce the provided observation sequence using the *Viterbi Algorithm* [32, 15]. The third is the far most difficult problem and can be solved iteratively, such as by *Baum-Welch* [7]. Furthermore, it is possible to calculate the distance between two HMMs, for example using a *Kullback-Leibler* distance as in [22].

In the remainder of this paper, we make use of existing solutions to the first two problems (without going into further detail about these solutions). That is, we will compute the probability of a given observation sequence with respect to a given HMM, and we will need to determine the most likely state sequence for a given observation sequence with respect to a given HMM.

3.3 Petri nets

A Petri net [13] is a dynamic structure that consists of a set of *transitions*, which are indicated by boxes and relate to some task, or action that can be executed, a set of *places*, which are indicated by circles and may hold one or more *tokens* (indicated as black dots), and a set of *directed arcs* that connect these transitions and places with each other in a bipartite manner. Transitions are *enabled* as soon as all of their input places (places connected to this transition via an incoming arc) contain a token. If a transition is enabled, it may *fire* whereas it consumes a token from each of its input places and produces a token for each of its output places (places connected to this transition via an outgoing arc). This way, the firing of a transition may change the *marking* of a net, and therefore the state of the process, which is defined by the distribution of tokens over the places. Examples of Petri nets can be seen in Figure 1(b)–(e).

We use Labeled Place/Transition nets, which is a variant of the classic Petri-net model, and which from now on we refer to as Labeled Petri nets for simplicity. They are defined as follows.

Definition 3 (Labeled Petri net) *A labeled Petri net is a tuple* (P, T, F, L, l)*, where:*

- P *is a finite set of* places*,*
- T *is a finite set of* transitions *such that* $P \cap T = \emptyset$*,*
- $F \subseteq (P \times T) \cup (T \times P)$ *is a set of directed arcs, called the* flow relation*,*
- L *is a finite set of* labels*, and*
- $l : T \not\rightarrow L$ *is a partial* labeling function*.*

Note that observable log elements are linked to the transitions in the Petri net model by the labeling function l. Because l is a partial function, there may be transitions in the model that are *unlabeled*, and therefore cannot be observed in the event log. They are also called 'invisible tasks' and used for routing purposes, such as to "skip" a particular part of the process. These unlabeled tasks are denoted as small transitions filled with black color. For example, in the model depicted in Figure 1(b) the transition G can be skipped by such an invisible

task. Furthermore, there can be multiple transitions in the Petri net that have the *same label*. They are also called 'duplicate tasks' and their occurrence cannot be distinguished in the event log. Duplicate tasks add expressiveness to the formalism as they allow to model behavior in different contexts (e.g., at the beginning and at the end of the process). For example, the model depicted in Figure 1(e) contains many transitions with the same label.

We assume that each process model belongs to a well-investigated subclass of Petri nets that is typically used to model business processes, which is the class of *sound WF-nets* [1]. A WF-net requires the Petri net to have (i) a single *Start* place, (ii) a single *End* place, and (iii) every node must be on some path from *Start* to *End*, i.e., the process is expected to define a dedicated begin and end point and there should be no "dangling" tasks in between. The soundness property further requires that (iv) each task can be potentially executed (i.e., there are no dead tasks), and (v) that the process—with only a single token in the *Start* place—can always terminate properly (i.e., finish with only a single token in the *End* place). Note that the soundness property guarantees the absence of deadlocks and live-locks. This way, we abstract from *correctness* problems and solely focus on the issue of *model quality*.

When evaluating process models based on HMMs, we restrict ourselves to simple Petri nets as defined in the following.

Definition 4 (Simple Petri net) *Let* $\bullet x = \{y \mid (x,y) \in F\}$ *denote the input nodes and* $x\bullet = \{y \mid (y,x) \in F\}$ *denote the output nodes of* $x \in P \cup T$*. A simple Petri net is then a labeled Petri net* (P, T, F, L, l) *where the following two additional constraints hold:*

- $\forall_{t \in T} : |t\bullet| \leq 1$
- $\forall_{t \in T} : |\bullet t| \leq 1$

The two constraints ensure that no transition has more than one outgoing and one incoming arc and, therefore, no parallelism is present in the the model. This sub class of Petri nets is called state machines and also excludes so-called long-distance dependencies. These restrictions are necessary due to differences in the representational power of labeled Petri nets and HMMs as we will show in Section 6. In the next section, we define a mapping of these simple Petri nets onto HMMs.

4 Constructing HMMs for Process Model and Event Log

To evaluate the quality of a simple Petri net model (with respect to a given event log) by HMM-based techniques we need to define a mapping from Petri nets to HMMs: First, we create an HMM based on the given Petri net (Section 4.1). Second, we relate sequences from the event log to this HMM to be able to evaluate their "match" (Section 4.2).

4.1 Mapping a Simple Petri Net onto an HMM

Figure 3 depicts a simple Petri net. In this Petri net, after placing a token in the left-most place, transition A is enabled and can be fired. Afterwards, either transition B or C (but not both) can be fired. Finally, A is fired and the process ends. Thus, there are precisely two valid firing sequence for this process, namely ABA and ACA.

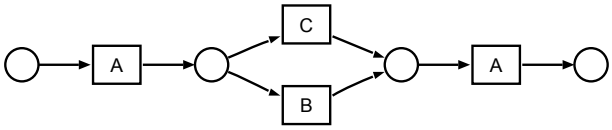


Fig. 3. A simple Petri net modeling the choice between C and B. Possible firing sequences are thus ABA and ACA.

Now we want to map this Petri net onto an HMM. For this, we basically represent each labeled transition by a state in the HMM, whereas we link the corresponding observation element with 100% probability to that state (but multiple states may be linked to the same observation). Unlabeled tasks are not represented by a separate state in the HMM as they are not observable.

As explained earlier, an HMM—in contrast to an ordinary Markov chain—does *not* assume that the states correspond to directly observable events. Instead, the observation is a probabilistic function of the state (for example, in a particular state two types of observable events might be equally likely to be produced). Thus, each state is, in addition to the parameters present in an observable Markovian model, associated with a vector of observation element probabilities (which again sum up to 1). Note that for our mapping we need this separation of states and observations although each state produces exactly one type of observation with 100% certainty (see Figure 4). However, because two states may produce the same observation element (for example, state *0* and state *3* both produce observation element A), from the mere observation we cannot conclude in which state we are. Thus, the states are not observable.

While a Markov chain is an inherently stochastic model, plain Petri nets are an analytical representation and do not directly support probabilistic descriptions[4]. We thus need to infer the probabilistic parameters of the Markovian model from the structure of the Petri net. This can be done as follows.

Step 1: Each labeled transition in the Petri net is represented by exactly one **state** in the Markov model (associated with a degenerate probabilistic observation function as described earlier). Transitions corresponding to invisible tasks are not represented by a state in the HMM.

[4] Note that there are more expressive representations, such as Colored Petri nets (CPNs) [21], which extend Petri nets—next to, e.g., hierarchy and time—also with probabilistic capabilities.

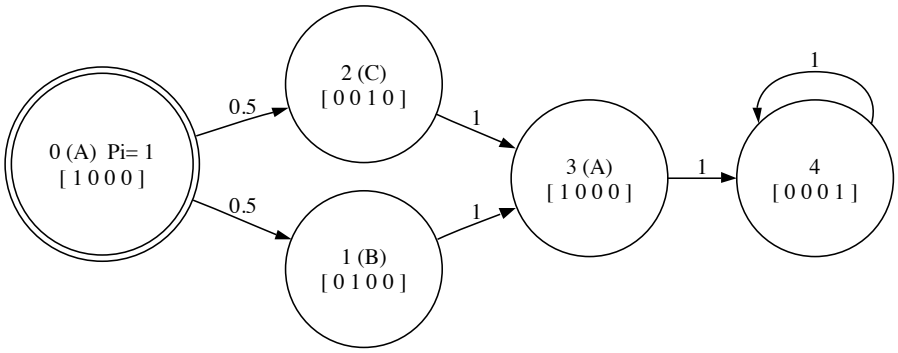


Fig. 4. The HMM constructed from the Petri net model in Figure 3. In our mapping, we create a direct link of the state and the observation element to be produced. For example, in state *0* we definitely produce the event A, in state *1* we produce event B etc. The observation event probability vector corresponds to [A B C e].

Step 2: All states that correspond to transitions connected to the *Start* place of the Petri net are assigned an equal probability of being the **initial state** of the HMM. In the Petri net of Figure 3 transition A is the only start task (i.e., all possible event sequences start with A). Therefore, state *0* has an initial probability $Pi = 1$ and all other states have an initial probability $Pi = 0$ (see Figure 4). Initial probabilities equal to 0 are not shown.

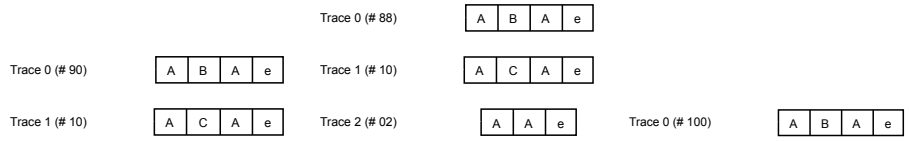
Step 3: To assign **transition probabilities** between states we analyze the structure of the Petri net and assign an equal transition probability from the state corresponding to the current transition to all states corresponding to possible successor transitions. For example, after executing A in the Petri net in Figure 3 either B or C are possible. Thus, in the HMM depicted in Figure 4 the transition probabilities from state *0* to state *1* and the transition probability from state *0* to state *2* are both 0.5. After firing C it is only possible to fire A. Therefore, the transition probability from state *1* to state *3* is 1 etc. Invisible tasks are traced, i.e., they are ignored and their successor transitions are determined instead (until only labeled successor transitions are left). Transition probabilities equal to 0 are not shown.

Step 4: To model the end of the process, a separate **final state** is introduced, which does not produce any observable events from the set L. Instead, it is associated to some dummy end element e. Once it is reached, it cannot be left anymore. In the HMM depicted in Figure 4 this is state *4*. Transition probabilities from states that correspond to possible final tasks in the Petri net (i.e., tasks that can happen at the very end of the process) are not only split among possible successor states but also the final state. In the Petri net shown in Figure 3 the process always ends after firing transition A. Thus, the transition probability from state *3* to state *4* is 1.

4.2 Relating Event Sequences to the HMM

So far, we have used the given Petri net to create an HMM, which is simply another representation of the same process. For example, according to the Petri net in Figure 3 it is not possible to fire the second transition A directly after firing the first A. Correspondingly, the transition probability from state *0* to state *3* in the HMM depicted in Figure 4 is 0.

However, recall that our goal is to evaluate a Petri net model with respect to a given event log. Therefore, consider Figure 5, which depicts three different example scenarios containing each 100 process instances. We want to evaluate the Petri net in Figure 3 with respect to each of these event logs. In the first scenario, the event sequences ABA and ACA have been observed, whereas ABA occurred 90 times and ACA occurred 10 times. In the second scenario, three different event sequences have been recorded. In addition to ABA (88 times) and ACA (10 times) also the sequence AA (2 times) was observed. Finally, in the third scenario only the sequence ABA was observed (100 times). The element e is not actually in the event log, but we artificially inserted it to mark the end of the sequence.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Fig. 5. Three different event logs are given in the scenarios 1–3.

If we now want to relate these observation sequences to the HMM constructed from the model to evaluate their "match", then we cannot do this directly although each state is linked with 100% certainty to a specific observation element. The reason is that there can be more than one state that are linked to the same observation element (for example, both state *0* and state *3* are linked to the observation A). Fortunately, we can make use of existing solutions to the common HMM problem of finding the best sequence of states for a given observation sequence, such as the Viterbi algorithm (cf. Section 3.2).

So, using such an existing mechanism to retrieve the best state sequence $s_1, s_2, ..., s_n, s_{final}$ for a given observation sequence $o_1, o_2, ..., o_n, e$, we could, for example, relate the first trace $ABAe$ in the log depicted in Figure 5(a) to the state sequence *0134* in the HMM depicted in Figure 4. Similarly, we obtain the state sequence *0234* for the second trace $ACAe$ in Figure 5(a).

However, if we try to do the same for the event log depicted in Figure 5(b), we will find out that the last trace AAe cannot be related to any state sequence of equal length as the observation sequence. The reason is that in the HMM

in Figure 4 there is no possible way to transition from the start state *0*, which does produce observation A, to a state also producing A (both the transition probability from state *0* to state *0* as well as the transition probability from state *0* to state *3* is 0). So, for this observation sequence the Viterbi algorithm cannot give us a result (there is no "most likely" sequence as there is no *possible* sequence at all).

Nevertheless, we want to be able to obtain a state sequence for *any* given observation sequence. For this, we have to adapt the HMM that was constructed from the process model in such a way that it *also allows for state transitions that are not possible for the input model*. Furthermore, the initial state probabilities must be adapted as an observation sequence can potentially start with any observation. But on the other hand we want that in the case of multiple possible state sequences those paths that are correct according to the input model are chosen over incorrect state sequence, if they are available. Therefore, *correct state transitions in the adapted HMM must be much more likely than incorrect state transitions*. We call this adapted HMM to be used to relate event sequences to the input model an ϵ-HMM.

Definition 5 (ϵ-HMM) *Let $\lambda_M = (N, L, A_M, B, \pi_M)$ be the HMM created from the simple Petri net model, and ϵ be a sufficiently small value. The state transition probabilities A_ϵ and the initial state probabilities π_ϵ of the ϵ-HMM $\lambda_\epsilon = (N, L, A_\epsilon, B, \pi_\epsilon)$ are defined as follows:*

$$A_\epsilon(s, s') = \begin{cases} 1 & \text{if } s' = s_{final} \\ \frac{1-\epsilon}{|\{(s,s'') \mid A(s,s'')>0\}|} & \text{if } A(s,s') > 0 \\ \frac{\epsilon}{|\{(s,s'') \mid A(s,s'')=0\}|} & \text{if } A(s,s') = 0 \end{cases}, \; s, s', s'' \in N$$

$$\pi_\epsilon(s) = \begin{cases} 0 & \text{if } s = s_{final} \\ \frac{1-\epsilon}{|\{s' \mid \pi(s')>0\}|} & \text{if } \pi(s) > 0 \\ \frac{\epsilon}{|\{s' \mid \pi(s')=0\}|} & \text{if } \pi(s) = 0 \end{cases}, \; s, s' \in N$$

Figure 6 shows the ϵ-HMM for the HMM depicted in Figure 4 with ϵ being 0.01. For example, while the transition probability from state *0* to state *1* and from state *0* to state *2* was 0.5 for the model HMM, it is now $\frac{1-0.01}{2} = 0.495$ for the ϵ-HMM. The transition probabilities from state *0* to the remaining states *0*, *3*, and *4* where previously 0, and are now $\frac{0.01}{3} \approx 0.003$. Similarly, the initial state probability for the previously only possible initial state, state *0*, is $\frac{1-0.01}{1} = 0.99$, while the remaining states, which were previously impossible initial states, have now an initial state probability of $\frac{1-0.01}{3} \approx 0.003$.

Now, using the ϵ-HMM, we are able to find a state sequence for the observation sequence AAe. The most likely state sequence is *034*. Note that this state sequence contains a transition from state *0* to state *3*, which was not possible according to the initial model HMM. These situations are interesting because according to the model AA should not be possible (B or C must happen in between), but it still occurred in the event log. So, the model fails to capture all the observed behavior. Coming from a model perspective, we call these situations

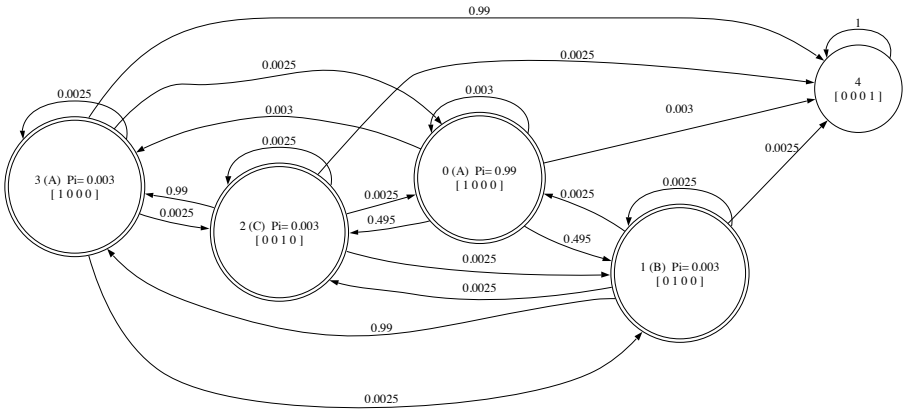


Fig. 6. The ϵ-HMM. To allow the detection of the most likely state sequence for each event sequence, impossible transitions and initial states must be made possible. Here, the ϵ-Hmm for the Petri net model in Figure 3 is depicted with $\epsilon = 0.01$

'false negatives', and we will use them later to measure the *fitness* of the process model.

Definition 6 (False Negatives) *Let $\lambda_M = (N, L, A_M, B, \pi_M)$ be the HMM created from the simple Petri net model and $\omega = (L, E, m)$ the event log. Furthermore, assume that given any observed event sequence $\sigma = o_1, o_2, ..., o_n$ of length $n \in N_{\geq 2}$ the most likely state sequence $\gamma = s_1, s_2, ..., s_n, s_{final}$ can be determined for $o_1, o_2, ..., o_n, e$ with respect to the ϵ-HMM. The false negatives FN over a set of event sequences E is defined as follows:*

$$FN = \bigcup_{\sigma \in E} \{(s_i, s_{i+1}) \mid (A_M(s_i, s_{i+1}) = 0 \ , \ 1 \leq i \leq n-1\}$$

Finally consider the log depicted in Figure 5(c), which only contains the event sequence *ABAe*. The most likely state sequence for this observation sequence with respect to the ϵ-HMM (and also with respect to the initial model HMM) is *0134*. Looking at the model HMM in Figure 4, we notice that not all initially possible state transitions have been observed (for example from state *0* to state *2* and from state *2* to state *3*). These situations are interesting because according to the model *AC* and *CA* should be possible, but they did not occur in the event log. So, the model actually captures more than the observed behavior. We call these situations 'false positives', and we will use them later to measure the *precision* of the process model.

Definition 7 (False Positives) *Let $\lambda_M = (N, L, A_M, B, \pi_M)$ be the HMM created from the simple Petri net model and $\omega = (L, E, m)$ the event log. Furthermore, assume that given any observed event sequence $\sigma = o_1, o_2, ..., o_n$ of length $n \in N_{\geq 2}$ the most likely state sequence $\gamma = s_1, s_2, ..., s_n, s_{final}$ can be*

determined for $o_1, o_2, ..., o_n, e$ with respect to the ϵ-HMM. The false postives FP over a set of event sequences E set is defined as follows:

$$FP = (N \times N) \setminus \bigcap_{\sigma \in E} \{(s_i, s_{i+1}) \ , \ 1 \leq i \leq n-1\}$$

5 Metrics and Evaluation of Data

Based on the mapping presented in the previous section, we now define several metrics to evaluate the *fitness* (Section 5.1) and *precision* (Section 5.2) of the process model with respect to the given event log. These metrics will then be used together with other metrics to evaluate a larger set of models in the experiments in Section 8.

5.1 Fitness

Fitness essentially measures the extent of observed behavior that is rejected by the model. Depending on the viewpoint, it either evaluates to which degree "legal sequences are not properly captured by the model" (log-based view) or to which degree "sequences are erroneous given the model" (model-based view). These viewpoints are equivalent and matter mostly when it comes to interpreting the results to take corrective actions.

A very simple notion of fitness is defined by the fraction of process instances in the event log that can be represented by the model without any error. This notion has been used before, for example by the 'Parsing Measure' defined in [33]. We state an error if there are two events in the trace directly following each other, but in the model-based HMM the transition probability between the corresponding states is 0. This can be directly measured by $Pr(\sigma|\lambda)$, the probability of the given observation sequence σ with respect to the HMM λ, which yields 0 if it contains a transition which is impossible. Fortunately, the probability of a given observation sequence with respect to a given HMM can be efficiently computed (see [26]).

Metric 1 (Trace Fitness) *Let $\lambda_M = (N, L, A_M, B, \pi_M)$ be the HMM created from the simple Petri net model and $\omega = (L, E, m)$ the event log. Furthermore, $E_0 = \{\sigma \mid Pr(\sigma = o_1, o_2, ..., o_n|\lambda_M) = 0, \ \sigma \in E\}$ is the set of non-fitting event sequences. The trace fitness f_{trace} is defined as follows:*

$$f_{trace} = 1 - \frac{\sum_{\sigma \in E_0} m(\sigma)}{\sum_{\sigma \in E} m(\sigma)}$$

Assuming that the event log is not empty, this metric yields a value between 0 (none of the observed sequences is possible according to the model) to 1 (all observed sequences are possible). For example, for the Scenario 2 depicted in Figure 5(b) the trace fitness yields $1 - \frac{2}{100} = 0.98$.

A second way to look at fitness is to evaluate how many "forbidden" transitions in the model have been "broken" by the log. So, the fitness is measured in direct relation to the causal relations of the model. Here, we can use the notion of 'false negatives' defined in Section 4.2. Note that we ignore the final state here as our goal is not to evaluate the "proper completion" of the traces. That is, sequences that do not lead to the final state but are otherwise possible with respect to the model are considered to be correct[5].

Metric 2 (Model Fitness) *Let $\lambda_M = (N, L, A_M, B, \pi_M)$ be the HMM created from the simple Petri net model and $\omega = (L, E, m)$ the event log. Furthermore, $AN = \{(s, s') \in N \times N \mid A_M(s, s') = 0 \ \wedge \ s' \neq s_{final}\}$ is the set of all negative transitions in the model-based HMM. The model fitness f_{model} is defined as follows:*

$$f_{model} = 1 - \frac{|FN|}{|AN|}$$

Assuming that at least one transition probability in the HMM is 0, the metric ranges from 0 (all "forbidden" transitions in the model have been "broken") to 1 (none of them has been "broken"). For example, in the Scenario 2 depicted in Figure 5(b) $(3, 0) \in FN$ and the model fitness yields $1 - \frac{1}{12} \approx 0.92$.

A third way to define fitness is to take the direct succession of events as a reference point and to punish the occurrence of subsequences that are "forbidden" by the model based on their frequency in relation to the whole log.

Metric 3 (Log Event Fitness) *Let $\lambda_M = (N, L, A_M, B, \pi_M)$ be the HMM created from the simple Petri net model and $\omega = (L, E, m)$ the event log. Furthermore, $m'_\sigma : N \times N \to \mathbb{N}$ is a frequency function for how often a pair of states $(s, s') \in N \times N$ has been listed as direct successors, i.e., $s = s_i \wedge s' = s_{i+1}$, in the most likely state sequence $\gamma = s_1, s_2, ..., s_n, s_{final}$ determined for $o_1, o_2, ..., o_n, e$ with respect to the ϵ-HMM, given an event sequence $\sigma = o_1, o_2, ..., o_n$ of length $n \in N_{\geq 2}$. The log event fitness f_{event} is defined as follows:*

$$f_{event} = 1 - \frac{\sum_{\sigma \in E} \sum_{(s,s') \in FN} \ (m(\sigma) \cdot m'_\sigma(s, s'))}{\sum_{\sigma \in E} \sum_{(s,s') \in (N \times (N \setminus s_{final}))} \ (m(\sigma) \cdot m'_\sigma(s, s'))}$$

The metric yields a value between 0 (none of the observed event pairs should be possible according to the model) to 1 (all the observed transitions were allowed according to the model). For example, for the Scenario 2 depicted in Figure 5(b) the event fitness yields $1 - \frac{2}{198} \approx 0.99$.

5.2 Precision

Precision essentially measures the extent to which non-observed behavior is accepted by the model. As discussed earlier, the notion of precision is highly linked to how we define the completeness of a log.

[5] This is motivated by our later simulation experiments where we generate traces that are not necessarily completed.

Similarly to the model fitness metric defined in Section 5.1 we can define a precision metric based on how many "allowed" transitions in the model have "not been used" by the log. So, the precision is measured in direct relation to the causal relations of the model. Here, we can use the notion of 'false positives' defined in Section 4.2.

Metric 4 (Model Precision) *Let $\lambda_M = (N, L, A_M, B, \pi_M)$ be the HMM created from the simple Petri net model and $\omega = (L, E, m)$ the event log. Furthermore, $AP = \{(s_i, s_j) \in N \times N \mid A_M(s_i, s_j) > 0 \wedge s_i \neq s_{final} \wedge s_j \neq s_{final}\}$ is the set of all positive transitions in the model-based HMM. The model precision p_{model} is defined as follows:*

$$p_{model} = 1 - \frac{|FP|}{|AP|}$$

Assuming that at least one transition probability in the HMM is not 0, which is always true for a non-empty model as there is at least one state leading to the s_{final}, the metric ranges from 0 (none of the "allowed" transitions in the model have been "used") to 1 (all of them have been "used"). For example, in the Scenario 3 depicted in Figure 5(c) $(3, 1), (1, 0) \in FP$ and the model precision yields $1 - \frac{2}{4} = 0.5$.

Furthermore, we can again use the fact that the probability of a given observation sequence with respect to the HMM can be efficiently computed, and add up the probabilities of all different traces in the log with respect to the model HMM to see how much of the model behavior is covered. Note that this essentially enables us to efficiently compute the portion of observed sequences in relation to the total number of possible sequences as, for example, defined by the 'completeness' precision metric defined in [17], despite the fact that the number of sequences allowed by the model may be infinite!

Metric 5 (Log Completeness) *Let $\lambda_M = (N, L, A_M, B, \pi_M)$ be the HMM created from the simple Petri net model and $\omega = (L, E, m)$ the event log. The total precision p_{total} is defined as follows:*

$$p_{total} = \sum_{\sigma \in E} Pr(\sigma = o_1, o_2, ..., o_n | \lambda_M)$$

For the Scenario 3 depicted in Figure 5(c) the only sequence in the log has a probability of 0.5, and thus the log completeness yields 0.5.

6 Representational Power of HMMs and Petri Nets

In our mapping we have associated each labeled task with only one state in the Markov process. To do this, we had to pose restrictions on the process models and limit them to Simple Petri nets The assumption of Markovian processes is to determine the next step only based on the current state (ignoring the

history), which is usually not appropriate for process models that may exhibit long-distance dependencies (e.g., some later choice depends on the outcome of an earlier choice in the process) and concurrency. However, being inclined to look more than one step forward then also renders the situation inherently more complicated.

For example, if one wants to map the situation more precisely then one could model the state space of the process model by representing each possible marking by a separate state. This then corresponds to a so-called reachability graph (or coverability graph if infinitely growing markings are to be caught in ω-states). But unfortunately the state space of a model grows exponentially fast ("state space explosion"), and already seemingly small models may not be computable anymore with contemporary personal hardware (see Appendix A for an example). Therefore, theoreticians strive for finding ways to evaluate interesting properties by structural analysis, i.e., investigating the *structure* of the Petri net rather than the *state space*.

Formal methods often employ abstraction techniques such as in model checking [9], or the reduction rules in Petri nets [25], to verify interesting properties on only a subset of the original model. In a similar way, we can apply our mapping of one state per labeled task to models beyond the limits of Simple Petri nets (which can be truthfully represented) and see it as an abstraction of the original model.

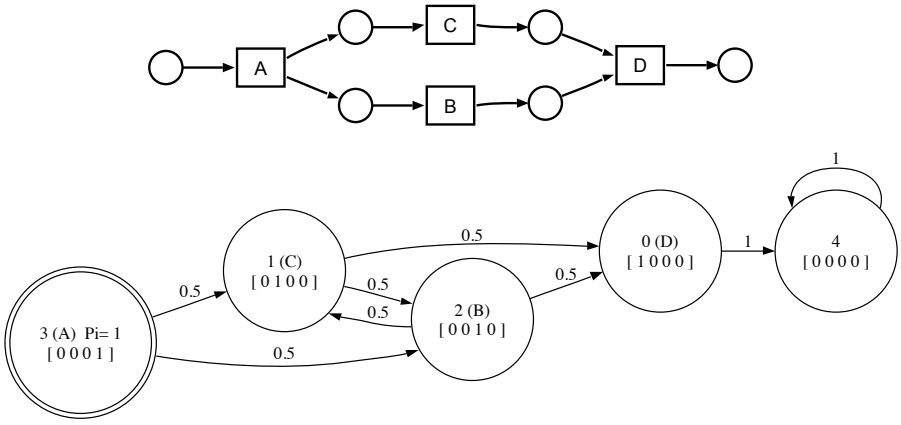


Fig. 7. A Petri net that includes concurrency (B and C can be executed in parallel) and its HMM mapping.

Consider, for example, the concurrent Petri net in Figure 7. The process model contains two parallel tasks B and C, which can be executed in any order after A but must be completed before starting D. If we map this Petri net onto our HMM representation with one state per labeled task we obtain the HMM

shown in Figure 7. Due to the simplification, this HMM allows for observation sequences that were not possible for the Petri net (e.g., $ABCBCBD$ would be a possible observation sequence with respect to the HMM but not a legal sequence for the Petri net).

Interestingly, we can still say *something* about the Petri net based on only looking at the HMM, namely that sequences that are impossible for the HMM must also be impossible for the Petri net (e.g., AD). Thus, if we find a sequence to be illegal with respect to the HMM, we can conclude that the sequence is also illegal for the Petri net, but this does not hold for legal sequences (possible sequences for the HMM do not need to be legal for the Petri net). Thus, we obtain *optimistic fitness* results based on our mapping. Conversely, we can look at the precision of the mapped model with respect to some log. While we cannot be sure that a sequence that would be possible according to the HMM and is not present in the log should be also possible with respect to the Petri net, we can say that those sequences that are represented by the Petri net are also represented by the HMM. Thus, we obtain *pessimistic precision* results based on our mapping.

Note that it is possible to "unfold" the concurrent Petri net from Figure 7 by listing each possible interleaving as an alternative path using duplicate tasks, which are allowed for our simple Petri nets, like depicted in Figure 8. However—due to the exponentially increasing number of possible interleavings—this leads exactly to the same problem as the state space explosion described before (the abstraction level of the model is simply lowered towards its state space), and, thus, is only of limited applicability.

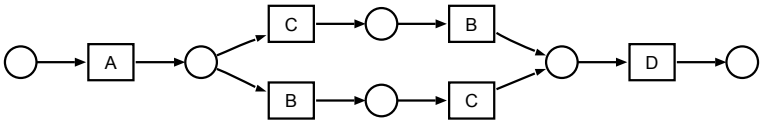


Fig. 8. The concurrent Petri net from Figure 7 as a simple Petri net, unfolded using duplicate tasks.

Finally, one could say that—due to the fact that we link each state to precisely one of the possible observation elements—the potential of HMMs has not been fully leveraged. However, if we aim at an exact representation, which enables us to calculate metrics like presented in Section 5, there is not much room to do so. While the HMM from Figure 4 can be further abstracted by collapsing two states without changing the behavior of the model as shown in Figure 9, already for the Petri net model depicted before in Figure 8 cannot be mapped anymore to such a more concise HMM representation without adding unwanted behavior.

Nevertheless, there may be other applications of HMMs in the context of process mining. For example, it would be interesting to see how one could leverage the stochastic nature of HMMs to capture very unstructured processes that typ-

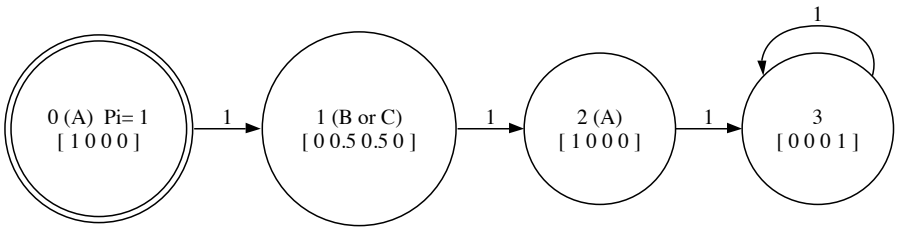


Fig. 9. The HMM from Figure 4 further abstracted by collapsing two states.

ically result in so-called "spaghetti-models" when traditional mining techniques are employed. As indicated in Section 3.2, several learning approaches are available to "fit" the probabilistic parameters of an HMM to a set of observation sequences (cf. third common problem for HMMs).

7 Genererating Noise

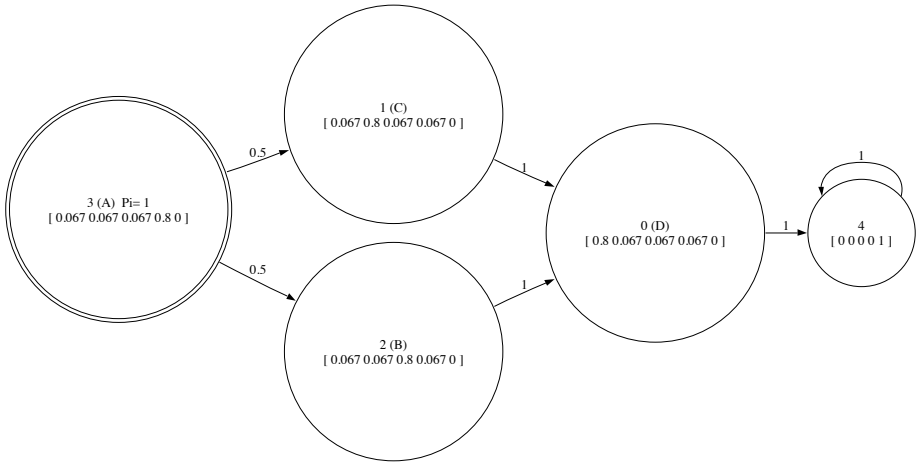
In our experiments we now concentrate on the fitness dimension and want to evaluate a model with respect to logs containing different degrees of noise. For this, we first define two different types of noise (Section 7.1 and Section 7.2). Then, we provide an overview about the setup and present the ProM plug-in realizing our experimental approach (Section 7.3).

7.1 Observation Noise

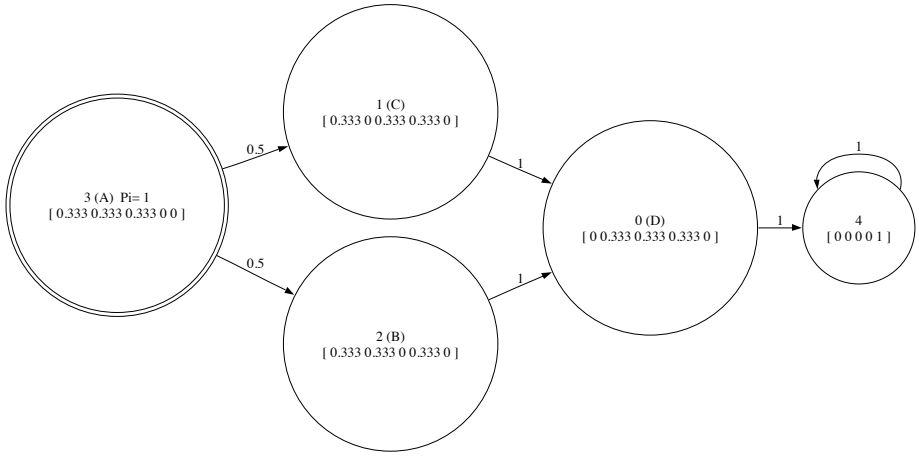
One possibility to generate distorted logs is to make use of the separation of states and observation elements in the HMM by adjusting the observation element probabilities for each state. Previously, each state could only produce the one observation element that was linked to this state. Now, with increasing degree of noise we can decrease the probability of the "right" observation element and introduce some probability to generate a "faulty" observation element for each state. This way, we can conveniently use the modified HMM to generate observation sequences by means of simulation.

Consider the example in Figure 10(a), where we adjusted the observation probabilities of the HMM depicted in Figure 4 to introduce 20% noise. In each state, the correct observation element is being generated with a probability of 0.8, and a wrong observation element is being generated with a probability of 0.2 (each of the possible wrong observations has a probability of 0.067). Now consider the example in Figure 10(b), which represents the same HMM with degree of 100% noise. That is, in each state the correct observation is impossible, i.e., an incorrect observation element is produced with 100% certainty.

More specifically, we can determine the observation element probabilities for the noisy HMM in each state as follows.



(a) 20% noise



(b) 100% noise

Fig. 10. Examples HMMs for different levels of observation noise.

Definition 8 (Observation Noise) *Let $\lambda = (N, L, A, B, \pi)$ be an initial HMM, $nl \in \mathbb{N}_{\geq 1}$ the number of noise levels to be generated, and $\Delta = \frac{1}{nl}$ the difference by which the correct observations are to be reduced on each noise level. The observation probabilities B_i of the HMM on noise level $i, 1 \leq i \leq nl$, i.e., $\lambda_i = (N, L, A, B_i, \pi)$, are defined as follows:*

$$B_i(s,o) = \begin{cases} 0 & \text{if } s = s_{final} \\ 1 - i \cdot \Delta & \text{if } B(s,o) = 1 \\ \frac{i \cdot \Delta}{|L|-1} & \text{if } B(s,o) = 0 \end{cases}, \; s \in N, o \in L$$

While the correct observation element probability ($B(s, o) = 1$ in the initial Hmm) decreases the incorrect observation element probabilities ($B(s, 0) = 0$ in the initial Hmm) increase with an increasing noise level. Note that the observation probabilities of the final state do not change, since the final state has the special role of representing the end of the process and does not actually produce an observation element from the set L.

Note that for this kind of noise we only distorted the observation element probabilities of the HMM, but the transition probabilities remain the same. This means that the generated traces for the running example will still always have exactly three events, since the structure of the HMM is properly followed. We can see this kind of noise as a faulty logging mechanism. That is, the right task is being executed, but the corresponding log element that is produced is "noisy".

7.2 State Transition Noise

While we only modified the observation element probabilities for the previous type of noise, we now change the behavior of the process by modifying the transition probabilities between the states. This way, the process may increasingly "jump" from one state to another although it would not be possible to the original process description. We can see this kind of noise as a behavioral noise.

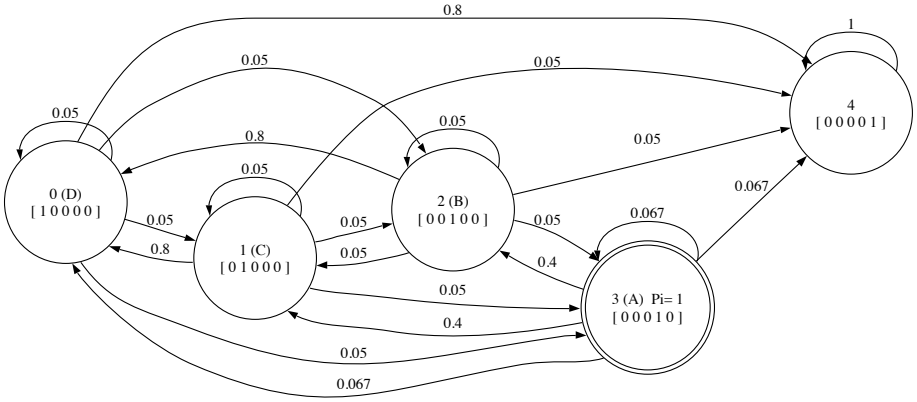
Consider the example in Figure 11(a), where we adjusted the state transition probabilities of the HMM depicted in Figure 4 to introduce 20% noise. In each state, a correct state transition is performed with a probability of 0.8 (for example, both transition probabilities from state *3* to state *1* and from state *3* to state *2* are 0.4), and a wrong state transition is performed with a probability of 0.2 (each of the otherwise wrong state transitions has a probability of 0.067). Now consider the example in Figure 11(b), which represents the same HMM with degree of 100% noise. That is, in each state the correct transitions are impossible, i.e., an incorrect state transition is performed with 100% certainty.

More specifically, we can determine the observation element probabilities for the noisy HMM in each state as follows.

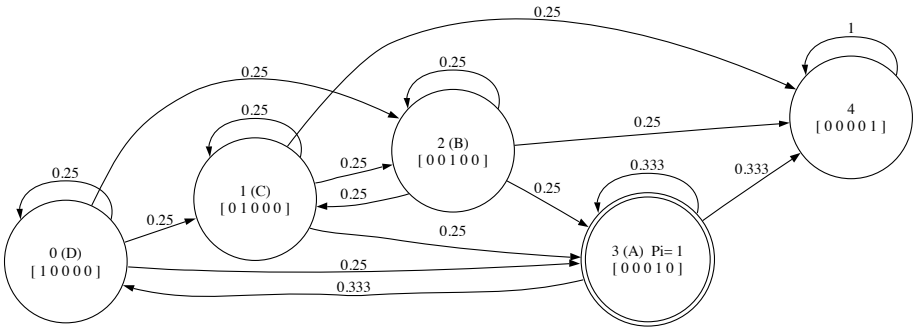
Definition 9 (Transition Noise) *Let $\lambda = (N, L, A, B, \pi)$ be an initial HMM, $nl \in \mathbb{N}_{\geq 1}$ the number of noise levels to be generated, and $\Delta = \frac{1}{nl}$ the difference by which the correct observations are to be reduced on each noise level. The state transition probabilities A_i of the HMM on noise level $i, 1 \leq i \leq nl$, i.e., $\lambda_i = (N, L, A_i, B, \pi)$, are defined as follows:*

$$A_i(s_j, s_k) = \begin{cases} 1 & \text{if } s_j = s_{final} \wedge s_k = s_{final} \\ 0 & \text{if } s_j = s_{final} \wedge s_k \neq s_{final} \\ \frac{1 - i \cdot \Delta}{|\{(s_j, s_l) | A(s_j, s_l) > 0\}|} & \text{if } A(s_j, s_k) > 0 \\ \frac{i \cdot \Delta}{|L| - |\{(s_j, s_l) | A(s_j, s_l) > 0\}| + 1} & \text{if } A(s_j, s_k) = 0 \end{cases}, \; s_j, s_k, s_l \in N$$

Note that transitions from the final state do not change as the final state still represents the end of the process and should not be left once it has been reached.



(a) 20% noise



(b) 100% noise

Fig. 11. Examples HMMs for different levels of transition noise.

7.3 Overview Experimental Setup

Figure 12 illustrates our experimental approach. The upper part shows the *HMM Analyzer* component, which—based on the Petri net—creates the initial model-based HMM and the ϵ-HMM. These two HMMs and the event log are then used to calculate the metrics presented in Section 5.

The *Noise Experimenter* component generates noisy logs based on the model-based HMM. Each of these noisy logs is then evaluated with respect to the input Petri net model using the metrics presented in Section 5 and potentially using other existing process mining metrics.

Furthermore, for each noise level—instead of only one log—we can also generate multiple logs based on the same HMM. These *replications* are independent of each other and could be used to derive statistical measures, such as the confidence interval of an average fitness value.

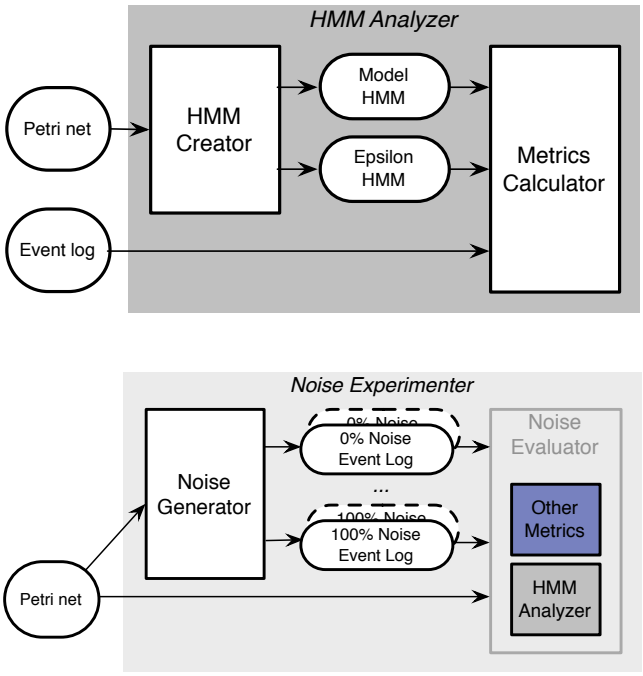


Fig. 12. The *HMM Analyzer* component calculates the presented metrics, which is then used by the *Noise Experimenter* to evaluate logs of different levels of noise.

The HMM Experimenter plug-in in the ProM framework[6] (see Figure 13) realizes the described approach, and offers a number of parameters to configure the experiment. More precisely, the following parameters are availble:

No. of Noise Levels The number of different noise levels to be generated. Being always distributed from 0% to 100%, the number of noise levels affects the granularity, or noise step size. For example, 10 noise levels result in logs generated with 10%, 20%, ..., 100% noise.

No. of Traces The number of traces to be generated for each log.

[6] Both software (including source code) and documentation are freely available from the websites *http://prom.sf.net* and *www.processming.org*.

Maximum No. of Events The maximum number of events to be generated for each trace (needed since some models may allow for infinitely long sequences).

No. of Replications If replications are desired, then this is the number of logs to be generated for each noise level.

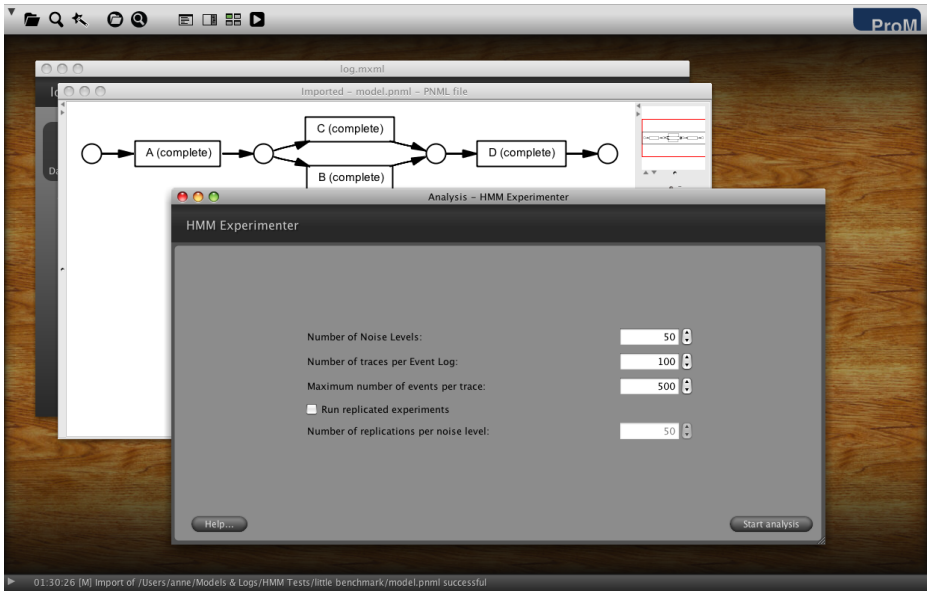


Fig. 13. Screenshot of the HMM Experimenter in ProM.

8 Experimental Results

We used the described experimental setup to perform experiments for varying degrees of noise based on process models from different domains. First, we used four process models that were mined based on log data collected by the CMDragons team during the international robot soccer competition 'RoboCup' 2007, and thus constitute models of the behavior in a multi-agent robotic system [30]. Second, we evaluated three different models of administrative processes within a municipality in the Netherlands. The same processes were used for mining experiments in [12]. Finally, we selected three suitable models from the SAP reference models, which is a publically available model that contains more than 600 enterprise models [24], and analyzed them with our approach.

The original models were given in terms of Heuristics nets and EPCs, and they were translated into Petri nets using conversion facilities in the ProM framework, respectively. We performed experiments on these models with varying parameters. The results of these experiments are very similar, and in the following we use a single, but representative, example to point out the main conclusions that we can draw from them. The detailed results are provided in Appendix B, C, and D.

As for the process model evaluation metrics, we used the HMM-based metrics defined in this paper as well as the following two other metrics from the process mining domain.

- The token Fitness [27] is based on replaying the log in a Petri net process model and relating the number of "missing" and "remaining" tokens during log replay to the total number of produced and consumed tokens (here referred to as *Token Based*). A similar metric is the Continuous Parsing Measure [33], which measures the number of missing and remaining activations while replaying a Heuristics net.
- The *Improved Continuous* semantics fitness [12] is used by the Genetic Miner to select the best process models in each generation of the genetic algorithm, and incorporates a fitness evaluation similar to the Continuous Parsing Measure, a precision evaluation, and gives some extra weight based on the number of traces that have problems.

Now consider Figure 14, which depicts the fitness values (y axis) for 50 different noise levels (x axis), with 100 traces per log, and a maximum of 100 events per trace, for one of the models mined from the robot soccer data. Since this figure is representative for the larger set of experiments, we can use it to illustrate our main conclusions from the experiments.

1. *Existing metrics have a bias when applied to simple models.* We can see that the Token Based fitness, which stands for other similar fitness approaches in the process mining domain, does not drop to much less than a fitness of 0.4 throughout the whole experiment. This can be explained by the fact that the log replay "leaves tokens behind" for potential later use, which is appropriate for models containing parallelism (as one needs to look more than one step forward to satisfy all the dependencies) but not for simple Petri nets as evaluated in our experiments. Thus, in these situations, and more severely with an increasing level of noise, this metric provides overly optimistic results.
2. *Metrics should measure only one thing.* We can see that the *Improved Continuous* fitness suddenly drops dramatically when the noise level is increased above 40%. This can be explained by the fact that this metric was designed to steer a genetic algorithm based on a mixture of different measurements. For that purpose, it is a good metric. However, if one wants to use the metric to gain insight into the quality of a process model, it becomes difficult as the interpretation of results becomes difficult ("Which aspect contributed to this value?", "Is it the property I want to measure?").

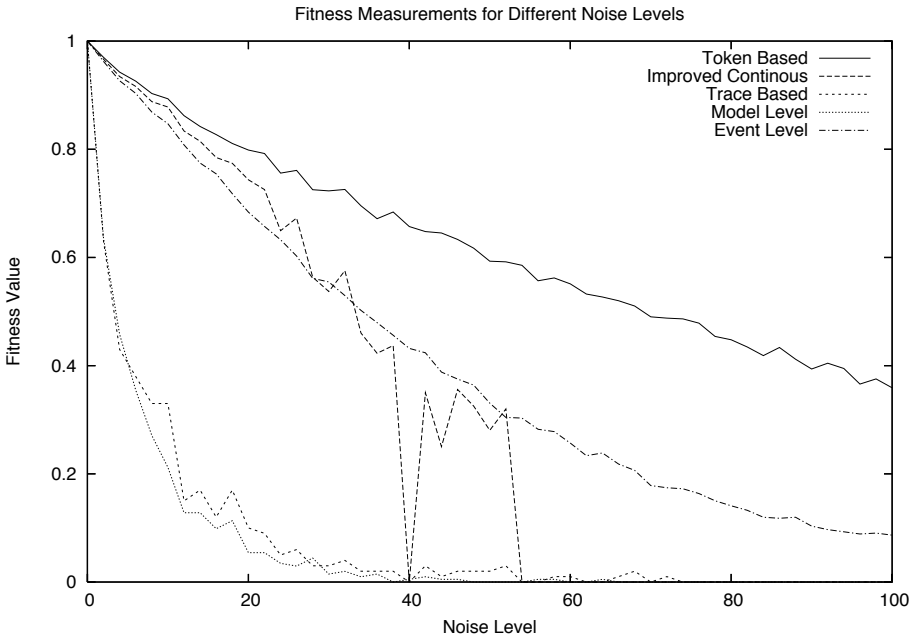


Fig. 14. Fitness values for 50 different observation noise levels on 'BSmart' model from Appendix B.

3. *Trace-based metrics do not make much sense for large models and long process instances.* Based on the *Trace Based* fitness, we can see that this metric drops very quickly towards 0, already for low levels of noise. This is due to the fact that in case of longer process instances (as here up to 100 events per trace) already one small error in the trace renders the whole trace to have a negative impact on the overall fitness. With an increasing level of noise, there will be soon no more traces that have no errors, thus rendering the measure to provide pessimistic results if compared to our notion of noise.

The underlying assumption here is that a fitness metric should decrease as uniformly as possible with an increasing portion of noise, which is what one would intuitively expect without further knowledge of the metric's behavior. We can translate this assumption into the requirement that a fitness metric should be inversely proportional to the percentage of noise. If we further normalize the ratio of noise and inverse fitness by subtracting 1, then we obtain an indicator that yields 0 in the *ideal* situation, positive values if the fitness measurement is *optimistic* given the noise level, and negative values in the case that the fitness measurement is *pessimistic* given the noise level.

For example, the average noise fitness ratios for one of the models are shown in Table 1.

Definition 10 (Noise Fitness Ratio) *With a fitness value f being calculated for a log with n degrees of noise ($n = 1$ corresponding to 100% of noise), the fitness noise ratio r_{fn} is defined as follows:*

$$r_{fn} = \begin{cases} \frac{n}{1-f} - 1 & \text{if } f \neq 1 \\ \frac{f}{1-n} - 1 & \text{if } f = 1 \end{cases}, \ 0 \leq f, n \leq 1$$

Table 1. Average noise fitness ratios during the replicated experiments for model '1Be_2xk1' from Appendix D with (1) observation noise and (2) transition noise.

	5% Noise	*10% Noise*	*20% Noise*	*50% Noise*	*100% Noise*
Token	(1) -0.201	(1) -0.168	(1) -0.111	(1) 0.050	(1) 0.516
Based	(2) 0.292	(2) 0.568	(2) 0.463	(2) 0.433	(2) 0.567
Improved	(1) -0.376	(1) -0.333	(1) -0.264	(1) -0.216	(1) 0.0
Continuous	(2) -0.024	(2) 0.219	(2) 0.153	(2) 0.132	(2) 0.0
Trace	(1) -0.716	(1) -0.677	(1) -0.614	(1) -0.444	(1) 0.0131
Based	(2) -0.704	(2) -0.615	(2) -0.582	(2) -0.365	(2) 0.141
Model	(1) -0.853	(1) -0.81	(1) -0.72	(1) -0.465	(1) 0.029
Level	(2) -0.827	(2) -0.771	(2) -0.728	(2) -0.496	(2) 0.0
Event	(1) -0.468	(1) -0.452	(1) -0.380	(1) -0.284	(1) 0.112
Level	(2) -0.251	(2) -0.076	(2) -0.105	(2) -0.046	(2) 0.0

In the appendices we provide graphs of the fitness values for different observation and transition noise levels. They visualize consistently that the *Token Based* metric yields optimistic results for logs containing a high portion of noise, and that the *Trace Based* metric and *Model Level* metric yield rather pessimistic results. Furthermore, if we calculate the average variance of the fitness noise ratios of the replications within one noise level, then the *Improved Continuous* metric consistently yields an average variance which is about one magnitude higher than the average variance values of the remaining metrics, thus highlighting its instability. The best metrics seem to be the *Event Level* metric.

Overall, the results for the two different kinds of noise are very similar and confirm the same trend. Two differences can be noted:

- If the transition noise has a level 100%, then the *Event Level* metric really falls to the value 0, which is not the case for the observation noise. The reason is that by generating random observation elements, it may be still the case that, accidentally, correct sub sequences are created. This is similar for noise generation mechanism typically used in the process mining domain, such as deleting and exchanging log events as defined in [33], but it is not true for the transition noise, which with a noise level of 100% is guaranteed to produce no valid sub sequence anymore.
- The *Trace Based* metric often drops not as quickly for the transition noise as for the observation noise. This may be due to the fact that among the

noisy transitions is also the transition to the final state, which then leads to the end of the event sequence. Such shorter process instances are then less likely to have an error than longer sequences.

Of course these results are influenced by our notion of noise and our assumption that it is desirable for a fitness metric to scale as linearly as possible with respect to the degree of distortion in the log. In principle, there may be different notions of noise (e.g., introducing distortions gradually over different parts of the process), which would render the *Model Level* metric less pessimistic, or one might prefer the metric to scale linearly from 0% (metric yields 1) to 20% noise (metric yields 0), since lower portions of noise (e.g., 5% or 10%) are much more common in real processes than a very high portion (e.g., 80%). However, for a first general evaluation the presented setup seems reasonable. In any case, knowledge about the behavior of a metric is important as we interpret the calculated values correspondingly ("How good or bad is a result of 0.7?").

9 Discussion

We generally use metrics to obtain information about the quality of a process model. But to be sure that the conclusions that we draw from the measurements are valid, we must first ensure the *quality of the metric itself*. The following requirements are are generally considered relevant [23] to ensure the usefulness of a metric: *validity* (the measure and the property to measure must be sufficiently correlated), *reproducibility* (be independent of subjective influence), *stability* (be as little as possible affected by properties that are *not* measured) and *analyzability* (relates to the properties of the measured values).

Given the fact that the validity and reproducibility are typically not an issue, in this paper we have proposed a structured approach to evaluate the analyzability and stability of certain process mining metrics. As a first step, we focused on simple process models and the fitness and precision quality dimensions. We have seen that—in this simple setting—existing process mining metrics can yield overly optimistic results, which affects their *analyzability*. Furthermore, some metrics measure more than one aspect, which makes interpretation of the results difficult and negatively affects their *stability*. Further research is required to develop evaluation approaches for further dimensions and more complex scenarios. The ultimate vision for process model evaluation would then be to have a methodology that assists in selecting the "right" metric for the "right" situation, and based on the goal of the evaluation.

In this paper, we have used HMMs to define quality metrics for process models, and to generate logs with determined levels of noise. Possible next steps could be the following:

– Carry out experiments for varying degrees of completeness to evaluate existing precision metrics. For example, the log completeness metric defined in Section 5.2 could be iteratively calculated while generating logs for a determined level of completeness.

– Perform experiments with Petri nets that contain parallelism using our one-state-per-labeled-task mapping to see how optimistic the fitness and how pessimistic the precision measurements really are.

Finally, further applications of HMMs in the process mining area could be explored as indicated in Section 6.

Acknowledgements

This research is supported by the IOP program of the Dutch Ministry of Economic Affairs. The authors also thank the CMDragons team for sharing their RoboCup log data, and particularly Stefan Zickler and Douglas Vail who helped to create the ASCII logs used for conversion to the MXML format. Furthermore, we thank the town hall in the Netherlands for letting us use their process models, and Florian Gottschalk and Boudewijn van Dongen for the EPC reference models. Finally, we thank all ProM developers for their on-going work on process mining techniques. Special thanks to Christian W. Günther for NikeFS2, which makes it possible to work with large real-world logs, and for his UI redesign, which continues to impress people and makes using ProM so much more fun.

References

1. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
2. W.M.P. van der Aalst, V. Rubin, B.F. van Dongen, E. Kindler, and C.W. Günther. Process Mining: A Two-Step Approach using Transition Systems and Regions. BPM Center Report BPM-06-30, BPMcenter.org, 2006.
3. 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.
4. 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.
5. 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.
6. L.E. Baum and J.A. Egon. An Inequality with Applications to Statistical Estimation for Probabilistic Functions of a Markov Process and to a Model for Ecology. *Bull. Amer. Meteorology Soc.*, 73:360–363, 1967.
7. L.E. Baum, T. Petrie, G. Soules, and N. Weiss. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *Ann. Math. Statist.*, 41(1):164–171, 1970.
8. L.E. Baum and G.R. Sell. Growth Functions for Transformations on Manifolds. *Pac. J. Math.*, 27(2):211–227, 1968.
9. B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer Verlag, 2001.

10. 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.
11. A.K. Alves de Medeiros. *Genetic Process Mining*. PhD thesis, Department of Technology Management, Technical University Eindhoven, 2006.
12. A.K. Alves de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2006.
13. J. Desel, W. Reisig, and G. Rozenberg, editors. *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2004.
14. B.F. van Dongen and W.M.P. van der Aalst. Multi-Phase Process Mining: Building Instance Graphs. In P. Atzeni, W. Chu, H. Lu, S. Zhou, and T.W. Ling, editors, *International Conference on Conceptual Modeling (ER 2004)*, volume 3288 of *Lecture Notes in Computer Science*, pages 362–376. Springer-Verlag, Berlin, 2004.
15. G.D. Forney. The Viterbi Algorithm. *Proceedings IEEE*, 61(3):268–278, 1973.
16. E.M. Gold. Complexity of Automaton Identification from Given Data. *Information and Control*, 37(3):302–320, 1978.
17. G. Greco, A. Guzzo, L. Pontieri, and D. Sacca. Discovering expressive process models by clustering log traces. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1010–1027, 2006.
18. 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.
19. K. Han and M. Veloso. Automated robot behavior recognition. 2008.
20. 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.
21. 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.
22. B.H. Juang and L.R. Rabiner. A Probabilistic Distance Measure for Hidden Markov Models. *AT&T Technical Journal*, 64(2):391–408, 1985.
23. P. Liggesmeyer. *Software-Qualität – Testen, Analysieren und Verifizieren von Software*. Spektrum Akademischer Verlag, Heidelberg, Berlin, 2002.
24. J. Mendling, M. Moser, G. Neumann, H.M.W. Verbeek, van B.F. Dongen, and W.M.P. van der Aalst. Faulty EPCs in the SAP Reference Model. In S. Dustdar, J.L. Fiadeiro, and A.P. Sheth, editors, *International Conference on Business Process Management (BPM 2006)*, volume 4102 of *Lecture Notes in Computer Science*, pages 451–457. Springer-Verlag, Berlin, 2006.
25. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
26. L.R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
27. 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.
28. A. Rozinat, A.K. Alves de Medeiros, C.W. Günther, A.J.M.M. Weijters, and W.M.P. van der Aalst. Towards an Evaluation Framework for Process Mining Algorithms. BPM Center Report BPM-07-06, BPMcenter.org, 2007.

29. A. Rozinat, A.K. Alves de Medeiros, C.W. Günther, A.J.M.M. Weijters, and W.M.P. van der Aalst. The Need for a Process Mining Evaluation Framework in Research and Practice. In Arthur H. M. ter Hofstede, Boualem Benatallah, and Hye-Young Paik, editors, *Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*. Springer, 2008.
30. A. Rozinat, S. Zickler, M. Veloso, and W.M.P. van der Aalst. Analyzing Multi-agent Activity Logs Using Process Mining Techniques. Submitted to DARS'08, 2008.
31. J.M.E.M. van der Werf, B.F. van Dongen, K.M. van Hee, C.A.J. Hurkens, and A. Serebrenik. Process discovery using integer linear programming. In K.M. van Hee and R. Valk, editors, *Applications and Theory of Petri Nets (29th International Conference, Petri Nets 2008, Xi'an, China, June 23-27, 2008, Proceedings)*, volume 5062 of *LNCS*, pages 368–387. Springer Verlag, Berlin, 2008.
32. A.J. Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimal Decoding Algorithm. *IEEE Transactions on Information Theory*, 183(2):260–269, 1967.
33. 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.
34. L. Wen, J. Wang, and J.G. Sun. Detecting Implicit Dependencies Between Tasks from Event Logs. In *Asia-Pacific Web Conference on Frontiers of WWW Research and Development (APWeb 2006)*, Lecture Notes in Computer Science, pages 591–603. Springer, 2006.

A Small Petri net Model with Very Complex State Space

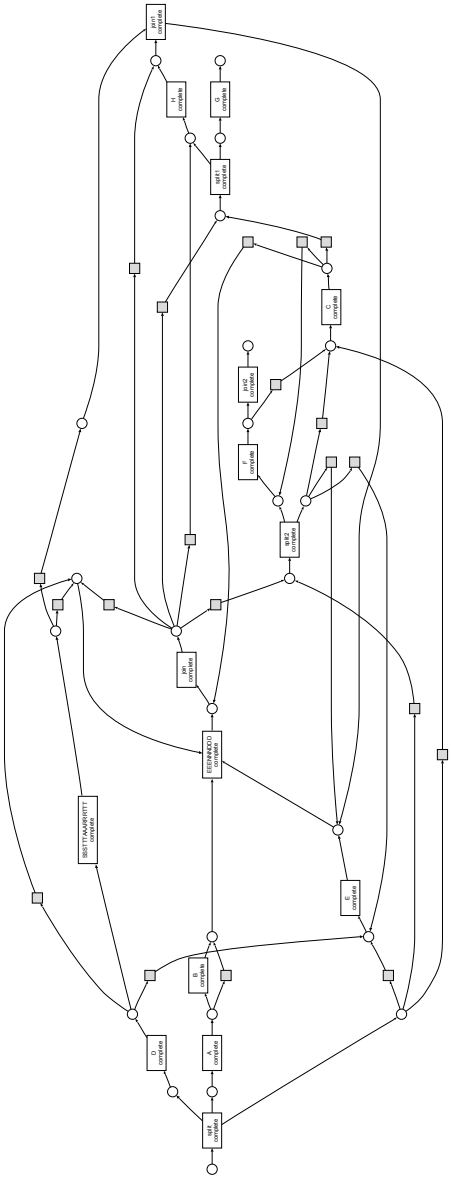


Fig. 15. Relatively small Petri net model, which already produces a coverability graph that cannot be computed with contemporary personal hardware.

B Robot Soccer Model Evaluations

The models used in these experiments were mined based on log data collected by the CMDragons team during the international robot soccer competition 'RoboCup' 2007, and thus constitute models of the behavior in a multi-agent robotic system [30]. The data was gathered over four round robin games, followed by the quarter-final, semi-final, and final games. The models were discovered using the Heuristics Miner and converted to a Petri net in ProM, and from all 7 models we selected those 4 that fulfilled the Simple Petri net requirement.

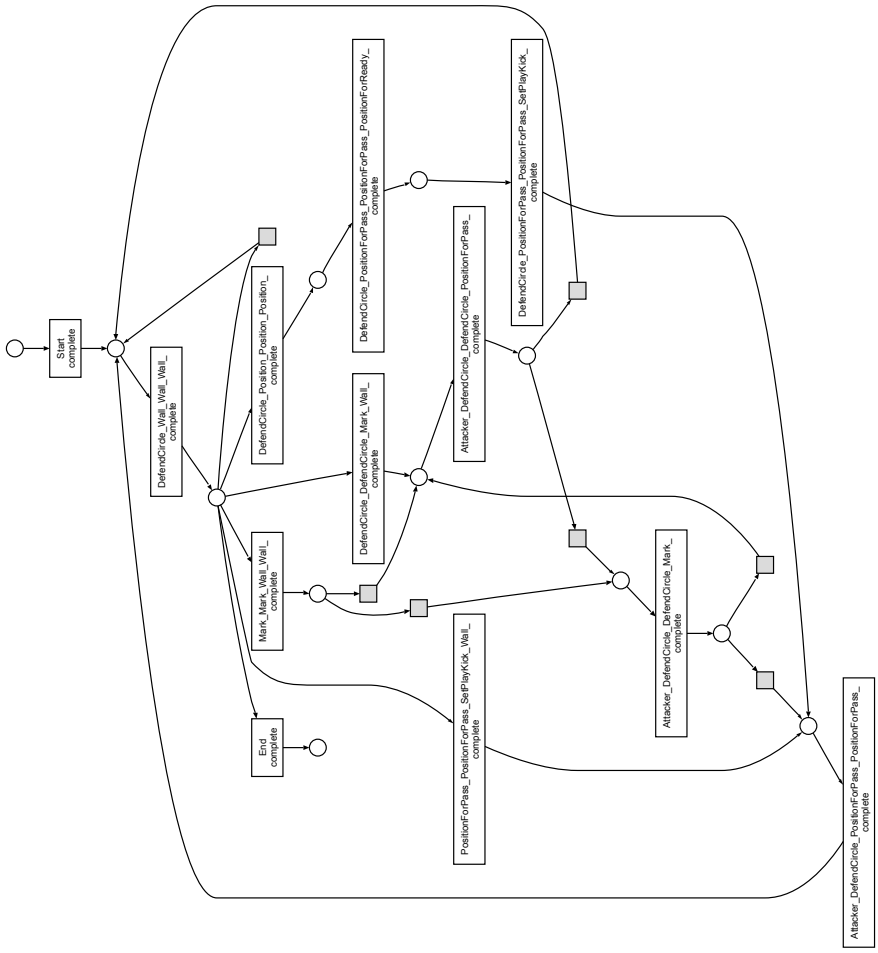
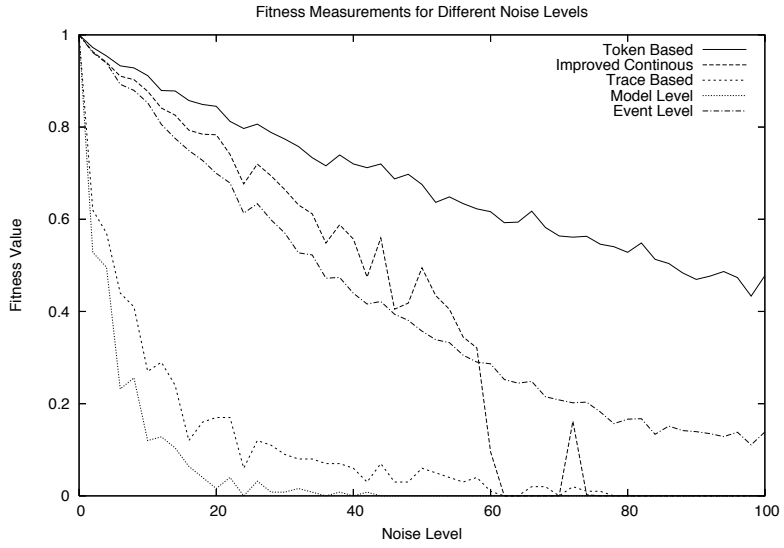
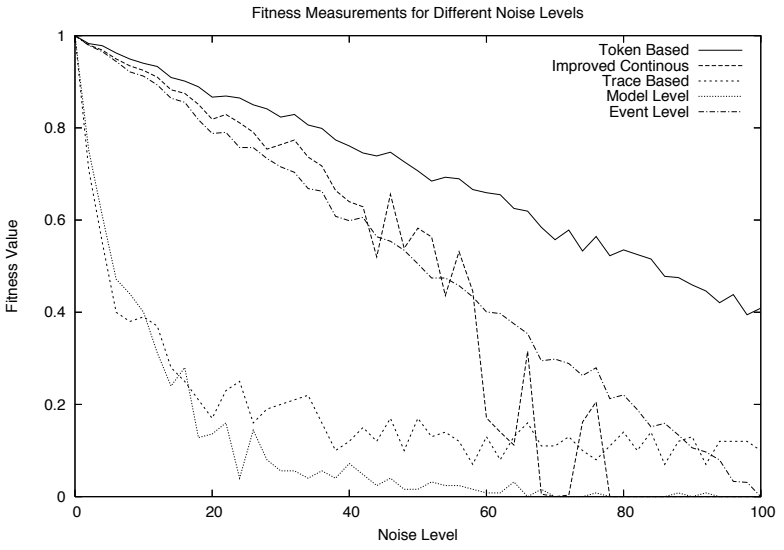


Fig. 16. The Petri net model mined from the round robin game between "CMDragons" and "Botnia".



(a) Fitness values for 50 different *observation noise* levels, 100 traces per log, and maximum 500 events per trace.



(b) Fitness values for 50 different *transition noise* levels, 100 traces per log, and maximum 500 events per trace.

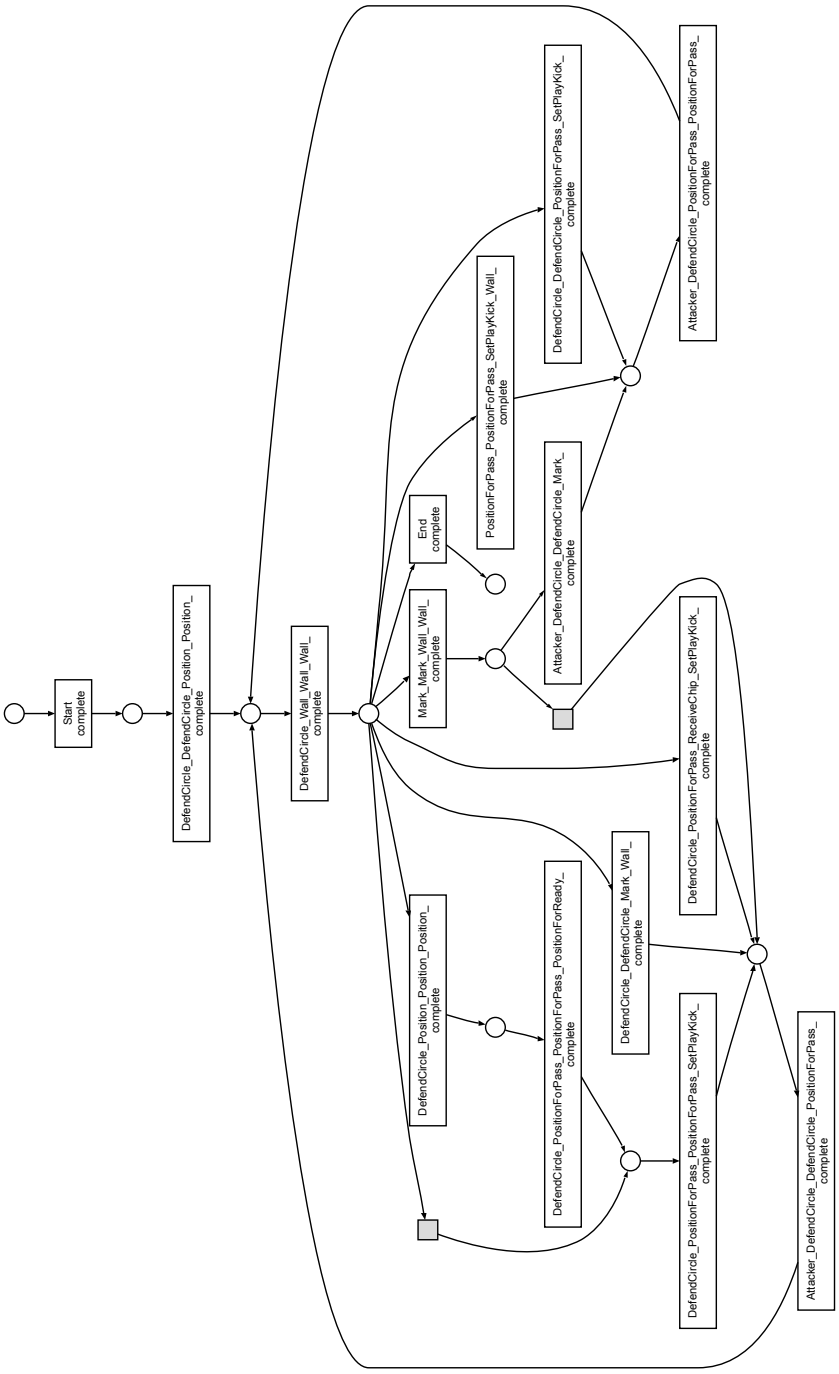
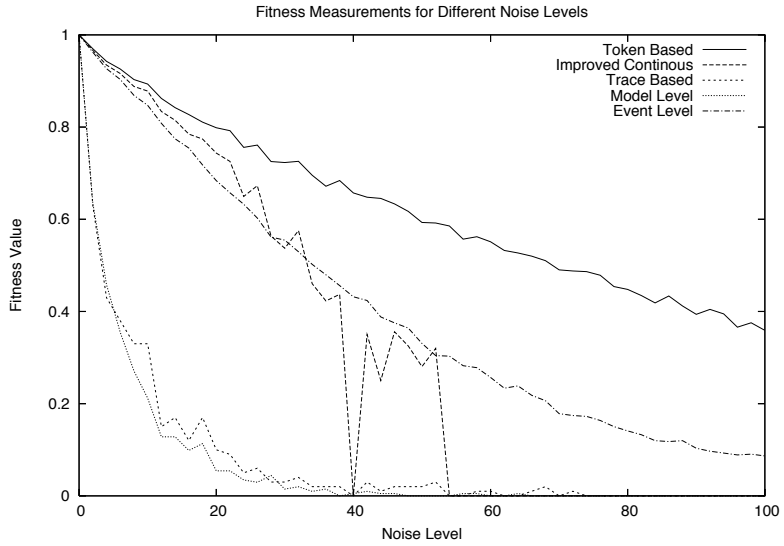
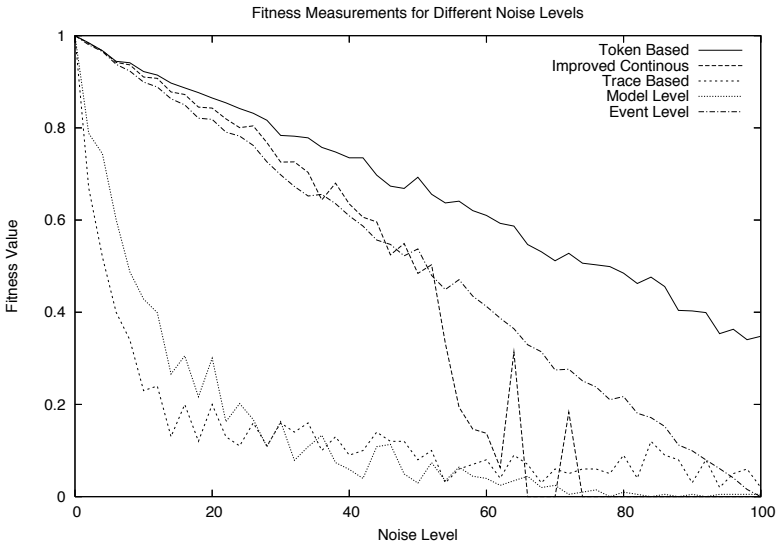


Fig. 17. The Petri net model mined from the round robin game between "CMDragons" and "BSmart".



(a) Fitness values for 50 different *observation noise* levels, 100 traces per log, and maximum 500 events per trace.



(b) Fitness values for 50 different *transition noise* levels, 100 traces per log, and maximum 500 events per trace.

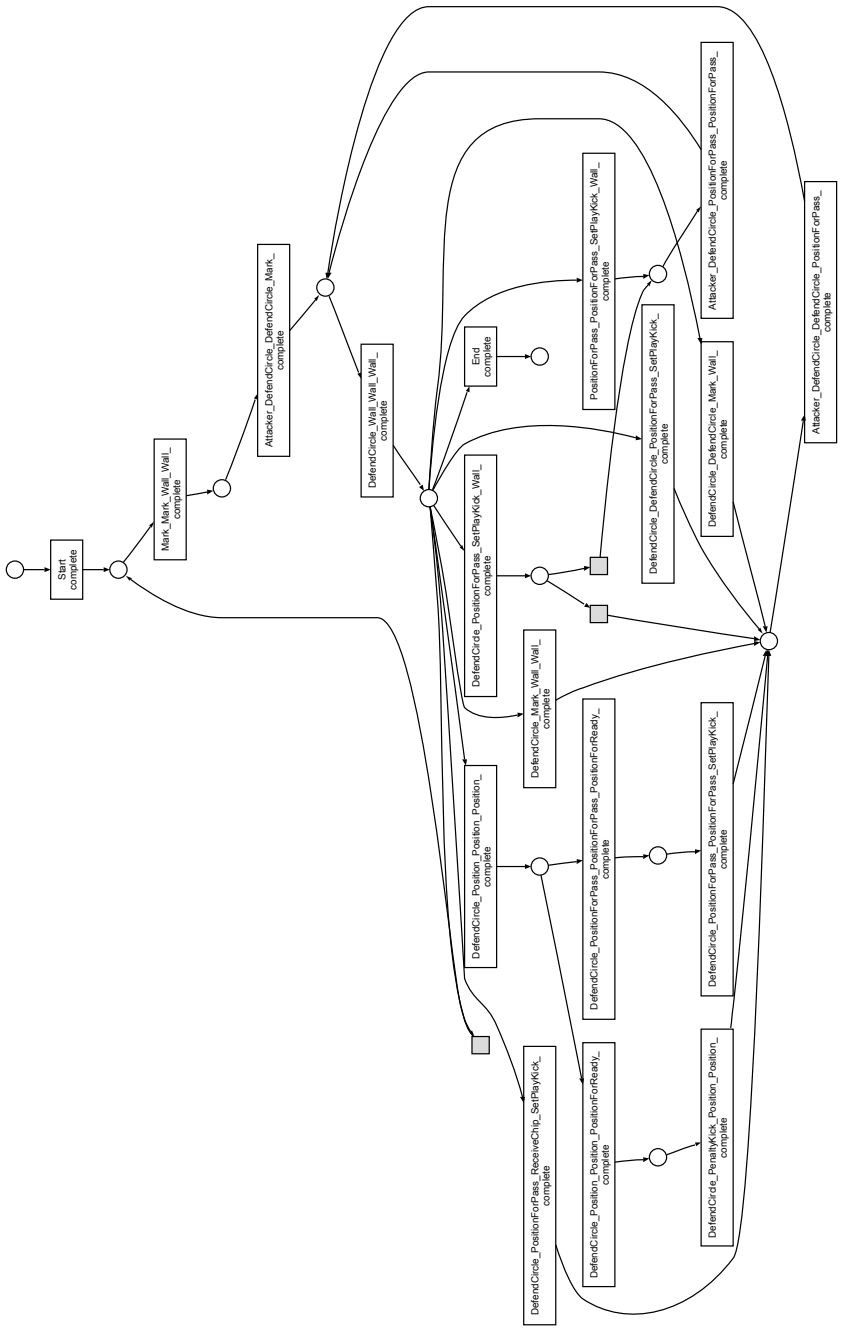
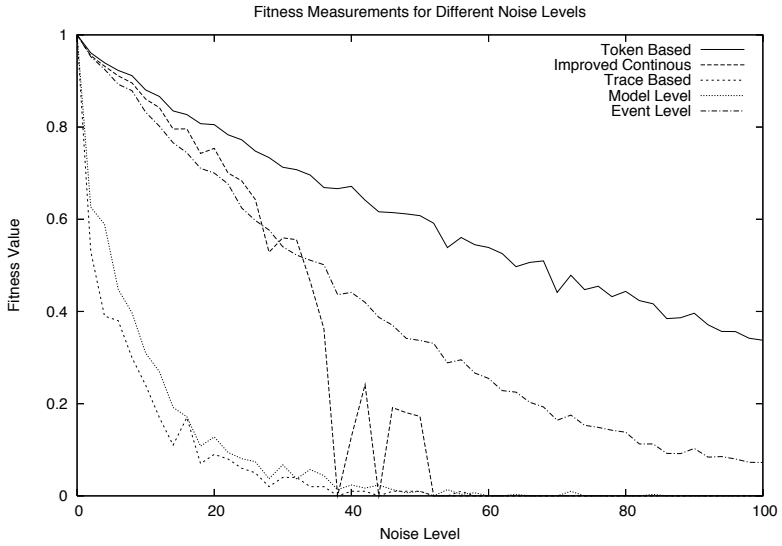
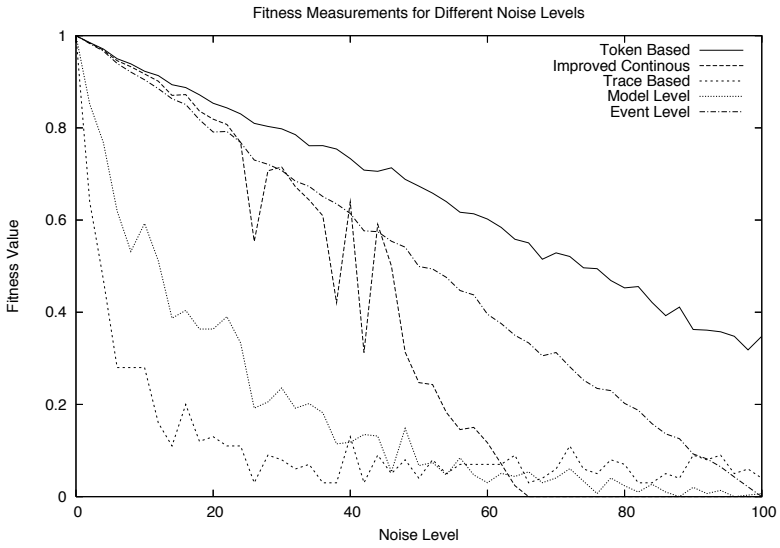


Fig. 18. The Petri net model mined from the quarter final game between "CMDragons" and "EagleKnights".



(a) Fitness values for 50 different *observation noise* levels, 100 traces per log, and maximum 500 events per trace.



(b) Fitness values for 50 different *transition noise* levels, 100 traces per log, and maximum 500 events per trace.

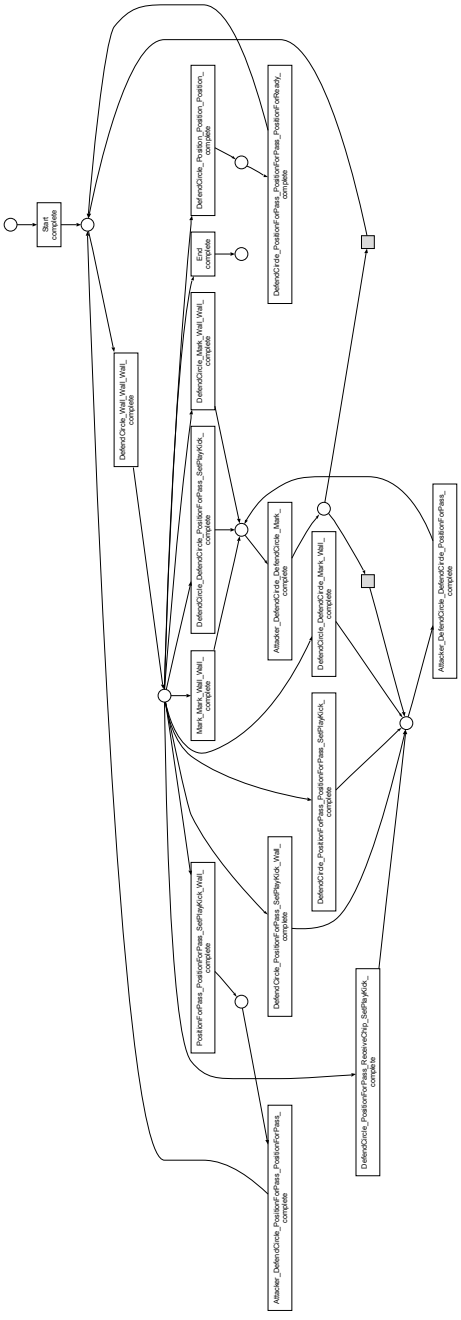
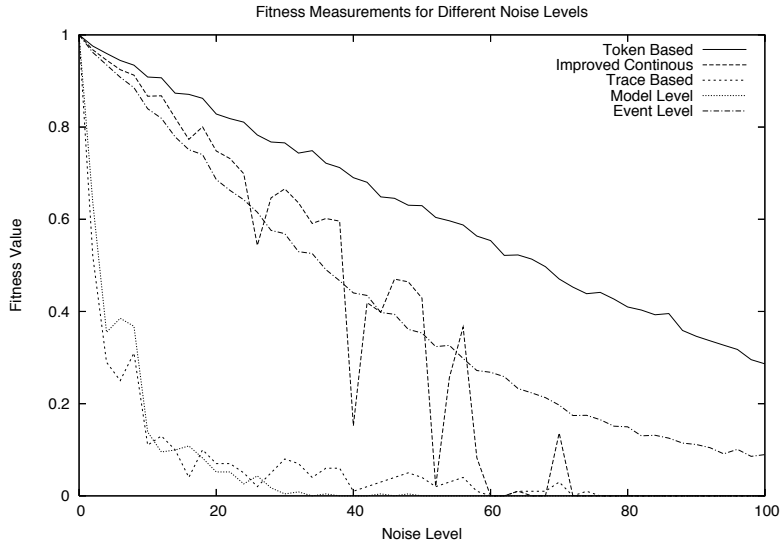
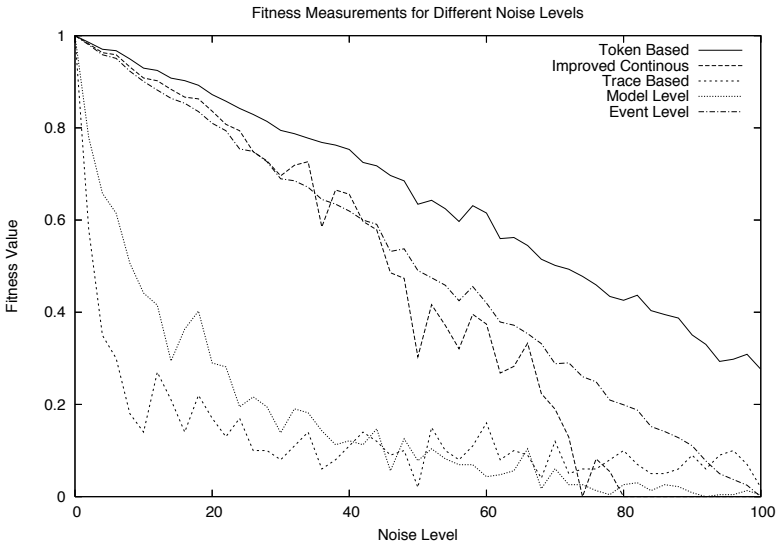


Fig. 19. The Petri net model mined from the round robin game between "CMDragons" and "WrightEagle".



(a) Fitness values for 50 different *observation noise* levels, 100 traces per log, and maximum 500 events per trace.



(b) Fitness values for 50 different *transition noise* levels, 100 traces per log, and maximum 500 events per trace.

C Town Hall Model Evaluations

Some of the process mining experiments in [12] were performed on real-life logs from administrative processes at a town hall in the Netherlands. Furthermore, the original process descriptions of these processes were available in the form of models. Three out of four of these process models fulfill the conditions of Simple Petri nets as defined in this paper, and we used them for our experiments. They all deal with the handling of complaints.

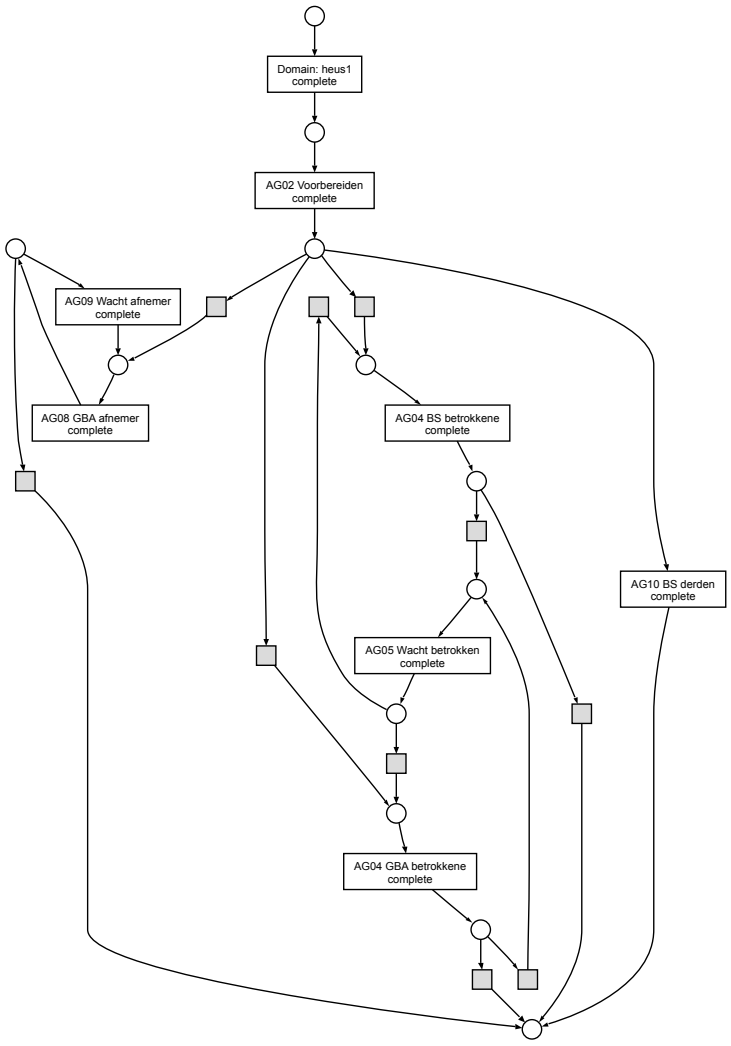
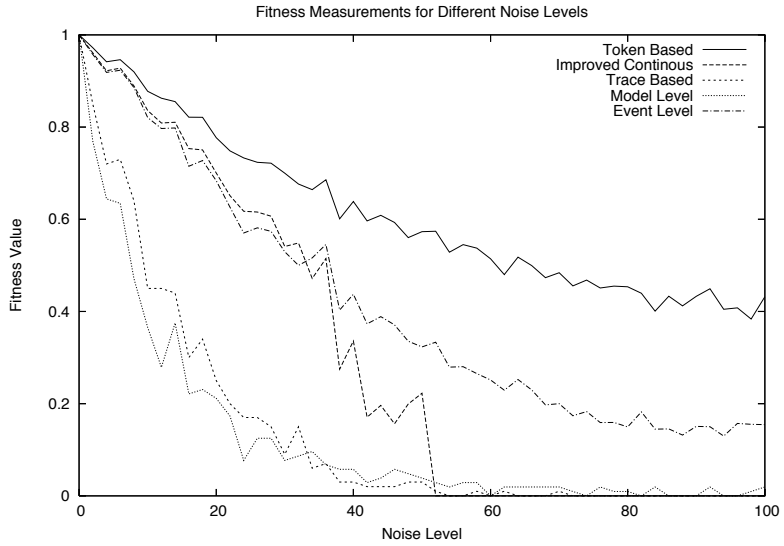
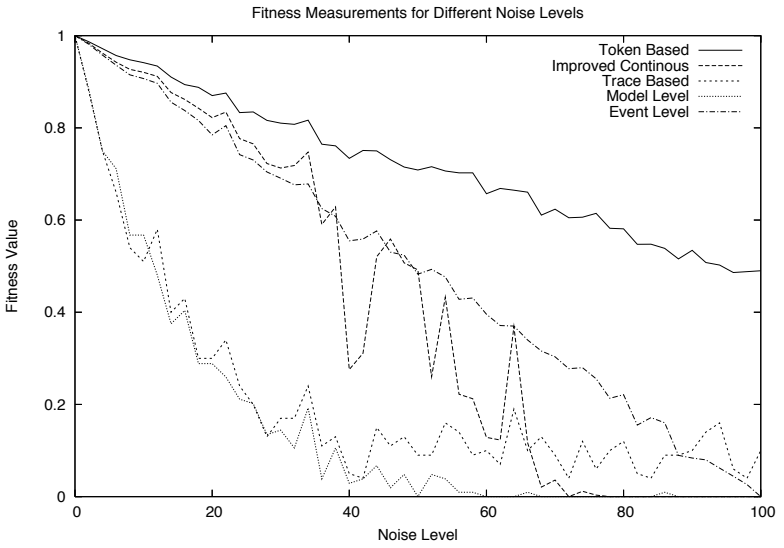


Fig. 20. The Petri net model depicting the 'afschriften' process in the town hall.



(a) Fitness values for 50 different *observation noise* levels, 100 traces per log, and maximum 500 events per trace.



(b) Fitness values for 50 different *transition noise* levels, 100 traces per log, and maximum 500 events per trace.

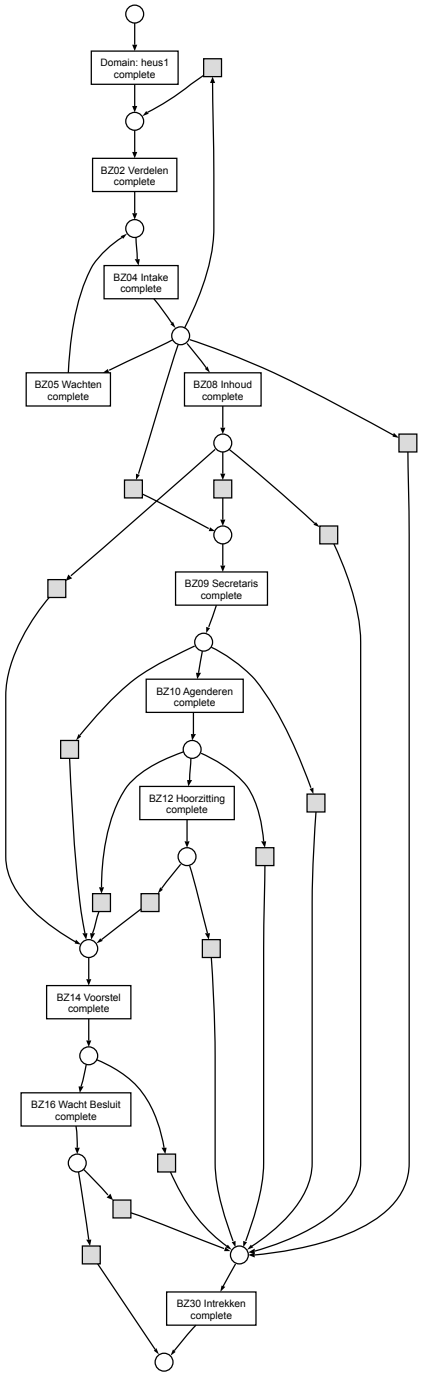
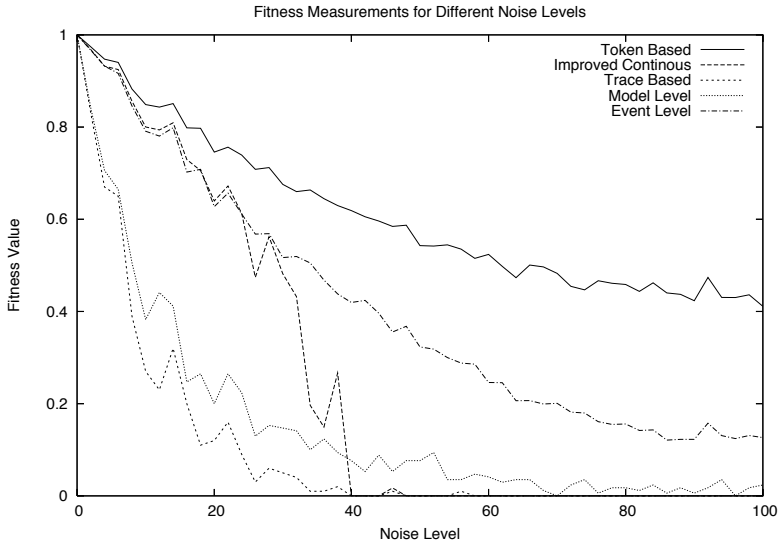
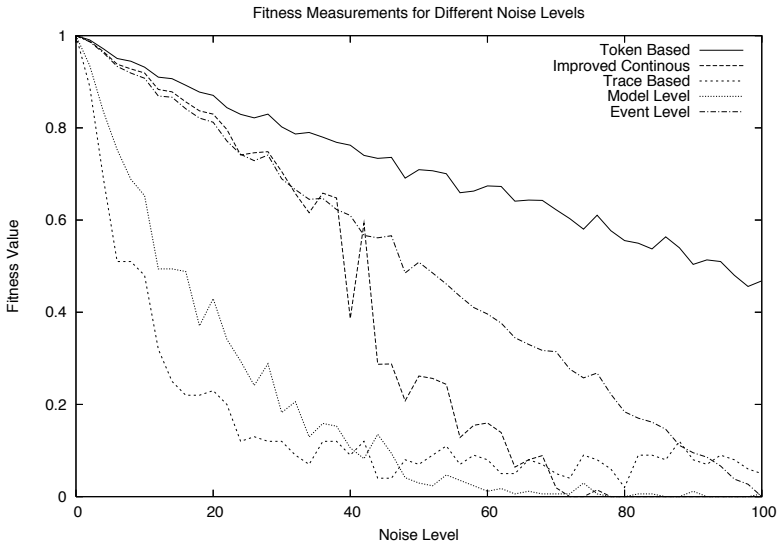


Fig. 21. The Petri net model depicting the 'bezwaar' process in the town hall.



(a) Fitness values for 50 different *observation noise* levels, 100 traces per log, and maximum 500 events per trace.



(b) Fitness values for 50 different *transition noise* levels, 100 traces per log, and maximum 500 events per trace.

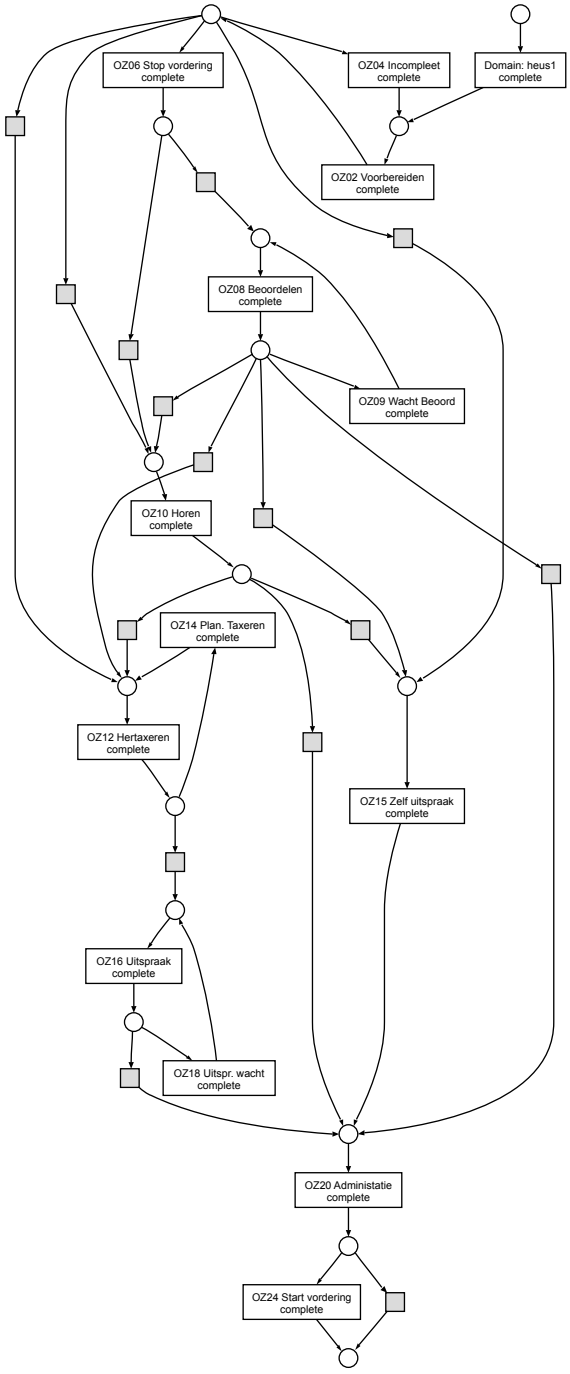
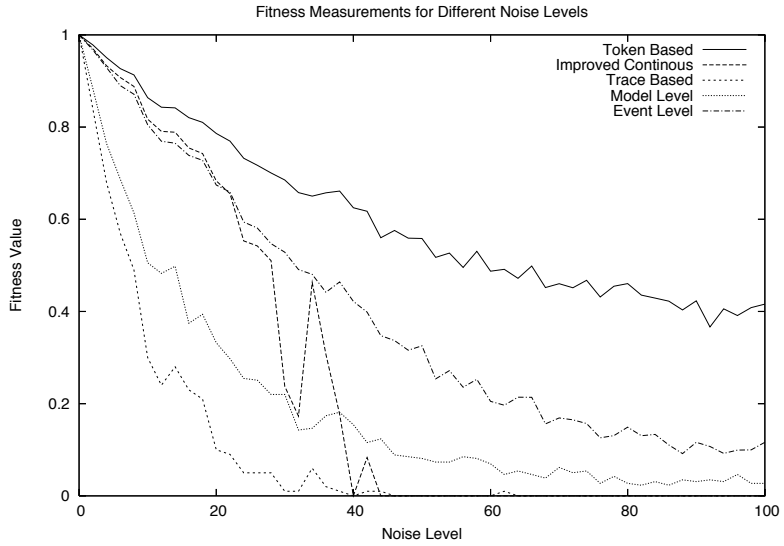
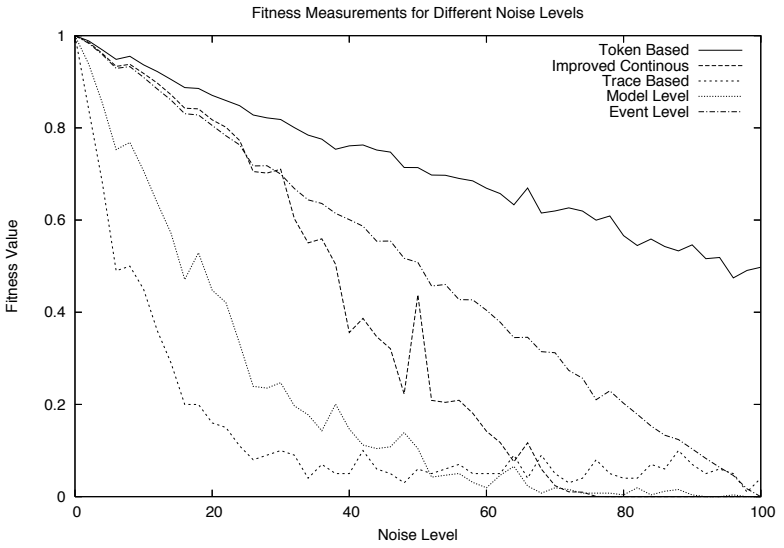


Fig. 22. The Petri net model depicting the 'bezwaarWOZ' process in the town hall.



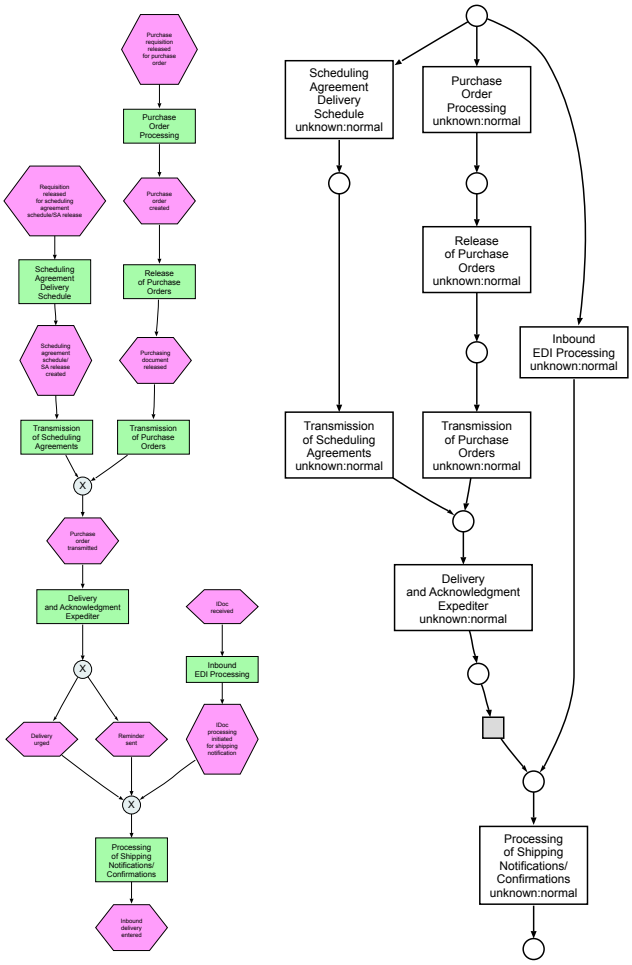
(a) Fitness values for 50 different *observation noise* levels, 100 traces per log, and maximum 500 events per trace.



(b) Fitness values for 50 different *transition noise* levels, 100 traces per log, and maximum 500 events per trace.

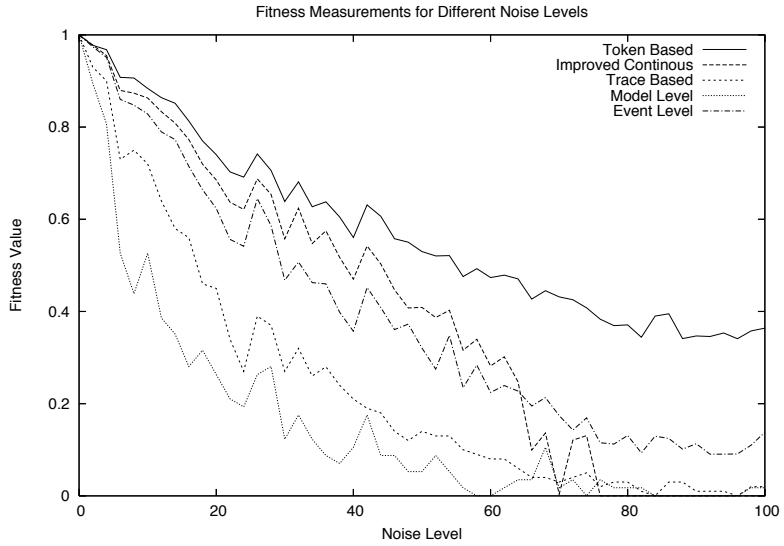
D EPC Reference Model Evaluations

The *SAP reference model* is a publically available model that contains more than 600 enterprise models expressed in terms of *Event-driven Process Chains* (EPCs) [24], of which—after conversion to Petri nets—118 fulfill the conditions of Simple Petri nets as defined in this paper. Most of these 118 models are trivial models, with only 8 models containing more than 4 tasks (i.e., *functions* in terms of EPCs). We selected the 3 most complex models from these 8 enterprise models for our experiment.

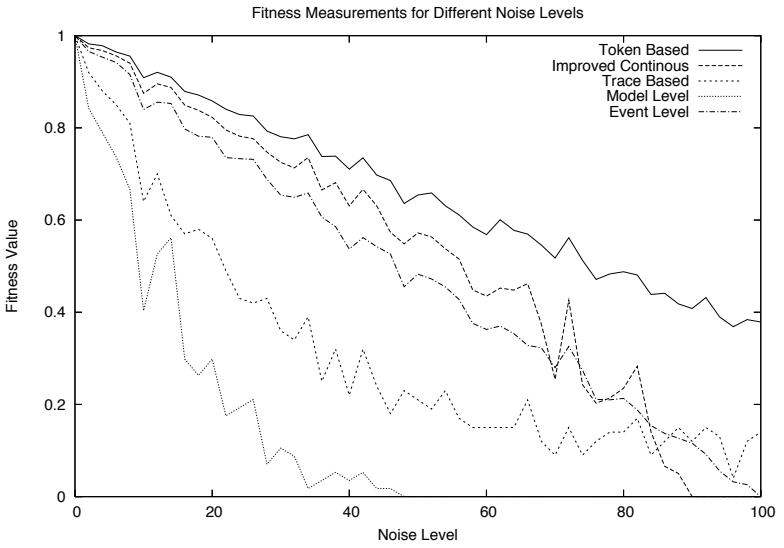


(c) EPC model (d) Converted Petri net model

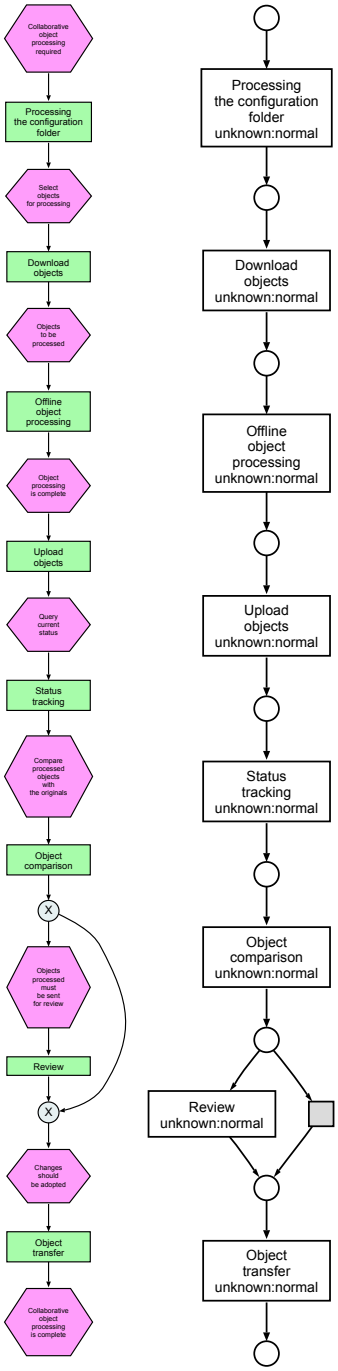
Fig. 23. The EPC '1Be_2xk1' from the SAP reference model.



(a) Fitness values for 50 different *observation noise* levels, 100 traces per log, and maximum 500 events per trace.

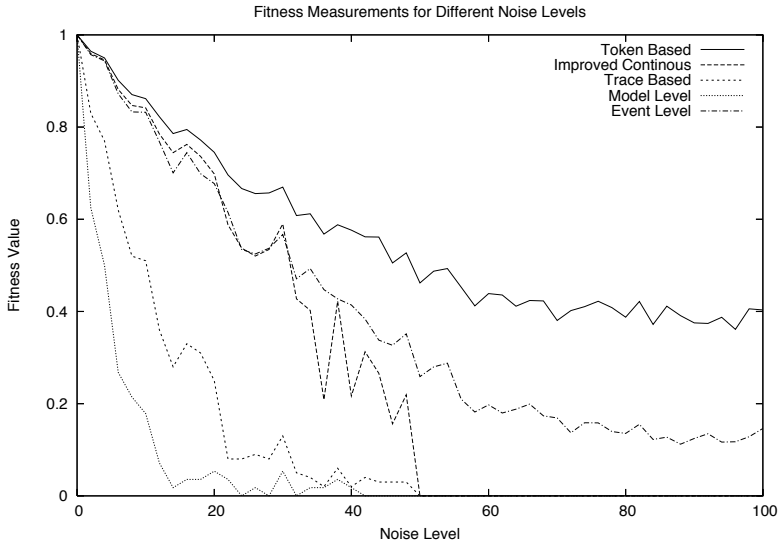


(b) Fitness values for 50 different *transition noise* levels, 100 traces per log, and maximum 500 events per trace.

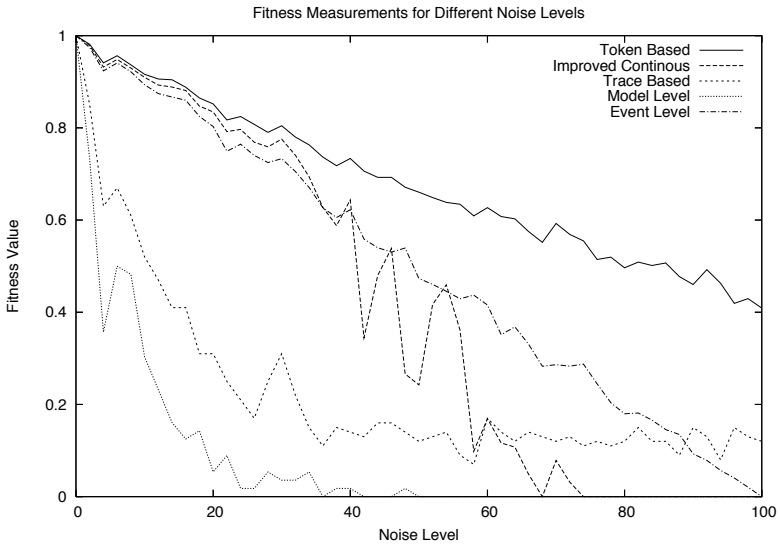


(c) EPC (d) Petri net

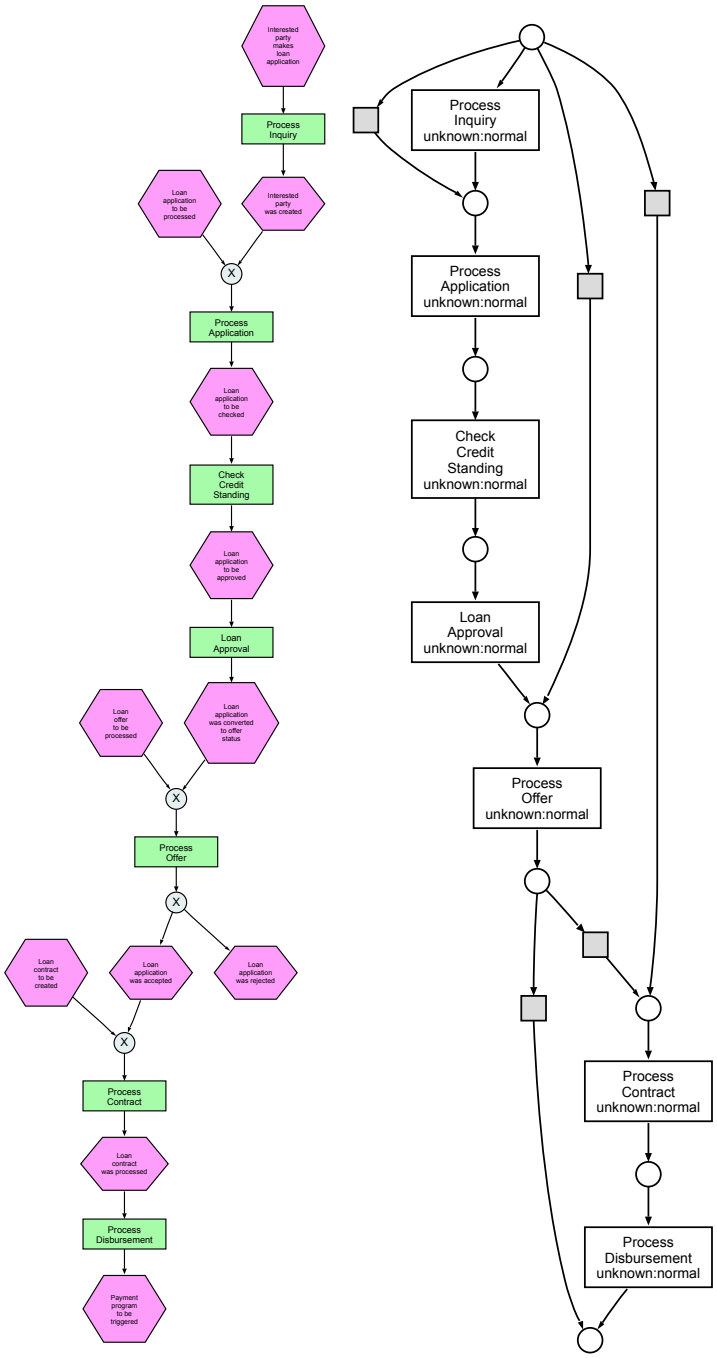
Fig. 24. The EPC '1Pr_mpk-' from the SAP reference model.



(a) Fitness values for 50 different *observation noise* levels, 100 traces per log, and maximum 500 events per trace.

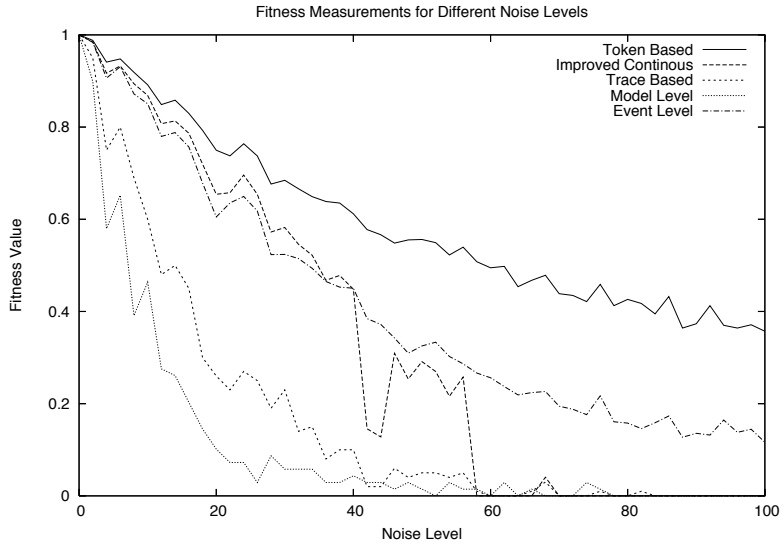


(b) Fitness values for 50 different *transition noise* levels, 100 traces per log, and maximum 500 events per trace.

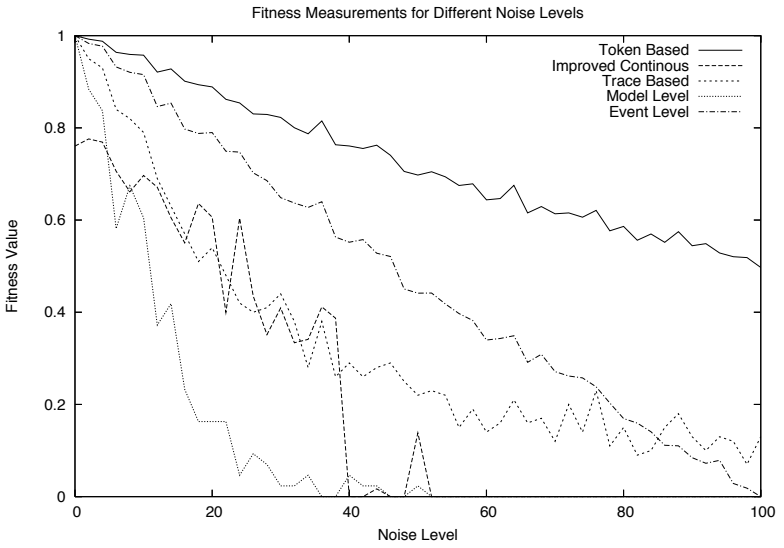


(c) EPC (d) Petri net

Fig. 25. The EPC '1Tr_fyhp' from the SAP reference model.



(a) Fitness values for 50 different *observation noise* levels, 100 traces per log, and maximum 500 events per trace.



(b) Fitness values for 50 different *transition noise* levels, 100 traces per log, and maximum 500 events per trace.