ISSN 0146-4116, Automatic Control and Computer Sciences, 2016, Vol. 50, No. 7, pp. 460–470. © Allerton Press, Inc., 2016.
Original Russian Text © A.A. Mitsyuk, I.S. Shugurov, 2014, published in Modelirovanie i Analiz Informatsionnykh Sistem, 2014, Vol. 21, No. 4, pp. 181–198.

On Process Model Synthesis Based on Event Logs with Noise

A. A. Mitsyuk* and I. S. Shugurov**

National Research University Higher School of Economics, Laboratory of Process-Aware Information Systems, Moscow, 125319 Russia

**e-mail: amitsyuk@hse.ru*

***e-mail: shugurov94@gmail.com*

Received August 23, 2014

Abstract—Process mining is a new emerging discipline related to process management, formal process modelling, and data mining. One of the main tasks of process mining is model synthesis (discovery) based on event logs. A wide range of algorithms for process model discovery, analysis, and enhancement is developed. The real-life event logs often contain noise of different types. In this paper, we describe the main causes of noise in the event logs and study the effect of noise on the performance of process discovery algorithms. The experimental results of application of the main process discovery algorithms to artificial event logs with noise are provided. Specially generated event logs with noise of different types were processed using the four basic discovery techniques. Although modern algorithms can cope with some types of noise, in most cases, their use does not lead to obtaining a satisfactory result. Thus, there is a need for more sophisticated algorithms to deal with noise of different types.

Keywords: process mining, Petri net, event log, event log generation, ProM

DOI: 10.3103/S0146411616070154

INTRODUCTION

Process mining is a generic name for different methods of process models synthesis, analysis, and enhancement by working with event logs of information systems (and other types of systems). There are three main branches of process mining, i.e., (1) process model synthesis, (2) checking fitness between process models and event logs, and (3) enhancement of process models using various information from event logs and other sources [1]. Process event logs provide initial information for applying model discovery approaches; models and event logs - for fitness checks; model, event logs, and (probably) additional sources provide initial information for enhancement. A lot of algorithms were developed for solving problems in each of these three categories [1–4, 9–12, 22, 23]. The Alpha algorithm [2] is the first and the simplest algorithm of process model discovery. There are many variations of this algorithm.

Among other mining algorithms, there are three main ones that define whole classes of approaches. Heuristic approaches use frequency characteristics of a process to synthesize a model [23]. Algorithms using linear programming are based on theory of regions [22]. Inductive algorithms are intended for synthesis of structured models [11]. Applying a mining algorithm can result in a process model represented in one of the formalisms: Petri net, workflow net (WF-net), BPMN net, heuristic net, and others [1].

All these algorithms were implemented as extensions of ProM Framework, a model-mining environment [8, 13]. All four basic approaches of process model mining were used in this work: alpha algorithm, heuristic algorithms, linear programming algorithms, and an inductive algorithm. Algorithms versions for ProM 6.3 were used.

To make it possible to apply process mining, an event entry in a log must have at least two components, i.e., a name and a timestamp. The process model is built based on causal dependences. Event sequence is identified by its timestamps. Transitions (with names matching event names in the log) are placed sequentially in the model being discovered. In the case of identical event timestamps, the corresponding transitions are placed in parallel.

In addition to these necessary components an event entry can contain the others as well: names of actors, numeric data, records about resources involved in performing the operation. There are algorithms that use these components for process model discovery in other formalisms, e.g., in the form of social networks.

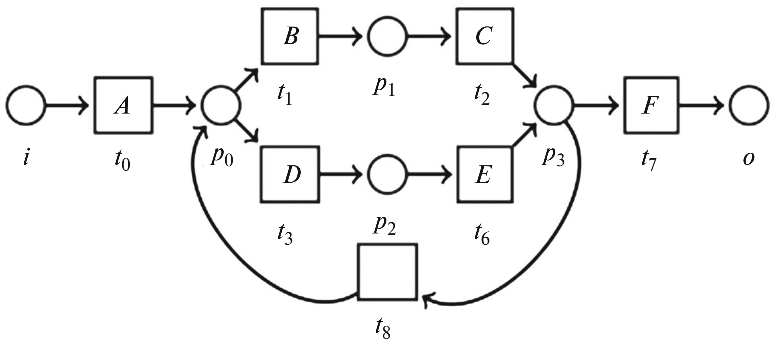

Fig. 1. Example of a workflow net.

In real life, information systems rarely perform perfectly [18, 19]. The outside world makes its adjustments, e.g., technical obstacles, malfunctions, and human error by people participating in process execution. All of these events influence the behavior of systems registered in an event log. The purpose of this paper is studying various noise encountered in event logs.

The main results of this work are as follows:

(1) The classification of different types of noise found in event logs has been presented.

(2) The results of experiments that study the effect of noise in event logs on the performance of process model discovery methods have been presented.

(3) The need to take into account various additional information for discovering high-quality models is justified.

(4) Strong and weak points of process-mining algorithms have been described.

1. WORKFLOW NETS

In this paper, we use workflow nets, a special class of Petri nets [2], to represent processes in information systems. A workflow net has one initial state and one final state. All other states are located on paths between them.

An example of a workflow net that models a simple medical process is shown in Fig. 1. The circles denote places in a net and the squares denote transitions. The initial place *i* and the final place *o* are shown. Places can contain filled circles depicting tokens in positions. The current marking is depicted by adding the necessary number of tokens (filled circles) into each place. The marking denotes current process state. A token at the initial place is the initial marking for a workflow net.

A possible process is described with the model in Fig. 1 as follows: a patient is being admitted to a hospital (marker in the initial position *i*). Based on the analysis results, specialists make a decision (*A*) about the further treatment i.e., (*B, C*) conservative or (*D, E*) interventional. After the course of treatment, the patient is either checked out of the hospital (*F*, marker in position *o*) or is readmitted for treatment (*G*).

The execution of an action or an operation in the process is being modeled by transition firing. Firing a transition consumes the necessary amount of tokens from entry places for the fired transition. At the same time, produced tokens are placed in the output positions for the corresponding transition. Thus, firing a transition changes the model marking (system state).

In a classical Petri net, transitions and places are nameless. However, the process analysis requires naming process actions or operations. Therefore, this paper considers labelled workflow net where every node has a name. A strict formal description workflow nets is provided, for e.g., in [1, 2].

2. EVENT LOGS, XES FORMAT

Event logs of information systems can be presented in different formats. In the general case, the event log is a set of text records that contain information about the events and actions of a process. Process mining employs a formalized definition of an event log [1]. A *log* consists of entries that correspond to events. As a rule, a process is executed repeatedly. Each log entry must belong to one and only one of the process executions. Event entries in a log are related to performing an operation of the process. If the process is

Table 1. Example of an event log fragment that corresponds to the model in Fig. 1

Trace	ID	Event name	Timestamp	Patient
1	1001141	A (admission, tests)	10-01-2014:15.12	Ivanov
1	1001142	B (admission into therapy)	10-01-2014:19.10	Ivanov
1	1501141	C (discharge from therapy)	15-01-2014:10.10	Ivanov
1	1601141	F (tests, discharge)	16-01-2014:9.14	Ivanov
2	1101141	A (admission, tests)	11-01-2014:18.12	Petrov
2	1101142	B (admission into therapy)	11-01-2014:20.00	Petrov
2	1301141	C (discharge from therapy)	13-01-2014:18.10	Petrov
2	1401141	G (decision on treatment)	14-01-2014:8.23	Petrov
3	1401142	A (admission, tests)	14-01-2014:15.00	Semenov
2	1401143	D (admission into surgery)	14-01-2014:16.10	Petrov
2	2001141	E (discharge from surgery)	20-01-2014:10.25	Petrov
2	2101145	F (tests, discharge)	21-01-2014:9.01	Petrov
...	...	...	...	...

modeled with a labelled workflow net, performing an operation corresponds to firing a transition with the same label in the net. Each event entry in a log contains not only the name of an operation, but also the timestamp that reflects time (and date) of operation execution. Event entries may contain additional information (names or roles of action performers, cost, numeric or text data). Table 1 shows a fragment of an event log that corresponds to the model in Fig. 1.

More formally, an event log is a multiset of traces where each trace contains exactly one execution of a process in form of event entries sequence. Each event entry contains operation name, timestamp, and other additional attributes. Each trace can be recorded in a log more than once [1].

There is an open-source format for storing event logs, OpenXES [14]. OpenXES is an XML schema that describes the format of text files that contain the behavior of an information system represented as an event flow. There is a sophisticated standard that describe the specifics of format usage. At the time of this paper was written, the newest version is XES Standard Definition 2.0 [15].

The standard inherits all XML advantages (and disadvantages), including simplicity of processing and verification, extensibility, and relatively convenient reading and writing operations. However, performance is not its strongest point. Storing markup requires much more memory than storing useful data. Search operations are performed slowly.

Most process-mining tools, for both research and scientific purposes, support the XES standard. In fact, this is the basic format used in ProM 6 Framework along with MXML. In purposes of this paper, we have also used XES to store event logs. It is important to note that the standard is extensible and, therefore, anyone can design custom event log formats. However, in this case, the operability of standard tools is intended for processing standardized event logs and cannot be guaranteed.

3. FITNESS OF MODELS AND EVENT LOGS

The quality of the information-system model is estimated, first of all, by its ability to reproduce the correct behavior of a system and to filter out incorrect behavior. For models of processes discovered from event logs, the key aspects of quality are as follows: (1) fitness, (2) simplicity, (3) precision, and (4) generalization [1].

The fitness of a model is only determined for a particular event log. An event log trace is considered to conform to a model if it can be obtained as a result of executing this model. An event log perfectly conforms to a model if all of its traces fit fully with this model. In real life, perfectly conformant log–model pairs are rare. Therefore a log is usually considered to conform to a model if a certain fraction of traces in this log conform with the model. The threshold is set by an expert in every specific case.

To calculate fitness, the model consistently executes an event log. The initial model marking is being configured (one token in initial position). The algorithm consequently processes every log trace. For every consequent event, there are two options, i.e., (1) a transition with a corresponding label is active and can be fired or (2) there is no active transition with a corresponding label. In the first case, the transition is fired, the marking of the model changes, and the algorithm moves to the next event of the log. In the second case, there is no sufficient number of tokens in some of the places needed to fire the corresponding transition of the model. An artificial token that allows one to continue the execution of the model is being added to this place. The place is marked as containing the corresponding number of missing tokens. During the consequent execution, if a similar situation arises again, the number of missing tokens in the considered position increases.

It is also possible that, upon reaching the final marking, the net still contains tokens that were not consumed during execution. The places where these tokens are located are then marked as holding remaining tokens.

In [17], the fitness of a model N and an event log L is defined as follows:

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

where m_σ is the number of missing tokens, r_σ is the number of remaining tokens, c_σ is the number of consumed tokens, p_σ is the number of produced tokens, and σ is a particular trace.

There are other, more sophisticated ways of evaluating the fitness of a model and an event log. The fitness calculation methodology based on usage of so-called *alignments* is suggested in [3–5]. Fitness tables for log traces and model executions are being built; then, the optimization problem is solved in order to choose the best alignments. Mismatches in alignments are assigned with certain costs, and the total score is calculated as their sum. The described method is complex and has high requirements to computational resources, but it provides a more accurate and realistic fitness assessment. A detailed description of the method is presented in [3–5]. Both methods of evaluation were used in this paper. The results presented in the paper were obtained mainly by using the second methodology. The algorithm based on this methodology provides a more accurate assessment thanks to using alignments [3].

The simplicity of the model can be determined in different ways. This metric shows how easy it is for an arbitrary expert to understand the model. For simplicity, we can consider a model that bears good fitness with a given event log and, at the same time, contains the minimum quantity of nodes (places and transitions), as well as of arcs that link them. In this case, a model containing a large quantity of nodes and arcs is considered to be complex. Specific values of metric corresponding to a simple or complex model are determined based on expert judgment and particular conditions of a problem being solved. In this paper, we have used an evaluation of the size of the model and entanglement by number of nodes and arcs as metrics of its simplicity.

Using simplicity and fitness as the only criteria does not always lead to models with the required characteristics. One can construct an ultimately simple model that allows almost any behavior, including actions registered in the provided event log. Precision and generalization criteria are meant to eliminate such cases. An precise (in relation to a certain log) model allows the behavior registered in a log, but no other behavior. Naturally, an imprecise model allows any behavior. Keeping the model precision balanced is an important issue for analyzing process models and construction. In this paper, we have used the precision-evaluation method suggested in [3, 6].

An excessively exact model is basically another format of an event log (it does not allow generalization) and, thus, is meaningless. The task of process mining is to build models capable of not only reproducing behavior that is already known, but also of making assumptions regarding behavior that is yet unknown, but possible. The criterion of a model's generalization shows the breadth of the allowed behavioral ranges. A model that only allows behavior written in an original event log is called *overfitting*; these models are not of much interest. A model that allows arbitrarily various behavior along with behavior registered in a log is called *underfitting*. The methodology of generalization calculations used in this paper was proposed in [3, 6].

Keeping a model in balance and development of balanced criteria of model quality are open and important problems in process mining. To evaluate the quality of the model, in this paper, we use the fitness, precision, and simplicity as metrics of the model. Construction of good generalization metrics is a problem not yet completely solved. Existing metrics have their disadvantages and require objective assessment. For example, the results of calculating generalization with the main existing methodology are almost always very high. Thus, one can say that this metric does not separate models into generalization classes well enough. More detailed information is available in [7].

4. CLASSIFICATION OF NOISE IN EVENT LOGS

When solving real-life problems using process mining approaches, a researcher has an event log that describes system behavior for a certain time interval as source data. In some cases, a process model synthesized from a log or manually can be available as well. As a rule, the essence of the problem is building a model of real system behavior and discovering bottlenecks. Sometimes, it is unknown whether the information system has bottlenecks or underperforming components in the first place. Building a model of actual behavior and analyzing it can allow one to determine if further system research is reasonable. Interferences in event logs can make obtaining correct models much more difficult.

This section provides the classification of different types of interferences and distortions that can appear in event logs of information systems. The main causes of interferences in event logs can be separated into three large categories, i.e., (1) technical and system failures, (2) incorrect interpretation of events, and (3) erroneous timestamps. The nature of interferences that arise in an event log is different for every group.

System failures lead to the following types of interference: missing events in a log (a transition in a model was fired, so the process operation was performed, but there was no record in an event log), extra events in a log (a failure results in an entry without an actual corresponding operation), and random distortions of an event log. Any event log is essentially text information formatted in a certain way stored in digital form in databases or in analog format (as prints and records). Obviously, a system failure can result in damage to entries, introduce random interference, compromise formatting, etc. A failure makes reading an event log partly impossible. In these cases, usually only part of a log gets damaged, and log storage format makes it possible to read it after filtering out the damaged parts.

As we already mentioned, this paper considers logs in XES format. This is an XML-based format, so it is possible to correct an XES log by filtering out the damaged traces. Correcting event logs is discussed in greater detail in [16]. Obviously, there is no sense in considering heavily damaged event logs. In this case, it is extremely difficult to distinguish correctly recorded system behavior from erroneous behavior because the faulty part can be much larger than the correct part.

Purely technical failures are not the only cause of problems. An incorrect interpretation of processes or executed operations results in a lot of problems in event logs. Here, *incorrect interpretation* primarily means an erroneous record of the event identifier or additional information (when, e.g., the log contains at some position an identifier of an incorrect event (action) that is also present in the process). These errors can be caused by various factors, some of which are improper configuration of information system software or hardware. Errors in software code of an information system can also cause false events to be registered in a log. Most of these errors arise if information about an executed action or event is entered by a human (information system user). To err is human. Even in mission-critical operations, the probability of erroneous data input cannot be neglected.

Entries with interference in timestamps are another type of incorrect entry in event logs. As we have already discussed, every log entry must contain a timestamp that binds an event or an action to a certain date and time. The event sequence is important for model building. A lot of model discovery algorithms take into consideration only similar dependences (in other algorithms, this information plays crucial role). Thus, errors of this type are among the most significant in terms of analyzing the performance of an information system based on event logs.

Interferences in timestamps can be either random or periodic. There is an easy example: people rarely recognize how long it took for them to perform an action (or the length of the time period). People tend to think in terms of "I did it for about 15 min," rather than "I performed this task in 13 min 29 s." The outcome of such rounding can differ; it depends on a person's culture, temper, and occupation. At dangerous plants or airports, time may be recorded to the nearest tenth, while in a hospital or government institution, an employee will most likely provide a rather approximate estimate. These periodic interferences can lead to rather strange results during event log analysis. Users of information systems frequently fill out report fields post factum, which also leads to estimates and periodic noises.

Event logs of real-life information systems can sport other typical interferences caused by incorrect timestamp entry. Sometimes, a correct day is entered, but the default month or the year remains unchanged. Time can be entered incorrectly in systems with a 12-h time format. Many similar errors can be eliminated by redesigning the user interface. For example, switching input fields to a 24-h format decreases the number of errors during time input.

Lastly, various desynchronizations of information-system nodes also lead to problems with timestamps. The clock on some computers or in network servers can be configured incorrectly. Software code errors can also result in incorrect timestamps.

5. STUDYING EVENT LOGS WITH NOISE

Results of researching the impact of different types of noise in event logs on change of properties of the model being synthesized on their basis are presented in this section.

Experimental setup. For testing purposes, we have used an event log generator designed in the PAIS laboratory [20, 21]. The *Gena* event log generator is implemented as a plug-in for the ProM 6.3 Framework environment [8, 13], which is intended for working with process mining algorithms. The ProM environment contains many extensions and supports custom extensions. The Gena generator is one of these extensions.

The approach to generating event logs implemented in the Gena program includes three important components, i.e., (1) the generation of a single log, (2) the generation of a set of event logs, and (3) the generation of additional distortions in an event log. From a technical point of view, the process of an event log generation can be described as follows.

1. Tokens are placed in all places listed in the initial marking set. These tokens contain special trace ID. Thus, the system can contain tokens that correspond to different traces (different process instances) at the same time.

2. Selecting enabled transitions that can be fired in the configured marking from all transitions.

3. Selecting a single transition (randomly or based on user-defined preferences) from the set of enabled ones. If there are arcs that lead to several transitions from one place (i.e., the branching of the control flow is possible), then the user of the plug-in can define the priority branch by configuring preferences for the corresponding transition.

4. The selected transition fires and the corresponding entry with the name and timestamp of the performed action are recorded in the event log. Gena can also record data about resources used by a process. However, this functionality is beyond the scope of this work. The generator supports two operating modes, i.e., direct timestamp recording during firing or recording two events that correspond to the beginning and end of an action. If the second mode is selected, then during transition firing, the program records the event of the beginning of the action, after which log generation continues. After a certain time period (set by a user for the specific action), it is considered to be finished. An entry of action ending and the timestamp are added to the log.

5. If the mode of generating a log with noise is selected, then during transition firing, additional work is performed in order to determine whether it is absolutely correct to fire this transition or if noise must be modeled. Based on user-defined settings and preferences, as well as on random numbers, Gena decides if it is necessary to skip transition firing (do not write information about activation in the log), to repeat the firing, to replace the name, or to distort the timestamp or resource name. By default, in noise mode, any transition firings can be distorted with equal probability. The probability of distortion during the firing of a particular transition is determined based on a user-specified overall threshold level (0–100).

6. The necessary number of tokens with corresponding trace IDs is added to all the places linked with the fired transition by arcs.

7. The corresponding number of tokens is removed from the input places of the fired transition.

8. The same steps are repeated until the user-defined final marking is achieved.

9. If the final marking was not achieved in the number of steps defined by the user (e.g., for the model shown in Fig. 1, a system error can create an infinite treatment loop), the trace is considered to have failed to complete successfully. The Gena program can either leave these traces in a log or remove them, leaving only the ones that were completed successfully. It is important to remember that, in the case of large models with loops, the number of steps necessary to complete the execution of the process can be very high, which affects the time of log generation.

10. New traces start either at a random point of time during generation or end in a situation when a final marking is achieved but new enabled transitions cannot be found. To start a new trace, the program adds the necessary number of tokens that contain the new trace ID to places that correspond to the initial marking.

The program user specifies the number of traces in the event log being generated, as well as the number of logs. After the generation of one log with the specified number of traces, the system automatically restarts and moves to the generation of the next log. So, several logs are generated sequentially. However, the timestamps of the generations can be set so that logs describe the behavior executed at the same time. This can be useful for the comparative testing of algorithms.

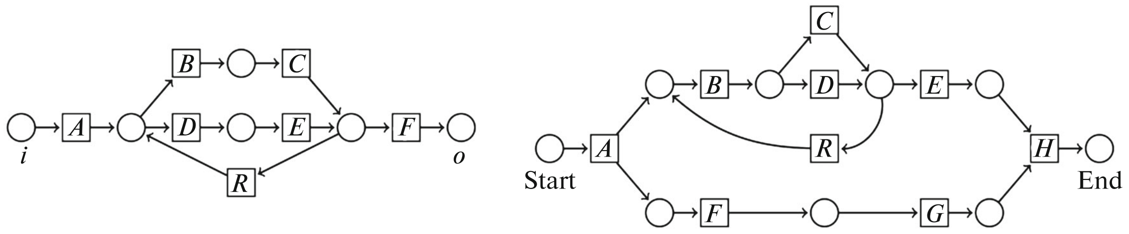

Fig. 2. Models T (7 transitions, 6 places) and S (9 transitions, 9 places).

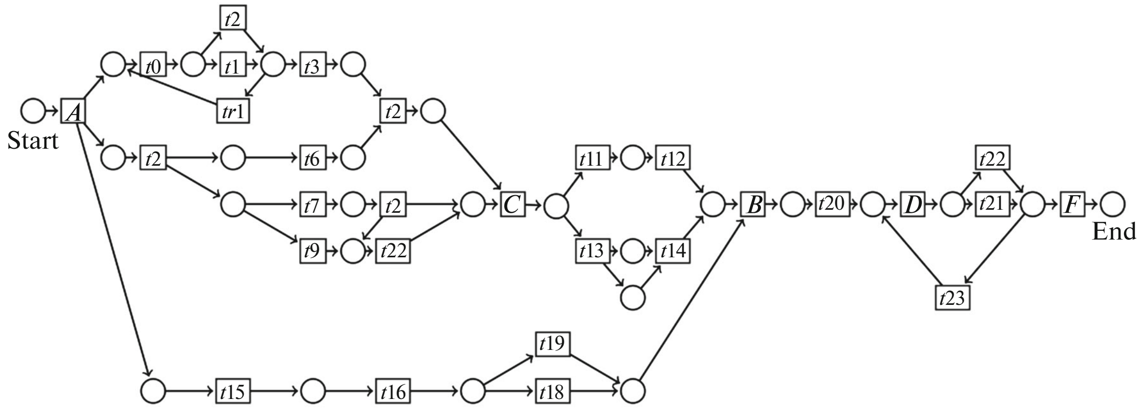

Fig. 3. Model M (29 transitions, 27 places).

DESCRIPTION OF THE EXPERIMENT

Three test models were used as source data: T (Fig. 2), S (Fig. 2), and M (Fig. 3).

The following experiment was conducted with each model:

1. The first step is the generation of an event log without noise that corresponds to the perfectly correct behavior of an information system. For simplicity, during the experiment, the lengths of process actions were configured in order to make one execution of a process to take approximately 24 h. Default settings were used for generation, and the process did not use model resources or additional characteristics. For this initial event log, the characteristics of its fitness with a model were measured. Later, these characteristics were used as a reference and were compared to the fitness characteristics of models and event logs obtained by generation with the addition of artificial noise.

2. The Gena program was used to generate event logs with additional noise of two levels (with 3 and 10% probabilities of noise on transition firing). The following types of interferences were used:

(1) random permutation of event names (during transition firing, the name of another transition, randomly chosen from the same model, was recorded in the event log);

(2) the recording of one of three special randomly selected events (event names: *No1*, *No2*, *No3*);

(3) the interference-type combination of (1) and (2), i.e., the name permutation within a model and addition of new events;

(4) interferences at timestamp recording, i.e., the random time shift for the event and periodic noise with time rounding up to 5 and 15 min (this simulates system clock with a rough scale or rounding by staff).

Thus a set of test event logs that correspond to interference of different natures was generated for every test model.

3. We used four ProM plug-ins to build four models from every event log, i.e., (1) Alpha Miner, (2) ILP Miner, (3) Heuristics Miner, and (4) Inductive Miner. Each of these plug-ins is an implementation of a basic process model discovery algorithm with certain specifics of application.

Alpha Miner implements the alpha algorithm, which is actually the first and simplest algorithm of process-model discovery [1, 2]. We do not provide a detailed description of model discovery algorithms in this paper because this description would be too large. It is important to note that the alpha algorithm is not sufficiently effective, as well as it does not possess mechanisms to process event logs with noise. The *Alpha Miner* implements rather simple algorithm and thus does not provide any settings. It should be noted that the alpha algorithms family is mostly adapted to maximize the criterion for model simplicity [7]. A more detailed description of the algorithm is provided in [2].

An algorithm that uses linear programming in model discovery is more complex. Without further elaboration, we will note that, in many algorithms, the Petri net is synthesized from an event log with an intermediate step, i.e., the transition system. The solution of the linear programming problem is meant to minimize the number of states in this system. The *ILP Miner* plug-in implements this algorithm [22]. This plug-in provides some settings related to solver selection and performance limitations. Default settings were used in this experiment. We should note that this plug-in runs very slowly on large models (logs containing a lot of different event classes). In some cases, *ILP Miner* failed to discover the model in this experiment. The algorithm is described in greater detail in [22].

Heuristics Miner implements a heuristic algorithm of model discovery [23]. A distinctive feature of this algorithm takes into account the frequency characteristics of events in a log. The heuristic model discovered with this algorithm contains information about what model nodes and arcs are used more frequently during execution of the given event log. This makes it possible to separate the main and secondary parts of the model. The heuristic algorithm copes with noise and is able to produce a model that reflects only mainstream behavior from that written in the event log. The algorithm discards details and rare features of process instantiation. The algorithm and the heuristic model properties are described in greater detail in [23]. It is important to note the *Heuristics Miner* plug-in generates a heuristic model, unlike the other plug-ins used in this paper. Fortunately, the model discovery package also contains a plug-in transforming a heuristic model into a Petri net. Default plug-in settings were used in this experiment.

An algorithm of inductive model discovery implemented in the *Inductive Miner* plug-in [11] is the most modern algorithm. This algorithm also analyzes an event log in terms of probabilities, so it can cope with noise. Its main advantage is the fact that it always generates correct and executable models. Other algorithms do not guarantee this result. Moreover, as described in [11], the algorithm contains special tools for working with incomplete and corrupt event logs. Default plug-in settings were used in this experiment.

4. The models discovered with different plug-ins for every model log were evaluated according to fitness criteria.

RESULTS OF THE EXPERIMENT

For source models shown in Fig. 2 (T, 7 transitions, 6 places, 14 arcs), Fig. 2 (S, 9 transitions, 9 places, 20 arcs), Fig. 3 (M, 29 transitions, 27 places, 67 arcs), and event logs without noise that correspond to perfectly fitting behavior were generated, as well as logs with noise.

Table 2 contains a list of experimental event logs created in the experiment, as well as settings of their generation. The number of events in logs is different; it varies from 176 events in the log without noise for the smallest model to 561 for a large model. We deliberately avoided using enormous logs in this paper. Our purpose was not to evaluate the performance of algorithms, but testing their capabilities to process logs with noise.

For every event log in Table 2, we built models by using four model discovery algorithms, i.e., Alpha, Heuristics, ILP, and Inductive. It is impossible to present images of all models in this paper due to volume limitations. The results of checking fitness (in different criteria) between the original event log without noise and the models built during the experiment are presented in Table 3. Methods of measuring fitness, precision, and generalization are described in [3–6]. Numbers of model nodes and arcs were used as the metric of model complexity.

The Heuristics and ILP algorithms produce fairly complex models, which adds a relatively large quantity of additional elements (e.g., for a log without noise the heuristic algorithm increases the model size twice). Moreover, these algorithms do not guarantee the executability of the resulting model. ILP and Inductive algorithms guarantee the perfect fitness of resulting model. Although this negatively impacts the model precision, generalization, and size.

In some cases, model characteristics were not calculated within a reasonable time. This often happens, e.g., with models built using the ILP algorithm; their structure turns out to be highly complex. As we have discussed, the Alpha algorithm is completely unable to cope with noise in a model, which is confirmed by data even for the simplest models. For logs eT3-2—eT3-4, eS3-2—eS3-4, and eM3-2—eM3-4, this algorithm produces more or less meaningless models with extremely low fitness.

Table 2. Generated event logs

Model	Log	Noise, %	Generation settings
T	eT0	0	20 traces, pauses: 900–1200 s, operation execution: 6200–8200 s
T	eT3-1	3	Artificial events
T	eT3-2	3	A, C, R activities order permutation
T	eT3-3	3	Activity permutation + artificial events
T	eT3-4	3	Noise in timestamps with max deviation of 14400 s
T	eT5-1	10	Artificial events
T	eT5-2	10	A, C, R activities order permutation
T	eT5-3	10	Activity permutation + artificial events
T	eT5-4	10	Noise in timestamps with max deviation of 14400 s
S	eS0	0	20 traces, pauses: 900-1200 s, operation execution: 6200–8200 s
S	eS3-1–eS3-4	3	Same as for logs eT3-1–eT3-4
S	eS5-1–eS5-4	10	Same as for logs eT5-1–eT5-4
M	eM0	0	20 traces, pauses: 900-1200 s, operation execution: 2600–4600 s
M	eM3-1–eM3-4	3	Same as for logs eT3-1–eT3-4
M	eM5-1–eM5-4	10	Same as for logs eT5-1–eT5-4

Table 3. Fitness of the original event log for model T to models discovered from logs with different characteristics

Log	Algorithm	Fitness	Precision	Generalization	Nodes	Arcs
eT0	Alpha	1.0	0.86	0.99	13	14
eT0	Heuristics	0.83	0.82	0.99	29	30
eT0	ILP	0.89	0.82	0.99	12	13
eT0	Inductive	1.0	0.86	0.99	15	16
eT3-1	Alpha	0.31	1.0	0.96	23	35
eT3-1	Heuristics	0.46	0.34	0.87	>40	>40
eT3-1	ILP	1.0	–	–	>50	>50
eT3-1	Inductive	1.0	0.77	0.99	20	22
eT3-2	Alpha	0.11	0.15	0.0	>20	>20
eT3-2	Heuristics	0.69	0.88	0.95	20	20
eT3-2	ILP	1.0	0.60	0.86	11	23
eT3-2	Inductive	1.0	0.19	0.99	22	28
eT3-3	Alpha	0	0	0	>30	>30
eT3-3	Heuristics	0.1	–	–	30	30
eT3-3	ILP	0.97	0.43	0.97	15	>30
eT3-3	Inductive	1.0	0.86	0.99	15	16
eT3-4	Alpha	1.0	0.86	0.99	13	14
eT3-4	Heuristics	0.69	0.88	0.95	20	20
eT3-4	ILP	1.0	0.86	0.99	12	13
eT3-4	Inductive	1.0	0.86	0.99	15	16

The Heuristics algorithm produces good results for noisy logs, regardless of the model size. The inductive algorithm copes with noise and retains the fitness between the model and the log at about 1; generalization also remains at a fairly high level. The weak spot of this algorithm is the reduction of precision with an increase in the amount of noise in a model.

The Alpha algorithm ignores timestamps and takes into account only the sequence of log entries, which is its strong point. Thanks to this feature, noise in timestamps does not affect its functioning at all.

Table 4. Fitness of the source model to event logs for model M

Log	Fitness	Precision	Generalization
eM0	1.0	0.62	0.85
eM3-1	0.92	0.61	0.83
eM3-2	0.93	0.60	0.78
eM3-3	0.92	0.61	0.83
eM3-4	0.85	0.58	0.78
eM5-1	0.82	0.61	0.84
eM5-2	0.79	0.61	0.83
eM5-3	0.84	0.60	0.78
eM5-4	0.77	0.61	0.89

The results also show that transposition of activities is easier for the algorithms to deal with than new activities in a log.

This paper is limited in size and, thus, contains only a fraction of experimental data. The patterns revealed in the data presented can also be observed for the rest of the models.

Table 4 contains results of fitness check (based on various criteria) between the source model and different event logs with and without noise. The table demonstrates how noise in an event log affects its fitness with the model. It is curious that change in the event sequence at firing influences precision the most, whereas noise in timestamps affects the fitness. For the eM5-4 log, the generalization of the model increased after adding noise to the log. This effect is quite expected. Behavior with noise recorded in an event log differs significantly from the model behavior. At the same time, the model allows much more varied behavior compared to specific executions of a process contained in an event log.

6. CONCLUSIONS

The conducted experiments show that various types of noise change the resulting discovered model. The type of approach applied makes a difference, but noise influences any applied algorithm. The nature of the influence depends on the type of noise. In the presence of noise in the event-log timestamps, the resulting models can differ from those obtained by using noiseless logs. Many algorithms cope with permutations of events in a log (without corrupting timestamps).

For large logs that contain every version of the execution of the process in several copies, the removal of incorrect traces does not influence the final model in general. However, if the event log represents a very small fraction of the process behavior, mining algorithms become unstable. A small change in the log leads to significant changes in the final model.

Developers of the information system must pay close attention to the correctness of writing data about process functioning, especially regarding timestamps, if further tracking of the system functioning or process parameters improvement is assumed. Many algorithms capable of handling logs with noise were designed in the area of process mining (Heuristics, Genetic, Flexible, Inductive). Nevertheless, processing event logs with noise is often reduced to simple filtering with the removal of traces that contain noise or ignoring events with incorrect entries [1]. As a rule, algorithms that successfully cope with noise do not produce models in the form of classical Petri nets (see, e.g., [23]). Transformation into a Petri net is possible for almost all types of models, but it increases the number of potential errors and makes the model much more complex.

Some kinds of noise (e.g., noise in timestamps) are very cunning and successfully resist processing attempts. An algorithm produces a model that seems to be correct but is totally different from the model for the same log without noise. This means two things. First, researchers must be careful in applying process mining methodologies to event logs containing noise. Second, the development of more perfect algorithms for discovering the process model from logs with noise is necessary. In particular, there is good potential in developing algorithms that take into consideration additional information related to events during model discovery, including data, resources, executor roles, etc. These algorithms will be more tolerant to interferences in timestamps.

ACKNOWLEDGMENTS

This work is supported by the Basic Research Program at the National Research University Higher School of Economics (2014).

REFERENCES

1. Van der Aalst, W.M.P., *Process Mining: Discovery, Conformance and Enhancement of Business Processes,* Springer, 2011.
2. Van der Aalst, W.M.P., Weijters, A.J.M.M., and Maruster, L., Workflow mining: Discovering process models from event logs, *IEEE Trans. Knowl. Data Eng.,* 2004, vol. 16, no. 9, pp. 1128–1142.
3. Van der Aalst, W.M.P., Adriansyah, A., and Van Dongen, B.F., Replaying history on process models for conformance checking and performance analysis, *Wiley Interdiscip. Rev.: Data Mining Knowl. Discovery,* 2012, vol. 2, no. 2, pp. 182–192.
4. Adriansyah, A., Van Dongen, B.F., and Van der Aalst, W.M.P., Conformance checking using cost-based fitness analysis, *15th IEEE International Conference on Enterprise Distributed Object Computing Conference (EDOC),* 2011, pp. 55–64.
5. Adriansyah, A., Van Dongen, B.F., and Van der Aalst, W.M.P., Towards robust conformance checking, in *Business Process Management Workshops,* Springer, 2011, pp. 122–133.
6. Adriansyah, A., Munoz-Gama, J., Carmona, J., Van Dongen, B.F., and Van der Aalst, W.M.P., Alignment based precision checking, in *Business Process Management Workshops,* Springer, 2012, pp. 137–149.
7. Buijs, J.C.A.M., Van Dongen, B.F., and Van der Aalst, W.M.P., On the role of fitness, precision, generalization and simplicity in process discovery, *20th International Conference on Cooperative Information Systems (CoopIS 2012),* 2012.
8. Van Dongen, B.F., Van der Aalst, W.M.P., Günther, C.W., Rozinat, A., Verbeek, E., and Weijters, T., ProM: The process mining toolkit, in *Business Process Management Demonstration Track (BPMDemos2009),* Medeiros, A.K.A.d. and Weber, B., Eds., 2009, vol. 489, pp. 1–4.
9. Kalenkova, A.A. and Lomazova, I.A., Discovery of cancellation regions within process mining techniques, *Proceedings of the 22nd International Workshop on Concurrency, Specification and Programming,* Warsaw, 2013, pp. 232–244.
10. Kalenkova, A.A., Lomazova, I.A., and Van der Aalst, W.M.P., Process model discovery: A method based on transition system decomposition, *Lect. Notes Comput. Sci.,* 2014, vol. 8489, pp. 71–90.
11. Leemans, S.J.J., Fahland, D., and Van der Aalst, W.M.P., Discovering block-structured process models from incomplete event logs, in *Tech. Rep. BPM-14-05,* Eindhoven University of Technology, 2014.
12. Munoz-Gama, J., Carmona, J., and Van der Aalst, W.M.P., Conformance checking in the large: Partitioning and topology, *Lect. Notes Comput. Sci.,* 2013, vol. 8094, pp. 130–145.
13. Verbeek, H.M.W., Buijs, J.C.A.M., Van Dongen, B.F., and Van der Aalst, W.M.P., Prom 6: The process mining toolkit, in *Proceedings of BPM Demonstration Track,* 2010, vol. 615, pp. 34–39.
14. Verbeek, H.M.W., Buijs, J.C.A.M., Van Dongen, B.F., and Van der Aalst, W.M.P., XES, XESame, and ProM 6, *Lect. Notes Bus. Inf. Process.,* 2011, vol. 72, pp. 60–75. doi 10.1007/978-3-642-17722-4_5
15. http://www.xes-standard.org/xesstandarddefinition.
16. Rogge-Solti, A., Mans, R.S., Van der Aalst, W.M.P., and Weske, M., Repairing event logs using timed process models, *Lect. Notes Comput. Sci.,* 2013, vol. 8186, pp. 705–708.
17. Rozinat, A., Process mining: Conformance and extension, *PhD Thesis,* Eindhoven University of Technology, 2010.
18. Rubin, V.A., Lomazova, I.A., and Van der Aalst, W.M.P., Agile development with software process mining, *Proceedings of the 2014 International Conference on Software and System Process (ICSSP 2014),* Nanjing, 2014, pp. 70–74.
19. Rubin, V.A., Mitsyuk, A.A., Lomazova, I.A., and Van der Aalst, W.M.P., Process mining can be applied to software too!, *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2014),* Torino, 2014, pp. 57:1–57:8.
20. Shugurov, I. and Mitsyuk, A.A., Generation of a set of event logs with noise, *Proceedings of the 8th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2014),* 2014, pp. 88–95.
21. http://pais.hse.ru/research/projects/gena.
22. Van der Werf, J.M.E.M., et al., Process discovery using integer linear programming, in *Applications and Theory of Petri Nets,* Springer Berlin Heidelberg, 2008, pp. 368–387.
23. Weijters, A., Van der Aalst, W.M.P., and De Medeiros, A.K.A., Process mining with the heuristics miner-algorithm, *Tech. Univ. Eindhoven, Tech. Rep. WP,* 2006, vol. 166, pp. 1–34.

Translated by I. Kashukov