Discovering process models from empirical data

Laura Măruşter (l.maruster@tm.tue.nl), Ton Weijters (a.j.m.m.weijters@tm.tue.nl) and Wil van der Aalst (w.m.p.aalst@tm.tue.nl)
Eindhoven University of Technology, PO Box 513, 5600 MB Eindhoven, The Netherlands

Antal van den Bosch (antal.vdn.bosch@uvt.nl) and Walter Daelemans (walter.daelemans@uvt.nl)
Tilburg University, Tilburg 5000 LE, The Netherlands

Abstract. Effective information systems require the existence of explicit process models; a completely specified process design needs to be developed in order to enact a given business process. This development is time consuming and often subjective and incomplete. We propose a method that discovers the process model from process logs where process events are recorded as they have been executed over time. We induce a rule-set that predict causal, exclusive, and parallel relations between process events. The rule-set is induced from simulated process log data that are generated by varying process characteristics (e.g. noise, log size). Tests reveal that the induced rule-set has a high performance on new data. Knowing the causal, exclusive and parallel relations we can build the process model expressed in the Petri net formalism. We also evaluate the results using a real-world case study.

Keywords: rule induction, process models, Petri nets

1. Introduction

The managing of complex business processes calls for the development of powerful information systems, able to control and support the underlying process. In order to support a structured business process, an information system has to offer generic modelling and enactment capabilities. Workflow management systems (WfMS) are good candidates for this (Aalst, 1998). However, many problems are encountered when designing and employing such information systems. One of the problems is that these systems presuppose the existence of the process design, i.e. a designer has to construct a detailed model accurately describing the whole process. The drawback of such an approach is that the process model requires considerable effort from the process designers, workers and management, is time consuming and often subjective and incomplete.

As an alternative to hand-designing the process, we propose to collect the sequence of events produced over time by that process, and discover the underlying process model from these sequences. We assume

Table I. A process log example

Case number	Executed tasks
Case 1	a f g h i k l
Case 2	a b c e j l
Case 3	a f h g i k l
Case 4	a f g i h k l
Case 5	a b c e j l
Case 6	a b d j l
Case 7	a b c e j l

that it is possible to record events such that (i) each event refers to a task, (ii) each event refers to a case (i.e. process instance) and (iii) events are totally ordered. We call a set of such recorded sequences the process log. We call the method of distilling a structured process description from a process log *process discovery* (sometimes referred as workflow or process mining (Aalst, 2002a)).

To illustrate the idea of process discovery, consider the process log from Table I. In this example, there are seven cases that have been processed; twelve different tasks occur in these cases. We can notice the following: for each case, the execution starts with task a and ends with task l, if c is executed, then e is executed. Also, sometimes we see task h and i after g and h before g.

Using the information shown in Table I, we can discover the process model shown in Figure 1. We represented the model using workflow nets (Aalst, 1998), where all tasks are expressed as transitions. Workflow nets are a special variants of Petri nets (Reisig, 1998). After executing a, either task b or task f can be executed. If task f is executed, tasks h and g can be executed in parallel. A parallel execution of tasks h and g means that they can appear in any order.

In this simple example, the construction of the Petri net was straightforward. However, in the case of real-world processes where much more tasks are involved and with a high level of parallelism, the problem of discovering the underlying process becomes very complex. Moreover, the existence of noise into the log complicates the problem even more.

The idea of discovering models from process logs was previously investigated in contexts such as software engineering processes and workflow management (Agrawal, 1998), (Cook, 1998a), (Herbst, 2000), etc. Cook and Wolf propose three methods for process discovery in the case of software engineering processes: a finite-state-machine method, a neural network and a Markov approach (Cook, 1998a). Their methods

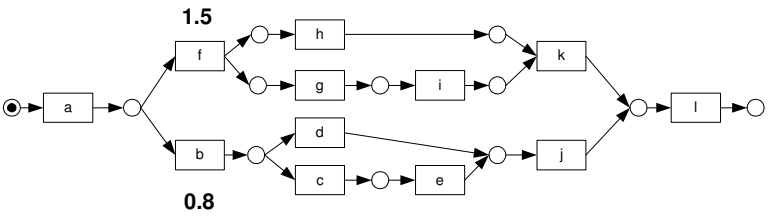


Figure 1. A process model for the process log shown in Table I

focus on sequential processes. Also, they have provided some specific metrics for detection of concurrent processes, like entropy, event type counts, periodicity and causality (Cook, 1998b). Herbst and Karagiannis used a hidden Markov model in the context of workflow management, in the case of sequential processes (Herbst, 2000a; Herbst, 1998; Herbst, 1999; Herbst, 2000) and concurrent processes (Herbst, 2000b). In the works mentioned, the focus was on identifying the dependency relations between events. In (Maruster, 2001), a technique for discovering the underlying process from hospital data is presented, under the assumption that the workflow log does not contain any noisy data. A heuristic method that can handle noise is presented in (Weijters, 2001); however, in some situations, the used metric is not robust enough for discovering the complete process. Theoretical results are presented in (Aalst, 2002a), being proven that for certain subclasses it is possible to find the right process model. In (Aalst, 2002b) the method used in (Aalst, 2002a) is extended to incorporate timing information.

In this paper, the problem of process discovery from process logs is defined as a two-step method: (i) find the causal relations (i.e., for each task, find its direct successor tasks) and (ii) find the parallel/exclusive relations (i.e. for tasks that share the same cause or the same direct successor, detect if they can be executed in parallel or there is a choice between them). We aim to use an experimental approach for inducing the rule-sets required in the two-step method. It is assumed that the process log contains noisy information and there is imbalance of execution priorities. Knowing the relations between tasks, a Petri net model can be constructed.

The experimental approach used to discover a process was previously introduced in (Maruster, 2002). In the work mentioned, a logistic regression model has been developed to detect the direct successors from a process logs, in the presence of noise and imbalance of task execution probabilities. The method from (Maruster, 2002) requires a global threshold value for deciding when there is a direct succession relation between two tasks. The use of a global threshold has the draw-

back of being too rigid, thus real relations may not be found and false relations may be considered in data of which the general characteristics deviate from the data on which the threshold was based. Therefore, in this paper we aim to develop a more accurate and flexible model which does not need a threshold.

In practical situations it seems realistic to assume that process logs contain noise. Noise can have different causes, such as missing registration data or input errors. Moreover, the log can be incomplete. We say that a process log is *complete* if all tasks that potentially directly follow each other, in fact will directly follow each other in some trace in the log. In case of a complex process, the process log will contain not enough information to detect the causal relation between two tasks. The notion of completeness is formally defined in (Aalst, 2002a). Another source of problems is the existence of imbalances between the task execution priorities. In Figure 1, after task a is executed, either task b or f can be executed. A task execution imbalance means that if task b has an execution priority of 0.8 and task f 1.5, task f is more likely to be executed after task a.

Given the fact that in practical situations the process logs are incomplete, contain noise and can exist imbalances between the task execution priorities, the discovery problem becomes more problematic.

The content of this paper is organized as follows: in Section 2 the types of relations that can exist between two tasks are presented. The methodology of generating experimental data that serves to induce the rule-sets is presented in Section 3. In Section 4 the metrics used to induce the rule-set are introduced. Section 5 contains the description of the rule-sets that detects the relations between tasks. The issue of performance in case of different test experiments is presented in Section 6. In Section 7 we discuss the results obtained and the influence of process characteristics on rule-set model performance. A case-study is presented in Section 8. We end with discussing issues for further research.

2. The log-based relations

Our method of discovering the process model from a log file is based on finding the relations that can exist between tasks. For example, if a task is always followed by another task, it is likely that there is a causal relation between both tasks. In order to find the ordering relations between tasks, we use the dependency/frequency table.

2.1. The dependency/frequency table

The construction of a so-called dependency/frequency (D/F) table from the process log information is the starting point of our method and was first used in (Weijters, 2001). An excerpt from the D/F table for the Petri net presented in Figure 1 is shown in Table II. For each pair of tasks x and y, the following information is abstracted out of the process log: (i) the identifiers for tasks x and y, (ii) the overall frequency of task x (notation $|X|$ [1]), (iii) the overall frequency of task y $|Y|$, (iv) the frequency of task x directly preceded by another task y $|Y > X|$, (v) the frequency of task x directly succeeded by another task y $|X > Y|$, (vi) the frequency of x directly or indirectly preceded by another task y, but before the next appearance of x $|Y >>> X|$, (vii) the frequency of x directly or indirectly succeeded by another task y, but before the next appearance of x $|X >>> Y|$. The frequencies (ii)-(vii) are used to find the log-based ordering relations, which are presented in Section 4.

Table II. An excerpt from the D/F table for the Petri net presented in Figure 1.

x	y	$\|X\|$	$\|Y\|$	$\|Y > X\|$	$\|X > Y\|$	$\|Y >>> X\|$	$\|X >>> Y\|$
a	f	1800	850	0	850	0	850
f	g	850	850	0	438	0	850
c	d	446	504	0	0	0	0
g	h	850	850	412	226	412	438
b	f	950	850	0	0	0	0
i	h	850	850	226	212	638	212

2.2. The log-based relations

Discovering a model from process logs involves determining the dependencies among tasks. Four types of event dependencies have been introduced in (Cook, 1998b): direct, sequential, conditional and concurrent dependence. In (Aalst, 2002a) and (Maruster, 2001), the notions of log-based relations are formally introduced in the context of process logs and process traces. In the last two references, the terms workflow log and workflow trace were used (Aalst, 2002a) and (Maruster, 2001).

[1] We use a capital letter when referring to the number of occurrences of some task.

In this paper we refer to process log and process trace, because the results are not limited to the workflow domain.

DEFINITION 1. *Process trace, process log*

Let T be a set of tasks. $\delta \in T^*$ is a process trace and $W \in \mathcal{P}(T^*)$ is a process log.

An example of a process log is given in Table I. A process trace for case 1 is "*afghikl*". Looking to the process log presented in Table I we can notice that the traces for cases 1, 3, 4 and 6 appear in the log just once, while the trace "*abcejl*" appears three times, e.g. for cases 2, 5 and 7. In (Aalst, 2002a) and (Maruster, 2001) the log is considered to be free of noise and in that case, the frequencies of specific traces were not used. In the current paper we assume that the log may contain noise. In such a situation, using the trace frequencies is crucial for process discovery from process logs.

DEFINITION 2. *Succession relation*

Let W be a process log over T, i.e. $W \in \mathcal{P}(T^*)$. Let a, $b \in T$. Then between a and b there is a *succession* relation (notation $a > b$), i.e. b succeeds a if and only if there is a trace $\delta = t_1 t_2 ... t_n$ and $i \in \{1, ..., n-1\}$ such that $\delta \in W$ and $t_i = a$ and $t_{i+1} = b$.

The succession relation $>$ describes which tasks appeared in sequence, i.e. one directly following the other. In the log from Table I, $a > f$, $f > g$, $b > c$, $h > g$, $g > h$, etc.

DEFINITION 3. *Causal, exclusive and parallel relations*

Let W be a process log over T, i.e. $W \in \mathcal{P}(T^*)$ and $a, b \in T$. If we assume that there is no noise in W, then between x and y there is:

1. a *causal* relation (notation $x \rightarrow y$), i.e. x *causes* y if and only if $x > y$ and $y \not> x$. We consider the inverse of the causal relation $\rightarrow^{-1}$, i.e. $\rightarrow^{-1} = \{(y, x) \in T \times T \mid x \rightarrow y\}$. We call task x the *cause* of task y and task y is the *direct successor* of task x.
2. an *exclusive* relation (notation $x \# y$) if and only if $x \not> y$ and $y \not> x$;
3. a *parallel* relation (notation $x \parallel y$) if $x > y$ and $y > x$.

The relations $\rightarrow, \rightarrow^{-1}, \#$ and $\parallel$ are mutually exclusive and partition $T \times T$ (Aalst, 2002a).

To illustrate the above definitions, let's consider again the process log from Table I corresponding to the Petri net from Figure 1. If there is no noise, there are three possible situations in which a pair of events can be:

1. events c and e are in sequence: then $c > e$, $e \not> c$, thus $c \rightarrow e$;
2. there is a choice between events b and f: then $b \not> f, f \not> b$, thus $b \# f$ (and $f \# b$);
3. events h and i are in parallel: then $h > i$, $i > h$, thus $h \parallel i$ (and $i \parallel h$).

However, in case of noise, the notions presented in Definition 3 are not useful anymore. If we want to investigate the relation between c and e, we find that $c > e$. However, because of some noisy sequences, we may see also that $e > c$. Applying Definition 3, we could conclude that events c and e are parallel, which is wrong, because they are actually in a causal relation. Similarly, looking at events b and f, it can happen that $b > f$ and $f > b$, because of noise. Investigating the relation between h and i, we can see that $h > i$ and $i > h$, in situations with and without noise.

Suppose now that we are aware of the existence of noise in a process log (which is a realistic assumption) and for two generic tasks x and y we have $x > y$ and $y > x$. What is the relation between x and y: causal, exclusive or parallel?

In the rest of our paper we plan to induce decision rules that can be used to detect the relations between events, from noisy process logs. Next, we use the α algorithm (Aalst, 2002a) that constructs a process model using the Petri net formalism. The α algorithm considers first all tasks that stand in a causal relation. Then for all tasks that share locally the same input (or output) task, the exclusive/parallel relations are included to build the Petri net. Based on the choice for the α algorithm to build the Petri net, we plan to develop a method that adopts its sequence of actions: first detect the causal relations, and then determine the exclusive/parallel relations for all tasks that share the same local input (or output) task. In other words, our method consists on two distinct steps where the second depends on the output of the first. First, we induce a rule-set that detects the causal relations between two tasks. In the second step, we focus only on the instances (pairs of tasks) that share the same cause or the same direct successor and we induce a second rule-set for detecting the exclusive/parallel relations.

In order to induce such decision rule-sets, we have to generate appropriate learning material. When all relations between tasks become

known, the process model represented as a Petri net model is build using the α algorithm.

In Section 3 the generation of the experimental learning material is described. The induction and the evaluation of the decision rule-sets are presented in Section 5 and Section 6, respectively. An example of constructing the Petri net model is discussed in Section 7.

3. Experimental setting and data generation

The learning material that we use to induce the rule-sets should resemble realistic process logs. Of the possible elements that vary from process to process and subsequently affect the process log, we identified four: (i) the total number of events types, (ii) the amount of available information in the process log, (iii) the amount of noise and (iv) the execution priorities in OR-splits and AND-splits.

Our experimental setting consists of variations of four process log characteristics:

1. The number of task types: we generate Petri nets with 12, 22, 32 and 42 event types.

2. The amount of information in the process log or log size: the amount of information is expressed by varying the number of lines (one line or trace represents the processing of one case). We consider logs with 200, 400, 600, 800 and 1000 lines.

3. The amount of noise: we generate noise performing four different operations, (i) delete the head of a event sequence, (ii) delete the tail of a sequence, (iii) delete a part of the body and (iv) interchange two randomly chosen events. During the noise generation process, minimally one event and maximally one third of the sequence is deleted. We generate five levels of noise: 0% noise (the common workflow log), 5% noise, 10%, 20% and 50% (we select 5%, 10%, 20% and respectively 50% of the original event sequences and we apply one of the four above described noise generation operations).

4. The imbalance of execution priorities: we assume that tasks can be executed with priorities between 0 and 2. In Figure 1, after executing the event a (which is an OR-split), it is possible to exist an imbalance between executing task b and task f. For example, task b can have an execution priority of 0.8 and task f 1.5. This implies that after a, in 35 percent of the cases task b is selected, and in 65 percent of the cases, task f is executed.

The execution imbalance is produced on four levels:

- level 0, no imbalance: all tasks have the execution priority 1;
- level 1, small imbalance: each task can be executed with a priority randomly chosen between 0.9 and 1.1;
- level 2, medium imbalance: each task can be executed with a priority randomly chosen between 0.5 and 1.5;
- level 3, high imbalance: each task can be executed with a priority randomly chosen between 0.1 and 1.9.

First, we design four types of Petri nets: with 12, 22, 32 and 42 event types. Second, for each type of Petri net, we produce four unbalanced Petri nets, corresponding to the four levels of execution imbalance. Third, for each resulting Petri net, we generate a log file with 0%, 5%, 10%, 20% and 50% noise. Fourth, we vary the amount of information, i.e. we vary the number of lines in the log: each resulting noisy log is partitioned, considering the first 20% lines, then the first 40%, and so on, until 100% of material is considered. For each of the 400 resulted log files a D/F table is built and finally all the 400 D/F tables are combined into one big file used to induce the rule-sets for detecting the relations between tasks. In the next section we see how the information contained in the D/F table is used to detect the log-based relations.

4. The relational metrics

The information contained in the D/F table is the basic material that we use to induce the rule-sets for detecting the log-based relations. However, the raw frequencies of the D/F table cannot be used directly as input features for inducing the rule-set. Rather, we have to develop useful relational metrics from these raw data that can be used as input features.

When thinking about good measures that can be used to detect the causal relation $x \rightarrow y$ between tasks x and y, we noticed that the frequencies $|X > Y|$ and $|Y > X|$ from the D/F table are important to predict the causal relation. Namely, when the difference between $|X > Y|$ and $|Y > X|$ is large enough, there is a high probability that x causes y. We develop three different measures that use the difference between $|X > Y|$ and $|Y > X|$: the causality metric CM, the local metric LM and the global metric GM.

$|X > Y|$ and $|Y > X|$ frequencies are also useful to detect exclusive/parallel relations. If both frequencies are zero or very small

numbers, then it is likely that x and y are in an exclusive relation, while if they are both sufficiently high, then it is likely that x and y are in parallel relation. Therefore, we construct the metrics YX and XY which are obtained by dividing the frequencies $|X > Y|$ and $|Y > X|$ with the minimum of $|X|$ and $|Y|$.

The causality metric CM

The causality metric CM was first introduced in (Weijters, 2001). If for a given workflow log it is true that when task x occurs, shortly later task y also occurs, it is possible that task x causes the occurrence of task y. The CM metric is computed as follows: if task y occurs after task x and n is the number of events between x and y, then CM is incremented with a factor $(\delta)^n$, where δ is a causality factor, $\delta \in [0.0, 1.0]$. We set $\delta = 0.8$. The contribution to CM is maximally 1, if task y appears right after task x and consequently $n = 0$. Conversely, if task x occurs after task y and again the number of events between x and y is n, CM is decreased with $(\delta)^n$. After processing the whole log, CM is divided with the minimum of the overall frequency of x and y.

The local metric LM

Considering tasks x and y, the local metric LM is expressing the tendency of the succession relation $x > y$ by comparing the magnitude of $|X > Y|$ versus $|Y > X|$.

The formula for the local metric LM is:

$$LM = P - 1.96\sqrt{\frac{P(1-P)}{N+1}}, P = \frac{|X > Y|}{N+1}, N = |X > Y| + |Y > X| \tag{1}$$

The idea of this measure is borrowed from statistics and it is used to calculate the confidence intervals for errors. For more details, see (Mitchell, 1995). In our case, we are interested to know with a probability of 95% the likelihood of causality relation, by comparing the magnitude of $|X > Y|$ versus $|Y > X|$. For example, if $|A > B| = 30$, $|B > A| = 1$ and $|A > C| = 60$, $|C > A| = 2$, what is the most likely: a causes b or a causes c? Although both ratios $\frac{|A>B|}{|B>A|}$ and $\frac{|A>C|}{|C>A|}$ equal 30, a is more likely to cause c than b. Our LM measure for tasks a and b gives a value of $LM = 0.85$ and for tasks a and c gives a value of $LM = 0.90$, which is in line with our intuition.

Let's now consider again the Petri net from Figure 1. If we suppose that the number of lines in the log corresponding to this Petri net is equal to 1000 (i.e. $\#L$=1000), we can have the following three situations:

1. $|C > E|$=1000, $|E > C|$=0, LM=0.997,
2. $|H > G|$=600, $|G > H|$=400, LM=0.569,
3. $|F > B|$=0, $|B > F|$=0, LM=0.

In the sequential case (situation 1), because e always succeeds c, $LM \cong 1$. When h and g are in parallel, in situation 2, $LM = 0.569$, i.e. a value much smaller than 1. In the case of choice between f and b, in situation 3, $LM = 0$. In general, we can conclude that the LM measure has a value close to 1 when there is a clear tendency of causality between tasks x and y. When the LM measure is close to 0, there is no causality relation between tasks x and y. When the LM measure has a value close to 0.5, then $x > y$ and $y > x$, but a clear tendency of causality cannot be identified.

The global metric GM

The previous measure LM was expressing the succession tendency by comparing the magnitude of $|X > Y|$ versus $|Y > X|$ at a local level. Let us now consider that the number of lines in our log is $\#L$=1000 and the frequencies of tasks a, b and c are $|A|$=1000, $|B|$=1000 and $|C|$=1000. We also know that $|A > B| = 900, |B > A| = 0$ and $|A > C| = 50$ and $|C > A| = 0$. The question is: a is the most likely cause of b or c or both? For a causes b, $LM = 0.996$ and for a causes c, $LM = 0.942$, so we can conclude that a causes both b and c. However, one can argue that c succeeds a less frequently, thus a should be considered the cause of b.

Therefore, we build a second measure, the global metric GM:

$$GM = ((A > B) - (B > A))\frac{\#L}{(A) * (B)} \tag{2}$$

The values for the GM and LM metrics are given in Table III.

Table III. Illustration of GM and LM measures.

X	No. of events	$\|X > A\|$	$\|A > X\|$	LM	GM
B	1000	0	900	0.99	0.90
C	1000	0	50	0.94	0.05

In conclusion, for determining the likelihood of causality between two events x and y, the GM metric is indeed a global metric because it takes into account the overall frequencies of tasks x and y, while

the LM metric is a local metric because it takes into account only the magnitude of $|X > Y|$ versus $|Y > X|$.

The YX and XY metrics

The three metrics presented before were especially developed to be used as predictors for the causality relation, but they are not very useful for deciding between exclusive and parallel relations. However, $|X > Y|$ and $|Y > X|$ frequencies can be used again to decide between exclusive and parallel relations. When between x and y there is an exclusive relation, both $|X > Y|$ and $|Y > X|$ frequencies should be zero or a small value, while for the parallel case both should be relatively high. Because the rule-set that will be induced using these metrics as predictors must be general, we have to take into account also the frequencies of tasks x and y. Therefore we divide $|X > Y|$ and $|Y > X|$ with the minimum of $|X|$ and $|Y|$.

Thus, YX and XY are defined as follows:

- YX: the proportion of $|Y > X|$ accounted by the minimum frequency of x and y i.e. $YX = |Y > X|/min\{|X|, |Y|\}$;
- XY: the proportion of $|X > Y|$ accounted by the minimum frequency of x and y i.e. $XY = |X > Y|/min\{|X|, |Y|\}$;

In Table IV the values for the relational metrics of some task pairs for the Petri net shown in Figure 1 are presented.

5. The induction of the decision rule-sets

In Section 2 we introduced five relational metrics CM, GM, LM, YX and XY to be used in determining the causal and exclusive/parallel relations. The idea is to use the learning material generated in Section 3, to compute the relational metrics and to induce decision rule-sets that detect the relations between two tasks.

When choosing a suitable learning algorithm, we have to establish some criteria. First, we want to obtain a model that can be easily understood and second, we are interested in a fast and efficient algorithm. Ripper is an algorithm that induces rule-sets (Cohen, 1995). It has been shown that Ripper is competitive with the alternative algorithm C4.5rules (Quinlan, 1992) in terms of error rates, but more efficient than C4.5rules on noisy data (Cohen, 1995), thus it seems to meet our requirements.

For inducing a rule-set, we have to provide a set of examples, each of which has been labelled with a *class*. In our case, we have four possible

classes which are the types of log-based relations that can exist between two tasks: “c” for causal, “e” for exclusive, “p” for parallel and “i” for an inverse causal relation. However, we are interested to induce rule-sets for detecting the first three relations, i.e. ”c”, “e” and “p” (the “i” relation is not interesting, because it is not used by the α algorithm to construct the Petri net).

Because we have to induce two independent rule-sets, we need to separate the learning material needed in the first step from the learning material needed in the second step. Detecting the causal relations is the first step, thus we label each instance of the generated learning material (in Section 3) with a “c”, whether there is a causal relation between the tasks and with an “n” if not. In the second step, we select only those pairs of tasks which share the same cause or the same direct successor task. We label these instances with an “e” or a “p”, whenever between the tasks there is an exclusive or a parallel relation.

An excerpt of the table with the class labelling is presented in Table IV. Note the pairs (c,d) and (g,h) which are labelled in Step 1 with an “n” (in the first step they are used as non-causal examples), while in Step 2 they are labelled “e” and “p” respectively, being selected to induce rules that distinguish between the exclusive and the parallel relation.

Table IV. Excerpt from the learning materials used to induce the rule-set for detecting in Step 1 the causal relations and in Step 2, the exclusive/parallel relations, from the log generated by the Petri net presented in Figure 1. x and y represent the task identifiers, CM, GM, LM, YX and XY are the calculated relational measures, and “Rel” contains the “c”, “e” and “p” letter to label the pairs in causal, exclusive and parallel relations.

Step	x	y	CM	GM	LM	YX	XY	Rel
1	a	f	1.000	1.000	0.998	0.000	1.000	c
1	a	b	1.000	1.000	0.998	0.000	1.000	c
1	f	g	0.903	1.091	0.996	0.000	0.515	c
1	f	h	0.857	1.026	0.995	0.000	0.485	c
1	b	a	-1.000	-1.000	0.000	1.000	0.000	n
1	c	d	0.000	0.000	0.000	0.000	0.000	n
1	g	h	-0.019	-0.436	0.317	0.485	0.266	n
2	b	f	0.000	0.000	0.000	0.000	0.000	e
2	c	d	0.000	0.000	0.000	0.000	0.000	e
2	g	h	-0.019	-0.436	0.317	0.485	0.266	p
2	i	h	-0.404	-0.035	0.437	0.266	0.249	p

The induction of the two rule-sets is described in the following two subsections.

5.1. The induction of the rule-set for detecting causal relations

As described in Section 3, the computed relational measures corresponding to the 400 logs are stored into one file that serves as training material for the induction of the rule-sets. In order to obtain these rule-sets, we use Ripper (Cohen, 1995). Ripper algorithm produces ordered rules, by using different methods. We use the default method, i.e. order by increasing frequency. After arranging the classes, Ripper finds rules to separate class1 from classes class2, ..., classn, then rules to separate class2 from classes class3, ..., classn, and so on. To obtain a rule-set for detecting the causal relations, we use only the instances labelled with "c" or "n". We obtain 33 ordered rules for class "c" ("n" is the default class); we refer this rule-set as RIPPER_CAUS. The training error rate for RIPPER_CAUS is 0.08% (the training error rate represents the rate of incorrect predictions made by the model over the training data set). Below is presented a selection of rules that have a coverage higher than 100 positive instances.

Rule1: IF LM>=0.949 AND XY>=0.081 THEN class c [10797 pos, 0 neg]

Rule2: IF LM>=0.865 AND YX=0 AND GM>=0.224 THEN class c [1928 pos, 6 neg]

Rule3: IF LM>=0.844 AND CM>=0.214, CM<=0.438 THEN class c [525 pos, 1 neg]

Rule4: IF LM>=0.741 AND GM>=0.136 AND YX<=0.009 AND
CM>=0.267 AND CM<=0.59 THEN class c [337 pos, 0 neg]

Rule5: IF XY>=0.6 AND CM<=0.827 THEN class c [536 pos, 0 neg]

Rule6: IF LM>=0.702 AND YX<=0.009 AND GM>=0.36 THEN class c [273 pos, 0 neg]

Rule7: IF LM>=0.812 AND CM<=0.96 AND GM>=0.461 THEN class c [142 pos, 0 neg]

Because the feature LM appears multiple times in several rules, we simplify these rules by considering the intersection of the intervals specified by the LM metric. We choose to show the rules with a coverage of over 100 positive instances and less than 7 negative instances. We can remark that these rules cover quite a lot of positive instances and have few negative counterexamples.

Let us interpret these rules. Suppose that we want to detect the relation between two tasks x and y. Rule1 has the highest coverage of positive examples, i.e. almost 70% of "c" instances match this rule. E.g., if the LM measure has a very high value (i.e. there is a big difference in magnitude between $|X > Y|$ and $|Y > X|$ frequencies) and the XY measure is exceeding a small value, there is a high chance there to be a causal relation between x and y. The first condition of

Rule2 specifies LM to be high. The second condition requires the global measure GM to exceed 0.2, i.e. the difference between $|X > Y|$ and $|Y > X|$ frequencies accounted by the overall frequencies of x and y should be sufficiently high. The third condition specify that YX measure must be 0, i.e. $|Y > X| = 0$. In general, the rules require the LM measure to exceed a high value, YX to be a value close to zero, while XY should be bigger than 0. Also, CM and GM measures should be sufficient large.

The conclusion is that we successfully developed (i) measures that are useful to predict causal relations and (ii) the rule-set RIPPER_CAUS seems to have a high performance. In Section 6 we provide further evaluation of our model.

5.2. The induction of the rule-set for detecting exclusive/parallel relations

In order to induce the rule-set for detecting the exclusive/parallel relations, from the whole material generated in Section 3, we select only the pairs of tasks which share the same cause or the same direct successor task. In Table IV, at Step 2, the pairs of tasks in exclusive and parallel relations and the corresponding relational measures are shown. We see that tasks g and h have as same common cause the task f and tasks b and f have as same common cause the tasks a. The pairs in exclusive relation are labelled with "e" (e.g. the pair of tasks (b, f)) and those in parallel relations with "p" (e.g. the pair (g, h)).

When inducing the rule-set for detecting causal relations, we were primarily interested in rules that predict the "c" class. Here we want to develop rules for both exclusive and parallel relations ("e" and "p" classes) and to inspect the difference (if any). We run Ripper with the method to produce unordered rules: Ripper will separate each class from the remaining classes, thus ending up with rules for every class. Conflicts are resolved by deciding in favor of the rule with lowest training-set error. We obtain the RIPPER_ANDOR rule-set with 15 unordered rules, 7 for class "e" and 8 for class "p", with the training error rate 0.38%.

The 14 unordered rules are the following (we omit one rule with very low coverage):

Rule1: IF XY=0 AND GM>=0 THEN class e [4734 pos, 32 neg]

Rule2: IF XY<=0.01 AND CM<=-0.35 AND YX<=0.04 THEN class e [486 pos, 0 neg]

Rule3: IF YX<=0.01 AND LM<=0.31 AND CM>=-0.02 AND CM<=0.04 THEN class e [3006 pos, 2 neg]

Rule4: IF YX<=0.01 AND CM<=-0.26 THEN class e [588 pos, 8 neg]

Rule5: IF YX<=0.01 AND XY<=0 AND CM>=-0.06 AND CM<=0.01 THEN class e

[2704 pos, 7 neg]
Rule6: IF XY<=0.01 AND CM>=0.29 THEN class e [253 pos, 0 neg]
Rule7: IF XY>=0.01 AND YX>=0.02 THEN class p [5146 pos, 0 neg]
Rule8: IF XY>=0.02 AND CM>=-0.24 AND LM>=0.33 THEN class p [3153 pos, 0 neg]
Rule9: IF YX>=0.01 AND CM>=-0.26 AND CM<=-0.07 THEN class p [1833 pos, 1 neg]
Rule10: IF XY>=0.01 AND CM>=-0.24 AND CM<=-0.04 THEN class p [2227 pos, 3 neg]
Rule11: IF YX>=0.01 AND CM>=0.06 THEN class p [1523 pos, 1 neg]
Rule12: IF GM<=-0.01 AND CM>=0.08 THEN class p [223 pos, 0 neg]
Rule13: IF YX>=0.02 AND GM<=-0.03 THEN class p [1716 pos, 1 neg]
Rule14: IF XY>=0.06 THEN class p [3865 pos, 0 neg]

Let us inspect first the RIPPER_ANDOR rule-set for class "p". First, Rule7, which has the highest coverage (it matches almost 93% from "p" instances), requires that both the XY and YX measures exceed zero, what we actually expected: if there are sufficient occurrences of task x and task y next to each other, then there is likely to be a parallel relation between them; if there are few such occurrences, it is likely to be some noise involved and then the relation between tasks is the exclusive one. Rule14 goes in the same direction as Rule7, but requires only the measure XY to be higher than zero. The rest of rules for class "p" have also high coverage; differently from Rule7 and Rule14, they include combinations of all five measures. For example, Rule8 specifies three conditions: the first one requires XY to be higher than zero. The second condition specifies LM to be higher that 0.33: a value for LM that has to exceed 0.33 means that the difference between $|X > Y|$ and $|Y > X|$ frequencies should be relatively small, which is understandable in case of parallel tasks. The third condition that involves the CM measure is less easy to interpret in the context of Rule8.

Looking to the rules for class "e", we expect to have complementary conditions. Rule1 has the highest coverage, but has also 32 counterexamples. This rule specifies that XY should be zero and $GM >= 0$, which makes sense: in case of choice between tasks x and y, we should not see any occurrence of x and y next to each other, which indeed leads to XY=0 and GM=0. In the rest of rules for class "e" we see that mostly of time XY and YX should be smaller than 0.01, that ensure the detection of an exclusive relation when there is noise. The involvement of the CM measure becomes clearer when inspecting all rules, both for the "e" and the "p" class. In general, in case of class "e", CM should be found inside an interval around zero (Rule3 and Rule5), while in case of "p" class, CM should not reach zero (Rule9 and Rule10). Rule6 and Rule11 specify both that CM should be bigger than zero; the decision to be an exclusive or a parallel relation is based

on the XY measure (Rule3), that should be smaller than 0.01, and on YX (Rule11) that should be bigger than 0.01. If there is a choice between tasks x and y and there exist cycles, then x and y do not appear next to each other (rather, y appears somewhere later after x), so the CM measure has to exceed a certain value, as in Rule6.

We can conclude that the obtained rules (i) make a good distinction between exclusive and parallel relations and (ii) the rule-set seems to have a good performance.

6. Rule-sets evaluation

In the previous section we shown the induction of two rule-sets: one for detecting the causal relations and one for detecting exclusive or parallel relations. A natural step is to inspect how well these two rule-sets generalize. We perform two different types of evaluation tests: (i) 10-fold cross-validation and (ii) check the rule-set performance on new test material.

6.1. 10-fold cross validation

K-fold cross-validation (k-fold cv) is an evaluation method that can be used to check how well a model will generalize to new data. The data set is divided into k subsets. Each time, one of the k subsets is used as the test set and the other k-1 subsets are put together to form a training set. Subsequently, the average error across all k trials is computed. Every data point gets to be in a test set exactly once, and gets to be in a training set k-1 times. The variance of the resulting estimate is reduced as k is increased. We set k to the commonly used value of 10.

In order to compare the performance of the 10 obtained models, we compare three averaged performance indicators: the error rate, precision and recall. Error rate is not always an adequate performance measure, because it gives skewed estimates of generalisation accuracy when classes are imbalanced in their frequency. In the case of identifying the relations between tasks, we are interested to see an aggregate of the cost of false positives and false negatives, expressed in terms of recall and precision. In case of causal relations, false positives are false causal relations found, i.e. linking tasks which are not causally related. False negative are actual causal relations that are omitted from the Petri net. Asserting that precision and recall are equally important, we use the combined F-measure (Equation 3).

$$F = \frac{2 * TP}{2 * TP + FP + FN} \tag{3}$$

In Equation 3, TP are class members classified as class members, FP are class non-members classified as class members and FN are class members classified as class non-members.

Performing 10-fold cv experiments with Ripper, we obtain for class "c" an average error rate of 0.11%, 99.35 precision, 98.09 recall and 98.72 F-measure. Detecting classes "e" and "p", Ripper gets an averaged error rate of 0.46%. On class "e" Ripper obtains 98.99 precision, 99.68 recall and 99.33 F-measure, while for class "p" gets 99.72 precision, 99.08 recall and 99.40 F-measure.

Table V. The averaged error rates, precision, recall and F-measure for the 10-fold cv experiments run with Ripper.

10-fold cv	*error rate*	*precision*	*recall*	F[0.1]
Ripper "c" class	0.11%	99.35	98.09	98.72
Ripper "e" class	0.46%	98.99	99.68	99.33
Ripper "p" class	0.46%	99.72	99.08	99.40

Next, we measure the propagated error rate. So far, we inspected the performance of (i) the first rule-set for detecting causal relations and (ii) the second rule-set for detecting exclusive/parallel relations separately. When we induced the second rule-set, we initially selected all the task pairs that share a common cause or a common direct successor. This selection is made from "perfect" data, because we know which are the task pairs that share a common cause or a common direct successor in the learning material. It is interesting to check the performance of a model based on predicted data, i.e., to use the first model to predict the causal relations. From this new learning material, we select the task pairs that share a common cause or a common direct successor and we induce with Ripper a new rule-set that detects exclusive/parallel relations. The 10-fold averaged error rate of this new second rule-set is 0.36% and the averaged F-measure for "e" and "p" classes is 99.83 and 99.85, respectively. These performance indicators are comparable with the performance indicators of the first rule-set induced from perfect data (the averaged error rate is 0.46% and the F-measure is 99.33 for "e" and 99.40 for "p" classes). Because the performance indicators do not differ significantly, we have enough support to use for future predictions the first rule-set for detecting causal relations induced from perfect data.

Based on the 10-fold cross validations experiments, we can say that both rule-sets (i.e. the rule-set that detects causal relations and the rule-set that detects exclusive/parallel relations) seem to have a high

performance on new data. However, this performance was checked on test data that are randomly extracted from the generated learning material. The learning (and testing) material used so far was generated based on a fixed set of Petri-nets. In order to check how robust our rule-sets are on completely new data, we build a Petri net structurally different from the Petri nets used to induce the rule-sets.

6.2. Testing on new data

We build a new Petri net with 33 event types. This new Petri net has 6 OR-splits, 3 AND-splits and three loops (our training material contains Petri nets with at most one loop). We used the same methodology to produce noise, imbalance and different log size as presented in Section 3.

Applying the rule-set RIPPER_CAUS on this new test material results in an error rate of 0.31% and applying the rule-set RIPPER_ANDOR results in an error rate of 0.90%. The confusion matrix and the F-measure for the new test material by applying RIPPER_CAUS and RIPPER_ANDOR rule-sets are presented in Table VI. The performance indicators for the new test material are comparable, although the performance indicators of the two Ripper 10-folds experiments presented in the previous section are slightly better (see Table V for comparison).

Table VI. The confusion matrix and performance results for the rule-sets RIPPER_CAUS and RIPPER_ANDOR on the new test data.

	Predicted			*Predicted*	
Observed	*c*	*n*	*Observed*	*e*	*p*
c	4246	254	*e*	1181	19
n	79	104321	*p*	0	900
Recall	94.36	99.92	*Recall*	98.42	100.00
Precision	98.17	99.76	*Precision*	100.00	97.93
$F[0.1]$	96.23	99.84	$F[0.1]$	99.20	98.96

We can conclude that our rule-sets show good performance also in case of new data, generated by a Petri net with a very different structure that the Petri nets used to induce the rule-sets.

7. Discussion

Based on the performance results obtained on both 10-fold cross-validation and on log data generated by a totally different Petri net, we can conclude that our rule-sets are able to predict with high accuracy causal, exclusive and parallel relations.

Finding the causal, exclusive and parallel relations with our method does not necessarily results in Petri nets equivalent with the original Petri nets used to generate the learning material. It was already formally proven which class of Petri nets it is possible to rediscover the original net using the α algorithm, assuming log completeness and no noise in the process log (Aalst, 2002a). The presented method provides a solution to construct the Petri net model from a process log when the log is incomplete and noisy. However, the degree of incompleteness and noise is affecting in a certain extent the quality of the discovered process model.

By generating experimental data where variations appear in the number of event types, imbalance, noise and log size, we attempt to control how our method misses or incorrectly predicts some relations. We are interested now to investigate the influence of these variations on the rule-sets performance.

In order to inspect the rule-sets performance when number of event types, imbalance, noise and log size are varied, we record the F-measure obtained by applying rule-sets RIPPER_CAUS and RIPPER_ANDOR on each of the 400 individual log files. Three types of F-measures can be calculated:

1. F_C: the F-measure obtained applying the rule-set RIPPER_CAUS. This F-measure is calculated with the formula from Equation 3, where TP are the number of task pairs in "c" relation classified as "c", FP are the number of task pairs in "n" relation classified as "c" and FN are the number of task pairs in "c" relation classified as "n".

2. F_E: the F-measure obtained with the rule-set RIPPER_ANDOR, without considering the propagated error. This means that the previous step of predicting causal relations is considered to execute without errors. The F_E measure is calculated with the same formula from Equation 3, where TP are the number of task pairs in "e" relation classified as "e", FP are the number of task pairs in "p" relation classified as "e" and FN are the number of task pairs in "e" relation classified as "p".

3. F_E_PROP: the F-measure obtained with rule-set RIPPER_ANDOR, considering the propagated error. This means that in the previous

step, some causal relations were missed or incorrectly found. The F_E_PROP is also calculated with formula from Equation 3. TP is the number of task pairs that meet the requirement to be in "e" relation, but this also includes the pairs which apparently have the same cause or the same direct successor (because some causal relation were incorrectly found in the previous step). FP is the number of task pairs in "p" relation classified as "e" and FN is the number of task pairs in "e" relation classified as "p".

Similar formulas are used to compute the F_P and F_P_PROP for pairs of tasks in parallel relations.

We are interested to investigate how the number of event types, imbalance, noise and log size influence the prediction of causal and exclusive/parallel relations. We consider the averaged values of F_C, F_E_PROP and F_P_PROP for all 400 logs. The reason why we consider only F_E_PROP and F_P_PROP is that we are interested in the propagated performance, and not on a performance that assume perfect predictions in the previous step.

In Figure 2 a. we can see how the number of event types is influencing the averaged F_C. Note that the performance is dropping a little in case of the Petri net with 22 event types. A possible explanation is that this particular Petri net has a complex structure which is more difficult to be predicted. The same effect is depicted in Figure 2 b.

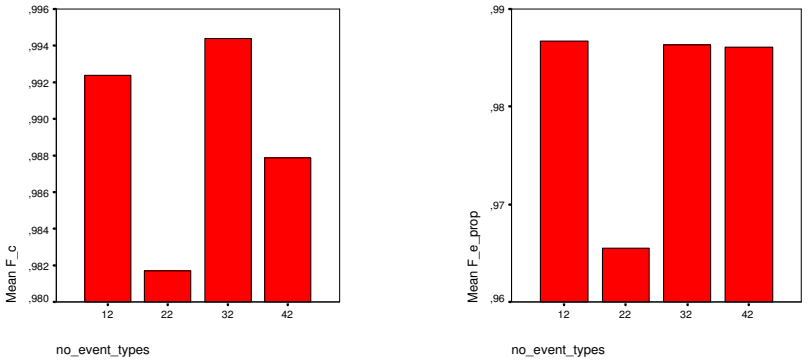
How imbalance in AND/OR splits affects the performance is shown in Figure 3 a. Looking at F_C measure, we see that when the imbalance is increased, the performance is decreasing. A different situation is shown in Figure 3 b., where it seems that if the imbalance is increasing, the performance of finding exclusive relations is also increasing. It seems that a higher level of imbalance helps in distinguishing between exclusive and parallel relations. Inspecting the data, we remark that when the Petri nets are more balanced, than pairs in "e" relation are easier confused with pairs in "p" relation. A possible explanation can be that a rule for "p" class with a very high coverage often misclassifies "e" instances in certain conditions. Rule7 from the model RIPPER_ANDOR has the highest coverage, as we can see below:

Rule7: IF XY>=0.01 AND YX>=0.02 THEN class p [5146 pos, 0 neg]

When classifying "e" instances in case of balanced Petri nets, both XY and YX can exceed 0.01 and 0.02 (because both "xy" and "yx" can occur in the log with comparable probability), thus such instances will be incorrectly classified as "p". When classifying "e" instances in case of unbalanced Petri nets, either XY will exceed 0.01, or YX will exceed 0.02, thus such instances have smaller chance to be classified as "p".

The influence of noise on both performance measures F_C and F_E_PROP are presented respectively in Figure 4 a and b. They show the same expected behavior, i.e. if the noise is increasing, the performance is decreasing.

Figure 5 a and b illustrates how the performance measures F_C and F_E_PROP are influenced by the log size. As we expected, the incompleteness of the log is affecting the performance of finding causal relations: as log size increases, performance increases. However, as the log size increases, the performance of detecting exclusive relations decreases. Inspecting the data, we remark that when the log is larger, than pairs in "e" relation are sometimes easier confused with pairs in "p" relation. The explanation also relates Rule7. When classifying "e" instances in case of a complete log, both XY and YX can exceed 0.01 and 0.02 (because both "xy" and "yx" can occur in the log with comparable probability), thus such instances will be incorrectly classified as "p". When classifying "e" instances in case of incomplete log, either XY will exceed 0.01, or YX will exceed 0.02, thus such instances have smaller chance to be classified as "p".

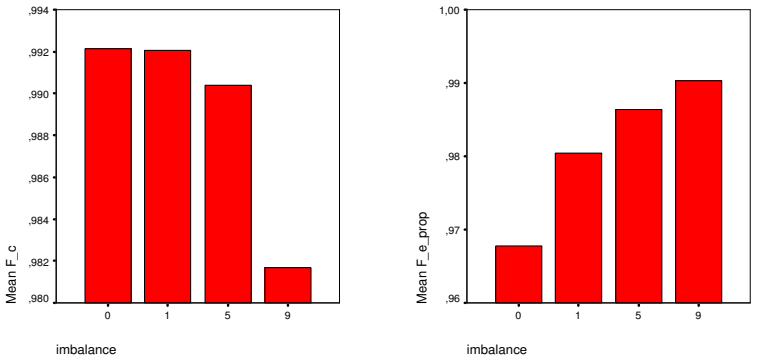


a. No. of event types vs. F_C b. No. of event types vs.F_E_PROP

Figure 2. The effect of the number of event types on rule-set performance.

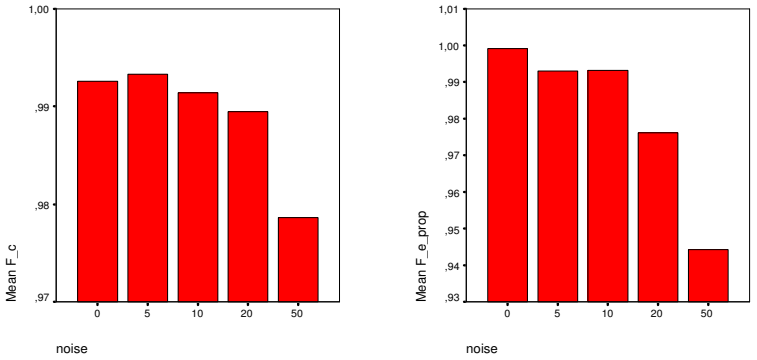
Based on the above findings, we can formulate the following conclusions:

- As expected, more noise, less balance and less cases, each have a negative effect on the quality of the results. The causal relations can be predicted more accurately if there is less noise, more balance and more cases.
- There is no clear evidence that the number of event types has an influence on the performance of predicting causal relations.



a. Imbalance vs. F_C b. Imbalance vs. F_E_PROP

Figure 3. The effect of imbalance on rule-set performance.



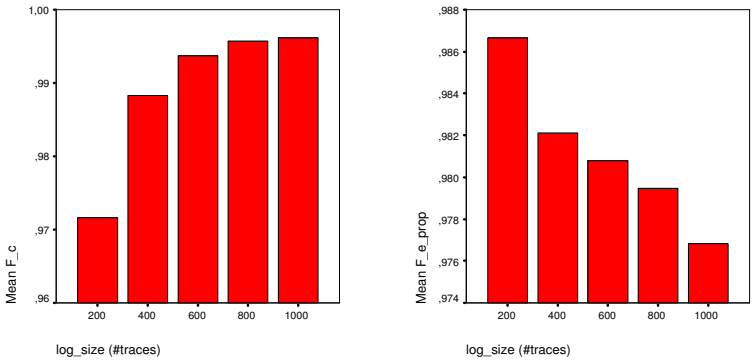
a. Noise vs. F_C b. Noise vs. F_E_PROP

Figure 4. The effect of noise on rule-set performance.

However, causal relations in a structurally complex Petri net can be more difficult to detect.

– Because the detection of exclusive/parallel relations depends on the detection of the causal relations, it is difficult to formulate specific conclusions for the quality of exclusive/parallel relations. It appears that noise is affecting exclusive and parallel relations in a similar way as the causal relations, e.g., if the level of noise is increasing, the accuracy of finding the excusive/parallel relations is decreasing.

When discovering real process data, the above conclusions can play the role of useful recommendations. Usually it is difficult to know the level of noise and imbalance beforehand. However, during the discov-



a. Size log vs. F_C b. Size log vs. F_E_PROP

Figure 5. The effect of log size on rule-set performance.

ery process it is possible to collect data about these metrics. This information can be used to motivate additional efforts to collect more data.

In the next section we present a case study in which we use our method to discover the underlying process from a Dutch governmental institution.

8. Case study

When performing a real-world case study, we have to choose appropriate data. Not all data collected from businesses have an underlying process. For example, the products purchased in one day in a supermarket can show certain interesting patterns, but there is not an underlying *structured process*. A structured process assumes the existence of tasks that are executed in a certain order. An example of suitable processes are those supported by e.g. workflow management systems.

We choose to find the underlying process of handling fine-collection. The case study is done using data from a Dutch governmental institution responsible for fine-collection [1]. A case (process instance) is a fine that has to be paid. If there are more fines related with the same person, each fine corresponds to an independent case. This process has the particularity that as soon as the fine is paid, the process stops. In total there are 99 distinct activities, denoted by numbers, which can be either manually or automatically executed. We select the fines information for 130136 cases. We construct the process log and we apply our process discovery technique to this log.

[1] The name of the organization is not given for reasons of confidentiality.

A screen-shot of the discovered process is given in Figure 6. Because it is difficult to discuss this complex process model, we focus only on parts of the process. We want to compare the discovered model with the process model resulting from a case study done in the traditional approach, i.e. by interviewing the people involved into the process (Veld, 2002). In this study, two sub-processes have been investigated: (i) the COLLECTION sub-process and (ii) the RETURN OF THE UNDELIVERABLE LETTER sub-process.

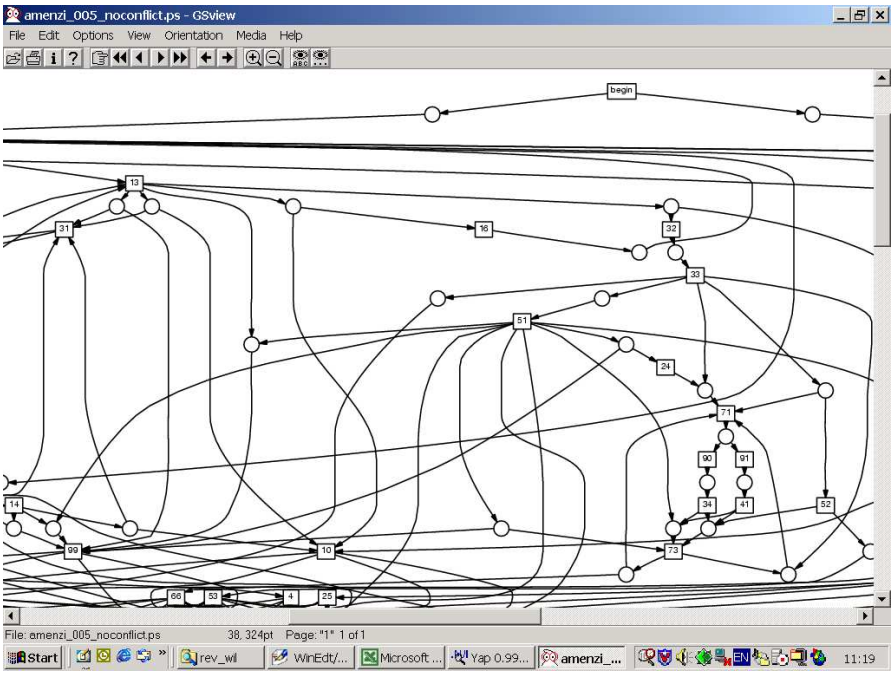


Figure 6. Screen-shot for the process.

8.1. The COLLECTION sub-process

In Figure 7 the process model for the COLLECTION sub-process is presented, as resulting from the case study (Veld, 2002). The process model is designed using workflow modelling blocks. The COLLECTION sub-process starts by receiving from the police the case related documents and then the automated task "initial regulation" (identified by "2") is executed (e.g., a bank transfer form is sent, specifying the fine amount that must be paid). If after 8 weeks the fine is not paid, a reminder is automatically sent (task "6" - "first reminder"). If the fine is not paid within another 8 weeks, a second last reminder is sent (task "7" - "second reminder"). If after these last 8 weeks the fine is still not

paid, a standard verification takes place. This includes the verification of the address done with the help of the Municipal Basic Administration (MBA) (task "23" - "electronic MBA verification"). The verification is followed by a manual activity, "judge standard verification" (task "13"). Note that after any of tasks "2", "6", "7" and "13", a payment (the task "pay", represented by an explicit OR-join) can follow, and then the sub-process stops. Task "13" is represented as an explicit OR-split, thus either the payment is made or the sub-process stops.

Because in the process log it is not recorded a task to mark the completion of cases, we add the task "end" at the end of each trace corresponding to a case. Our method finds that task "2" is directly followed by task "6", "13" and "end". Task "6" is directly followed by task "7". In its turn, task "7" is directly followed by task "23" and "end", and task "23" is directly followed by task "13". Subsequently, there is a parallel relation between pairs of tasks ("6","13"), ("6","end") and ("2","7") and an exclusive relation between ("23", "end") and ("13","end").

Comparing the relations found by our discovery method with the relations from the designed model, we can note that in addition to the designed model, our method finds a causal relation between tasks "2" to "13", that can reveal the existence of an alternate path in practice. Also, in our model, the task "6" ("first reminder") is directly followed only by task "7" ("second reminder"), and not by the ending task "end" (that would imply the payment, as in the designed model). This can correspond to the fact that only after the seconder reminder people are more willing to pay the fine. In Figure 8 the discovered Petri net model is shown.

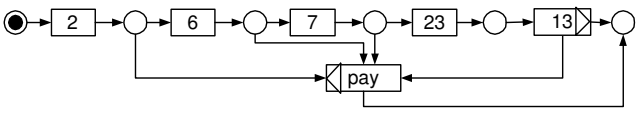


Figure 7. The designed workflow net for the COLLECTION sub-process.

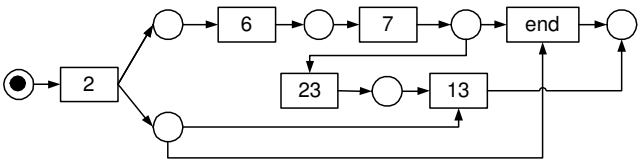


Figure 8. The discovered Petri net for the COLLECTION sub-process.

8.2. The UNDELIVERABLE LETTER RETURN (ULR) sub-process

In case the person that has to pay the fine cannot be found at the specified address (he/she has moved or deceased), the sanction is called an "undeliverable letter return" (ULR).

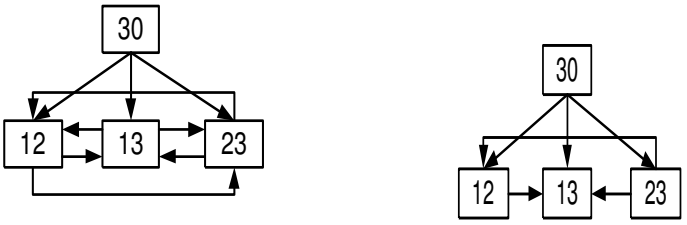
There are two reasons that make the comparison between the designed model and the discovered model difficult. First, for the ULR sub-process a workflow model with specific workflow construct is designed in (Veld, 2002). Second, in the designed model tasks were used that do not appear in the process log. In order to make the comparison possible, we focus only on the tasks that appear in the process log and we compare only the causal relations.

The ULR sub-process starts with the task "30" - "undelivered letter return", that can be directly followed by three tasks: "12", "13" and "23". A written verification ("12") is requested if the sanction is for a company. An electronic MBA verification ("23") is requested in case of a person. The case can be directly judged by an employee ("13"). This may happen also because a wrong type of verification has been issued. Before the case is leaving the sub-process, it must be anyway judged by an employee, even without verification.

In Figure 9 a. and b. are presented the designed model and the discovered model. In both models, task "30" is directly followed by tasks "12", "13" and "23". Also in both models, task "13" is directly following tasks "12" and "23", which is in line with the description made in the previous paragraph. Task "23" is directly followed by task "12" in both models; the explanation can be that when the sanction is for a company, a GBA verification ("23") instead of a written verification is incorrectly required and only afterwards the written verification is required ("12").

However, we can note that in case of the designed model, there are also "reversed" direct connections: task "13" is directly followed by tasks "12" and "23" and task "12" is directly followed by task '23". The explanation can be that such reversed relations can exist, but rather as exceptions than common practice. This reveals that maybe our method is able rather to capture the general process model than the process model containing exceptional paths. We have to conduct more real case studies in order to ascertain this assumption.

When discovering both sub-processes, we came to process models comparable with the designed sub-processes. The usefulness of the discovered process model is manifesting in combination with the designed model, i.e. the common parts of these two models can be considered as the "unquestioning" part of the process, while the differences can be used to detect the questionable aspects of the investigated process.



a. Designed ULR sub-process b. Discovered ULR sub-process

Figure 9. The designed and the discovered ULR sub-process in case of selected tasks "30", "12", "13" and "23".

The discovered models have been inspected by the domain experts. They concluded that our discovered models were able to grasp the important aspects of the process. Moreover, the discovered models revealed aspects that are often questioned when discussing the process model. We conclude that process discovery can provide useful insights into the current practice of a process.

9. Conclusions and future directions

We developed a method that discovers the underlying process from a process log. We generated artificial experimental data by varying the number of event types, noise, execution imbalance and log size. Using these data we induced rule-sets which show high accuracy on classifying new data.

We developed a two-step method: the first step employs a rule-set to detect the causal relations; after the causal relations are found, the second rule-set detects the exclusive/parallel relations between tasks that share the same cause or the same direct successor. Knowing the causal and exclusive/parallel relations, the Petri net is built to obtain the process model.

Our two-step method has a very high performance in classifying new data, being able to find almost all relations in the presence of parallelism, imbalance and noise. Also, we tested our method on a process log generated by a more complex Petri net than the learning material, resulting in a performance close to that on normal held-out test material.

Using the experimental data we investigated the influence of process log characteristics on our model performance. The causal relations can be predicted more accurately if there is less noise, more balance and more cases. However, causal relations in a structurally complex Petri

net can be more difficult to detect. How process log characteristics are influencing the prediction of exclusive/parallel relations is less clear.

A case study was done using a large set of data from a Dutch governmental institution responsible for the collections of fines. We focused on two sub-processes and we compared our discovered models with the designed models. The conclusion was that the discovered models conform reality and moreover, they provide insights into the process current practice.

The current experimental setting confirmed some of our intuitions, e.g. that noise, imbalance and log size are factors that indeed affect the quality of the discovered model. However, as our case study revealed, in real processes more complex situations than we are aware of could be encountered. Therefore, we plan as future work to perform more real-world case studies and consequently adapt our method by considering other factors that may influence the characteristics of the process logs.

Acknowledgements

We would like to thank Mr. R. Dorenbos, H.J. de Vries and Hajo Reijers for their valuable support. Also, we want to thank Alexander in 't Veld for his support and efforts relating the data collection and the presented case study.

References

Aalst, W.M. P. The Application of Petri nets to Workflow Management. *J. of Circuits, Systems and Computers*, 8(1):21–66, 1998.

Aalst, W.M. P., Weijters, A.J.M. M., Maruster., L. Workflow Mining: Which Processes can be Rediscovered? BETA Working Paper Series, WP 74, Eindhoven University of Technology, Eindhoven, 2002.

Aalst, W.M. P., Dongen, B. F. Discovering Workflow Performance Models from Timed Logs. In Y. Han, S. Tai, and D. Wikarski, editors, *International Conference on Engineering and Deployment of Cooperative Information Systems (EDCIS 2002)*, volume 2480 of Lecture Notes in Computer Science, pages 45-63, Springer-Verlag, Berlin, 2002.

Agrawal, R., Gunopulos, D., Leymann, F. Mining Process models from Workflow Logs. In Sixth International Conference on Extended Database Technology. *In Sixth International Conference on Extended Database Technology*, pg. 469–483, 1998.

Cohen, W. W. Fast Effective Rule Induction. *Proceedings of the Twelfth Int. Conference of Machine Learning ICML95*, 1995.

Cook, J. E., Wolf, A. L. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215-249, 1998.

Cook, J. E., Wolf, A. L. Event-Based Detection of Concurrency. *In Proceedings of the Sixth International Symposium on the Foundations of Software Engineering (FSE-6), Orlando, FL*, pp. 35-45, November, 1998.

Cook, J. E., Wolf, A. L Software Process Validation: Quantitatively Measuring the correspondence of a Process to a Model. *ACM Transactions on Software Engineering and Methodology*, 8(2): 147-176, 1999.

Herbst, J. A Machine Learning Approach to Workflow Management. *In 11th European Conference on Machine Learning, volume 1810 of Lecture Notes in Computer Science, Springer, Berlin, Germany*, pp. 183-194, 2000.

Herbst, J. Dealing with Concurrency in Workflow Induction. In U. Baake, R. Zobel, and M. Al-Akaidi, editors, *European Concurrent Engineering Conference*. SCS Europe, 2000.

Herbst, J. *Ein induktiver Ansatz zur Akquisition und Adaption von Workflow-Modellen*. PhD thesis, Universität Ulm, November 2001.

Herbst, J., Karagiannis, D. Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models. In *Proceedings of the Ninth International Workshop on Database and Expert Systems Applications*, pages 745–752. IEEE, 1998.

Herbst, J., Karagiannis, D. An Inductive Approach to the Acquisition and Adaptation of Workflow Models. In M. Ibrahim and B. Drabble, editors, *Proceedings of the IJCAI'99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business*, pages 52–57, Stockholm, Sweden, August 1999.

Herbst. J, Karagiannis, D. Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models. *International Journal of Intelligent Systems in Accounting, Finance and Management*, 9:67–92, 2000.

Maruster, L., Aalst, W.M. P, Weijters, A.J.M. M, Bosch, A., Daelemans, W. Automated Discovery of Workflow Models from Hospital Data. In B. Kröse, M. de Rijke, G. Schreiber, and M. van Someren, editors, *Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2001)*, pages 183–190, 2001.

Maruster, L., Weijters, A.J.M. M, Aalst, W.M. P, Bosch, A. Process Mining: Discovering Direct Successors in Process Logs. In *Proceedings of the 5th International Conference on Discovery Science (Discovery Science 2002)*, Lecture Notes in Computer Science 2534, S. Lange, K. Satoh, C. Smith (Eds.), pages 364-373, Springer-Verlag, Berlin, 2002.

Mitchell, T. Machine Learning. Mc-GrawHill, 1995.

Quinlan, J. Ross. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1992.

Reisig, W., Rosenberg, G. (eds.) Lectures on Petri nets II. Basic models, Springer 1998.

Weijters, A.J.M.M., Aalst, W.M.P. Process Mining: Discovering Workflow Models from Event-Based Data. In B. Kröse, M. de Rijke, G. Schreiber, and M. van Someren, editors, *Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2001)*, pages 283–290, 2001.

Veld, A. J. WFM, een last of een lust?. Master project report (confidential), Eindhoven, June 2002.