Data Mining and Knowledge Discovery, 13, 67–87, 2006

DOI: 10.1007/s10618-005-0029-z

A Rule-Based Approach for Process Discovery: Dealing with Noise and Imbalance in Process Logs

LAURA MĂRUŞTER l. maruster@rug.nl
University of Groningen, P.O. Box 800, 9700 AV, Groningen, NL

A.J.M.M. (TON) WEIJTERS a.j.m.m.weijters@tm.tue.nl
WIL M.P. VAN DER AALST w.m.p.v.d.aalst@tm.tue.nl
Eindhoven University of Technology, P.O. Box 513, 5600 MB, Eindhoven, NL

ANTAL VAN DEN BOSCH antal.vdnbosch@uvt.nl
Tilburg University, P.O. Box 90153, 5000 LE, Tilburg, NL

Published online: 12 May 2006

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 constructs the process model from process log data, by determining the relations between process tasks. To predict these relations, we employ machine learning technique to induce rule sets. These rule sets are induced from simulated process log data generated by varying process characteristics such as noise and log size. Tests reveal that the induced rule sets have a high predictive accuracy on new data. The effects of noise and imbalance of execution priorities during the discovery of the relations between process tasks are also discussed. Knowing the causal, exclusive, and parallel relations, a process model expressed in the Petri net formalism can be built. We illustrate our approach with real world data in a case study.

Keywords: rule induction, process mining, knowledge discovery, Petri nets

1. Introduction

Managing complex business processes calls for the development of powerful information systems, able to control and support the underlying processes. To support a structured business process, such information systems have to offer generic process modelling and process execution capabilities. Because problems are encountered when designing and employing such information systems, the interest in Business Process Analysis and Continuous Process Improvement Efforts increases. Yet, whatever the goal is (e.g. modelling, designing, redesigning or implementing business processes), it needs to be preceded by an analysis of the existing processes. The growing interest into the automation of analysing existing processes, *process mining* can be explained by the availability of logged information, which most information systems (traditional or process-aware) support.

The goal of process mining is to abstract process information from transaction logs (Aalst et al., 2003). Process mining focuses on different levels. Accordingly, this leads to different mining perspectives, such as the process perspective, the organizational perspective, and the case perspective. The *process perspective* focuses

on the control flow, i.e., the ordering of activities. The goal of this type of mining is to find the possible relations between tasks, expressed in terms of a process model, e.g., expressed in terms of a Petri net (Reisig and Rosenberg, 1998) or an Event-driven Process Chain (EPC) (IDS Scheer, 2002; Keller and Teufel, 1998). For process mining with a focus on the process perspective, the specific terms *process discovery* or *workflow mining* are used (Aalst et al., 2004). Using this perspective, it is assumed that it is possible to record events such that (i) each event refers to a task, (ii) each event occurs in a case (i.e., process instance) and (iii) events are totally ordered. A set of such recorded sequences is called *a process log*. For mining the other perspectives, we refer to (Aalst et al., 2003), and http://www.processmining.org. In this paper we will focus on the process perspective.

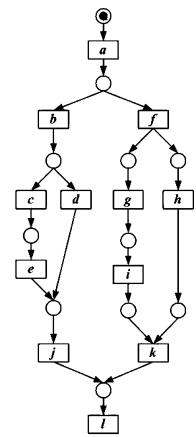
The idea of discovering models from process logs was previously investigated in contexts such as software engineering and workflow management (Agrawal et al., 1998; Cook and Wolf, 1998a; Herbst, 2000a) etc. Cook and Wolf propose alternative methods for process discovery in case of software engineering, focusing on sequential (Cook and Wolf, 1998a) and concurrent processes (Cook and Wolf, 1998b). Herbst and Karagiannis use a hidden Markov model in the context of workflow management, focusing on sequential (Herbst and Karagiannis, 2000; Herbst, 2000b) and concurrent processes (Herbst, 2000a). In Măruşter et al. (2002), a technique for discovering workflow processes in hospital data is presented. Theoretical results are presented in Aalst et al. (2004), providing proof that for certain subclasses of processes it is possible to find the correct process model.

To illustrate the idea of process discovery, consider the process log from Figure 1(a). In this example seven executed cases are logged. Twelve different tasks occur in these cases. We can notice the following example regularities: for each case, the execution starts with task *a* and ends with task *l*; if *c* is executed, then *e* is executed immediately afterwards.

Using the information shown in the process log from Figure 1(a), we can discover the process model shown in Figure 1(b). We represented the model using Petri nets (Reisig and Rosenberg, 1998), where all tasks are expressed as transitions. Petri net

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*

(a) The process log



(b) The Petri net

Figure 1. An excerpt of a process log and the corresponding Petri net process model.

formalism has several advantages, therefore they are often used to represent process models (Aalst, 1998): formal semantics (a clear and precise definition), graphical nature (intuitive and easy to learn), expressiveness (support all primitives needed to model a process), properties (the mathematical foundation allows for reasoning of Petri Nets properties), analysis (many analysis techniques to prove properties and calculate performance measures), vendor independent (not based on software package of a specific vendor). In Figure 1(b), 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 the case of real-world processes which can involve many more tasks and which can exhibit higher levels of parallelism, the problem of discovering the underlying process can become prohibitively complex. Moreover, process mining can be harmed and hindered when process logs contain *noise*—random replacements or insertions of incorrect symbols—or have missing information. A process log is *complete* when all tasks that potentially directly follow each other, in fact do directly follow each other in some trace in the log. In case of a complex process, incomplete process logs will not contain enough information to detect the causal relation between tasks. The notion of completeness is formally defined in Aalst et al. (2004). Note that a process log can be complete without containing all possible cases. A heuristic process discovery method, based on simple count statistics, able to handle certain levels of noise is described in Weijters and Aalst (2001). Nevertheless, in some situations this heuristic method is not robust enough for discovering the complete process. Tackling the problem of process discovery at a more robust level was subsequently introduced in Măruşter, Weijters et al. (2002), using an empirical data-driven approach; more specifically, a logistic regression model able to detect the causal relations (or direct successors) from process logs. However, that logistic regression approach 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 drawback of being too rigid, thus real relations may not be found and false relations may be considered. In Medeiros, Weijters, and Aalst (2004) subsequent advanced issues in robustness towards noisy data and finding causality between tasks are tackled by using genetic algorithms. An overview of issues and related work about Process Mining can be found in Aalst and Weijters (2004).[1]

The problem of noisy and incomplete process log is not the only difficulty which may occur during process mining. A review of challenging process mining problems is made in Aalst and Weijters (2004), which refer to mining hidden tasks, mining duplicate tasks, mining loops, using time, mining different perspectives, and dealing with noise and incompleteness.

In Aalst et al. (2004) it is developed an algorithm called 'the α algorithm', which given a complete process log, it can (re-)discover quite a large class of Petri nets (the discussion about the properties of these Petri nets is beyond the scope of this paper and it is addressed in Aalst et al. (2004)). However, the α algorithm has some limitations, such as (i) mining loops and (ii) dealing with incomplete and noisy process logs. In Medeiros et al. (2004), an extension of the α algorithm is provided, that address the first limitation, e.g. it can handle short loops. In this paper, we address the second limitation of the α algorithm presented in Aalst et al. (2004), namely dealing with incomplete and noisy process logs, to allow its applicability to real-world processes.

The aim of this article is two-fold. First, we describe a rule-based approach for process discovery, assuming the existence of noisy information in the process log and imbalance in execution priorities. Second, we want to gain insight into the effects of noise and imbalance during the process discovery. Our goal is to use machine learning techniques to induce classification rules for (i) 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). Knowing these relations between tasks, a process model can be constructed by using the α algorithm (Aalst et al., 2004).

The article is organized as follows: in Section 2 the types of relations that can exist between two tasks are described. The methodology for generating experimental data used to induce the rule sets is presented in Section 3. In Section 4 the methods for inducing the rule sets are introduced. In Section 5 we evaluate the rule sets, and in Section 6 we discuss the results obtained, focusing on the influence of process characteristics on rule sets performance. In Section 7 we illustrate our approach using real data from a case study. We end with discussing issues for further research in Section 8.

2. The log-based relations

Discovering a model from process logs involves determining the dependencies among tasks. We choose to express these dependencies as log-based relations. The *log-based* relations are formally introduced in Măruşter et al. (2002) and Aalst et al. (2004), in the context of workflow logs and workflow traces. Because we focus on the process perspective, we use the same definitions as in Aalst et al. (2004), this time referring to process logs and process traces.

Definition 1. Process trace, process log

Let T be a set of tasks. $\delta \in T^*$ is a process trace and $W : T^* \rightarrow \mathcal{N}^2$

Figure 1(a) is an example of a process log, "`afghikl` " is an example of a process trace belonging to case 1. This process trace is unique (i.e., $W($`afghikl` $) = 1$). However, the process trace "`abcejl` " appears three times (e.g. for cases 2, 5 and 7) in the log (i.e., $W($`abcejl` $) = 3$). Especially in the case that logs may contain noise the use of frequency information appears crucial.

Definition 2. Succession relation

Let W be a process log over the tasks T with $a, b \in T$. 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 $\delta = t_1\ t_2\ \ldots\ t_n$ in W (i.e., $W(\delta) > 0$), where $i \in \{1, \ldots, n-1\}$ and $t_i = a$, $t_{i+1} = b$. The succession relation $>$ describes which tasks appeared in sequence, i.e., one directly following the other. In the log from Figure 1(a), $a > f, f > g, b > c, h > g, g > h$, etc.

Definition 3. Causal, exclusive and parallel relations

Let W be a process log over the tasks T with $a, b \in T$. 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 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 et al., 2004).

To illustrate the above definitions, let's consider again the process log from Figure 1(a) corresponding to the Petri net from Figure 1(b). If there is no noise, there are three possible situations in which a pair of events (henceforth referred to as tasks) can be related, namely causal, exclusive, and parallel:

causal relation. Tasks c and e have a causal relation, because $c > e$, $e \not> c$, thus $c \rightarrow e$;
exclusive relation. There is a choice between tasks b and f, because $b \not> f, f \not> b$, thus $b \# f$ (and $f \# b$);
parallel relation. Tasks h and i are in parallel, because $h > i$, $i > h$, thus $h \parallel i$ (and $i \parallel h$).

The information on all three types of relations occurring between all tasks is necessary and sufficient to construct the Petri net model using the α algorithm (Aalst et al., 2004). The α algorithm considers first all tasks that stand in a causal relation. Then, for all tasks that share the same immediately-neighboring input or output task, their exclusive or parallel relations are incorporated in the Petri net. Although this algorithm can (re-)discover quite a large class of Petri nets, it also has some limitations, particularly with respect to incomplete and noisy process logs.

The existence of incompleteness and noise in a process log is disturbing the application of the notions presented in Definition 3. Considering the Petri net from Figure 1(b), suppose that we want to discover the relations between pairs of tasks c and e, b and f, and h and i, given a particular example log file. We may find in this file that $c > e$ ten times; however, because of some noisy sequences, we may also find that $e > c$ once. Applying Definition 3, we could conclude that $c \parallel e$, which is incorrect, because actually $c \rightarrow e$. Also, we have to find at least once in the log that $c > e$ in order to determine $c \rightarrow e$, otherwise the log is incomplete and we cannot detect the causal relation between c and e. Similarly, when noise exists, we may find in our noisy example log that both $b > f$ and $f > b$ occur once, which according to Definition 3 means that b and f stand in a parallel relation (actually, $b \# f$!).

We want to be able to use the α algorithm on noisy logs. Therefore, instead of using the definitions given in Definition 3 that break down in noisy circumstances, we use machine learning techniques to induce noise-robust rule sets to determine the status of relations among task pairs. Given these relations, we can apply the α algorithm to construct the Petri net process model.

3. Experimental setting and data generation

Our experimental setup assumes the presence of learning material for inducing rule sets to detect causal, parallel, and exclusive relations. This learning material should resemble realistic process logs and should be sufficiently general to allow for generic rule sets to be induced. We assume here that the following four characteristics underly a typical realistic process, where variations of these characterisics affect the process logs: (i) the

number of possible task types, (ii) the size of 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 these four process log characteristics:

1. The number of task types: we construct Petri nets with different number of task types.
2. The process log size: the log size is expressed by varying the number of traces, where one trace represents the processing of one case.
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 random 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.
4. The imbalance of execution priorities: we assume that tasks can be executed with priorities between 0 and 2. Suppose that in the Petri net from Figure 1(b), after executing task f (which is an AND-split), an imbalance may exist in the priorities of the subsequent execution of tasks g and h. For example, task h can have an execution priority of 0.8 and task g 1.5. This implies that after f, in 35 percent of the cases task h is executed, and in 65 percent of the cases task g is executed.

Note that an imbalance in priorities can affect the rediscovery process negatively. In our example, when $f > h$ is observed less frequently than $f > g$, the causal relation $f \rightarrow h$ may be more difficult to determine than the causal relation $f \rightarrow g$. Moreover, a false causal relation $g \rightarrow h$ may be determined because of some possible occurrences of $g > h$.

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.

Our overall data generation procedure is as follows. 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. Applying this procedure we generate 400 different log files.

4. The relational metrics

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 and Aalst (2001). An excerpt from the D/F table for the Petri net presented in Figure 1(b) is shown in Table 1. For each pair of tasks x and y, the following information is abstracted out of the process log:

1. The overall frequency of task x (notation $|X|$[3]);
2. The overall frequency of task y $|Y|$;
3. The frequency of task x directly preceded by y $|Y > X|$;
4. The frequency of task x directly succeeded by y $|X > Y|$;
5. The frequency of x directly or indirectly preceded by y, but before the next appearance of x, $|Y >>> X|$;
6. The frequency of x directly or indirectly succeeded by y, but before the next appearance of x, $|X >>> Y|$.

The information fields contained in the D/F table, exemplified in Table 1, provide a basic representation of the data on which we intend to induce the rule sets for detecting the log-based causal, exclusive, or parallel relations. However, the raw, unnormalized frequencies of the D/F table cannot be used directly as input features for inducing the rule set. We propose normalized relative metrics from these raw data that can be used more generically to represent the cases to be given as training material to the rule induction method.

The frequencies $|X > Y|$ and $|Y > X|$ from the D/F table are essential for predicting the causal relation $x \rightarrow y$ between tasks x and y. When the difference between $| X > Y|$ and $|Y > X|$ is substantially large, there is a high likelihood that x causes y. We develop three different relational metrics that use the difference between $| X > Y|$ and $|Y > X|$: the causality metric *CM*, the local metric *LM* and the global metric *GM*.

The causality metric CM was first introduced in Weijters and Aalst (2001). If for a given process 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

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

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

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 was also first introduced in Weijters and Aalst (2001). 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, considering the probability of 95% likelihood of the causality relation, is:

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

The basis of this measure is a statistical confidence interval estimator (Mitchell, 1995). Using the expression from Formula 1, we estimate with a probability of 95% the likelihood of the 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(b). 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 the choice between f and b, in situation 3, $LM = 0$. In general, 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.

LM thus expresses the succession tendency by comparing the magnitude of $|X > Y|$ versus $|Y > X|$ at a local level. Consider, for example, 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 whether a is the most likely cause of b or of c. 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 rather considered the cause of b.

We therefore built another measure, the *global metric GM*:

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

Example values for the GM and LM metrics are given in Table 2.

Table 2. Illustration of GM and LM measures.

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

In determining the likelihood of causality between two events x and y, the *GM* metric acts as a global metric because it takes into account the overall frequencies of tasks x and y, while the *LM* metric is a local metric taking into account only the magnitude of $\mid X > Y \mid$ versus $\mid Y > X \mid$.

The *CM*, *LM*, and *GM* metrics have been developed specifically to be used as predictors for the causality relation. They are less useful for deciding between exclusive and parallel relations, for which we need to develop other adequate predictors. While we may know $a \rightarrow x$ and $a \rightarrow y$, we do not know whether $x \# y$ or $x \parallel y$. $\mid X > Y \mid$ and $\mid Y > X \mid$ frequencies from the D/F table can be used again to decide between exclusive and parallel relations. When between x and y there is an exclusive relation, both $\mid X > Y \mid$ and $\mid Y > X \mid$ frequencies should be zero or a small value, while for the parallel case both should be relatively high. Because the rule set to be induced using these metrics as predictors must be general, we have to take into account also the frequencies of tasks x and y; we therefore normalize $\mid X > Y \mid$ and $\mid Y > X \mid$ by dividing it by the minimum of $\mid X \mid$ and $\mid Y \mid$. We define the *YX* and *XY* metrics as follows:

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

In Table 3 the values for the relational metrics of some task pairs for the Petri net shown in Figure 1(b) are presented.

5. The induction and evaluation of decision rule sets

In Section 4 we introduced five relational metrics *CM*, *GM*, *LM*, *YX* and *XY* to be used as predictive features for determining the causal and exclusive/parallel relations between pairs of events. The idea is to use the learning material generated in Section 3, compute the five relational metrics, and induce decision rule sets that detect the existing log-based relations between 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; second, we are interested in a fast and efficient algorithm. Ripper is an algorithm that induces minimal description-length rule sets (Cohen, 1995). It has been shown that Ripper is competitive with the commonly-used alternative algorithm C4.5rules (Quinlan, 1993) in terms of error rates, but more efficient than C4.5rules on noisy data (Cohen, 1995), thus it seems to meet our requirements.

Table 3. Excerpt from the learning material used to induce the rule set for detecting causal relations (Step 1) and the exclusive/parallel relations (Step 2), from the log generated by the Petri net presented in Figure 1(b). x and y represent the task identifiers, *CM*, *GM*, *LM*, *YX* and *XY* are the calculated relational measures, and the relation class ("Rel"); "c" (causal), "n" (non-causal), "e" (exclusive), and "p" (parallel).

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

For inducing the rule sets we have to provide a set of examples, each of which has been labelled with a *class*. We label each example corresponding to the 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.

We induce two independent rule sets. First we separate the learning material needed in the first step, i.e., the detection of causal relations. Therefore, we label each instance of the generated learning material with a "c", whether there is a causal relation between the tasks, else with an "n". In the second step we select from the learning material only the pairs of tasks sharing the same cause or the same direct successor task. We label these instances with an "e" or a "p" when there is an exclusive or a parallel relation, respectively. An excerpt of learning material with this class labelling is presented in Table 3. 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, as labelled examples of exclusive and the parallel relations.

5.1. Induction a rule set for detecting causal relations

The computed relational measures corresponding to the 400 generated logs are stored into one file that serves as training material for the induction of the rule sets. This file contains a total of 341,577 data points. In order to obtain the rule sets, we use Ripper algorithm (Cohen, 1995). This algorithm produces ordered rules according to several optional algorithmic parameters. We use the default method, i.e., ordering by increasing frequency, with the most frequent class as the default rule. After arranging the classes, Ripper finds rules to separate $class_1$ from classes $class_2, \ldots, class_n$, then rules to separate $class_2$ from classes $class_3, \ldots, class_n$, 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 relabeling the training data set). Since training error is not relevant to assess the generalization performance and quality of a rule set, we estimate its generalization performance using test material in Section 5.3. Below we present a selection of rules that cover more 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 can 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.

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: almost 70% of "c" instances match this rule. 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 of a causal relation existing between x and y. Similarly, 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 specifies that the value for the *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.

5.2. *Inducing a rule set for detecting exclusive/parallel relations*

In order to induce the rule set for detecting exclusive/parallel relations from the labelled examples generated in Section 3, we select only the pairs of tasks which share the same cause or the same direct successor task. In Table 3 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 we induced 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 the exclusive and parallel relations ("e" and "p" classes). We employ Ripper with the rule

ordering parameter set to produce unordered rules: with this setting Ripper induces rules for both classes rather than leaving the default rule for one of the two. 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 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% of the "p" instances in the training data), requires that both the *XY* and *YX* measures exceed zero, as 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, there is likely to be some noise involved and then the relation between tasks is probably exclusive. Rule14 goes in the same direction as Rule7, but requires only the measure *XY* to be higher than zero. The remaining rules for class "p" have also high coverage; other than Rule7 and Rule14 they include different 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 involving the *CM* measure is not straightforward to interpret.

Inspecting the rules for class "e" we expect to find 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 a choice between tasks *x* and *y* we would not expect any occurrence of *x* and *y* next to each other, which indeed leads to $XY = 0$ and $GM = 0$. In the other rules for class "e" we see that *XY* and *YX* should be smaller than 0.01, which ensures 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 in an interval close to zero (Rule3 and Rule5), while in case of "p" class, *CM* should not reach zero (Rule9 and Rule10). Rule6 and Rule11 both specify that *CM* should be larger than zero; the decision on an exclusive or a parallel relation is based on the *XY* measure (Rule3), which should be smaller than 0.01, and on *YX* (Rule11), which should be larger than 0.01. If there is a choice between tasks x and y and cycles exist, 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 witnessed in Rule6.

5.3. Evaluation of the rule sets

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 by performing evaluation tests.

Estimating the generalization error can be done with a range of methods (Weiss and Kulikowski, 1991). *k-fold cross-validation* (k-fold CV) is a commonly-used evaluation method that can be used to evaluate 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 joined to form a training set. Subsequently, the average error across the k trials is computed. We set k to the commonly used value of 10 (Weiss and Kulikowski, 1991).

In order to compare the performance of the 10 obtained models, we consider 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 generalization accuracy when classes are imbalanced in their frequencies. 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 (Weiss and Indhurkya, 1998) (Eq. (3)). In Eq. (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.

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

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 achieves 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 (see Table 4). In our previous work (Măruşter, Weijters et al., 2002), we developed a logistic regression approach to detect direct successors (the "c" class), using a global threshold. In case of logistic regression models based on the same 10-fold CV experiments, we obtain an error rate of 2.70%, 99.10 precision, 97.52 recall and 98.29 F-measure. Performing a paired t-test, we compare the performance of the logistic regression model and the rule-set model to detect the "c" class. The outcome is that there is a significant difference between

Table 4. Averaged error rates, precision, recall and F-measures 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

the performance (expressed in error rates) of the two models, and the rule-based model significantly outperform the logistic regression model.

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. However, in practice we do not know which are the task pairs causally related. Therefore, it is interesting to check the performance of the rule set for detecting exclusive/parallel relations based on predicted data, i.e., to use the first rule set to predict the causal relations. From this new learning material we select the task pairs that share a predicted 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%; 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 support to use the induced first rule set for performing future predictions on causal relations.

Based on the outcomes of the 10-fold CV experiments we can conclude that both rule sets (i.e., the rule set that detects causal relations and the rule set that detects exclusive/parallel relations) have a high generalization accuracy on unseen data. However, this performance was checked on test data which is randomly extracted from the generated learning material. The learning (and testing) material used so far was generated on the basis of a fixed and limited set of Petri-nets. In order to check how robust our rule sets are on relatively new data not originating from the same Petri nets as the training material, we check the rule set performance on new test material of increased difficulty. We built a new Petri net with 33 event types, having 6 OR-splits, 3 AND-splits and three loops (our training material was based on 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%; 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 5.

We conclude that our rule sets show good generalization performance on new data, even when generated by a Petri net process model with a different and more complex

Table 5. The confusion matrix and performance results for the rule sets RIPPER_CAUS and RIPPER_ANDOR on new test data with from a more complex Petri net than the training material.

	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	96.23	99.84	*F*	99.20	98.96

structure than the Petri nets used to generate the training instances on which the rule sets were based.

6. Analysis: Effects of noise and imbalance

We concluded in the previous section that our rule sets are able to predict, with high accuracy, the presence of causal, exclusive and parallel relations between pairs of events. Nonetheless, the degree of incompleteness and noise of the process log will affect to a certain extent the quality of the process model. We are interested in investigating the influence of the number of event types, imbalance, noise and log size in the prediction of causal and exclusive/parallel relations.

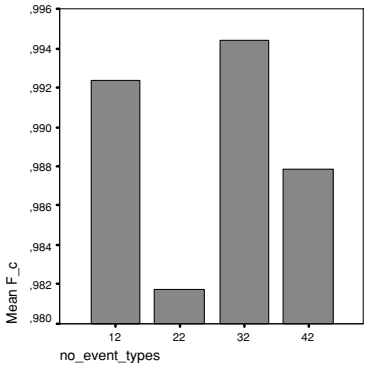
By generating experimental data where variations appear in the number of event types, imbalance, noise and log size, we attempt to control how our method misses or incorrectly predicts some relations. We are now interested to investigate the influence of these variations on the generalization performance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

In order to inspect the rule sets performance when number of event types, imbalance, noise and log size are varied, we apply rule sets RIPPER_CAUS and RIPPER_ANDOR on each of the 400 individual log files and we calculate the following three types of measures:

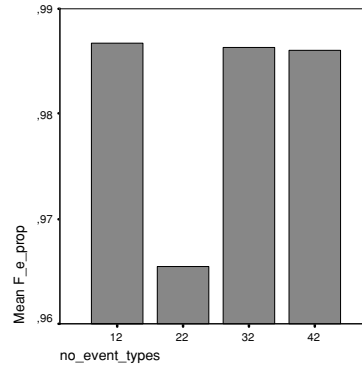
1. F_C: the F-measure obtained applying the rule set RIPPER_CAUS. This F-measure is calculated with the formula from Eq. (3), where *TP* are the number of task pairs in "c" relation classified as "c", *FP* are the number of task pairs in "n" relation classified as "c" and *FN* are the number of task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

An analogous formula is used to compute the F_P_PROP for pairs of tasks in parallel relations.

In Figure 2(a) it can be observed how the number of event types is influencing the averaged F_C. Generalization performance decreases slightly when using logs originating from the Petri net with 22 event types. A possible explanation is that this particular

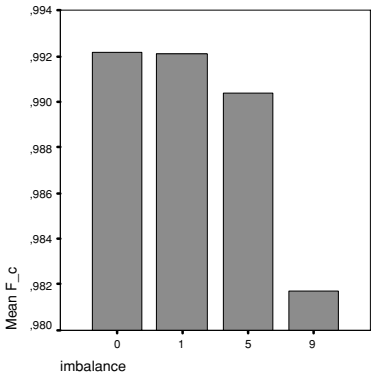


a. No. of event types vs. F_C

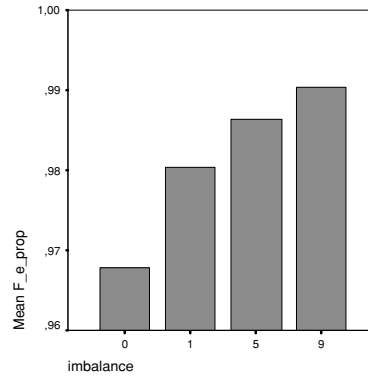


b. No. of event types vs. F_E_PROP

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



a. Imbalance vs. F_C



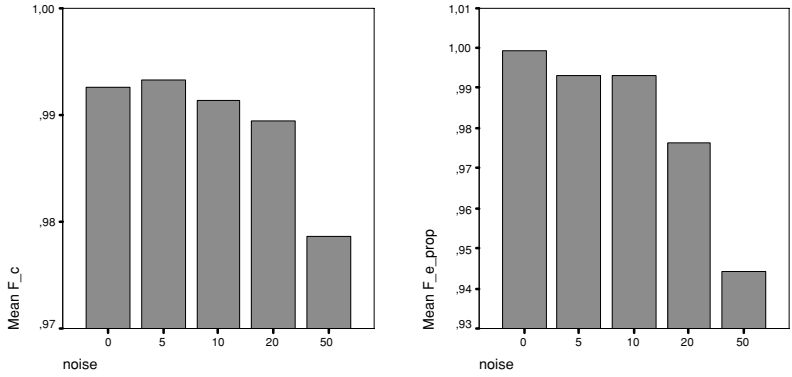
b. Imbalance vs. F_E_PROP

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

Petri net exhibits more parallel behavior, which is more difficult to be predicted. The same effect is depicted in Figure 2(b).

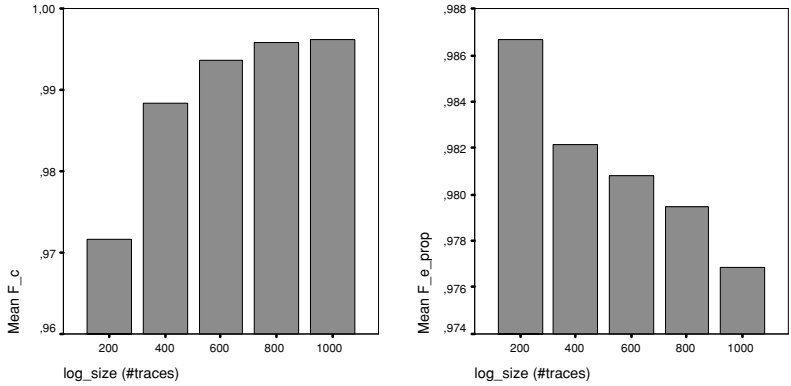
How the imbalance in AND/OR splits affects the performance is shown in Figure 3(a). Inspecting the F_C measure we see that when the imbalance is increased, generalization performance decreases. A different situation is shown in Figure 3(b), where it appears that if the imbalance is increasing, the generalization performance on detecting exclusive relations also increases. It seems that a higher level of imbalance helps in distinguishing between exclusive and parallel relations. When the Petri nets are more balanced, event pairs in an "e" relation are more easily confused with pairs in a "p" relation. A possible explanation is 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 (i.e., 5146 positive and 0 negative examples):

Rule7: IF XY $>=$ 0.01 AND YX $>=$ 0.02 THEN class p



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

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



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

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

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 only *XY* will exceed 0.01 or *YX* will exceed 0.02, thus such instances have a smaller chance to be classified as "p".

Figures 4(a) and (b) display the influence of noise on both performance measures F_C and F_E_PROP. They show the same expected behavior, namely that if the noise level increases, generalization performance decreases.

Figures 5(a) and (b) illustrate how the performance measures F_C and F_E_PROP are influenced by log size. As expected, the incompleteness of the log affecting the generalization 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, pairs in an "e" relation tend to become more easily confused with pairs in "p" relation. One

possible explanation relates again to Rule7. When classifying "e" instances in case of larger logs, both *XY* and *YX* can exceed 0.01 and 0.02 (because both "*xy*" and "*yx*" can occur with comparable probability), thus such instances will be incorrectly classified as "p". When classifying "e" instances in case of smaller logs, either only *XY* will exceed 0.01 or *YX* will exceed 0.02, thus such instances have smaller chance to be incorrectly classified as "p".

Based on the above findings, we can formulate four conclusions. First, more noise, less balance and less cases all have a negative effect on generalization performance. Causal relations can be predicted more accurately if there is less noise, more balance and more cases. Second, there is no clear evidence that the number of event types has an influence on the performance of predicting causal relations. Third, because the detection of exclusive/parallel relations depends on the detection of the causal relations, it is difficult to formulate separate 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 increases, the accuracy of finding the excusive/parallel relations decreases. Fourth, when mining real process data, the above conclusions can play the role of useful recommendations. It is difficult to know the level of noise and imbalance beforehand. However, during the mining process it is possible to collect data about these metrics through the five predictive metrics that form the basic features of our rule sets. This information can be used to motivate additional efforts to collect more data.

7. Case study

To illustrate our approach with real data, we used data from a Dutch governmental institution responsible for fine-collection[4]. A case (or process instance) is a fine that has to be paid; as soon as the fine is paid, the process stops. If there are more fines related with the same person, each fine corresponds to an independent case. In total there are 99 distinct activities, which can be either manually or automatically executed. We applied our technique to a process log consisting of 130136 cases.

Because the entire process model is very complex, we focus only on a part of the process, namely on a sub-process called 'RETURN OF THE UNDELIVERABLE LETTER". In case a person cannot be found at the specified address (he/she has moved or deceased), the sanction is called an "Undeliverable Letter Return" (ULR). We are comparing 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).

The ULR sub-process starts with the task "30" - "undelivered letter return". A written verification ("12") is requested if the sanction is for a company, or 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 Figures 6(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

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

Figure 6. The designed and the discovered ULR sub-process in case of selected tasks "30", "12", "13" and "23".

"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. 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.

8. Conclusions and future directions

We developed an empirical, experimental method for inducing rule sets from process logs to predict the relations between pairs of process tasks. We generated artificial experimental data by varying the number of event types, noise, execution imbalance and log size. On these data we induced rule sets which show high generalization accuracy on classifying new data.

Our method first employs a rule set to detect all 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, a process model can be built using the α algorithm (Aalst et al., 2004) which (re)discovers the Petri net process model that explains the data. Therefore, the contribution of this paper can be seen as successfully complementing the work reported in Aalst et al. (2004): it resolves limitations of the α algorithm, in dealing with noise and incomplete process logs.

The two rule sets have a markedly high performance in classifying new data. They are able to find virtually all relations in the presence of parallelism, imbalance and noise.

We also 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.

Analyzing the experimental data we investigated the influence of process log characteristics on our model performance, which were varied systematically in the generation of our experimental material. 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. We used a large set of data from a Dutch governmental institution responsible for the collections of fines. We focused on one sub-process and we compared our discovered model with the designed model. 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, in real processes more complex situations than we are aware of could be encountered. Therefore, in future work we plan to perform more real-world case studies and consequently adapt our method by discovering and considering other factors that may be influential characteristics of process logs.

Acknowledgments

We would like to thank Dr. Christine Pelletier (University of Groningen) for her valuable comments and remarks during the review of our paper.

Notes

1. For more information see http://www.processmining.org.
2. T^* is the set of all sequences that are composed of zero or more tasks of T. W: $T^* \rightarrow \mathcal{N}$ is a function from the elements of T^* to N (i.e., the number of times an element of T^* appears in the process log).
3. We use a capital letter and || when referring to the number of occurrences of some task.
4. The name of the organization is not given for confidentiality reasons.

References

Aalst, W. van der. 1998. The application of Petri nets to workflow management. The Journal of Circuits, Systems and Computers, 8(1):21–66.

Aalst, W. van der, Dongen, B. van, Herbst, J., Măruşter, L., Schimm, G., and Weijters, A. 2003. Workflow mining: A survey of issues and approaches. Data and Knowledge Engineering, 47(2):237–267.

Aalst, W. van der and Weijters, A. 2004. Process mining: A research agenda. Computers in Industry, 53(3):231–244.

Aalst, W. van der Weijters, A., and Măruşter, L. 2004. Workflow mining: Discovering process models from event logs. IEEE Transactions on Data and Knowledge Engineering 16(9):1128–1142.

Agrawal, R., Gunopulos, D., and Leymann, F. 1998. Mining process models from workflow logs. In Sixth International Conference on Extending Database Technology, pp. 469–483.

Cohen, W. 1995. Fast effective rule induction. In Proceedings of the Twelfth Int. Conference of Machine Learning ICML95.

Cook, J. and Wolf, A. 1998a. Discovering models of software processes from event-based data. ACM Transactions on Software Engineering and Methodology, 7(3):215–249.

Cook, J. and Wolf, A. 1998b. Event-based detection of concurrency. Proceedings of the Sixth International Symposium on the Foundations of Software Engineering (FSE-6), pp. 35–45.

Herbst, J. 2000a. Dealing with concurrency in workflow induction. In U. Baake, R. Zobel, and M. Al-Akaidi (Eds.), European Concurrent Engineering Conference. Society of Computer Simulation (SCS) Europe.

Herbst, J. (2000b). Inducing Workflow models from workflow instances. In Proceedings of the 6th European Concurrent Engineering Conference. Society of Computer Simulation (SCS) Europe, pp. 175–182.

Herbst, J. and Karagiannis, D. 2000. 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.

IDS Scheer. 2002. ARIS Process Performance Manager (ARIS PPM): Measure, analyze and optimize your business process performance (whitepaper). (IDS Scheer, Saarbruecken, Gemany, http://www.ids-scheer.com)

Keller, G. and Teufel, T. 1998. SAP R/3 Process Oriented Implementation. Reading MA: Addison-Wesley.

Măruşter, L., Aalst, W. van der, Weijters, A., Bosch, A. van den, and Daelemans, W. 2002. Automated discovery of workflow models from hospital data. In C. Dousson, F. Höppner, and R. Quiniou (Eds.), Proceedings of the ECAI Workshop on Knowledge Discovery from Temporal and Spatial Data, pp. 32–37.

Măruşter, L., Weijters, A., Aalst, W., and Bosch, A. 2002. Process mining: Discovering direct successors in process logs. In S. Lange, K. Satoh, and C.H. Smith (Eds.), Proceedings of the 5th International Conference on Discovery Science (Discovery Science 2002), Berlin: Springer-Verlag, vol. 2534: pp. 364–373.

Medeiros, A. de, Dongen, B. van, Aalst, W. van der and Weijters, A. 2004. Process Mining: Extending the α-algorithm to Mine Short Loops. BETA Working Paper Series, WP 113, Eindhoven University of Technology, Eindhoven, 2004.

Medeiros, A. de, Weijters, A. and Aalst, W. van der. 2004. Using genetic algorithms to mine process models: Representation, operators and results. BETA Working Paper Series, WP 124, Eindhoven University of Technology, Eindhoven, 2004.

Mitchell, T. 1995. Machine Learning. McGraw-Hill.

Quinlan, J. 1993. C4.5: Programs for Machine Learning. Morgan-Kaufmann.

Reisig, W. and Rosenberg, G. (Eds.). 1998. Lectures on Petri nets I. Basic models, Berlin: Springer-Verlag.

Veld, A. 2002. WFM, een last of een lust? (Confidential Report), Eindhoven University of Technology.

Weijters, A. and Aalst, W. 2001. Process mining: Discovering workflow models from event-based data, B. Kröse, M. Rijke, G. Schreiber, and M. Someren (Eds.), Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2001), pp. 283–290.

Weiss, S. and Indhurkya, N. 1998. Predictive Data Mining. San Francisco: Morgan Kaufmann.

Weiss, S. and Kulikowski, C. 1991. Computer Systems That Learn. Morgan Kaufmann.