KATHOLIEKE UNIVERSITEIT LEUVEN

Faculty of Economics and Applied Economics

A new approach for discovering business process models from event logs

Stijn Goedertier, David Martens, Bart Baesens, Raf Haesen and Jan Vanthienen

DEPARTMENT OF DECISION SCIENCES AND INFORMATION MANAGEMENT (KBI)

KBI 0716

A New Approach for Discovering Business Process Models From Event Logs

Stijn Goedertier[1], David Martens[1], Bart Baesens[1,2], Raf Haesen[1,3] and Jan Vanthienen[1]

[1] Katholieke Universiteit Leuven,
Department of Decision Sciences & Information Management,
Naamsestraat 69, B-3000 Leuven, Belgium
{stijn.goedertier;david.martens;jan.vanthienen}@econ.kuleuven.be

[2] University of Southampton, School of Management, United Kingdom,
Highfield Southampton, SO17 1BJ, United Kingdom,
bart@soton.ac.uk

[3] Vlekho Business School Brussels, Belgium
Koningsstraat 336, 1000 Brussels, Belgium
raf.haesen@vlekho.wenk.be

Abstract

Process mining is the automated acquisition of process models from the event logs of information systems. Although process mining has many useful applications, not all inherent difficulties have been sufficiently solved. A first difficulty is that process mining is often limited to a setting of non-supervised learning since negative information is often not available. Moreover, state transitions in processes are often dependent on the traversed path, which limits the appropriateness of search techniques based on *local* information in the event log. Another difficulty is that case data and resource properties that can also influence state transitions are time-varying properties, such that they cannot be considered as cross-sectional. This article investigates the use of first-order, ILP classification learners for process mining and describes techniques for dealing with each of the above mentioned difficulties. To make process mining a supervised learning task, we propose to include negative events in the event log. When event logs contain no negative information, a technique is described to add artificial negative examples to a process log. To capture history-dependent behavior the article proposes to take advantage of the multi-relational nature of ILP classification learners. Multi-relational process mining allows to search for patterns among multiple event rows in the event log, effectively basing its search on *global* information. To deal with time-varying case data and resource properties, a closed-world version of the Event Calculus has to be added as background knowledge, transforming the event log effectively in a temporal database. First experiments on synthetic event logs show that first-order classification learners are capable of predicting the behavior with high accuracy, even under conditions of noise.

1 Introduction

Process mining is the automated acquisition of process models from the event logs of information systems such as ERP, Role Based Access Control (RBAC), and Workflow Management (WfM) systems (van der Aalst and van Dongen, 2002; van der Aalst et al., 2003; van der Aalst et al., 2007). Event logs contain information about the occurrence of *business events*: who performed a particular activity in the context of a particular business process involving some particular business information at a particular time. In many organizations, such event logs conceal an untapped reservoir of knowledge about the way employees conduct every-day business transactions. To date these event logs have been put to use in the context of Business Activity Monitoring (BAM) and Business Process Intelligence (BPI) for the purpose of among others performance measurement, exception notification and escalation management. The vast quantity of available events in event logs, however, makes it difficult to get a good idea about the generally performed behavior using only descriptive statistics. Instead data mining and machine learning techniques can be used to reconstruct general models of the common behavior that is portrayed by the event logs. The latter is called *process mining*. Process models that have been discovered through process mining enable organizations to compare the behavior in the event log with the business conduct it would expect from its employees and other stakeholders. Such *delta analysis* (van der Aalst, 2004) can be helpful in the context of guaranteeing compliance to new regulations (e.g. the Sarbanes-Oxley Act) (Securities and Exchange Commission, U.S.A., 2002; Kaarst-Brown and Kelly, 2005) or in the context of business process redesign and optimization.

Currently many algorithms have been developed to describe or predict control-flow, data or resource-related aspects of processes and many of them have been integrated into the ProM framework (van Dongen et al., 2005). An important but difficult learning task in process mining is the discovery of sequence constraints from event logs, referred to as Process Discovery (van der Aalst et al., 2004; Alves de Medeiros et al., 2007). Other process learning tasks involve, among others, learning allocation policies (Ly et al., 2005) and revealing social networks (van der Aalst and Song, 2004; van der Aalst et al., 2005) from event logs. However, regardless of the type of knowledge that is to be obtained from the logs, process mining faces a number of inherent difficulties:

1. **Unsupervised learning:** First of all, process mining is often limited to a setting of unsupervised learning because negative information about state transitions that were prevented from taking place is often not available.
2. **Global search:** History often influences current flow of a business process. While a history of related events is a potentially strong predictor and is readily available in process logs, the inclusion of such an event history in the hypothesis space of process mining algorithms is non-trivial because non-locality is difficult to represent. Many current process mining algorithms are based on *local* search techniques in the event log.

3. **Time-varying properties:** Business processes do not only consist of activity events, which represent a state change in the life cycle of an activity, but also consist of data events, which represent a state change in the data life cycle of case data. Case data and resource properties often influence activity state transitions. However, these properties are as time-varying as activities themselves and cannot be considered as cross-sectional.

This article investigates the use of first-order, ILP classification learners for process mining and describes techniques for dealing with each of the above mentioned difficulties. To make process mining a supervised learning task, we propose to include negative events in the event log. When event logs contain no negative information, a technique is described to add artificial negative examples to a process log. To capture history-dependent behavior the article proposes to take advantage of the multi-relational nature of ILP classification learners. Multi-relational process mining allows to search for patterns among multiple event rows in the event log, effectively basing its search on *global* information. To deal with time-varying case data and resource properties, a closed-world version of the Event Calculus has to be added as background knowledge, transforming the event log effectively in a temporal database.

The remainder of this article is structured as follows. First an introduction is provided to first-order classification and it is shown how event logs can be represented in this logic context. In the next section the problem of lacking negative information is discussed and an algorithm is proposed to supplement the event log with artificial negative examples. Provided with this information several event operators are defined in section 4 that are used in section 5 for the purpose of Process Discovery. In section 6 it is shown how the time-varying nature of case and resource properties can be incorporated. Finally, section 7 provides a brief overview of related work.

2 First-Order Classification Learners

Classification learning is learning how to assign a data point to a predefined class or group according to its predictive characteristics. The result of a classification technique is a model which makes it possible to classify future data points based on a set of specific characteristics in an automated way. Classification techniques are often applied for credit scoring (Baesens et al., 2003; Martens et al., ming) and medical diagnostic (Pazzani et al., 2001). In process mining classification learning has, for instance, been applied for "Decision Mining" (Rozinat and van der Aalst, 2006) and Process Discovery (Maruster et al., 2006).

Inductive Logic Programming (ILP) (Muggleton, 1990) is a research domain in Machine Learning in which the learners use logic programming to represent data, background knowledge, the hypothesis space and the learned hypothesis. Formally, an ILP learner will search for a hypothesis $\mathcal{H}$ in a hypothesis space $\mathcal{S}$ that predicts or describes patterns in a data set $\mathcal{D}$. A particularly salient feature of ILP is that the representation language $\mathcal{L}$ of the hypothesis can be explicitly defined as a logic program. Such a representation language

ID	Gender	Age	Income	Important Costumer
C1	M	30	215000	Yes
C2	F	19	140000	Yes
C3	M	56	50000	No
C4	F	48	28000	No

(a) customer

Partner 1	Partner 2
C1	C2
C3	C4

(b) partner

Figure 1: *Multi-relational datasets*

is called the language bias and completely defines the hypothesis space $\mathcal{S}$ that needs to be searched. In addition, users of ILP learners can specify background knowledge $\mathcal{B}$ about the data, that often avoids a lot of preprocessing. Consequently, the effectiveness by which an ILP learner can be applied on a learning task depends on the choices that are made in representing the data, the background knowledge and the language bias. In this paper we make use of Tilde (Blockeel and De Raedt, 1998), a first-order decision tree learner available in the ACE-ilProlog data mining system (Blockeel et al., 2002). As many ILP algorithms (Džeroski and Lavrač, 2001), Tilde is the first-order variant of a propositional learner. In particular, the Tilde algorithm is a first-order generalization of the well-known C4.5 algorithm for decision tree induction (Quinlan, 1993). ILP learners are also called **multi-relational data mining** (MRDM) learners (Džeroski and Lavrač, 2001; Džeroski, 2003). Multi-relational data mining extends classical, uni-relational data mining in the sense it can not only learn patterns that occur within single tuples (within rows), but can also find patterns that may over different tuples of different relations (between multiple rows of a single or multiple tables). To understand the idea, consider the following example. The example database of Figure 1 consists of two tables, whereby the second table indicates which persons from the first table are married with each other (Džeroski, 2003). From this database, one wants to establish a decision tree so that the important customers can be identified swiftly. Propositional learners create classification rules of the following form: IF (income $>$ 100000) THEN important customer = YES. Observe that only the information from the first table was used for the creation of this rule. Relational algorithms on the other hand are able to use the relationships that exist among multiple tuples. An example of such a rule is: IF (x is married with a person with income $>$ 100000) THEN important customer (x) = YES. To allow propositional learners to exploit this multi-relational information the multi-relational data mining problem has to be converted a uni-relational problem, as shown in Figure 2. We can see that we need three extra columns to describe the properties of the partner. This technique has been applied to multi-relational data mining (Lavrac et al., 1991) and allows to keep on using propositional learners. Nonetheless, it is less elegant, as it transfers the problem of non-local search to the input space. More importantly, it also exponentially increases the dimensions of the input space. High dimensional input spaces are typically hard to handle by classical data mining techniques, a problem known as 'curse of dimensionality' (Tan et al., 2005).

For the purpose of discovering history-dependent patterns, this multi-relational prop-

ID	Gender	Age	Income	Gender Partner	Age Partner	Income Partner	Important Costumer
C1	M	30	215000	F	19	140000	Yes
C2	F	19	140000	M	30	215000	Yes
C3	M	56	50000	F	48	28000	No
C4	F	48	28000	M	56	50000	No

Figure 2: *Transformation to uni-relational problem*

erty is much desired, as it allows learning based on *global information* in the **event log**. Alternatively the event history of an event log instance could in part be represented as extra propositions (extra columns), for instance including all immediately preceding event information as extra columns in the event log. To use ILP learners on an event log, the log has to be represented as a logic program of ground facts. In our experiments, an activity event is represented as an atom `event(AId,AT,BId,ET,AgentId,PL,TS)` with following arguments:

- `AId` a unique non-business identifier for the activity
- `AT` represents the activity type
- `BId` represents a unique business identifier of the activity
- `ET` represents the activity event
- `AgentId` represents the worker that brings about the activity state transition
- `PL` is a list of parameters that pertain to the event
- `TS` is a time stamp

It is useful to think of a **process instance** as a *trajectory* in a *state space* (Bider et al., 2000), in which the domain of the different possible activities, events and business concepts span the state space. Declarative classification rules can be used to define the valid state transitions in that state space. Each activity in a process instance can undergo a number of distinct state transitions that are recorded as events, for instance:

- `create(AId,BId)`: creates a new activity instance AId with business identifiers BId. As a result a `created` event is added to the state of the process instance.
- `assign(AId,AgentId)`: the assignment of activity AId to an agent $AgentId$ that is recorded as an `assigned` event.
- `addConcept(AId,C)`, `removeConcept(AId,C)`, `updateConcept(AId,C1,C2)`: add, remove or update a business concept C in the state space. This is recorded respectively as a `conceptAdded`, a `conceptUpdated` or a `conceptRemoved` event. Case data, business concepts, are represented as triples of the form `concept(subject,predicate,object)`. Used in languages such as RDF, triples are in principle capable of representing any first-order atom.

- `complete(AId)`: requests the completion of activity AId, recorded as an event of the type `completed`.

In the sample event logs underneath this paragraph the activity life cycle of two activities within a credit application context is represented.

```
event(9001,applyForCredit,100,created,0,[concept(9001,parent,9000)],0).
event(9001,applyForCredit,100,conceptAdded,0,[concept(100,type,application)],0).
event(9001,applyForCredit,100,assigned,0,[concept(9001,assignedTo,1)],12).
...
event(9001,applyForCredit,100,conceptAdded,12,[concept(100,beneficiary,1)],14).
event(9001,applyForCredit,100,conceptAdded,12,[concept(100,applicant,10)],15).
event(9001,applyForCredit,100,conceptAdded,12,[concept(100,loanType,bullet_loan)],15).
event(9001,applyForCredit,100,conceptAdded,12,[concept(100,collateralType,consumer)],16).
event(9001,applyForCredit,100,completed,1,[],17).
...
event(9006,reviewCredit,100,created,0,[concept(9006,parent,9002)],56).
...
event(9006,reviewCredit,100,assignRejected,0,[concept(9006,assignedTo,10)],71).
event(9006,reviewCredit,100,assigned,0,[concept(9006,assignedTo,8)],78).
event(9006,reviewCredit,100,conceptAdded,8,[concept(100,risk,low)],80).
event(9006,reviewCredit,100,completed,8,[],81).
```

3 Inducing Artificial Negative Examples

Without negative information learning can be much harder. For instance, a two-year old will have more difficulties in learning a precise definition of the concept 'balloon' when shown only a balloon than when presented both a ball and a balloon and pointed to their difference. Event logs rarely contain this negative information. Consequently, it is difficult to identify the distinguishing properties that characterize the underlying process model. A more thorough discussion on on the lack of negative information in event logs can be found in (Alves de Medeiros et al., 2007). Because of the lack of negative information, many learning tasks in process mining are in principle an unsupervised learning tasks. In general, it is necessary to provide learners with a strong *inductive bias*, to obtain useful descriptions in this context of unsupervised learning (Mitchell, 1997). Another consequence of the often lacking negative information is that classification learners cannot be applied, as classification is namely a supervised learning task. To make process mining a supervised learning problem suitable for classification, we propose to include negative information in the event log in the form of negative events. A **negative event** reports that a state transition could not take place. For each positive activity event type one can think of a negative one. For instance, for the event types `created` and `assigned` the event types `createRejected` and `assignRejected` can be conceived. Learning the classification rules that predict whether, given the state of a process instance, a particular state transition can occur, then boils down to learning the classification rule that predicts when either a positive or a negative event occurs. In this way, we have formulated process mining tasks such as Process Discovery, authorization, task allocation, and input validation as classification problems. Sometimes, process logs naturally contain negative events. An access log, for instance,

contains information about the workers that have obtained authorization, and information about the workers who were refused authorization to perform a particular task. In many cases, however, information systems do not reveal their internal functioning in terms of negative events. For instance, when a WfMS creates a number of work items and assigns them to several work trays, it will not expose the work items it did not create or provide information about the work trays to which it could not allocate a work item.

Negative examples can be introduced by replaying each process instance t_i, representing an ordered list of events, event-by-event and verifying whether a state transition of interest ϵ could occur. At each event $e_{(i,k)} \in t_i$, it is tested for each possible activity state transition of interest ϵ whether there exists up to that point k similar traces $t_j : \forall l, l < k, similar(e_{(j,l)}, e_{(i,l)})$ in the event log in which at that point a state transition $e_{(j,k)}$ has taken place that is similar to ϵ, as denoted by a similarity operator $similar(e_{(j,k)}, \epsilon)$. If such a state transition does not occur in other traces, this is an indication that the state transition should be prevent from occurring. Consequently, a negative event can be added at this point k in the event trace t_i. If on the other hand a similar trace is found in which the state transition of interest does occur, than this behavior is present in the event log and no negative event is generated. To avoid an imbalance in the proportion of negative versus positive events the addition of negative events can be manipulated with a negative event injection probability π. More formally, this process of adding negative examples can be described as follows:

1 For each process instance t_i in the event log
2 For each event $e_{(i,k)}$ in t_i
3 For each activity state transition ϵ of interest
4 if $\nexists\ t_j : \forall l, l < k, similar(e_{(j,l)}, e_{(i,l)}) \wedge similar(e_{(j,k)}, \epsilon)$
5 then recordNegativeEvent(t_i,k,ϵ,π)

The addition of artificial negative examples to the input space of a learner, adds the assumption that all possible trajectories through the state space have corresponding process instances in the event log. Formulated differently, adding artificial negative examples to an event log on which later on classification is performed, forces a classification learner to conclude that trajectories that do not occur in the original event log, do not occur in the state space of the process model. This assumption is unrealistic, particularly for the event logs of processes with large state spaces. Consequently, the addition of artificial negative examples to a process log would prevent a learner from generalizing to unseen examples. However, this outcome needs to be put in the right perspective. Firstly, state is a relative notion, such that the similarity operator can be adapted to the learning task at hand. For instance, when learning sequence constraints among activities, the events involving the scheduling, assignment and data manipulation can possibly be left out of consideration. This notion of abstract state allows generalization beyond the observed examples. Secondly, it is useful to induce a process model that covers only the modeled examples. For instance, when mining a control-flow model for the purpose of delta-analysis,

the induced process model should preferably cover all the presented examples, and no more than the presented examples.

Notice that the procedure of injecting negative events potentially requires a large number of process instances. This is particularly the case when the underlying process model contains a lot of concurrent (parallel) activities. Because the trace of a process instance is linearized into a list, many process instances are required to cover each possibility. For instance, N pairwise parallel tasks have $N!$ possible orderings. A possible solution is to limit the number of possible activity events in the log, for instance, by only considering activity `completed` events. Another solution is to to leave out or regroup a number of concurrent tasks in the event log.

4 History-Dependent Processes

Sometimes the behavior of process instances is dependent on their own history. For instance, a worker is refused authorization to perform a certain task, when he or she has performed a related task in the past. Another example is the occurrence of history-based joins in the control flow of a business process (van Hee et al., 2006b). This *non-local* behavior of business processes presents many challenges, not only for process mining but also for process modeling (Alur and Henzinger, 1994; van Hee et al., 2006a).

In representing process mining as first-order classification, we propose to include the event history as a relevant element of the state space of business processes. The multi-relational nature of first-order classification learning allows to search for patterns among an unlimited number of rows in the event history log of each particular process instance. However, the effectiveness by which an ILP learner can be applied on a learning task depends in part on the chosen language bias $\mathcal{L}$. Too simple refinement steps result in refined hypotheses that have little or no extra explanatory power. Too complex refinement steps might specify too large a hypothesis space making search inefficient. In (Alur and Henzinger, 1994; van Hee et al., 2006a), Linear Temporal Logic (LTL) language for representing non-local guard conditions is proposed. LTL is nonetheless unsuitable to make up the language bias of an ILP learner, because there is no immediate transformation of the LTL modal operators to first-order predicate logic. Instead we use more simple event history operators, that, in combination with conjunction, disjunction and negation-as-failure provide a reasonably expressive language bias that yields good results in learning non-local classification problems. The following event operators are required:

- The "*historic event operator*" $\mathcal{HE}^{Time}_{BId}(AT, ET, HT)$ evaluates to true when in the history of process instance BId at time HT, relative to time point T, an activity event of type ET has taken place for an activity of type AT.

- The "*non-event operator*" $\mathcal{NE}^{Time}_{BId}(AT, ET, HT)$ evaluates to true when the specified

event does not occur within a time interval $[HT, Time]$ and is defined as follows:

$$\mathcal{NE}_{BId}^{Time}(AT, ET, HT) \Leftrightarrow \nexists HT_2 \in [HT, Time] : \mathcal{HE}_{BId}^{Time}(AT, ET, HT_2)$$

- The "*has no sequel*" evaluates to true when within a process instance a particular transition has taken place, but that it has not (yet) been followed by another specified state transition. The operator is defined as follows:

$$\mathcal{NS}_{BId}^{Time}(AT_1, ET1, AT_2, ET_2) \Leftrightarrow \mathcal{HE}_{BId}^{Time}(AT_1, ET_1, T_1) \wedge \mathcal{NE}_{BId}^{Time}(AT_2, ET_2, T_1)$$

For readability, the operators will be used in the remainder of the text without the indices and event type parameters, though these will be implicitly assumed.

5 Process Discovery as Learning Preconditions

Process Discovery entails the discovery of the process control flow from the event log (van der Aalst et al., 2004; Alves de Medeiros et al., 2007), and has been the main focus of process mining. Such induced models can be visualized by for example a Petri Net or a workflow net. To learn the control flow from events, several algorithms have been proposed, such as the α (van der Aalst et al., 2004), α^{++} (Wen et al., 2006) and a genetic based approach (Alves de Medeiros et al., 2007).

Process Discovery can be represented as the learning of first-order preconditions. To illustrate this we have taken the "Driver's License" example (Alves de Medeiros et al., 2007), a non-trivial example with non-local non-free choice and hidden activities, and extended it with a parallel construct and loop. This extended example is displayed in Figure 3. For the purpose of mining activity preconditions the above defined "*has no sequel*" event operator $\mathcal{NS}$ has shown to be particularly successful in representing the activity preconditions in the event log. As an illustration, we refer to the induced activity preconditions in Table 1 for the process model in Figure 3.

The following experimental setup was put into place. An artificial event log was generated with 450 process instances from the process model in Figure 3 with a maximum of three allowed loops. As is common, learning was performed on a training set, whereas the reported performance is done on the test set (out-of-sample performance), as to provide an objective measure for the predictive performance on new, unseen examples. The **test log** was created as follows. The entire event log, consisting of about 7300 activity `completed` events was first supplemented with about 7000 negative `completeRejected` events by applying the above described procedure with a negative event injection probability π of 100%. After this procedure the first 350 process instances (the first 350 drivers) were removed from event log to retain a test set of 100 process instances. Although the negative events in the

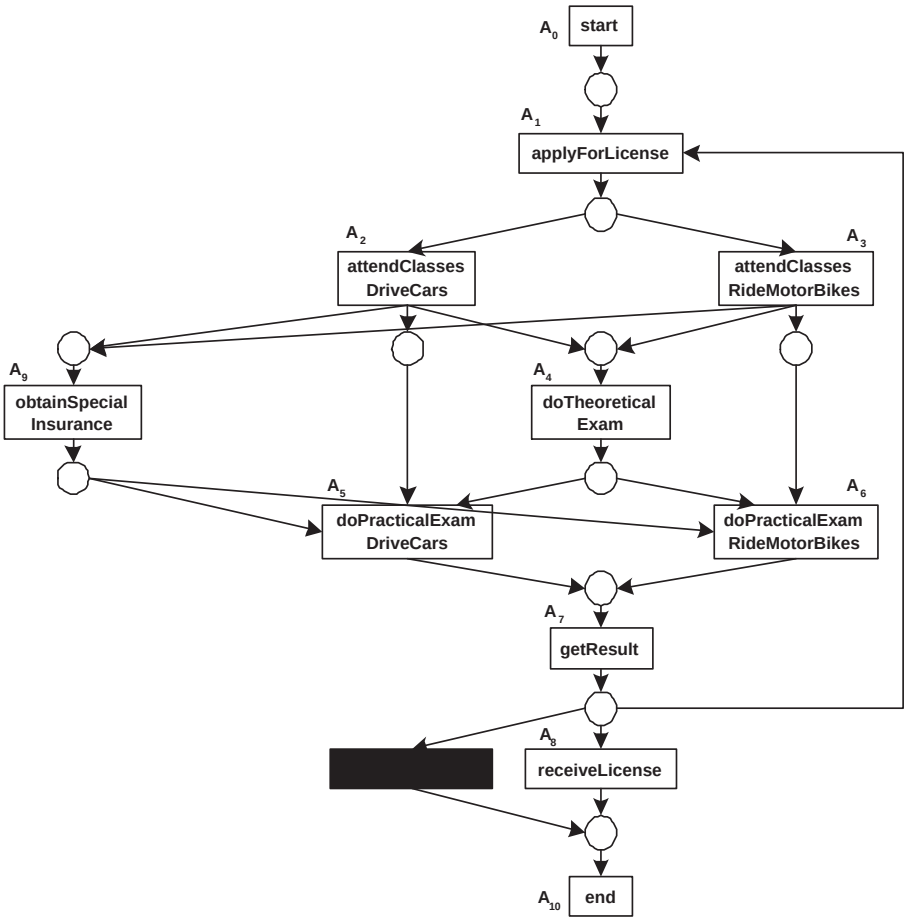

Figure 3: *An extended version of the Diver's License example (Alves de Medeiros et al., 2007)*

Table 1: *A representation of the preconditions of the driver's license process in Figure 3*

	activity	precondition
A_0	start	$\mathcal{NS}(A_0, A_1)$
A_1	applyForLicense	$\mathcal{NS}(A_0, A_1)\ \vee$
		$(\ count(A_1^{created}) < 3\ \wedge\ \mathcal{NS}(A_7, A_1)$
		$\wedge\ \mathcal{NS}(A_7, A_8)\ \wedge\ \mathcal{NS}(A_7, A_9)\ )$
A_2	attendClassesDriveCars	$\mathcal{NS}(A_1, A_2)\ \wedge\ \mathcal{NS}(A_1, A_3)$
A_3	attendClassesRideMotorBikes	$\mathcal{NS}(A_1, A_2)\ \wedge\ \mathcal{NS}(A_1, A_3)$
A_4	doTheoreticalExam	$\mathcal{NS}(A_2, A_4)\ \vee\ \mathcal{NS}(A_3, A_4)$
A_5	doPracticalExamDriveCars	$\mathcal{NS}(A_4, A_5)\ \wedge\ \mathcal{NS}(A_4, A_6)$
		$\mathcal{NS}(A_9, A_5)\ \wedge\ \mathcal{NS}(A_9, A_6)$
		$\wedge\ \mathcal{NS}(A_2, A_5)$
A_6	doPracticalExamRideMotorBikes	$\mathcal{NS}(A_4, A_5)\ \wedge\ \mathcal{NS}(A_4, A_6)$
		$\mathcal{NS}(A_9, A_5)\ \wedge\ \mathcal{NS}(A_9, A_6)$
		$\wedge\ \mathcal{NS}(A_3, A_6)$
A_7	getResult	$\mathcal{NS}(A_7, A_5)\ \vee\ \mathcal{NS}(A_7, A_6)$
A_8	receiveLicense	$\mathcal{NS}(A_7, A_1)\ \wedge\ \mathcal{NS}(A_7, A_8)$
A_9	obtainSpecialInsurance	$\mathcal{NS}(A_2, A_9)\ \vee\ \mathcal{NS}(A_3, A_9)$
A_{10}	end	$\mathcal{NS}(A_8, A_{10})\ \vee$
		$(\ count(A_1^{created}) >= 3\ \wedge\ \mathcal{NS}(A_7, A_1)$
		$\wedge\ \mathcal{NS}(A_7, A_8)\ \wedge\ \mathcal{NS}(A_7, A_9)\ )$

test log were created with information that is in part not present in the test log, this procedure allows best to evaluate the performance of learned classification rules with respect to prohibiting behavior that is not in the process instances. The **training log** is composed of the first 350 process instances. The log consisting of some 5300 `completed` events was supplemented with some 4400 negative `completeRejected` events on the basis of training log events only. To test the performance of first-order activity precondition learning under noise, the training set has been modified with different types of noise. After adding noise, the noisy training sets were supplemented with negative events also with a negative event injection probability π of 10%. In the literature, six noise types are described in (Maruster, 2003; Alves de Medeiros et al., 2007): *missing head*, *missing body*, *missing tail*, *swap tasks*, *remove task*, and *mix all*. For reasons of brevity we report performance results with *swap tasks*, identified by (Alves de Medeiros et al., 2007) as being the most difficult, and *mix all*, which a combination of all other noise types. The reported noise percentages of 10% and 30% are also consistent with the literature.

In classification out-of-sample accuracy results are often reported. However, accuracy has the disadvantage that it is relative to the underlying class distributions. For example, suppose we have a log with 100 positive activity events for one activity type and 9900 negative activity events. A classifier that classifies all activity events as negative, will have an accuracy of 99%, a high figure though classifying none of the positive examples correctly. In Process Discovery it is important that the discovered preconditions allow every event trace in the log (completeness) but preferable no more event traces that do not occur in the log (preciseness) (Alves de Medeiros et al., 2007). Rather than using accuracy as a performance measure, we therefore propose two performance measures that are more suitable to the problem domain of Process Discovery:

- **true positive rate** TP or **completeness**: the frequency of correctly classified positive events in the test set. This probability can be estimated as follows: $TP = E^{+}_{positive}/E^{total}_{positive}$, where $E^{+}_{positive}$ is the amount of correctly classified positive events and $E^{total}_{positive}$ is the total amount of positive events.
- **true negative rate** TN or **preciseness**: the frequency of correctly classified negative events in the test set. This probability can be estimated as follows: $TN = E^{-}_{negative}/E^{total}_{negative}$, where $E^{-}_{negative}$ is the amount of correctly classified negative events and $E^{total}_{negative}$ is the total amount of negative events.

Notice that the true negative rate gives an accurate idea of the preciseness of the learned precondition as negative events are precisely representatives for traces that are not in the sample log. In Table 2 we report these evaluation measures for each precondition learned under different noise circumstances.

To perform first-order classification, we have used the Tilde ILP classification learner. Like C4.5, **Tilde** (Blockeel and De Raedt, 1998; Blockeel et al., 2002) obtains classification rules by recursively partitioning the dataset according to logical conditions, that can be represented as nodes in a tree. This top-down induction of logical decision trees (Tilde)

Table 2: *Out-of-sample performance of the learned preconditions. Both completeness TP and preciseness TN is given as in the following pattern: (TP,TN).*

	Noise Type				
	no noise	*10% mix all*	*10% swap tasks*	*30% mix all*	*30% swap tasks*
A_0	**(1.00,1.00)**	**(1.00,1.00)**	**(1.00,1.00)**	**(1.00,1.00)**	**(1.00,1.00)**
A_1	(1.00,1.00)	(1.00,0.91)	(1.00,0.91)	(1.00,0.91)	(1.00,0.91)
A_2	**(1.00,1.00)**	**(1.00,1.00)**	(1.00,1.00)	(1.00,1.00)	(1.00,0.83)
A_3	**(1.00,1.00)**	**(1.00,1.00)**	(1.00,1.00)	(1.00,1.00)	(1.00,0.83)
A_4	**(1.00,1.00)**	**(1.00,1.00)**	(1.00,1.00)	(1.00,0.91)	(1.00,0.82)
A_5	(1.00,1.00)	(1.00,1.00)	(1.00,1.00)	(1.00,0.92)	(1.00,1.00)
A_6	(1.00,1.00)	(1.00,1.00)	(1.00,1.00)	(1.00,0.92)	(1.00,1.00)
A_7	(1.00,1.00)	(1.00,1.00)	(1.00,0.83)	(1.00,0.92)	(1.00,0.83)
A_8	**(1.00,1.00)**	**(1.00,1.00)**	(1.00,0.91)	(1.00,1.00)	(1.00,0.82)
A_9	(1.00,1.00)	(1.00,0.91)	(1.00,1.00)	(1.00,0.72)	(1.00,0.69)
A_{10}	(1.00,1.00)	(1.00,1.00)	(1.00,1.00)	(1.00,1.00)	(1.00,0.82)

is driven by refining the node criteria according to the provided language bias $\mathcal{L}$. Unlike C4.5, Tilde is capable of inducing first-order logical decision trees (FOLDT). A FOLDT is a tree that holds logical formula containing variables instead of propositions. Variables can be introduced in any node, but must not occur in higher nodes. The language bias $\mathcal{L}$ of Tilde was restricted to the "*has no sequel*" event operator $\mathcal{NS}(A, B)$ (on every combination of activity type A, B), the event operator $\mathcal{HE}(A)$ and aggregate predicate $count(\mathcal{HE}(A))$ (Van Assche et al., 2006) that counts the number of occurrences of an activity type within a specific process instance. Tilde's C4.5 gain ratio was used as a heuristic for selecting the best branching criterion. In addition, Tildes C4.5 post pruning method was used with a standard confidence level of 0.25.

```
doPracticalExamRideMotorBikes(BId,Time,-C)
completedWithoutSequel(attendClassesRideMotorBikes,doPracticalExamRideMotorBikes,BId,Time) ?
+--yes: completedWithoutSequel(attendClassesRideMotorBikes,doTheoreticalExam,BId,Time) ?
|       +--yes: [completeRejected]
|       +--no:  completedWithoutSequel(obtainSpecialInsurance,applyForLicense,BId,Time) ?
|               +--yes: [completed]
|               +--no:  [completeRejected]
+--no:  [completeRejected]
```

The above decision tree represents the precondition for activity `doPracticalExamRideMotorBikes` that has been learned under no noise conditions. As can be observed from the tree its conditions are not entirely structurally equivalent with the suggested preconditions in Table 1. Nonetheless one can observe perfect completeness and preciseness for this activity precondition (A_6 and all the other activities in Table 2 under no noise conditions). An explanation for this outcome is that the preconditions are logically equivalent with respect to the underlying model. For example the condition

```
completedWithoutSequel(obtainSpecialInsurance,applyForLicense,BId,Time)
```

can be shown to be equivalent with

$$\mathcal{NS}(A_9, A_5) \ \wedge \ \mathcal{NS}(A_9, A_6).$$

However, rather than favoring local preconditions the decision tree induction algorithm has favored preconditions with immediate discriminating power. For the moment, this non-preference for local conditions prevents the construction of a graphical model from the learned preconditions. In bold face we have indicated the learned preconditions that are structurally equivalent with the suggested rules. Under conditions of noise, it is observed with regard to the completeness criterion that every induced precondition portrays a perfect recall of the positive events. At least in this respect, the proposed Process Discovery technique is robust to noise. With respect to the preciseness criterion, however, it is observed that the preconditions relax, allowing negative events to take place and thus scoring lower on the preciseness criterion. For example, under 30% swap tasks noise, the induced activity precondition for the parallel task `obtainSpecialInsurance` deteriorates to 0.69, indicating that 31% of the identified negative events are not classified correctly. The reason is that noise invalidates the supplemented negative events as it portrays behavior that is not in the original process model. This behavior with regard to noise is consistent with other learners.

A classification never took more than five minutes to run. Having to learn the preconditions of 10 activities this means that the process model can be learned in under half an hour. In general first-order classification problems potentially have an extremely large search space. However, we have tried to limit the hypothesis space $\mathcal{H}$ by limiting the language bias $\mathcal{L}$ to the three aforementioned language constructs. The greedy search strategy performed by Tilde's C4.5 top down induction of decision trees also contributes to this computational efficiency result.

6 Including Time-Varying Properties

Business processes have a dynamic nature and as such it is naturally the case that case data and resource properties also portray time-varying behavior. For instance, a worker in a bank can move from the risk control department to the sales department. Another example is that a case data property is changed within a business process instance. If learners want to relate the behavior of business processes to the values of case data and resource properties, they will have to take into account this time-varying nature.

In first-order logic there is a formalism that elegantly captures the time-varying nature of facts, namely the Event Calculus. The **Event Calculus**, introduced by Kowalski and Sergot (Kowalski and Sergot, 1986), is a logic programming formalism to represent and reason about the effect of *events* and the state of the system expressed in terms of *fluents*. The Event Calculus is appealing for several reasons. For instance, the Event Calculus builds on a first-order predicate logic framework, for which efficient reasoning algorithms exist. In addition the Event Calculus has the ability to reason about time, in which fluents come to existence or cease to hold dynamically.

To take into account the time-varying nature of properties, we propose to included a closed-world version of the Event Calculus into the background knowledge $\mathcal{B}$ of an ILP

learner. With this background knowledge, we can express the meaning of the above mentioned `conceptAdded`, `conceptRemoved` and `concept Updated` events. As a consequence we can include time-dependent properties into the language bias of ILP learners. Rather than for example including predicates like

```
fromDepartment(agentA,sales)
```

we can now express these properties by including a time point at which they hold:

```
fromDepartment(agentA,sales,t8),
```

effectively transforming the event log into a temporal database.

To illustrate the use of time-varying properties, we have set up an experiment around an artificial credit approval process depicted in Figure 4. Credit approval in practice requires a good collaboration between the sales and the risk department of a bank. Moreover, **strict access control** policies have to be put in place, to prevent unlawful or unwanted acts. An example of such an access rules for the depicted process model can be: "Employees cannot perform the activity `ReviewCredit` when they also have done the activity `CheckDebt` on the `creditApplication`. Furthermore, agents have to be from the department `risk_control` and must not be the `applicant` of the `creditApplication`." In the experiment we have included 200 process instances, with the time-varying behavior that employees can randomly switch between the sales and the risk department. In the tree shown below, the outcome of one such experiment under conditions of zero noise is displayed. It can be observed that the dynamic access control rules is learned with perfect recall, demonstrating the representational power of the proposed language bias and background knowledge. For the reasons of article length, we will not go into more detail into these experiments.

```
authorizationReviewCredit(BId,AgentId,Time)
historicevent(CheckDebt,BId,assigned,AgentId,Time) ?
+--yes: [assignRejected]
+--no:  fromDepartment(AgentId,risk_control,Time) ?
        +--yes: applicant(BId,-F,Time),not(F=AgentId) ?
        |       +--yes: [assignRejected]
        |       +--no:  [assigned]
        +--no:  [assignRejected]
```

7 Related Work

Several authors have represented process mining as classification learning and have to some extent discussed un-supervised learning, history-dependent behavior and time-varying properties. For instance, Maruster et al. (Maruster et al., 2006) were among the first to investigate the use of rule-induction for Process Discovery. However, the authors use propositional rule induction techniques on a table of direct metrics for each process task in relation to the other process tasks, which is generated in a pre-processing step. This approach circumvents the lack of negative examples by this pre-processing step and uses

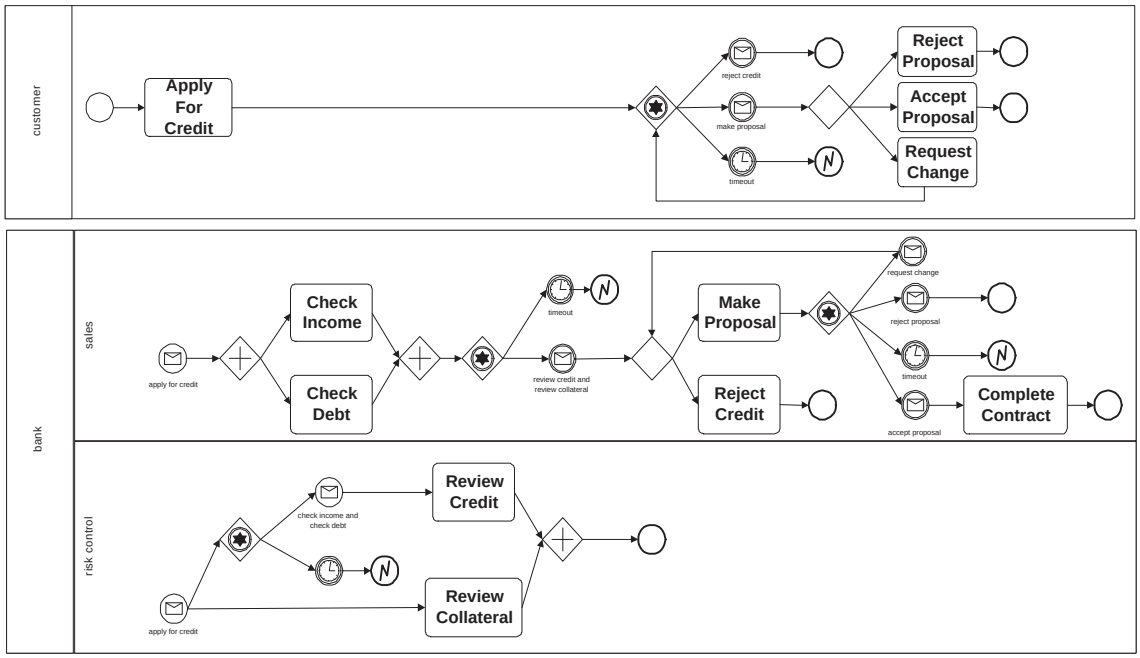

Figure 4: *A BPMN representation of the credit approval process model*

the uni-relational RIPPER algorithm (Cohen, 1995). The multi-relational nature of the ILP learner used in this paper allows to directly perform classification on the event log and is capable of dealing with non-local dependencies and time-varying properties.

Rozinat et al. (Rozinat and van der Aalst, 2006) discuss the use of propositional classification for the purpose of "decision mining". In decision mining so-called decision points are semi-automatically identified in process logs, and the classification problem consists of determining which case data properties lead to taking certain paths in the processes. Because of this preprocessing step, the authors can take into account the time-varying nature of case data properties. The activity precondition learning approach suggested in this paper also allows to take into account (time-varying) case data properties for predicting the conditions under which a task make come into existence, for instance for the discovery of mandatory case data properties in a context of case handling (van der Aalst et al., 2005).

Alves de Medeiros et al. (Alves de Medeiros et al., 2007) point out the difficulties that process mining algorithms have when only taking into account local information, i.e. the immediately preceding and succeeding tasks. The authors have implemented an approach based on genetic algorithms. In this approach no negative examples are introduced, but this problem is circumvented by the incorporation of both a completeness and preciseness measure in the fitness function that drives the genetic algorithm towards suitable models. However, as the algorithm focuses on control flow, the described approach is not capable of taking into account case data properties. As in (Alves de Medeiros et al., 2007), the approach of this article is based on *global information* in the event log and consequently, is capable of discovering sequence, choice, parallelism, loops, invisible tasks, and non-free-choice constructs. What is lacking at this point is a graphical representation of the

discovered process model.

8 Conclusion

This paper has illustrated that it is possible to represent process mining as a first-order classification problem on logs with positive as well as negative events. In particular, it has been shown how three inherent difficulties of process mining, the lack of negative information, history-dependent behavior and time-varying properties, can be elegantly dealt with in this representation. A first Process Discovery experiment has shown promising results on a non-trivial learning problem with loop, parallelism and non-local non-free choice. We have suggested two novel activity-level metrics for evaluating the completeness and preciseness requirements of Process Discovery, namely the true positive and the true negative rate. In the experiment without noise a model can be discovered with perfect completeness and preciseness indicating the suitability of the proposed language bias for Process Discovery. Additional experiments have shown the learner to be robust to noise. The Dynamic Access Control Discovery experiment has illustrated that a closed-world version of the Event Calculus can be applied to deal with time-varying aspects.

References

Alur, R. and Henzinger, T. A. (1994). A really temporal logic. *J. ACM*, 41(1):181–204.

Alves de Medeiros, A., Weijters, A. J., and Aalst, W. M. (2007). Genetic process mining: an experimental evaluation. *Data Min. Knowl. Discov.*, 14(2):245–304.

Baesens, B., Van Gestel, T., Viaene, S., Stepanova, M., Suykens, J., and Vanthienen, J. (2003). Benchmarking state-of-the-art classification algorithms for credit scoring. *Journal of the Operational Research Society*, 54(6):627–635.

Bider, I., Khomyakov, M., and Pushchinsky, E. (2000). Logic of change: Semantics of object systems with active relations. *Autom. Softw. Eng.*, 7(1):9–37.

Blockeel, H. and De Raedt, L. (1998). Top-down induction of first-order logical decision trees. *Artif. Intell.*, 101(1-2):285–297.

Blockeel, H., Dehaspe, L., Demoen, B., Janssens, G., Ramon, J., and Vandecasteele, H. (2002). Improving the efficiency of inductive logic programming through the use of query packs. *J. Artif. Intell. Res. (JAIR)*, 16:135–166.

Cohen, W. (1995). Fast effective rule induction. In Prieditis, A. and Russell, S., editors, *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA. Morgan Kaufmann Publishers.

Dustdar, S., Fiadeiro, J. L., and Sheth, A. P., editors (2006). *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings*, volume 4102 of *Lecture Notes in Computer Science*. Springer.

Džeroski, S. (2003). Multi-relational data mining: an introduction. *SIGKDD Explorations*, 5(1):1–16.

Džeroski, S. and Lavrač, N., editors (2001). *Relational Data Mining*. Springer-Verlag, Berlin.

Kaarst-Brown, M. L. and Kelly, S. (2005). It governance and sarbanes-oxley: The latest sales pitch or real challenges for the it function? In *HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 8*, page 236.1, Washington, DC, USA. IEEE Computer Society.

Kowalski, R. and Sergot, M. (1986). A logic-based calculus of events. *New Gen. Comput.*, 4(1):67–95.

Lavrac, N., Džeroski, S., and Grobelnik, M. (1991). Learning Nonrecursive Definitions of Relations with LINUS. In Kodratoff, Y., editor, *EWSL*, volume 482 of *Lecture Notes in Computer Science*, pages 265–281. Springer.

Ly, L. T., Rinderle, S., Dadam, P., and Reichert, M. (2005). Mining staff assignment rules from event-based data. In Bussler, C. and Haller, A., editors, *Business Process Management Workshops*, volume 3812, pages 177–190.

Martens, D., Baesens, B., Van Gestel, T., and Vanthienen, J. (forthcoming). Comprehensible credit scoring models using rule extraction from support vector machines. *European Journal of Operational Research*.

Maruster, L. (2003). *A machine learning approach to understand business processes*. PhD thesis, Eindhoven University of Technology, Eindhoven.

Maruster, L., Weijters, A. J. M. M., van der Aalst, W. M. P., and van den Bosch, A. (2006). A rule-based approach for process discovery: Dealing with noise and imbalance in process logs. *Data Min. Knowl. Discov.*, 13(1):67–87.

Mitchell, T. (1997). *Machine Learning*. McGraw Hill.

Muggleton, S. (1990). Inductive logic programming. In *Proceedings of the First International Conference on Algorithmic Learning Theory*, pages 42–62.

Pazzani, M., Mani, S., and Shankle, W. (2001). Acceptance by medical experts of rules generated by machine learning. *Methods of Information in Medicine*, 40(5):380–385.

Quinlan, J. R. (1993). *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Rozinat, A. and van der Aalst, W. M. P. (2006). Decision Mining in ProM. In Dustdar et al. (2006), pages 420–425.

Securities and Exchange Commission, U.S.A. (2002). Sarbanes oxley act 2002. Securities and Exchange Commission (SEC), U.S.A.

Tan, P.-N., Steinbach, M., and Kumar, V. (2005). *Introduction to Data Mining*. Addison Wesley.

Van Assche, A., Vens, C., Blockeel, H., and Džeroski, S. (2006). First order random forests: Learning relational classifiers with complex aggregates. *Machine Learning*, 64(1-3):149–182.

van der Aalst, W. (2004). Business alignment: Using process mining as a tool for delta analysis. In *Proceedings of the* 5th *Workshop on Business Process Modeling, Development and Support (BPMDS'04), Caise 04 Workshops*, pages 138–145.

van der Aalst, W., Reijers, H., and Song, M. (2005). Discovering social networks from event logs. *Computer Supported Cooperative Work*, 14(6):549–593.

van der Aalst, W., Reijers, H., Weijters, A., van Dongen, B., Alves de Medeiros, A., Song, M., and Verbeek, H. (2007). Business process mining: An industrial application. *Information Systems*, 32(5):713–732.

van der Aalst, W. and Song, M. (2004). Mining social networks: Uncovering interaction patterns in business processes. In *Proceedings of the International Conference on Business Process Management (BPM 2004)*, volume 3080 of *Lecture Notes in Computer Science*, pages 244–260. Springer.

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

van der Aalst, W. M. P. and van Dongen, B. F. (2002). Discovering workflow performance models from timed logs. In Han, Y., Tai, S., and Wikarski, D., editors, *EDCIS*, volume 2480 of *Lecture Notes in Computer Science*, pages 45–63. Springer.

van der Aalst, W. M. P., van Dongen, B. F., Herbst, J., Maruster, L., Schimm, G., and Weijters, A. J. M. M. (2003). Workflow mining: A survey of issues and approaches. *Data Knowl. Eng.*, 47(2):237–267.

van der Aalst, W. M. P., Weske, M., and Grünbauer, D. (2005). Case handling: a new paradigm for business process support. *Data Knowl. Eng.*, 53(2):129–162.

van Dongen, B. F., Alves de Medeiros, A. K., Verbeek, H. M. W., Weijters, A. J. M. M., and van der Aalst, W. M. P. (2005). The ProM Framework: A New Era in Process Mining Tool Support. In Ciardo, G. and Darondeau, P., editors, *ICATPN*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer.

van Hee, K., Oanea, O., Serebrenik, A., Sidorova, N., and Voorhoeve, M. (2006a). Modelling history-dependent business processes. In *MSVVEIS*, pages 76–85.

van Hee, K. M., Oanea, O., Serebrenik, A., Sidorova, N., and Voorhoeve, M. (2006b). History-based joins: Semantics, soundness and implementation. In Dustdar et al. (2006), pages 225–240.

Wen, L., Wang, J., and Sun, J.-G. (2006). Detecting implicit dependencies between tasks from event logs. In Zhou, X., Li, J., Shen, H. T., Kitsuregawa, M., and Zhang, Y., editors, *APWeb*, volume 3841 of *Lecture Notes in Computer Science*, pages 591–603. Springer.