Statistical Relational Learning for Workflow Mining

Elena Bellodi*[1], Fabrizio Riguzzi[2] and Evelina Lamma[1]

[1]Dipartimento di Ingegneria – Università di Ferrara – Via Saragat, 1 – 44122 Ferrara, Italy.
[2]Dipartimento di Matematica e Informatica – Università di Ferrara Via Saragat, 1 – 44122 Ferrara, Italy.
{*elena.bellodi,fabrizio.riguzzi,evelina.lamma*}*@unife.it*

June 1, 2015

Abstract

The management of business processes can support efficiency improvements in organizations. One of the most interesting problems is the mining and representation of process models in a declarative language. Various recently proposed knowledge-based languages showed advantages over graph-based procedural notations. Moreover, rapid changes of the environment require organizations to check how compliant are new process instances with the deployed models. We present a Statistical Relational Learning approach to Workflow Mining that takes into account both flexibility and uncertainty in real environments. It performs automatic discovery of process models expressed in a probabilistic logic. It uses the existing DPML algorithm for extracting first-order logic constraints from process logs. The constraints are then translated into Markov Logic to learn their weights. Inference on the resulting Markov Logic model allows a probabilistic classification of test traces, by assigning them the probability of being compliant to the model. We applied this approach to three datasets and compared it with DPML alone, five Petri net- and EPC-based process mining algorithms and Tilde. The technique is able to better classify new execution traces, showing higher accuracy and areas under the PR/ROC curves in most cases.

Keywords Workflow mining; Process Mining; Knowledge-based Process Models; Inductive Logic Programming; Statistical Relational Learning; Business Process Management.

*Corresponding author, Tel./Fax +390532974827

1 Introduction

Organizations usually rely on a number of processes to achieve their mission. These processes are typically complex and involve a large number of people. An organization performance critically depends on the quality and accuracy of its processes, which are therefore a fundamental asset. The area of Business Processes Management (see e.g.[23]) is devoted to the study of *process models*. The problem of automatically mining a structured description of a business process model from real data in the context of workflow management was first introduced in [1]. The input data consists of execution traces (or histories) of the process. Their collection is performed by information systems, which log the activities executed by the users. This problem has been called Process Mining or Workflow Mining.

In the last decade, many different proposals have been developed for mining process models from logs and for identifying deviations between logs and a given predefined process model, the so-called "conformance checking" problem.

Traditional systems based on modeling languages as Petri nets [51, 50] or Event-driven Process Chains (EPCs) [53] are sometimes too rigid because they impose a strictly predefined execution flow to the business process. The management of dynamic processes in rapidly changing organizations generated the need of different languages, more knowledge-based and *declarative*. Models in these languages express only *constraints* on the process execution rather than specifying the sequence (order) of tasks to be executed as paths in a net. DecSerFlow [46], ConDec [40] and $\mathcal{S}$CIFF [4, 3] are examples of such languages.

In particular, $\mathcal{S}$CIFF (Society Constraint IF and only iF) adopts first-order logic to represent the constraints. DPML (Declarative Process Model Learner) [31] is a process mining technique for learning models expressed in a subset of $\mathcal{S}$CIFF. DPML discovers a set of declarative constraints from a set of traces labeled as compliant or not. A compliant trace represents a correct course of a process execution. Logic is an expressive knowledge representation formalism which allows to cope with a variable number of entities as well as with the relationships that hold among them. However, logic manages only the complexity of real world domains but not the uncertainty that often characterizes them.

The Probabilistic Inductive Logic Programming [17] and Statistical Relational Learning (SRL) [24] fields aim at overcoming this limitation of logic. A multitude of different formalisms [27, 37, 12, 18, 43] combine probabilistic reasoning with logic, databases or Inductive Logic Programming (ILP) [38]. This combination has been highly successful in a variety of fields where uncertain relations hold among entities, from social networks analysis to entity resolution, from collective classification to information extraction.

In this paper we propose a SRL approach to process mining and conformance checking with two contributions: (1) a discovery procedure of process models from event logs which integrates existing ILP and SRL techniques; (2) probabilistic conformance analysis that, given a model and a new set of traces, provides the probability that they are compliant to the model.

To realize the first goal, we start from the DPML algorithm [31] that induces

expressive logical business rules and then we turn them into probabilistic rules using Markov Logic (ML) [43, 20]. Markov Logic is a SRL formalism which adds a weight (real number) to each formula of a first-order knowledge base reflecting "how strong" the constraint is. We propose to learn the weights of the induced process rules by means of the Alchemy system, a suite of algorithms for Markov Logic [29].

To realize the second contribution, we perform inference with Alchemy using as inputs the weighted process model and a set of unclassified traces, to verify their compliance in probabilistic terms. In this way it is possible to have a ranking of traces on the basis of their probability, from the less probably compliant to the more probably compliant ones.

We differ from traditional process mining research in the following aspects. First, we perform mining from both compliant and non compliant traces, while usually only compliant traces are used as input to the learning algorithm, see e.g. [1, 47, 25]. Learning a model from both compliant and non compliant traces is interesting if an organization has two or more sets of process executions and wants to understand in what sense they differ: for example, a bank may divide its transactions into fraudulent and normal ones and may desire to learn a model that is able to discriminate the two. This is also particularly useful for the cases under study in the paper. Moreover, in this way our approach is able to learn a model which expresses not only what should be done, but also what is forbidden.

Second, important advantages come from adopting Logic Programming (LP). A declarative representation of a process model shows in an intuitive and easily readable way the constraints without sacrificing expressiveness. Thanks to the LP representation, it is possible to consider structured atomic activities, taking into account attributes of events to properly model complex objects of real world domains, differently from Petri nets procedural language. As a consequence of the LP representation, it is possible to exploit ILP techniques for learning models, given a background knowledge together with a set of positive and negative examples. Examples represent compliant and non compliant execution traces so both kinds of traces are needed in our logs.

Third, with Markov Logic we can take into account uncertainty related to the fact that process models may be incomplete: all possible scenarios may not be covered by the model; business processes are dynamic and the workflows may change but the prescribed process models may not be updated accordingly.

The final aim of the paper is to demonstrate that, given a logical-probabilistic process model and an unknown log, the probabilistic classifications made are more accurate compared to non-probabilistic process models. Our approach is compared with various process mining/classification algorithms based on procedural or logical languages, that output a sharp binary classification of traces (compliant/non compliant). Our approach is illustrated by considering the NetBill e-commerce protocol [5, 2], the health care process of cervical cancer screening proposed by the health authority of the Emilia Romagna Italian region and the process of university students' careers at the University of Ferrara.

The paper is organized as follows. In Section 2 we discuss the representation of execution traces and process models using LP. Section 3 recalls the DPML

algorithm for mining logic models, while Section 4 illustrates how to make it probabilistic under Markov Logic. Section 5 presents related work. Section 6 shows the experimental evaluation of the proposed approach on three datasets and Section 7 concludes the paper.

2 Representing Process Traces with Logic

2.1 First-Order Logic

A *first order alphabet* Σ is a set of predicate symbols and function symbols (or functors) together with their arity. A functor with arity 0 is called a *constant*. A *term* is either a variable or a functor applied to a tuple of terms of length equal to the arity of the functor. An *atom* A is a predicate symbol applied to a tuple of terms of length equal to the arity of the predicate. A *literal* L is either an atom A or its negation $\neg A$; in the latter case it is called a *negative literal*. Here we use the LP convention of indicating predicates and constants with alphanumeric strings starting with a lowercase character and variables with alphanumeric strings starting with an uppercase character.

A clause C is a formula in the form

$$b_1 \wedge \ldots \wedge b_n \rightarrow h_1 \vee \ldots \vee h_m \tag{1}$$

where b_i are logical literals and h_i are logical atoms. We define $Head(C) = \{h_1, \ldots, h_m\}$ and $Body(C) = \{b_1, \ldots, b_n\}$. When $n = 0$ C is called a *fact*, when $m = 1$ C is called a *program clause*. The conjunction of a set of literals is called a *query*. A *theory* P is a set of clauses, a *normal logic program* B is a set of program clauses, i.e., a set of of formulas of the form

$$b_1, \ldots, b_n \rightarrow h$$

where h is an atom and all the b_is are literals. A normal logic program is *range--restricted* if all the variables appearing in the head of clauses also appear in positive literals in the body.

A term, atom, literal, clause or query is *ground* if it does not contain variables. A *substitution* θ is an assignment of variables to terms: $\theta = \{V_1/t_1, \ldots, V_n/t_n\}$. The application of a substitution to a term, atom, literal, query or clause T, indicated with $T\theta$, is the replacement of the variables appearing in T and in θ with the terms specified in θ. $T\theta$ is called an *instance* of T. The *Herbrand universe* $H_U(\Sigma)$ is the set of all the ground terms that can be built with function symbols of Σ. The *Herbrand base* $H_B(\Sigma)$ is the set of all the ground atoms that can be built with predicates of Σ and terms of $H_U(\Sigma)$. A *Herbrand interpretation* over $H_B(\Sigma)$ (or just interpretation) is a set of ground atoms, i.e. a subset of $H_B(\Sigma)$.

We now define the truth of a formula in an interpretation. Let I be an interpretation and T a formula. T is true in I, written $I \models T$, if

- $a \in I$, if T is a ground atom a;

- $a \notin I$, if T is a ground negative literal $\neg a$;
- $I \models a$ and $I \models b$, if T is a conjunction $a \wedge b$;
- $I \models a$ or $I \models b$, if T is a disjunction $a \vee b$;
- $I \models \phi\theta$ for all θ that assign a value to all the variables of $\mathbf{X}$, if $T = \forall \mathbf{X}\ \phi$, where $\mathbf{X}$ is the vector of all variables appearing in ϕ;
- $I \models \phi\theta$ for a θ that assigns a value to all the variables of $\mathbf{X}$, if $T = \exists \mathbf{X}\ \phi$, where $\mathbf{X}$ is the vector of all variables appearing in ϕ.

A clause C is true in an interpretation I iff, for all the substitutions θ grounding C, $(I \models Body(C)\theta) \rightarrow (Head(C)\theta \cap I \neq \emptyset)$. Otherwise, it is false.

A theory P is true in an interpretation I iff all of its clauses are true in I and we write $I \models P$. If P is true in I we say that I is a *model* of P. If at least one clause of the theory is false in an interpretation I, the whole theory P is false in I.

2.2 Process Traces

A trace t is a sequence of events. Each event is described by a number of attributes. The only requirement is that one of the attributes describes the event type. Other attributes may be the executor of the event or other specific information. An example of a trace is $\langle a, b, c \rangle$ representing that a, b and c are events executed in a sequence.

A *process model* H is a formula in a language for which an interpreter exists that, when applied to a model H and a trace t, returns answer "yes" if the trace is compliant with the description and "no" otherwise. In the first case we write $t \models H$, in the second case $t \not\models H$. We will refer to compliant and non compliant traces also as *positive* and *negative* traces. A bag of process traces L is called a *log*. The aim of Process Mining is to infer a process model from a log.

A process trace can be represented as a logical interpretation: each event is modeled with a ground atom whose predicate is the event type and whose arguments store its attributes. Moreover, the atom contains an extra argument indicating the position in the sequence or its execution time mapped to an integer. For example, the trace $\langle a, b, c \rangle$ can be represented with the interpretation $\{a(1), b(2), c(3)\}$ where 1, 2 and 3 represent the sequence position or the execution time. To represent a log in a file we enclose each interpretation in a block beginning with `begin(model(<id>)).` and ending with `end(model(<id>)).`, where `<id>` is a logical constant representing the trace's unique identifier. All facts in between the model delimiters describe events of the trace. Two extra facts - `pos/0` and `neg/0` - are used within a trace to label it as positive or negative respectively. The closed world assumption holds, i.e. all the events except for those appearing in the traces are false.

Besides the traces, we may have some general knowledge that is valid for all traces. This information will be called *background knowledge* and we represent it as a normal logic program B. Various semantics have been proposed for normal

logic programs, in this paper we consider Clark's completion [11] that assigns a program B a single interpretation $M(B)$, its model. The rules of B allow to complete the information present in a trace t, storing only once the rules that are not specific to a single trace but are true for every trace. For example, the background knowledge may contain clauses which define precedence and succession relationships involving the position argument of the events. Rather than simply t, we now consider $M(B \cup t)$ as the representation of a trace, i.e. the model according to Clark's completion of the program obtained by adding to B the atoms of t as ground facts.

2.3 Process Models

The process language we consider is a subset of the *declarative* $\mathcal{S}$CIFF language, originally defined in [3, 4] for specifying and verifying interaction in open agent societies.

A *process model* in our language is a set of Integrity Constraints (ICs). An IC C is a logical formula of the form

$$\begin{array}{l} Body \rightarrow \ \exists\ (ConjP_1) \vee \ \ldots \vee \ \exists\ (ConjP_n) \\ \vee\ \forall\ \neg(ConjN_1) \vee \ \ldots \vee \ \forall\ \neg(ConjN_m) \end{array} \tag{2}$$

where $Body$, $ConjP_i$ $(i = 1, \ldots, n)$ and $ConjN_j$ $(j = 1, \ldots, m)$ are conjunctions of literals built over event predicates or over predicates defined in the background knowledge. In particular, $Body$ is of the form $b_1 \wedge \ldots \wedge b_l$ where the b_i are literals; $ConjP_i$ is a formula of the form $event(attr_1, \ldots, attr_r) \wedge d_1 \wedge \ldots \wedge d_k$ where $event/r$ is an event predicate and d_i are literals; $ConjN_j$ is also a formula of the form $event(attr_1, \ldots, attr_r) \wedge d_1 \wedge \ldots \wedge d_k$. Literals d_k are generally taken from the background knowledge. Variables in the $Body$ are implicitly universally quantified with scope the entire formula; the quantifiers in the head apply to all the variables not appearing in $Body$, so (2) is range-restricted.

We will use $Body(C)$ to indicate $Body$ and $Head(C)$ to indicate the formula $\exists(ConjP_1) \vee \ldots \vee \exists(ConjP_n) \vee \forall\neg(ConjN_1) \vee \ldots \vee \forall\neg(ConjN_m)$ and call them respectively the *body* and the *head* of C.

$Body(C)$, $ConjP_i$ $i = 1, \ldots, n$ and $ConjN_j$ $j = 1, \ldots, m$ will be sometimes interpreted as sets of literals, the intended meaning will be clear from the context. All the formulas $ConjP_j$ in $Head(C)$ will be called P *disjuncts* and all the formulas $ConjN_j$ in $Head(C)$ will be called N *disjuncts*.

An example of IC is:

$$\begin{array}{l} order(bob, alice, camera) \rightarrow \\ (ship(alice, bob, camera), bill(alice, bob, 100)), \\ \vee \\ \forall V\ \neg bill(alice, bob, V) \end{array} \tag{3}$$

The meaning of the IC (3) is the following: if *bob* ordered a *camera* to *alice*, then *alice* must *ship* it and *bill bob* 100$, or *alice* must *not bill bob* any expense.

An IC C is true in an interpretation $M(B \cup t)$, written $M(B \cup t) \models C$, if, for every substitution θ for which $Body(C)$ is true in $M(B \cup t)$, there exists a disjunct in $Head(C)$ that is true in $M(B \cup t)$. If $M(B \cup t) \models C$ we say that the trace t is *compliant* with C; if $M(B \cup t) \not\models C$ we say that the trace t is *not compliant*. A process model H is true in an interpretation $M(B \cup t)$ if every IC of H is true in it and we write $M(B \cup t) \models H$. We also say that the trace t is *compliant* with H.

[16] showed that the truth of a range-restricted disjunctive clause of the form (1) in an interpretation I with range-restricted background knowledge B can be tested by asking the goal $? - Body(C), \neg Head(C)$. against a Prolog database containing the atoms of I as facts together with the rules of the normal program B. By $\neg Head(C)$ we mean $\neg h_1, \ldots, \neg h_m$. If the query fails C is true in I, otherwise C is false in I. Similarly to clauses, the truth of an IC C in an interpretation $M(B \cup t)$ can be tested by running the query $? - Body(C), \neg ConjP_1, \ldots, \neg ConjP_n, ConjN_1, \ldots, ConjN_m$.
against a Prolog database containing the clauses of B and the atoms of t as facts. If B is range-restricted, every answer to an atomic query Q against $B \cup t$ completely instantiates Q, i.e., it produces an element of $M(B \cup t)$. If the query finitely fails the IC is true in the interpretation. If the query succeeds, the IC is false in the interpretation. Otherwise nothing can be said.

This language extends clausal logic by allowing more complex formulas as disjuncts in the head of clauses. The ICs are more expressive than logical clauses, as can be seen from the query used to test them: for ICs we have the negation of conjunctions, while for clauses we have only the negation of atoms. This added expressiveness is necessary for dealing with processes because it allows to represent relationships between the execution times of two or more activities.

3 Learning Logical Integrity Constraints

Inductive Logic Programming (ILP) is a research field at the intersection of Machine Learning and Logic Programming. It is concerned with the development of learning algorithms that adopt logic programming for representing the input data and the induced models. The idea of exploiting ILP for *declarative* process mining comes from the similarities between learning a $\mathcal{S}$CIFF theory, composed of a set of ICs, and learning a clausal theory as described in the learning from interpretation setting of ILP [38].

In this section we introduce the technique for mining a process model as a set of Integrity Constraints, as described in the previous section. To this purpose we summarize the *Declarative Process Model Learner* (DPML) proposed in [31], which takes as input a set of process execution traces, previously labeled as compliant or not, and produces a set of ICs which correctly classifies them. In particular, DPML solves the following problem:
Given

- a space of possible process models $\mathcal{H}$

- a set I^+ of positive traces
- a set I^- of negative traces
- a normal logic program B (background knowledge)

Find: a process model $H \in \mathcal{H}$ such that

- for all $t^+ \in I^+$, $M(B \cup t^+) \models H$
- for all $t^- \in I^-$, $M(B \cup t^-) \not\models H$

H is a set of ICs C. If $M(B \cup t) \models C$ we say that C *covers* the trace t and if $M(B \cup t) \not\models C$ we say that C *rules out* the trace t. The theory (process model) composed of all the ICs must be such that all ICs are true when considering a compliant trace and at least one IC is false when considering a non compliant one.

The search space of ICs is defined by the *language bias*, that consists of a set of IC templates specifying the body literals b_i and the P/N head disjuncts allowed in the construction of ICs. A refinement operator is used to explore the search space according to the language bias and the notion of generality among ICs [31].

DPML consists of two nested loops: a covering loop (main function) and a generalization loop. In each iteration of the covering loop a new IC C is added to the process model H. C rules out some negative interpretations from the set I^-, so the loop ends when I^- is empty or when no IC is found by the generalization loop.

The IC to be added in every iteration is returned by the generalization loop that performs a beam search. The initial beam contains the most specific IC $\mathit{false} \leftarrow \mathit{true}$ (ruling out all the negative and positive traces). ICs in the beam are gradually generalized by using the refinement operator. Each generated refinement is evaluated with a heuristic function and is compared with the heuristic value of the best IC found so far: if its value is higher, the best IC is updated and this refinement is inserted in the beam. The heuristic function is the *precision* of the IC, i.e., the number of negative traces ruled out by an IC over the total number of traces (positive and negative) ruled out by it. The initial most specific clause is assigned heuristic value 0. At the end of the refinement cycle, when the beam is empty, the best IC found so far is returned to the covering loop.

DPML looks for formulas that cover as many positive traces as possible and rule out as many negative traces as possible, maximizing the *accuracy* ((true positives+true negatives)/total traces) of the final theory. See [31] for the detailed description of the language bias and the pseudo-code of the algorithm.

4 Learning Probabilistic Integrity Constraints

4.1 Markov Logic

A set of ICs can be seen as a "hard" first-order theory that constrains the set of possible worlds: if a world violates even one formula, it is considered impossible. In fact, the induced logical model classifies a trace t as compliant by testing each IC on the trace and by returning compliant only if all ICs are true in $M(B \cup t)$; if there is *at least one* IC that is false, non compliant is returned.

Markov Logic (ML) [43] is a language whose basic idea is to soften these constraints, so that when a world violates one of them it is just less probable, but not impossible. ML extends first-order logic by attaching weights to formulas. The weight associated with each formula reflects how strong the constraint is: the higher the weight, the greater the difference in probability between a world that satisfies the formula and one that does not, other things being equal. This semantics derives from the fact that, in most domains, stating non-trivial formulas that are *always* true is very difficult and these formulas capture only a fraction of the relevant knowledge. ML adds the ability to soundly handle uncertainty and tolerate contradictory knowledge. In the infinite-weight limit, ML reduces to standard first-order logic.

Definition 1 (Markov logic network) *A Markov logic network (MLN) L is a set of pairs (F_i, w_i), where F_i is a formula in first-order logic and w_i is a real number. Together with a finite set of constants $C = \{c_1, c_2, \ldots, c_m\}$, it defines a Markov network $M_{L,C}$ as follows:*

1. *$M_{L,C}$ contains one binary node for each possible grounding of each atom appearing in L. The value of the node is 1 if the ground atom is true, and 0 otherwise.*

2. *$M_{L,C}$ contains one feature (real-valued function) for each possible grounding of each formula F_i in L. The value of this feature is 1 for a possible world if the ground formula is true in it, and 0 otherwise. The weight of the feature associated to F_i is w_i.*

A *possible world* $\mathbf{x}$ is an assignment of truth values to every ground atom. The probability distribution specified by the ground Markov network $M_{L,C}$ over possible worlds $\mathbf{x}$ is given by

$$P(\mathbf{x}) = \frac{1}{Z} \exp\left(\sum_{i=1}^{F} w_i n_i(\mathbf{x})\right) \tag{4}$$

where F is the number of formulas in the MLN, $n_i(\mathbf{x})$ is the number of true groundings of F_i in $\mathbf{x}$, Z is a partition function given by $\sum_{\mathbf{x}} \exp\left(\sum_{i=1}^{F} w_i n_i(\mathbf{x})\right)$ that ensures that $P(\mathbf{x})$ sums to one.

Some assumptions are made in ML: different constants refer to different objects (unique names assumption), the only objects in the domain are those

representable using the constant and function symbols (domain closure assumption) and for each function appearing in L, the value of that function applied to every possible tuple of arguments is known and is an element of C (known functions assumption). They ensure that the number of possible worlds is finite and that the Markov logic network will give a well-defined probability distribution.

The system Alchemy[1] contains both inference and learning algorithms for MLNs.

4.2 Probabilistic Conformance Checking

Once a *logical* process model has been learned from a log by DPML, the log and the integrity constraints are transformed into ML formulas to be given as input to Alchemy.

The process traces $I^+ \cup I^-$ are converted as follows. Recall that the traces were represented as logical interpretations in an input file to DPML so that each trace was uniquely identified by its model's id (see subsection 2.2); in order to generate an equivalent log in Markov Logic, the trace delimiters `begin(model(<id>))` and `end(model(<id>))` are removed and the model's `<id>` is added as an extra constant argument to all the ground atoms representing the events, so that a sequence of events belonging to the same trace is distinguished from any other sequence of events. Moreover, a ground atom `neg(<id>)` is added if `<id>` identified a negative interpretation, in order to preserve the labeling of the original log. For instance, a negative interpretation of the type:

```
begin(model(m1)).
event1(arg1,...,argN).
event2(arg1,...,argM).
...
neg.
end(model(m1)).
```

is translated in ML syntax into:

```
event1(arg1,...,argN,m1).
event2(arg1,...,argM,m1).
neg(m1).
```

The training and test logs in this format will represent the input (`.db`) files to Alchemy.

Each IC of the form (2) is translated as follows:

$$\begin{array}{l} Body' \wedge \neg(ConjP'_1) \wedge \ldots \wedge \neg(ConjP'_n) \\ \wedge(ConjN'_1) \wedge \ldots \wedge (ConjN'_m) \rightarrow neg(T) \end{array} \quad (5)$$

where the prime symbols are obtained by adding to each atom an extra argument that is the variable T. In absence of disjuncts in the head, the IC $Body \rightarrow false$

[1] `http://alchemy.cs.washington.edu`

reduces to $Body' \rightarrow neg(T)$. The head of all formulas always contains only the atom $neg(T)$, while all disjuncts in the head are moved to the body.

For instance, the translation of IC (3) into a ML formula is:

$$order(bob, alice, camera, T) \ \wedge$$
$$\neg(ship(alice, bob, camera, T) \wedge bill(alice, bob, 100, T)) \ \wedge$$
$$bill(alice, bob, V, T) \rightarrow neg(T)$$

An MLN equivalent to the process model returned by DPML is built by declaring all the domain predicates with the types of their parameters, followed by the translation of the ICs into ML formulas according to the ML syntax[2]. This will represent the input (`.mln`) file to the learning algorithm of the Alchemy system.

Then, weights are learned for the MLN using the preconditioned rescaled conjugate gradient discriminative weight learning algorithm of [32] that is implemented in the Alchemy system. Both compliant and non compliant traces are considered in order to better tune the weights. Discriminative learning can be exploited when one knows a priori which atoms will be evidence and which ones will be queried (atoms for $neg/1$), and the goal is to correctly predict the latter given the former. After learning, a weighted MLN is returned.

Finally, given a set of unclassified process traces, i.e. a log where the ground atoms `neg(<id>)` have been removed, the weighted MLN can be used to *infer* the probability of `neg(<id>)`, that is the probability that the trace identified by the constant `<id>` is negative. To do this, we use a probabilistic inference algorithm of Alchemy that outputs the marginal probabilities of each grounding of the query predicates given the evidence. The most widely used methods for approximate inference in Markov networks are Markov Chain Monte Carlo (MCMC) or Belief Propagation; for the experiments we used one of the variations of MCMC, called MC-SAT [41].

5 Related Work

The idea of applying process mining to workflow management was proposed in [1]. The authors described an approach for inducing a process representation in the form of a directed graph encoding the precedence relationships. Starting from this, in the last two decades dozens of new process discovery techniques have been proposed, typically aiming at the induction of a conventional process model (e.g., a Petri net or EPC) that represent the allowed sequences of events as fixed and sometimes complex paths.

The α-algorithm of [51] performs workflow mining to find a WorkFlow net (WF-net), a class of Petri nets specifically tailored towards the control-flow dimension of a workflow. It examines causal relationships between tasks: for example, one specific task might always precede another specific task in every execution trace. An extension of the mining capacity of the α-algorithm to the detection of "prime invisible tasks" from an event log has been proposed

[2] `http://alchemy.cs.washington.edu/user-manual/manual.html`

with the $\alpha^{\#}$ algorithm [58], which tries to identify dependencies between tasks reflecting those used for routing purposes that exist only in a process model but are not recorded in its event log.

The Heuristics Miner [56] closely follows the α-algorithm to address its limitations and makes it much more effective in practice. It generates a Heuristic Net as a result of three steps: dependency graph mining of causal relations among the activities; for each activity, construction of the input and output expressions; mining of "long distance" dependencies among activities. An heuristic function is used to score the dependency relations found, based on the 'directly follows' relation. The Multi-phase algorithm [52] builds instance graphs for each process instance. Each instance graph is then converted into an instance Event-driven Process Chain (EPC) describing the causal relations. An EPC is an acyclic graph containing only AND-split and AND-joins connectors. The Genetic algorithm of [15] uses a genetic search method for identifying the most appropriate model out of the search space of candidate process models. Starting from a population of individuals, each one is assigned a fitness measure to indicate its quality: in this case, an individual is a possible process model and the fitness is a function that evaluates how well the individual is able to reproduce the behavior in the log. The internal representation of an individual is a causal matrix that can be mapped onto Petri nets and EPCs. Other works focusing on Petri-net representations are [57], where non-free-choice constructs are used, and [44], where Colored Petri Nets are used.

All these algorithms mine the control-flow of a log, i.e., the dependencies among its tasks, and are supported by corresponding "discovery" plug-ins in the ProM open-source framework [49][3]. This framework takes input logs in the XES or MXML format and includes plug-ins for process mining, analysis, monitoring and conversion.

Besides these approaches of a procedural nature, various proposals based on *declarative* languages have appeared. Condec [40] expresses constraints between tasks by means of Linear Temporal Logic (LTL) and a corresponding graphical representation; a set of constraint templates defines various types of dependencies for simplifying the creation of constraints. A "sister language" of Condec is DecSerFlow [46], particularly targeted to the specification of service flows. These languages are instances of DECLARE [39], a constraint-based framework which facilitates the definition of different declarative languages. In [35] the authors proposed an approach for the discovery of DECLARE models, which is made more efficient in [33]. DECLARE is characterized by templates and is based on an LTL semantics. The discovery of DECLARE models allows to specify which kinds of templates the user is interested in, in order to extract the properties that are most relevant for her. One difference between this technique and our approach is that process discovery using DPML is based on the assumption that both compliant and non-compliant traces of execution are provided, while DECLARE is applied over positive only instances; moreover, the business rules of DECLARE's discovery algorithm are expressed as LTL rules

[3] `http://www.promtools.org`.

(similarly to ConDec), while we use logical formulas (ICs). The main similarity is that both DECLARE and DPML rely on declarative languages; instead of explicitly specifying the flow of the interactions among process events, they describe a set of constraints which must be satisfied - the first through existence, relation, negative relation and choice templates, the second through a language bias indicating the atoms for building the ICs.

$\mathcal{S}$CIFF [4, 3] is a declarative language based, instead, on computational logic, which models processes with Social Integrity Constraints as forward rules of the form $Body \rightarrow Head$, where $Body$ can contain literals and happened events, and $Head$ contains a disjunction of conjunctions of expectations of events and literals. ConDec/DecSerFlow can be translated into $\mathcal{S}$CIFF and a subset of $\mathcal{S}$CIFF can be translated into ConDec/DecSerFlow [10].

In [31] the authors proposed both a language subset of $\mathcal{S}$CIFF for describing process models and the algorithm DPML for mining the integrity constraints. Examples (process traces) are true and false interpretations of a target theory, and the target theory is a set of clauses seen as constraints; DPML tries to find the best theory that discriminates the two types of interpretations. In the experiments it is compared with the new extended approach DPML+Alchemy. Other works deal with learning integrity constraints, in particular [19, 16, 26], but they are less expressive than our formalism.

Starting from an event log and an existing model (encoded as a Petri net) that is not in line with the log, the authors of [34] apply a non-monotonic ILP technique for learning minimal revisions to the model so that the revised Petri net fits the logs. While it uses positive and negative examples as DPML, it starts the search from the most general set of rules using a top-down abductive learning system.

In [7] a framework for inducing first-order logical decision trees (FOLDT) is presented. It works in the learning from interpretation setting as DPML. A FOLDT is a binary decision tree in which the nodes contain a conjunction of literals while each leaf represents a class label. Clauses can be derived from logical decision trees, since each test on the path from root to leaf is a literal or conjunction of literals that is part of the clause. The learned tree can directly be used for classification of unseen examples: if a query associated with a leaf succeeds (on the interpretation plus the background knowledge), the leaf indicates the class of the example. The Tilde system [6], included in the ACE Data Mining system, implements this framweork. We used Tilde in our experiments. It is an upgrade of the decision tree learner C4.5 [42] towards relational data mining.

Our paper extends the works [31, 30, 9] by including a probabilistic component in the process model learnt by DPML. This allows to better model domains where the relationships among events are uncertain. This is done by employing the Markov Logic representation [43] for the constraints. This representation aims at combining probability and first-order logic. The Alchemy system can perform three basic SRL tasks according to the Markov Logic: inference, weight learning and structure learning. The first involves inferring the probability or the most likely state of query atoms given a database of evidence

ground atoms. The latter two involve learning the parameters or the structure of a model given a training database consisting of logical ground atoms. In the experiments we mine models from scratch by exploiting two structure learning algorithms of MLNs. The first [28] takes a top-down approach, heuristically searching the space of models using a statistical measure of the fit of the models to the training data. The second, called BUSL, applies a bottom-up method and uses the training data to directly construct promising structural changes or additions to the model [36]. Our approach is compared with both algorithms.

Other works have considered process mining in a probabilistic context, starting from the observation that real life logs are noisy and incomplete.

In [45] the authors discuss the mining of process models in the form of AND/OR workflow graphs that are able to represent probabilistic information: each event is considered as a binary random variable that indicates whether the event happened or not and techniques from the field of Bayesian networks are used to build a probability distribution over events. The paper presents a learning algorithm that induces a model by identifying the probabilistic relationships among the events from data. It provides a probabilistic extension to traditional graph-based models, while we extend declarative modeling languages by relying on a first-order probabilistic language.

Against the assumption of working with 'complete' logs, [54] extends Petri nets with a weight function that allows to compute the probability of a transition to fire given a marking, and of a sequence of transitions (process trace) to fire. A probability distribution Π over all the possible transition matrices for traces and a probabilistic lower bound for a log to be complete are defined for three subclasses of Petri nets. This bound implies that the model discovered from the process log is an exact representation of the process that has generated that log.

In [55] the authors proposed to view a process as a distribution over traces and to view mining algorithms in terms of their ability to learn such distributions; the framework, applied to the α-algorithm, allows to compute the probability of identifying the correct process from a given log of data. The approaches [21, 54, 55] all deal with procedural model languages such as Petri nets, while we focus on logic-based declarative languages, and are more concerned with computing a probability associated with the log than a probability of the compliance of traces.

6 Experiments

The experiments have been performed over two real datasets and one artificial dataset. The real datasets regard a health care process and the careers of students at the University of Ferrara, while the artificial dataset regards an e-commerce protocol (NetBill). Statistics about the domains are reported in Table 1. In particular, the number of traces with missing values provides an indication of the log incompleteness and noise. While NetBill is complete, for the other two the number of process traces containing at least one event with at least one null argument is reported.

Table 1: Characteristics of the datasets for the experiments: number of positive traces, number of negative traces, number of predicates, of constants, of folds, number of traces with missing values.

Dataset	Pos Tr	Neg Tr	Pred	Const	Folds	Tr with missing values
NetBill	2000	2000	9	6	5	0
Screening	55	102	7	88	5	91
Students	304	472	13	899	5	430

Each logical predicate used to represent the activities in the logs has a number N of parameters which reflects the corresponding attributes of the activities stored in the databases and is indicated with "$/N$" following the predicate name.

6.1 Logs

6.1.1 Cervical Cancer Screening

This log collects data about cervical cancer, a disease in which malignant cells form in the tissues of the cervix of the uterus. Screening protocols are used to early detect and treat cervical cancer. A screening protocol is usually composed of five phases: planning, invitation, first level test with pap-test, second level test with colposcopy and possibly biopsy.

The dataset records 157 traces from the database of an Italian cervical cancer screening center [8], each corresponding to a different female patient. The traces have been analyzed by a domain expert and labeled as compliant or non compliant with respect to the protocol adopted by the center. In other words, positive traces represent patients who correctly followed the protocol and negative traces those who deviate from it. We want to determine whether the protocol is respected by new unseen traces.

Every trace is transformed into an interpretation by using the activities composing the screening process as predicates. The complete list of logical predicates is the following:

- $invitation/2$, which stores the screening center inviting the patient and the exam code;
- $refusal/1$, which stores the screening center and represents the case in which the patient decides to reject the exam;
- $exam\_execution/3$, which stores the pap-test center where the exam is executed, the exam code and type (pap-test, colposcopy or biopsy, the latter two potentially executed if pap-test gives a positive result);
- $sample\_shipping/3$, which stores the medical laboratory where the sample is sent, the exam code and type;
- $result\_posting/4$, which stores the screening center, the analysis result (positive, negative, doubtful, inadequate), the exam code and type;

- *neg_notification/3*, storing the same arguments as *exam_execution* and indicating that a *negative* result is notified to the patient;
- *pos_notification/3*, corresponding to the previous one for the case of a *positive* result.

An example of an interpretation for a patient, representing a *positive* process trace, is the following:

```
begin(model(p1)).
invitation(center1,<exam_code>).
exam_execution(laboratory1,<exam_code>,paptest).
sample_shipping(laboratory2,<exam_code>,paptest).
result_posting(center1,negative,<exam_code>,paptest).
neg_notification(center1,<exam_code>,paptest).
pos.
end(model(p1)).
```

The example describes the invitation of a patient `p1` to a pap-test, the collection of a cellular sample, the shipment of the sample to the laboratory, the communication of the result to the screening center and, finally, its notification to the patient.

Instead, an example of an interpretation for a patient, representing a *negative* process trace, is:

```
begin(model(p2)).
invitation(center1,null).
neg.
end(model(p2)).
```

The example indicates that patient `p2` did not answer the invitation to execute the test, so she has broken the protocol.

This dataset contains missing information in 91 non compliant traces for the exam code argument of the *invitation*/2 event.

6.1.2 NetBill

NetBill is a security and transaction protocol optimized for the selling and delivery of low-priced information goods, such as software or journal articles, across the Internet [13]. The protocol involves three parties - the customer, the merchant, the NetBill server - and is composed of two phases: negotiation and transaction. In the *negotiation* phase, the customer requests a price for a good from the merchant, the merchant proposes a price for it and the customer can accept the offer, refuse it or make another request to the merchant, thus initiating a new negotiation. The *transaction* phase starts if the customer accepts the offer: the merchant delivers the good to the customer encrypted with key K; the customer creates an electronic purchase order (EPO) that is countersigned by the merchant, who adds also the value of K and sends the EPO to the

NetBill server; the NetBill server checks the EPO and if the customer's account contains enough funds it transfers the amount to the merchant's account and sends a signed receipt that includes the value K to the merchant; the merchant records the receipt and forwards it to the customer (who can then decrypt her encrypted good).

The dataset collects 2000 positive traces and 2000 negative traces randomly generated as follows. The length of the negotiation phase is selected randomly between 2 and 5 and two possible values for the price quote are allowed. Next, either an accept or a refuse message is added to the trace and the transaction phase is entered with probability 4/5, otherwise the trace is closed. In the transaction phase, the messages *deliver*, *epo, epo_and_key*, *receipt* and *receipt_client* are added to the trace. With probability 1/4 a message from the whole trace is then removed. As for the screening dataset, we want to establish whether the protocol is respected on the basis of the phases executed.

Every trace is transformed into an interpretation by using the activities of the two phases as logical predicates:

- for the negotiation phase, *request*/4 and *present*/4, storing the sender (customer and merchant resp.), the receiver (merchant and customer resp.), the good, the price quote;
- for the transaction phase, *accept*/4, $refuse/4$, *deliver*/4, *epo*/4, *epo_and_key*/4, *receipt*/4, *receipt_client*/4 storing the same 4 arguments as the negotiation. In particular, the customer is the sender for the *accept*, *refuse* and EPO messages towards the merchant; the merchant is the sender for the *deliver* and *receipt_client* actions towards the customer; the NetBill server intervenes in the *epo_and_key* and *receipt* messages, where it is the receiver and the sender respectively towards the merchant.

An example of an interpretation for a transaction `n1`, representing a *positive* process trace, is the following:

```
begin(model(n1)).
request(c,m,software,10).
present(m,c,software,10).
accept(c,m,software,10).
deliver(m,c,software,10).
epo(c,m,software,10).
epo_and_key(m,s,software,10).
receipt(s,m,software,10).
receipt_client(m,c,software,10).
pos.
end(model(n1)).
```

The example describes a *request* from the customer c to the merchant m for the good $software$ at price 10, and the merchant's answer with the same price (*present* message). This exchange constitutes the negotiation phase. The subsequent transaction phase follows the description given above, where the NetBill server is indicated as s.

An example of an interpretation for a transaction `n2`, representing a *negative* process trace, is:

```
begin(model(n2)).
request(c,m,software,10).
present(m,c,software,10).
accept(c,m,software,10).
neg.
end(model(n2)).
```

This trace is closed after the *accept* message: the transaction phase is missing so the protocol is not respected.

6.1.3 Students' careers

This dataset collects data about the careers of students enrolled at the Faculty of Engineering of the University of Ferrara from 2004 to 2009. Each career records the main events such as all the chronological enrollments, the exams taken and the career conclusion (degree or not). The dataset records 776 traces each corresponding to a different student career. The careers of students who graduated are positive traces (compliant) while those who did not finish their studies are negative traces. We want to determine whether a student graduates on the basis of her complete career.

Every trace is transformed into an interpretation by using the activities of the university career as predicates. The complete list of logical predicates is the following:

- *student_info*/*6*, which stores some personal and high school information about a student, such as type, final mark and year of high school diploma (h.s.d.), town and country of residence, the student's identification number;

- *registration*/2, which stores the first enrollment (from high school) with attributes academic year and course year (always 1);

- *enrollment*N/3, with $N = 1 \ldots 9$, which stores the enrollments for the years following the first of registration (for $N = 2 \ldots 9$) or the first enrollment (for $N = 1$) if the student had previously registered to another faculty/university, with parameters academic year, course year (1,2,3), student's status (regular or not);

- *exam*/4, with parameters: exam code, mark (18-30), honours (yes/no) and mark category (low, medium, high);

- *career_end*/1, which stores the career conclusion as degree (positive traces) or drop-out, not-renewed enrollment, transfer to another faculty, transfer to another University (negative traces).

An example of an interpretation for a student's career, representing a *positive* process trace, is the following:

```
begin(model(s1)).
student_info(scientific h.s.d.,low,2003,rovigo,italy,117230399).
registration(2005, 1).
exam(015092, 30, no, high).
...<other exams>...
enrollment2(2006, 2, regular).
exam(006435, 25, no, medium).
...<other exams>...
enrollment3(2007, 3, regular).
exam(015606, 30, yes, high).
...<other exams>...
career_end(degree).
pos.
end(model(s1)).
```

The example describes an Italian student with identification number '117230399' who got a scientific high school diploma in 2003 with a low final mark; he enrolled at the faculty of Engineering in 2005 for the first time and then for other two years as a regular student until the degree. Enrollments alternate with all the exams.

Instead, an example of an interpretation for a student representing a *negative* process trace, is:

```
begin(model(s2)).
student_info(technical h.s.d.,low,2006,bologna,italy,117229399).
registration(2005, 1).
exam(015092, 25, no, medium).
enrollment2(2006, 2, regular).
exam(000470, 18, no, low).
career_end(transfer).
neg.
end(model(s2)).
```

The example indicates that the student only completed the first year, enrolled in the second one passing only two exams overall and finally transferred to another faculty/university.

This dataset contains missing information in 439 *exam* events for the mark and mark category arguments, distributed among 426 traces, and in 10 *student_info* events for the high school category argument.

6.2 Methodology

The proposed approach for the probabilistic classification of process traces is referred as DPML+Alchemy. It has been compared with (1) DPML, (2) α-algorithm, $\alpha^{\#}$-algorithm, Multi-phase Macro plug-in, Genetic algorithm and Heuristics miner available in the ProM suite [49] (3) Tilde, included in the ACE Data Mining System [6], (4) two structure learning algorithms of Markov Logic Networks, the one embedded in the Alchemy package[4] and BUSL[5].

6.2.1 Process Mining Settings

Five experiments have been performed on all the logs with each of the algorithms.

For the screening process, five-fold cross validation was used, i.e., the dataset was divided into 5 sets and in each experiment 4 were used for training (process model mining) and the remaining for testing (conformance checking). In particular, each fold contains either 11 positive and 21 negative traces or 11 positive and 20 negative traces. For NetBill, the training and testing sets were generated with the procedure sketched in subsection 6.1.2 with different seeds for the random function for each experiment; all sets contain 2000 positive and 2000 negative traces. For the students' careers, five-fold cross validation was used, with each fold containing 60 or 61 positive and 94 or 95 negative traces.

DPML DPML derives several settings from ICL [19]. The maximum number of literals in the head and in the body of a rule was set to 8; the minimal accuracy (number of positive traces correctly classified plus the number of negative traces correctly classified divided by the total number of traces) for each individual rule was set to 0.75; the minimum coverage (the minimum number of negative examples that a clause must rule out) was kept to the default value 1; the maximum number of rules to be kept in the beam (beam size) was set to 10; the number of clause refinement iterations was set to 50.

ProM mining plug-ins The ProM mining plug-ins were applied with their default settings. In the training phase, since these algorithms take as input a single set of traces, we provided them with the compliant traces only. The input log is in MXML format[6].

Tilde We used the most recent version of Tilde (3.0) included in the ACE System 1.2.20.

Among the general settings which apply to all systems available in ACE, we set:

[4] `http://alchemy.cs.washington.edu/`

[5] `http://www.cs.utexas.edu/~ml/mlns/`

[6] `http://www.processmining.org/WorkflowLog.xsd`

- `classes([pos,neg])` to define the classes to be used for the classification task (which is binary in our case);
- `load(models)`, which tells ACE that data are in *models* format (the same format used for the input traces to DPML, cf. subsections 6.1.1, 6.1.2, 6.1.3);

Among the settings specific for TILDE we set:

- `sampling_strategy(none)`, to use the whole training dataset when deciding which test should be put in a node of the tree;
- `minimal_cases(2)` for screening/NetBill and `minimal_cases(10)` for students, the minimal number of examples that a leaf of the tree should cover;
- `output_options([c45,prolog,roc01,likelihood])`, to visualize, in addition to the default tree in a C4.5-like output format and the corresponding Prolog program, the area under the ROC curve (AUCROC) and the log-likelihood (LL) over the test set.

Alchemy and BUSL As regards *weight learning* on the ICs theories induced by DPML, the `learnwts` executable available in Alchemy was used. Learning may be generative or discriminative, according to whether the aim is to accurately predict all or a specific predicate respectively. We applied discriminative learning [32] with the following options: `-i <input .mln file> -o <output .mln file with weights> -t <training .db file> -ne Neg`, with the `-ne` option specifying the non-evidence predicate *neg*.

As regards *structure learning* of MLNs from the datasets, we applied:

1. the `learnstruct` executable available in Alchemy with the same specified parameters as `learnwts`, plus the `-startFromEmptyMLN` option to start structure learning from an empty MLN;
2. BUSL (`busl` executable) both with the same parameters as `learnstruct` and with additional parameters. In all cases, it was not able to terminate on all datasets in 72h, since it didn't go beyond the construction of conjunctions of literals that serve for creating clauses.

6.2.2 Conformance Testing

ProM mining plug-ins Compliance of the testing traces with respect to the learned models (Petri nets or EPCs) has been evaluated by using the *Conformance checker* ProM plug-in for the algorithms only available in ProM5.2 ($\alpha^{\#}$, Multi-phase Macro) and the *Replay a log on petri net for conformance analysis* [48] for the other plug-ins available in ProM6.3, by providing them with the positive and the negative test traces in two consecutive stages. In this manner, by considering the fitting and non-fitting traces to the process model, it is possible to derive the four entries of the confusion matrix and, therefore, the average accuracy as (true positives+true negatives)/total traces.

Tilde We performed the training and testing phases jointly, by exploiting the `leave_one_out_from_list(tilde,<list>)` command in each of the five experiments. This command allows to provide TILDE with the training and testing process traces together, since the latter are identified by the additional predicate *testid*(c), whose constant c has to be specified in the `<list>` argument of the command. From the specified output settings (cf. subsection 6.2.1) we obtained directly the average accuracy and log-likelihood on the test log, and the AUCROC. From the ROC points available in output we computed the corresponding PR points and the AUCPR.

DPML, DPML+Alchemy, MLNs structure learning Probabilistic inference from the weighted models learned by DPML+ALCHEMY and `learnstruct` was performed with the `infer` executable by specifying the MC-SAT algorithm (option `-ms`), the weighted MLN as input model (option `-i`), the test set (`.db` file with the option `-e`) and `Neg` as query atom (option `-q`). We used MC-SAT because it greatly outperforms Gibbs sampling and simulated tempering when deterministic dependencies (formulas) are present [41]. This is particularly useful for performing inference from a "hard" theory such as that learned by DPML alone. In fact, to evaluate DPML through the same inference procedure as DPML+ALCHEMY, we translated the learned ICs theories into MLNs whose formulas are terminated by a period [29], to mean that they are "hard" constraints, or equivalently, have infinite weight. An infinite-weight MLN corresponds to a first-order theory. In this manner we could handle a purely logical theory with Alchemy's tools.

With MC-SAT we compute the marginal probabilities of being compliant for each test trace representing a patient for the screening log, a transaction for the NetBill log or a student at the University of Ferrara for the students' careers log. From the probability estimates we computed the average accuracy and the average Area Under the Precision Recall curve and ROC curve (AUCPR and AUCROC respectively) using the methods reported in [14, 22]. Accuracy is computed as the average of the greatest accuracies over the folds. As in [22], which generates a stack of ROC points from the probabilistic estimates for the traces, we create a list of accuracy values every time TP and FP are updated and at the end we pick the greatest value from this list.

Tables 2, 3 and 4 show the accuracies, AUCPR and AUCROC, log-likelihood (LL) over the test set averaged over the folds for all algorithms and datasets. Table 5 shows the average execution times for all algorithms except the ProM plug-ins, since the time is always less than 1 minute. Tables 6, 7 and 8 show the p-value of a paired two-tailed t-test at the 5% significance level of the difference in: (1) AUCPR and AUCROC between DPML+ALCHEMY and DPML/TILDE/`learnstruct`, (2) average accuracy between DPML+ALCHEMY and all the other algorithms (significant differences in favor of DPML+ALCHEMY in bold).

6.2.3 Results on Cervical Cancer Screening

ProM mining plug-ins All the algorithms are able to accurately capture the screening protocol, as confirmed by high values of accuracy in Table 2. As an example, we show the net learned by one of the algorithms with the highest accuracy, the Heuristics miner, in Fig. 1.

DPML For inducing the process model we specified a language bias which allowed all the predicates listed in subsection 6.1.1 as body literals and as head disjuncts:

- one atom for every value of the exam type parameter for the *exam_execution*, *sample_shipping*, *neg_notification*, *pos_notification* predicates;
- one atom for every combination of values of *result* and *exam type* for the *result_posting* predicate;
- unground atoms for the *invitation* and $refusal$ predicates (we are interested only in the events themselves).

The five ICs theories learned contain 3 or 4 rules. An example from the first fold is:

```
true → ∃A,B exam_execution(A,B,paptest) ∨ ∃C refusal(C).
result_posting(A,positive,B,paptest) →
                                    ∃C,D exam_execution(C,D,colposcopy).
result_posting(A,doubtful,B,colposcopy) →
                                    ∃C,D exam_execution(C,D,biopsy).
```

The first IC states that there must be a pap-test execution or a refusal; the second one that if there is a positive pap test then there must be also a colposcopy; the third one that if there is a doubtful colposcopy then there must be also a biopsy.

Tilde The logical decision trees induced by TILDE in the five experiments show low complexity, ranging from 1 to 4 nodes; the equivalent Prolog programs range from 2 to 5 clauses, similarly to the DPML theories. An example of Prolog program equivalent to the tree learned from the first fold is:

```
class([pos]) :- exam_execution(A,B,C),
                neg_notification(D,E,biopsy),
                pos_notification(F,G),!.
class([neg]) :- exam_execution(A,B,C),
                neg_notification(D,E,biopsy),!.
class([pos]) :- exam_execution(A,B,C),!.
class([neg]).
```

The predicates used differ from those of DPML, such as `neg_notification` and `pos_notification`.

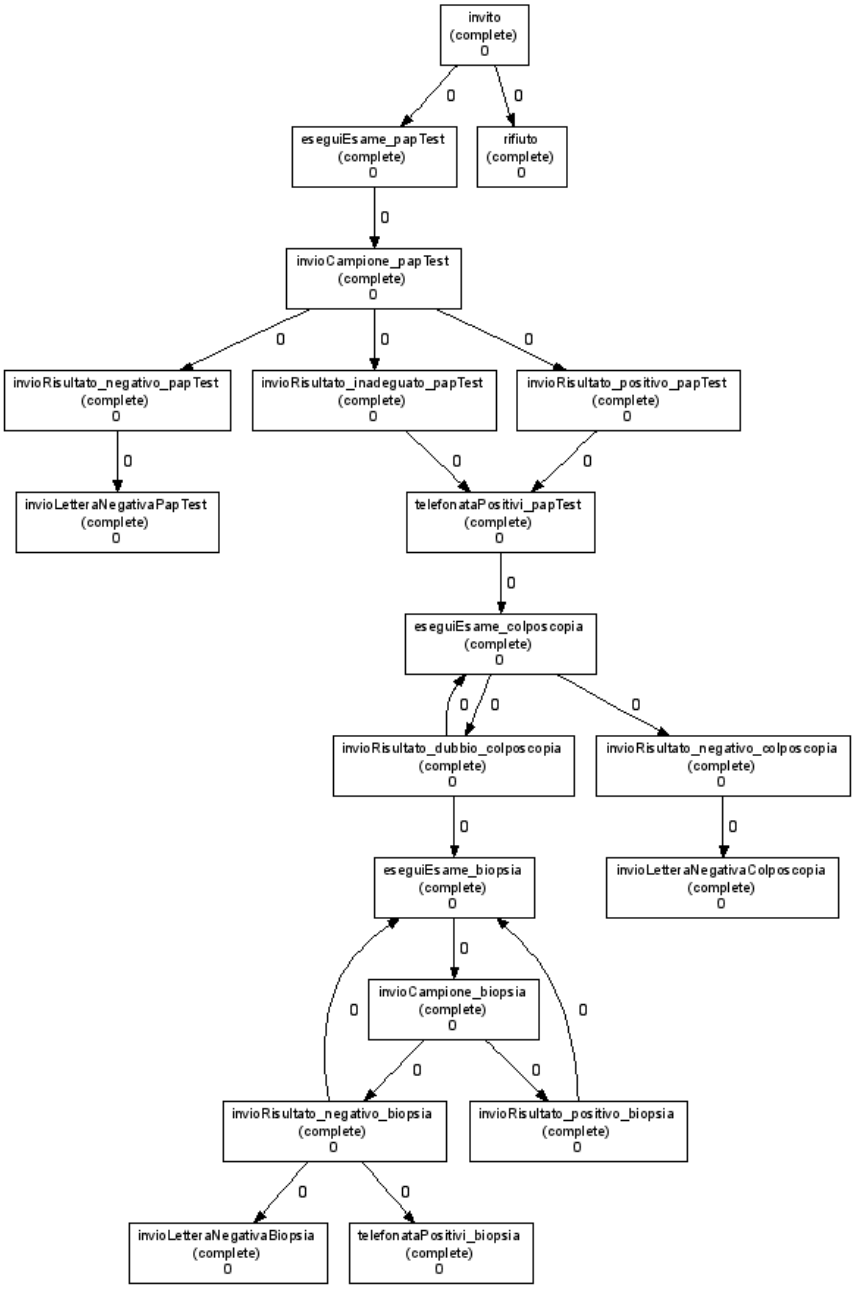

Figure 1: Heuristic net learned by the Heuristics miner on the first fold of the cervical cancer screening log.

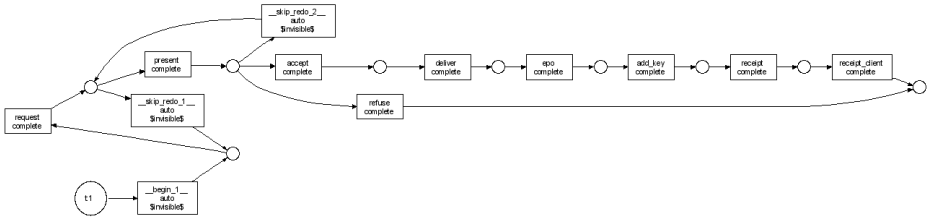

Figure 2: Petri net learned by the $\alpha^{\#}$ miner on the first fold of the NetBill log.

DPML+Alchemy By applying weight learning with Alchemy on the DPML model we get (T is the variable identifying a trace):

```
0.00157 ¬exam_execution(A,B,papTest,T) ∧ ¬refusal(C,T)
        → neg(T)
0.00127 result_posting(A,positive,B,papTest,T) ∧
        ¬exam_execution(C,D,colposcopy,T) → neg(T)
0.00005 result_posting(A,doubtful,B,colposcopy,T) ∧
       ¬exam_execution(C,D,biopsy,T) → neg(T)
```

MLNs structure learning We were able to learn MLNs with the `learnstruct` executable containing 2 or 3 formulas, while BUSL was not able to terminate. An example of MLN learned from the first fold is:

```
3.7824    exam_execution(A,B,C,T) ∧
          neg_notification(D,B,paptest,T) → ¬neg(T)
```

6.2.4 Results on NetBill

ProM mining plug-ins Several algorithms are not able to correctly model the negotiation phase in the sequence of the *request* and *present* events: *present* is the merchant's proposal of a price for the good previously *requested* by the customer. There may occur various combinations of these two events: multiple requests of goods followed by only one or by more proposals, or multiple requests alternated with the corresponding proposals. In particular, the α-algorithm does not link the two activities, the Heuristics miner does not capture the possible alternation of the activities, the Genetic algorithm the possible repetition of the requests, the Multi-phase macro considers them as possible parallel starting activities. Only the $\alpha^{\#}$-algorithm correctly models the process, as confirmed by its much higher accuracy than the others (cf. Table 3). As an example, we show the Petri net learned by this algorithm in Fig. 2.

DPML For inducing the process model by means of DPML, we specified a language bias which allowed all the predicates of the domain both as body literals and as head disjuncts, for every value of the price quote parameter.

The five ICs theories learned contain 10 to 13 rules. An extract from the first fold is:

```
request(A,B,C,D) → ∃E,F,G,H present(E,F,G,H).
present(A,B,C,20) → ∃D,E,F,G ¬accept(D,E,F,G).
accept(A,B,C,D) → ∃E,F,G,H deliver(E,F,G,H).
epo_and_key(A,B,C,D) → ∃E,F,G,H accept(E,F,G,H).
```

The first IC states that if there is a request (from the customer) then there must be an offer (from the merchant); the second one that if there is an offer at price 20, the offer is not accepted; the third one that if the customer has accepted the offer then the good will be delivered; the fourth one that if there is an EPO with key message, the offer must have been accepted.

Tilde The logical decision trees induced by TILDE in the five experiments all have 10 nodes and the equivalent Prolog programs 11 clauses, similarly to DPML. The initial equivalent Prolog program from the first fold is:

```
class([pos]) :- present(A,B,C,D),request(E,F,G,H),receipt(I,J,K,L),
accept(M,N,O,P),receipt_client(Q,R,S,T),deliver(U,V,W,X),epo(Y,Z,A1,B1),
epo_and_key(C1,D1,E1,F1), !.
class([neg]) :- present(A,B,C,D),request(E,F,G,H),receipt(I,J,K,L),
accept(M,N,O,P),receipt_client(Q,R,S,T),deliver(U,V,W,X),
epo(Y,Z,A1,B1),!.
class([neg]) :- present(A,B,C,D),request(E,F,G,H),receipt(I,J,K,L),
accept(M,N,O,P),receipt_client(Q,R,S,T),deliver(U,V,W,X), !.
....
```

As can be seen, the number of body literals in the clauses is much greater than that of DPML clauses in this dataset.

DPML+Alchemy By applying weight learning with Alchemy on the DPML model we get for the first 4 rules (T is the variable identifying a trace):

```
0.168 request(A,B,C,D,T) ∧ ¬present(E,F,G,H,T) → neg(T)
0 present(A,B,C,20,T) ∧ accept(D,E,F,G,T) → neg(T)
-1.316 accept(A,B,C,D,T) ∧ ¬deliver(E,F,G,H,T) → neg(T)
2.683 epo_and_key(A,B,C,D,T) ∧ ¬accept(E,F,G,H,T) → neg(T)
```

MLNs structure learning We were able to learn MLNs, with the `learnstruct` executable, containing 1-2 formulas, which however in three folds out of five are not significant since only the single fact `neg` is present. BUSL did not terminate.
The most significant MLN learned by `learnstruct` is:

```
7.02366    ¬accept(A,B,C,D,T) ∧ deliver(B,A,C,D,T) → neg(T)
```

6.2.5 Results on Students' Careers

ProM mining plug-ins None of the algorithms is able to correctly capture the alternation among the *registration*, *enrollmentN* and *exam* events: multiple exams are taken after the registration and each enrollment. The models learned either allow the repetition of the exam event but at the end of the chain of enrollments before the career end ($\alpha^{\#}$ and Heuristics miners) or are not able to represent correctly the order of enrollments (Genetic miner) or the exam event is in parallel with the chain of the remaining events (Multi-phase Macro).

DPML For inducing the process model we specified a language bias which allowed the following two templates:

- one atom for every value of the honors parameter for the *exam* predicate as body literal; unground atoms for the *registration* and *enrollment*1 predicates as head disjunct;
- one atom for every value of the academic year parameter for the *registration* predicate as body literal; one atom for every value of academic year, course year, student's status and for every value of the couple course year+student's status for the *enrollmentN* predicate as body literal; one atom for every value of town and country of residence for the *student_info* predicate as head disjunct.

The five ICs theories learned contain 21-23 rules. An example from the first fold is:

```
true → ∃A ¬enrollment8(A,3,non-regular).
student_info(A,B,vicenza,D,E,I) → false.
true → ∃A ¬registration(2005,A) ∨ ∃B,C ¬enrollment4(B,C,non-regular).
```

The first IC states that students at the eighth enrollment for the third year as non-regular students didn't graduate; the second one that students living in the province of Vicenza didn't graduate; the third one that careers with a registration in 2005 and a fourth enrollment as a non-regular student were not successful.

Tilde The logical decision trees induced by TILDE over the five folds are constituted by 2 or 3 nodes and an equivalent Prolog program is for example:

```
class([neg]) :- registration(2007,A),!.
class([neg]) :- enrollment6(2009,A,B),!.
class([pos]) :- enrollment3(A,3,B),!.
class([neg]).
```

As can be seen, DPML is able to learn more and more complex rules.

Table 2: Results of the experiments in terms of *Area Under the PR and ROC Curves*, *accuracy* and *log-likelihood (LL)* over the test set, averaged over the folds, for the *Cervical Cancer Screening* dataset. Enclosed by dashed lines the ProM plug-ins. '-' for BUSL means the algorithm did not terminate, for the other systems means the measure is not available. In bold the highest value in each column.

System	AUCROC	AUCPR	Accuracy	LL
α-algorithm	-	-	**0.961**	-
$\alpha^{\#}$-algorithm	-	-	0.949	-
Multi-phase macro	-	-	0.949	-
Genetic algorithm	-	-	**0.961**	-
Heuristics miner	-	-	**0.961**	-
DPML	0.237	0.338	0.65	-227.311
TILDE	0.778	**0.869**	0.910	-7.919
DPML+Alchemy	**0.949**	0.824	0.949	-27.842
`learnstruct`	0.922	0.817	0.936	-61.252
BUSL	-	-	-	

DPML+Alchemy By applying weight learning with Alchemy on the previous DPML model we obtained:

```
3.4202  enrollment8(A,3,non-regular,T) → neg(T)
2.0626  student_info(A,B,vicenza,D,E,I,T) → neg(T)
1.7450  registration(2005,A,T) ∧ enrollment4(B,C,non-regular,T) →neg(T)
```

MLNs structure learning Neither `learnstruct` nor BUSL were able to terminate due to a memory error during the execution.

6.2.6 Comments

The results in Tables 2, 3 and 4 show that DPML+ALCHEMY obtains the highest AUCROC on the screening log and the highest AUCROC/ AUCPR/accuracy on the Netbill and Students logs. For the screening log, the difference in AUCPR between DPML+ALCHEMY and TILDE, the algorithm getting the highest value, is not statistically significant, as well as the accuracy differences between DPML+ALCHEMY and the ProM plug-ins. For the other logs, all the differences are statistically significant in favor of DPML+ALCHEMY except two or three. On the whole, the differences between DPML+ALCHEMY and the other systems are statistically significant in its favor in 24 out of 39 cases at the 5% significance level.

The ILP learning algorithm DPML is able to learn more significant clauses with respect to the Markov Logic structure learning algorithm `learnstruct`,

Table 3: Results of the experiments in terms of *Area Under the PR and ROC Curves*, *accuracy* and *log-likelihood (LL)* over the test set, averaged over the folds, for the *NetBill* dataset. Enclosed by dashed lines the ProM plug-ins. '-' for BUSL means the algorithm did not terminate, for the other systems means the measure is not available. In bold the highest value in each column.

System	AUCROC	AUCPR	Accuracy	LL
$\alpha-$algorithm	-	-	0.668	-
$\alpha^{\#}$-algorithm	-	-	0.909	-
Multi-phase macro	-	-	0.605	-
Genetic algorithm	-	-	0.5	-
Heuristics miner	-	-	0.5	-
DPML	0.5	0.5	0.5	-39613.950
Tilde	0.873	0.836	0.846	-2010.395
DPML+Alchemy	**0.929**	**0.866**	**0.913**	-10214.018
`learnstruct`	0.634	0.599	0.636	-5662.788
BUSL	-	-	-	

Table 4: Results of the experiments in terms of *Area Under the PR and ROC Curves*, *accuracy* and *log-likelihood (LL)* over the test set, averaged over the folds, for the *Students' Careers* dataset. Enclosed by dashed lines the ProM plug-ins. '-' means the algorithm did not terminate, for the other systems means the measure is not available. In bold the highest value in each column.

System	AUCROC	AUCPR	Accuracy	LL
$\alpha-$algorithm	-	-	0.607	-
$\alpha^{\#}$-algorithm	-	-	0.607	-
Multi-phase macro	-	-	0.607	-
Genetic algorithm	-	-	0.677	-
Heuristics miner	-	-	0.607	-
DPML	0.5	0.392	0.608	-1537.021
Tilde	0.701	0.594	0.718	-inf
DPML+Alchemy	**0.791**	**0.639**	**0.732**	-215.100
`learnstruct`	-	-	-	
BUSL	-	-	-	

Table 5: *Execution time* in minutes of the experiments; each value is the average over the folds. The single steps for each system (mining+inference on the model) are shown, together with the average total time.

System	Screening	NetBill	Students
DPML	2.30	539.66	4.38
Inference	0.03	1.57	0.53
Total	*2.33*	*541.23*	*4.91*
DPML	2.30	539.66	4.38
Weight learning	2.69	2.13	0.02
Inference	0.06	1.63	0.0048
Total **DPML+Alch**	*5.05*	*543.42*	*4.405*
TILDE	*2e-4*	*0.012*	*0.0039*
`learnstruct`	2.290	1.77	-
Inference	0.001	0.60	-
Total	*2.291*	*2.37*	-
BUSL	-	-	-

Table 6: Results of *t-test* relative to AUCPR, AUCROC and accuracy for the *Cervical Cancer Screening* dataset. p is the p-value of a paired two-tailed t-test between DPML+ALCHEMY and the other systems (significant differences in favor of DPML+ALCHEMY at the 5% level in bold). D+A is DPML+ALCHEMY, lstruct is `learnstruct`.

System Couple	AUCROC	AUCPR	Accuracy
D+A-DPML	**0.001**	**0.002**	**5.26e-5**
D+A-lstruct	0.296	0.799	0.46
D+A-Tilde	0.977	0.433	**0.033**
D+A-α-alg.	-	-	0.59
D+A-$\alpha^{\#}$-alg.	-	-	0.99
D+A-Multi Phase	-	-	0.99
D+A-Genetic	-	-	0.59
D+A-Heuristics	-	-	0.59

Table 7: Results of *t-test* relative to AUCPR, AUCROC and accuracy for the *NetBill* dataset. p is the p-value of a paired two-tailed t-test between DPML+Alchemy and the other systems (significant differences in favor of DPML+Alchemy at the 5% level in bold). D+A is DPML+Alchemy, lstruct is `learnstruct`.

System Couple	AUCROC	AUCPR	Accuracy
D+A-DPML	**4.60e-6**	**5.72e-5**	**0.036**
D+A-lstruct	**0.014**	**0.006**	**0.017**
D+A-Tilde	**0.011**	0.179	**0.004**
D+A-α-alg.	-	-	**3.35e-5**
D+A-$\alpha^{\#}$-alg.	-	-	0.8
D+A-Multi Phase	-	-	**1.28e-4**
D+A-Genetic	-	-	**3.61e-6**
D+A-Heuristics	-	-	**3.61e-6**

Table 8: Results of *t-test* relative to AUCPR, AUCROC and accuracy for the *Students' Careers* dataset. p is the p-value of a paired two-tailed t-test between DPML+Alchemy and the other systems (significant differences in favor of DPML+Alchemy at the 5% level in bold). D+A is DPML+Alchemy.

System Couple	AUCROC	AUCPR	Accuracy
D+A-DPML	**5.91e-6**	**9.10e-5**	**3.19e-4**
D+A-Tilde	**0.044**	0.184	0.576
D+A-α-alg.	-	-	**3.20e-4**
D+A-$\alpha^{\#}$-alg.	-	-	**3.20e-4**
D+A-Multi Phase	-	-	**3.20e-4**
D+A-Genetic	-	-	0.349
D+A-Heuristics	-	-	**3.20e-4**

especially for NetBill where the latter performs poorly. This results in DPML+Alchemy performing better in almost all cases. The other ML-based algorithm, BUSL, is never able to complete structure learning.

Among the ILP-based learning algorithms, Tilde performs better than the logical approach DPML, but worse than the probabilistic one DPML+Alchemy on the NetBill and students logs, with a statistically significant difference in 3 cases out of 6; on the screening log Tilde performs better in AUCPR but the difference is not statistically significant.

The ProM plug-ins show equal or greater accuracy on the screening process, but quite low on the NetBill and Students processes (cf. Sections 6.2.4 and 6.2.5), where they do not completely capture the repetition of couples of alternated activities, while DPML is able to find constraints on these kinds of events.

Table 5 shows that DPML+Alchemy requires longer times due to its different phases, in particular on the NetBill process where the mining of the model by DPML takes several hours. This log has a much greater size than the Cervical Cancer Screening log both in the number of traces and in the number of events per trace, especially for the negative traces: in the screening log most negative traces are such because patients did not answer the invitation to pap test, which therefore is the only recorded event, while in the NetBill log negative traces may have more events than the positive ones, due to the random generation of events used to build this dataset. DPML and DPML+Alchemy have comparable times on the Students and Screening logs in spite of the larger number of traces and constants of the former, since the language bias templates only specify partial instantiation of the atoms for the ICs (according to which arguments we are interested in, cf. Section 6.2.5).

We demonstrated that a probabilistic approach at conformance checking achieves better results than a sharp approach realized by purely logical or procedural process mining techniques. In this paper we have considered classification problems with two classes (binary decision problems). A discrete classifier - the ProM plug-ins, DPML and Tilde - outputs only a class decision, i.e., a Yes or No on each instance, representing the predicted positive or negative class. A probabilistic classifier - the approaches using inference in Markov Logic - instead yields a probability, a numeric value that represents the degree to which an instance is a member of the positive class. DPML+Alchemy, in particular, outputs the probability that each test set example is *negative* so that one can sort the examples in ascending order from the least negative ones (which are more likely to belong to the positive class - to be compliant to the model) to the most negative ones (which are more likely to belong to the negative class - to be non compliant). These probabilities give an indication of how likely it is that the class label *neg* applies.

In addition, DPML+Alchemy better performance w.r.t. SRL structure learning algorithms (learnstruct and BUSL) indicates that the combination of ILP and SRL techniques is a key point of the proposed approach.

7 Conclusions

We proposed a methodology, based on Statistical Relational Learning, for analyzing a log containing process traces labeled as compliant or non-compliant. From them we learn a set of declarative logical constraints with the DPML algorithm of [31] and subsequently we represent them in Markov Logic, a probabilistic logical language, in order to perform probabilistic conformance checking of new traces. In this way we get a ranking from the less compliant to the most compliant traces with respect to the induced model. We evaluated the performance of this technique (DPML+Alchemy) over three processes: an e-commerce protocol, a cervical cancer screening protocol and a log of university students' careers, and compared it with DPML alone, five procedural process mining algorithms of the ProM framework and Tilde, in terms of accuracy, areas under the PR and ROC curves and execution time. The probabilistic classifications made by DPML+Alchemy resulted more accurate than those performed by the other non probabilistic systems.

References

[1] R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT 1998)*, pages 469–483, Valencia, Spain, 23-27 March 1998. Springer, Berlin Heidelberg.

[2] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. On the automatic verification of interaction protocols using *g-*$\mathcal{S}$CIFF. Technical Report DEIS-LIA-04-004, LIA Series no. 72, DEIS-University of Bologna, Bologna, Italy, 2005.

[3] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Verifiable agent interaction in abductive logic programming: The SCIFF framework. *ACM Transactions on Computational Logic*, 9(4):1–43, 2008.

[4] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. An abductive interpretation for open societies. In *Proceedings of the 8th Congress of the Italian Association for Artificial Intelligence (AI*IA 2003)*, pages 287–299, Pisa, Italy, 23-26 September 2003. Springer-Verlag, Berlin.

[5] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Specification and verification of agent interaction using social integrity constraints. *Electronic Notes in Theoretical Computer Science*, 85(2):94 – 116, 2004.

[6] H. Blockeel, L. Dehaspe, J. Ramon, J. Struyf, A. van Assche, C. Vens, and D. Fierens. The ACE Datamining System, User's Manual. `https://dtai.cs.kuleuven.be/ACE/doc/ACEuser-1.2.16.pdf`.

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

[8] Cervical cancer screening web site. `http://www.cancer.gov/cancertopics/pdq/screening/cervical/healthprofessional`.

[9] F. Chesani, E. Lamma, P. Mello, M. Montali, F. Riguzzi, and S. Storari. Exploiting inductive logic programming techniques for declarative process mining. *LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC)*, 5460:278–295, 2009.

[10] F. Chesani, P. Mello, M. Montali, and S. Storari. Towards a DecSerFlow declarative semantics based on computational logic. Technical Report DEIS-LIA-07-002, LIA Series no. 79, DEIS-University of Bologna, Bologna, Italy, 2007.

[11] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*. Plenum Press, New York, USA, 1978.

[12] V. S. Costa, D. Page, M. Qazi, and J. Cussens. CLP(BN): Constraint logic programming for probabilistic knowledge. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI 2003)*, pages 517–524, Acapulco, Mexico, 7-10 August 2003. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[13] B. Cox, J.C. Tygar, and M. Sirbu. NetBill security and transaction protocol. In *Proceedings of the First USENIX Workshop on Electronic Commerce (WOEC 1995)*, pages 77–88, New York, NY, USA, 11-12 July 1995. USENIX Association, Berkeley, CA, USA.

[14] J. Davis and M. Goadrich. The relationship between Precision-Recall and ROC curves. In *Machine Learning, Proceedings of the 23rd International Conference (ICML 2006)*, pages 233–240, Pittsburgh, Pennsylvania, USA, 25-29 June 2006. ACM, New York, NY, USA.

[15] A. K. A. de Medeiros, A. J. M. M. Weijters, and W. M. P. van der Aalst. Genetic process mining: an experimental evaluation. *Data Mining and Knowledge Discovery*, 14:245–304, 2007.

[16] L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26(2-3):99–146, 1997.

[17] L. De Raedt, P. Frasconi, K. Kersting, and S. Muggleton, editors. *Probabilistic Inductive Logic Programming - Theory and Applications*. Springer, Berlin Heidelberg, 2008.

[18] L. De Raedt, A. Kimmig, and H. Toivonen. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 2462–2467, Hyderabad, India, 6-12 January 2007. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[19] L. De Raedt and W. van Laer. Inductive constraint logic. In *Proceedings of the 6th Conference on Algorithmic Learning Theory (ALT 1995)*, pages 80–94, Fukuoka, Japan, 18-20 October 1995. Springer, Berlin Heidelberg.

[20] P. Domingos, S. Kok, D. Lowd, H. Poon, M. Richardson, and P. Singla. Markov logic. In Luc De Raedt, Paolo Frasconi, Kristian Kersting, and Stephen Muggleton, editors, *Probabilistic Inductive Logic Programming*. Springer, Berlin Heidelberg, 2008.

[21] R. Z. Farkhady and S. H. Aali. A probabilistic approach for process mining in incomplete and noisy logs. In *Proceedings of the International MultiConference on Engineers and Computer Scientists (IMECS 2011)*, volume 1, pages 415–421, Hong Kong, China, 16-18 March 2011. Newswood Limited, Hong Kong.

[22] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters - Special issue: ROC analysis in pattern recognition*, 27(8):861–874, 2006.

[23] D. Georgakopoulos, M. F. Hornick, and A. P. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.

[24] L. Getoor and B. Taskar, editors. *Introduction to Statistical Relational Learning*. MIT Press, Cambridge, MA, USA, 2007.

[25] G. Greco, A. Guzzo, L. Pontieri, and D. Saccà. Discovering expressive process models by clustering log traces. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1010–1027, 2006.

[26] A. Jorge and P. Brazdil. Integrity Constraints in ILP using a Monte Carlo approach. In *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 229–244, Stockholm, Sweden, 26-28 August 1996. Springer, Berlin Heidelberg.

[27] K. Kersting and L. De Raedt. Towards combining inductive logic programming with bayesian networks. In *Proceedings of the 11th International Conference on Inductive Logic Programming (ILP 2001)*, pages 118–131, Windsor Great Park, United Kingdom, 31st July - 3rd August 2001. Springer, Berling Heidelberg.

[28] S. Kok and P. Domingos. Learning the structure of markov logic networks. In *Proceedings of the International Conference on Machine Learning and Applications (ICMLA 2005)*, pages 441–448, Los Angeles, California, 15-17 December 2005. ACM, New York, NY, USA.

[29] S. Kok, P. Singla, M. Richardson, P. Domingos, M. Sumner, H. Poon, D. Lowd, J. Wang, and A. Nath. The Alchemy System for Statistical Relational AI: User Manual. `http://alchemy.cs.washington.edu/user-manual/manual.html`.

[30] E. Lamma, P. Mello, M. Montali, F. Riguzzi, and S. Storari. Inducing declarative logic-based models from labeled traces. In *Proceedings of the 5th International Conference on Business Process Management (BPM 2007)*, pages 344–359, Brisbane, Australia, 24-28 September 2007. Springer, Berlin Heidelberg.

[31] E. Lamma, P. Mello, F. Riguzzi, and S. Storari. Applying inductive logic programming to process mining. In *Proceedings of the 17th International Conference on Inductive Logic Programming (ILP 2007)*, pages 132–146, Corvallis, OR, USA, 19-21 June 2008. Springer, Berlin Heidelberg.

[32] D. Lowd and P. Domingos. Efficient weight learning for Markov Logic Networks. In *Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 200–211. Springer-Verlag, 2007.

[33] F. M. Maggi, R. P. Jagadeesh Chandra Bose, and Wil M. P. van der Aalst. Efficient discovery of understandable declarative process models from event logs. In *Proceedings of the 24th International Conference on Advanced Information Systems Engineering (CAiSE 2012)*, pages 270–285, Gdańsk, Poland, 25-29 June 2012. Springer, Berlin Heidelberg.

[34] F. M. Maggi, D. Corapi, A. Russo, E. Lupu, and G. Visaggio. Revising process models through inductive learning. In *Business Process Management Workshops - BPM 2010 International Workshops and Education Track, Revised Selected Papers*, pages 182–193, Hoboken, NJ, USA, 13-15 September 2010. Springer, Berlin Heidelberg.

[35] F. M. Maggi, A. J. Mooij, and Wil M. P. van der Aalst. User-guided discovery of declarative process models. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011)*, pages 192–199, Paris, France, 11-15 April 2011. IEEE.

[36] L. Mihalkova and R. J. Mooney. Bottom-up learning of Markov Logic Network structure. In *Machine Learning, Proceedings of the 24th International Conference (ICML 2007)*, pages 625–632, Corvallis, Oregon, USA, 20-24 June 2007. ACM, New York, NY, USA.

[37] S. Muggleton. Learning structure and parameters of stochastic logic programs. In *Proceedings of the 12th International Conference on Inductive Logic Programming (ILP 2002)*, pages 198–206, Sydney, Australia, 9-11 July 2002. Springer, Berlin Heidelberg.

[38] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19(20):629–679, 1994.

[39] M. Pesic, H. Schonenberg, and Wil M. P. van der Aalst. DECLARE: Full support for loosely-structured processes. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*

(EDOC 2007), pages 287–300, Annapolis, Maryland, USA, 15-19 October 2007. IEEE Computer Society Press.

[40] M. Pesic and W. M. P. van der Aalst. A declarative approach for flexible business processes management. In *Proceedings of the 2006 International Conference on Business Process Management Workshops (BPM 2006)*, pages 169–180, Vienna, Austria, 4-7 September 2006. Springer, Berlin Heidelberg.

[41] H. Poon and P. Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *Proceedings of the 21st National Conference on Artificial intelligence (AAAI 2006)*, pages 458–463, Boston, MA, USA, 16-20 July 2006. AAAI Press.

[42] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[43] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

[44] A. Rozinat, R. S. Mans, M. Song, and Wil M. P. van der Aalst. Discovering colored Petri nets from event logs. *International Journal on Software Tools for Technology Transfer (STTT)*, 10(1):57–74, 2008.

[45] R. Silva, J. Zhang, and J. G. Shanahan. Probabilistic workflow mining. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2005)*, pages 275–284, Chicago, Illinois, USA, 21-24 August 2005. ACM, New York, NY, USA.

[46] W. M. P. van der Aalst and M. Pesic. DecSerFlow: Towards a truly declarative service flow language. In *Proceedings of the 3rd International Workshop on Web Services and Formal Methods (WS-FM 2006)*, pages 1–23, Vienna, Austria, 8-9 September 2006. Springer, Berlin Heidelberg.

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

[48] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *WIREs Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.

[49] Wil M. P. van der Aalst, Boudewijn F. van Dongen, Christian W. Günther, R. S. Mans, Ana Karla Alves de Medeiros, Anne Rozinat, Vladimir Rubin, Minseok Song, H. M. W. (Eric) Verbeek, and A. J. M. M. Weijters. Prom 4.0: Comprehensive support for *Real* process analysis. In *Petri Nets and Other Models of Concurrency, International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, ICATPN 2007*, volume 4546 of *LNCS*, pages 484–494. Springer, 2007.

[50] Wil M. P. van der Aalst and Kees M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, MA, USA, 2002.

[51] Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

[52] B. F. van Dongen. Multi-Phase process mining: Building instance graphs. In *Proceedings of the 23rd International Conference on Conceptual Modeling (ER 2004)*, pages 362–376, Shanghai, China, 8-12 November 2004. Springer, Berlin Heidelberg.

[53] B. F. Van Dongen and H. M. W. Verbeek. Verification of EPCs: Using reduction rules and Petri nets. In *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE 2005)*, pages 372–386, Porto, Portugal, 13-17 June 2005. Springer, Berlin Heidelberg.

[54] K. M. van Hee, Z. Liu, and N. Sidorova. Is my event log complete? a probabilistic approach to process mining. In *Proceedings of the 5th International Conference on Research Challenges in Information Science (RCIS 2011)*, pages 1–7, Guadeloupe - French West Indies, France, 19-21 May 2011. IEEE.

[55] P. Weber, B. Bordbar, and P. Tino. A principled approach to the analysis of process mining algorithms. In *Proceedings of the 12th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL 2011)*, pages 474–481, Norwich, UK, 7-9 September 2011. Springer, Berlin Heidelberg.

[56] A. J. M. M. Weijters, W.M.P. van der Aalst, and A. K. Alves de Medeiros. Process Mining with the HeuristicsMiner algorithm. Technical Report WP 166, BETA Working Paper Series, Eindhoven University of Technology, Eindhoven, Netherlands, 2006.

[57] L. Wen, Wil M. P. van der Aalst, J. Wang, and J. Sun. Mining process models with non-free-choice constructs. *Data Mining and Knowledge Discovery*, 15(2):145–180, 2007.

[58] L. Wen, J. Wang, Wil M. P. van der Aalst, B. Huang, and J. Sun. Mining process models with prime invisible tasks. *Data Knowledge Engineering*, 69(10):999–1021, 2010.