CEUR-WS.org/Vol-4117/paper_1_PLP2025.pdf

A Scalable Approach to Probabilistic Compliance in Declarative Process Mining

Michela Vespa[1], Elena Bellodi[1]

[1]Department of Engineering, University of Ferrara, Via Saragat 1, Ferrara, Italy

Abstract
In the field of Process Mining, one of the main tasks is the assessment of compliance between actual process executions and a specific model, which is called compliance or conformance checking. Recently, a few works have started to propose probabilistic declarative process models, where the model is a set of constraints associated with a probability to model uncertainty, leading to the task of probabilistic conformance checking. In this paper, we propose to adopt PASCAL, an algorithm that learns Probabilistic Constraint Logic Theories in the learning from interpretation setting and that assigns to interpretations the probability of belonging to the positive class, for efficiently computing the probability of conformance of process traces against probabilistic declarative process models. We compare PASCAL with two other existing tools that perform probabilistic conformance checking. Results show that PASCAL achieves superior performance in terms of execution time, handling a much larger numbers of constraints and traces with a logarithmic computational complexity.

Keywords
Declarative Process Mining, Declare, Probabilistic Integrity Constraints, Probabilistic compliance

1. Introduction

Process Mining (PM) is a research field aimed at extracting structured knowledge from event logs generated by information systems. It encompasses multiple activities, including the discovery of process models from recorded event data, assessing compliance between actual process executions and predefined models (called *conformance checking*), and identifying opportunities for process improvement. Procedural models, such as Petri nets [1] or BPMN [2] diagrams, prescribe step-by-step sequences of actions, explicitly defining the allowed paths and control flow of the process. On the other hand, when a process model imposes constraints that must hold during process execution without enforcing a fixed sequence of steps, it is called *declarative*. Declarative process modeling languages, such as Declare [3] or Dynamic Condition Response (DCR) Graphs [4], focus on specifying rules and restriction, such as which activities must eventually occur, which cannot occur together, or temporal dependencies, thus enabling greater flexibility and adaptability in processes with high variability.

Often, in real-world scenarios, process models do not capture rigid, deterministic behaviors but rather describe guidelines or regulations that hold under uncertain conditions. For instance, if a patient is admitted to an hospital to treat a specific condition, some guidelines might prescribe a counseling session before undergoing surgery, but it is not strictly mandatory. This leads to a thorny issue for the task of declarative conformance checking: *how to evaluate compliance when the process model itself is uncertain*? Classical conformance checking techniques, which assume crisp constraints in process models, are insufficient because they only produce a binary ”yes/no” answer: a trace is either complaint or not with a given process model.

To address the intrinsic uncertainty of processes, recent works have introduced the notion of probability attached to process constraints into the Declare formalism, leading to the concept of probabilistic conformance checking or probabilistic compliance.

[5] and then [6] address this by introducing the ProbDeclare framework, which extends Declare constraints with an associated probability. A probabilistic constraint is defined as a triple $\langle \varphi, \bowtie, p \rangle$, where φ is an LTL$_f$ formula representing a temporal constraint over traces, $\bowtie$ is a comparison operator

© 2025 Copyright © 2025 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2025-12-03

(such as $=, \leq, \geq$), and p is a probability in the $[0, 1]$ interval. A log satisfies such a constraint if the total probability mass of traces in the log that satisfy φ conforms to the condition $\bowtie p$. For instance, $(\varphi, 0.7, \geq)$ means that a constraint φ is satisfied if the number of traces complaint with it is at least $\geq 70\%$ of the total traces in the log. This means the probability is interpreted with a *frequentist* meaning.

Differently from these works, in [7] we introduced the concept of Probabilistic Declarative Process Specification (PDS) starting from Probabilistic Logic Programming (PLP) and the Distribution Semantics [8]. A PLP program under that semantics defines a probability distribution over normal logic programs called possible worlds. In a PLP program, logic formulas are assigned a probability, which is interpreted as the probability of them appearing or not in a normal logic program. Similarly, the probability of a Declare constraint is treated as the probability that such constraint will appear (or not) in a possible process specification. The meaning of the probability attached to the constraint represents *how important* or strong that constraint is.

In addition, in [7] we compute the probability of compliance using the algorithm presented in [9], which however shows an exponential trend in execution time as the number of either crisp or probabilistic constraints increases, as it is based on the enumeration of all possible worlds. [10] proposed a solution to overcome this problem, leveraging an Answer Set Programming tool and the inclusion-exclusion principle [11], a mathematical technique used to calculate the probability of the union of multiple events, to compute the total compliance probability without explicit enumeration of all possible worlds.

In [12] we introduced the PASCAL algorithm, which both learns Probabilistic Constraint Logic Theories (PCLTs) in a learning from interpretations setting, and performs probabilistic classification of interpretations, computing their probability to belong to the positive class. PCLTs are sets of integrity constraints annotated with a probability.
In this paper we propose to exploit PASCAL's capabilities of performing probabilistic classification in the context of process mining, by using it to compute the probability that an observed process trace (an interpretation) is compliant (positive) with respect to a PDS (which is seen as a PCLT).

We perform an experimental validation based on the execution time for computing the probability of compliance, by comparing PASCAL with the results already presented in [7] and with [10], on a PDS generated from a real-world medical guideline: we demonstrate PASCAL's superiority in handling a much larger number of probabilistic process constraints, thanks to an execution time which is logarithmic in the number of constraints that are violated by a process trace, both in static scenario (a process trace is available at once) and in a run-time scenario (events of a trace arrive one at a time).

The paper is organized as follows: Section 2 introduces background on probabilistic Declarative PM and Probabilistic Constraint Logic Theories, Section 3 describes the application of PASCAL for probabilistic conformance checking, Section 4 reports the experimental evaluation and Section 5 concludes the paper.

2. Background

2.1. (Probabilistic) Declarative Process Models

In Process Mining, a *trace* is defined as a sequence of events representing one completed execution of a process instance. Each event is associated with an activity label and a timestamp that imposes a total order on the events in the trace. Formally, a trace t is expressed as $t = \langle e_1, e_2, \dots, e_n \rangle$, where each event e_i corresponds to an occurrence of an activity at a specific point in time, and the timestamps satisfy $\mathit{timestamp}(e_i) \leq \mathit{timestamp}(e_{i+1})$ for $1 \leq i < n$. An *event log* $\mathscr{L}$ is a collection (multiset) of traces, capturing the recorded executions of multiple process instances.

Declare [3] is a declarative process modeling language equipped with the formal semantics given by the LTL_f logic [13], and provides a set of (graphical) constraint templates. This language was developed specifically for business process modeling and offers high-level, parameterized constraint templates based on common business rules patterns. This formalism allows for the specification of behavioral constraints such as "*activity* a *must eventually be followed by* b" (known as *response(a,b)*), "activity *a* must be the first activity of the process" (known as *init(a)*), "*activity* a *and* b *cannot occur in the same trace*" (known as *not-coexistence(a,b)*) or "*activity* a *cannot be executed*" (known as *absence(a)*).

Definition 1 (Declarative Process Specification, from [14]). *A Declarative Process Specification is a triple $DS = (\textsc{Rep}, \Sigma, C)$, where $\textsc{Rep}$ is a finite set of constraint templates $\mathcal{C}(x_1, \dots, x_m)$ (with arity $m \in \mathbb{N}$), Σ is a finite set of activity names, and C is a finite set of instantiated constraints $\mathcal{C}(a_1, \dots, a_m)$ with $a_i \in \mathscr{A}$.*

Usually the constraints $\mathcal{C}(a_1, \dots, a_m)$ in a DS are considered as being in logical conjunction.

Each Declare template maps to one or more logical formulas φ, allowing logical entailment to define the concept of *compliance* of a trace t with respect to a constraint φ. Formally, a trace t is compliant with a constraint φ if and only if $t \models \varphi$ (trace t models or satisfies φ).

Definition 2 (Compliance of a trace versus a Declarative Process Specification). *A trace t is compliant with a DS if it entails the conjunction of the formulas φ_i corresponding to the $\mathcal{C}_i \in C$: $t \models \varphi_1 \wedge \ldots \wedge \varphi_n$ where n is the cardinality of C.*

Definition 3 (Probabilistic Constraint, from [7]). *Given a finite, non-empty set $\textsc{Rep}$ of constraint templates and a set Σ of activity names $a_1, \dots, a_m$, a* probabilistic constraint *is a constraint template $\mathcal{C} \in \textsc{Rep}$ instantiated over Σ that is annotated with a real number $p \in [0..1]$ (the* probability *of the constraint). We will indicate a generic i-th probabilistic constraint with the syntax:*

$$p_i \ :: \ \mathcal{C}_i(a_1, \dots, a_m)$$

Definition 4 (Probabilistic Declarative Process Specification, from [7]). *A* Probabilistic Declarative Process Specification *PDS is a Declarative Process Specification DS where each constraint $\mathcal{C}_i \in DS$ is a probabilistic constraint.*

We will refer to constraints annotated with probability $p_i = 1$ as crisp constraints.

Definition 5 (Compliance of a trace versus a PDS [7]). *Given a PDS, the probability of compliance of a trace t w.r.t. a PDS is defined as:*

$$Comp(t, PDS) = \sum_{t \models DS_i} P(DS_i), \tag{1}$$

where DS_i represents each deterministic process specification induced by the selection (or not) of probabilistic constraints over the PDS, and $P(DS_i)$ is the probability associated with each deterministic specification.

Taking inspiration from the Distribution Semantics, the idea in Def. 5 is to interpret the PDS as defining a probability distribution over a set of standard (i.e., non-probabilistic) Declarative Specifications (DS), each representing a possible world. These worlds are generated by selecting or excluding the probabilistic constraints $\mathcal{C}_i$, and always keeping the crisp constraints in the DS. Each selection identifies a specific DS, and its probability is computed as the product of the probabilities of the included constraints and the complements (i.e., $1 - p_i$) of the excluded ones. For a more detailed description of the connection with the Distribution Semantics, see [7].

Example 1. *Consider the PDS modeling a customer support process with the following constraints:*

$$C = \{ \quad \begin{array}{lll} 0.75 \ :: \ \textit{response(ticket\_opened, ticket\_resolved)} & & (\mathcal{C}_1) \\ 0.80 \ :: \ \textit{precedence(ticket\_opened, assign\_agent)} & & (\mathcal{C}_2) \quad \} \end{array}$$

Constraint $\mathcal{C}_1$ requires that every opened ticket is eventually resolved (with probability 0.75), and constraint $\mathcal{C}_2$ requires that the ticket's assignment to an agent is preceded by the ticket opening (with probability 0.8).

Consider the trace

$$t = \langle \textit{ticket\_opened}, \textit{assign\_agent} \rangle,$$

which complies with the precedence constraint c_2 since assignment happens after ticket opening, but violates the response constraint c_1 because the trace does not contain ticket_resolved.

Therefore, after generating all DSs shown in Table 1, trace t is compliant with DS_3 and DS_4, which do not include constraint c_1, and not compliant with DS_1 and DS_2. According to Def. 5, the compliance probability of t versus the PDS is:

$$Comp(t, PDS) = P(DS_3) + P(DS_4) = 0.2 + 0.05 = 0.25.$$

Note that the probability distribution over the DS_i is such that the sum of all probabilities $P(DS_i)$ always equals 1.

Declarative Process Specification (DS)	$P(DS_i)$
DS_1 = {resp(ticket_opened, ticket_resolved), prec(ticket_opened, assign_agent)}	$P(DS_1) = 0.75 \times 0.8 = 0.60$
DS_2 = {resp(ticket_opened, ticket_resolved)}	$P(DS_2) = 0.75 \times 0.2 = 0.15$
DS_3 = {prec(ticket_opened, assign_agent)}	$P(DS_3) = 0.25 \times 0.8 = 0.20$
DS_4 = { }	$P(DS_4) = 0.25 \times 0.2 = 0.05$

Table 1
Declarative Process Specifications generated by the PDS in Example 1 and corresponding probabilities. Constraints' names are abbreviated.

2.2. Probabilistic Constraint Logic Theories

[12] introduced Probabilistic Constraint Logic Theories (PCLTs), sets of integrity constraints annotated with a probability that assign a probability of being positive to logical interpretations. Specifically, a *Probabilistic Integrity Constraint* (PIC) C_i is of the form:

$$p_i \ :: \ L_1, \dots, L_b \rightarrow \exists(P_1); \dots; \exists(P_n); \forall\neg(N_1); \dots; \forall\neg(N_m) \tag{2}$$

where $p_i \in [0,1]$ is a probability, each L_i is a literal, and each P_j and N_j is a conjunction of literals. We call each $\exists(P_j)$ a *P disjunct* and each $\forall\neg(N_k)$ an *N disjunct.* $L_1, \dots, L_b$ is called the body of C ($Body(C)$) and $\exists(P_1); \dots; \exists(P_n); \forall\neg(N_1); \dots; \forall\neg(N_m)$ is called the head of C ($Head(C)$). The semicolon here represents a disjunction. The variables that occur in the body are quantified universally with scope the PIC. The variables in the head that do not occur in the body are quantified existentially if they occur in a P disjunct and universally if they occur in an N disjunct, with scope the disjunct they occur in.

A set of PICs is called a PCLT $T = \{p_1 \ :: \ C_1, \dots, p_n \ :: \ C_n\}$.

The approach for assigning a semantics to PCLTs is inspired by the distribution semantics [8]: a PCLT T defines a probability distribution on non-probabilistic ground constraint logic theories called *worlds*, such that for each grounding of the body of each IC, we include the IC in a world with probability p_i and we assume all groundings to be independent. The probability is to be interpreted as the strength of the IC: a probability p_i means that the sum of the probabilities of the possible theories where a grounding of the constraint is present is p_i.

[12] also presents an algorithm, PASCAL, for "ProbAbiliStic inductive ConstrAint Logic", for learning a PCLT according to the learning from interpretation setting of ILP [15]: while the latter learns a theory that discriminates the positive from the negative interpretations, PASCAL learns a PCLT able to compute the probability of the positive class given an interpretation I and a background knowledge **BG**. This corresponds to the probability of a PCLT T satisfying I given **BG**, and is referred to as $P(\oplus|I)$. [12] demonstrated that the computation of $P(\oplus|I)$ depends only on the PICs that are *not* satified in I, i.e.:

$$P(\oplus|I) = \prod_{i=1}^{n}(1 - p_i)^{m_i} \tag{3}$$

where m_i is the number of groundings of the n integrity constraints C_i that are not satisfied in I. So, computing the probability of the positive class given an interpretation is $O(n \log m)$, where m is the maximum number of groundings, and computing m is polynomial in the database size.

3. Probabilistic Compliance Computation with PASCAL

In this paper we are interested in using the inference module of PASCAL that, given an intepretation I, allows one to compute $P(\oplus|I)$. In fact, computing the probability of compliance of a trace to a PDS is equivalent to computing the probability of being positive of an interpretation I w.r.t a PCLT: a set of events in a trace can be seen as a logical interpretation of ground facts, while a PDS and a PCLT are both composed of probabilistic constraints.

The motivation behind the use of PASCAL is given by the fact that, as shown in Example 1, the enumeration of all possibile DSs is exponential in the number of probabilistic constraints in the PDS, hence the computation of the compliance probability is impractical with large PDSs.

PASCAL is distributed as a SWI-Prolog pack[1]. We show that a Prolog predicate of PASCAL can be easily exploited to return the probability of compliance.

In fact, the `test_prob_pascal/6` predicate with the following interface: $test\_prob\_pascal(+T : list, +List\_of\_folds : list, -NPos : int, -NNeg : int, -LL : float, -ExampleList : list)$ can be used to compute $P(\oplus|I)$ where T is the input PCLT and $list\_of\_folds$ is the list of all interpretations that we want to test against the PCLT.

At the end of the execution, the predicate `test_prob_pascal/6` yields: $NPos/NNeg$ (numbers of positive/negative traces resp.), the global log-likelihood of the test examples *LL*, and the list *ExampleList* containing pairs *Probability–Example*: *Example* is a for a positive example a, and `\+(a)` for a negative example `\+(a)`, and *Prob* is the probability of a being positive.

The PCLT T is given as a list of probabilistic integrity constraints written in the form:

$$\texttt{rule}(([(\texttt{Sign}, [\texttt{Head}])] : -[\texttt{Body}]), \texttt{Probability}).$$

where `[(Sign, [Head])]` represents a list of disjuncts in the head of the rule, each annotated with a sign (+ for positive, − for negative). `[Body]` is a list of literals. This corresponds to Eq. 2.

Note that a PCLT should perform discriminative learning, as its goal is to assign a probability of being positive to interpretations by encoding a distribution on the class variable. This means the process trace, represented as an interpretation, must include an extra fact `pos(id)` or `neg(pos(id))` (where `id` is the identifier of the interpretation) to specify the class (positive or negative) of the interpretation. In our case, a process trace is labeled as `pos`, because the PASCAL algorithm returns the probability that a given interpretation belongs to the positive class. By giving as input a Probabilistic Declarative Process Specification (PDS) in the T parameter and one process trace in `List_of_folds`, we can directly obtain the probability of the trace being positive from *ExampleList*, which corresponds to its probability of compliance $\text{comp}(t, PDS)$ to the PDS.

Example 2. *Suppose we have to compute the probability of compliance as in Example 1, but with the inference module of PASCAL.*

The PDS T given as input is:

```
pclt([rule(([((+),[ticket_resolved(T2),T2>T1])]:-[ticket_opened(T1)]), 0.75),
      rule(([((+),[(ticket_opened(T1),T1<T2])]:-[assign_agent(T2)]), 0.8)]).
```

The computation can be executed this way:

```
:- pclt(T),test_prob_pascal(T,[1],NPos,NNeg,LL,ExampleList).
```

where the second argument contains the trace's identifier.

The result is: `ExampleList = [0.25-1]`*, which indicates that trace* 1 *has probability 0.25 of being compliant to the PDS.*

This coincides with the result that we would obtain from Equation 3, with interpretation I corresponding to trace 1*,* $n = 1, m_i = 1$*: the probability of compliance is* $1 - 0.75 = 0.25$*, with* $p_i = 0.75$ *being the probability of the violated constraint* c_1*.*

[1]https://eu.swi-prolog.org/pack/list?p=pascal

4. Experimental Results

We conduct here a thorough comparison with our prior compliance evaluation based on the IFF^{Prob} framework [9] and with the solution presented in [10] based on the inclusion-exclusion principle. Our experiments assess the PASCAL and IFF^{Prob} across two scenarios, static (all events in a trace available at once) and run-time (events arrive incrementally one at a time). We record the execution time for computing the probability of compliance against probabilistic declarative process specifications with IFF^{Prob} and PASCAL in subsections 4.1 and 4.2. In subsection 4.3 we compare PASCAL with the results published in [10] in the static scenario.

The testing environment was set up on a Linux machine equipped with an Intel® Xeon® E5-2630v3 processor running at 2.40 GHz, with 250 GB of RAM. The Prolog stack was allocated the same amount of memory to maximize performance and handle a large number of worlds. Each job was allowed a maximum execution time of 8 hours.

4.1. Static Compliance Evaluation

To conduct the tests we generated several synthetic PDSs modeled on a healthcare guideline from [16]. These PDSs always include 21 constraints (examples of which can be found in [7]), but with a varying proportion of crisp and probabilistic ones (for instance, 3 crisp constraints and 18 probabilistic). For each test we used a single trace of 21 events corresponding to a patient, that was kept constant across runs. Figure 1 illustrates the comparison in execution times between IFF^{Prob} and PASCAL.

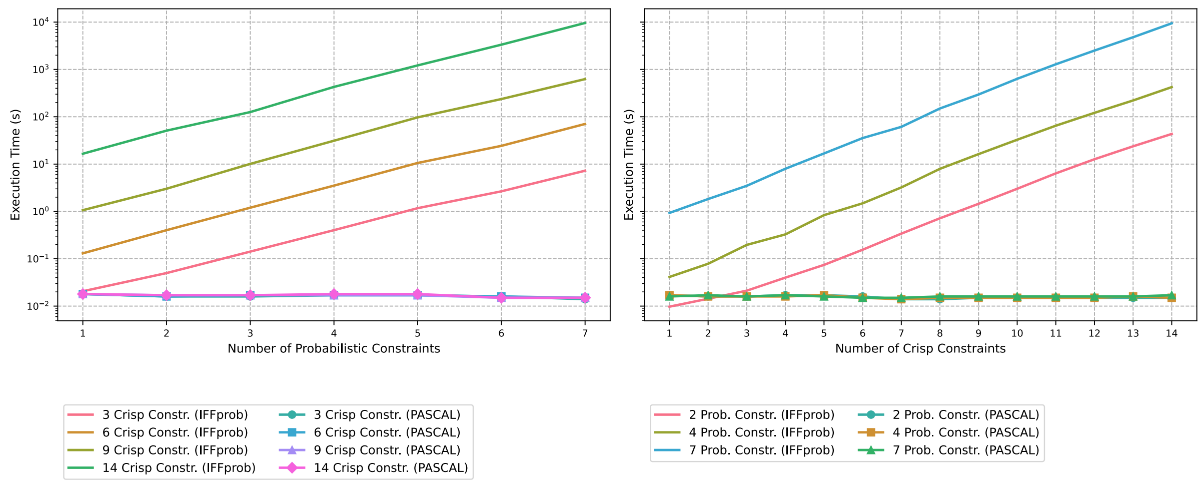

Figure 1: Execution times for computing the probability of compliance of one trace against a PDS with a varying number of crisp and probabilistic constraints. The scale is logarithmic.

The left graph shows the impact of increasing the number of probabilistic constraints from 1 to 7, while keeping the number of crisp constraints fixed at 3, 6, 9, and 14 for different runs. The right graph shows the execution time when increasing crisp constraints from 1 to 14, while keeping the number of probabilistic constraints fixed at 2, 4, and 7.

Results show that, in both cases, IFF^{Prob} execution time grows exponentially as more probabilistic or crisp constraints are added (note the logarithmic scale on the graph). Each time the body of a probabilistic integrity constraint is satisfied, two possible worlds are generated, one including the constraint, one excluding it, causing an exponential enumeration equal to 2^n with n the number of probabilistic constraints. In contrast, PASCAL shows constant execution times.

4.2. Run-time Compliance Evaluation

For this experiment we adapted PASCAL's inference module to compute probabilistic compliance incrementally, as new events arrive one at a time, by updating the trace and invoking `test_prob_pascal/6` after each event.

Figure 2 presents execution times of IFF^{Prob} and PASCAL when we test the compliance of a trace (from the same medical domain) that grows up to 21 events at run-time, against the PDS of 21 probabilistic constraints of the previous experiment. Again, IFF^{Prob} exhibits a rapid increase in memory usage beyond 14 probabilistic constraints, eventually exceeding the available memory. PASCAL shows a constant behaviour. Table 2 reports detailed execution times.

Despite these limitations, IFF^{Prob} successfully handles up to 3,188,646 worlds within the available memory. It can generate 209,952 explanations in approximately 88 seconds when testing compliance against a model with 14 probabilistic constraints and no crisp constraints.

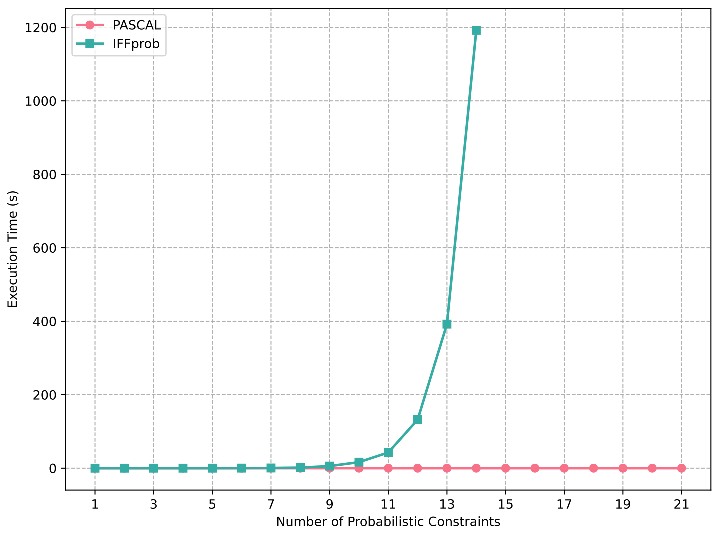

Figure 2: Execution times for computing the probability of compliance of a trace with dynamically arriving events, with varying numbers of probabilistic constraints in the process model.

# Prob.	Execution Time (s)		# Prob.	Execution Time (s)	
Constraints	PASCAL	IFF^{Prob}	Constraints	PASCAL	IFF^{Prob}
1	0.017	0.001	12	0.016	131.958
2	0.017	0.003	13	0.018	392.186
3	0.020	0.006	14	0.018	1192.141
4	0.018	0.018	15	0.019	T.O.
5	0.017	0.055	16	0.019	T.O.
6	0.018	0.181	17	0.020	T.O.
7	0.019	0.526	18	0.019	T.O.
8	0.017	1.562	19	0.020	T.O.
9	0.018	5.722	20	0.021	T.O.
10	0.018	16.408	21	0.019	T.O.
11	0.018	42.748			

Table 2
Execution times (in seconds) of PASCAL and IFF^{Prob} for run-time compliance. T.O. indicates time out.

4.3. Comparison with the inclusion-exclusion principle

This section compares PASCAL performance with an alternative compliance checker based on Answer Set Programming (ASP) presented in [10] in a static scenario. Both tools take as input a probabilistic declarative process specification and a collection of execution traces; each trace will be assigned a probability that represents its degree of conformance with the specification.

PASCAL's computational cost is primarily influenced by the number of input traces rather than the number of constraints (probabilistic or crisp) in the PDS or the trace length.

The ASP-based approach relies on the inclusion-exclusion principle and requires considering subsets of probabilistic constraints that are either satisfied (SAT set) or violated (VIO set) by the trace. These subsets are used to calculate the probability of compliance because VIO and SAT are defined as the union of non-disjoint events, so their probabilities can be computed by applying the inclusion-exclusion principle. The computational complexity of this method is primarily determined by the number and size of the subsets considered. Specifically, the complexity grows exponentially with respect to the minimum between the VIO and SAT sets' size, which represents the number of "effective constraints".

We tried to make the comparison as fair as possible with the results presented in Table 1 of [10], that we report in Table 3 together with the execution times of PASCAL. As indicated in [10], in the worst case, the number of effective constraints is half of the number of probabilistic constraints so, firstly, we doubled the number of constraints in our PDSs. Secondly, we conducted the experiments on a laptop equipped with an AMD Ryzen 7 5800H processor running at 3.2 GHz; as we do not have this information from [10], we decided to increase the number of probabilistic constraints in our PDSs beyond 54 to clearly show that PASCAL has a constant execution time, slightly increasing when scaling up at 200 constraints. The total number of different PDS generated was 11, composed of a number of PIC in the range 42 - 200 as shown in Table 3.

Thirdly, as done in [10], we performed 50 runs of compliance verification for each PDS. However, we did not change the constraints across the runs as changing the *type* of constraints does not influence PASCAL's execution time, that only depends on the number of constraints. In each run, compliance for a set of 10 different traces was verified 50 times. These traces are composed of a different number of events, chosen arbitrarily, with length ranging from 2 to 14 events. As previously noted, trace length (number of events per trace) does not impact inference time; instead, the only variable that affects PASCAL's performance is the number of traces in the log. The very low standard deviation observed is likely due to minimal CPU fluctuations.

[10] show that, in the worst case, complexity is of the order of $O\left(2^{\frac{n}{2}}\right)$, which explains why the method can perform significantly better than a complete enumeration of 2^n subsets. However, PASCAL has a lower complexity of the order of $O(n \log m)$ with respect to the number of constraints n.

# Effective Constraints	# Probabilistic Constraints	Incl-Excl. Principle		PASCAL	
		Avg Runtime	Std Dev	Avg Runtime	Std Dev
21	42	1.391	0.031	**0.082**	0.006
22	44	2.936	0.060	**0.082**	0.002
23	46	6.012	0.135	**0.083**	0.005
24	48	12.980	1.571	**0.085**	0.003
25	50	25.850	0.621	**0.086**	0.005
26	52	54.676	1.206	**0.086**	0.005
27	54	109.370	5.096	**0.087**	0.004
-	60	-	-	0.088	0.001
-	80	-	-	0.098	0.002
-	100	-	-	0.107	0.003
-	200	-	-	0.151	0.005

Table 3
Comparison of the execution time for returning the probability of compliance between the solution based on the inclusion-exclusion principle and PASCAL's inference module.

5. Conclusions

In this work we proposed to apply the inference module of the PASCAL algorithm for performing probabilistic conformance checking in declarative process mining. PASCAL was designed, in a previous work, to handle Probabilistic Constraint Logic Theories in the learning from interpretation settings: such theories are seen here as probabilistic declarative process specifications (PDS), while the probability of an interpretation being positive corresponds to the probability that a process trace complies with a PDS both in static and run-time scenarios. We compared PASCAL with two other algorithms for computing the probability of compliance, the IFF^{Prob} framework and an ASP-based tool. The results demonstrate that PASCAL scales much better as the number of probabilistic constraints grows.

Acknowledgments

Research funded by the Italian Ministerial grant PRIN 2022 "Probabilistic Declarative Process Mining (PRODE)", n. 20224C9HXA - CUP F53D23004240006, funded by European Union – Next Generation EU.

Declaration on Generative AI

The authors declare that in the planning, drafting, and/or revision of the work have made no use of generative AI tools.

References

[1] J. L. Peterson, Petri nets, ACM Comput. Surv. 9 (1977) 223–252. URL: https://doi.org/10.1145/356698.356702. doi:10.1145/356698.356702.

[2] O. M. Group, Business Process Model and Notation (BPMN), Version 2.0, 2011. URL: https://www.bpmn.org/.

[3] M. Pesic, H. Schonenberg, W. M. van der Aalst, Declare: Full support for loosely-structured processes, in: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), 2007, pp. 287–287. doi:10.1109/EDOC.2007.14.

[4] T. T. Hildebrandt, R. R. Mukkamala, Declarative event-based workflow as distributed dynamic condition response graphs, in: K. Honda, A. Mycroft (Eds.), Proceedings Third Workshop on Programming Language Approaches to Concurrency and communication-cEntric Software, PLACES 2010, Paphos, Cyprus, 21st March 2010, volume 69 of *EPTCS*, 2010, pp. 59–73. URL: https://doi.org/10.4204/EPTCS.69.5. doi:10.4204/EPTCS.69.5.

[5] F. M. Maggi, M. Montali, R. Peñaloza, A. Alman, Extending temporal business constraints with uncertainty, in: Business Process Management: 18th International Conference, BPM 2020, Seville, Spain, September 13–18, 2020, Proceedings 18, Springer, 2020, pp. 35–54.

[6] A. Alman, F. M. Maggi, M. Montali, R. Peñaloza, Probabilistic declarative process mining, Inf. Syst. 109 (2022) 102033. doi:10.1016/J.IS.2022.102033.

[7] M. Vespa, E. Bellodi, F. Chesani, D. Loreti, P. Mello, E. Lamma, A. Ciampolini, Probabilistic compliance in declarative process mining, in: G. D. Giacomo, V. Fionda, F. Fournier, A. Ielo, L. Limonad, M. Montali (Eds.), Proceedings of the 3rd International Workshop on Process Management in the AI Era (PMAI 2024) co-located with 27th European Conference on Artificial Intelligence (ECAI 2024), Santiago de Compostela, Spain, October 19, 2024, volume 3779 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024, pp. 11–22. URL: https://ceur-ws.org/Vol-3779/paper1.pdf.

[8] T. Sato, A statistical learning method for logic programs with distribution semantics, in: L. Sterling (Ed.), Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, Tokyo, Japan, June 13-16, 1995, MIT Press, 1995, pp. 715–729.

[9] E. Bellodi, M. Gavanelli, R. Zese, E. Lamma, F. Riguzzi, Nonground abductive logic programming with probabilistic integrity constraints, Theory and Practice of Logic Programming 21 (2021) 557–574. doi:10.1017/S1471068421000417.

[10] M. Alviano, A. Ielo, F. Ricca, Efficient compliance computation in probabilistic declarative specifications, volume 3799, CEUR-WS, 2024.

[11] S. S. Sane, The inclusion-exclusion principle, Hindustan Book Agency, Gurgaon, 2013, pp. 57–79. doi:10.1007/978-93-86279-55-2_4.

[12] F. Riguzzi, E. Bellodi, R. Zese, M. Alberti, E. Lamma, Probabilistic inductive constraint logic, Machine Learning 110 (2021) 723–754. doi:10.1007/s10994-020-05911-6.

[13] G. D. Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: F. Rossi (Ed.), IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013, IJCAI/AAAI, 2013, pp. 854–860. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997.

[14] C. D. Ciccio, M. Montali, Declarative process specifications: Reasoning, discovery, monitoring, in: W. M. P. van der Aalst, J. Carmona (Eds.), Process Mining Handbook, volume 448 of *LNBIP*, Springer, 2022, pp. 108–152. doi:10.1007/978-3-031-08848-3_4.

[15] L. De Raedt, W. Van Laer, Inductive constraint logic, in: Proceedings of the 6th International Conference on Algorithmic Learning Theory, ALT '95, Springer-Verlag, Berlin, Heidelberg, 1995, p. 80–94.

[16] U. O. Gustafsson, et al., Guidelines for perioperative care in elective colorectal surgery: Enhanced recovery after surgery (eras®) society recommendations: 2018, World Journal of Surgery 43 (2019) 659–695. doi:10.1007/s00268-018-4844-y.