On deviant traces and other stranger things

Federico Chesani[a], Chiara Di Francescomarino[b], Chiara Ghidini[b], Daniela Loreti[a,*], Fabrizio Maria Maggi[c], Paola Mello[a], Marco Montali[c], Sergio Tessaris[c]

[a] *DISI - University of Bologna, Italy*
[b] *Fondazione Bruno Kessler, Trento, Italy*
[c] *Free University of Bozen/Bolzano, Italy*

Abstract

abstract goes here

Keywords: Process mining, Process discovery, Declarative process models, Deviant traces

---

1. Introduction

Process discovery is an important research field of process mining [1]. It encompasses techniques to automatically learn a business process model from a given set of execution cases, usually recorded in a log file. The model going to be learned must rely on a shared language to express the possible evolutions of business cases; just as the log file must have a clear, unambiguous syntax to express the relevant events occurred during the business process.

DL: DSS page limit: 34

Process discovery algorithms are usually classified into two categories, procedural and declarative, according to the language they employ to represent the output process model. Procedural techniques envisage the process model as a synthetic description of all possible sequence of actions that the business allows from a certain beginning to an end. Declarative discovery algorithms—which are the subject of this work—return the model as a set of constraints which must be fulfilled by the business execution cases.

*Corresponding author
Email address: `daniela.loreti@unibo.it` (Daniela Loreti)

Albeit extremely intuitive in some cases, procedural discovery may show poor results when the business process is unstructured and characterised by high variability [2]. In that case, forcing the vision of the business process towards the template of a begin-to-end sequence of activities, may result in a so-called "spaghetti" model [3]. Declarative approaches are preferable in these situations because allow expressing the model as a simple elicitation of permitted and prohibited behaviours.

Besides the declarative/procedural classification, the process discovery approaches can be also divided into two categories according to their vision on the model-extraction task. The vast majority of works [4, 5, 6, 7] intend discovery as an unsupervised classification task, where the set of traces in the input log must be analysed to extract valuable information about the frequency of occurrence of certain behavioural templates. This information is then used to build the execution model. Typically, the approaches in this category make use of thresholds, and language biases to drive the discovery task. A smaller number of works [8, 9, 10] intend the model-extraction task as an inductive-learning process, where a set of logical clauses is produced by the analysis of the input log.

Both the categories have their advantages and shortcomings. Differently from induction-based techniques which do not usually stand out for their performance, classification-oriented discovery has reached high performance and effectiveness—provided that suitable metrics (e.g., constraint support, coverage, etc.) are defined to clearly assess the quality of the extracted model. Furthermore, while inductive-learning approaches have a solid theoretical background because inductive reasoning has been studied since the dawning of artificial intelligence, classification-oriented techniques do not seek perfection, but just a good approximation of the most common behaviours. Depending on the values assigned to parameters such as thresholds on support and coverage, classification-oriented approaches can provide completely different results. In general, the tuning of these parameters is not a straightforward task: the thresholds and language biases defined for a certain model extraction task, might not be suitable for a different use case. On the other hand, inductive-learning approaches need both *positive* and *negative* examples to properly work, i.e. business execution cases that are compliant with the model going to be discovered are necessary as well as non-compliant cases. Classification-oriented discovery instead, works on positive examples only and discards as noise the negative ones, whenever they are present in the log. In particular, we can say that the availability of labelled positive and negative business execution examples is a crucially discriminative factor to opt for one process discovery view or the other. Some studies endorse the thesis that, since in most of the real-life situations we cannot distinguish positive and negative cases in the input log, we should work as if the latter do not exist. Nonetheless, it is indisputable that, for each (meaningful) discovered business process model, there is a set of traces that are necessarily excluded because they are not compliant with the model. Such set constitutes a sort of "upside-down world", specular to the real world of positive, common and allowed cases.

In this work, we propose a view on process discovery that deviates from the two presented so far. Like allowed traces can be exploited to extract information about the usual process model, we explore the possibility that the "upside-down world" of negative execution traces could be used—if it was accessible—to understand the reasons why deviations from the common process model occur. This information would be useful not only to better clarify what should be deemed compliant with the model and what should not, but also to specify parts of the business process in a more synthetic and effective way—by converting for example, a set of positive execution constraints into a single negative one.

Our work envisage the process discovery task as a *satisfiability problem* and intertwines the constructive elements of both classification- and inductive-logic-oriented approaches into a single technique able to discover declarative process models by actively making use of both the positive traces and the "upside-down world" of deviant and negative examples—whenever they are available in the log.

This attempt yields higher efficiency and efficacy avoiding the known drawbacks of the two most common views of process discovery.

DL: Refine once our performance are known.

2. Motivations

Process discovery focuses on the analysis of an event log in order to automatically learn the process model underpinning the cases of such input log. In general, the techniques in this field assume that the input log is not the complete elicitation of all the expected cases. Other traces not reported in there might be deemed compliant with the expected behaviour of the system. Therefore, the aim of process discovery techniques is necessarily to generalise the log by finding a compact way to express the usual behaviour of the systems, for example, by means of a structured or declarative process model. Such generalisation causes the resulting model to allow as compliant a larger set of traces w.r.t. the input log [11]. This is one of the challenges of process discovery. If we could rely on logs reporting all expected behaviours, the extraction of the model would be a rather straightforward task, because we would need to learn a model exhaustively covering all the elicited cases. Also, if the model going to be learned is aimed for a following compliance checking task, model extraction would not be crucial, because a simple algorithm verifying the membership of a trace to the input set would serve the purpose. On the other hand, in order to avoid the so-called "spaghetti" models, process discovery must also prevent overfitting [7]. To this end, a widespread strategy is to overlook those process cases that show a particularly infrequent behaviour. A practice that is often obtained by checking that the extracted model constraints meet certain thresholds according to predefined metrics, e.g. reach a certain level of recall and specificity. It is therefore evident that, besides the usual attempt to generalise while extracting the model, process discovery necessarily performs an opposite attempt to increase the specificity by excluding some traces from the learning task.

The overlooked traces represent a sort of "stranger" behaviour, a deviation from the usual and expected conduct, which is the subject of *deviance mining*. Indeed, deviance mining is a field of process mining that encompasses techniques precisely to explain the reasons why a business process deviates from its normal execution. According to theory, deviations can have a positive or negative connotation. Positive deviations refer to desirable cases, where the business process shows particularly high performance, such as short execution time, low cost, or particularly profitable outcomes [12]. On the contrary, negative deviances usually refer to unwanted cases, that is situations non-conforming with the expected behaviour—because, for example, they produce an unwanted outcome or exceed the conventional execution time or cost [13].

Most process discovery techniques do not consider negative examples. Indeed, a widespread position in this regard is that negative traces are usually not available. Event logs usually report a restricted number of non-compliant or unwanted case blurred in a much larger number of positive and more common examples. Nonetheless, we could say that even those process discovery techniques that are not explicitly based on the availability of negative example, actually make use of them in an implicit way, by assuming that they are somewhere present in the log and stating that they must be overlooked. Differently from most previous approaches, we believe that negative (as well as positive) deviances might still have some informative content that is important to consider when discovering the process model. A more conscious shift is needed toward considering not only the positive and usual traces in the log, but also all the others, which provide information on negative and/or less frequent cases.

Another popular practice among process discovery techniques is that of defining a language bias, that clarifies the order in which the template of constraints must be considered during the learning task. Indeed, given a certain language to express the business model, different results may arouse depending from which patterns we search first among the log's cases. The choice of a language bias over another can be seen as a way to drive the discovery process towards a certain direction. Obviously, different results correspond to different ways to classify those traces that were not yet observed i.e., those case that are not present in the log, but might occur in the future. In a sense, the language bias required by some process discovery techniques is a sort of implicit heuristic to decide in which order we must perform the generalisation or specialisation steps, while composing the business model.

Besides the benefit of taking into account deviant traces, we claim that process discovery should be able to take advantage from a more explicit heuristic to explore the search space. Among all possible behavioural patterns occurring in the logs, the choice of one over another should be driven by a more explicable strategy than simply defining an order of preferred constraints to be considered.

The contributions of this work can be listed as follows.

- A novel approach to deviance mining, which makes use of the information brought by both positive and negative execution cases to determine a set of possible models to represent the business process.
- A heuristic to select the preferred model according to predefined goals of generalisation and specialisation.
- An evaluation of the performance of the proposed approach w.r.t. other relevant works in the same field.

3. Preliminaries

Our technique relays on the key concept of *event log*, intending it as a set of observed business process executions, logged into a file in terms of all the occurred events. Each event is related to a specific process *instance*, and describes the occurrence of a well-defined step in the process, namely an *activity*. The logged set of events composing a process instance is addressed as *trace* or *case*. From the analysis of the *event log*, we want to extract a Declare [**?**] *model* of the business process (or refine a preexisting one), able to represent the logged traces in a synthetic way, trough a set of constraints. In particular, we assume some of the logged traces are *positive*, i.e. they fulfil all the constraints in the business model, whereas others are *negative* in the sense that they diverge from the expected behaviour by violating at least one constraint in the model. We denote with L^+ the set of positive traces in the input event log, and with L^- the set of the negative ones. We also consider a set P of Declare constraints that are known to be valid on the positive traces. Such set can be the expression of domain knowledge or the result of a classification-oriented discovery algorithm previously applied to L^+. Obviously, P can be also the empty set.

Given a set of Declare templates D and activities A, we identify with $D[A]$ the set of all possible grounding of templates in D w.r.t. A, i.e. all the constraints that can be built using the given activities. Our technique makes use of the concept of *closure* to account for deduction over a set of Declare constraints. For example, if we consider a set of constraints $D = \{\mathsf{INIT}(\mathsf{a})\}$, the closure of such simple set is $D = \{\mathsf{INIT}(\mathsf{a}), \mathsf{EXISTENCE}(\mathsf{a})\}$, because the fact that the process must start with an activity a implies that all process instances must contain a. Since we do not have a complete calculus for the language of conjunctions of Declare constraints we impose just correctness; that is we require that for any closure $cl : 2^{D[A]} \to 2^{D[A]}$ the set of traces compliant with $C \subseteq D[A]$ is the same as the ones compliant with $cl(C)$, i.e.

DL: siamo sicuri della doppia implicazione?

$$\forall C, t | C \subseteq D[A], t \models C \iff t \models cl(C) \tag{1}$$

The goal of our technique is therefore to refine a previously learned (possibly empty) model P by selecting a set of Declare constraints such that all positive traces and none of the negative are compliant[1], (where a trace is compliant with the set of constraints iff each constraint is satisfied by the trace). Clearly, there can be several sets satisfying these conditions and we need to introduce a notion of fitness to select the preferred ones.

In some context, *generality* can be the fitness measure, that is we want to identify the set that is less committing in terms of restricting the admitted traces. In some other context on the contrary, we might be interested in the identification of a more *specific* model. So besides allowing all traces in L^+

[1] The conditions on all the positives and none of the negatives can be relaxed requiring a percentage of them; but this is outside the focus of the present work, and left for future investigation

and forbidding all traces in L^-, the choice between a general or specific model, obviously affects the classification of the unknown traces.

Intuitively, a model M is more general than another M' if M allows a superset of the traces accepted by M', i.e. defining $\mathcal{C}_M$ the set of all traces compliant with M,

Definition 3.1. *Model generality/specificity.* A model M is more general than another M'—and symmetrically M' is more specific than M—if and only if $\mathcal{C}_{M'} \subset \mathcal{C}_M$.

Obviously, testing the generality of a model according to this definition is not feasible, because it requires considering all the allowed/disallowed traces. The closure operator can be employed for such purpose. The two methods—comparing the set of traces $\mathcal{C}_M$ and $\mathcal{C}_{M'}$, or comparing their closures $cl(M)$ and $cl(M')$—are not equivalent because the deductive system deriving from the Declare language is not complete. Nonetheless, as the system is correct, the closure operator can be used to identify which model is more general/specific.

4. The approach

For the sake of modularity and easiness of experimenting with different hypotheses and parameters, we divide our algorithm into two clearly separate stages: one to identify the candidate models, and one to asses which is the best according the fitness measure we decide to apply. However these two steps can be merged into a single monolithic search-based algorithm.

As regards the first step, we are interested in the subsets S of $D[A]$ satisfying the conditions:

1. $\forall t \in L^+, t \models S$, i.e. all positive traces are compliant with S;
2. $\forall t \in L^-, t \not\models (S \cup P)$, i.e. none of the negative traces is compliant with the union of S and P.

Algorithm 1 reports the procedure to generate all possible candidate models satisfying these two conditions. It is implemented via a brute force strategy

Algorithm 1 Generation of all possible models allowing all traces in L^+ and disallowing at least one trace in L^-.

Input: $D[A], L^+, L^-$

Output: *compatibles* $\subseteq D[A]$, *choices* $: L^- \to 2^{D[A]}$

```
1: procedure CandidateGeneration(D[A], L⁺, L⁻)
2:     compatibles = {c ∈ D[A] | ∀t ∈ L⁺ ⊨ c}
3:     for t ∈ L⁻ do
4:         choices(t) = {c ∈ compatibles | t ⊭ c}
5:     end for
6:     return compatibles, choices
7: end procedure
```

that first collects the set of all the constraints that are satisfied by all the positive traces (Line 2). Subsequently, each negative trace is associated (by means of the function *choices*) with the subset of those constraints that are not satisfied by the trace (Line 4). From the point of view of the implementation, the algorithm leverages the semantics of Declare patterns defined by means of regular expressions [14] to verify the compliance of the traces. It is implemented in Go language employing a regexp implementation that is guaranteed to run in time linear in the size of the input[2].

Concerning the second step, we use an approximation algorithm (see Algorithm 2) that makes use of the closure operator to guide the search of the optimal solution over the set of constraints. In practice, the algorithm uses the Answer Set Programming (ASP) [15] system clingo as an optimisation engine. The selection of the optimal stable model does not require the compliance verification on the traces, which are therefore not necessary as input. The candidate set and the positive discovery model are encoded as facts, while the closure operator is implemented as a set of ASP rules.

[2] For more details, see the Go package regexp documentation at https://golang.org/pkg/regexp/

Algorithm 2 Selection of the best model according to custom model fitness.

Input: $choices : L^- \rightarrow 2^{D[A]}$, $compatibles,\ A, D[A], P, cl : 2^{D[A]} \rightarrow 2^{D[A]}$

Output: $S \subseteq D[A]$

1: **procedure** SELECTION($compatibles, choices, A, D[A], P, cl$)
2: **select** $S \subseteq compatibles$ **s.t.**
3: 1. $\forall t \in L^-, choices(t) \cap cl(S \cup P) \neq \varnothing$
4: 2. $is\_optimal(S)$
5: 3. $\nexists S' \subset S \mid cl(S \cup P) = cl(S' \cup P)$
6: **return** S
7: **end procedure**

DL: nella linea 2 avrei potuto dire $S \subseteq C$ dove $C = \{c \in D[A] \mid \exists t \in L^-, c \in choices(t)\}$, sarebbe stata un'ottimizzazione, ma non so se sarebbe valida anche per la ricerca del modello massimale

The first condition of Algorithm 2 (Line 3) specifies that S must be a subset of *compatibles* able to discard all traces in L^- as non-compliant.

The second condition (Line 4) accounts for the model fitness function, which can be customized according to the specific needs. For example, we could consider *generality* as a fitness measure.

In that case, we could implement the $is\_optimal(S)$ function as a search for the *minimal* S, intending it as the set S of constraints for which there is no S' such that $cl(S' \cup P) \subset cl(S \cup P)$.

On the other hand, if we want to find the most specific set of constraints—i.e. the model composed of Declare templates in D that excludes the higher number of unknown traces—the $is\_optimal(S)$ operation must be implemented as a search for the *maximal* S, intending it as the S for which there is no S' such that $cl(S \cup P) \subset cl(S' \cup P)$.

Finally, the last selecting condition (Line 5) allows reducing the redundancy of the extracted model. This condition is desirable because—even when the user is interested in the most specific model—redundancy compromises the readability of the solution, without adding any value. Nonetheless, it is important to notice that the condition of Line 5 may not be sufficient to completely avoid redundancy because we do not have a complete calculus for the language of conjunction of Declare constraints.

Note that the set of activities A is required as input to Algorithm 2 because the closure operator might generate constraints which range over all existing activities. For example, the constraint INIT(a) implies any constraint PRECEDENCE(a,X) where X is an arbitrary activity.

Even if Algorithm 2 reduces the number of candidate solutions by excluding all those non fulfilling conditions 1., 2., and 3., it is not guaranteed to return a unique solution. If the number of solutions provided by the procedure is too high for human intelligibility, the optimality condition could be further refined by inducing a preference order in the returned solution. For example, one can be interested in being reported first the solutions with the lower number of constraints, or with certain Declare templates. The advantage of our approach is precisely in the possibility to implement off-the-shelves optimisation strategies, where—adapting the $is\_optimal(S)$ function or even the definition of the closure operator—the developer can easily experiment with different model fitness functions.

In order to better clarify the approach, we apply it to a very simple example. Consider the sets of positive and negative traces composed by only one trace each: $L^+ = \{\mathsf{bac}\}$ and $L^- = \{\mathsf{ab}\}$. The alphabet of activities is clearly just $A = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$. Suppose we want to learn the most general model composed by only the Declare templates $D = \{$EXISTS, INIT$\}$.

DL: non ho inserito TRUE perchè non vorrei confondesse le idee quando si arriva a sceglie il set minimale

In this case, the set $D[A]$ can be easily elicited: $D[A] = \{$EXISTS(a), EXISTS(b), EXISTS(c), INIT(a), INIT(b), INIT(c)$\}$. Algorithm 1 elects the following compatible constraints: $compatibles = \{$EXISTS(a), EXISTS(b), EXISTS(c), INIT(b)$\}$, and emits $choices(\mathsf{ab}) = \{$EXISTS(c), INIT(b)$\}$.

In this simple case, the subsets of *compatibles* satisfying the first condition

(Line 3) of Algorithm 2 would be:

$$\begin{aligned} S_1 = &\quad \{\mathsf{EXISTS(c)}\} \\ S_2 = &\quad \{\mathsf{INIT(b)}\} \\ S_3 = &\quad \{\mathsf{EXISTS(c)}, \mathsf{INIT(b)}\} \\ S_4 = &\quad \{\mathsf{EXISTS(c)}, \mathsf{EXISTS(a)}\} \\ \ldots & \\ S_n = &\quad \{\mathsf{EXISTS(c)}, \mathsf{INIT(b)}, \mathsf{EXISTS(a)}\} \\ \ldots & \end{aligned}$$

As we are interested in the most general model, both the solutions $S_1 = \{\mathsf{EXISTS(c)}\}$ and $S_2 =\{\mathsf{INIT(b)}\}$ are valid. Note that these two solutions cannot be compared according to the definitions of generality or specificity because there exist traces (such as the unknown trace b) compliant with S_2 and non-compliant with S_1, i.e. there is no subset relation between $\mathcal{C}_{S_1}$ and $\mathcal{C}_{S_2}$.

On the contrary, if we are interested in the most specific set of constraints, the $is\_optimal(S)$ operation must return the maximal model {EXISTS(a), EXISTS(b), EXISTS(c), INIT(b)}. Finally, the redundancy check operated by the third selecting condition discards the constraint EXISTS(b), and correctly return the set S ={EXISTS(a), EXISTS(c), INIT(b)}. According to this model, the unknown trace b is negative and bca is positive.

5. Experimental evaluation

DL: x Fede, Chiara DFM, Sergio: qui bisognerebbe introdurre la metodologia per provare la validità del nostro approccio, cioè abbiamo preso questo esempio, abbiamo generato delle tracce, alcune positive, altre negative che però violano dei vincoli che in questo modello non sono presenti e poi abbiamo visto se riusciamo a re-impararli. Perchè proprio questi vincoli e non altri?

5.1. Motivating example

A bank carries out the following *loan application* process[3] in order to grant loans. The process starts when the *loan application* is received. In order to decide whether to *reject the application* or to *send the acceptance pack* to the customer for receiving the customer's official acceptance, the bank has to *assess* the *eligibility* of the loan. To this aim, the customer's *property* has to be *appraised* and the *loan risk assessed* by the bank. The bank can optionally *notify* the customer about the loan *approval*, of course in case the loan is not rejected. During the process the bank can also *receive positive* or *negative feedback* according to the experience of the loan requester. To ease the understanding of the loan application process, a Declare model of the process is reported in Fig. 1

DL: x Chiara DFM: La figura non corrisponde esattamente al modello usato da federico. Bisognerebbe: 1. rimuovere not-coexistense tra Send acceptance pack e Reject application. 2. aggiungere not-coexistence tra receive positive feedback e receive negative feedback

.

Among the processed loan application requests, some are considered as negative by the bank, while others as positive. In detail, the negative cases are the ones in which:

- the bank receives a negative feedback, although the acceptance pack is sent to the customer;
- the time required for carrying out the whole procedure is huge (this happens when the property appraisal is performed after the loan risk assessment).

Being aware of the process executions that deviate from its expectations (the negative cases), the bank would like to discover a process model of the loan

[3]The process is inspired by the Loan Application process reported in (2018, Dumas).

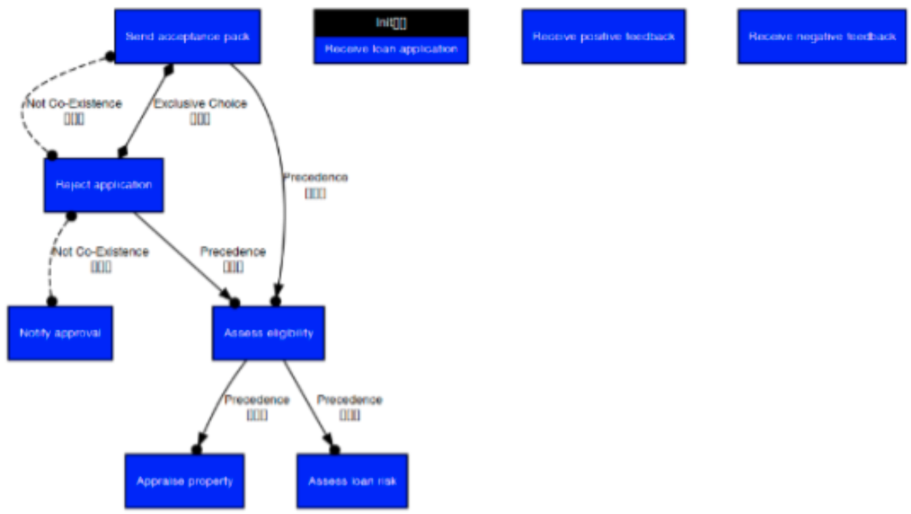

Figure 1: Loan approval declare process model

application procedure in which only the positive cases are included, while the deviant ones are excluded.

5.2. Datasets

DL: x Chiara DFM: questa sezione credo sia da ripopolare da capo perchè l'esempio è cambiato

5.3. Results

6. Related work

Process discovery is generally considered the most challenging task of process mining [16]. The majority of works in this field are focused on discovering a business process model from a set of input traces that are supposed compliant with it. In this sense, process discovery can be seen as the application of a machine learning technique to extract a grammar from a set of positive sample data. Angluin et al. [17] provide an interesting overview on this wide field. Differently from grammar learning, where the model is often expressed with automata, regular expressions or production rules, process discovery usually adopts formalisms that can express concurrency and synchronization in a more understandable way [18]. The language to express the model is a crucial point, which inevitably influences the learning task itself. Indeed, the two macro-categories of business process discovery approaches differ precisely by the type of language to express the model: procedural approaches envisage to uncover structured processes, whereas declarative ones are more suitable for unstructured models. Well known examples of procedural process discoverer are the ones presented in the works [4, 6, 7, 19, 8, 20]. In particular, the α-algorithm [4] is one of the first and most famous process discovery approaches. Its simple structure does not allow the discovery of complicated routing constructs and can be negatively influenced by the presence of noise in the log. Definitely more robust techniques are the heuristics miner [5] and the fuzzy miner [6], which can deal with unbalanced, incomplete, or noisy logs. The work [7] seeks a overfitting/underfitting balance between different parts of the extracted model by means of a two-step approach. The first step builds a transition system from the observation of occurrence, sequence and multi-set of activities in the log's traces. The second step converts the transition system into a Petri net, which can easily express the procedural nature of the business process. In [21], Augusto et al. present an extensive review of the procedural approaches to process discovery with a BPMN or Petri net output. The comparison between the various tools is conducted on the basis of a set of rather popular and well-established metrics: namely fitness, precision, generalization and complexity. The result of this comparison highlights how Inductive Miner [8], Evolutionary Tree Miner [19], and Split Miner [20] outperform all the other approaches, despite being rather limited when dealing with large-scale unfiltered logs. Like most procedural approaches to process discovery, all the works described so far consider deviant examples as a form of non-informative noise in the log, which should be separated from the rest of the log and disregarded.

Traditional declarative approaches to process discovery stem from the necessity of a more friendly language to express loosely-structured processes. Indeed—

as also pointed out by [16]—process models are sometimes less structured than one could expect. The application of a procedural discovery could produce spaghetti-models. In that case, a declarative approach is more suitable to briefly list all the required or prohibited behaviours in a business process. Similarly to our technique, the one exposed by Maggi et al. in [22] starts by considering the set of all activities in the log and building a set of all possible candidate Declare constraints. This work stems from the idea that Apriori-like approaches—such as sequence mining [23] and episode mining [24]—can discover local patterns in a log, but not rules representing prohibited behaviours and choices. Therefore, differently from our algorithm, the candidate Declare constraints are then translated into the equivalent Linear Temporal Logic (LTL) and checked (one at a time) against all the log content employing the technique of [25]. The process continue until certain levels of recall and specificity are reached. The performance of this technique is improved in [16] with an interesting two-step approach to both reduce the search space of candidate constraints and exclude from the model those LTL formulas that are vacuously satisfied. Also the work [26] by Schunselaar et al. proposes a model refinement to efficiently exclude vacuously satisfied constraints. Interestingly, instead of learning the model from the structure of a single trace, this approach works on sets of traces in the event log. The MINERFul approach described in [27] proposes to employ four metrics to guide the declarative discovery approach: support, confidence and interest factor for each constraint w.r.t. the log, and the possibility to include in the search space constraints on prohibited behaviours. Particularly relevant for our purposes is the work by Di Ciccio et al. [14], who focus on refining Declare models to remove the frequent redundancies and inconsistencies. The algorithms and the hierarchy of constraints described in that work were particularly inspiring to define our discovery procedure. Similarly to the procedural approaches, all the declarative ones described so far do not deal with negative example, although the vast majority of them envisage the possibility to discard a portion of the log by setting thresholds on the value of specific metrics that the discovered model should satisfy.

In the wider field of grammar learning, the foundational work by Gold [28] showed how negative examples are crucial to distinguish the right hypothesis among an infinite number of grammars that fit the positive examples. Both positive and negative examples are required to discover a grammar with perfect accuracy. Since process discovery does not usually seek perfection, but only a good level recall and specificity, it is not surprising that many procedural and declarative discoverers disregard the negative examples. Nonetheless, in this work we instead claim that negative traces are extremely important when learning declarative process models. Among traditional grammar learning approaches, the ones by Angluin [29] and Mooney [30] are particularly relevant for our work. The article [29] focuses on identifying an unknown model referred as ?regular set? and represented through Deterministic Finite-state Acceptor (DFA). Coherently with Gold?s theory [28], Angluin propose a learning algorithm that starts from input examples of the regular set?s members and nonmembers. The learning process is realized through the construction of an ?observation table?. The approach of Mooney et al. [30] shows three different algorithms to learn Conjunctive Normal Form (CNF), Disjunctive Normal Form (DNF) and Decision trees from a set of positive and negative examples.

The information contained in the negative examples is actively used in a subset of the declarative process discovery approaches [9, 10, 31, 32]. All these works can be reconnected to the basic principles of the Inductive Constraint Logic (ICL) algorithm [33], whose functioning principle is intrinsically related to the availability of both negative and positive examples. The Declarative Process Model Learner (DPML) described in [9] by Lamma et al. focuses on learning integrity constraints expressed as logical formulas. The constraints are later translated into an equivalent construct of the declarative graphical language DecSerFlow [34]. Similarly to this approach, the DecMiner tool described in [10], learns a set of SCIFF rules [35] which correctly classify an input set of labelled examples. Such rules are then translated into ConDec constraints [36]. An important difference w.r.t our approach is precisely in the fact that [10] expressly requires negative examples to perform the classification, whereas the algorithm we propose can work even in absence of counterexamples.DPML has been later used in [31] to extract integrity constraints, then converted into Markov Logic formulas. The weight of each formula is determined with a statistical relational learning tool. Taking advantage of the negative examples, the approach of [31] is improved in [32], thus obtaining significantly better results than other process discovery techniques. Since for all these works the availability of negative examples is crucial, recent years have seen the development of synthetical log generators able to produce not only positive but also negative process cases [37, 38, 18, 39, 40]. DL: right?

Particularly related to our approach are the works by Neider et al. [41], Camacho et al. [42], and Reiner [43] where a SAT-based solver is employed to learn a simple set of LTL formulas consistent with an input data set of positive and negative examples. In particular, Neider et al. [41]employ decision tree to improve the performance and manage large example sets; Camacho et al. [42] exploit the correspondence of LTL formulae with Alternating Finite Automata (AFA); whereas Reiner uses partial Directed Acyclic Graphs (DAGs) to decompose the search space into smaller subproblems. The concept of negative example used in this work could be related to both the definitions of syntactical and semantic noise of [44]. In particular, besides being able to extract relevant syntactic information that characterise the positive examples w.r.t. negative, our approach could also be useful to deal with the semantic concept of modification noise i.e., the semantic difference between traces from the same process model, which has been partially or totally modified at a certain point in time.

It is important to underline that also a limited number of procedural approaches envisage the need for taking into account the information contained into the negative examples (when they are available). In particular, the AGNEs tool described in [18] increases the dimension of an event log with artificially generated negative examples, then Inductive Logic Programming (ILP) multi-relational classification is used to discover a Perti net model. Negative examples are generated in a rather syntactical way, by considering each trace as a sequence of activities, and in each position of such sequence, what are the activities that are never observed to follow. ILP is also used in [45], where the authors suppose a set of negative examples provided by domain experts. The approach uses partial-order planning to discover a structured model. More recently, the works [46, 47] showed how synthetically generated traces can be employed to improve the robustness of the compliance monitor task.

Deviant cases - intended as traces whose sequence of activities deviates from the expected behaviour - are the subject of deviance mining approaches reviewed and evaluated by Nguyen et al. in [13]. Some applications of deviance mining tend to highlight the differences between models discovered from deviant and non-deviant traces [48, 49]. Other works intend deviance mining as a classification task, where the miner is required to identify normal and deviant traces given a set of examples. The classification inherently causes the discovery of patterns which distinguish different types of traces. In this sense, deviance mining is particularly similar to sequence classification. The discovered patterns can be based on the simple frequency of individual activities as in [50, 51], their co-occurrence as in [52], or the occurrence of specific subsequences [53, 54, 55].

Finally, the performance of our approach could be boosted through a parallel approach as the one presented in [56]—related to compliance checking—and [57]—focused on process discovery. Both these works envisage two possible directions to decompose the process mining task: the set of constraints (to be checked for compliance or learned), and the business log. In this regard, the algorithm presented in this work could easily adopt the first kind of partitioning, whereas the second might be more challenging.

7. Discussion

DL: x Dani: sezione da raffinare alla fine. Molte riflessioni penso non abbiano più senso.

It is worth to underline that a declarative process discoverer taking advantage of explicitly defined positive and negative examples is not necessarily an alternative to procedural discovery techniques. Indeed in some cases, when correct thresholds and language biases are adopted, procedural discoverers have the great advantage to provide the user with a rather easy-to-understand definition of the process model. Nonetheless, the informative content provided by those process cases that are discarded by procedural discoverer (e.g., in order to avoid spaghetti models) can still be extremely important. A declarative process discoverer taking advantage of explicitly defined positive and negative examples can extract valuable information from such discarded traces and synthesize it into declarative constraints. The resulting output would be an hybrid procedural/declarative process model, showing a simple and handy structured representation of the main business process together with a set of declarative constraints. The goal of such constraints would be to account for less frequent deviances and prohibited behaviours in a much more synthetic and easy-to-understand way with respect to an equivalent spaghetti-like procedural formulation.

Furthermore, such hybrid solution could also greatly simplifying the elicitation of long-term dependencies between activities that occur at the beginning of the process and those carried out towards the end. Indeed, the structured nature of procedural approaches makes them not properly suitable to express such dependencies. One current way to tackle such issue is through the employment of global variables and if statements to control the execution flow of each instance. Kalenkova et al. (2020, Kalenkova) propose e process discovery technique devoted to repair free-choice procedural workflows with additional modeling constructs, which can more easily capture non-local dependencies. Nonetheless, since such additional constraints are intended to preserve the procedural nature of the model, the result may increase its complexity and ultimately affect its readability. An hybrid procedural/declarative model formulation would maintain a structured form to express the model while integrating it with handy declarative long-term constraints involving activities occurring far from each other in the workflow.

For example, consider the load application process depicted in Fig. ... The procedural nature of the model makes it particularly easy to understand for a human subject. Nonetheless, if we want to add a rather simple constraint such as: "Do not ask customer feedback if the application was cancelled", a substantial modification of the model is required. Indeed, the model must state that the branch including "Cancel application" (and then "Notify cancellation") must be followed by the END event, whereas any other branch can still include "Ask for customer feedback" before the END event. In practice, adding a constraint of such kind force us to add a alternative branch towards the end of the model. If many alternative paths are present in the model and we need to add many conditions of this kind, the diagram in Fig.. may quickly turn into a spaghetti model.

A more compact and readable way to apply this modification is to maintain the present process model structure and equipping it with a declarative elicitation of the prohibited behaviours. In the considered case, the simple inclusion of a constraint such as "NOT PRECEDENCE(Cancel application, Ask for customer feedback)" prevents all forbidden paths.

This idea of a hybrid procedural/declarative model formulation has been explored by various works and proved to be particularly effective in the field of medical clinical guidelines (2009a, Bottrighi) (2009b, Bottrighi) (2011, Bottrighi). A wider landscape of applications is considered by Maggi et al. in the work (2018b, Maggi). The technique starts from a structured business process model and adopts non-deterministic finite-state automaton manipulation to detect violations of compliance requirements expressed as temporal declarative rules.

8. Conclusion

References

[1] W. van der Aalst, A. Adriansyah, A. K. A. de Medeiros, F. Arcieri, T. Baier, T. Blickle, J. C. Bose, P. van den Brand, R. Brandtjen, J. Buijs, A. Burattin, J. Carmona, M. Castellanos, J. Claes, J. Cook, N. Costantini, F. Curbera, E. Damiani, M. de Leoni, P. Delias, B. F. van Dongen, M. Dumas, S. Dustdar, D. Fahland, D. R. Ferreira, W. Gaaloul, F. van Geffen, S. Goel, C. Günther, A. Guzzo, P. Harmon, A. ter Hofstede, J. Hoogland,

J. E. Ingvaldsen, K. Kato, R. Kuhn, A. Kumar, M. La Rosa, F. Maggi, D. Malerba, R. S. Mans, A. Manuel, M. McCreesh, P. Mello, J. Mendling, M. Montali, H. R. Motahari-Nezhad, M. zur Muehlen, J. Munoz-Gama, L. Pontieri, J. Ribeiro, A. Rozinat, H. Seguel Pérez, R. Seguel Pérez, M. Sepúlveda, J. Sinur, P. Soffer, M. Song, A. Sperduti, G. Stilo, C. Stoel, K. Swenson, M. Talamo, W. Tan, C. Turner, J. Vanthienen, G. Varvaressos, E. Verbeek, M. Verdonk, R. Vigo, J. Wang, B. Weber, M. Weidlich, T. Weijters, L. Wen, M. Westergaard, M. Wynn, Process mining manifesto, in: F. Daniel, K. Barkaoui, S. Dustdar (Eds.), Business Process Management Workshops, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 169–194.

[2] D. Fahland, D. Lübke, J. Mendling, H. A. Reijers, B. Weber, M. Weidlich, S. Zugal, Declarative versus imperative process modeling languages: The issue of understandability, in: BMMDS/EMMSAD, Vol. 29 of Lecture Notes in Business Information Processing, Springer, 2009, pp. 353–366.

[3] F. M. Maggi, A. Marrella, G. Capezzuto, A. Armas-Cervantes, Explaining non-compliance of business process models through automated planning, in: ICSOC, Vol. 11236 of Lecture Notes in Computer Science, Springer, 2018, pp. 181–197.

[4] W. M. P. van der Aalst, T. Weijters, L. Maruster, Workflow mining: Discovering process models from event logs, IEEE Trans. Knowl. Data Eng. 16 (9) (2004) 1128–1142.

[5] A. J. M. M. Weijters, W. M. P. van der Aalst, Rediscovering workflow models from event-based data using little thumb, Integr. Comput. Aided Eng. 10 (2) (2003) 151–162.

[6] C. W. Günther, W. M. P. van der Aalst, Fuzzy mining - adaptive process simplification based on multi-perspective metrics, in: BPM, Vol. 4714 of Lecture Notes in Computer Science, Springer, 2007, pp. 328–343.

[7] W. M. P. van der Aalst, V. A. Rubin, H. M. W. Verbeek, B. F. van Dongen, E. Kindler, C. W. Günther, Process mining: a two-step approach to balance between underfitting and overfitting, Software and Systems Modeling 9 (1) (2010) 87–111.

[8] Q. Guo, L. Wen, J. Wang, Z. Yan, P. S. Yu, Mining invisible tasks in non-free-choice constructs, in: BPM, Vol. 9253 of Lecture Notes in Computer Science, Springer, 2015, pp. 109–125.

[9] E. Lamma, P. Mello, F. Riguzzi, S. Storari, Applying inductive logic programming to process mining, in: ILP, Vol. 4894 of Lecture Notes in Computer Science, Springer, 2007, pp. 132–146.

[10] F. Chesani, E. Lamma, P. Mello, M. Montali, F. Riguzzi, S. Storari, Exploiting inductive logic programming techniques for declarative process mining, Trans. Petri Nets Other Model. Concurr. 2 (2009) 278–295.

[11] W. M. P. van der Aalst, Process Mining - Discovery, Conformance and Enhancement of Business Processes, Springer, 2011.

[12] G. M. Spreitzer, S. Sonenshein, Toward the construct definition of positive deviance, American Behavioral Scientist 47 (6) (2004) 828–847. `doi:10.1177/0002764203260212`.

[13] H. Nguyen, M. Dumas, M. L. Rosa, F. M. Maggi, S. Suriadi, Business process deviance mining: Review and evaluation, CoRR abs/1608.08252.

[14] C. D. Ciccio, F. M. Maggi, M. Montali, J. Mendling, Resolving inconsistencies and redundancies in declarative process models, Inf. Syst. 64 (2017) 425–446.

[15] V. Lifschitz, What is answer set programming?, in: AAAI, AAAI Press, 2008, pp. 1594–1597.

[16] F. M. Maggi, R. P. J. C. Bose, W. M. P. van der Aalst, Efficient discovery of understandable declarative process models from event logs, in: CAiSE, Vol. 7328 of Lecture Notes in Computer Science, Springer, 2012, pp. 270–285.

[17] D. Angluin, C. H. Smith, Inductive inference: Theory and methods, ACM Comput. Surv. 15 (3) (1983) 237–269.

[18] S. Goedertier, D. Martens, J. Vanthienen, B. Baesens, Robust process discovery with artificial negative events, J. Mach. Learn. Res. 10 (2009) 1305–1340.

[19] S. J. J. Leemans, D. Fahland, W. M. P. van der Aalst, Discovering block-structured process models from event logs - A constructive approach, in: Petri Nets, Vol. 7927 of Lecture Notes in Computer Science, Springer, 2013, pp. 311–329.

[20] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, Split miner: Discovering accurate and simple business process models from event logs, in: ICDM, IEEE Computer Society, 2017, pp. 1–10.

[21] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, F. M. Maggi, A. Marrella, M. Mecella, A. Soo, Automated discovery of process models from event logs: Review and benchmark, IEEE Trans. Knowl. Data Eng. 31 (4) (2019) 686–705.

[22] F. M. Maggi, A. J. Mooij, W. M. P. van der Aalst, User-guided discovery of declarative process models, in: CIDM, IEEE, 2011, pp. 192–199.

[23] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: VLDB, Morgan Kaufmann, 1994, pp. 487–499.

[24] H. Mannila, H. Toivonen, A. I. Verkamo, Discovery of frequent episodes in event sequences, Data Min. Knowl. Discov. 1 (3) (1997) 259–289.

[25] W. M. P. van der Aalst, H. T. de Beer, B. F. van Dongen, Process mining and verification of properties: An approach based on temporal logic, in: OTM Conferences (1), Vol. 3760 of Lecture Notes in Computer Science, Springer, 2005, pp. 130–147.

[26] D. M. M. Schunselaar, F. M. Maggi, N. Sidorova, Patterns for a log-based strengthening of declarative compliance models, in: IFM, Vol. 7321 of Lecture Notes in Computer Science, Springer, 2012, pp. 327–342.

[27] C. D. Ciccio, M. H. M. Schouten, M. de Leoni, J. Mendling, Declarative process discovery with minerful in prom, in: BPM (Demos), Vol. 1418 of CEUR Workshop Proceedings, CEUR-WS.org, 2015, pp. 60–64.

[28] E. M. Gold, Language identification in the limit, Inf. Control. 10 (5) (1967) 447–474.

[29] D. Angluin, Learning regular sets from queries and counterexamples, Inf. Comput. 75 (2) (1987) 87–106.

[30] R. J. Mooney, Encouraging experimental results on learning CNF, Mach. Learn. 19 (1) (1995) 79–92.

[31] E. Bellodi, F. Riguzzi, E. Lamma, Probabilistic logic-based process mining, in: CILC, Vol. 598 of CEUR Workshop Proceedings, CEUR-WS.org, 2010.

[32] E. Bellodi, F. Riguzzi, E. Lamma, Statistical relational learning for workflow mining, Intell. Data Anal. 20 (3) (2016) 515–541.

[33] L. D. Raedt, W. V. Laer, Inductive constraint logic, in: ALT, Vol. 997 of Lecture Notes in Computer Science, Springer, 1995, pp. 80–94.

[34] W. M. P. van der Aalst, M. Pesic, Decserflow: Towards a truly declarative service flow language, in: WS-FM, Vol. 4184 of Lecture Notes in Computer Science, Springer, 2006, pp. 1–23.

[35] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, P. Torroni, Verifiable agent interaction in abductive logic programming: The SCIFF framework, ACM Trans. Comput. Log. 9 (4) (2008) 29:1–29:43.

[36] M. Pesic, W. M. P. van der Aalst, A declarative approach for flexible business processes management, in: Business Process Management Workshops,

Vol. 4103 of Lecture Notes in Computer Science, Springer, 2006, pp. 169–180.

[37] F. Chesani, C. D. Francescomarino, C. Ghidini, D. Loreti, F. M. Maggi, P. Mello, M. Montali, V. Skydanienko, S. Tessaris, Towards the generation of the "perfect" log using abductive logic programming, in: CILC, Vol. 2396 of CEUR Workshop Proceedings, CEUR-WS.org, 2019, pp. 179–192.

[38] D. Loreti, F. Chesani, A. Ciampolini, P. Mello, Generating synthetic positive and negative business process traces through abduction, Knowl. Inf. Syst. 62 (2) (2020) 813–839.

[39] T. Stocker, R. Accorsi, Secsy: A security-oriented tool for synthesizing process event logs, in: BPM (Demos), Vol. 1295 of CEUR Workshop Proceedings, CEUR-WS.org, 2014, p. 71.

[40] K. M. van Hee, Z. Liu, Generating benchmarks by random stepwise refinement of petri nets, in: ACSD/Petri Nets Workshops, Vol. 827 of CEUR Workshop Proceedings, CEUR-WS.org, 2010, pp. 403–417.

[41] D. Neider, I. Gavran, Learning linear temporal properties, in: FMCAD, IEEE, 2018, pp. 1–10.

[42] A. Camacho, S. A. McIlraith, Learning interpretable models expressed in linear temporal logic, in: ICAPS, AAAI Press, 2019, pp. 621–630.

[43] H. Riener, Exact synthesis of LTL properties from traces, in: FDL, IEEE, 2019, pp. 1–6.

[44] C. W. Günther, Process mining in flexible environments, Ph.D. thesis, Technische Universiteit Eindhoven (2009).

[45] H. M. Ferreira, D. R. Ferreira, An integrated life cycle for workflow management based on learning and planning, Int. J. Cooperative Inf. Syst. 15 (4) (2006) 485–505.

[46] S. K. L. M. vanden Broucke, J. D. Weerdt, J. Vanthienen, B. Baesens, Determining process model precision and generalization with weighted artificial negative events, IEEE Trans. Knowl. Data Eng. 26 (8) (2014) 1877–1889.

[47] S. K. L. M. vanden Broucke, Advances in process mining: Artificial negative events and other techniques, Ph.D. thesis, Katholieke Universiteit Leuven (2014).

[48] S. Suriadi, R. Mans, M. T. Wynn, A. Partington, J. Karnon, Measuring patient flow variations: A cross-organisational process mining approach, in: AP-BPM, Vol. 181 of Lecture Notes in Business Information Processing, Springer, 2014, pp. 43–58.

[49] A. Armas-Cervantes, P. Baldan, M. Dumas, L. García-Bañuelos, Behavioral comparison of process models based on canonically reduced event structures, in: BPM, Vol. 8659 of Lecture Notes in Computer Science, Springer, 2014, pp. 267–282.

[50] S. Suriadi, M. T. Wynn, C. Ouyang, A. H. M. ter Hofstede, N. J. van Dijk, Understanding process behaviours in a large insurance company in australia: A case study, in: CAiSE, Vol. 7908 of Lecture Notes in Computer Science, Springer, 2013, pp. 449–464.

[51] A. Partington, M. T. Wynn, S. Suriadi, C. Ouyang, J. Karnon, Process mining for clinical processes: A comparative analysis of four australian hospitals, ACM Trans. Management Inf. Syst. 5 (4) (2015) 19:1–19:18.

[52] J. Swinnen, B. Depaire, M. J. Jans, K. Vanhoof, A process deviation analysis - A case study, in: Business Process Management Workshops (1), Vol. 99 of Lecture Notes in Business Information Processing, Springer, 2011, pp. 87–98.

[53] R. P. J. C. Bose, W. M. P. van der Aalst, Discovering signature patterns from event logs, in: CIDM, IEEE, 2013, pp. 111–118.

[54] D. Lo, S. Khoo, C. Liu, Efficient mining of iterative patterns for software specification discovery, in: KDD, ACM, 2007, pp. 460–469.

[55] M. L. Bernardi, M. Cimitile, C. D. Francescomarino, F. M. Maggi, Do activity lifecycles affect the validity of a business rule in a business process?, Inf. Syst. 62 (2016) 42–59.

[56] D. Loreti, F. Chesani, A. Ciampolini, P. Mello, A distributed approach to compliance monitoring of business process event streams, Future Gener. Comput. Syst. 82 (2018) 104–118.

[57] F. M. Maggi, C. D. Ciccio, C. D. Francescomarino, T. Kala, Parallel algorithms for the automated discovery of declarative process models, Inf. Syst. 74 (Part) (2018) 136–152.