Discovering Business Process models expressed as DNF or CNF formulae of Declare constraints

Federico Chesani[1,*,†], Chiara Di Francescomarino[2,†], Chiara Ghidini[2,†], Daniela Loreti[1,†], Fabrizio Maria Maggi[3,†], Paola Mello[1,†], Marco Montali[3,†], Elena Palmieri[1,†] and Sergio Tessaris[3,†]

[1]*DISI - University of Bologna, Italy*
[2]*Fondazione Bruno Kessler, Trento, Italy*
[3]*Free University of Bozen/Bolzano, Italy*

Abstract

In the field of Business Process Management, the Process Discovery task is one of the most important and researched topics. It aims to automatically learn process models starting from a given set of logged execution traces. The majority of the approaches employ procedural languages for describing the discovered models, but declarative languages have been proposed as well. In the latter category there is the Declare language, based on the notion of constraint, and equipped with a formal semantics on LTL*f*. Also, quite common in the field is to consider the log as a set of positive examples only, but some recent approaches pointed out that a binary classification task (with positive and negative examples) might provide better outcomes.

In this paper, we discuss our preliminary work on the adaptation of some existing algorithms for Inductive Logic Programming, to the specific setting of Process Discovery: in particular, we adopt the Declare language with its formal semantics, and the perspective of a binary classification task (i.e., with positive and negative examples).

Keywords

Process Discovery, Declare, Inductive Logic Programming

1. Introduction and motivations

The research field of Business Process Management (BPM) was initiated more than 20 years ago, and it is now a mature discipline that focuses on the many aspects related to the Business Processes and IT-solutions (but not only) for BPM. In particular, the mining of Business Processes (with the three main tasks of discovery, conformance checking and enhancement [1]) is a sub-field aimed to support decision-making in complex industrial and corporate domains. *Process*

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
*Corresponding author.
†These authors contributed equally.
federico.chesani@unibo.it (F. Chesani); dfmchiara@fbk.eu (C. Di Francescomarino); ghidini@fbk.eu (C. Ghidini); daniela.loreti@unibo.it (D. Loreti); maggi@inf.unibz.it (F. M. Maggi); paola.mello@unibo.it (P. Mello); montali@inf.unibz.it (M. Montali); e.palmieri@unibo.it (E. Palmieri); tessaris@inf.unibz.it (S. Tessaris)
0000-0003-1664-9632 (F. Chesani); 0000-0002-0264-9394 (C. Di Francescomarino); 0000-0003-1563-4965 (C. Ghidini); 0000-0002-6507-7565 (D. Loreti); 0000-0002-9089-6896 (F. M. Maggi); 0000-0002-5929-8193 (P. Mello); 0000-0002-8021-3430 (M. Montali); 0000-0001-5176-8843 (E. Palmieri); 0000-0002-3156-2669 (S. Tessaris)

discovery in particular deals with the automatic learning of a process model starting from a given set of logged traces, each one representing the execution of a business case. Accordingly to the language employed to represent the output process model, discovery algorithms fall into procedural or declarative techniques. The latter family of techniques—which represent the context of this work—return the model as a set of constraints (equipped with a declarative, logic-based semantics) that must be fulfilled by the traces at hand.

The Declare language [2] is one of the most used declarative languages, and consists of a set of template constraints that can be instantiated (grounded) with the process activities. The formal semantics of each constraint is based on LTL, and a process model is defined as a conjunction of grounded templates: hence, Declare does not (fully) support Conjunctive/Disjunctive Normal Forms. Moreover, the majority of the discovery algorithms conceive this task as a one-class supervised learning technique, while fewer works (e.g. [3, 4, 5, 6]) intend model-extraction as a two-class supervised task—provided that the log has been partitioned into two sets, usually named *positive* and *negative examples*.

In the field of Logic Programming, Inductive Logic Programming (ILP) is a well known family of learning techniques that address the learning task in terms of a binary classification problem. Noteworthy algorithms are the one proposed by Quinlan [7] and its subsequent generalization to DNF/CNF models proposed by Mooney [8]. There, the objective is to learn a logic-based description of two sets of ground facts.

In this paper, we discuss our preliminary investigations about the possibility of adapting the approach proposed by Mooney, to the specific setting of BP Discovery task, and Declare as the target language for describing the learned models. Hence, our approach will start from a log partitioned into two sets (positive and negative labeled traces), and the outcome will be a conjunction/disjunction of grounded Declare templates. The resulting model should be able then to discriminate positive from negative traces, as well as to properly classify novel traces. The proposed algorithms have been implemented in Prolog, and some preliminary testing has been done to evaluate the correctness of our approach, and the performances of the current implementation.

The paper is organized as follows: in Section 2 we provide some background on the field of Process Discovery and the Declare language, and on the original algorithms proposed by Mooney, from which we took inspiration. In Section 3 we introduce our extension/adaptation of the existing algorithms to the specific setting, and in Section 4 we experimentally evaluate our approach. In Section 5 we discuss some related works, while in Section 6 we discuss some conclusion remarks and future works.

2. Preliminaries

2.1. Process Discovery, and Declare

According to the Business Process Mining Manifesto [1], Process Discovery aims to "discover" a model of a process using the knowledge deduced from the event log without the use of a-priori information. A distinction between the many discovery algorithms can be done on the basis of the language adopted to output the learned process model. Indeed, two main categories of modeling languages can be easily identified: *procedural languages* and *declarative*

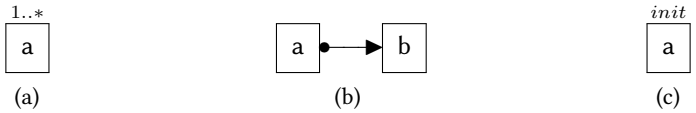

Figure 1: Examples of Declare constraints

languages. Procedural languages model the processes in terms of constructs like sequence, parallel executions, (exclusive) choices between different execution paths, etc. Declarative languages instead are more focused on the properties that each process execution should exhibit. While the former languages usually adopt a closed-approach (allowed execution paths are explicitly stated; everything else is forbidden by default), the latter approaches are usually based on a open-approach (whatever is not explicitly prohibited can be executed and is compliant with the process model). Notable examples of procedural modeling languages are YAWL [9] and BPMN [10, 11], while famous examples of declarative approaches are Declare [2] and Dynamic Condition Response Graphs [12].

Declare [2, 13] is one of the most well-established declarative process modeling languages. A process is modeled through a conjunction of constraints that affect the presence/absence of activity executions, and possibly the relative orders between activities. To this end, Declare provides a set of constraint templates that can be instantiated (grounded) by specifying the activities. Two main categories of constraint templates (or simply constraints, in the following) are available: *existence constraints* that involve only one activity, and *relation constraints*, that involve two of them. An example of an existence constraint is existence(a) (Figure 1a), that specifies that activity a must be executed at least once in every process instance. An example of relation constraint is response(a,b) (Fig. 1b): it states that, if activity a is executed, then it must be followed by the execution of activity b. Notice that the constraint is "triggered" by the execution of the activity a (graphically, a filled circle marks the triggering event), and that can be also vacuously satisfied if a is not executed. Each Declare template has been equipped with a formal semantics [2] in LTLf: for example, response(a,b) correspond to the expression $\Box(a \Rightarrow \Diamond b)$.

Some constraints are in a *subsumption* relation each other, meaning that traces satisfying a constraint will satisfy also another constraint, but not the opposite. For example, the init(a) constraint shown in Figure 1c states that every trace must begin with the execution of activity a: it is straightforward to see that every trace compliant with init(a) will be compliant also with existence(a), but not the other way round. Formally, as defined in [14], given a finite set A of activities, and A^* the set of sequences that can be generated from A, a constraint template C is *subsumed* by another constraint C', written C $\sqsubseteq$ C', if for every trace $t \in A^*$ and every parameter assignment γ_n from the parameters of C to tasks in A, whatever t complies with C/γ_n, then t also satisfies C'/γ_n. This hierarchy allows us to make specialization or generalization steps, as explained in the next section.

2.2. Learning CNF and DNF models: Mooney's approach

Mooney in [8], proposed two algorithms for learning Conjunctive and Disjunctive Normal Forms of logic models, respectively, starting from a labeled dataset. The DNF learner, called PFoil, is a propositional version of Quinlan's Foil [7], and it is composed of two cycles. The inner cycle focuses on the generation of terms, conjunctions of feature-value pairs that necessarily exclude all the negative examples in the event log. The outer cycle adds the returned clauses in disjunction to the model and ends when it covers all the positive traces. The next feature-value pair to add to the term is chosen calculating its *DNF gain*, a score based on the total number of positive and negative examples in the event log and on the number of covered ones. Intuitively, the best pair will be the one that covers more positive traces while excluding more negative ones.

Algorithm 1 PFoil: DNF learner by Mooney

```
Let Pos be all the positive examples.
Let DNF be empty.
Until Pos is empty do:
    Let Neg be all the negative examples.
    Set Term to empty and Pos2 to Pos.
    Until Neg is empty do:
        Choose the feature-value L that maximizes the function DNF-gain(L, Pos2, Neg).
        Add L to Term.
        Remove from Neg all the examples that do not satisfy L.
        Remove from Pos2 all the examples that do not satisfy L.
    Add Term as one term of DNF
    Remove from Pos all the examples that satisfy Term.
Return DNF.

Function DNF-gain(C, Pos, Neg)
    Let P be the number of examples in Pos
    Let N be the number of examples in Neg
    Let p be the number of examples in Pos that satisfy C
    Let n be the number of examples in Neg that satisfy C
    Return p×(log10(p/(p+n)) − log10(P/(P+N))).
```

The dual version of the algorithm outputs a CNF model. With respect to the Algorithm 1, the inner cycle focuses on the positives, while the outer cycle iterate over the negative exmaples. Consequently, also the gain function is adapted, and it is reported in Eq. 1.

$$\textit{CNF-gain} = n \times \left(log_{10} \frac{n}{p+n} - log_{10} \frac{N}{P+N} \right) \tag{1}$$

In this case though, p and n are the number of positive and negative traces that do not satisfy the feature-value pair in exam.

3. Applying Mooney's algorithm to the discovery of Declare process models

In this work we extend and adapt Mooney's algorithm to the process discovery task. With respect to the existing approaches for process discovery, here we consider the log as split in two (disjoint) classes of traces or, in other words, we conceive the discovery as a binary classification task. The goal, then, is to identify which are characteristics that allow us to discern if a not-labeled trace belongs to one class or another.

With respect to the original proposal by Mooney, we adapt the algorithm in several ways. First of all, the target language is Declare; secondly, the examples sets are indeed the traces belonging to a log, i.e. sequences of events that represent an execution of the process. Thirdly, when choosing the next constraint to be added in the resulting model, we introduce a further choice dimension (beside the gain function) by exploiting the subsumption relation between some Declare templates. Finally, we improve the algorithm for dealing with real logs and specific cases.

3.1. Declare as target language

Thanks to the declarative nature of Declare, and being the language based on the notion of constraints, it suffices to implement a specific test for checking when a trace satisfies or not a constraint. Our extended algorithm picks up constraints (rather than feature-value couples) from a list of candidates obtained by grounding the Declare constraint patterns.

Algorithm 2 Modified DNF Learner

```
Let Pos be all the positive traces.
Let DNF, ExcludedNeg and ExcludedPos be empty.
Until Pos is empty do:
    Let Neg be all the negative traces.
    Set Term to empty and Pos2 to Pos.
    Until Neg is empty do:
        If the list of candidate constraints is not empty:
            Choose the constraint C that maximizes the function DNF-gain(C, Pos2, Neg).
            Add C to Term.
            Remove from Neg all the traces that do not satisfy C.
            Remove from Pos2 all the traces that do not satisfy C.
        else:
            Set ExcludedNeg to Neg and Neg to empty.
    Remove from Pos all the traces that satisfy Term.
    If at least one positive trace satisfies Term:
        Add Term as one term of DNF
    Else:
        Set ExcludedPos to Pos and Pos to empty.
Return DNF.
```

The constraint that maximizes the gain function is chosen using Mooney's formula. In the DNF version of the algorithm, then, the chosen constraint is added to Term (in conjunction), thus performing a "specialization" step, since the resulting conjunction of constraints will exclude more negative traces but (possibly) also more positive ones. Similarly, the CNF algorithm will perform a generalization step, since adding a constraint in a disjunction will allow to possibly accept more positive and/or negative traces.

In the DNF algorithm, the inner cycle outputs a term that is a conjunction of constraints that rules out all the negative examples. It is possible, however, that a term rules out also all the positive examples. In such a case, the term would be redundant in the final model. We add a control step that checks if at least one positive example is satisfied, and only in that case the term is added to the model. Moreover, if a term does not allow any positive trace, the algorithm will never be able to find another term (if there exists such a term, the gain function would have selected proper constraints in earlier iterations). Therefore, it is wiser to stop the computation and to ignore the remaining positive traces. The same thing can happen in the CNF algorithm; if a clause, that has to satisfy all the positive traces, does not exclude any negative one, then it is discarded.

Example 1 shows a simple log, and one among the many possible Declare models. Each trace is represented as a Prolog structure, where the first argument is the trace identifier, while the second is a list of events. In turn, each event is described by the name of the process activity that has been executed, and the timestamp (an integer).

Example 1. *Traces labeled as positive examples:*
trace(tp1, [event(a,1), event(b,2), event(c,3), event(d,4)]).
trace(tp2, [event(a,1), event(b,2), event(b,3), event(c,4)]).
trace(tp3, [event(a,1), event(c,2), event(b,3), event(d,4)]).
trace(tp4, [event(k,1), event(c,2), event(a,3), event(d,4)]).
Traces labeled as negative examples:
trace(tn1, [event(b,1), event(c,2), event(e,3), event(d,4)]).
trace(tn2, [event(c,1), event(b,2), event(a,3), event(a,4)]).
A possible DNF model could be:

(existence(a) AND precedence(a,b)) OR existence(k)

Analogously, a CNF model would be:

(existence(a) OR existence(k)) AND precedence(a,b)

□

3.2. Exploiting the subsumtption hierarchy

Some Declare templates are in a subsumption relation with each other, as pointed out in [14]. For example, the init(a) constraint imposes that each trace should begin with the execution of activity a; consequently, any trace that satisfies such constraint will satisfies also the more

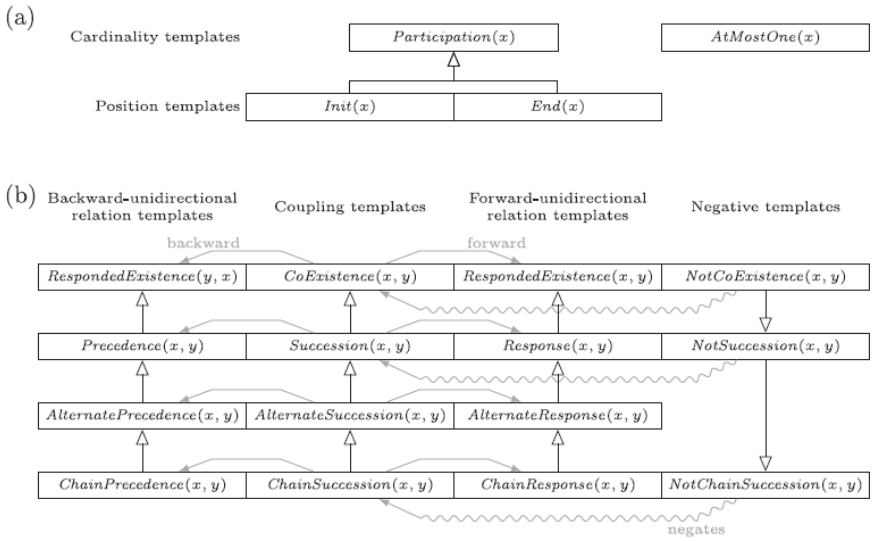

Figure 2: Subsumption map of Declare templates [14]. Note that *Participation(x)* corresponds to the template *existence(X)*, *End(x)* corresponds to *last(X)*, and *AtMostOne(x)* to *absence2(X)*

.

general constraint existence(a). In Figure 2 we report the subsumption hierarchy proposed in [14]. We exploit such relations in two ways.

First of all, the gain function will select candidates starting from two initial sets of constraints. As the inner cycle of the DNF specializes the current term, its starting set will contain the more general templates, accordingly to the sumbsumption hierarchy. Analogously, the CNF version will start considering the more specialized constraints.

Secondly, the specialization step (in the DNF algorithm) is extended as well: beside adding a new constraint in conjunction to the term, the subsumption relation allows us to specialize a constraint already in the term. Suppose for example that the current term constructed by the inner cycle is (existence(a)). The specialization step could then opt to add a new constraint in conjunction, for example responded_existence(b,c), or specialize the existing one, for instance, in init(a). The resulting models would be [existence(a) AND responded_existence(b,c)] in the former case, and [init(a)] in the latter.

Analogous consideration hold for the CNF algorithm, with the obvious difference that the subsumption hierarchy is explored towards the generalization.

3.3. Dealing with real-life logs

It might not be always possible to "perfectly" separate positive form negative examples. This because of two possible reasons: the Declare language provides a bias about the allowed LTL formulas, and to the best of our knowledge, there is not any proof of completeness of such language w.r.t. the classification task. Moreover, real-life logs might be inconsistent, i.e. a trace might have been labeled as positive and as negative at the same time. To cope with these exceptions, whenever our algorithms find negative traces that are impossible to exclude

or positive ones that cannot be covered, they simply remove them from the event log under consideration and continue with the discovery task. At the end, the ignored traces are returned together with the found model, if any.

Sometimes real-life logs contains positive traces only, and no negative examples are provided. Such case affects both the DNF and the CNF original algorithms, as the DNF version's termination condition is given by the set of negative examples becoming empty, whereas the CNF version would be stuck in an infinite loop as it would generate empty clauses. We designed our algorithm to be able to deal with such logs, and to return process models that just describe the positive traces.

4. Experimental evaluation

We implemented both the DNF and the CNF algorithms in Prolog. The core of both the algorithms is the predicate that chooses the next constraint to be added to the term. In the DNF version, at the first iteration over a term, the predicate chooses from the set of the most general constraints. In the subsequent iterations it will choose between the most general constraints (not yet in the term) and the specialization of an already selected constraint. In the CNF version, the same will happen, a part that the specialization step will bu substituted by the generalization one.

We evaluated both the algorithms against two logs: a synthetic, controlled log whose process model was already known, and a real-life event log (about a PAP test screening process), whose model was not known in advance.

4.1. The synthetic, controlled event log

The controlled event log contains a set of 64000 positive traces and three different sets containing respectively 10240, 12800 and 25600 negative ones, with 16 different activities. Each one of the negative example sets violates a single, specific constraint: hence for each log a constraint would be enough to discriminate between the positive and the negative traces.

All the negative examples contained in the first negative set can be ruled out by the constraint exclusive_choice(send_acceptance_pack, receive_negative_feedback). Both the DNF and the CNF algorithms returned the same model:

```
exclusive_choice(send_acceptance_pack, receive_negative_feedback)
```

The two remaining negative sets violate a precedence constraint grounded over different activities, affecting the overall process in different manner. The precedence(X, Y) template however is neither in the starting set of the DNF algorithm nor in the one of the CNF version. The second set of negative traces was found to be completely ruled out by not_chain_succession(assess_loan_risk, appraise_property) and chain_succession(receive_loan_application, appraise_property), which were the models returned by the algorithm, and indeed are correct w.r.t. the original violated constraint. The model returned by the CNF version with the third set of negative traces contained the expected constraint. On the other hand, the DNF version returned a correct but more complex model, composed of three constraints.

	DNF version	CNF version
Negative Set #1	1129	297
Negative Set #2	700	307
Negative Set #3	1648	522

Table 1
Average time (in seconds) of execution for the controlled event log

From a performance point of view, as visible in Table 1, the CNF version of the algorithm was always faster than the DNF one. This might be the consequence of the fact that, roughly speaking, the CNF and DNF algorithms proceed with the specialization/generalization of the returned model: thus, they start to explore different initial constraints. Figures 3 and 4 show the occupation of the global and local stacks right before the termination of the algorithms, when the final model has already been found.

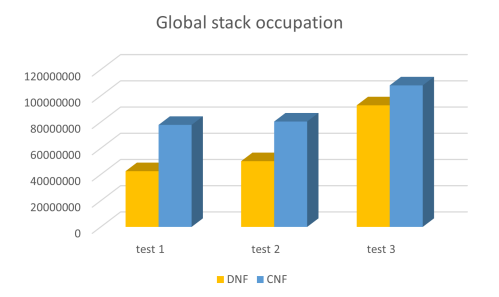

Figure 3: Global stack occupation right before the termination of the algorithms.

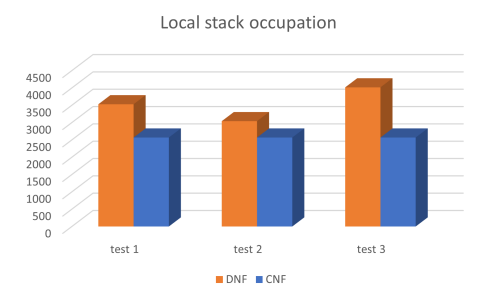

Figure 4: Local stack occupation right before the termination of the algorithms.

4.2. A real-life event log: the PAP test

Once performances have been assessed through the use of a synthetic controlled log, we evaluated the algorithms on a real-life event log that contains traces relative to PAP test screenings. The number of activities in this log is slightly higher than in the controlled event log (19 vs. 16), but the number of traces is way lower as there are only 55 positive examples and 102 negative ones. The discovered DNF and CNF models were respectively:

```
choice(refuse, send_result_inadequate_papTest)
OR
(
  exactly1(send_letter_negative_papTest)
  AND
  choice(send_letter_negative_papTest, execute_colposcopy_exam)
)
```

And:

```
(
  exclusive_choice(send_letter_negative_biopsy,
      send_result_inadequate_papTest)
  OR
  chain_succession(phone_call_positive_papTest,
      execute_colposcopy_exam)
)
AND
(
  chain_succession(invite, refuse)
  OR
  chain_succession(invite, execute_papTest_exam)
)
```

Regarding the performances, the discovered models were returned in a very short time as the number of traces is extremely lower than the one in the controlled event log. Again, the performance of the CNF version is better than the DNF's one, with respectively 3 and 5 seconds taken on average to return the model.

5. Related works

Traditional process discovery approaches aim at extracting a process model from positive examples of business executions. As pointed out by Goedertier et al. [4], they can be interpreted as machine learning techniques to extract a grammar from a set of positive sample data [15]. However, in process discovery authors typically make use of formalisms to express concurrency and synchronization in a more understandable way w.r.t. grammar learning (where automata, regular expressions or production rules are often employed to represent the model). Since the learning task is inevitably influenced by the type of language used for the model, this element is often used to classify process discovery approaches into two macro-categories: procedural and declarative.

Procedural approaches envisage to uncover structured processes [16, 17, 18, 19, 20, 21, 22, 23]. For the sake of our comparison, it is also important to underline that most of these works contemplate the presence of negative information in the log in the shape of non-informative noise, which should be discarded. The approach in this work is instead an example to declarative process discovery Since process models are sometimes less structured than one could expect [24], procedural discovery can lead to the identification of spaghetti-models. Declarative approaches [25, 26, 24, 27, 28, 14, 29] aim at overcoming this issue by offering a compact way to briefly list the required or prohibited behaviours in a business process. Similarly to the procedural approaches, the declarative discoverers listed so far do not deal with negative examples. Nonetheless, they indirectly envisage the possibility to discard a portion of the log by setting thresholds on metrics that the discovered model should satisfy.

In the field of grammar learning, Gold [30] showed how both positive and negative examples are required to discover a grammar with perfect accuracy. A claim particularly relevant to our

work is that, in order to distinguish the right hypothesis among an infinite number of grammars that fit the positive examples, the key element is the availability of negative examples. The reason why many procedural and declarative discoverers do not take into account negative examples can be identified in the fact that these approaches do not usually seek perfection, but focus on good performance according to defined metrics. Among traditional grammar learning approaches, the ones by Angluin [31] and Mooney [8] are particularly relevant for our work. The article [31] focuses on identifying an unknown model referred as "regular set" and represented through Deterministic Finite-state Acceptor (DFA). Coherently with Gold's theory [30], Angluin propose a learning algorithm that starts from input examples of the regular set's members and non-members. The learning process is realised through the construction of an "observation table". As discussed in this article, the approach of Mooney et al. [8] shows instead three different algorithms to learn CNF, DNF and decision trees from a set of positive and negative examples.

A subset of the declarative discoverers [32, 33, 5, 34, 35] is related to the basic principles of Inductive Constraint Logic (ICL) [36]–which depends on the availability of both negative and positive examples. In particular, similarly to the approach of this work, DecMiner [5] starts from an input set of labelled examples and learns a set of SCIFF rules [37], subsequently translated into ConDec constraints [38]. Differently from [5], our approach avoids intermediate language and aims at learning Declare constraints directly. The work [6] by Slaats et al. propose a universal declarative miner, applicable but not limited to Declare language, which makes use of negative and positive traces. W.r.t. our work, the generality of the approach in [6] hiders the use of subsumption to avoid redundancy and identify the most general model. Other relevant works are those of Neider et al. [39], Camacho et al. [40], and Reiner [41], which start form an input data set of positive and negative examples and employ a SAT-based solver to learn a simple set of LTL formulas. Differently from these works, we opt for Declare formulas with LTL_f semantics. In [42], a SAT-based solver is also used to discover a Declare model from a log with both positive and negative traces. According to the classification of Gunther et al. [43], the concept of negative example used in these works (as well as in this one) is connected to both the concepts of syntactical and semantic noise.

Another research field related to our work is that of deviance mining [44], which aims at characterizing those log traces that deviates from the expected behaviour. In particular, some works focus on the differences between the models discovered from deviant and non-deviant traces [45, 46], whereas other works intend deviance mining as a classification task similarly to sequence classification [47, 48, 49, 50, 51, 52, 53].

A limited number of recently proposed procedural approaches [54, 4, 55, 56, 57] also actively take into account negative examples. Finally, the development of synthetical log generators producing both positive and negative process cases [58, 59, 60, 4, 61, 62] is another sign that underlines how the process discovery research field is increasingly considering negative traces as informative examples.

6. Conclusions and future work

In this work we presented an adaptation of well known discovery algorithms from previous works [7, 8]. In particular, we chose as target language for process descriptions the Declare language: being based on a formal semantics expressed in LTL*f*, the adaptation of the existing approaches was quite seamless. Then, we exploited the existence of a subsumption relation between some Declare templates to extend the specialization/generalization steps towards in the original algorithms.

From the perspective of the BPM research field, and of the Declare-based approaches, it is worthy to notice that our discovery algorithms are quite innovative w.r.t. existing approaches. Firstly because we put a strong emphasis on the use of negative examples. Secondly, and more important, because we suggest new Declare models based on DNF or CNF formulas. Indeed, at its core, Declare allows only models defined in terms of conjunction of constraints, and the disjunction is not fully supported. Hence, Declare in its original definition would not support CNF models, nor DNF, as instead we do in this paper. It is highly debatable, however, if the introduction of full DNF/CNF models allows to obtain simpler, or more meaning process models w.r.t. the original limitations imposed by Declare. In turn, usability of the whole system might be affected by the type of discovered model. These are indeed topics of future investigation.

Besides this, there are many aspects that we plan to investigate in future research activities. First of all, Declare models are defined in terms of completely grounded constraints: the introduction of variables in the constraints might result in better process models, and technically speaking, Inductive Logic Programming algorithms would provide already an interesting solution (at a higher computational cost, unfortunately). The use of variables would also offer another way for specializing/generalizing the models.

Another aspect that might enjoy the use of variables in the models, and the adoption of ILP techniques, is related to the presence of data in the logs. It is quite common to encounter process logs where activities in a trace are associated with more information than just their name or timestamp. Being able to support all the data associated to each activities could make it possible to perform other tasks, different from the generation of the model. For example, having not only the information that someone logged into their account at a certain time, but also knowing who it was and what password was used could be useful for a statistical research on how many times someone tried to log into a certain profile, leading to the identification of hacking attempts and of the processes adopted in the attempts.

From a technical viewpoint, the introduction of variables in the Declare model would require a different semantics (the current one is based on propositional LTL over finite traces). In this sense, Constraint Logic Programming (CLP) over finite domains might be a viable alternative, supporting the semantics and the implementation at the same time. With a minimum amount of code it would be feasible to specify, for example, that a certain variable "X" can only assume values associated with a finite set of activities. As a side advantage, the definition of some constraints would be easier: for example, a quite common business constraint is that a certain activity should not be executed twice consecutively; this would be achieved through a chain_response(X,Y) constraint, with a further CLP constraint $X \neq Y$.

Another interesting research direction regards the gain function used in the algorithm. Currently, the gain function only takes into account the number of positive and negative traces

covered by a constraint. However, we might imagine scenarios where the users want to express desiderata and preferences over the discovered process models. For example, users might have a preference for a specific constraint template like init(X), or constraints grounded on certain activities rather than others. This could be achieved by defining a different gain function or, exploiting the existing literature on the topic, by investigating the relations with the existing preference logics. In turn, such perspective open up a question about the optimality of a model, that indeed was not investigated in this work.

Finally, a deeper comparison with existing approaches should be carried on, in order to better understand the quality of the discovered models, the usability of the approach and the usability of the discovered models from the final user viewpoint, and also to assess the performances of our approach w.r.t. state-of-the-art process discovery algorithms.

Acknowledgments

This work has been partially supported by the European Union's H2020 projects HumaneAI-Net (g.a. 952026), StairwAI (g.a. 101017142), and TAILOR (g.a. 952215).

References

[1] W. M. P. van der Aalst, al., Process mining manifesto, in: F. Daniel, K. Barkaoui, S. Dustdar (Eds.), BPM Workshops - BPM 2011 International Workshops, Clermont-Ferrand, France, 2011, Revised Selected Papers, Part I, volume 99 of *LNBIP*, Springer, 2011, pp. 169–194. doi:10.1007/978-3-642-28108-2_19.

[2] M. Pesic, Constraint-based workflow management systems : shifting control to users, Ph.D. thesis, Industrial Engineering and Innovation Sciences, 2008. doi:10.6100/IR638413.

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

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

[5] 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.

[6] T. Slaats, S. Debois, C. O. Back, Weighing the pros and cons: Process discovery with negative examples, in: BPM, volume 12875 of *LNCS*, Springer, 2021, pp. 47–64.

[7] J. R. Quinlan, Learning logical definitions from relations, Mach. Learn. 5 (1990) 239–266. URL: https://doi.org/10.1007/BF00117105. doi:10.1007/BF00117105.

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

[9] M. Adams, A. V. Hense, A. H. ter Hofstede, Yawl: An open source business process management system from science for science, SoftwareX 12 (2020) 100576. doi:https://doi.org/10.1016/j.softx.2020.100576.

[10] Business process model and notation (BPMN), https://www.bpmn.org/, 2022.

[11] P. Wohed, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, N. Russell, On the suitability of BPMN for business process modelling, in: S. Dustdar, J. L. Fiadeiro, A. P. Sheth (Eds.), BPM, 4th Intl. Conf., BPM 2006, Vienna, Austria, September 5-7, 2006, Procs., volume 4102 of *LNCS*, Springer, 2006, pp. 161–176. doi:10.1007/11841760_12.

[12] 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.

[13] W. M. P. van der Aalst, M. Pesic, H. Schonenberg, Declarative workflows: Balancing between flexibility and support, Comput. Sci. Res. Dev. 23 (2009) 99–113.

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

[16] 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 (2003) 151–162.

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

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

[19] 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 (2010) 87–111.

[20] 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, volume 7927 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 311–329.

[21] Q. Guo, L. Wen, J. Wang, Z. Yan, P. S. Yu, Mining invisible tasks in non-free-choice constructs, in: BPM, volume 9253 of *LNCS*, Springer, 2015, pp. 109–125.

[22] 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.

[23] 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 (2019) 686–705.

[24] 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, volume 7328 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 270–285.

[25] 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.

[26] 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), volume 3760 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 130–147.

[27] D. M. M. Schunselaar, F. M. Maggi, N. Sidorova, Patterns for a log-based strengthening of declarative compliance models, in: IFM, volume 7321 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 327–342.
[28] C. D. Ciccio, M. H. M. Schouten, M. de Leoni, J. Mendling, Declarative process discovery with minerful in prom, in: BPM (Demos), volume 1418 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2015, pp. 60–64.
[29] C. O. Back, T. Slaats, T. T. Hildebrandt, M. Marquard, DisCoveR: accurate and efficient discovery of declarative process models, Int. J. Softw. Tools Technol. Transfer (2021).
[30] E. M. Gold, Language identification in the limit, Inf. Control. 10 (1967) 447–474.
[31] D. Angluin, Learning regular sets from queries and counterexamples, Inf. Comput. 75 (1987) 87–106.
[32] E. Lamma, P. Mello, F. Riguzzi, S. Storari, Applying inductive logic programming to process mining, in: ILP, volume 4894 of *LNCS*, Springer, 2007, pp. 132–146.
[33] E. Lamma, P. Mello, M. Montali, F. Riguzzi, S. Storari, Inducing declarative logic-based models from labeled traces, in: G. Alonso, P. Dadam, M. Rosemann (Eds.), Business Process Management, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 344–359.
[34] E. Bellodi, F. Riguzzi, E. Lamma, Probabilistic logic-based process mining, in: CILC, volume 598 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2010.
[35] E. Bellodi, F. Riguzzi, E. Lamma, Statistical relational learning for workflow mining, Intell. Data Anal. 20 (2016) 515–541.
[36] L. D. Raedt, W. V. Laer, Inductive constraint logic, in: ALT, volume 997 of *Lecture Notes in Computer Science*, Springer, 1995, pp. 80–94.
[37] 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 (2008) 29:1–29:43.
[38] M. Pesic, W. M. P. van der Aalst, A declarative approach for flexible business processes management, in: Business Process Management Workshops, volume 4103 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 169–180.
[39] D. Neider, I. Gavran, Learning linear temporal properties, in: FMCAD, IEEE, 2018, pp. 1–10.
[40] A. Camacho, S. A. McIlraith, Learning interpretable models expressed in linear temporal logic, in: ICAPS, AAAI Press, 2019, pp. 621–630.
[41] H. Riener, Exact synthesis of LTL properties from traces, in: FDL, IEEE, 2019, pp. 1–6.
[42] F. Chesani, C. D. Francescomarino, C. Ghidini, D. Loreti, F. M. Maggi, P. Mello, M. Montali, S. Tessaris, Process discovery on deviant traces and other stranger things, IEEE Transactions on Knowledge and Data Engineering (2021). Under review.
[43] C. W. Günther, Process Mining in Flexible Environments, Ph.D. thesis, Technische Universiteit Eindhoven, 2009.
[44] H. Nguyen, M. Dumas, M. L. Rosa, F. M. Maggi, S. Suriadi, Business process deviance mining: Review and evaluation, CoRR abs/1608.08252 (2016).
[45] S. Suriadi, R. Mans, M. T. Wynn, A. Partington, J. Karnon, Measuring patient flow variations: A cross-organisational process mining approach, in: AP-BPM, volume 181 of *Lecture Notes in Business Information Processing*, Springer, 2014, pp. 43–58.
[46] A. Armas-Cervantes, P. Baldan, M. Dumas, L. García-Bañuelos, Behavioral comparison of

process models based on canonically reduced event structures, in: BPM, volume 8659 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 267–282.

[47] 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, volume 7908 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 449–464.

[48] 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 (2015) 19:1–19:18.

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

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

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

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

[53] J. D. Smedt, G. Deeva, J. D. Weerdt, Mining behavioral sequence constraints for classification, IEEE Trans. Knowl. Data Eng. 32 (2020) 1130–1142.

[54] H. P. de León, C. Rodríguez, J. Carmona, K. Heljanko, S. Haar, Unfolding-based process discovery, in: ATVA, volume 9364 of *LNCS*, Springer, 2015, pp. 31–47.

[55] H. P. de León, L. Nardelli, J. Carmona, S. K. L. M. vanden Broucke, Incorporating negative information to process discovery of complex systems, Inf. Sci. 422 (2018) 480–496.

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

[57] 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 (2014) 1877–1889.

[58] F. Chesani, C. Di 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, volume 2396 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2019, pp. 179–192.

[59] F. Chesani, A. Ciampolini, D. Loreti, P. Mello, Abduction for generating synthetic traces, in: BPM Workshops, volume 308 of *LNBIP*, Springer, 2017, pp. 151–159.

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

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

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