Optimising Business Process Discovery using Answer Set Programming

Federico Chesani[1], Chiara Di Francescomarino[2], Chiara Ghidini[2], Giulia Grundler[1], Daniela Loreti[1], Fabrizio Maria Maggi[3], Paola Mello[1], Marco Montali[3], Sergio Tessaris[3]

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

Abstract. Declarative business process discovery aims at identifying sets of constraints, from a given formal language, that characterise a workflow by using pre-recorded activity logs. Since the provided logs represent a fraction of all the consistent evolution of a process, and the fact that many sets of constraints covering those examples can be selected, empirical criteria should be employed to identify the "best" candidates. In our work we frame the process discovery as an optimisation problem, where we want to identify optimal sets of constraints according to preference criteria. Declarative constraints for processes are usually characterised via temporal logics, so different solutions can be semantically equivalent. For this reason, it is difficult to use an arbitrary finite domain constraints solvers for the optimisation. The use of Answer Set Programming enables the combination of deduction rules within the optimisation algorithm, in order to take into account not only the user preferences but also the implicit semantics of the formal language. In this paper we show how we encoded the process discovery problem using the ASPrin framework for qualitative and quantitative optimisation in ASP, and the results of our experiments.

Keywords: preferences, answer set programming, optimisation, process mining, process discovery, declarative process models

1 Introduction

Process discovery is one of the most investigated process mining techniques [13]. It deals with the automatic learning of a process model from a given set of logged traces, each one representing the digital footprint of a specific execution of the process. Our work develops in the context of *binary* process discovery, in which the model-extraction is seen as a two-class supervised task (see [7,2,8,11]), where log traces are partitioned into two sets according to some business or domain-related criteria (the so-called *positive* and *negative* – i.e., undesired – traces). The target of the learning process is a model that discriminates one set from the other.

Process discovery algorithms are also classified according to the language they employ to represent the output model: procedural and declarative. Techniques of the first kind envisage the process model as a synthetic description of all possible sequences of actions that the process accepts from an initial to an ending state. In declarative discovery—the focus of this work—models are sets of constraints, characterised by a declarative, logic-based semantics. Both approaches have their strengths and weaknesses depending on the characteristics of the considered process. Procedural techniques may generate hard to understand "spaghetti"-like models, and in these cases declarative-based approaches might be preferable [5]. A problem that remains unsolved in process discovery, is the need to select, among all possible discovered models, the ones that best fit the expectations of the user. This problem is manifesting in techniques that rely only on one set of traces (the positive); where the risk is to generate overfitting models. Therefore, mechanisms are introduced to "select" specific behaviours; e.g., the frequency of a certain element (e.g., an activity or a path), or the presence of certain modelling patterns. In spite of the possibility of exploiting the negative information, binary discovery techniques are also affected by the same problem. As recently shown in [11], perfect binary miners (able to discover models that accept all positive examples and none of the negative examples) do not necessarily exist; many suboptimal models can be identified by the discovery process, leading to the issue of identifying criteria for preferring one model. In most of the techniques in literature, the criteria are built in the discovery process, leaving small room for dedicated user-driven preferences. In [2] we introduced a novel algorithm that splits the discovery process in two stages: first, the set of all candidate constraints are identified, and then the selection of the model is framed as an optimisation problem selecting one (or more) subsets according to given preferences. Its implementation (`NegDis`) is available in [12]. In this short paper we focus on the optimisation stage, showing how we used Answer Set Programming Optimisation to encode and solve the second stage.

2 Declarative Process Discovery

The discovery approach we introduce in this paper is based on `Declare`, a language for describing declarative process models first introduced in [10]. A `Declare` model consists of a set of constraints on a finite set of (atomic) activities. Constraints are ground instantiation from a given set of abstract parametrised patterns (*templates*); where parameters are substituted with activities. Templates have a graphical representation and their semantics can be formalised using different logics, the main one being linear temporal logic (LTL) over finite traces, making them verifiable and executable. The major benefit of using templates – e.g., instead of LTL – is that analysts do not have to be aware of the underlying logic-based formalisation to understand the models. Table 1 summaries some common `Declare` templates. The reader can refer to [10] for a full description of the language. It is important to emphasise that `Declare` is a *family* of languages defined by a set of templates.

Table 1. Example of `Declare` templates

Template	Explanation
existence(A)	A occurs at least once
init(A)	A is the *first* to occur
response(A, B)	If A occurs, then B occurs after A
alternate_response(A, B)	Each time A occurs, then B occurs afterwards, before A recurs
precedence(A, B)	B occurs only if preceded by A
co_existence(A, B)	If B occurs, then A occurs, and vice versa
not_succession(A, B)	A never occurs before B

Given a finite set of `Declare` templates D and a finite set of activities A, we indicate with $D[A]$ all possible groundings of templates in D w.r.t. A, i.e., all the constraints that can be built using activities from A. Traces—i.e., finite sequences of activities from A—can be understood as (logical) models for constraints, and we say that $M \subseteq D[A]$ *accepts* a trace t iff, for each constraint $c \in M$, $t \models c$ w.r.t. its semantics [10]. The semantics of `Declare` introduces a natural notion of *generality* between process models; i.e. a model M is more general than M' ($M' \preceq M$) if the latter accepts all the traces accepted by M. In [3], templates are organised into a *subsumption* hierarchy, and this relation (between constraints) is used as a preference for guiding the discovery process. We generalise this notion by introducing the *deductive closure operator* based on a given set R of (correct) deduction rules,[4] as a function $cl_R : \mathcal{P}(D[A]) \rightarrow \mathcal{P}(D[A])$ that associates any set $M \in D[A]$ with all the constraints that can be logically derived from M by applying one or more deduction rules in R. For brevity, in the rest of the paper we will omit the set R, and we will simply write $cl(M)$ to indicate the deductive closure of M. The complete set deduction rules that we considered, including those introduced in [3], is available in the source code [12].[5] All the rules we analysed in the literature can be encoded as Normal Logic Program rules (more on this in Section 3).

Although `NegDis` takes as input the set of templates and deduction rules, for the sake of simplicity, in the rest of the paper we assume that they are fixed and input consists on the sets of positive and negative examples (denoted by L^+ and L^-). Candidate solutions for the discovery task are any set of constraints $S \subseteq D[A]$ s.t. *(i)* $\forall t \in L^+$ we have $t \models S$; *(ii)* S maximizes the set $\{t \in L^- \mid t \not\models S\}$.

In the first stage of the algorithm, `NegDis` builds a function (*sheriffs*) that associates each trace in L^- with the set of constraints, chosen from those accepting all traces in L^+, that reject it:

$$\mathit{sheriffs}(t) = \{c \in D[A] \mid t \not\models c \wedge \forall t' \in L^+ . t' \models c\} \tag{1}$$

Note that, due to the fact that not all the pairs of negative and positive sets of traces can be perfectly separated using `Declare` [11], there can be traces t in L^- for which $\mathit{sheriffs}(t)$ is empty, meaning that those traces cannot be excluded by any model that guarantees the acceptance of all the positive ones. The actual

[4] Identifying whether there is a complete set of rules for a specific set of templates is an open problem outside the scope of this work.

[5] The file `declare_rules.txt` in the `data` directory.

implementation of the first stage is outside the scope of this paper and the reader is referred to [2]. Based on *sheriffs*, the space of all solutions can be defined as

$$\mathcal{Z} = \{M \subseteq \bigcup_{t \in L^-} \mathit{sheriffs}(t) \mid \forall t \in L^- \; t \not\models M \vee \mathit{sheriffs}(t) = \emptyset\} \tag{2}$$

That is, the subsets of the set of all constraints in $\mathit{sheriffs}(t)$ that reject all the negative traces (excluding those that cannot be rejected, i.e., $\mathit{sheriffs}(t) = \emptyset$). In the next section we show how we use an ASP optimisation system to order $\mathcal{Z}$ and select the "best" process models.

3 ASP Encoding and Evaluation

For our experiments we used the *Clingo* system [6] because it supports *function* terms, and an advanced optimisation frontend (*ASPrin* [1]), enabling the declarative specification of preferences. The encoding of the optimisation stage in ASP follows the common Guess/Check/Optimise (GCO) ASP paradigm [9]: the *guessing* part selects subsets of $\bigcup_{t \in L^-} \mathit{sheriffs}(t)$ using a *choice rule* [6], the *checking* part selects only (ASP) models that "reject" the negative traces, while the *optimisation* part depends on selected preferences.

The *sheriffs* input is encoded as a binary predicate `choice/2` where the first argument is a trace ID (an integer) and the second a constraint that "rejects" the trace. The "output" predicate, identifying the selected constraints, is the unary predicate `selected/1`.[6] We decided to encode constraints as function terms in order to avoid the ad-hoc handling of the number of template parameters (e.g., terms like `decl(init,a)`), so the fact that the constraint $\mathsf{init}(a)$ rejects the third trace is encoded by the fact `choice(3,decl(init,a))`.

The *guessing* part is composed by a single choice rule

```
{ selected(C) : choice(_,C) }.
```

The *checking* part must take into account not only the selected constraints, but also their closure, since it affects the optimisation preferences. To this end we introduced a `derived/1` predicate, and the checking is encoded as

```
derived(C) :- selected(C).
rejected(T) :- choice(T,C), derived(C).
:- choice(T,_), not rejected(T).
```

The *guessing* and *checking* parts above enables the generation of all models corresponding to the sets in $\mathcal{Z}$ (Eq. 2). Deduction rules are encoded using the `derived/1` predicate; e.g., the rule $\mathsf{init}(\mathsf{A}) \rightarrow \mathsf{precedence}(\mathsf{A}, \mathsf{B})$ is encoded as

```
derived(decl(precedence,X,Y)) :- derived(decl(init,X)), action(Y).
```

Assuming a finite number of activities, `Declare` deduction rules studied in literature can be encoded as *full tuple-generating dependencies* [4]. Therefore, for any subset of $\bigcup_{t \in L^-} \mathit{sheriffs}(t)$, the extension of `derived/1` is unique. Moreover, since

[6] In the actual code the predicate names are slightly different to avoid potential clashes with names used by *ASPrin*, and they can be parametrised.

each constraint in *sheriffs* accepts all traces in L^+, any subset of $\bigcup_{t \in L^-} \mathit{sheriffs}(t)$ is consistent.

Enumerating all models is too expensive, and doesn't provide any guide to select the most suitable (from the user point of view). For the *optimisation* part we started experimenting with cardinality preferences over the deductive closure and selection; which can be simply implemented via *Clingo* minimisation statements (grounding macros for *weak constraints* [6]):

```
#minimize{1@2,C: derived(C)}.
#minimize{1@1,C: selected(C)}.
```

Specifying more elaborate preferences require complex encodings and ASP techniques, which are difficult to manage and error-prone to non ASP experts. To simplify the specification we exploit the *ASPrin Clingo* frontend [1], which provides a general framework for optimising qualitative and quantitative preferences in ASP. For example, "subset" optimality can be encoded using

```
#preference(p1,subset){ derived(C) : constraint(C) }.
#preference(p2,less(cardinality)){ selected(C) : choice(_, C) }.
#preference(p10,lexico){ 1::**p2; 2::**p1 }.
#optimize(p10).
```

which prefers models with a (subset) smaller closure, and (cardinality) smaller selected in case of ties. The built-in directives of *ASPrin* can be used to specify also preferences of specific properties of the models; e.g., to prefer models with specific templates:

```
not_nice_model :- selected(C), template_name(C,not_succession).
nice_model :- not not_nice_model.
#preference(p1,aso){ nice_model >> not_nice_model }.
#preference(p2,subset){ derived(C) : constraint(C) }.
#preference(p10,lexico){ 1::**p2; 2::**p1 }.
#optimize(p10).
```

The encoding has been evaluated in the context of the discovery process using both synthetic and real datasets. For the description of the datasets and details on the results the reader is referred to [2]. In this paper we focus on the optimisation stage, considering the size and structure of the *sheriffs* input: the number of traces and the average number of "rejecting" constraints per trace. Table 2 shows the optimisation time, for the "subset" and "cardinality" criteria above, compared to the time spent to calculate the *sheriffs* input. By considering the whole discovery problem, in most of the datasets, the runtime of the optimisation is an order of magnitude smaller than the first stage, and it seems to be correlated with the size of the minimal (process) model discovered.

Table 2. Running time

Dataset	*sheriffs* input		CPU time (sec)		
	size	avg	*sheriffs*	subset	card
SYNT$_a$	25600	32.2491	113.06	15.085	11.559
SYNT$_b$	10240	9.40303	97.47	1.377	1.201
CERV$_{compl}$	102	10.402	0.33	0.065	0.045
SEPSIS$_{mean}$	9	2.44444	1.04	0.039	0.035
SEPSIS$_{median}$	141	24.0851	1.05	0.2	0.087
BPIC12$_{mean}$	70	8.84286	31	0.096	0.066
BPIC12$_{median}$	2394	9.15748	37.63	359.164	43.846

4 Conclusions

In this paper we demonstrate the use of ASP optimisation to encode preferences in complex domains where the optimisation criteria cannot be fixed beforehand; e.g., in our case process models can be preferred because of the presence of some patterns which are domain dependent. The flexibility of a rule-based system enables the handling of complex interactions between the components of a solution and its optimisation. In our domain, because of the need to take into account the deductive dependency between `Declare` constraints, we cannot use traditional finite domain solvers. Our empirical evaluation shows that the *Clingo* solver can efficiently handle the optimisation stage for the preferences we selected. We plan to investigate whether the system can be pushed further with more complex preferences; e.g., interaction between different templates or activities.

References

1. Brewka, G., Delgrande, J.P., Romero, J., Schaub, T.: asprin: Customizing answer set preferences without a headache. In: AAAI. pp. 1467–1474. AAAI Press (2015)
2. Chesani, F., Di Francescomarino, C., Ghidini, C., Loreti, D., Maggi, F.M., Mello, P., Montali, M., Tessaris, S.: Process discovery on deviant traces and other stranger things. CoRR abs/2109.14883 (2021), `https://arxiv.org/abs/2109.14883`
3. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: Resolving inconsistencies and redundancies in declarative process models. Inf. Syst. 64, 425–446 (2017)
4. Fagin, R.: Tuple-Generating Dependencies. In: LIU, L., ÖZSU, M.T. (eds.) Encyclopedia of Database Systems, pp. 3201–3202. Springer US, Boston, MA (2009)
5. Fahland, D., Lübke, D., Mendling, J., Reijers, H.A., Weber, B., Weidlich, M., Zugal, S.: Declarative versus imperative process modeling languages: The issue of understandability. In: BMMDS/EMMSAD. vol. 29, pp. 353–366. Springer (2009)
6. Gebser, M., Kaminski, R., Kauffman, B., Schaub, T.: Multi-shot asp solving with clingo. Theory and Practice of Logic Progr. 19(1), 27–82 (2019)
7. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events. J. Mach. Learn. Res. 10, 1305–1340 (2009)
8. de León, H.P., Nardelli, L., Carmona, J., vanden Broucke, S.K.L.M.: Incorporating negative information to process discovery of complex systems. Inf. Sci. 422, 480–496 (2018)
9. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Transactions on Computational Logic 7(3), 499–562 (Jul 2006)
10. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007). pp. 287–300 (2007)
11. Slaats, T., Debois, S., Back, C.O.: Weighing the pros and cons: Process discovery with negative examples. In: Business Process Management - 19th International Conference, BPM 2021. vol. 12875, pp. 47–64 (2021)
12. Tessaris, S., Di Francescomarino, C., Chesani, F.: Negdis: code for the experiments (2022), `https://doi.org/10.5281/zenodo.6396859`
13. van der Aalst, et al.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) Business Process Management Workshops. pp. 169–194. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)