

Automata-Theoretic Techniques for Declarative Process Mining

Francesco Chiariello

DIAG - Sapienza University of Rome
chiariello@diag.uniroma1.it

Table of Contents

- 1 Introduction
- 2 Background
 - DPM
 - LTLf
 - ASP
- 3 ASP for DPM
- 4 DFA Discovery
- 5 Conclusion

Table of Contents

- 1 Introduction
- 2 Background
 - DPM
 - LTLf
 - ASP
- 3 ASP for DPM
- 4 DFA Discovery
- 5 Conclusion

Motivation

- Use of Finite State Automata for Process Mining (PM)
- Automata can be used as a formalism to model processes, similar to Petri nets in classical PM.
- Their relation with temporal languages makes them an interesting tool for Declarative PM.
- Automata and Formal Languages are natural choices for modeling processes.

Contributions

- 1 Answer Set Programming (ASP) for Declarative PM (DPM)
 - Temporal Reasoning in ASP
- 2 Discovering automata from event logs

Table of Contents

- 1 Introduction
- 2 Background
 - DPM
 - LTLf
 - ASP
- 3 ASP for DPM
- 4 DFA Discovery
- 5 Conclusion

Table of Contents

- 1 Introduction
- 2 Background
 - DPM
 - LTLf
 - ASP
- 3 ASP for DPM
- 4 DFA Discovery
- 5 Conclusion

Process Mining

- Process Mining (van der Aalst & Carmona, 2022) = Business Process Management + Data Mining.
- Extract information from event logs.
- In Declarative PM (DPM) processes are expressed with constraints, e.g., in `DECLARE` (van der Aalst et al., 2009) or `LTLf` (De Giacomo & Vardi, 2013).

Event Log Example

Case	Activity	Timestamp	Resource	Customer
pizza-56	buy ingredients (bi)	18:10	Stefano	Valentina
pizza-57	buy ingredients (bi)	18:12	Stefano	Giulia
pizza-57	create base (cb)	18:16	Mario	Giulia
pizza-56	create base (cb)	18:19	Mario	Valentina
pizza-57	add tomato (at)	18:21	Mario	Giulia
pizza-57	add cheese (ac)	18:27	Mario	Giulia
pizza-56	add cheese (ac)	18:34	Mario	Valentina
pizza-56	add tomato (at)	18:44	Mario	Valentina
pizza-56	add salami (as)	18:45	Mario	Valentina
pizza-56	bake in oven (bo)	18:48	Stefano	Valentina
pizza-57	add salami (as)	18:50	Mario	Giulia

Process Perspectives

- **Control-flow** perspective focuses on activities. ✓
- **Data-aware** perspective focuses on attributes. ✓
- **Time** perspective focuses on timestamps. ✗

Table of Contents

- 1 Introduction
- 2 Background
 - DPM
 - LTLf
 - ASP
- 3 ASP for DPM
- 4 DFA Discovery
- 5 Conclusion

Temporal Logics on Finite Traces

- Both in PM and AI, we usually consider terminating tasks.
- DECLARE and LTL_f for finite sequences.

LTL_f: Syntax

- Given a set \mathcal{P} of propositional symbols, the syntax is defined by the following grammar

$$\varphi ::= A \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{X}\varphi \mid \varphi_1 \mathbf{U}\varphi_2$$

with $A \in \mathcal{P}$.

- Common abbreviations used are:
 - $\text{true}, \rightarrow, \vee$
 - $\mathbf{F}\varphi \equiv \text{true}\mathbf{U}\varphi$
 - $\mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi$
 - $\tilde{\mathbf{X}}\varphi \equiv \neg\mathbf{X}\neg\varphi$
 - $\varphi_1 \mathbf{W}\varphi_2 \equiv \varphi_1 \mathbf{U}\varphi_2 \vee \mathbf{G}\varphi_1$

LTL_f: Semantics

- Given a formula φ , a trace $\pi = \pi_1, \pi_2, \dots, \pi_{len(\pi)} \in (2^{\mathcal{P}})^+$, and a time instant i , with $1 \leq i \leq len(\pi)$, the semantics is defined as follows:
 - $\pi, i \models A$ iff $A \in \pi_i$,
 - $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$,
 - $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$,
 - $\pi, i \models \mathbf{X}\varphi$ iff $i < len(\pi)$ and $\pi, i + 1 \models \varphi$,
 - $\pi, i \models \varphi_1 \mathbf{U}\varphi_2$ iff $\pi, j \models \varphi_2$ for some j , with $i \leq j \leq len(\pi)$, and $\pi, k \models \varphi_1$ for all $k = i, \dots, j - 1$.

We write $\pi \models \varphi$, and we say that π satisfies φ , if $\pi, 1 \models \varphi$.

DECLARE

- DECLARE: declarative process modeling language that consists of a set of templates
- A model is a conjunction of constraints, i.e., templates' instantiations
- Semantics grounded in LTL_f

DECLARE Templates

DECLARE	LTL _f
INIT(a)	a
ATLEASTONE(a)	$\mathbf{F}(a)$
ATMOSTONE(a)	$\mathbf{G}(a \rightarrow \tilde{\mathbf{X}}(\mathbf{G}(\neg a)))$
END(a)	$\mathbf{F}(a \wedge \text{Last})$
RESPONDEDEXISTENCE(a,b)	$\mathbf{F}(a) \rightarrow \mathbf{F}(b)$
RESPONSE(a,b)	$\mathbf{G}(a \rightarrow \mathbf{F}(b))$
ALTERNATEPRECEDENCE(a,b)	$\mathbf{G}(a \rightarrow (\mathbf{X}(\neg a \mathbf{U} b)))$
CHAINRESPONSE(a,b)	$\mathbf{G}(a \rightarrow \mathbf{X}b)$
PRECEDENCE(a,b)	$\neg b \mathbf{W} a$
ALTERNATEPRECEDENCE(a,b)	$(\neg b \mathbf{W} a) \wedge \mathbf{G}(b \rightarrow \tilde{\mathbf{X}}(\neg b \mathbf{W} a))$
CHAINPRECEDENCE(a,b)	$\mathbf{G}(\mathbf{X}b \rightarrow a) \wedge \neg b$
COEXISTENCE(a,b)	$\mathbf{F}a \leftrightarrow \mathbf{F}b$
SUCCESSION(a,b)	$\mathbf{G}(a \rightarrow \mathbf{F}b)$
ALTERNATESUCCESSION(a,b)	$\mathbf{G}(a \rightarrow \mathbf{X}(\neg a \mathbf{U} b)) \wedge \mathbf{G}(b \rightarrow \tilde{\mathbf{X}}(\neg b \mathbf{W} a)) \wedge \neg b \mathbf{W} a$
CHAINSUCCESSION(a,b)	$\mathbf{G}(a \leftrightarrow \mathbf{X}b) \wedge \neg b$

Table of Contents

- 1 Introduction
- 2 Background
 - DPM
 - LTLf
 - ASP
- 3 ASP for DPM
- 4 DFA Discovery
- 5 Conclusion

Answer Set Programming

- Answer Set Programming (ASP): declarative approach for search and optimization problems (Marek & Truszczynski, 1999; Niemelä, 1999).
- Provide a modeling language for writing logic programs.
- Programs' models are computed with ASP systems such as
 - *clingo* (Gebser et al., 2019)
 - DLV (Alviano et al., 2017)

Generate-and-test Methodology

- Generate-and-test (also called Guess-and-Check) methodology:
 - ① Generate: guess a candidate solution
 - ② Test: check if the candidate is a proper solution
- Differences from brute force:
 - candidate's selection
 - evaluation of partial candidates

Table of Contents

- 1 Introduction
- 2 Background
 - DPM
 - LTLf
 - ASP
- 3 ASP for DPM**
- 4 DFA Discovery
- 5 Conclusion

Temporal Reasoning in ASP¹²³

Objectives

- Perform Temporal Reasoning with ASP.
- Demonstrate its application to Declarative Process Mining.

Solution Approach

- Represent the automaton associated to the specifications,
- Simulate the run of traces over the automaton.

¹Chiariello, F., Maggi, F. M., & Patrizi, F. (2022b). ASP-Based Declarative Process Mining. *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI*

²Chiariello, F., Maggi, F. M., & Patrizi, F. (2022c). ASP-Based Declarative Process Mining (Extended Abstract). *Proceedings of the 38th International Conference on Logic Programming (ICLP)*

³Chiariello, F., Maggi, F. M., & Patrizi, F. (2022a). A tool for compiling Declarative Process Mining problems in ASP. *Softw. Impacts, 14*

Temporal Reasoning in ASP (cont'd)

Technical contribution

- ASP as a tool to perform Temporal Reasoning
- Automata as an approach to deal with time

Multidisciplinary contribution

- Process Mining (PM)
- Temporal Logics (TL)
- Answer Set Programming (ASP)

Motivation

- Minimality of ASP semantics allows one to easily represent and reason with automata.

Log Generation

Log generation (Skydaniienko et al., 2018): generating a log compliant with a process model.

- Input
 - Model: `RESPONSE(a,b)`, `ATLEASTONE(c)`
 - Trace length: 5
 - Log size: 2
- Output: `(a,b,c,c,b)`; `(c,b,c,c,c)`

Conformance Checking

Conformance checking (Burattin et al., 2016): checking whether traces are compliant with a process model.

- Input
 - Model: (1) RESPONSE(a,b), (2) ATLEASTONE(c)
 - Log: (i) (a,b,c,c,b); (ii) (c,b,c,c,c)
- Output: (1,i),(1,ii),(2,i),(2,ii)

Query Checking

Query checking (Räim et al., 2014): finding properties of a process by checking possible templates against its event log.

- Input
 - Log: (a,b,c,c,b); (c,b,c,c,c)
 - Template: RESPONSE(?a,?b)
 - (optional) Constraints number: 1
- Output: RESPONSE(a,b)

Framework

- An *activity* is an expression of the form $A(a_1, \dots, a_{n_A})$, where
 - A is the *activity name*
 - a_1, \dots, a_{n_A} are *attribute names*.
- An *event* is an expression of the form $e = A(v_1, \dots, v_{n_A})$, where
 - $v_i \in D_A(a_i)$ is a possible value of a_i
- A *process trace* is a finite sequence of events $\pi = e_1 \cdots e_n$.
- An event log is a finite set of traces.

LTL_f with local conditions (L-LTL_f)

Given a finite set of activities Act , the formulas φ of L-LTL_f over Act are inductively defined as follows:

$$\varphi = \mathbf{true} \mid A \mid a \odot a' \mid a \odot v \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U}\varphi,$$

where: a and a' are attribute names from some activity in Act , $v \in D_A(a)$, for some $A \in Act$, \odot is an operator from $\{<, \leq, =, \geq, >\}$, and A is an activity name from Act .

Automata Representation of L-LTL_f formulas

- For every L-LTL_f formula φ there exists a finite-state automaton (FSA) \mathcal{A}_φ that accepts exactly the traces that satisfy φ .
- Such automata are standard FSA with transitions labeled by event formulae (i.e. without temporal operators).

Approach

- Convert specifications into automata.
- Represent automata in ASP.
- Represent traces in ASP.
- Modeling how automata read traces.
- Add generation and test rules.

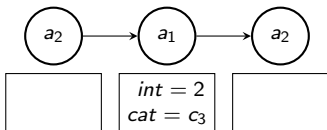
Example: Traces

Data Types

- $act(a_1)$. $has_attr(a_1, int)$. $has_attr(a_1, cat)$.
- $act(a_2)$.
- $value(int, 1..10)$.
- $value(cat, c1)$. $value(cat, c2)$. $value(cat, c3)$.

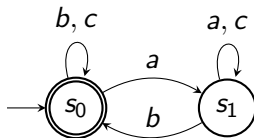
Traces

- $trace(a_2, 0)$.
- $trace(a_1, 1)$. $has_value(int, 2, 1)$. $has_value(cat, c_3, 1)$.
- $trace(a_2, 2)$.



Example: Automata

- $init(s_0)$.
- $acc(s_0)$.
- $trans(s_0, 1, s_1)$.
- $holds(1, T) \leftarrow trace(a, T)$.
- $trans(s_1, 2, s_0)$.
- $holds(2, T) \leftarrow trace(b, T)$.
- $trans(s_0, 3, s_0)$.
- $holds(3, T) \leftarrow trace(b, T)$.
- $holds(3, T) \leftarrow trace(c, T)$.
- $trans(s_1, 4, s_1)$.
- $holds(4, T) \leftarrow trace(a, T)$.
- $holds(4, T) \leftarrow trace(c, T)$.



Experiments: tools

SoA (data-aware) tools used for comparison

- Log Generation: MP-Declare Log Generator (Skydaniienko et al., 2018)
- Conformance Checking: Declare Analyzer (Burattin et al., 2016)
- Query Checking: no existing tool

Language

- Our approach supports the full language LTL_f
- The available tools support only `DECLARE`
- For comparison, the experiments are limited to `DECLARE`

Experiments: Log Generation

- 8 synthetic models and 8 model from real-life event logs
- Synthetic models allow us to test scalability of the tools in a controlled environment
- Real models used to test the tools in real environments

Logs

Log Name	Total traces	Distinct traces	Total events	Activity types	Trace length		
					min	avg	max
LOAN	13,087	4,336	164,506	23	3	13	96
DD	10,500	99	56,437	17	1	5	24
ID	6,449	753	72,151	34	3	11	27
PL	7,065	1,478	86,581	51	3	12	90
PTC	2,099	202	18,246	29	1	9	21
RT	150,370	231	561,470	11	2	4	20
Sepsis	1,050	846	15,214	16	3	14	185

Table: Real life logs (available at data.4tu.nl)

A log in detail

- Permit Log (PL) (van Dongen, 2020): travel permits log at TU/e university
- Traces look like:
 - (Start Trip;
End Trip;
Permit SUBMITTED by EMPLOYEE;
Permit APPROVED by ADMINISTRATION;
Permit FINAL_APPROVED by SUPERVISOR)
 - (Permit SUBMITTED by EMPLOYEE;
Permit FINAL_APPROVED by SUPERVISOR;
Start Trip;
End Trip)

Results Log Generation: Real Model

Models learned from real-life event logs using Declare Miner 2.0 (Maggi et al., 2018), using a 80% support

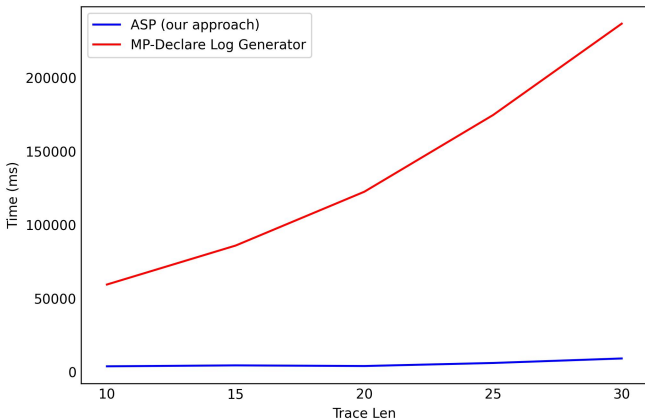


Figure: PL: results for generating 10,000 traces

Results Log Generation: Synthetic

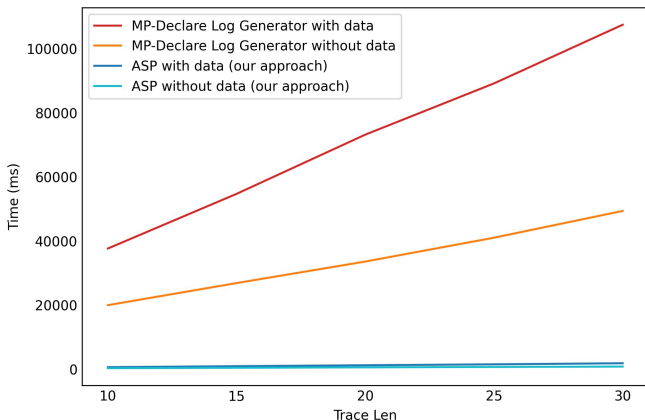


Figure: Results for a synthetic model with 10 constraints

Experiments: Conformance Checking

- Both synthetic and real-life models are considered (the same as before)
- For synthetic experiments we generate the logs (using previous techniques)

Results Conformance Checking: Real Model

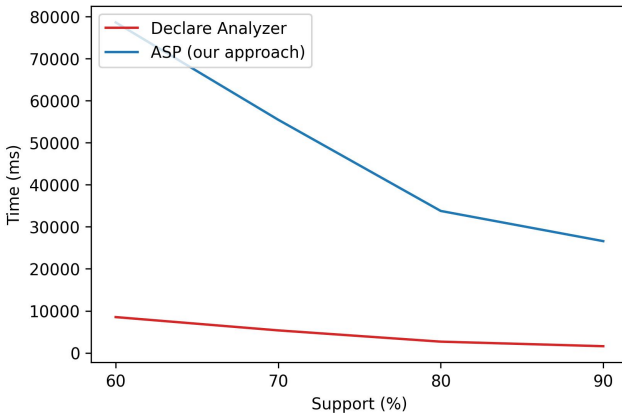


Figure: Results for a log of 10000 traces

Results Conformance Checking: Synthetic

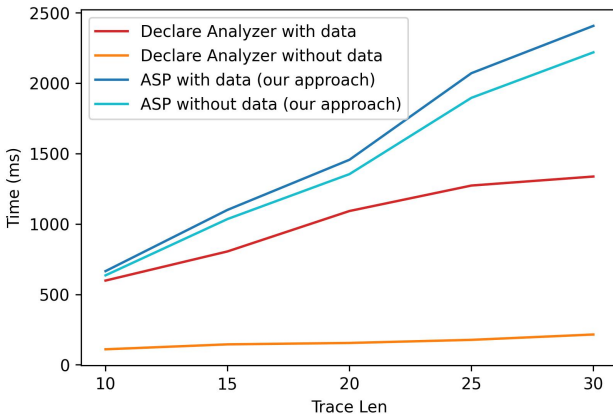


Figure: Results for a log of 1000 traces

Results Query Checking

Only tool for data-aware Query Checking

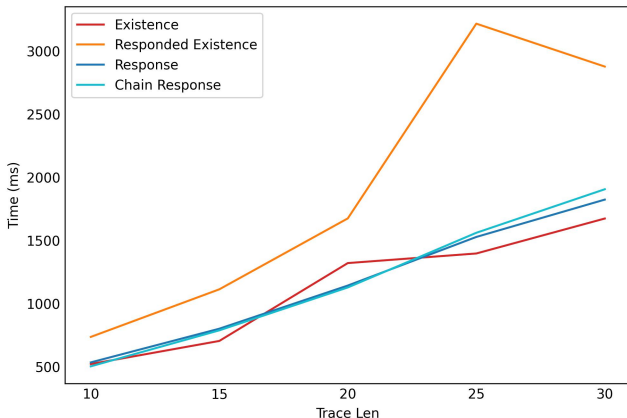


Figure: Results for a log of 1000 traces

Results

- Log Generation: outperforms the SoA tool MP-Declare Log Generator
- Conformance Checking: generalizes Declare Analyzer (Burattin et al., 2016)
- Query Checking: first tool to solve data-aware version

Table of Contents

- 1 Introduction
- 2 Background
 - DPM
 - LTLf
 - ASP
- 3 ASP for DPM
- 4 DFA Discovery**
- 5 Conclusion

DFA Discovery⁴

Objective

- Leverage Model Learning (ML) for the automated discovery of Deterministic Finite-state Automata (DFAs) from event logs.
- Process Discovery using automata as models.
- No additional information is used, such as DECLARE templates.

⁴Agostinelli, S., Chiariello, F., Maggi, F. M., Marrella, A., & Patrizi, F. (2023b).

Process mining meets model learning: Discovering deterministic finite state automata from event logs for business process analysis. *Inf. Syst.*, 114

Discovering DFAs

- DFA have been used for various BPM use cases (Process Discovery, Conformance Checking, Compliance Monitoring)
- To support them, techniques for discovering DFAs from event logs are very valuable.
- **Model Learning** (Vaandrager, 2017): algorithms for constructing models of systems from observed input–output data

Model Learning

- Main categories of ML algorithms:
 - **Active Learning** algorithms work by posing queries to a Minimally Adequate Teacher (MAT).
 - **Passive Learning** algorithms learn the behavioral model of the System Under Learning (SUL) from a pre-defined set of training data.

Contribution

- We investigate the effectiveness of the best-known passive learning algorithms.
- Cases considered:
 - only positive traces,
 - both positive and negative traces.
- Standard PM quality metrics have been customized and extended to negative traces
- The use of negative traces is greatly beneficial in improving the quality of the learned process.

Passive Learning Algorithms

- Algorithms considered (Raffelt et al., 2005; De la Higuera, 2010)
 - Minimum Description Length (MDL)
 - Regular Positive and Negative Inference (RPNI)
 - Evidence Driven State Merging (EDSM)
 - L^* (Angluin, 1987; Isberner et al., 2014)⁵
- Comparison with the SoA tool Declare Miner (Maggi et al., 2018).

⁵Adapted for passive learning.

Quality Metrics

- Metrics to evaluate the quality of the discovered DFAs (Augusto et al., 2022):
 - **Precision**: degree to which the behaviors allowed by the process model are observed in the event log;
 - **Fitness** (or **Recall**): degree to which the behaviors observed in the event log are allowed by the process model;
 - **Generalization**: estimation of how well a model inferred will reproduce future behaviors not seen;
 - **Simplicity**: size of the model.

Quality Metrics: Markov abstraction (MA)

- A k -th order Markovian abstraction characterizes all the strings σ indistinguishable from a set β of traces when a memory of size k is used to compare σ against β .
- β can be either a log or a process
- σ is a possible trace of β if it is obtained by combining only substraces of length k from β , in such a way that when a k -length sliding window W is used to scan σ , only substraces from β are observed through W .

Quality Metrics

Metrics based on comparing MA of log ℓ with MA of $G_{\mathcal{M}}$

Definition (MA-based Precision)

$$MAP^k(\ell, G_{\mathcal{M}}) = 1 - \frac{\sum_{e \in E_{G_{\mathcal{M}}}} C(e, \mu_C(e))}{|E_{G_{\mathcal{M}}}|}$$

Definition (MA-based Fitness)

$$MAF^k(\ell, G_{\mathcal{M}}) = 1 - \frac{\sum_{e \in E_{\ell}} C(e, \mu_C(e)) F_e}{\sum_{e \in E_{\ell}} F_e}$$

Definition (Generalization)

$$G_{\{\ell_1, \dots, \ell_h\}}(\mathcal{I}) = \frac{1}{h} \sum_{i=1}^h MAF^k(\ell_i, \mathcal{I}(\ell - \ell_i))$$

Quality Metrics: Negative logs

- The above metrics are defined over a single (positive) log
- It is possible to define a negative version of the metric.
- The negative metric is equal to the positive of the complement automaton
- e.g., $MAP_-^k(\ell^-, G_M) = MAP^k(\ell^-, \overline{G_M})$

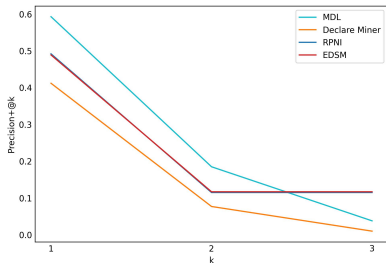
Real-life Logs

Log Name	Total traces	Total ⁺ traces	Total ⁻ traces	Distinct traces (%)	Total events	Activity types	Trace length		
							min	avg	max
LOAN	13,087	8164	4923	33.4	164,506	36	3	20	175
ROAD	150,370	82737	67633	0.2	561,470	11	2	4	20
SEPSIS	1,050	838	212	80.6	15,214	16	3	14	185
REIMB	6,449	4248	2201	11.7	72,151	34	3	11	27
TRAVEL	7,065	4249	2816	20.9	86,581	51	3	12	90

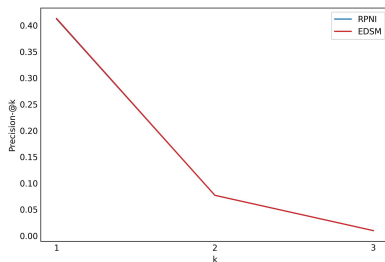
Table: Descriptive statistics of real-life logs.

Results: Precision

Declare Miner and passive learning algorithms construct DFAs with similar values of generalization and precision.



(a) Precision₊@k for ROAD log



(b) Precision₋@k for ROAD log

Results: Generalization

Declare Miner and passive learning algorithms construct DFAs with similar values of generalization and precision.

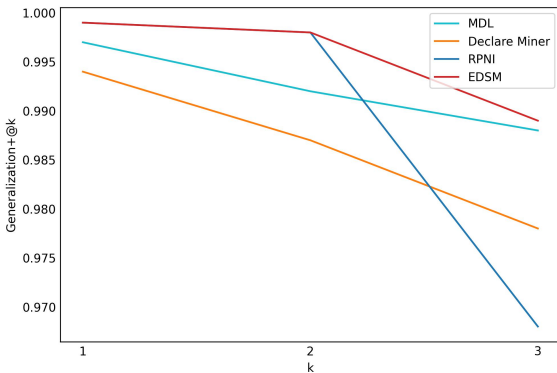


Figure: Generalization+@k for ROAD log

Results: Simplicity

Passive learning algorithms generate simpler DFAs than Declare Miner.

(a) MDL

Log (ℓ)	$S(\mathcal{M})$	t
LOAN	(42, 123)	23344 ms
ROAD	(8, 33)	249 ms
SEPSIS	(4, 19)	360 ms
REIMB	(8, 53)	393 ms
TRAVEL	(8, 107)	2352 ms

(b) RPNI

Log (ℓ)	$S(\mathcal{M})$	t
LOAN	(185, 2442)	723 ms
ROAD	(23, 99)	462 ms
SEPSIS	(39, 313)	274 ms
REIMB	(39, 568)	261 ms
TRAVEL	(51, 1057)	1380 ms

(c) EDSM

Log (ℓ)	$S(\mathcal{M})$	t
LOAN	/	/
ROAD	(21, 97)	2164 ms
SEPSIS	(55, 398)	947039 ms
REIMB	(38, 521)	3489343 ms
TRAVEL	(59, 1088)	21105718 ms

(d) Declare Miner

Log (ℓ)	$S(\mathcal{M})$	t
LOAN	(435, 1673)	21732 ms
ROAD	(121, 433)	710 ms
SEPSIS	(631, 3259)	2409 ms
REIMB	(3565, 13168)	4182 ms
TRAVEL	(182763, 1424349)	38752548 ms

Table of Contents

- 1 Introduction
- 2 Background
 - DPM
 - LTLf
 - ASP
- 3 ASP for DPM
- 4 DFA Discovery
- 5 Conclusion**

Wrapping Up

- We investigated the use of automata for various process-related tasks
 - Log Generation, Conformance Checking, and Query Checking → a solution with ASP
 - Process discovery in the form of FSA → a solution with Automata Learning
- Models as the input vs models as the output

Contribution

- Tools and techniques to deal with LTL_f specifications
- ASP encoding of important Process Mining tasks
- Evaluation of ASP approach wrt SoA tools
- Evaluation of Automata Learning algorithms
 - Considering one-class classification and binary classification
 - Proposed new metrics for evaluating discovered models

Impact

- Benefit to DPM, ASP, and TL communities
- ASP-based Log Generator integrated in the rule mining toolkit RuM^a
- Ielo et al. (2022) use the ASP approach to discover DECLARE constraints with user-defined preferences.

^arulemining.org

Future Work

- Add time-perspective and correlation condition to ASP tool
- Application of ASP to other problems
 - Process Discovery
 - Process Model Repair
 - Trace Alignment
- ... and domains
 - Discrete Event Systems
 - Planning
- Learning formulae
 - Directly from logs
 - Going through automata (DFA, AFA)
- Approximating models via Deep Learning (LSTMs, Transformers)

Publications 1/2






- Chiariello, F., Maggi, F. M., & Patrizi, F. (2022b). ASP-Based Declarative Process Mining. *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI*
- Chiariello, F., Maggi, F. M., & Patrizi, F. (2022c). ASP-Based Declarative Process Mining (Extended Abstract). *Proceedings of the 38th International Conference on Logic Programming (ICLP)*
- Chiariello, F., Maggi, F. M., & Patrizi, F. (2022a). A tool for compiling Declarative Process Mining problems in ASP. *Softw. Impacts, 14*
- Agostinelli, S., Chiariello, F., Maggi, F. M., Marrella, A., & Patrizi, F. (2023a). Discovering Deterministic Finite State Automata from Event Logs for Business Process Analysis. *On the Effectiveness of Temporal Logics on Finite Traces in AI (TLf@AAAI-SSS'23)*

Publications 2/2





- Chiariello, F. (2023). Solving Declarative Process Mining Problems Using Declarative Problem Solving. *AAAI 2023 Bridge Program on Artificial Intelligence and Business Process Management (AI4BPM)*
- Chiariello, F., Maggi, F. M., & Patrizi, F. (2023). Temporal Reasoning in ASP and its Application to Declarative Process Mining. *On the Effectiveness of Temporal Logics on Finite Traces in AI (TLf@AAAI-SSS'23)*
- Agostinelli, S., Chiariello, F., Maggi, F. M., Marrella, A., & Patrizi, F. (2023b). Process mining meets model learning: Discovering deterministic finite state automata from event logs for business process analysis. *Inf. Syst.*, 114

Thank you!!




Bibliography I

-  Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2), 87–106.
-  Marek, V. W., & Truszczyński, M. (1999). Stable models and an alternative logic programming paradigm. In *The logic programming paradigm* (pp. 375–398). Springer.
-  Niemelä, I. (1999). Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell.*, 25(3-4), 241–273.
-  Raffelt, H., Steffen, B., & Berg, T. (2005). Learnlib: A library for automata learning and experimentation. *FMICS*, 62–71.
-  van der Aalst, W. M. P., Pesic, M., & Schonenberg, H. (2009). Declarative workflows: Balancing between flexibility and support. *Comput. Sci. Res. Dev.*, 23(2), 99–113.




Bibliography II

-  De la Higuera, C. (2010). *Grammatical inference: Learning automata and grammars*. Cambridge University Press.
-  De Giacomo, G., & Vardi, M. Y. (2013). Linear temporal logic and linear dynamic logic on finite traces. *Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence*.
-  Isberner, M., Howar, F., & Steffen, B. (2014). The TTT algorithm: A redundancy-free approach to active automata learning. *RV*, 8734, 307–322.
-  Räum, M., Di Ciccio, C., Maggi, F. M., Mecella, M., & Mendling, J. (2014). Log-based understanding of business processes through temporal logic query checking. *On the Move to Meaningful Internet Systems: OTM 2014 Conferences*, 75–92.





Bibliography III

-  Burattin, A., Maggi, F. M., & Sperduti, A. (2016). Conformance checking based on multi-perspective declarative process models. *Expert Syst. Appl.*, 65.
-  Alviano, M., Calimeri, F., Dodaro, C., Fuscà, D., Leone, N., Perri, S., Ricca, F., Veltri, P., & Zangari, J. (2017). The ASP system DLV2. In M. Balduccini & T. Janhunen (Eds.), *Logic programming and nonmonotonic reasoning - 14th international conference, LPNMR 2017, espoo, finland, july 3-6, 2017, proceedings* (pp. 215–221). Springer. https://doi.org/10.1007/978-3-319-61660-5_19
-  Vaandrager, F. W. (2017). Model learning. *Commun. ACM*, 60(2), 86–95.

Bibliography IV

-  Maggi, F. M., Di Ciccio, C., Di Francescomarino, C., & Kala, T. (2018). Parallel algorithms for the automated discovery of declarative process models. *Inf. Syst.*, 74(Part), 136–152.
-  Skydanienco, V., Di Francescomarino, C., Ghidini, C., & Maggi, F. M. (2018). A tool for generating event logs from multi-perspective declare models. *Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018*.
-  Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2019). Multi-shot ASP solving with clingo. *Theory Pract. Log. Program.*, 19(1), 27–82.
<https://doi.org/10.1017/S1471068418000054>

Bibliography V

-  van Dongen, B. (2020). Bpi challenge 2020: Travel permit data. <https://doi.org/10.4121/uuid:ea03d361-a7cd-4f5e-83d8-5fbdf0362550>
-  Augusto, A., Armas-Cervantes, A., Conforti, R., Dumas, M., & Rosa, M. L. (2022). Measuring fitness and precision of automatically discovered process models: A principled and scalable approach. *IEEE Trans. Knowl. Data Eng.*, 34(4), 1870–1888.
-  Chiariello, F., Maggi, F. M., & Patrizi, F. (2022a). A tool for compiling Declarative Process Mining problems in ASP. *Softw. Impacts*, 14.
-  Chiariello, F., Maggi, F. M., & Patrizi, F. (2022b). ASP-Based Declarative Process Mining. *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI*.

Bibliography VI

-  Chiariello, F., Maggi, F. M., & Patrizi, F. (2022c). ASP-Based Declarative Process Mining (Extended Abstract). *Proceedings of the 38th International Conference on Logic Programming (ICLP)*.
-  Ielo, A., Ricca, F., & Pontieri, L. (2022). Declarative mining of business processes via ASP. *PMAI@IJCAI, 3310*, 105–108.
-  van der Aalst, W. M. P., & Carmona, J. (Eds.). (2022). *Process mining handbook* (Vol. 448). Springer.
<https://doi.org/10.1007/978-3-031-08848-3>
-  Agostinelli, S., Chiariello, F., Maggi, F. M., Marrella, A., & Patrizi, F. (2023a). Discovering Deterministic Finite State Automata from Event Logs for Business Process Analysis. *On the Effectiveness of Temporal Logics on Finite Traces in AI (TLf@AAAI-SSS'23)*.

Bibliography VII




-  Agostinelli, S., Chiariello, F., Maggi, F. M., Marrella, A., & Patrizi, F. (2023b). Process mining meets model learning: Discovering deterministic finite state automata from event logs for business process analysis. *Inf. Syst.*, 114.
-  Chiariello, F. (2023). Solving Declarative Process Mining Problems Using Declarative Problem Solving. *AAAI 2023 Bridge Program on Artificial Intelligence and Business Process Management (AI4BPM)*.
-  Chiariello, F., Maggi, F. M., & Patrizi, F. (2023). Temporal Reasoning in ASP and its Application to Declarative Process Mining. *On the Effectiveness of Temporal Logics on Finite Traces in AI (TLf@AAAI-SSS'23)*.

Table of Contents

6 ASP for DPM

7 Results in detail

ASP for DPM: data

Predicates:

- $act(A)$: A is an activity.
- $has_attr(A, N)$: activity A has attribute N .
- $val(N, V)$: a possible value of attribute N is V .

Example

Activities $a_1(int, cat)$ and $a_2()$, with $int \in \{1, \dots, 10\}$ and $cat \in \{c_1, c_2, c_3\}$ becomes:

- $act(a_1). has_attr(a_1, int). has_attr(a_1, cat).$
- $act(a_2).$
- $value(int, 1..10).$
- $value(cat, c1). value(cat, c2). value(cat, c3).$

ASP for DPM: traces

Predicates:

- $trace(A, T)$: activity A happens at time T .
- $has_value(N, V, T)$: attribute N has value V at time T .

Example

Trace $a_2()$, $a_1(2, c_3)$, $a_2()$ becomes:

- $trace(a_2, 0)$.
- $trace(a_1, 1)$. $has_value(int, 2, 1)$. $has_value(cat, c_3, 1)$.
- $trace(a_2, 2)$.

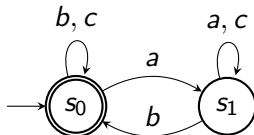
ASP for DPM: automata

- $init(S)$: S is the initial state.
- $acc(S)$: S is an accepting state.
- $trans(S, F, S')$: there exists a transition from state S to state S' labeled with event formula F .
- $holds(F, T)$: event formula F holds at time T .

ASP Formula Encoding Example

The ASP encoding of the formula $\varphi = \mathbf{G}(a \rightarrow \mathbf{F}b)$ is:

- $init(s_0)$.
- $acc(s_0)$.
- $trans(s_0, 1, s_1)$.
- $holds(1, T) \leftarrow trace(a, T)$.
- $trans(s_1, 2, s_0)$.
- $holds(2, T) \leftarrow trace(b, T)$.
- $trans(s_0, 3, s_0)$.
- $holds(3, T) \leftarrow trace(b, T)$.
- $holds(3, T) \leftarrow trace(c, T)$.
- $trans(s_1, 4, s_1)$.
- $holds(4, T) \leftarrow trace(a, T)$.
- $holds(4, T) \leftarrow trace(c, T)$.



ASP Formula Encoding Example (data)

For the data-aware formula $\varphi' = \mathbf{G}((a \wedge n < 5) \rightarrow \mathbf{F}b)$ it is sufficient to modify the rule for $holds(1, T)$:

- $holds(1, T) \leftarrow trace(a, T), has_value(n, V, T), V < 5.$

ASP for DPM: reading trace

Predicate *state* models execution of automaton on trace

- $state(S, T)$: S is current state at time T .

and updated as

- $state(S, 0) \leftarrow init(S)$.
- $state(S', T) \leftarrow$
 $state(S, T - 1), trans(S, F, S'), holds(F, T - 1)$.

Log Generation

Problem: given an L-LTL_f formula φ and trace length t , generate a trace of length t satisfying φ .

Generate traces:

- $\{trace(A, T) : activity(A)\} = 1 \leftarrow time(T)$.
- $\{has_value(N, V, T) : value(N, V)\} = 1 \leftarrow trace(A, T), has_attribute(A, N)$.

Test traces:

- $sat \leftarrow state(S, t), accepting(S)$.
- $\leftarrow not\ sat$.

Conformance Checking

Given a set of traces.

- Add the trace index i to predicate sat .
- Check whether $sat(i)$ holds.

Query Checking

Problem: Check instantiation of an input template, against a log

Define the predicates:

- $var(V)$: V is a variable.
- $assgnmt(V, A)$: activity A is assigned to variable V .

The body of the rule for *holds* is modified by replacing $trace(act, T)$ with $trace(A, T)$, $assgnmt(v, A)$, with v being the variable in place of activity act .

Then for generating

- $\{assgnmt(V, A) : activity(A)\} = 1 \leftarrow var(V)$.

and for testing we check that the formula is satisfied by the trace.

Example

Consider formula $\varphi = \mathbf{G}((?A \wedge n < 5) \rightarrow \mathbf{F}b)$.

Rule for $holds(1, T)$ is:

$$holds(1, T) \leftarrow$$
$$trace(A, T), assignmt(varA, A), has_value(n, V, T), V < 5.$$

Table of Contents

6 ASP for DPM

7 Results in detail

Results Log Geration: Real Model

Models learned from real-life event logs using Declare Miner 2.0 (Maggi et al., 2018)

Model (80) →	BPI2012	DD	ID	PL	PTC	RP	RT	Sepsis
Trace len ↓								
10	656	100*	726*	3901	1183	119*	319	460
15	817	887	2865	4538	1820	1069	353	564
20	846	832	3160	4102	2194	813	860	640
25	1061	930	4129	6169	2889	1063	483	780
30	1433	1026	5226	9231	2370	1220	630	923
10	31935	2364*	30762*	59468	65783	2703*	24909	38241
15	43337	58572	152188	85942	97098	66641	34408	57178
20	57596	80665	237777	122511	146420	95005	44608	85808
25	72383	118975	359665	174596	221434	134851	54808	120110
30	86910	181027	563794	236697	330753	187972	63379	174838

Results Log Generation: Synthetic

# constr. →	3	5	7	10
Trace len ↓				
10	35975	35786	36464	37688
15	50649	51534	54402	54749
20	69608	70342	73122	73222
25	85127	85598	87065	89210
30	101518	101882	106062	107520
10	18733	18947	19539	20007
15	25700	25723	27344	26897
20	32047	33837	33107	33615
25	39114	38666	40556	41055
30	46207	46706	47613	49410
10	595	614	622	654
15	876	894	904	956
20	1132	1155	1178	1250
25	1364	1413	1444	1543
30	1642	1701	1746	1874
10	249	270	289	340
15	349	390	408	457
20	436	496	538	601
25	519	568	611	712
30	622	666	726	837

Results Conformance Checking: Real Model

Table: Conformance Checking

	BPI2012	DD	ID	PL	PTC	RP	RT	Sepsis
60	33426	13084	49969	78625	8412	9354	49501	7116
70	33242	13245	46388	55475	5596	9359	35537	3796
80	24482	10176	29969	33775	4699	6836	35483	1778
90	8445	4568	17576	26590	2787	5608	35483	731
60	2882	2771	9800	8521	1549	2122	15262	1194
70	2852	3249	7416	5358	959	2102	10351	705
80	2291	2103	3993	2677	755	1532	11285	318
90	1691	1525	1946	1595	404	1091	10628	250

Results Conformance Checking: Synthetic

Table: Conformance Checking

Tool →	ASP		Declare Analyzer	
Trace Len ↓	data	no data	data	no data
10	665	635	598	110
15	1100	1035	805	145
20	1456	1354	1092	155
25	2071	1896	1273	177
30	2407	2219	1337	215

Results Query Checking

Constraints → Trace len ↓	Existence	Responded Existence	Response	Chain Response	Absence	Not Resp. Existence	Not Resp.	Not Chain Response
10	521	736	534	503	566	783	602	385
15	704	1113	801	788	784	1180	879	606
20	1321	1675	1143	1128	1373	1821	1304	865
25	1397	3218	1528	1561	1562	2823	1807	1104
30	1674	2878	1824	1906	1905	2784	2028	1301

Table: Time (in ms) to check different DECLARE constraints (with both activation and target activity, if any, unknown) against a log of 1000 traces.

Real-life Logs

Log Name	Total traces	Total ⁺ traces	Total ⁻ traces	Distinct traces (%)	Total events	Activity types	Trace length		
							min	avg	max
LOAN	13,087	8164	4923	33.4	262,200	36	3	20	175
ROAD	150,370	82737	67633	0.2	561,470	11	2	4	20
SEPSIS	1,050	838	212	80.6	15,214	16	3	14	185
REIMB	6,449	4248	2201	11.7	72,151	34	3	11	27
TRAVEL	7,065	4249	2816	20.9	86,581	51	3	12	90

Table: Descriptive statistics of real-life logs.

Precision MDL Declare Miner

Table: Precision₊@k of the DFAs generated with MDL and Declare Miner(a) Precision₊@k (MDL)

Log (ℓ)	k = 1	t	k = 2	t	k = 3	t
LOAN	0.395	8.94s	0.088	16.98s	0.014	105.86s
ROAD	0.593	3.58s	0.185	3.71s	0.038	14.76s
SEPSIS	0.672	4.06s	0.216	6.80s	0.046	1m42s
REIMB	0.284	6.48s	0.035	27.11s	0.003	17m56s
TRAVEL	0.214	11.91s	0.013	10m54s	/	timeout

(b) Precision₊@k (Declare Miner)

Log (ℓ)	k = 1	t	k = 2	t	k = 3	t
LOAN	0.191	9.91s	0.020	12.99s	0.001	17.82s
ROAD	0.412	3.02s	0.077	3.17s	0.010	4.45s
SEPSIS	0.358	9.97s	0.063	12.90s	0.008	49.24s
REIMB	0.139	43.98s	0.009	56.70s	0.0004	1m39s
TRAVEL	/	timeout	/	timeout	/	timeout

Precision RPN1

Table: Precision₊@k and Precision₋@k of the DFAs generated with RPN1(a) Precision₊@k (RPN1)

Log (ℓ)	k = 1	t	k = 2	t	k = 3	t
LOAN	0.222	17.37s	0.022	47.37s	0.002	16m40s
ROAD	0.492	3.82s	0.115	4.84s	0.017	20.25s
SEPSIS	0.457	5.44s	0.101	13.95s	0.015	4m4s
REIMB	0.156	3.75s	0.010	88.38s	0.0005	62m46s
TRAVEL	0.185	12.37s	0.009	9m24s	/	timeout

(b) Precision₋@k (RPN1)

Log (ℓ)	k = 1	t	k = 2	t	k = 3	t
LOAN	0.191	16.97s	0.020	1m3s	0.001	40m5s
ROAD	0.412	6.56s	0.077	4.99s	0.010	92m30s
SEPSIS	0.358	3.07s	0.063	6.89s	0.008	169.18s
REIMB	0.139	3.73s	0.009	1m1s	0.0004	62m13s
TRAVEL	0.156	11.08s	0.007	9m19s	/	timeout

Precision EDSM

Table: Precision₊@k and Precision₋@k of the DFAs generated with EDSM

(a) Precision₊@k (EDSM)

Log (<i>ℓ</i>)	k = 1	t	k = 2	t	k = 3	t
LOAN	N/A DFA		N/A DFA		N/A DFA	
ROAD	0.489	2.47s	0.117	2.73s	0.018	13.35s
SEPSIS	0.448	3.29s	0.096	9.22s	0.014	4m20s
REIMB	0.160	3.66s	0.011	59.62s	0.001	47m7s
TRAVEL	0.187	10.14s	0.010	8m7s	/	timeout

(b) Precision₋@k (EDSM)

Log (<i>ℓ</i>)	k = 1	t	k = 2	t	k = 3	t
LOAN	N/A DFA		N/A DFA		N/A DFA	
ROAD	0.413	2.14s	0.077	2.58s	0.010	13.63s
SEPSIS	0.358	2.87s	0.063	7.26s	0.008	3m7s
REIMB	0.139	3.92s	0.009	1m40s	0.0004	87m27s
TRAVEL	0.157	16.68s	0.007	10m9s	/	timeout

Generalization MDL

(a) Generalization₊@k (MDL)

Log (ℓ)	$k = 1$	t	$k = 2$	t	$k = 3$	t
LOAN	0.999	2m50s	0.999	3m9s	0.999	3m35s
ROAD	0.997	19s	0.992	18s	0.988	23s
SEPSIS	0.999	21s	0.999	22s	0.998	55s
REIMB	0.998	22s	0.997	30s	0.994	6m57s
TRAVEL	0.998	33s	0.997	3m42s	/	timeout

(b) Generalization₊@k (Declare Miner)

Log (ℓ)	$k = 1$	t	$k = 2$	t	$k = 3$	t
LOAN	0.999	53s	0.999	57s	0.999	1m20s
ROAD	0.994	20s	0.987	21s	0.978	21s
SEPSIS	0.998	58s	0.997	1m12s	0.996	2m48s
REIMB	0.992	6m36s	0.984	3m56s	0.975	6m8s
TRAVEL	/	timeout	/	timeout	/	timeout

Generalization RPNI

(a) Generalization₊@k (RPNI)

Log (ℓ)	k = 1	t	k = 2	t	k = 3	t
LOAN	1.0	55s	1.0	1m52s	0.999	21m2s
ROAD	0.999	18s	0.998	21.58s	0.968	27s
SEPSIS	1.0	24s	1.0	31s	1.0	3m25s
REIMB	0.999	19s	0.999	1m9s	0.999	50m57s
TRAVEL	0.999	49s	0.999	7m7s	/	timeout

(b) Generalization₋@k (RPNI)

Log (ℓ)	k = 1	t	k = 2	t	k = 3	t
LOAN	1.0	1m29s	1.0	4m27s	1.0	1h38m
ROAD	1.0	23s	1.0	24s	1.0	32s
SEPSIS	1.0	22s	1.0	34s	1.0	3m28s
REIMB	1.0	29s	1.0	1m32s	1.0	1h20s
TRAVEL	1.0	40s	1.0	7m4s	/	timeout

Generalization EDSM

(a) Generalization₊@k (EDSM)

Log (ℓ)	$k=1$	t	$k=2$	t	$k=3$	t
LOAN	N/A DFA		N/A DFA		N/A DFA	
ROAD	0.999	31s	0.998	32s	0.989	39s
SEPSIS	1.0	11m17s	0.999	16m18s	0.999	17m39s
REIMB	0.999	50m20s	0.999	57m39s	0.998	1h24m
TRAVEL	0.999	7h23m	0.998	15h50m	/	timeout

(b) Generalization₋@k (EDSM)

Log (ℓ)	$k=1$	t	$k=2$	t	$k=3$	t
LOAN	N/A DFA		N/A DFA		N/A DFA	
ROAD	1.0	29s	1.0	35s	1.0	45s
SEPSIS	1.0	23m31s	1.0	31m50s	1.0	37m3s
REIMB	1.0	1h11m	1.0	48m35s	1.0	2h1m
TRAVEL	1.0	10h8m	1.0	17h28m	/	timeout

Simplicity

Table: Simplicity of the DFAs generated with MDL, RPNI, EDSM, and Declare Miner

(a) MDL

Log (ℓ)	$S(\mathcal{M})$	t
LOAN	(42, 123)	23344 ms
ROAD	(8, 33)	249 ms
SEPSIS	(4, 19)	360 ms
REIMB	(8, 53)	393 ms
TRAVEL	(8, 107)	2352 ms

(b) RPNI

Log (ℓ)	$S(\mathcal{M})$	t
LOAN	(185, 2442)	723 ms
ROAD	(23, 99)	462 ms
SEPSIS	(39, 313)	274 ms
REIMB	(39, 568)	261 ms
TRAVEL	(51, 1057)	1380 ms

(c) EDSM

Log (ℓ)	$S(\mathcal{M})$	t
LOAN	/	/
ROAD	(21, 97)	2164 ms
SEPSIS	(55, 398)	947039 ms
REIMB	(38, 521)	3489343 ms
TRAVEL	(59, 1088)	21105718 ms

(d) Declare Miner

Log (ℓ)	$S(\mathcal{M})$	t
LOAN	(435, 1673)	21732 ms
ROAD	(121, 433)	710 ms
SEPSIS	(631, 3259)	2409 ms
REIMB	(3565, 13168)	4182 ms
TRAVEL	(182763, 1424349)	38752548 ms

Simplicity (synthetic)

Table: Simplicity of the DFAs generated with MDL, RPNI, EDSM and L*

(a) MDL

Log(ℓ)	$S(\mathcal{M})$	t
log_3.10	(3, 11)	728 ms
log_3.15	(2, 16)	435 ms
log_3.20	(2, 17)	343 ms
log_3.25	(2, 15)	466 ms
log_3.30	(2, 16)	457 ms
log_5.10	(3, 19)	387 ms
log_5.15	(5, 27)	472 ms
log_5.20	(4, 28)	432 ms
log_5.25	(2, 18)	434 ms
log_5.30	(3, 20)	507 ms
log_7.15	(3, 17)	388 ms
log_7.20	(2, 17)	428 ms
log_7.25	(2, 18)	452 ms
log_7.30	(2, 17)	489 ms
log_10.20	(3, 19)	373 ms
log_10.25	(2, 18)	455 ms
log_10.30	(2, 18)	421 ms

(b) RPNI

Log(ℓ)	$S(\mathcal{M})$	t
log_3.10	(7, 27)	202 ms
log_3.15	(6, 45)	278 ms
log_3.20	(7, 53)	222 ms
log_3.25	(11, 73)	206 ms
log_3.30	(11, 66)	211 ms
log_5.10	(5, 37)	206 ms
log_5.15	(6, 53)	238 ms
log_5.20	(6, 58)	218 ms
log_5.25	(11, 70)	209 ms
log_5.30	(9, 92)	215 ms
log_7.15	(5, 40)	241 ms
log_7.20	(7, 65)	210 ms
log_7.25	(8, 83)	213 ms
log_7.30	(8, 71)	224 ms
log_10.20	(6, 56)	268 ms
log_10.25	(7, 73)	219 ms
log_10.30	(7, 73)	255 ms

(c) EDSM

Log(ℓ)	$S(\mathcal{M})$	t
log_3.10	(6, 22)	218 ms
log_3.15	(5, 42)	303 ms
log_3.20	(5, 52)	277 ms
log_3.25	(10, 88)	406 ms
log_3.30	(5, 52)	354 ms
log_5.10	(5, 36)	326 ms
log_5.15	(7, 53)	346 ms
log_5.20	(8, 73)	363 ms
log_5.25	(8, 85)	430 ms
log_5.30	(11, 123)	714 ms
log_7.15	(6, 47)	370 ms
log_7.20	(8, 77)	478 ms
log_7.25	(7, 70)	460 ms
log_7.30	(9, 94)	427 ms
log_10.20	(6, 58)	303 ms
log_10.25	(7, 79)	455 ms
log_10.30	(7, 71)	457 ms

(d) L*

Log(ℓ)	$S(\mathcal{M})$	t
log_3.10	(24, 277)	373 ms
log_3.15	(77, 1217)	462 ms
log_3.20	(124, 2092)	709 ms
log_3.25	(262, 3916)	1780 ms
log_3.30	(256, 4081)	2215 ms
log_5.10	(46, 631)	436 ms
log_5.15	(101, 1601)	670 ms
log_5.20	(174, 2942)	890 ms
log_5.25	(300, 5084)	2584 ms
log_5.30	(407, 6903)	3768 ms
log_7.15	(99, 1569)	483 ms
log_7.20	(180, 3044)	846 ms
log_7.25	(382, 6478)	3541 ms
log_7.30	(361, 6121)	3050 ms
log_10.20	(152, 2568)	867 ms
log_10.25	(347, 5883)	3199 ms
log_10.30	(429, 7277)	4703 ms