

Hybrid Process Mining: Inference & Evaluation Across Imperative & Declarative Approaches

Christoffer Olling Back

This thesis has been submitted to the
PhD School of The Faculty of Science,
University of Copenhagen



UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE

Abstract

Processes modeling languages are often characterized as falling within an imperative or declarative paradigm: the former describing a sequential flow of events, while the latter focuses on rules, constraints, goals or properties. Research suggests that highly structured processes are more aptly characterized by imperative models, while more flexible processes are more succinctly modeled using declarative models. Formalisms have also been proposed that combine aspects of both paradigms, aiming to harness the strengths of each.

This thesis explores the task of automatically learning various classes of process models from data. We begin with an information theoretic approach to discovering structure in event logs: investigating empirically whether various entropy estimators are able to distinguish process structures associated with respectively imperative and declarative models. The empirical focus continues with a systematic comparison, based on model-agnostic metrics, of existing process discovery algorithms representative of the two paradigms. An in-depth case study is then presented: after an on-the-ground investigation of workflows at a surgical ward, we apply process mining techniques to the associated dataset to build a model of patient flows, competing for resources and subject to regulations. The resulting stochastic model is then parametrized using real-world data, and we demonstrate how simulation can be used to predict resource utilization as well as determine the risk of policy violation. Finally, novel algorithms for learning and parametrizing declarative and hybrid process models are presented. We present a view of process mining in terms of computational learning theory, present theoretical results for a restricted class of binary miners, and end by outlining a probabilistic approach to learning hybrid models that capture resource dependencies between process flows.

Abstract

(Danish)

Proces modelerings formalismer bliver ofte karakteriseret som værende indenfor paradigmerne imperative eller deklarative: den første beskriver en sekventiel række af hændelser, mens den sidste fokuserer på regler, betingelser, mål, eller egenskaber. Forskningen tyder på, at meget regulære/strukturerede processer beskrives bedst ved imperative modeller, mens mere fleksible processer kan beskrives mere kortfattet og passende ved hjælp af deklarative modeller. Formalismer har også været bekskrevet som kombinerer de to paradigmer - såkaldte hybride formalismer/modeller - med henblik på at udnytte fordelene ved begge.

Denne adhandling undersøger udfordringen i, automatisk at lære sådanne modeller fra data. Vi begynder med en tilgang baseret på informations teori, hvor målet er at finde struktur i event log data. Vi undersøger empirisk om forskellige estimatorer for entropy kan skelne mellem process strukturer som stammer fra henholdsvis imperative og deklarative modeller. Den empiriske fokus fortsætter med en systematisk sammenligning - baseret på model-blinde metrikker - af eksisterende "process discovery" algoritmer som er representative for de to paradigmer. Derefter beskriver vi en dybdegående case-study: efter en undersøgelse af arbejdsgangene på en operationsafdeling på the Royal Infirmary of Edinburgh, analysere vi et stort tilhørende dataset ved hjælp af process mining algoritme for at bygge en model af patient forløb under betingelse af begrænsninger på delte resourcer og gældende regulativer. Resultatet er en stokastisk model som parametriseres på baggrund af virkelige data, og kan bruges til at forudsige forventet belastning af resourcer og sandsynligheden for brud på service krav. Til sidst fremlægges der nye algoritmer til at lære strukturen af, og parametrisere, deklarative og hybride modeller. Vi gennemgår en analyse af process mining set fra computational learning theory, præsenterer teoretiske resultater for en begrænset klasse af binære process mining algoritmer, og afslutter ved at fremlægge en probabilistisk tilgang til at lære hybride modeller som beskriver resource afhængigheder mellem flere processer.

Contents

0.1	Objectives	3
0.2	State of the Art	3
0.3	Conclusions & Future Work	4
0.3.1	On the Imperative/Declarative Distinction	5
0.3.2	On the Inevitability of Probability	9
0.4	Code	10
0.5	Overview of Papers	10
1	An Information Theoretic Approach	12
1.1	Discussion	12
1.2	Summaries	12
2	Evaluation and Comparison	14
2.1	Discussion	14
2.2	Implementation	16
2.3	Summaries	20
3	Applications	22
3.1	Discussion	22
3.2	Summaries	23
4	Mining Algorithms & Learning	26
4.1	Discussion	26
4.2	Summaries	26
5	Manuscripts	34
	Towards an Entropy-based analysis of Log Variability	34
	Entropy as a Measure of Log Variability	52
	Towards an Empirical Evaluation of Imperative and Declarative Process Mining	80
	Imperative or Declarative: An Empirical Comparison of Logs . . .	86

Mining Massive SAP Transaction Logs	104
Modelling Operating Theatre Workflows: A Brief Literature Review	107
Mining Patient Flow Patterns in a Surgical Ward	116
Stochastic Workflow Modeling in a Surgical Ward: Towards Simulating and Predicting Patient Flow	127
Discovering Responsibilities with Dynamic Response Graphs . . .	152
DisCoveR: Accurate & Efficient Discovery of Declarative Process Models	167
Weighing the Pros and Cons: Process Discovery with Negative Examples	191
Process Mining via Hidden Markov Models: A Feasibility Study	207
Towards Inference of Resource Dependency Grammars from Event Streams	216

0.1 Objectives

The objective of this thesis is to investigate the notion of hybrid process mining, i.e. learning mixed-paradigm process models from event logs, consisting of both imperative and declarative modeling constructs. A corollary objective - one which proved to warrant significant attention - is addressing methods for one-to-one evaluation across paradigms. Finally, the development of a set of software tools for accomplishing aforementioned objectives and facilitating further work on these topics.

0.2 State of the Art

Process mining is a relatively young field of research which came to prominence in the early 2000's borne of the growing need to develop automated approaches to deriving actionable insights from the enormous amount of data surrounding (business) process execution generated by modern IT systems, such as event-resource planning (ERP) systems like SAP, other process-aware information systems, as well as unstructured data stemming from workflow executions. Such processes may span heterogeneous systems from customer relation management (CRM), accounting and HR software, to logistics, inventory and manufacturing software [8].

In contrast to classical data mining and machine learning tasks, which are sometimes described as a “flat” approach of learning black-box mappings from input data to target classes/objective functions, process mining takes a “structured” approach of learning complex models with executable semantics. As a simple example, consider a linear regression task in which the underlying structure of the model is given a priori and the learning task consists of optimizing parameters to fit training data.

While this distinction is arguably overly simplistic and ignores recent development in structured machine learning approaches [47] - and approaches such as Bayesian belief networks which involve learning model structure as well as parametrization [35] - it highlights the focus on learning end-to-end, interpretable process models, specifically from event log data. As a sequential/temporal learning task, process mining is not unrelated to other tasks involving similar data, such as speech recognition, natural language processing, time series analysis, and activity recognition - and indeed, there is some overlap. One distinction lies in the general focus on learning certain classes of high-level process models with sophisticated semantics capable of capturing complex control constructs like loops and concurrency.

This focus stems from the roots of process mining in disciplines like formal language theory, model checking and the theory of computation - specifically models of computation such as Petri nets. This class of models has dominated the field, but “declarative” (often logic-based) approaches have also played an important role and are increasingly relevant due to the ubiquity of flexible workflow processes such as knowledge work [50]. Hybrid process models which com-

bine these paradigms have drawn attention in recent years but remain under-addressed in the research. This thesis is intended to contribute to addressing that gap.

A secondary issue this thesis addresses is the presumed gap between established machine learning approaches and process mining - specifically in terms of evaluation. After all, proposing an algorithm for generating a process model given an event log is arguably trivial absent an evaluation measure. The crucial question concerns the quality of the generated model w.r.t. to data, e.g. is a hybrid model *better* than alternative models? This is the fundamental question in learning from data: what is the relationship between the *data*, the *algorithm*, and the *loss function* [1]. The latter - the need for robust evaluation procedures and metrics - is an aspect of the process mining task which has seen growing interest in recent years, with comprehensive benchmarking studies [4], as well as critical inquiries into existing metrics from within the community [51], and the establishment of the Process Discovery Contest which is framed as a binary classification tasks and evaluated using a blind training/testing procedure ¹.

In scenarios that are not amenable to being framed as a classification task (e.g. missing labels for data) - the process mining essentially becomes an unsupervised learning task. This has often been the argument for framing process mining as a descriptive data mining task, usually performed on in-sample data alone. When evaluating on out of sample data, an additional alignment procedure is required to be able to calculate log-model replay based metrics, since the model may not in fact permit the unseen behavior and some notion is needed of how *far* the model is from capturing the observed data [52, 2]. One of the conclusions of this project is that a more convincing alternative can be found by adopting a probabilistic approach and corresponding likelihood-based evaluation metrics. In this sense, the project comes full circle from the initial work regarding information theoretic approaches, to the applied stochastic models and the final paper proposing a probabilistic hybrid model.

0.3 Conclusions & Future Work

Some of the open questions that arose during work on the individual papers are listed in the roadmap in Table 1, which also indicates the “red thread” in the flow of research via references to those papers which address specific questions. Roughly speaking, the course of the research project can be said to have come full circle via the following sequence of research focus:

Inference (simple) \rightarrow Evaluation \rightarrow Application \rightarrow Inference (complex)

¹<https://www.tf-pm.org/competitions-awards/discovery-contest>

In reality, most of the works focus on more than one of these aspects, as reflected in the roadmap. Nonetheless, this captures the general progression and take-away message of the project. Attempts at developing inference techniques in the papers in Chapter 1 highlighted the need for rigorous evaluation methods, addressed in Chapter 2.1, these were extended and applied in real-world settings in Chapter 3.1, and finally lessons from all the preceding chapters inspired more principled approaches to developing mining/inference algorithms in Chapter 4.1.

Conclusions and future work are addressed in the individual chapters and papers, but overall a key take-away from the project is that there exists a rich potential in further developing process mining techniques which integrate formal modeling techniques - imperative, declarative or hybrid - with the insights from the statistical/machine learning community. Resulting models will be more expressive, interpretable, and statistically well-founded. Natural approaches for bridging the apparent gap do indeed exist - I hope to illustrate this in the work that follows.

What follows in the remainder of this section are two brief notes summarizing some remaining thoughts regarding the imperative/declarative distinction and the role of probability.

0.3.1 On the Imperative/Declarative Distinction

Are the concepts *imperative* vs. *declarative* a distinction without a difference? In the course of this project, I have found the task of explaining the distinction to non-experts challenging, and at times had some doubt of the usefulness of the distinction. At a high level, the distinction is intuitive: on the one hand a focus on *control-flow* vs. *constraints* or *rules* such that imperative models outline exactly the sequence of actions/decision whereas declarative models allow any behavior that does not violate the proscribed constraints. This is captured by the classic graphic, recreated from [44] in Figure 1.

In the end, a concrete sequence of events must be settled upon, even in a process which is specified declaratively - just as a logic program may be specified by the programmer, but relies on an underlying reasoning engine to find a solution. Similarly, database query languages like SQL can be specified declaratively, but obviously require an underlying search strategy to find the relevant results, and often a familiarity with the workings of the underlying engine is necessary to formulate efficient queries.

The structure vs. flexibility distinction, while intuitively straightforward, becomes muddled when we try to pin it down formally. We know, for example that standard Petri nets are *more* expressive than Linear Temporal Logic (LTL) or Dynamic Condition Response (DCR) Graphs. While *extended* versions DCR Graphs and Petri nets are both Turing complete, e.g. DCR* with subprocesses [27] and Petri nets with inhibitor arcs [57], infinite color nets [45] and timed differentiable nets [31] - standard Petri nets can capture all regular languages as well as some context-free languages [19]. In contrast, the language of LTL is a subset of ω -regular languages [54] while DCR Graphs can capture regular and ω -regular languages [38]. Furthermore, we know that the LTL templates of the

PAPER	Inference	Evaluation	Application	DESCRIPTION	OPEN QUESTIONS	Addressed By
Towards an Entropy-based Analysis of Log Variability [9]	X	X		Preliminary analysis of entropy estimators to determine best mining paradigm	Normalized estimators Efficient estimators Clustering-based estimators Ground truth evaluation	[11] [11] [11] [11]
Entropy as a Measure of Log Variability [11]	X	X		Extended analysis of entropy estimators with evaluation against real logs and artificial ground truth models	Efficient implementation Evaluation of miner choice Log splitting on local structure Estimator stability analysis Sample size analysis	Sec. 2.2 [10] - - -
Towards an Empirical Evaluation of Imperative and Declarative Process Mining [10]		X		Outline of metrics applicable across paradigms and statistical evaluation	Efficient implementation Statistical evaluation	Sec. 2.2 [12]
Imperative or Declarative? An Empirical Evaluation of Event Logs [12]		X		Analysis of different event logs' suitability to imperative or declarative miners	Overcoming perfect fitness requirement More comprehensive comparison Statistical validity of metrics	Sec. 0.3.2 [6], [7] - -
Mining Patient Flows in a Surgical Ward [40]	X	X	X	Analysis of patient flow flow data through surgical ward w.r.t. control-flow and cycle time	Modeling process constraints from Domain Knowledge Harnessing conditional timing distributions	[41] [41] -
Stochastic Workflow Modelling in a Surgical Ward [41]	X	X	X	Modeling of patient flow based on domain knowledge regarding constraints and dependencies. Probabilistic parametrization of model	Mining resource dependencies Mining declarative constraints from metadata/text Conditional Bayesian models of transition probabilities	[7] - -
Discovering Responsibilities with DCR Graphs [39]	X	X		Novel algorithm for mining DCR Graphs with evaluation	Detailed formalization Efficient implementation Case-study	[13] [13] [13]
DisCoveR: Accurate & Efficient Discovery of Declarative Process Models [13]	X	X	X	Improved implementation, thorough formalization and evaluation	Parameter search via optimization w.r.t. evaluation metrics More complex DCR structures	- -
Weighing the Pros & Cons: Process Discovery with Negative Examples [49]	X	X		Provides formal foundation for unary and binary process mining, novel binary mining algorithm	VC-dimension analysis PAC analysis & generalization bounds	- -
Inferring Process Models via Hidden Markov Models [6]	X	X		Outlines a bottom-up, approach to learning process models via translation from transition systems	Translation procedures to Petri nets, declarative, and hybrid models	
Towards Inference of Resource Dependency Grammars from Event Streams [7]	X			Outlines and implements a hybrid process grammar integrated with factorial hidden Markov models. Proposes inference procedures	End-to-end inference procedures for growing parse tree via expectation maximization Implementation of sampling procedures Evaluation	- - -

Table 1: Roadmap of the work in this thesis, indicating the focus of the papers and open questions raised underway. The flow of research can be followed via the *Addressed By* column, with questions for future work indicated by -

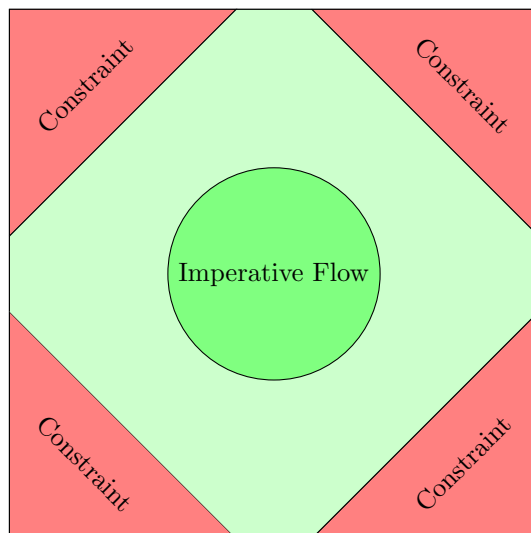


Figure 1: An abstract representation of the imperative/declarative distinction recreated from [44]. The light green region represents behavior that is permitted by a process, but not explicitly permitted by the imperative model.

Declare language can be translated into Petri net constructs (albeit requiring reset/inhibitor arcs) [26].

So can we say that declarative languages are better suited to capture flexible processes when they, in several cases, are less expressive than Petri nets? Likely, the more accurate claim is that for processes with a large degree of variability, declarative models more *succinctly* capture the most relevant requirements of the process, facilitating interpretability and understandability by avoiding so-called spaghetti models that enumerate an excessive number of pathways.

This points to what I believe are the true advantages of the declarative paradigm. In general, they:

- More closely resemble **natural language**
- Succinctly capture the **high-level** aspects of a system/process
- Capture **properties** of a system, rather than execution details

In some work, this distinction is very clear: consider an example from [28], ensuring straightforward Metric Interval Temporal Logic (MTL) constraints (declarative) on the movement of a robot arm whose motion is dictated by a system of differential equations (flow-based). Whether the arm veers slightly in one direction or another is unimportant, so long as it enters the correct positions within the time constraints.

In this example the distinction between the imperative and declarative components is stark and unambiguous. Arguably this is not always the case in BPM where Petri nets are the de facto modeling paradigm. The term *flow-based* seems quite aptly to describe a trajectory space determined by a system of differential equations - is it as appropriate to describing Petri nets? Even moderately complex Petri net structures capture behavior that can be quite tricky to follow in terms of anything resembling a “flow”: with loops, parallelism, complex conditions, and even counting.

Finally, it is a bit curious that so much research in declarative languages - at least within the business process modeling community - has so closely mirrored the imperative approach in strictly modeling temporal relations between actions. Perhaps a more fruitful focus is to capture high-level properties of the system *state* declaratively - in a manner akin to that encountered in the model checking literature. This thought is in part what inspired the final paper in this compendium, which admittedly only scratches the surface of this line of work.

One important nuance afforded by the declarative approach is the ability to perform *runtime verification* by distinguishing between constraints being *permanently* satisfied/violated and *possibly* violated [56]. A non-accepting terminal state in the the resulting automaton translates as a permanent violation of the model, for example. Alternatively, rewrite rules can be employed such that re-playing an event results in a new set of formulas representing the constraints to be satisfied in the remaining execution of the process [14]. This perspective lends itself naturally - not only to prediction, but also prescription - of best courses of action.

One approach worth mentioning is based on linear logic (not to be confused with LTL) is a resource-oriented logic, which allows for correct-by-design, optimally concurrent, process formulation [43].

Recently, extensions to such *propositional* temporal logics incorporating timing and data have seen growing interest. Going beyond simple ordering constraints, these logics allow specification of constraints w.r.t. absolute and relative time and quantified statements about the world. These allow the formulation of constraint such as, “the vehicle must enter region *A* within the first 5 minutes, and subsequently enter region *B* within 3 minutes of entering region *A*”.

Metric (interval) temporal logic (MTL/MITL) is one such logic, variants of which differ in terms of semantics: point-based vs. interval-based; and time representation: dense vs. countable [42]. Some of these fragments can be checked via mapping to LTL [33] or via timed automata [56], allowing for time-based runtime verification and operational support [15].

Metric *first-order* temporal logic (MFOTL) extends MTL with first order formulae quantified over infinite domains [34], allowing constraints stating that first-order statements be true at a given time or interval. Conformance checking for MFOTL is in general undecidable, but for a subset of constraints based on Declare, algorithms are presented in [17].

DCR graphs differs from the temporal logics discussed above, in particular in its non-monotonic properties. That is, adding a rule does not necessarily

further restrict the behavior permitted by a model - it may indeed increase it. Nonetheless, DCR constraints can also be converted to (Büchi) automata for conformance checking [37].

0.3.2 On the Inevitability of Probability

Most discussions of process modeling or model checking begin with a non-probabilistic approach. The model simply captures the possible behavior of a system, remaining agnostic to the likelihood of alternate traces: they are either permitted or not permitted. Sometimes this binary framing leads to the view of process models as classifiers between legal/illegal behaviors (combining model and decision procedure). As I note in [6], at the very least, this can be seen as equivalent to a *bound* on the probabilities over traces, essentially capturing an entire class of probabilistic models.

To see this, consider the underlying transition system of a Petri net and denote by m_i the state corresponding to marking i . If an edge exists between states m_1 and m_2 , then clearly the probability of seeing marking 2 after marking 1 is not zero, i.e. it is possible. Conversely, if no edge exists between, say m_1 and m_3 , then the probability of seeing m_3 and after m_1 is exactly zero according to the model, i.e. it is impossible.) Formally,

$$\mathbb{P}(m_2 \mid m_1) > 0 \qquad \mathbb{P}(m_3 \mid m_1) = 0$$

Obviously all probabilities will be bounded above by 1 and the sum of all outgoing transitions will need to sum to 1, unless it is a sink state (alternatively a sink state could have a single self-transition:

$$\forall j. \sum_i \mathbb{P}(m_i \mid m_j) = 1$$

Depending on the number of outgoing transitions, we can use this to derive further bounds. The simplest of which is that when only one outgoing transition exists, it will have a probability of 1, i.e. it is inevitable.

Clearly, these bounds permit an enormous variety of distributions with drastically different behaviors. Nonetheless, the point remains: *any non-probabilistic process model implicitly defines an entire class of probabilistic models*. Thus, they cannot be said to be fully agnostic to the probabilistic view.

The probabilistic aspect enters the picture again once the task becomes to learn models *from data*. In it's most loose formulation, this task ("process mining") amounts to a mapping:

$$\gamma; \mathcal{L} \rightarrow \mathcal{H}$$

from the space of event logs \mathcal{L} to the space of process models \mathcal{H} (or hypotheses) [13]. Devising such a mapping is nearly trivial without the notion of a loss functions capturing *how well* the model fits the data. This is where probability and notions of sampling become unavoidable.

All metrics for evaluating model quality are functions of the model and data (event log), and all rely on some frequency-based measure. That is, the metric *counts* some phenomenon, i.e. number of traces that violate the model; the number of remaining tokens in a Petri net; the number of escaping edges relative to state visits during log replay. Probability necessarily enters the picture by virtue of the metrics being functions of the dataset at hand.

Let us consider the situation where we are given a guarantee that all possible traces permitted by a model are represented in our dataset. Even then, we do not avoid the probabilistic aspect: how many times have we seen every trace variant, and how large is the event log? Perhaps the provider of the dataset tells us, “the model is not probabilistic”, but then what determines the execution trace at a given time? If the execution is deterministic, well then we know every state transition has a probability of 1.0.

Even in the contrived example above, the dataset is unavoidably a *sample* from a distribution. Realistically, we can rarely assume that our event log contains examples of all possible behavior. Hence, the challenge becomes: given a finite sample from the generating process, how likely is γ to generate a model that achieves our objective, whether correctly predicting future events, detecting anomalies, or even just generating models that users deem “informative”.

To my knowledge, the formulation given above - which amounts to the standard probably approximately correct (PAC) paradigm (well established in computational learning theory [1]) - is not accounted for in most process mining research. I believe this to be an important aspect that needs to be addressed. I make the first attempts at doing so by appropriately formalizing process discovery in these terms in [13], and by developing a fully probabilistic approach in [7].

0.4 Code

All publicly available code developed in conjunction with the research presented in this thesis can be found at:

<https://github.com/backco/phd>

0.5 Overview of Papers

YEAR	TITLE	FIRST AUTHOR	PAPER TYPE	VENUE	PUBLICATION STATUS
2017	Towards An Entropy-based Analysis of Log Variability [9]	✓	Short	BPM 2017 (BPAI Workshop)	Published
2019	Entropy as a Measure of Log Variability [11]	✓	Journal	Journal of Data Semantics	Published
2018	Towards an Empirical Evaluation of Imperative and Declarative Process Mining [10]	✓	Short	EmpER 2018	Published
2018-	Imperative or Declarative? An Empirical Comparison of Event Logs [12]	✓	Full		Resubmission Pending
2020	Mining Patient Flow Patterns in a Surgical Ward [40]	✓	Full	HealthInf (BIOSTEC)	Published
2020	Stochastic Workflow Modeling in a Surgical Ward: Towards Simulating and Predicting Patient Flow [41]	✓	Journal	Springer CCIS Book Series	Awaiting Publication
2019	Discovering Responsibilities with Dynamic Condition Response Graphs [39]		Full	CAiSE	Published
2020	DisCoveR: Accurate & Efficient Discovery of Declarative Process Models [39]	✓	Journal	Journal on Software Tools for Technology Transfer	Accepted Publication Pending
2020-	Weighing the Pros & Cons: Process Discovery with Negative Examples [49]		Full		Resubmission Pending
2019	Inferring Process Models via Hidden Markov Models [6]	✓	Short		Draft
2020	Towards Inference of Resource Dependency Grammars from Event Streams [7]	✓	Full		Draft

Table 2: Overview of papers in compendium.

Chapter 1

An Information Theoretic Approach

1.1 Discussion

The work in this chapter stems directly from the driving hypothesis of this project: that declarative modeling languages are better suited to capturing flexible processes, and imperative languages to capturing highly structured processes. It represents a first step in building an inference procedure for building hybrid models by developing methods for identifying more or less structured portions of processes as candidates for either mining paradigm.

A natural measure of structure is entropy, a concept stemming from Information Theory, and which underlies many pattern recognition algorithms, for example as *information gain* in decision trees or in the minimization of Kullback-Leibler divergence within the derivation of the expectation maximization (EM) algorithm [16, 36]. This led to an in-depth investigation of various formulations and estimators of entropy in symbolic sequences, and whether they indeed correlate with processes being more declarative or imperative in [11].

1.2 Summaries

Towards an Entropy-based Analysis of Log Entropy [9] outlines the basic approach and defines some estimators along with a preliminary evaluation.

Entropy as a Measure of Log Variability [11] significantly extends the first paper with several additional estimators (many of which were discussed as future work in the short paper). Furthermore, we addressed some of the theoretical aspects more thoroughly, such as the assumptions regarding stationarity and ergodicity underlying many estimators and include a thorough exposition of the distinctions between entropy and entropy *rate* of a process.

The empirical investigation is also a crucial component of this publication. As before, we present results from the standard event logs in the process mining community, based on indications in the literature that these logs are more or less suited to imperative/declarative modeling. Perhaps more convincingly, we present results from artificially generated logs¹ that try to capture various control constructs, noise, parallelism/concurrency, all using Petri net models. Then we also generated logs from Declare models, using modeling constructs built systematically on a spectrum from more to less flexibility. The results of all estimators across these event logs give good indications as to various estimators strengths and shortcomings. Finally, estimator complexity and runtime is addressed.

The practicalities of computing some estimators was not trivial. As many aspects of the implementation turned out to be useful for the model evaluation discussed in Chapter 2.1, we will address it in that chapter.

¹Some from existing literature, some we built ourselves.

Chapter 2

Evaluation and Comparison

2.1 Discussion

In this body of work, we present a systematic comparison of imperative and declarative model in terms of evaluation metrics that are commonly accepted in the process mining community, but have nearly exclusively been applied to Petri net models.

The simplest, and least granular, formulation of evaluation metric is that of trace- or case-level fitness, i.e. whether entire traces are permitted by a model or not [3]. Note that an event log is generally assumed to contain only positive examples and no negative labels, precluding the formulation of traditional classification metrics. This metric is used as the basis for the *Parsing Measure* of model quality in [55], and as the authors note, it glosses over important details in the degree and nature of model-log inconsistency. Instead, measures of violations at the level of the individual event are needed, not only for accuracy, but to assist in locating potential areas for improvement [29, 30].

A prevalent approach to quantifying the *degree* of model-log incongruity is that of alignment, which defines a distance function between a trace and a model based on the minimum number of extra “moves” that must be inserted into either trace or model to replay the trace and reach an accepting state in the model [2, 52]. Any trace replayable on a model without modification will have an alignment distance of zero. Alignment can also capture the severity of incongruities by assigning costs to individual edits [21]. Alignment based approaches are not formalism-specific and have been applied to declarative models in [21, 24, 23].

As we note, in our approach we have tried to avoid relying on alignment, since this essentially introduces an additional confounding variable between the mining algorithm itself and evaluation against the data, since it introduces the additional layer of the alignment algorithm and weightings on the edits/moves in searching for optimal alignments.

It has been more common to report formalism-specific metrics, such as token-

based measures for Petri nets (specifically workflow nets) based on the number of tokens that were missing in the model when replaying a trace/log, and the number remaining in the model after replay, relative to the total number of tokens consumed and produced overall [48]. In [46], a conformance checking technique based on data-aware Petri nets is presented which handles both control-flow and timing constraints by modeling time as a data attribute. Some declarative models can be translated into workflow nets, and are thereby amenable to token-based metrics. See [25] for an overview of translations from Declare constraints to Petri nets with reset and inhibitor arcs.

Evaluation metrics for declarative models vary depending on the specific formalism, though translation to finite state automata is common. Temporal logics, such as linear temporal logic (LTL), computational tree logic (CTL), CTL*, event calculus [18] form the foundation of much of declarative process modeling, with Declare being a canonical example originally formulated as set of templates based on LTL and since extended (see below). Stochastic process algebras such as PEPA are similar to other declarative languages in their compositional nature and amenability to qualitative analysis while facilitating quantitative analysis via translation to stochastic Markov models [32].

Conformance checking of temporal logic based languages can be done via translation into a set of finite-state automata (FSA), (non-)deterministic FSA (NFA/DFA) in the case of constraints on finite traces, and Büchi automata in the infinite case [20, 22]. The conjunctive nature of these logics means that the product of corresponding automata is sufficient to capture any potential interplay.

This approach forms the basis of the evaluation metrics investigated here [10, 12]: precision and generalization metrics are formulated based on the underlying state transition system such that models based on any formalism can be compared on equal footing, avoiding any reliance on model-specific attributes such as leftover tokens.

Despite being a clever approach to defining formalism-agnostic evaluation metrics, one of the notable shortcomings of transition system metrics is the inability to replay non-fitting traces: if an event is encountered which is not permitted, we simply do not know which state the model should transition to and how to continue¹. For this reason, and because we wanted to avoid introducing confounding alignment procedures, we restricted our investigation to algorithms which could be guaranteed to produce perfectly fitting models.

Clearly this only works when evaluating on in-sample, i.e. training data, since a previously unseen trace variant might be encountered in out-of-sample data. Evaluation on in-sample data is the de-facto standard in the process mining community, which we follow here, though we recognize the shortcomings of doing so. The “generalization” metric is intended to compensate for the lack of evaluation on out-of-sample data, but as we show, it appears doubtful that

¹A probabilistic approach would avoid this, since a probability distribution over states and transitions - rather than fully deterministic semantics - would still permit replay and simply assign a low probability to the trace [6].

it is capturing what it is intended to: namely the trade-off between over- and underfitting.

2.2 Implementation

While not reflected in detail in the publications due to space limitations, a large proportion of the work behind this line of investigation laid in the implementation of a reasonably efficient data structure for replaying event logs on models. While this may sound trivial - and indeed a naive implementation is trivial for small event logs and models - for very large models and event logs, naive implementations become untenable.

The data structure is based on a simple prefix tree built from event logs, where each trace can be seen as a “word” and events and symbols. Each node represents a unique prefix and with it is associated the relevant attributes for computing the evaluation metrics: model state, enabled activities in model and log, and occurrence counts. Furthermore, a map of model state to nodes is maintained for quick lookup of which events visit which states [12]. Figure 2.2 shows an example of the prefix tree based on a Petri net model and simple event log. The time complexity gains stem primarily from avoiding repeated replay of the same trace prefixes.

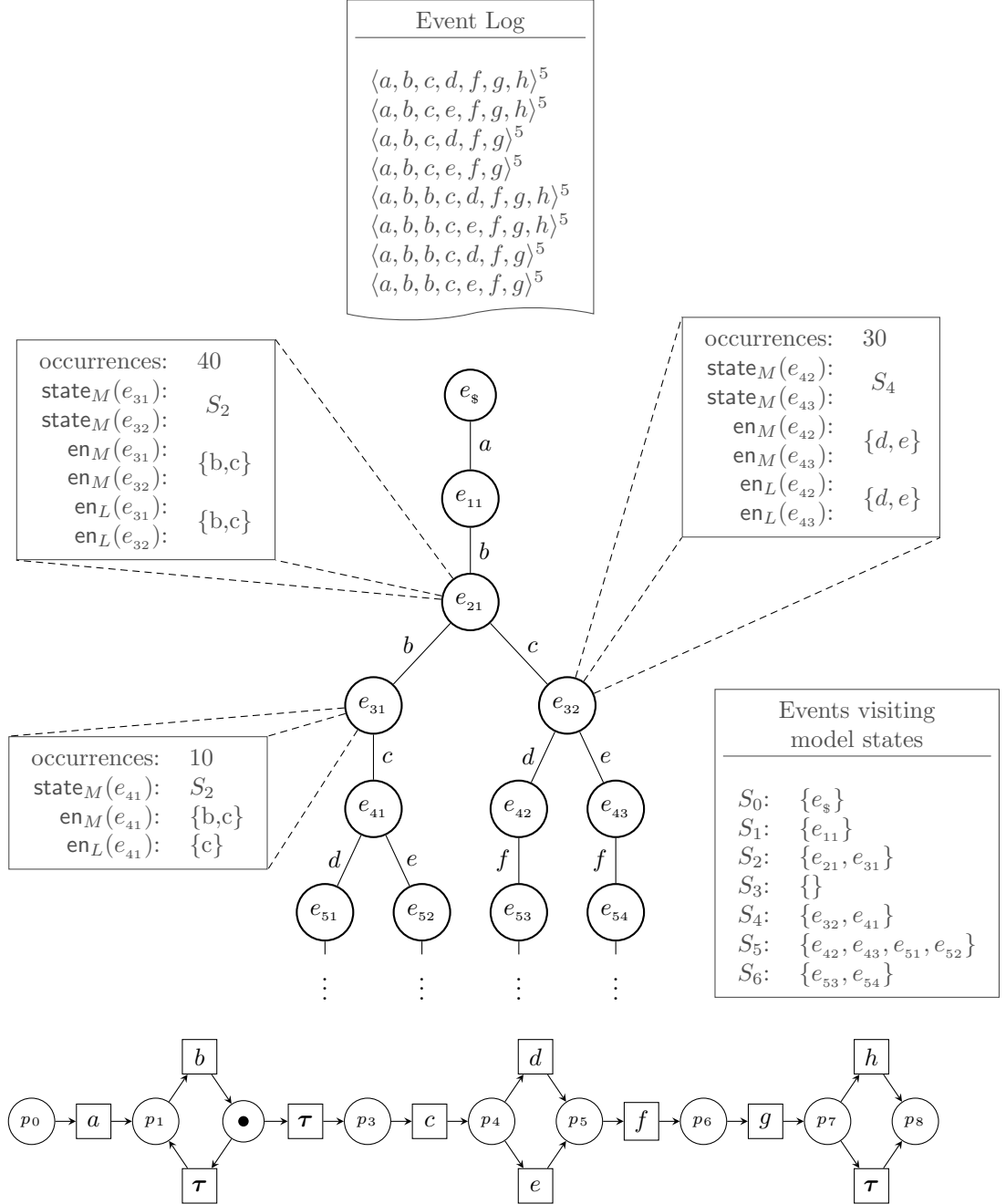
Undoubtedly, even more efficient implementations exist, but this was sufficient for our purposes. As noted in Chapter 1, the same data structure was also employed to compute entropy estimators, again exploiting the structure of the trie, once constructed, to drastically improve time complexity of computing several of the proposed entropy estimators.

One interesting detail arose in implementing the replay mechanism specifically for Petri net with silent transitions (denoted τ) which are commonly generated by some mining algorithms such as the Inductive Miner. The issue arises when multiple silent transitions can be executed in some state.

To see this, consider the Petri net in Figure 2.1. Assume the next activities to be executed in the event log are a and then c . We begin with a token in place 1, then after executing τ_1 , we have a token in places 2 and 3. At this point, if we executed τ_3 , we can still execute τ_2 subsequently, thus enabling a . Now we need to execute c , but the net is currently in a marking with a token in place 5 and no way of getting a token in place 6, which is a precondition for executing transition c .

We have, so to speak, moved to far ahead in our search for a path to enabling transition a . This could result from a depth first search or a breadth first search which happens to choose to the wrong path in the reachability graph leading to transition a being enabled. This is illustrated in Figure 2.2 which shows the full reachability graph for the Petri in Figure 2.1. The bold path denotes the shortest path to enabling a , but several other paths exist which lead to a being enabled, but unnecessarily executing other silent transitions and inadvertently blocking the execution of subsequent transitions.

In our implementation, we found that the intuitive strategy of greedily choos-



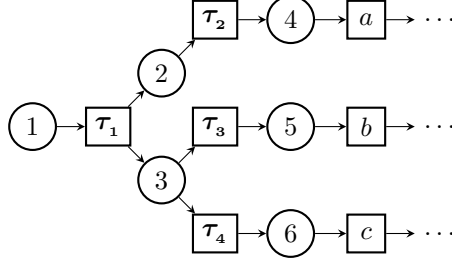


Figure 2.1: A Petri net with multiple silent transitions. When searching for a firing sequence of silent transitions, unnecessarily firing τ_3 or τ_4 could block the subsequent execution of b or c

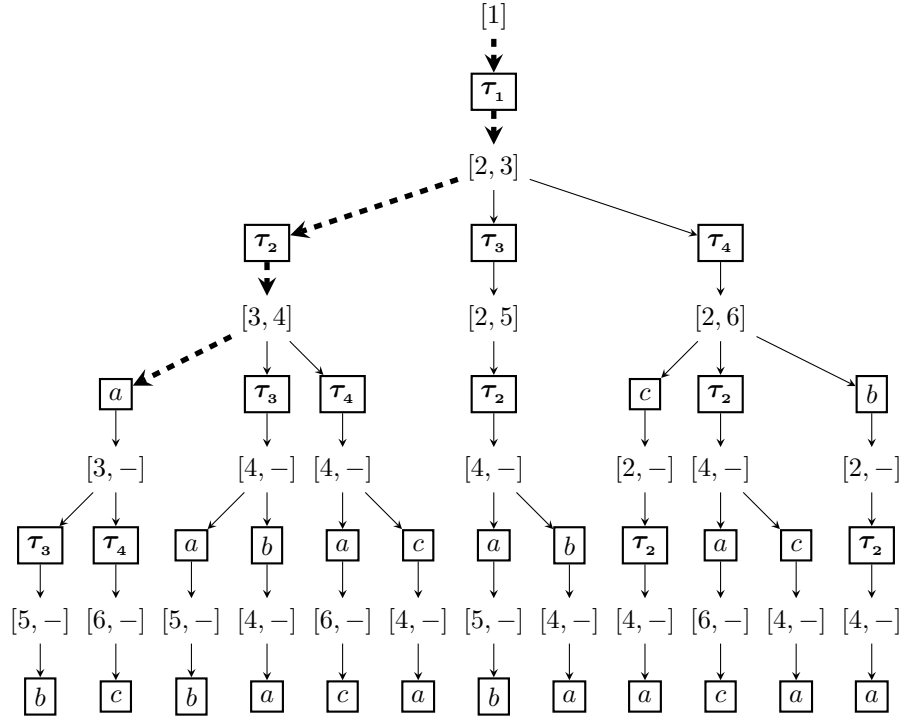


Figure 2.2: The reachability graph of the Petri net in Figure 2.1. The square nodes correspond to the transitions in the Petri net, numbers in brackets denote which place nodes contain a token. Multiple paths exist to enable transition a , but only the shortest path (dashed) avoids blocking subsequent enabling of transitions b and c .

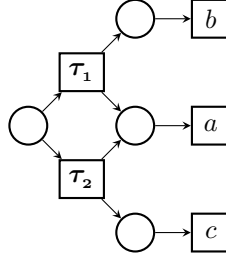


Figure 2.3: A Petri net with two shortest paths via silent transitions to execute a . Executing the wrong silent transition could unintentionally block b or c subsequently.

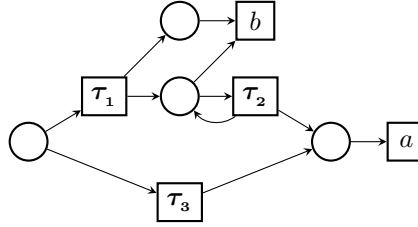


Figure 2.4: A Petri net in which the shortest path to enable a via silent transition τ_3 , leads to a failure to replay for any trace of the form a^*b .

ing the shortest path via silent transitions to the next non-silent activity, was sufficient in all cases to replay event logs on Petri nets generated by the Inductive Miner.

We can show, however that this is not true in general by means of two simple counter examples:

Competing shortest paths The Petri net in Figure 2.3 illustrates a case in which two shortest paths to enabling event a exist, and choosing one or the other will consequently also enable b or c respectively. Here a choice must be made between the shortest paths, potentially resulting in a subsequently blocked activity.

Shortest path is blocking The Petri net in Figure 2.4 illustrates a case in which a longer path along silent transitions is necessary to execute a , followed by either b or a any number of times prior to b . Choosing the shortest path is valid only for the trace consisting solely of one occurrence of a .

We leave a more rigorous investigation into search strategies on the reachability graphs of specific classes of Petri nets containing silent transitions for future work.

2.3 Summaries

Towards an Empirical Evaluation of Imperative and Declarative Process Mining [10] briefly outlines the approach which was later implemented and expanded upon in the full paper [12]. Due to the page limit on this paper, the discussion is a bit a high-level and the formal details are fleshed out more thoroughly in the full version.

Imperative or Declarative? An Empirical Comparison of Event Logs [12] represents the extended version of the preceding outline of our evaluation approach, though with some alterations. After a good deal of discussion and experimentation, we decided to abandon the generalization metric referred to in the section titled *Metric Selection*. This metric essentially attempts to capture the degree of overfitting, but without reference to any out-of-sample data. Instead it uses an estimate of the expectation - over model states - of seeing unanticipated events. Both an event-based and state-based estimator is defined (originally in [52]). For completeness, we briefly describe these.

For event-based generalization, let \mathcal{E} denote set of events and

$$\text{sim}(e) = \{e' \in \mathcal{E} | \text{state}_M(e') = \text{state}_M(e)\}$$

the set of events which share the same model state immediately prior to being executed,

$$\text{diff}(e) = \{\text{act}(e') | e' \in \text{sim}(e)\}$$

the set of unique activities observed leaving model state immediately prior to executing event e . Then the estimated probability of seeing a new activity in the state associated with model M is given by

$$\text{pnew}(|\text{sim}(e)|, |\text{diff}(e)|) = \frac{\text{diff}(e)(\text{diff}(e) + 1)}{\text{sim}(e)(\text{sim}(e) + 1)}$$

. This estimator assumes an unlimited number of possible activities and a multinomial distribution. The generalization metric is then given by

$$1 - \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \text{pnew}(|\text{diff}(e)|, |\text{sim}(e)|)$$

In the state-based variant is as follows (with $\text{state}_M(e)$ replaced by s):

$$1 - \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \text{pnew}(|\text{diff}(s)|, |\text{sim}(s)|)$$

. Though undoubtedly a clever formulation, we decided to employ a more standard, cross-validation based estimate of out-of-sample error [1]. This was to a large part for the sake of interpretability and because it has not been established - and we were not convinced, based on our experiments - that it

actually measures what it intends, and investigating the statistical validity of the metric itself was outside the scope of this paper.

We also introduce a variation on the original precision metric, namely one which is normalized w.r.t. the log using the flower model. That is, it considers the range of possible precision values, from a completely permissive model (precision is not θ in this case), and a perfectly fitting model (precision is 1.0). This metric was formulated in connection with work on [39].

Finally, we formalize our hypotheses and relax the Pareto improvement criteria in [10] by introducing a formal notion of projection onto an ideal vector which is orthogonal to an exact trade-off. This can be biased as needed by weighting one metric over another.

Unfortunately, the paper was not accepted on the first submission attempt. We are currently in the process of reworking and resubmitting based on reviewer feedback.

Chapter 3

Applications

3.1 Discussion

This chapter focuses on two projects undertaken during the PhD concerning process mining applied in practice. The first was a project in collaboration with private industry, specifically Gekkobrain A/S and their client Vestas Wind Systems [8]. The second project was undertaken during my stay abroad at the University of Edinburgh, School of Informatics/Usher Institute [40, 41].

The project with Gekkobrain and Vestas was formally undertaken during a hiatus from the PhD, from February to June 2019. Despite not officially being a part of the PhD project, I have chosen to include a brief discussion of it here since the subject matter and insights are highly relevant. The project concerned the analysis of enormous datasets from Vestas’ global SAP system using process mining techniques. As it turns out, our focus remained mostly confined to the low-level data extraction and preprocessing steps, that would usually be performed prior to the high level analysis of control-flow that is typically the focus of process mining algorithms. Indeed, the very notion of a *trace* or an *event log* seem to be taken for granted by many researchers - and entirely absent from real-world systems, often requiring a large amount of domain expert involvement to extract. Unfortunately, the insights from this project have not yet resulted in a formal publication since it was difficult to identify clear research contributions - most of our time being spent on developing efficient code to handle the enormous datasets. Nonetheless, I present here a draft summary of the challenges and insights along with an attempt at identifying areas that may be worthy of further research.

The second project, undertaken during my stay in Edinburgh, was started in June 2019 and facilitated by my contact there, Areti Manataki, a Lead Researcher affiliated with both the School of Informatics and the Usher Institute which focuses on Molecular, Genetic and Population Health Sciences. Specifically, Areti facilitated a collaboration with Professor of Surgery and Data Science (and practicing surgeon) Ewen Harrison. With Prof. Harrison’s assistance

and insight, I was able to access and analyze a dataset of patient flows from the Royal Infirmary of Edinburgh spanning the years 2010-2018. This resulted in two publications: a full conference paper which I presented at the HealthInf conference in Malta and subsequent journal extension which is awaiting publication. Finally, during my stay, Areti introduced me to researchers in the Automated Reasoning group in the School of Informatics (namely Prof. Jacques Fleuriot and Dr. Petros Papapanagiotou). I had the opportunity to give a presentation regarding my work and process mining to the Automated Reasoning Reading Group and received valuable feedback. It was particularly interesting to learn about Jacques and Petros previous work in workflow modeling using a Linear Logic based formalism and tool called WorkflowFM. A recurring theme in their, and later my own work, was the challenge of bridging the gap between theory and tools on one hand and real-world adoption on the other.

3.2 Summaries

Gekko Brain A/S & Vestas Wind Systems [8] is a brief description of the 4-month project with Gekko Brain and Vestas in mining massive SAP transactions logs.

Modelling Operating Theatre Workflows: A Brief Literature Review [5] was written in connection with the subsequent work on analyzing data from a surgical ward at the Royal Infirmary of Edinburgh. It was never intended for publication, but rather to gain an overview of gaps in the research - from the process mining community, as well as related field - and help orient our research efforts.

Mining Patient Flow Patterns in Surgical Ward [40] summarizes the results of several months of work with a dataset recording patient flows through the surgical ward at the Royal Infirmary of Edinburgh. The dataset required a great deal of cleaning, and due to the legal restrictions associated with accessing this somewhat sensitive dataset, all analysis was performed on a remote server running an *RStudio* session, requiring some effort in applying standard process mining techniques. Furthermore, familiarization with the semantics of the dataset - e.g. procedure codes, patient condition codes, anesthetic codes, location codes, etc - via data dictionaries, and in terms of regulations and guidelines, was an important part of the process.

We were fortunate to be able to visit the infirmary and talk to staff, even observing a liver transplant, during which a nurse explained demonstrated the data entry process to us. As noted in the paper, I was somewhat taken aback at the unreliability of the data in terms of missing, duplicate or anomalous entries. My response may well have been unwarranted however: my colleagues at the Usher Institute bawled at the fact that *only* 10% of the dataset was invalid. In any case, one take-home lesson here was the importance of the pre-processing

steps of the data science workflow and indeed the utility of process mining tools as a means of anomaly detection in that regard.

After cleaning the data, the resulting control-flow was indeed revealed to be linear, in line with our input from the nurses and surgeons. This led us to investigate other aspects of the process that could be informative. We learned from Prof. Harrison that a main concern in the Infirmary is the under-utilization of resources, and avoiding unnecessary cancellations by better coordinating activities. This is supported by a number of reports on service improvement goals I was able to find. One consistently recurring theme in this regard is timing: procedures which run over time risk leading to cancellations and cascading reschedulings while underruns lead to underutilisation of operating rooms and personnel. This lead us to investigate timing in depth in the paper that follows. It should be noted that we did investigate other aspects such as network analysis handover of work and collaboration between anesthetists and surgeons, but in the end timing was the aspect most clearly amenable to an informative analysis.

Stochastic Workflow Modeling in a Surgical Ward: Towards Simulating and Predicting Patient Flow [41] was an invited submission extending the work in *Mining Patient Flow Patterns in a Surgical Ward*. It presents a much more comprehensive attempt at modeling the patient flow process, and presents some insights that were left out of the first submission due to page length limitations.

This publication demonstrates an interesting view of process mining and modeling, in that the data-oriented aspect is almost entirely contained in the *parametrization* of the model. Furthermore, this work really delves into a type of *hybrid* modeling - combining both imperative, flow oriented aspects, and rule- or constraint-oriented aspects. In some ways this may not resemble what is classically presented as process mining, which so often narrowly focuses on the control-flow viewed from one aspect of the process (a trace ID). Here, we try to capture multiple interacting patient flows, competing for resources subject to guidelines and regulations regarding prioritization, and all of this subject to probability distributions learned from real data. Our models came nowhere near capturing all of the complexities of the real world processes involved - indeed, some rules and regulations of which we were aware had to be left out - nonetheless it can serve as a proof-of-concept of an end-to-end process of modeling and model parametrization from data.

Some interesting points to note, particularly in light of the overall focus the thesis are: the way in which “declarative” aspects are captured in the Petri modeling software. Specifically, I would note that most of the guards can be equivalently - albeit much less legibly - captured using classic Petri net constructs, speaking to my note on the matter in Section 0.3.1. The insights regarding shared resources constraints were in large part what motivated my work on hybrid process mining based on declarative resource constraints in [7].

Acknowledgements I would like to thank Areti Manatakis for hosting me during my stay in Edinburgh and arranging access to the dataset via our contact at the infirmary, Ewen Harrison (practicing surgeon and professor of data science) who was very helpful in providing insights into the dataset and operations in the surgical ward.

Chapter 4

Mining Algorithms & Learning

4.1 Discussion

This section contains work on novel mining algorithms and associated theoretical and empirical results. The final two papers outline what I believe to be a convincing approach to addressing challenges in process mining evaluation: by coupling models with executable semantics with probabilistic models, proper out-of-sample evaluation becomes straightforward despite the absence of labels, and the event log replay challenge [12] - requiring perfect fitness or alignment procedures - becomes irrelevant.

4.2 Summaries

Discovering Responsibilities with Dynamic Condition Response Graphs [39] presents an algorithm for mining DCR Graphs from event logs with bijective event to activity functions. The paper utilizes the evaluation approaches presented in [10, 12] and the implementation described in Section 2.2.

DisCoveR: Accurate & Efficient Discovery of Declarative Process Models [13] extends the preceding paper and was invited for submission in connection with the impressive performance of the algorithm in the 2019 Process Discovery Contest, which frames process discovery as a binary classification task. The earlier insights we gained in framing process discovery in these terms and model selection based on proper out-of-sample error estimation were clearly reflected in that our submitted models attained an accuracy of 96.1% on the unseen test data: the next-best performance in the contest. This extension presents a very fast bit-vector implementation and corresponding complexity and run-time analysis, but also a thorough mathematical formalization of the

algorithm, a formal treatment of process discovery as in terms of computational learning theory, as well as case studies.

This paper represents the current state-of-the art in mining algorithms for the declarative DCR Graphs formalism, next steps in this line of work involves incorporating more sophisticated aspects of that formalism such as time and resource perspectives, as well as sub processes and non-bijective labeling functions.

Weighing the Pros and Cons: Process Discovery with Negative Examples [49] was motivated by our ongoing discussions regarding process mining on labeled event logs. It formalizes the notion of unary and binary mining algorithms, derives several theoretical results, and presents a simple but effective mining algorithm arising naturally from the theoretical underpinnings.

This paper helps lay the groundwork for future work in a PAC analysis, and establishment of generalization bounds, for learning various classes of process models, and specifically establishing their Vapnik-Chernovenkis dimension.

Bottom-up Process Mining via Hidden Markov Models: A feasibility Study [6] is a draft paper originally submitted as an assignment for a course on probabilistic graphical models. It builds on the observation that regardless of high-level formalism (imperative, declarative, other) that the formalism essentially *summarizes* the transition system semantics underlying it, and frames the process mining task as learning a transition system within Hidden Markov Model, only subsequently interpreting this into a high-level model. Employing the probabilistic hidden Markov model framework solves the problem we encountered in [10] and [12] of being unable to replay - and hence compute evaluation metrics - for non-fitting traces, since all traces can be assigned a probability.

This is solved by two components in the HMM model: the emission matrices determining the probability of observed data given model state, and the probabilistic transitions between model states. The former allows us to assign a nonzero probability to seeing any event in any state, essentially capturing the notion of “noise”. The latter allows nonzero probabilities of transitions between states that may not be present in the deterministic model, capturing a degree of uncertainty regarding what state the model is in, such that if one state has an extremely low probability of emitting an observed event, we may assign a higher likelihood to being in another state.

The main thrust behind this paper is that a notation-blind model might be built up at the transition system level that best fits the data. Then the transition system would be translated to an imperative, declarative or hybrid model for interpretability. This would be a natural extension of so called region-based process mining algorithms that work by translating transition systems to Petri nets [53].

Towards Inference of Resource Dependency Grammar from Event Streams

[7] is a draft paper that represents an attempt to draw together many of the insights gained through the course of the project to build a probabilistic, resource/attribute-oriented, hybrid mining algorithm that bridges the formal approach from process mining with a statistical machine learning paradigm. The framing of the problem in terms of inducing sequential orderings on events subject to constraints on other event attributes was originally inspired by my work with the dataset from the Royal Infirmary of Edinburgh, in which it was clear that events could be meaningfully viewed from many perspectives: e.g. a patient, a surgeon, and operating room. Taking such views w.r.t. each attribute respectively results in more or less structured orderings of events, and of course many views overlap. The applicability of this view was bolstered by input from domain experts during the collaboration with industry described in [8], from which it became clear that resource dependencies is an area of strong interest, for example to ensure safe migration of interdependent software components to new systems.

Initially, I addressed the problem of measuring the usefulness of an ordering on events using the entropy measures in [11], and spent quite a bit of time developing a mutual-information based approach to finding “words” in sequences, which seemed quite convincing. I also explored the entropy of random walks on the directly follows graphs induced by an attribute constraint as a method for building up formulas in the attribute grammar. The idea here is essentially akin to that used in building decision trees: if we “split” (induce an ordering) based on this attribute, how much information do we gain?

However, fully integrating these approaches in an end-to-end model inference process became convoluted enough that I opted for a more straightforward, clustering based technique for the present paper. This was also to avoid obscuring the presentation of the overall problem formulation, specific solutions to which can be explored in later work. Any candidate strategy for building a grammar and encoding the subsequent event ordering in a control-flow model is valid.

The integration with factorial hidden Markov models means that the evaluation problem is solved in a natural manner based on posterior likelihoods [6], rather than esoteric precision metrics and sequence alignment techniques [52]. We simply consider the probability of seeing the data given the model, and if we want to turn the model into a classifier or use it for anomaly detection, we simply set a threshold on this value. The probabilistic approach also means we can impute missing data and provide recommendations in a natural manner.

Bibliography

- [1] Yaser S Abu-Mostafa, Malik Magdon-Ismael, and Hsuan-Tien Lin. *Learning from data*, volume 4. AMLBook New York, NY, USA:, 2012.
- [2] Arya Adriansyah, Jorge Muñoz-Gama, Josep Carmona, Boudewijn F van Dongen, and Wil MP van der Aalst. Alignment based precision checking. In *International Conference on Business Process Management*, pages 137–149. Springer, 2012.
- [3] Arya Adriansyah, Boudewijn F van Dongen, and Wil MP van der Aalst. Towards robust conformance checking. In *International Conference on Business Process Management*, pages 122–133. Springer, 2010.
- [4] Adriano Augusto, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, Andrea Marrella, Massimo Mecella, and Allar Soo. Automated discovery of process models from event logs: Review and benchmark. *IEEE transactions on knowledge and data engineering*, 31(4):686–705, 2018.
- [5] Christoffer Olling Back. Modelling operating theatre workflow: A brief literature review. In *PhD Thesis*, pages 107–115, 2019.
- [6] Christoffer Olling Back. Inferring process models via hidden markov models. In *PhD Thesis*, pages 207–215, 2020.
- [7] Christoffer Olling Back. Towards inference of resource dependency grammars from event streams. In *PhD Thesis*, pages 216–235, 2020.
- [8] Christoffer Olling Back. Mining massive SAP transaction logs. In *PhD Thesis*, pages 104–107, 2021.
- [9] Christoffer Olling Back, Søren Debois, and Tijs Slaats. Towards an entropy-based analysis of log variability. In *International Conference on Business Process Management*, pages 53–70. Springer, 2017.
- [10] Christoffer Olling Back, Søren Debois, and Tijs Slaats. Towards an empirical evaluation of imperative and declarative process mining. In *International conference on conceptual modeling*, pages 191–198. Springer, 2018.

- [11] Christoffer Olling Back, Søren Debois, and Tijs Slaats. Entropy as a measure of log variability. *Journal on Data Semantics*, 8(2):129–156, 2019.
- [12] Christoffer Olling Back, Søren Debois, and Tijs Slaats. Imperative or declarative? an empirical comparison of event logs. In *PhD Thesis*, pages 86–103, 2021.
- [13] Christoffer Olling Back, Tijs Slaats, Thomas Troels Hildebrandt, and Morten Marquard. Discover: Accurate & efficient discovery of declarative process models. *arXiv preprint arXiv:2005.10085*, 2020.
- [14] David Basin, Felix Klaedtke, Samuel Müller, and Eugen Zălinescu. Monitoring metric first-order temporal properties. *J. ACM*, 62(2):15:1–15:45, May 2015.
- [15] David Basin, Felix Klaedtke, and Eugen Zălinescu. Algorithms for monitoring real-time properties. *Acta Informatica*, 55(4):309–338, Jun 2018.
- [16] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.
- [17] Andrea Burattin, Fabrizio Maggi, and Alessandro Sperduti. Conformance checking based on multi-perspective declarative process models. *Expert Systems with Applications*, 65, 03 2015.
- [18] Nihan Kesim Cicekli and Ilyas Cicekli. Formalizing the specification and execution of workflows using the event calculus. *Information Sciences*, 176(15):2227–2267, 2006.
- [19] J Dassow, G Mavlankulov, M Othman, S Turaev, MH Selamat, and R Stiebe. Grammars controlled by petri nets. *book: Petri nets: Manufacturing and Computer Science*, pages 337–358, 2012.
- [20] Giuseppe De Giacomo, Riccardo De Masellis, and Marco Montali. Reasoning on ltl on finite traces: Insensitivity to infiniteness. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI’14, pages 1027–1033. AAAI Press, 2014.
- [21] Giuseppe De Giacomo, Fabrizio Maria Maggi, Andrea Marrella, and Sebastian Sardina. Computing trace alignment against declarative process models through planning. In *Twenty-Sixth International Conference on Automated Planning and Scheduling*, 2016.
- [22] Giuseppe De Giacomo, Riccardo Masellis, Marco Grasso, Fabrizio Maggi, and Marco Montali. Monitoring business metaconstraints based on ltl & ldl for finite traces. 09 2014.
- [23] Massimiliano de Leoni, Fabrizio M Maggi, and Wil MP van der Aalst. An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. *Information Systems*, 47:258–277, 2015.

- [24] Massimiliano De Leoni, Fabrizio Maria Maggi, and Wil MP van der Aalst. Aligning event logs and declarative process models for conformance checking. In *International Conference on Business Process Management*, pages 82–97. Springer, 2012.
- [25] Johannes De Smedt, Seppe Broucke, Jochen Weerdt, and Jan Vanthienen. A full r/i-net construct lexicon for declare constraints, 02 2015.
- [26] Johannes De Smedt, Seppe Vanden Broucke, Jochen De Weerdt, and Jan Vanthienen. A full r/i-net construct lexicon for declare constraints. *Available at SSRN 2572869*, 2015.
- [27] Søren Debois, Thomas Hildebrandt, and Tijs Slaats. Towards a foundation for modular run-time adaptable process-aware information systems. *Flexible Process Notations for Cross-organizational Case Management Systems*, page 217, 2015.
- [28] Jie Fu and Ufuk Topcu. Computational methods for stochastic control with metric interval temporal logic specifications. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 7440–7447. IEEE, 2015.
- [29] Stijn Goedertier, David Martens, Bart Baesens, Raf Haesen, and Jan Vanthienen. Process mining as first-order classification learning on logs with negative events. In *International Conference on Business Process Management*, pages 42–53. Springer, 2007.
- [30] Stijn Goedertier, David Martens, Jan Vanthienen, and Bart Baesens. Robust process discovery with artificial negative events. *Journal of Machine Learning Research*, 10(Jun):1305–1340, 2009.
- [31] Serge Haddad, Laura Recalde, and Manuel Silva. On the computational power of timed differentiable petri nets. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 230–244. Springer, 2006.
- [32] Jane Hillston. Process algebras for quantitative analysis. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS’05)*, pages 239–248. IEEE, 2005.
- [33] Ullrich Hustadt, Ana Ozaki, and Clare Dixon. Theorem proving for metric temporal logic over the naturals. In *International Conference on Automated Deduction*, pages 326–343. Springer, 2017.
- [34] Andrés Jiménez Ramírez, Irene Barba, Juan Fdez-Olivares, Carmelo Del Valle, and Barbara Weber. Time prediction on multi-perspective declarative business processes. *Knowledge and Information Systems*, 03 2018.
- [35] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

- [36] David JC MacKay and David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.
- [37] Raghava Rao Mukkamala and Thomas Hildebrandt. From dynamic condition response structures to büchi automata. *2012 Sixth International Symposium on Theoretical Aspects of Software Engineering*, 0:187–190, 08 2010.
- [38] Raghava Rao Mukkamala and Thomas T Hildebrandt. From dynamic condition response structures to büchi automata. In *2010 4th IEEE International Symposium on Theoretical Aspects of Software Engineering*, pages 187–190. IEEE, 2010.
- [39] Viktorija Nekrasaite, Andrew Tristan Parli, Christoffer Olling Back, and Tijs Slaats. Discovering responsibilities with dynamic condition response graphs. pages 595–610, 2019.
- [40] Christoffer Olling Back, Areti Manataki, and Ewen Harrison. Mining patient flow patterns in a surgical ward. 2020.
- [41] Christoffer Olling Back, Areti Manataki, Angelos Papanastasiou, and Ewen Harrison. Stochastic workflow modeling in a surgical ward: Towards simulating and predicting patient flows. *Journal Name TBD*, 2021.
- [42] Joël Ouaknine and James Worrell. Some recent results in metric temporal logic. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 1–13. Springer, 2008.
- [43] Petros Papapanagiotou and Jacques Fleuriot. Workflowin: A logic-based framework for formal process specification and composition. In *International Conference on Automated Deduction*, pages 357–370. Springer, 2017.
- [44] Maja Pesic and Wil MP Van der Aalst. A declarative approach for flexible business processes management. In *International conference on business process management*, pages 169–180. Springer, 2006.
- [45] James L Peterson et al. A note on colored petri nets. *Inf. Process. Lett.*, 11(1):40–43, 1980.
- [46] Elham Ramezani Taghiabadi, Dirk Fahland, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Diagnostic information for compliance checking of temporal compliance requirements. In Camille Salinesi, Moira C. Norrie, and Óscar Pastor, editors, *Advanced Information Systems Engineering*, pages 304–320, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- [47] Kaspar Riesen. Structural pattern recognition with graph edit distance. *Advances in computer vision and pattern recognition, Cham*, 2015.
- [48] Anne Rozinat and Wil MP Van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.

- [49] Tijs Slaats, Søren Debois, and Christoffer Olling Back. Weighing the pros & cons: Process discovery with negative examples. In *PhD Thesis*, pages 191–206, 2021.
- [50] Tijs Slaats, Thomas T Hildebrandt, Marco Carbone, and Hagen Völzer. *Flexible process notations for cross-organizational case management systems*. IT University of Copenhagen, Theoretical computer Science section, 2015.
- [51] Niek Tax, Xixi Lu, Natalia Sidorova, Dirk Fahland, and Wil MP van der Aalst. The imprecisions of precision measures in process mining. *Information Processing Letters*, 135:1–8, 2018.
- [52] Wil Van der Aalst, Arya Adriansyah, and Boudewijn van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.
- [53] Wil MP van der Aalst, Vladimir Rubin, Boudewijn F van Dongen, Ekkart Kindler, and Christian W Günther. Process mining: A two-step approach using transition systems and regions. *BPM Center Report BPM-06-30*, *BPMcenter.org*, 6, 2006.
- [54] Moshe Y Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and computation*, 115(1):1–37, 1994.
- [55] AJMM Weijters, Wil MP van Der Aalst, and AK Alves De Medeiros. Process mining with the heuristics miner-algorithm. *Technische Universiteit Eindhoven, Tech. Rep. WP*, 166:1–34, 2006.
- [56] M Westergaard and Fabrizio Maggi. Looking into the future using timed automata to provide a priori advice about timed declarative process models. 09 2012.
- [57] DA Zaitsev and ZW Li. On simulating turing machines with inhibitor petri nets. *IEEEJ Transactions on Electrical and Electronic Engineering*, 13(1):147–156, 2018.

Chapter 5

Manuscripts

Towards an Entropy-Based Analysis of Log Variability*

Christoffer Olling Back¹✉ (0000-0001-7998-7167), Søren Debois² (0000-0002-4385-1409), and Tijds Slaats¹ (0000-0001-6244-6970)

¹ Department of Computer Science, University of Copenhagen
Emil Holms Kanal 6, 2300 Copenhagen S, Denmark
{back, slaats}@di.ku.dk

² Department of Computer Science, IT University of Copenhagen
Rued Langgaards Vej 7, 2300 Copenhagen S, Denmark
{debois}@itu.dk

Abstract. Process mining algorithms can be partitioned by the type of model that they output: *imperative* miners output flow-diagrams showing all possible paths through a process, whereas *declarative* miners output constraints showing the rules governing a process. For processes with great variability, the latter approach tends to provide better results, because using an imperative miner would lead to so-called “spaghetti models” which attempt to show all possible paths and are impossible to read. However, studies have shown that one size does not fit all: many processes contain both structured and unstructured parts and therefore do not fit strictly in one category or the other. This has led to the recent introduction of *hybrid miners*, which aim to combine flow- and constraint-based models to provide the best possible representation of a log. In this paper we focus on a core question underlying the development of hybrid miners: given a log, can we determine *a priori* whether the log is best suited for imperative or declarative mining? We propose using the concept of entropy, commonly used in information theory. We consider different measures for entropy that could be applied and show through experimentation on both synthetic and real-life logs that these entropy measures do indeed give insights into the complexity of the log and can act as an indicator of which mining paradigm should be used.

Keywords: Process Mining · Hybrid Models · Process Variability · Process Flexibility · Information Theory · Entropy · Knowledge Work

1 Introduction

Two opposing lines of thought can be identified in the literature on process modelling notations. The *imperative* paradigm, including notations such as Petri nets [1] and BPMN [2] focuses on describing the flow of a process and is considered to be well-suited to structured processes with little variation. The *declarative paradigm*, including notations such as Declare [3], DCR Graphs [4], and GSM [5] focuses on describing the rules of a process and is considered to be well-suited to unstructured processes with

* This work is supported by the Hybrid Business Process Management Technologies project (DFF-6111-00337) funded by the Danish Council for Independent Research.

large degrees of variation. However, recent studies [6, 7] have shown that one size does not fit all: many processes do not fit strictly in one category or the other and instead contain both structured and unstructured parts. This has led to the recent introduction of a *hybrid* paradigm [6, 8], which aims to combine the strengths of these two approaches.

Following the introduction of the hybrid modelling paradigm, a number of hybrid mining algorithms have been developed: in [9] the authors use a heuristic approach based on the directly-follows-graph to divide activities between structured and unstructured parts of the model; in [10] the authors take a mixed approach and mine both a declarative and imperative model which are then overlain; and in [11] the authors take a model-based approach, where an imperative model is mined and analysed for pockets of unstructured behaviour, and for these pockets, a declarative alternative is mined.

All these approaches avoid an important research question, first identified in [12]: *can we, based on an a priori analysis of the input log, measure if it is best suited to imperative or declarative mining?* Such a measure:

- (i) Would give us greater insights into what type of miner we should use for a log;
- (ii) could be combined with existing partitioning techniques [13–16] to determine for each partition if it is more suited for imperative or declarative mining, thereby providing an efficient method to construct a hybrid model; and
- (iii) could be used for the development of novel partitioning techniques that specifically aim to separate structured and unstructured behaviour in a log.

In this paper we propose basing such a measure on the notion of *entropy* from the field of information theory. Introduced by Shannon in his seminal 1948 paper [17], entropy measures the information content of a random variable. Intuitively, we can think of entropy as the “degree of surprise” we will experience when obtaining additional information about a system [18].

We propose that the entropy of an event log can serve as a predictor of whether the generating process is best modelled using declarative or imperative models. Highly structured processes should generate more homogeneous (low entropy) traces and more flexible processes should generate more varied (high entropy) traces. While information theoretic tools have been previously applied to predictive modelling [19], our application to discriminating mining techniques is novel.

To find such a measure, we first introduce a number of example logs that we use to illustrate our ideas and concepts in Section 2. In Section 3 we introduce three entropy measures on event logs: (i) trace entropy measures only the entropy on the level of distinct traces, (ii) prefix entropy measures entropy by taking into account all unique prefixes of the log, and (iii) block entropy measures entropy by considering all unique sub-strings present in the log. In Section 4 we report on an implementation of these measures and the results of applying them to both synthetic and real-life logs. We show that block entropy is the most successful measure, but suffers from a high computational complexity which becomes apparent on large logs with long traces. In addition it becomes clear that the current proposed measures are not yet absolute and that both further research and a more detailed evaluation are needed to arrive at such a measure. We discuss how we intend to do so in Section 5 and conclude in Section 6.

2 Running Example

We will use a running example of three logs to illustrate how we can use entropy to measure the variability of process logs. Recall the definitions of events, traces and logs.

Definition 2.1 (Events, Traces and Logs). Let Σ be an alphabet of activities. An event $e \in \Sigma$ is a specific occurrence of an activity. A trace $\sigma \in \Sigma^* = \langle e_1, \dots, e_n \rangle$ is a sequence of events e_1, \dots, e_n , with each $e_i \in \Sigma$. Finally, a log is a multiset $[\sigma_1^{w_1}, \dots, \sigma_n^{w_n}]$ where each $\sigma_i \in \Sigma^*$ and each $w_i \in \mathbb{N}$

Notice that we have defined a trace as the sequence of activities observed in a particular process instance. A log is a multiset of such traces, representing explicitly the number of process instances exhibiting the particular trace.

Example 2.2. As a running example, consider the three logs L_1 , L_2 , and L_3 in Figure 1. L_1 is a very structured log, for which we can easily find a compact imperative model, for example the Petri net shown in Figure 2. L_2 is the same log, except some traces are now more frequent than others. The last log L_3 is a much less structured, which is more complex to describe with an imperative model, e.g. the Petri net in Figure 3.

This log can be more effectively explained by a declarative model, as shown in Figure 4. The declarative model uses the Declare notation [3] and shows that: (i) a and b can not occur in the same trace, (ii) after an a we always eventually see an h , (iii) we must have seen at least one a before we can see a c , (iv) we must have seen at least one d before we can see a c , (v) we must have seen at least one d before we can see an e , (vi) after an e we always eventually see an f , (vii) we must have seen at least one f before we can see a g , (viii) after an f we will immediately see a g . One should note that in addition to giving a more straightforward view of the process, this model is also much more precise than the Petri net in Figure 3 (i.e. it allows less behaviour for which there is no evidence in the log).

L_1	L_2	L_3
$\langle a, b, c, d, f, g, h \rangle^5$	$\langle a, b, c, d, f, g, h \rangle^{15}$	$\langle h, a, h, d, c \rangle^5$
$\langle a, b, c, e, f, g, h \rangle^5$	$\langle a, b, c, e, f, g, h \rangle^8$	$\langle a, d, a, c, a, c, e, h, h, f, g \rangle^5$
$\langle a, b, c, d, f, g \rangle^5$	$\langle a, b, c, d, f, g \rangle^5$	$\langle d, e, h, f, g, e, f, g \rangle^5$
$\langle a, b, c, e, f, g \rangle^5$	$\langle a, b, c, e, f, g \rangle^2$	$\langle h, b, b, h, h \rangle^5$
$\langle a, b, b, c, d, f, g, h \rangle^5$	$\langle a, b, b, c, d, f, g, h \rangle^3$	$\langle b, h, d, b, e, h, e, d, f, g \rangle^5$
$\langle a, b, b, c, e, f, g, h \rangle^5$	$\langle a, b, b, c, e, f, g, h \rangle^4$	$\langle a, d, a, d, h \rangle^5$
$\langle a, b, b, c, d, f, g \rangle^5$	$\langle a, b, b, c, d, f, g \rangle^1$	$\langle b, f, g, d \rangle^5$
$\langle a, b, b, c, e, f, g \rangle^5$	$\langle a, b, b, c, e, f, g \rangle^2$	$\langle f, g, h, f, g, h, h, h \rangle^5$

Fig. 1. Example logs. L_1 , L_2 are structured logs, differing only in number of occurrences of complete traces. L_3 is an unstructured log.

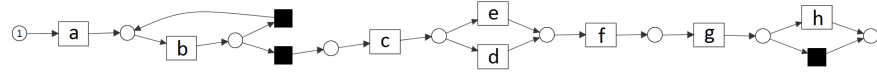


Fig. 2. Petri net for log L_1

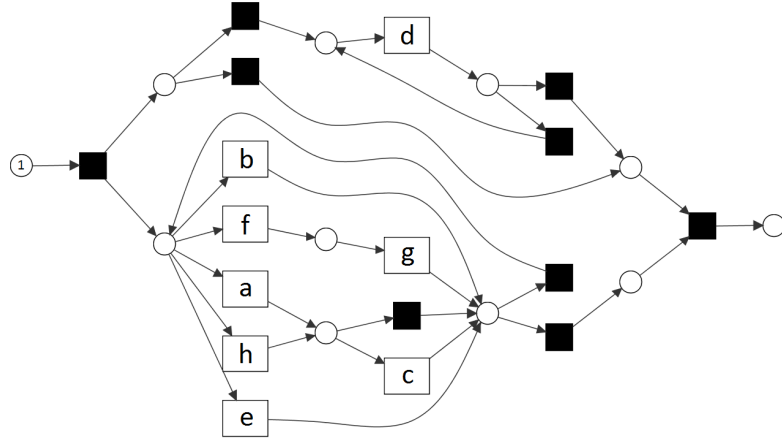


Fig. 3. Petri net for log L_3

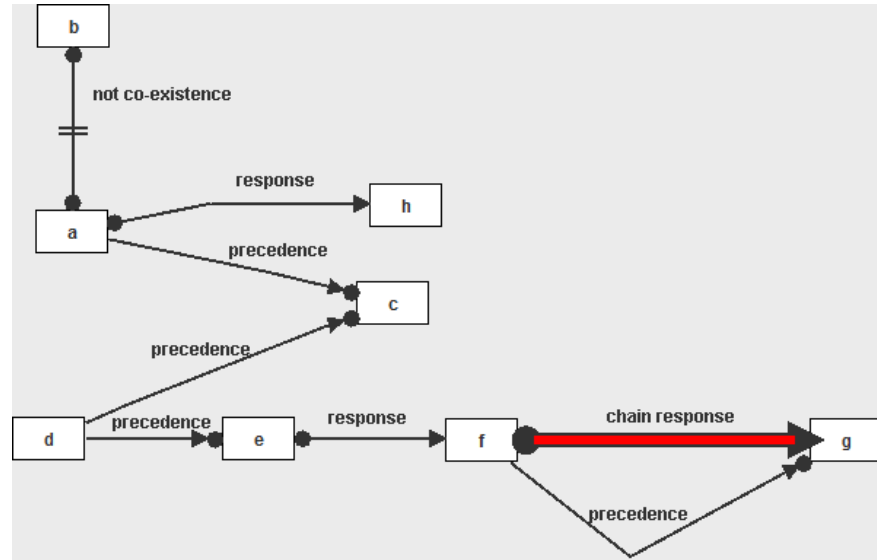


Fig. 4. Declare model for log L_3

3 Log Entropy

Entropy is a measure of the information required to represent an outcome of a stochastic variable, intuitively indicating the “degree of surprise” upon learning a particular outcome [18]. For this paper we focus on Shannon entropy [17], which forms the foundation of the field of information theory.

Given a discrete random variable, X , taking on m possible values with associated probabilities p_1, p_2, \dots, p_m , (Shannon) entropy, denoted H , is given by the expected value of the information content of X :

$$H = - \sum_{i=1}^m p_i \log_b p_i \quad (1)$$

Here b corresponds to the choice of coding scheme (i.e. for binary $b = 2$ and for decimal $b = 10$). We shall use the binary logarithm in the sequel.

Shannon justified this choice of measure with the fact that it is (1) continuous w.r.t. p_i (2) monotonically increasing w.r.t. n under uniform distributions and (3) additive under decomposition of choices, i.e., $H(p_1, p_2, p_3) = H(p_1, (p_2 + p_3)) + (p_2 + p_3)H(p_2, p_3)$.

The key question in using entropy as a measure of log complexity is *what would be the random variable implicit in a given log?*

3.1 Trace Entropy

One very simple answer to this question is to take the underlying random variable as ranging over exactly the traces observed in the log, with probabilities exactly the frequencies observed in the log. This idea gives rise to the notion *trace entropy*.

Definition 3.1 (Trace entropy). Let $L = [\sigma_1^{w_1}, \dots, \sigma_n^{w_n}]$ be a log. The trace entropy $t(L)$ of L is the entropy of the random variable that takes the value σ_i with the following probability.

$$p_i = \frac{w_i}{\sum_{i=1}^m w_i} \quad (2)$$

Example 3.2. Even though the traces of L_1 and L_3 internally have radically different structure, they have the same number of occurrences of distinct traces, and so the same trace entropy:

$$t(L_1) = t(L_3) = -8 \times \frac{5}{40} \log_2 \frac{5}{40} = 3 \quad (3)$$

Computing the trace entropy of L_2 , we find

$$t(L_2) = - \left(\frac{15}{40} \log_2 \frac{15}{40} + \dots + \frac{2}{40} \log_2 \frac{2}{40} \right) = 2.55 \quad (4)$$

This example demonstrates that trace-entropy is likely *not* a good measure for determining if a model should be modelled imperatively or declaratively: L_1 and L_2 intuitively should mine to the same model, but have distinct trace-entropy. On the other hand, L_3 has much more variable behaviour than L_1 , yet has the same trace entropy.

In general, if we are only interested in mining models with perfect fitness [20], then logs that differ only in the number of particular trace occurrences should not mine to different models. We are interested in the number of choices *available* at a particular point in a given trace, not the number of times a particular choice *was made* across all traces. We formalise this observation, using that in this simplistic setting, a “model” is really just a predicate on traces: a *language*.

Definition 3.3 (Language equivalence). *Define logs L, L' to be language equivalent iff they are identical as sets, that is, for each $\sigma^w \in L$, there exists $\sigma^{w'} \in L'$ for some w' , and vice versa.*

Lemma 3.4. *Let P be a predicate on traces; lift it to logs pointwise, ignoring multiplicity. Then if logs L, L' are language equivalent, we have $P(L) \text{ iff } P(L')$.*

Proof. Consider language equivalent logs $L = [\sigma_1^{w_1}, \dots, \sigma_n^{w_n}]$ and $L' = [\sigma_1^{w'_1}, \dots, \sigma_n^{w'_n}]$. By definition $P(L) \text{ iff } \forall i. P(\sigma_i) \text{ iff } P(L')$.

That is, taking the simultaneously abstract and simplistic view that mining a log L is tantamount to coming up with a predicate P such that $P(L)$, the above Lemma says that a mined model can never be used to distinguish language equivalent logs. Because the output model cannot tell the difference between language equivalent logs, it would be unfortunate for our entropy measure to do so.

Definition 3.5. *An entropy measure is a function from logs to the reals. An entropy measure e respects language equivalence iff for any two language equivalent logs L, L' , we have $e(L) = e(L')$.*

Trace entropy is unhelpful, then, because it does not respect language equivalence.

Example 3.6. The logs L_1, L_2 of Example 3.2 are language equivalent. However, they have different trace entropy measures. It follows that trace entropy does not respect language equivalence.

There is on an intuitive level also a second reason that trace entropy is unhelpful: it does not consider the behaviour exhibited *within* the traces. We saw this in Example 3.2, where $t(L_1) = t(L_3)$; that is, trace entropy cannot distinguish internal structure of traces. To consider the full behaviour of a log, we need to determine the entropy on the level of individual events.

3.2 Prefix Entropy

We must find a suitable notion of random variable that “generates” the traces we observe in the log, while at the same time characterises the internal structure of the individual traces.

Recall that a trace is the execution of a single process instance, taking the form of a sequence of events, or activity executions. At each point in a process execution, we will have a prefix of a completed trace. The distribution of these prefixes reflect the structure of the process.

Notation. We write $\langle e_1, \dots, e_n \rangle$ for a finite string. If s, s' are finite strings, we write $s \sqsubseteq s'$ to indicate that s is a prefix of s' .

Definition 3.7 (Prefix entropy). Let L be a log. The prefix entropy of L , written $\epsilon(L)$ is defined as the entropy of the random variable which ranges over all prefixes of traces in L , and for each prefix $\langle e_1, \dots, e_k \rangle \sqsubseteq \sigma$ of a trace σ observed in a log L assigns as its probability the frequency of that prefix among all occurrences of prefixes in L .

Example 3.8. In the log L_2 , the prefix $\langle a, b, c, d \rangle$ occurs in 20 traces; the log contains a total of $15 \times 7 + 8 \times 7 + \dots + 2 \times 7 = 280$ prefix occurrences, for a probability of $1/14$.

However, this notion of prefix entropy *does not* respect language equivalence, since logs differing only in the number of occurrences of a particular trace also differ in the set of occurrences of prefixes. Intuitively, we are interested in prefixes only as a measure of how much internal structure a log has, not how often various bits of that structure occurs. Hence, we disregard multiplicities of traces, in effect *flattening* the log.

Definition 3.9. Let f be the function on logs which given a log L produces the corresponding set, i.e.,

$$f([\sigma_1^{w_1}, \dots, \sigma_n^{w_n}]) = [\sigma_1^1, \dots, \sigma_n^1] = \{\sigma_1, \dots, \sigma_n\} \quad (5)$$

The flattened prefix entropy of L is $\epsilon \circ f(L)$, that is the prefix entropy applied to the flattened log.

Example 3.10. In the log $f(L_2)$, the prefix $\langle a, b, c, d \rangle$ occurs just twice, among a total of only 56 prefix occurrences, for a probability of $1/26$.

We conjecture that transitioning from a log L to a flattened log $f(L)$ does not materially affect prefix entropy; we leave an investigation of exactly which properties are and are not preserved as future work.

Proposition 3.11. The flattened prefix entropy $\epsilon \circ f$ respects language equivalence.

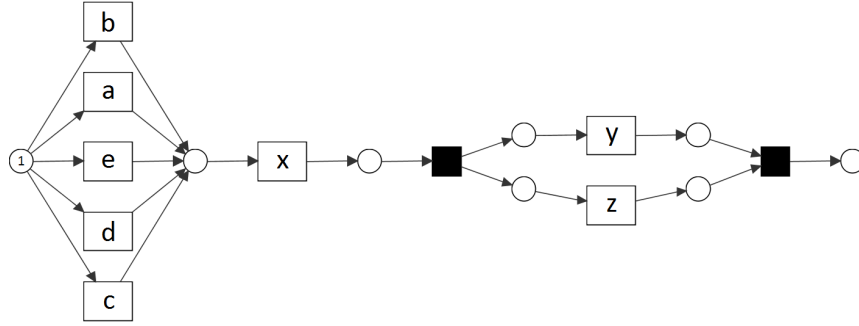
Proof. Immediate by definition of ϵ .

Example 3.12. Computing the flattened event entropy of the example logs of Example 3.2, we find:

$$\begin{aligned} \epsilon \circ f(L_1) &= 4.09 = \epsilon \circ f(L_2) \\ \epsilon \circ f(L_3) &= 5.63 \end{aligned}$$

While the notion of flattened event entropy seems promising, there is one caveat. Because it is based on prefixes, it fails to appreciate common structure appearing distinct prefixes.

Example 3.13. Consider the log L_4 in Figure 5. This log is highly structured: it always contains exactly 4 activities; the first is a choice between $\{a, b, c, d, e\}$, the second an x , the third and fourth either x, y or y, x . See Figure 6 for a Petri net admitting this behaviour. However, this log has a trace entropy of $t(L_4) = 4.82$, higher than the apparently *less* structured logs L_1 and L_2 .

$$\begin{aligned}
&\langle a, x, y, z \rangle^5 \\
&\langle a, x, z, y \rangle^5 \\
&\langle b, x, y, z \rangle^5 \\
&\langle b, x, z, y \rangle^5 \\
&\langle c, x, y, z \rangle^5 \\
&\langle c, x, z, y \rangle^5 \\
&\langle d, x, y, z \rangle^5 \\
&\langle d, x, z, y \rangle^5 \\
&\langle e, x, y, z \rangle^5 \\
&\langle e, x, z, y \rangle^5
\end{aligned}$$
Fig. 5. Log L_4 (highly structured).**Fig. 6.** Petri net for log L_4

3.3 Block Entropy

To address the weaknesses of prefix entropy, we apply ideas from natural language processing [21], where entropy is studied in terms of n -length substrings known as “ n -grams”.

We consider an individual trace a “word”, in which case our log is a multiset of such words, and look at the observed frequencies of arbitrary substrings within the entire log. That is, rather than looking at the frequencies of prefixes, we look at frequencies of substrings.

We shall see that while computationally more expensive, this idea alleviates the problems of prefix entropy and that observed structure is weighted equally, regardless of where it occurs in the log.

Definition 3.14 (*k -block entropy*). Let L be a log. The k -block entropy of L , written $b_k(L)$ is defined as the entropy of the random variable which ranges over all k -length substrings of traces of L , assigning to each such substring s as probability the frequency of the number of occurrences of that substring among all occurrences of k -length substrings.

Example 3.15. In the log L_4 in Figure 5, the 2-block $\langle x, y \rangle$ occurs 5 times; the log contains a total of $10 \times 3 = 30$ occurrences of 2-blocks, for a probability of $1/6$.

Following [21], we compute the k -block entropy $b_k(-)$ directly:

Lemma 3.16. *Let L be a log. The k -block entropy of L is given by*

$$b_k(L) = - \sum_{\langle s_1, \dots, s_k \rangle \in \Sigma^k} p(\langle s_1, \dots, s_k \rangle) \log p(\langle s_1, \dots, s_k \rangle) \quad (6)$$

Often in the literature on estimating the entropy of natural languages, text corpora are used in which all punctuation has been removed, meaning that sentences are ignored and blocks can cover the end of one sentence and beginning of another. For event logs we want to avoid finding spurious correlations among events at the end of one trace and beginning of another trace, so in our approach we keep a clear separation between traces.

We now define block entropy for all substrings up to the length of the longest trace. That is, instead of restricting the measure to blocks of length k , we include all blocks, from length 1 up to the length of the longest trace, in one entropy measure.

Definition 3.17 (All-block entropy). *Let L be a log. The all-block entropy of L , written $b(L)$, is the entropy of the random variable which ranges over all substrings of traces of L , assigning to each such substring s as probability the ratio of occurrences of that substring over all occurrences of substrings.*

Example 3.18. In the log L_3 in Figure 1, the substring (2-block) $\langle a, d \rangle$ occurs 3 times, once in the second entry, twice in the sixth. The log contains a total of 248 occurrences of substrings: $\Sigma_1^9 k = 15$ in the first trace, $\Sigma_1^{11} k = 66$ in the second, and so on. Altogether, the probability of $\langle a, d \rangle$ is $3/248$.

As for the prefix entropy, the all-block entropy does not respect language equivalence, but its flattening does.

Proposition 3.19. *The flattened all-block entropy $b \circ f$ respects language equivalence.*

Example 3.20. We give the flattened all-block entropy for the examples L_1 through L_4 .

	L_1	L_2	L_3	L_4
$b \circ f(-)$	5.75	5.75	7.04	4.75

Notice how L_3 is still the highest-entropy log, but now L_4 is properly recognised as containing information than does L_1 and L_2 .

We conclude this section by noting that while the all-block entropy looks promising, it may be computationally infeasible to apply to large logs. Naively computing the all-block entropy of a log requires, in the worst case, tabulating the frequencies of all substrings seen in that log, an operation that takes polynomial space.

Assume a log has n traces, all of length k . A string of length k contains exactly $k - (i - 1)$ substrings of length i : one starting at each index except for the last $i - 1$ indices, where there is no longer room for a substring of length i .

Thus, by summing over all traces and all substring lengths, we can establish an upper bound on the size of the frequency tables for the substrings of a log:

$$n \times \sum_{i=0}^{k-1} k - i = \mathcal{O}(n \times k^2) \quad (7)$$

So in a concrete case where a log has 20.000 traces of length 100; we would in the worst case need a table of 2×10^{10} substrings. In the next section, we shall in one instance see our naive prototype implementation run out of memory on a somewhat smaller dataset.

4 Implementation and Early Experiments

To test the various measures we implemented a rudimentary ProM [22] plugin with support for computing the trace, prefix and block entropy of a given log. To get an indication of the utility of the entropy measures we applied them to the examples L_1 , L_2 , L_3 , and L_4 of the preceding sections, as well as to a selection of real-life logs. In particular we used the BPI Challenge 2012³, BPI Challenge 2013 (incidents)⁴, hospital⁵, sepsis cases⁶, and road traffic fines⁷ logs.

There is not yet a clear agreement in the literature on which of these logs should be mined imperatively, and which should be mined declaratively. However, it can be observed that the BPI Challenge 2012 log is commonly used as a base-case for declarative mining algorithms and that both the sepsis cases and hospital log result from highly flexible and knowledge-intensive processes within a Dutch hospital. A recent investigation involving the BPI Challenge 2013 (incidents) log seemed to indicate that an imperative approach may be the most successful, but no concrete conclusions were drawn [11].

We ran two sets of experiments: one to contrast the notions of trace, prefix and all-block entropy; and one to investigate more thoroughly the notion of k -block entropy.

4.1 Comparative Measurements

We report measurements of trace, prefix and all-block entropy of the above-mentioned logs in Table 1. The results are promising for the real-life logs we experimented on. In particular, we see that the BPI Challenge 2012 and sepsis cases logs score highly in terms of all-block entropy, whereas the BPI Challenge 2013 log scores somewhat lower,

³ <https://data.4tu.nl/repository/uuid:3926db30-f712-4394-aebc-75976070e91f>

⁴ <https://data.4tu.nl/repository/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07>

⁵ <https://data.4tu.nl/repository/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54>

⁶ <https://data.4tu.nl/repository/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460>

⁷ <https://data.4tu.nl/repository/uuid:270fd440-1057-4fb9-89a9-b699b47990f5>

Log	Event classes	Unique traces	Shortest trace	Longest trace	Entropy		
					Trace	Prefix	All-block
L_1	8	8	6	8	3.00	4.09	5.75
L_2	8	8	6	8	2.55	4.09	5.75
L_3	8	8	4	11	3.00	5.63	7.04
L_4	8	10	4	4	3.32	4.82	4.75
BPI Challenge 2012	36	4366	3	175	7.75	12.53	16.01
BPI Challenge 2013	13	1511	1	123	6.66	11.32	12.21
Sepsis Cases	16	846	3	185	9.34	10.59	14.67
Road Traffic Fines	11	231	2	20	2.48	6.50	8.73
Hospital Log	624	981	1	1814	9.63	DNF	DNF

Table 1. Trace, flattened prefix and flattened all-block entropy measures for select logs.

which fits the intuition that the first two are more suited for declarative mining, whereas the latter is more suited for imperative mining.

We were unable to compute the all-block entropy for the hospital log. This log has traces up to length 1800, and thus requires a large amount of memory to store the intermediary table of substring frequencies.

We conclude that (a) the flattened all-block entropy is the most promising measure for indicating whether a log is best mined imperatively or declaratively; and (b) that a computationally more efficient approximation to the all-block entropy is needed.

Furthermore, for using the metric to inform the choice of imperative or declarative miner, we must determine a cut-off entropy value. But as seen in Table 1, prefix and block entropy grow in proportion to the number of unique traces and event classes. This reflects the second rationale for Shannon’s entropy measure, that it be monotonically increasing. This can potentially be addressed by measuring the *entropy rate*, discussed in Section 5.2.

4.2 Block Entropy Measures

To understand the flattened block entropy measure in more detail, in particular in the hope of finding an efficient approximation of it, we analyse its constituent parts (1-blocks, 2-blocks etc.) in our selection of logs. The results are visualised in Figure 7.

We note that when blocks become longer than the longest trace, k -block entropy falls to zero since we are effectively counting the occurrences of impossible events and $\lim_{n \rightarrow 0} n \log(n) = 0$. This contrasts with a system with one certain outcome in which case we also have $H = 0$ since $1 \log(1) = 0$. We emphasize that this plays no role in our all-block entropy measure since it includes only blocks up to the length of the longest trace.

Note that the entropy of L_3 , a declarative process, is never less than those of the more structured logs, L_1 , L_2 , and L_4 . What is otherwise apparent from this figure is that there is no immediately obvious shortcut to the flattened all-block entropy obtainable as any particular k -block size: no single k seems representative of the full measure. This

lack of representation is further evident from the number of crossings in the diagram: it would appear that from no single k onwards does the entropy of individual logs maintain their relative positioning. E.g., at $k = 10$, the BPI 2013 log measures more complex than 2012; however, they meet and switch places at $k = 18$.

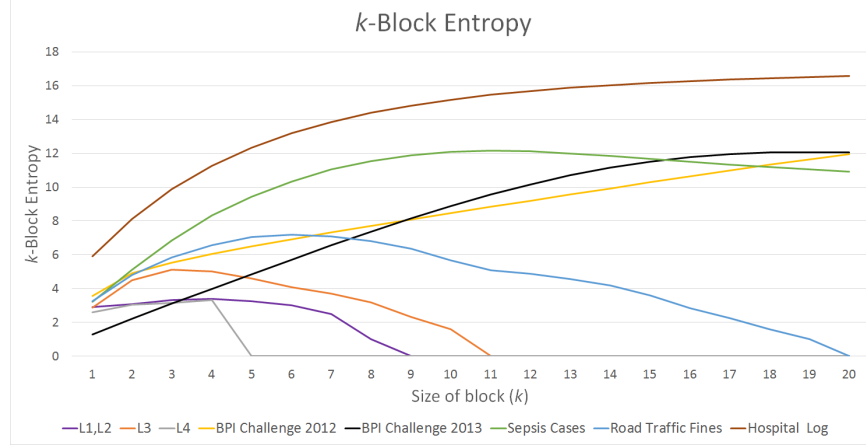


Fig. 7. k -block entropy of flattened logs using different block lengths.

5 Discussion and Future Work

Our experiments show that entropy is a promising indicator of log complexity. However, several questions are left open:

- (i) How can we perform a more thorough evaluation of the suitability of our entropy measures?
- (ii) Next to flattening the log, should we perform any additional normalisations to arrive at a fair measure of entropy?
- (iii) Can we find entropy measures with a reasonable computational complexity so that we can deal with large logs?
- (iv) How can we incorporate our approach with clustering techniques? (Both in an effort to find more efficient entropy estimations and use entropy measures to find suitable clusters for hybrid mining.)

In this section we shortly discuss these open challenges and provide possible avenues of future research to alleviate them.

5.1 More Thorough Experiments

In the previous section we reflected on the types of models we expected to be most suitable for the real-life logs that we experimented on. These were primarily educated

guesses and to do a more thorough evaluation we should perform an analysis of these logs to determine whether they are more suited for imperative or declarative mining.

One way to approach this could be to mine each log with imperative and declarative miners and compare the resulting models according to their precision and simplicity [20]. In addition it would be useful to experiment on a larger set of logs, including both a more comprehensive set of synthetic logs and additional real-life logs.

5.2 Additional Normalisation

Our experiments show that larger logs and more event classes result in a higher entropy measure. It is questionable however whether a larger log by definition always is better suited to declarative modelling, and a measure should not simply proxy for other log attributes. One approach to normalising the entropy measure of different sequences is to use the *entropy rate*, the change in entropy between a block of length k and $k - 1$ with increasing k . That is,

$$h = \lim_{k \rightarrow \infty} H_k - H_{k-1} \quad (8)$$

Computing this directly, however, requires extremely long sequences, as it breaks down when $d^k \approx K$, where d is the size of the alphabet(events), k is the block length and K the length of the sequence [23]. Fortunately, estimators of h exist, some of which are discussed in 5.3 and 5.4.

Possible additional normalisations could be based on the number of unique activities in a log, the number of traces, the average length of traces, or the number of events.

5.3 Complexity of Entropy Measures

In certain cases, such as the hospital log, our proposed measure of flattened block entropy is computationally infeasible, at least for our naive implementation. Fortunately, the problem of efficient entropy estimation is well-studied, most notably in physics and natural language processing.

One very efficient approach is based on building prefix trees of non-overlapping blocks. One example of this is the Ziv-Lempel algorithm, which sequentially parses sequences into unique phrases, composed of their previously seen prefix plus a new symbol. In this way a tree structure is built with each phrase defined by a tuple representing a pointer to its prefix and the new symbol. Borrowing an example from [24], the string ABBABBABBBBABABAA would be parsed as:

A	B	BA	BB	AB	BBA	ABA	BAA
(0,A)	(0,B)	(2,A)	(2,B)	(1,B)	(4,A)	(5,A)	(3,A)

With the integers referring to the dictionary reference of earlier encountered prefixes (0 for root prefixes). It can be shown that the compression ratio of the Ziv-Lempel coding scheme converges to the entropy rate, h , of sequences generated by an ergodic process [24].

In [23], the authors found that on very short sequences, block entropy tended to lead to overestimates on low entropy sequences, while they outperformed Ziv-Lempel on high entropy sequences and suggest a two step process which uses a preliminary quick estimate of entropy to inform the choice of proper estimator.

5.4 Clustering of Logs

In determining whether the declarative or imperative modelling paradigm is most appropriate for a given event log, we may want to look more specifically at the similarity *between* traces rather than *within* traces. In other words, an event log consisting of nearly identical, but complex, traces may nonetheless be best modelled using an imperative approach, while a log of simple, but highly varied traces, may be best described by a declarative model. By clustering traces according to some distance metric, we can get an idea of the diversity of an event log by using the distribution of traces across clusters as the probability distribution for calculating Shannon entropy.

Typically, clustering is performed using Euclidean distance metrics, meaning that data must be represented as a d -dimensional vector. Even techniques such as expectation maximisation clustering, that do not rely directly on computing Euclidean distances, do assume that the data is independent and identically distributed. That is, that observed variables are not directly dependent on each other, so $p(a, b|c) = p(a|c)p(b|c)$. This is an issue for sequential data, a well-known problem in natural language processing, where the “bag-of-words” approach nonetheless leads to impressive results, for example in topic modelling and sentiment analysis [25]. In this approach, word order is simply ignored and documents are represented as multisets (a.k.a. bags) of words, which can then be represented as vectors, with word counts comprising the vector elements.

Similar approaches have been used for trace clustering, by representing traces as vectors of event occurrence counts, ignoring event ordering [14, 26]. For our purposes, this approach is not adequate: For event logs, event ordering cannot be ignored. The reason for this can clearly be seen from the interpretation of entropy as the amount of information gained upon learning the next symbol in a sequence *given* the preceding sequence. For example, in English the letter q is always followed by u , and so $p(u|q) = 1$ and therefore $h = H_n - H_{n-1} = -p(q, u) \log(u|q) = 0$.

To avoid the loss of ordering information which results from collapsing traces to vectors of event counts, we would need to find ways of estimating entropy which allow us to use non-Euclidean distance metrics, for example string edit distances [16, 27, 28]. This allows us to distinguish between traces consisting of the same (or similar) events, but in different orderings.

Another issue with techniques like k -means clustering, is that it is often not clear how to choose the optimal number of clusters, k . Previous research in trace clustering has dealt with this in part by using agglomerative hierarchical clustering techniques, which allow one to “zoom in” and “zoom out” on a hierarchy of clusters to find the optimal partitioning [29].

Correlation clustering is a method for grouping a set of objects into optimal clusters without specifying the number of clusters [30, 31]. Just as important, correlation clustering is a graph-based approach, meaning that data is defined solely by edge weights be-

tween nodes, where edge weight can represent the degree of similarity between nodes. For our current purposes, nodes would represent individual traces and edges the distance measure between them, for example string edit distance.

Another clustering-based estimator that looks promising is Kozachenko and Leonenko's *nearest neighbour entropy estimator* [32], which requires as input only distance measures and some choice of d , the number of dimensions on which the distance metric is defined. This allows us to bypass the actual clustering process, but while it doesn't require the collapsing of traces into vector representations, it does require that we choose a value for d . Using string edit distance, for example, it is not immediately clear what this value should be.

5.5 Noise

Noise is a source of variability, and noisy logs will tend to have a large degree of entropy. The primary challenge is to distinguish between unintentional variability (noise) and intentional variability.

One approach could be to first filter the log for noise using existing techniques, and then measure its entropy afterwards, accepting the risk of accidentally removing interesting behaviour from the log. Alternatively one could assume that the log contains no noise, measure its entropy, mine the log imperatively, declaratively, or hybridly based on the measure, and then analyse the resulting model for unintended flexibility.

6 Conclusion

In this paper we reported on an initial investigation of how entropy can be used as a measure for the complexity of a process log and thereby be used as a basis for determining if a log should be mined and modelled imperatively or declaratively. We investigated three possible entropy measures, each building on the insights gained from the former. We arrived at the notion of block-entropy for process logs and showed through experiments on synthetic and real-life logs that this measure best matches our expectations of log complexity and accordingly which paradigm should be used for mining it. Finally, we proposed 4 distinct paths along which the current work can be extended and we intend to follow-up on these suggestions in future work.

Acknowledgments. We would like to thank both anonymous the reviewers and Jakob Grue Simonsen for valuable and constructive feedback.

References

1. W. M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems and Computers*, 08:21–66, February 1998.
2. Object Management Group. Business Process Modeling Notation Version 2.0. Technical report, Object Management Group Final Adopted Specification, 2011.
3. M. Pesic, H. Schonenberg, and W.M.P. van der Aalst. Declare: Full support for loosely-structured processes. In *EDOC 2007*, pages 287–300, 2007.

4. Søren Debois, Thomas Hildebrandt, and Tijs Slaats. Safety, liveness and run-time refinement for modular process-aware information systems with dynamic sub processes. In *International Symposium on Formal Methods*, pages 143–160. Springer International Publishing, 2015.
5. R. Hull, E. Damaggio, R. De Masellis, F. Fournier, M. Gupta, F.T. Heath, S. Hobson, M.H. Linehan, S. Maradugu, A. Nigam, P. Noi Sukaviriya, and R. Vaculín. Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In *DEBS 2011*, pages 51–62, 2011.
6. H.A. Reijers, T. Slaats, and C. Stahl. Declarative modeling-an academic dream or the future for bpm? In *BPM 2013*, pages 307–322, 2013.
7. Søren Debois, Thomas T. Hildebrandt, Morten Marquard, and Tijs Slaats. Hybrid process technologies in the financial sector. In *BPM '15 (Industry track)*, pages 107–119, 2015.
8. Tijs Slaats, Dennis M. M. Schunselaar, Fabrizio M. Maggi, and Hajo A. Reijers. The semantics of hybrid process models. In *CoopIS*, pages 531–551, 2016.
9. Fabrizio Maria Maggi, Tijs Slaats, and Hajo A. Reijers. The automated discovery of hybrid processes. In *Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings*, pages 392–399, 2014.
10. Johannes De Smedt, Jochen De Weerd, and Jan Vanthienen. Fusion miner: Process discovery for mixed-paradigm models. *Decision Support Systems*, 77:123–136, 2015.
11. Dennis M. M. Schunselaar, Tijs Slaats, Hajo A. Reijers, Fabrizio M. Maggi, and Wil M. P. van der Aalst. Mining hybrid models: A quest for better precision, 2017. Unpublished manuscript. Available on request.
12. S. Debois and T. Slaats. The analysis of a real life declarative process. In *CIDM 2015*, pages 1374–1382, 2015.
13. G. Greco, A. Guzzo, L. Pontieri, and D. Sacca. Discovering expressive process models by clustering log traces. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1010–1027, Aug 2006.
14. Minseok Song, Christian W Günther, and Wil MP Aalst. Trace clustering in process mining. In *Business Process Management Workshops*, pages 109–120. Springer, 2009.
15. Adetokunbo A.O. Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 1255–1264, New York, NY, USA, 2009. ACM.
16. RP Jagadeesh Chandra Bose and Wil MP van der Aalst. Context aware trace clustering: Towards improving process mining results. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 401–412. SIAM, 2009.
17. C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, July 1948.
18. Christopher M.. Bishop. *Pattern recognition and machine learning*. Springer, 2006.
19. Dominic Breuker, Martin Matzner, Patrick Delfmann, and Jörg Becker. Comprehensible predictive models for business processes. *MIS Quarterly*, 40(4), 2016.
20. J.C.A.M. Buijs, B.F. Dongen, and W.M.P. Aalst. On the role of fitness, precision, generalization and simplicity in process discovery. In *On the Move to Meaningful Internet Systems: OTM 2012*, volume 7565, pages 305–322. Springer Berlin Heidelberg, 2012.
21. Thomas Schürmann and Peter Grassberger. Entropy estimation of symbol sequences. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 6(3):414–427, 1996.
22. Wil M. P. van der Aalst, Boudewijn F. van Dongen, Christian W. Günther, Anne Rozinat, Eric Verbeek, and Ton Weijters. ProM: The process mining toolkit. In *BPM (Demos)*, 2009.
23. Annick Lesne, Jean-Luc Blanc, and Laurent Pezard. Entropy estimation of very short symbolic sequences. *Physical Review E*, 79(4):046208, 2009.

24. Joy A Thomas and Thomas M Cover. *Elements of information theory*. John Wiley and Sons, 2006.
25. Thomas Hofmann. Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 289–296. Morgan Kaufmann Publishers Inc., 1999.
26. Gianluigi Greco, Antonella Guzzo, Luigi Pontieri, and Domenico Sacca. Discovering expressive process models by clustering log traces. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1010–1027, 2006.
27. Pavlos Delias, Michael Doumpos, Evangelos Grigoroudis, and Nikolaos Matsatsinis. A non-compensatory approach for trace clustering. *International Transactions in Operational Research*, 2017.
28. Quang-Thuy Ha, Hong-Nhung Bui, and Tri-Thanh Nguyen. A trace clustering solution based on using the distance graph model. In *International Conference on Computational Collective Intelligence*, pages 313–322. Springer, 2016.
29. Minseok Song, H Yang, Seyed Hossein Siadat, and Mykola Pechenizkiy. A comparative study of dimensionality reduction techniques to enhance trace clustering performances. *Expert Systems with Applications*, 40(9):3722–3737, 2013.
30. Erik Demaine and Nicole Immorlica. Correlation clustering with partial information. *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*, pages 71–80, 2003.
31. Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pages 238–247. IEEE, 2002.
32. LF Kozachenko and Nikolai N Leonenko. Sample estimate of the entropy of a random vector. *Problemy Peredachi Informatsii*, 23(2):9–16, 1987.



Entropy as a Measure of Log Variability

Christoffer Olling Back¹ · Søren Debois² · Tijs Slaats¹

Received: 7 February 2018 / Revised: 11 May 2019 / Accepted: 1 June 2019 / Published online: 14 June 2019
 © Springer-Verlag GmbH Germany, part of Springer Nature 2019

Abstract

Process mining algorithms fall in two classes: imperative miners output flow diagrams, showing all possible paths, whereas declarative miners output constraints, showing the rules governing a process. But given a log, how do we know which of the two to apply? Assuming that logs exhibiting a large degree of *variability* are more suited for declarative miners, we can attempt to answer this question by defining a suitable measure of the variability of the log. This paper reports on an exploratory study into the use of *entropy measures* as metrics of variability. We survey notions of entropy used, e.g. in physics; we propose variant notions likely more suitable for the field of process mining; we provide an implementation of every entropy notion discussed; and we report entropy measures for a collection of both synthetic and real-life logs. Finally, based on anecdotal indications of which logs are better suited for declarative/imperative mining, we identify the most promising measures for future studies. For estimating overall entropy, global block and k -nearest neighbour estimators of entropy appear most promising and excel at identifying noise in logs. For estimating entropy *rate* we identify Lempel–Ziv and certain variants of k -block estimators performing well, and note that the former is more stable, but sensitive to noise, while the latter is less stable, being sensitive to cut-off constraints determining block size.

Keywords Process mining · Hybrid models · Process variability · Process flexibility · Information theory · Entropy · Knowledge work

Changes This paper is an extension of earlier work presented at the First Workshop on Business Process Innovation with Artificial Intelligence [1]. The present submission expands substantially on its predecessor by including: (1) a discussion of assumptions regarding ergodicity and stationarity in the business process management (BPM) context (Sect. 3.4). (2) A treatment of edit-distance-based entropy approximation (Sect. 3.6). (3) A discussion of the use of entropy (H) versus entropy rate (h) (Sect. 4). (4) A treatment of the Lempel–Ziv entropy rate approximation (Sect. 4.2). (5) An open-source implementation of these new measures (Sect. 5). (6) Experimental evaluation of both old and the above new measures on a selection of publicly available real-life logs (Sect. 5). (7) Experimental evaluation of all measures on artificial logs to test the effect of different modelling paradigms, concurrency and noise (Sect. 5). (8) A discussion of the experiments leading to a tenuous recommendation of potentially helpful measures (Sect. 6).

This work was supported by the Hybrid Business Process Management Technologies Project (DFF-6111-00337) funded by the Danish Council for Independent Research.

Christoffer Olling Back
 back@di.ku.dk

Søren Debois
 debois@itu.dk

Tijs Slaats
 slaats@di.ku.dk

1 Introduction

Two opposing lines of thought can be identified in the literature on process modelling notations. The *imperative* paradigm, including notations such as Petri nets [2] and BPMN [3], focuses on describing the flow of a process and is considered to be well suited to structured processes with little variation. The *declarative* paradigm, including notations such as Declare [4], DCR Graphs [5], and GSM [6], focuses on describing the rules of a process and is considered to be well suited to unstructured processes with large degrees of variation.

For processes with great variability, declarative miners are often at an advantage: the many possible paths through such a process may be succinctly represented by a small number of constraints, whereas an imperative miner must produce an impossible to read “spaghetti model” explicitly showing all

¹ Department of Computer Science, University of Copenhagen, Universitetsparken 1, 2100 Copenhagen Ø, Denmark

² IT University of Copenhagen, Rued Langgaards Vej 7, 2300 Copenhagen S, Denmark

these many paths. Conversely, for processes with great regularity, imperative miners are often at an advantage: a small number of explicit paths describes the process concisely, whereas a large and obtuse set of declarative constraints is necessary to capture that exact set of paths.

In this paper, we are motivated by the following question, first identified in [7]: *Can we, based on an a priori analysis of the input log, determine whether it is better suited to imperative or declarative mining?*

Such a measure is potentially important for *hybrid mining* [8,9], i.e. process mining where the output model is a combination of declarative and imperative models [10–12]. Here, our proposed measure could be combined with the existing partitioning techniques [13–16] to determine for each partition if it is more suited for imperative or declarative mining. Moreover, it is potentially useful for the development of novel partitioning techniques that specifically aim to separate structured and unstructured behaviours in a log.

We propose basing such a measure on the notion of *entropy* from the field of information theory. Introduced by Claude Shannon in his seminal 1948 paper [17], entropy measures the information content of a random variable. Intuitively, we can think of entropy as the “degree of surprise” we will experience when obtaining additional information about the state of a system [18].

We propose that the entropy of an event log can serve as a predictor of whether the generating process is structured or unstructured and accordingly, whether it is best modelled using declarative or imperative models. Highly structured processes should generate low-entropy logs, whereas more flexible processes should generate high-entropy logs. While information theoretical tools have been previously applied to predictive modelling [19], our application to discriminating mining techniques is novel.

The key contribution of the present paper is to study *exactly how* to measure the entropy of a given log. We study various potential measures based on both entropy and entropy rate, ranging from the near-trivial (trace), over language-inspired ones (prefix, k -block, global block), to methods from the study of dynamic systems and molecular structural analysis (nearest neighbour, Lempel–Ziv, block-based entropy rate). We follow our theoretical study with an empirical study, taking the various measures on both synthetic and real-life logs.

In the absence of an existing classification of available real-life logs into those suitable for declarative, respectively imperative, mining, we are unfortunately unable to objectively determine whether our entropy measures correctly distinguish declarative and imperative logs. However, we can approximate such a classification from the known properties of synthetic logs on the one hand and the common community understanding of select real-life logs on the other, and

as such, qualitatively identify the most promising measures for further study.

Altogether, the present paper contributes (1) a survey of several entropy measures; (2) an implementation of these measures; (3) an experimental evaluation on both synthetic and real-life logs; and (4) based on this evaluation, a selection of promising measures, with a discussion of their strengths and shortcomings.

Overview We first recall basic terminology and introduce four running example logs in Sect. 2. We recall Shannon entropy and study ways to apply it in Sect. 3. In particular, we propose naive measures (trace and prefix entropy) and measures from the literature (block entropy and edit distance measures). We proceed to consider entropy *rate* measures in Sect. 4, studying both block-based estimators and Lempel–Ziv estimators. We report on implementation and experimental results in Sect. 5 and discuss extensively in Sect. 6. Finally, in Sect. 7, we conclude.

2 Process Logs

We recall the standard definitions of events, traces and process logs [20]: an event is an occurrence of an activity in a particular process instance, a trace is a sequence of events of the same such instance, and a log is a multiset of such traces.

Definition 1 (*Events, Traces and Logs*) Let Σ be an alphabet of activities, $s \in \Sigma$.

A trace $\sigma = \langle e_1, \dots, e_n \rangle$ is a finite, non-empty sequence of activities, i.e. a mapping $\sigma : \{1, \dots, n\} \mapsto \Sigma$. An *event*, denoted e_i , is a specific occurrence of an activity in a trace, i.e. $e_i = \sigma(i)$.

We write $\sigma' \sqsubseteq \sigma$ to indicate that σ' is a prefix of σ .

Finally, a *log* is a multiset $[\sigma_1^{w_1}, \dots, \sigma_n^{w_n}]$ with $w_i \in \mathbb{N}$ denoting the multiplicity of trace σ_i .

In the sequel, we will refer extensively to the following running example.

Example 1 In Fig. 1 we present the following three logs:

- L_1 is a very structured log, for which we can easily find a compact imperative model: for example the Petri net shown in Fig. 2.
- L_2 is the same log as L_1 , except some traces are now more frequent than others.
- L_3 is a much less structured log, with many variations in the ways activities can be ordered.

Figure 3 shows a mined Petri net for the log L_3 , whereas Fig. 4 shows a Declare [4] model. The constraints of the Declare model mean that:

Fig. 1 Example logs. L_1 and L_2 are structured logs, differing only in number of occurrences of complete traces. L_3 is an unstructured log

L_1

- $\langle a, b, c, d, f, g, h \rangle^5$
- $\langle a, b, c, e, f, g, h \rangle^5$
- $\langle a, b, c, d, f, g \rangle^5$
- $\langle a, b, c, e, f, g \rangle^5$
- $\langle a, b, b, c, d, f, g, h \rangle^5$
- $\langle a, b, b, c, e, f, g, h \rangle^5$
- $\langle a, b, b, c, d, f, g \rangle^5$
- $\langle a, b, b, c, e, f, g \rangle^5$

L_2

- $\langle a, b, c, d, f, g, h \rangle^{15}$
- $\langle a, b, c, e, f, g, h \rangle^8$
- $\langle a, b, c, d, f, g \rangle^5$
- $\langle a, b, c, e, f, g \rangle^2$
- $\langle a, b, b, c, d, f, g, h \rangle^3$
- $\langle a, b, b, c, e, f, g, h \rangle^4$
- $\langle a, b, b, c, d, f, g \rangle^1$
- $\langle a, b, b, c, e, f, g \rangle^2$

L_3

- $\langle h, a, a, d, c \rangle^5$
- $\langle a, d, a, c, a, c, e, h, h, f, g \rangle^5$
- $\langle d, e, h, f, g, e, f, g \rangle^5$
- $\langle h, b, b, h, h \rangle^5$
- $\langle b, h, d, b, e, h, e, d, f, g \rangle^5$
- $\langle a, d, a, d, h \rangle^5$
- $\langle b, f, g, d \rangle^5$
- $\langle f, g, h, f, g, h, h, h \rangle^5$



Fig. 2 Petri net for log L_1

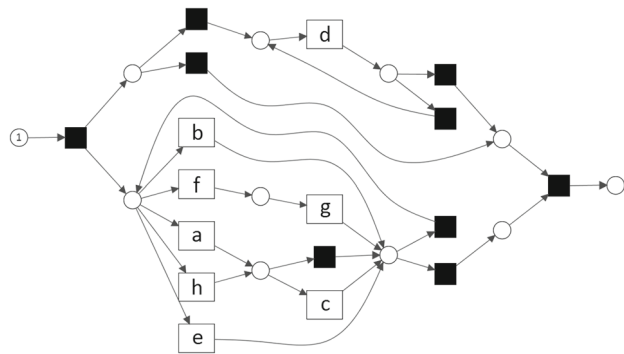


Fig. 3 Petri net for log L_3

1. a and b cannot occur in the same trace,
2. after an a we always eventually see an h ,
3. we must have seen at least one a before we can see a c ,
4. we must have seen at least one d before we can see an e ,
5. we must have seen at least one d before we can see an e ,
6. after an e we always eventually see an f ,
7. we must have seen at least one f before we can see a g ,
8. after an f we will immediately see a g .

Having a closer look at the Petri net will show that the Declare model gives a much more *precise* representation, meaning that it allows less behaviour for which there is no evidence in the log. In particular, in the Petri net the activity d can occur at any point in the process; the precedence relations to c and e are lost. Similarly, the lower branch of the Petri net allows almost any interleaving of activities, with the exception that g should always be preceded by f , and c should always be preceded by a or h . While there exists no conclusive research on measures of relative understandability of these different notations, and this remains a hot topic of debate, it is worth noting that the Petri net employs a significantly larger number of graphical elements than the Declare model.

We will need to disregard the multiplicities of traces, in effect *flattening* the log. This is a common operation in the literature, see, for example, [21].

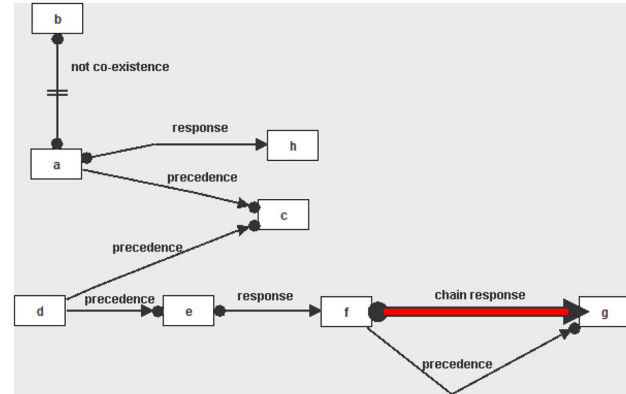


Fig. 4 Declare model for log L_3

Definition 2 Let *flatten* be the function on logs which given a log L produces the corresponding set, i.e.

$$\text{flatten}([\sigma_1^{w_1}, \dots, \sigma_n^{w_n}]) = [\sigma_1^1, \dots, \sigma_n^1] = \{\sigma_1, \dots, \sigma_n\}$$

Example 2 The logs L_1 and L_2 differ only in nonzero multiplicities of traces, so $\text{flatten}(L_1) = \text{flatten}(L_2)$.

Flattening logs is particularly useful when one is interested in finding deterministic models that approximate as accurately as possible the language exhibited by a log. Most state-of-the-art process mining algorithms generate models in deterministic notations (e.g. Declare, Petri nets or BPMN). If the intention is for the model to support all possible behaviour, and not just the most common behaviour, then it makes sense to treat each variation as equal when measuring entropy. In this way we avoid treating logs differently based solely on the statistical distribution of the traces (which will not be represented in the mined models), instead of the inherent entropy of the language that the models should represent.

3 Entropy and Process Logs

Entropy is a measure of the information required, e.g. the average number of bits, to represent an outcome of a stochastic variable, intuitively indicating the “degree of surprise” upon learning a particular outcome [18]. In this paper we

focus on Shannon's formulation of entropy [17], fundamental to the field of information theory.

Definition 3 (*Shannon entropy*) Given a discrete random variable, X , taking on n possible values with associated probabilities p_1, p_2, \dots, p_n , (Shannon) entropy, denoted as H , is given by the expected value of the information content of X :

$$H = H(X) = - \sum_{i=1}^n p_i \log_b p_i \quad (3.1)$$

The base of the logarithm corresponds to the choice of coding scheme (i.e. for binary $b = 2$ and for decimal $b = 10$). We shall use the binary logarithm in the sequel.

Shannon justified this choice of measure with the fact that it is (1) continuous w.r.t. p_i , (2) monotonically increasing w.r.t. n under uniform distributions and (3) additive under decomposition of choices, i.e.

$$H(p_1, p_2, p_3) = H(p_1, (p_2 + p_3)) + (p_2 + p_3)H(p_2, p_3)$$

Entropy can be seen as a measure of the structure or predictability of messages coming from some information source. This has important implications for encoding and compression schemes, since homogeneous (i.e. highly redundant) messages can be significantly compressed by assigning shorter codes to likely outcomes at the expense of using longer codes for very rare outcomes. In fact, Shannon's noiseless coding theorem [22] proves that H is a lower bound on the average number of measurement units (bits) needed for lossless compression.

Estimating the entropy of sequences of symbols, including natural languages, is an active field of research and has implications in areas such as bioinformatics, molecular analysis and chaotic dynamical systems which can be analysed using symbolic dynamics [23]. This problem is highly non-trivial, especially when only smaller samples are available and long-term correlations are present.

We are interested in applying entropy as a measure with which to predict the suitability of a log for imperative or declarative mining; specifically, we expect higher entropy (less structure, less predictive, more bits required for optimal coding) to indicate suitability of declarative models, and lower entropy that of imperative models. In this setting, noise is a source of variability, and noisy logs will tend to have a large degree of entropy. The primary challenge is to distinguish between unintentional variability (noise) and intentional variability. One approach could be to first filter the log for noise using existing techniques and then measure its entropy afterwards, accepting the risk of accidentally removing interesting behaviour from the log. Alternatively, one could assume that the log contains no noise, measure its

entropy, mine the log imperatively, declaratively, or hybridly based on the measure, and then analyse the resulting model for unintended flexibility.

In either case, we will insist that our measure of entropy respects language equivalence.

Definition 4 An *entropy measure for logs* is a function from logs to the reals. An entropy measure e *respects language equivalence* iff for any two language equivalent logs L, L' , we have $e(L) = e(L')$.

We note that any entropy measure can be forced to respect language equivalence simply by flattening the log before feeding it to the measure:

Lemma 1 Let e be an entropy measure. Then the function $e^f(L) = e(\text{flatten}(L)) = e \circ \text{flatten}(L)$ is an entropy measure that respects language equivalence.

Proof Clearly, e^f is a function from logs to reals. Now suppose logs L, L' are language equivalent. Then $\text{flatten}(L) = \text{flatten}(L')$ and it follows that $e^f(\text{flatten}(L)) = e^f(\text{flatten}(L'))$. \square

The key question in using entropy as a measure of log complexity is: *What is the random variable under consideration in the context of an event log?*

3.1 Trace Entropy

One very simple answer to this question is to take the underlying random variable as ranging over entire traces, with probabilities exactly the frequencies observed in the log. This idea gives rise to the notion *trace entropy*.

Definition 5 (*Trace entropy*) Let $L = [\sigma_1^{w_1}, \dots, \sigma_n^{w_n}]$ be a log. The *trace entropy* of L , written $\text{entropy}^{tr}(L)$, is the entropy of the random variable that ranges over the traces in L :

$$\text{entropy}^{tr}(L) = - \sum_{\sigma_i \in L} p_L(\sigma_i) \log p_L(\sigma_i)$$

We distinguish, here and later, between an entropy measure defined on the "true" probability distribution p (as in Definition 5 above) on the one hand, and an estimate \hat{p} of that distribution (Definition 6 below) on the other. We restrict ourselves to simple likelihood (frequency-based) estimators, but more sophisticated estimators exist [23]. For example, Bayesian estimators could incorporate prior probabilities based on domain knowledge.

Definition 6 (*Trace likelihood estimator*) Let L be a log, $L = [\sigma_1^{w_1}, \dots, \sigma_n^{w_n}]$ be a log. The likelihood estimator for trace σ_i , used to compute $\text{entropy}^{tr}(L)$, is given by

$$\hat{p}_L(\sigma_i) = \frac{w_i}{\sum_{i=1}^m w_i}$$

Example 3 Even though the traces of L_1 and L_3 internally have radically different structures, they have the same number of occurrences of distinct traces, and so the same trace entropy:

$$\text{entropy}^{tr}(L_1) = \text{entropy}^{tr}(L_3) = -8 \times \frac{5}{40} \log_2 \frac{5}{40} = 3$$

Computing the trace entropy of L_2 , we find

$$\text{entropy}^{tr}(L_2) = -\left(\frac{15}{40} \log_2 \frac{15}{40} + \dots + \frac{2}{40} \log_2 \frac{2}{40}\right) = 2.55$$

This example demonstrates that trace entropy is likely *not* a good measure for determining if a model should be modelled imperatively or declaratively: L_1 and L_2 intuitively should map to the same model, but have distinct trace entropy. On the other hand, L_3 has much more variable behaviour than L_1 , yet has the same trace entropy.

Example 4 The logs L_1, L_2 of Example 3 are language equivalent. However, they have different trace entropy measures. It follows that trace entropy does not respect language equivalence.

There is on an intuitive level also a second reason that trace entropy is unhelpful: it does not consider the behaviour exhibited *within* the traces. We saw this in Example 3, where $\text{entropy}^{tr}(L_1) = \text{entropy}^{tr}(L_3)$; that is, trace entropy cannot distinguish internal structure of traces. To consider the full behaviour of a log, we need to determine the entropy on the level of individual events.

3.2 Prefix Entropy

We look for a suitable notion of random variable that generates the traces we observe in the log, while at the same time characterising the internal structure of the individual traces.

Recall that a trace is the execution of a single process instance, taking the form of a sequence of activity executions. At each point in a process execution, we have a prefix of a completed trace. Presumably, the distribution of these prefixes reflects the structure of the process.

Definition 7 (*Prefix entropy*) Let L be a log. We write $\text{entropy}^{pr}(L)$ for the *prefix entropy* of L , defined as the entropy of the random variable which ranges over all prefixes of traces in L .

Formally, we write $p(\langle e_1, \dots, e_n \rangle)$ to denote the probability that the outcome of the random variable will be the prefix $\langle e_1, \dots, e_n \rangle$. Following directly from Eq. (3.1), prefix entropy is given by

$$\begin{aligned} \text{entropy}^{pr}(L) \\ = - \sum_{\langle e_1, \dots, e_n \rangle \in \Sigma^*} p_L(\langle e_1, \dots, e_n \rangle) \log p_L(\langle e_1, \dots, e_n \rangle) \end{aligned}$$

For $n \in \mathbb{N}$.

To estimate the probability distribution on prefixes, we again use a simple likelihood estimator. That is, for each prefix $\langle e_1, \dots, e_n \rangle \subseteq \sigma$ of a trace σ observed in a log L , we assign as its probability the frequency of that prefix among all occurrences of prefixes in L .

Definition 8 (*Prefix likelihood estimator*) Let L be a log. The likelihood estimator for prefix $\langle e_1, \dots, e_n \rangle$, used to compute $\text{entropy}^{pr}(L)$, is given by

$$\hat{p}_L(\langle e_1, \dots, e_n \rangle) = \frac{\sum_{\langle e_1, \dots, e_n \rangle \subseteq \sigma \in L} 1}{\sum_{\sigma \in L} |\sigma|} \quad (3.2)$$

where the sum in the denominator gives the total number of prefixes across the log.

Example 5 In the log L_2 , the prefix $\langle a, b, c, d \rangle$ occurs in 20 traces; the log contains a total of $15 \times 7 + 8 \times 7 + \dots + 2 \times 7 = 280$ prefix occurrences, for a probability of $1/14$.

However, this notion of prefix entropy *does not* respect language equivalence: logs differing only in the number of occurrences of a particular trace may also differ in the set of occurrences of prefixes; we therefore define *flattened prefix entropy* as the function composition of first flattening a log and then determining its prefix entropy.

Definition 9 (*Flattened prefix entropy*)

$$\text{entropy}^{fpr} = \text{entropy}^{pr} \circ \text{flatten}$$

Example 6 In the log $\text{flatten}(L_2)$, the prefix $\langle a, b, c, d \rangle$ occurs just twice, among a total of only 56 prefix occurrences, for a probability of $1/28$. Computing the flattened prefix entropy of the example logs of Example 3, we find:

$$\begin{aligned} \text{entropy}^{fpr}(L_1) &\approx \text{entropy}^{fpr}(L_2) \approx 4.09 \\ \text{entropy}^{fpr}(L_3) &\approx 5.63 \end{aligned}$$

While the notion of flattened prefix entropy may seem promising, there is one caveat: because it is based on prefixes, it fails to account for common structure appearing after distinct prefixes.

Example 7 Consider the log L_4 in Fig. 5. This log is highly structured: it always contains exactly 4 activities; the first is a choice between $\{a, b, c, d, e\}$, the second an x , the third and fourth either x, y or y, x . Figure 6 shows a Petri net admitting exactly this behaviour. However, this log has a trace entropy of $\text{entropy}^{tr}(L_4) = 4.82$, higher than the apparently *less* structured logs L_1 and L_2 .

Fig. 5 Log L_4 (highly structured)

L_4

$\langle a, x, y, z \rangle$	5
$\langle a, x, z, y \rangle$	5
$\langle b, x, y, z \rangle$	5
$\langle b, x, z, y \rangle$	5
$\langle c, x, y, z \rangle$	5
$\langle c, x, z, y \rangle$	5
$\langle d, x, y, z \rangle$	5
$\langle d, x, z, y \rangle$	5
$\langle e, x, y, z \rangle$	5
$\langle e, x, z, y \rangle$	5

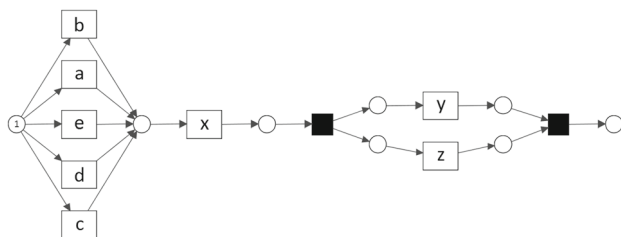


Fig. 6 Petri net for log L_4

3.3 Block Entropy

To address this weakness of prefix entropy, we apply ideas from natural language processing [23], where entropy is studied in terms of n -length substrings known as “ n -grams”. (We will use k instead of n .) We consider an individual trace a “word”, in which case our log is a multiset of such words, and look at the observed frequencies of arbitrary substrings within the entire log. That is, rather than looking at the frequencies of prefixes, we look at frequencies of substrings. We shall see that while computationally more expensive, this idea alleviates the problems of prefix entropy: observed structure is weighted equally, regardless of where it occurs in a trace.

Definition 10 (*k-block entropy*) Let L be a log. We define the *k-block entropy* of L , written $\text{entropy}_k^{bl}(L)$, as the entropy of the random variable which ranges over all k -length strings in Σ^* .

Formally, we write $p(\langle s_1, \dots, s_k \rangle)$ to denote the probability that the outcome of the random variable will be the substring $\langle s_1, \dots, s_k \rangle$ (in Sect. 3.4 it will become clear why we do not write $\langle e_1, \dots, e_k \rangle$).

Again, following directly from Eq. (3.1), block entropy is given by

$$\text{entropy}_k^{bl}(L) = - \sum_{\langle s_1, \dots, s_k \rangle \in \Sigma^*} p_L(\langle s_1, \dots, s_k \rangle) \log p_L(\langle s_1, \dots, s_k \rangle)$$

For fixed k .

The k -block entropy measures the amount of information contained in a block of length k . So, if some activities always

occur in the same order, little new information is added when they are encountered in that order.

Definition 11 (*k-block likelihood estimator*) Let L be a log. The likelihood estimator for blocks of length k , used to compute entropy_k^{bl} , is given by

$$\hat{p}_L(\langle s_1, \dots, s_k \rangle) = \begin{cases} 0 & \text{if } f(L, k) = 0 \\ \frac{\sum_{\sigma \in L} n_{s_1, \dots, s_k}}{f(L, k)} & \text{otherwise} \end{cases} \quad (3.3)$$

where

$$f(L, k) = \sum_{\sigma \in L} \max(0, |\sigma| - k + 1)$$

where n_{s_1, \dots, s_k} is the number of times block s_1, \dots, s_k occurred in a trace, and $f(L, k)$ gives the total number of k -blocks across the log. The zero condition accounts for all k -blocks longer than the longest trace.

Definition 12 (*Flattened k-block entropy*)

$$\text{entropy}_k^{fbl} = \text{entropy}_k^{bl} \circ \text{flatten}$$

Example 8 In the flattened version of log L_4 in Fig. 5, $\text{flatten}(L_4)$, the 2-blocks $\langle a, x \rangle$, $\langle b, x \rangle$, $\langle c, x \rangle$, $\langle d, x \rangle$ and $\langle e, x \rangle$ all occur 2 times, while $\langle x, y \rangle$, $\langle y, x \rangle$, $\langle y, z \rangle$ and $\langle z, y \rangle$ all occur 5 times. The log contains a total of $10 \times 3 = 30$ occurrences of 2-blocks, giving the 2-block entropy for $\text{flatten}(L_4)$:

$$\text{entropy}_k^{fbl}(L_4) = - \left(5 \times \frac{2}{30} \log \frac{2}{30} + 4 \times \frac{5}{30} \log \frac{4}{30} \right) \approx 3.03$$

We now define block entropy for all substrings up to the length of the longest trace. That is, instead of restricting the measure to blocks of length k , we include *all* blocks, from length 1 up to the length of the longest trace, in one entropy measure.

Definition 13 (*Global block entropy*) Let L be a log. The *global block entropy* of L , written $\text{entropy}^{bl}(L)$, is the entropy of the random variable which ranges over all strings in Σ^* .

Again, following directly from Eq. (3.1), global block entropy is given by

$$\text{entropy}^{bl}(L) = - \sum_{\langle s_1, \dots, s_k \rangle \in \Sigma^*} p_L(\langle s_1, \dots, s_k \rangle) \log p_L(\langle s_1, \dots, s_k \rangle)$$

For $k \in \mathbb{N}$.

Definition 14 (*Global block likelihood estimator*) Let L be a log. The likelihood estimator for blocks of length k , used to compute entropy^{bl}(L), is given by

$$\hat{p}_L(\langle s_1, \dots, s_k \rangle) = \frac{\sum_{\sigma \in L} n_{s_1, \dots, s_k}}{\sum_{\sigma \in L} \frac{|\sigma|(|\sigma| + 1)}{2}} \quad (3.4)$$

where n_{s_1, \dots, s_k} is the number times block $\langle s_1, \dots, s_k \rangle$ occurred in a trace, and the sum in the denominator gives the total number of k -blocks across the log for k ranging from 1 to the length of the longest trace.

Example 9 A trace of length n contributes $1 + 2 + \dots + (n - 1) + n = \Sigma_1^n k = \frac{n(n+1)}{2}$ distinct occurrences of substrings: one of length n ; two of length $n - 1$ starting at indexes 0 and 1, respectively; three of length $n - 2$; and so forth. Summing up the number of occurrences in each trace in the flattened log, we thus get $\Sigma_1^5 k = \frac{5 \cdot 6}{2} = 15$ in the first trace, $\Sigma_1^{11} k = \frac{11 \cdot 12}{2} = 66$ in the second, and so on, for a total of 248 distinct occurrences.

Counting the number of occurrences of the specific substring (2-block) $\langle a, d \rangle$, we find that it occurs 3 times: once in the second entry, twice in the sixth. Altogether, the probability of $\langle a, d \rangle$ is $3/248 \sim 0.012$.

As for the prefix and k -block entropy, the global block entropy does not respect language equivalence, but its flattening does.

Definition 15 The *flattened global block entropy* entropy^{fbl} is given by entropy^{fbl} = entropy^{bl} \circ flatten.

Example 10 We give the flattened global block entropy for the examples L_1 through L_4 .

	L_1	L_2	L_3	L_4
entropy ^{fbl} (—)	5.75	5.75	7.04	4.75

Notice how L_3 is still the highest-entropy log, but now L_4 is properly recognised as containing less information than L_1 and L_2 .

We conclude this section by noting that while the global block entropy looks promising, it is computationally challenging to apply to large logs. Naively computing the global block entropy of a log requires, in the worst case, tabulating the frequencies of all substrings seen in that log. For a log with n traces, all of length k , the running time is bounded by $\mathcal{O}(n \times k^2)$. This is ameliorated to some degree by using efficient data structures such as a suffix trie for counting k -blocks.

3.4 Stationarity and Ergodicity

In our original definition of entropy^{bl}, we made a notational distinction between the original sequences of events (traces) $\langle e_1, \dots, e_n \rangle$ and subsequences of activities within those traces $\langle s_1, \dots, s_n \rangle$, where e denotes a specific event at a specific position in the trace. This distinction stems from an assumption which underlies most of the entropy estimators we consider: that the underlying process generating traces is *stationary* and *ergodic*.

This notion is clearly illustrated in the case of entropy^{bl} _{k} where, without making these assumptions, we would denote the probability of seeing a specific sequence of activities $\langle s_1, \dots, s_k \rangle$ from index t to index $t + k$ in a trace:

$$p_L(\langle e_{t+1} = s_1, \dots, e_{t+k} = s_k \rangle)$$

That is, the probability that the *specific* event e_{t+1} was an instance of activity s_1 , and so on. In order to drop the index t and use a “sliding window” approach, counting any occurrences of $\langle s_1, \dots, s_k \rangle$ as equivalent regardless of position, we must assume that the position of a k -block in the trace does not influence which k -block is observed and the dynamics of the underlying process do not change over time, i.e. they are *stationary*. Otherwise, we would have to consider an occurrence of the same sequence of activities in the beginning and end of a trace as instances of two different outcomes.

More generally stated, *stationarity* assumes that the probability distribution underlying observed outcomes is *not a function of time*. In the current context, outcomes are the observed k -blocks, or individual events in traces for other metrics we will introduce. This is most likely an incorrect assumption for business processes since some events are often associated with the beginning or end of a process, or that some activity always occurs as, for example, the third activity if it occurs. We nonetheless make this assumption under the presumption that much of the structure in activity sequences will still be captured by k -blocks in a sliding window.

Ergodicity implies that this probability distribution can be reconstructed from the observation of a typical sequence (trace), i.e. that the average properties of the process over time (within a trace) is equivalent to the average properties across space (across the traces in a log) which would seem to run contrary to the fact that process outcomes will be determined by the needs of a specific case (e.g. a customer or patient), unless this variation is to be interpreted as statistical *noise*. It is not clear to what degree violation of these assumptions would actually skew the resulting entropy estimation.¹ It may

¹ See Sect. 4 for the theoretical basis of these assumptions, and an example in which it is impossible to define the entropy of a non-stationary process.

be the case that the proposed estimators nonetheless give reasonable indications of the degree of structure in event logs.

3.5 Block Entropy for Event Logs Versus Natural Language

Beginning with Shannon's original paper [17], research on estimating the entropy of natural languages has demonstrated that studies with human subjects² consistently result in a lower-entropy estimate than mathematical estimators, suggesting that structural information and long-term correlations are not fully captured [24].

NLP researchers often use text corpora modified by removing punctuation and capitalisation, meaning that sentences are ignored and a single block can span the end of one sentence and the beginning of another. For event logs we want to avoid spurious correlations among events at the end of one trace and the beginning of another, so we keep traces separate.

3.6 Nearest Neighbour Estimators

The measures discussed so far fail to capture aspects of the structuredness of the log: *prefix* and *block entropy* will become slightly skewed in the case where traces differ by just one "missing" activity since they consider any blocks overlapping these activities to be unique, and *trace entropy* cannot guarantee language equivalence for flattened logs since it relies on the distribution of unique traces in the log. Using *edit distance* we can obtain a measure which allows us to compare entire traces while also capturing the internal structure of traces, being tolerant to minor differences.

The distribution of traces in the metric space defined by the edit distance determines the entropy of the log: evenly distributed traces yield a high entropy, whereas similar traces grouped together in a few clusters yield a low entropy. In contrast to the trace entropy of Definition 5, we can meaningfully apply this approach to a flattened log.

In previous applications of trace clustering in a BPM context, authors used a *bag-of-activities* approach, redefining traces as vectors of activity counts: each element representing the occurrences of each activity [14,25,26], and using different vector-based distance metrics such as Minkowski distance and Jaccard distance. Researchers in natural language processing commonly use this approach as well [27]. Others propose clustering traces using more structured approaches based on outranking relations theory [28] and distance-graph-based representation [29].

² Participants are given prefixes of text and asked to predict the next letter. The entropy rate (Sect. 4) is calculated from the proportion of correct responses.

Edit-distance-based clustering was investigated in [16] where the authors propose an algorithm for automatically learning operation costs and use agglomerative hierarchical clustering.

Our approach differs from previous work in two ways. First, we use nearest neighbour distances to investigate entropy, rather than clustering similar traces in order to mine these clusters separately; second, the aim of our approach is to use entropy to choose between mining *paradigms*: imperative versus declarative.

3.6.1 Edit Distance Between Traces

Levenshtein edit distance is the most well-known string edit distance metric and is defined by the number of insertions, deletions and substitutions required to convert one string into another. Levenshtein distance $\text{lev}_{x,y}(|x|, |y|)$ between strings x and y , with non-negative operation costs, fulfils the axioms of a *metric*:

non-negativity $\text{lev}_{x,y}(|x|, |y|) \geq 0$

identity of indiscernibles a string can be transformed into itself with 0 operations. $\text{lev}_{x,y}(|x|, |y|) = 0 \implies x = y$

symmetry the same number of operations is required to convert string x to y as to convert y to x . $\text{lev}_{x,y}(|x|, |y|) = \text{lev}_{y,x}(|y|, |x|)$:

triangle inequality $\text{lev}_{x,z}(|x|, |z|) \leq \text{lev}_{x,y}(|x|, |y|) + \text{lev}_{y,z}(|y|, |z|)$

Edit distance allows us to define a metric space on our log in which the internal structure of the trace, i.e. the ordering of events, is captured. We normalise edit distance by the greatest possible distance between the traces to reflect that a distance of one operation on two very long strings should be considered less significant than on very short strings.

Definition 16 (*Normalised Levenshtein distance*) Let σ and ς be traces from the same event log. The Levenshtein distance between these two prefixes is given by

$$\begin{aligned} \text{lev}_{\sigma,\varsigma}(i, j) \\ = \begin{cases} \max(i, j) & \text{if } \min(i, j) = 0 \\ f(\sigma, \varsigma) & \text{otherwise} \end{cases} \end{aligned}$$

where

$$f(\sigma, \varsigma) = \begin{cases} \min \text{lev}_{\sigma,\varsigma}(i-1, j) + 1 \\ \text{lev}_{\sigma,\varsigma}(i, j-1) + 1 \\ \text{lev}_{\sigma,\varsigma}(i-1, j-1) + 1_{\sigma_i \neq \varsigma_j} \end{cases}$$

where $1_{\sigma_i \neq \varsigma_j}$ is the indicator function and returns 0 when $\sigma(i) = \varsigma(j)$ and 1 otherwise. That is, in the "otherwise"

clause, three types of edits are allowed: insertion, deletion and substitution, which in this case have the same cost, 1. The *normalised Levenshtein distance* is given by

$$\text{lev}_N(\sigma, \varsigma) = \frac{\text{lev}_{\sigma, \varsigma}(|\sigma|, |\varsigma|)}{\max(|\sigma|, |\varsigma|)}$$

where $\max(|\sigma|, |\varsigma|)$ is an upper bound on $\text{lev}_{\sigma, \varsigma}(|\sigma|, |\varsigma|)$, guaranteeing a range of $[0, 1]$ for lev_N . It is straightforward to verify that the normalised Levenshtein distance is in fact a metric.

Example 11 Consider two traces from $\log L_1$ in Fig. 5:

$\langle a, b, c, e, f, g, h \rangle$

$\langle a, b, c, d, f, g \rangle$

Converting the first to the second requires two edits: substituting d for e and deleting h . The worst-case scenario is the length of the longest trace, 7, giving a normalised distance of

$$\text{lev}_N(\langle a, b, c, e, f, g, h \rangle, \langle a, b, c, d, f, g \rangle) = \frac{2}{7} \approx 0.29$$

One approach to computing entropy based on edit distances would be to first cluster traces using a clustering algorithm, and interpreting these clusters as the outcomes of the random variable X , then computing Shannon entropy using the number of traces per cluster as the probability distribution over X . Aside from adding an extra clustering step, this approach adds the challenge of choosing an appropriate clustering algorithm and how many clusters into which to partition the log.

Fortunately, previous research provides nearest neighbour-based entropy estimators which estimate entropy *directly* from distances, removing the clustering step.

3.6.2 Distance-Based Estimators

Nearest neighbour-based entropy estimators are a class of nonparametric estimators widely used in machine learning for goodness-of-fit testing, parameter estimation and even for analysing molecular structure [30,31], which allow for estimating entropy directly from distances between data points. The nearest neighbour entropy estimator proposed by Kozachenko and Leonenko in 1987 [32] considers only the first nearest neighbour. This measure of entropy is originally formulated on a vector space and uses the dimension d of that space as a parameter. As we are working merely in a metric space, that d becomes simply a parameter of the measure.

Note that in the following definition ρ (“rho”, not p) signifies *distance* and not probability.

Definition 17 (*Kozachenko–Leonenko entropy*) Let L be a log, let ρ_σ denote the *distance* of trace σ to its nearest neighbour in L , and let d be positive integer. The Kozachenko–Leonenko entropy measure is given by

$$\text{entropy}^{KL}(L) = \frac{d}{|L|} \sum_{\sigma \in L} \log \rho_\sigma + \log \frac{\pi^{d/2}}{\Gamma(\frac{d}{2} + 1)} + \gamma + \log(|L| - 1) \quad (3.5)$$

where $\Gamma(n)$ is the generalisation of $(n - 1)!$ to real (and complex) numbers, $\gamma \approx 0.5772 \dots$ is Euler’s constant, and d denotes the dimensionality of the underlying metric space.

Definition 18 The *flattened Kozachenko–Leonenko entropy* is given by $\text{entropy}^{fKL} = \text{entropy}^{KL} \circ \text{flatten}(L)$.

Example 12 Consider the traces from logs L_1 and L_2 in Fig. 1. The nearest neighbour calculation for each trace in the flattened logs is shown in Table 1. Using these values and letting $d = 1$, we have

$$\begin{aligned} \text{entropy}^{fKL}(L_1) &= \frac{6}{8} \left(\log \frac{1}{8} + \log \frac{\pi^{1/2}}{\Gamma(\frac{3}{2})} + \gamma + \log(7) \right) \\ &\quad + \frac{2}{8} \left(\log \frac{1}{7} + \log \frac{\pi^{1/2}}{\Gamma(\frac{3}{2})} + \gamma + \log(6) \right) \\ &\approx 1.17 \end{aligned}$$

The nearest neighbour estimator was expanded upon by Singh et al. in [31] to the k th nearest neighbour.

Definition 19 (*kth nearest neighbour entropy*) Let L be a log. Let $\rho_{\sigma,k}$ denote the distance of trace σ to its k th nearest neighbour in L . The k th nearest neighbour entropy measure is given by

$$\text{entropy}^{kNN}(L) = \frac{d}{|L|} \sum_{\sigma \in L} \log \rho_{\sigma,k} + \log \frac{\pi^{d/2}}{\Gamma(\frac{d}{2} + 1)} + \gamma - f(k - 1) + \log(|L|)$$

where

$$f(x) = \begin{cases} 0 & \text{if } x = 0 \\ \sum_{y=1}^x \frac{1}{y} & \text{if } x \geq 1 \end{cases}$$

Definition 20 (*Flattened kth nearest neighbour entropy*)

$$\text{entropy}^{fkNN} = \text{entropy}^{kNN} \circ \text{flatten}(L)$$

Note that we used the *normalised Levenshtein distance* to define ρ_σ in entropy^{KL} and entropy^{kNN} , but any distance metric fulfilling the axioms in 3.6.1 can be used.

Table 1 Nearest neighbour calculations for $\text{flatten}(L_1)$ and $\text{flatten}(L_2)$. It happens to be the case for these logs that every trace in these logs has a nearest neighbour with an unnormalised edit distance of 1

Trace (σ)	Nearest neighbour (ς)	$\text{lev}(\sigma , \varsigma)$	$\max(\sigma , \varsigma)$	$\text{lev}_N(\sigma, \varsigma)$
$\langle \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{f}, \mathbf{g}, \mathbf{h} \rangle$	$\langle a, b, b, c, d, f, g, h \rangle$	1	8	$1/8 = 0.125$
$\langle \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{e}, \mathbf{f}, \mathbf{g}, \mathbf{h} \rangle$	$\langle a, b, b, c, e, f, g, h \rangle$	1	8	$1/8 = 0.125$
$\langle \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{f}, \mathbf{g} \rangle$	$\langle a, b, c, d, f, g, h \rangle$	1	7	$1/7 \approx 0.143$
$\langle \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{e}, \mathbf{f}, \mathbf{g} \rangle$	$\langle a, b, c, e, f, g, h \rangle$	1	7	$1/7 \approx 0.143$
$\langle \mathbf{a}, \mathbf{b}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{f}, \mathbf{g}, \mathbf{h} \rangle$	$\langle a, b, c, d, f, g, h \rangle$	1	8	$1/8 = 0.125$
$\langle \mathbf{a}, \mathbf{b}, \mathbf{b}, \mathbf{c}, \mathbf{e}, \mathbf{f}, \mathbf{g}, \mathbf{h} \rangle$	$\langle a, b, c, e, f, g, h \rangle$	1	8	$1/8 = 0.125$
$\langle \mathbf{a}, \mathbf{b}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{f}, \mathbf{g} \rangle$	$\langle a, b, b, c, d, f, g, h \rangle$	1	8	$1/8 = 0.125$
$\langle \mathbf{a}, \mathbf{b}, \mathbf{b}, \mathbf{c}, \mathbf{e}, \mathbf{f}, \mathbf{g} \rangle$	$\langle a, b, b, c, e, f, g, h \rangle$	1	8	$1/8 = 0.125$

While edit distance allows us to capture local structural differences in traces, allowing us to bin traces together which differ by only a few events, it may fail to capture important similarities among traces. For example, the traces $\langle a, b \rangle$ and $\langle a, b, a, b, a, b \rangle$ will be heavily penalised for being of different lengths, even though the latter is clearly a repetition of the former. In the next section we discuss measures like Lempel–Ziv which are able to capture this type of structure.

4 Entropy Rate of Stochastic Processes

Up to this point we have explored ways to interpret process logs as the outcome of some stochastic variable X and finding the entropy of this variable. We shall see in Sect. 5 that logs with longer traces and more activities tend to result in higher-entropy measures. It is questionable, however, whether a larger log by definition is always better suited to declarative modelling.

A more nuanced interpretation of entropy is to consider each *individual event*, rather than the trace as a whole, as the outcome of a separate variable, generated by a stochastic process. This approach is invariant to the number of activities and trace lengths and should better capture what we are interested in: the degree of structure in the underlying process. In the literature, this is referred to as the *entropy rate* of a *process*: the increase in entropy upon considering an additional outcome of the process [33].

We present three estimators of entropy rate in Sects. 4.1 and 4.2, but first we will formally define the entropy rate for a process for which the probability distribution is known. We cover two alternative definitions of entropy rate, which turn out to be equivalent in the limit for stationary processes. First, we will need the definitions of *joint entropy* and *conditional entropy*. See [34] or [33] for details.

Lemma 2 (Joint entropy) *Let e_1, \dots, e_n be discrete random variables. The joint entropy is the Shannon entropy extended to more than one variable. In the present context, the outcomes under consideration are activities $s \in \Sigma$:*

$$H(e_1, \dots, e_n) = - \sum_{s_1 \in \Sigma} \dots \sum_{s_n \in \Sigma} p_n \log p_n$$

where

$$p_n = p(e_1 = s_1, \dots, e_n = s_n)$$

It is the expected value of the information content of the variables:

$$H(e_1, \dots, e_n) = \mathbb{E}[\log p(s_1, \dots, s_n)]$$

Note that this definition is equivalent to our initial definition of Shannon entropy (3.1), if the outcome of variables e_1, \dots, e_n is instead interpreted as a single vector-valued variable.

Example 13 Consider L_4 from Fig. 5 where

$$\Sigma = \{a, b, c, d, e, x, y, z\}.$$

If we assume that the log is exactly representative of the outcomes of e_1, e_2, e_3 and e_4 , then we have the following marginal probabilities for the individual events,

$$\begin{aligned} p(e_1 = a) &= 1 & p(e_2 = x) &= 1 \\ p(e_1 = b) &= 1 & p(e_3 = y) &= \frac{1}{2} \\ p(e_1 = c) &= 1 & p(e_3 = z) &= \frac{1}{2} \\ p(e_1 = d) &= 1 & p(e_4 = y) &= \frac{1}{2} \\ p(e_1 = e) &= \frac{1}{5} & p(e_4 = z) &= \frac{1}{2} \end{aligned} \quad (4.1)$$

The probability for all other outcomes (e.g. $e_1 = x, e_2 = a$, etc.) is 0. The joint entropy is given by

$$\begin{aligned} H(e_1, e_2, e_3, e_4) &= - \sum_{s_1 \in \Sigma} \sum_{s_2 \in \Sigma} \sum_{s_3 \in \Sigma} \sum_{s_4 \in \Sigma} p(s_1, s_2, s_3, s_4) \\ &\quad \log p(s_1, s_2, s_3, s_4) \\ &= -10 \times \left(\frac{1}{5} \times 1 \times \frac{1}{2} \times \frac{1}{2}\right) \log \left(\frac{1}{5} \times 1 \times \frac{1}{2} \times \frac{1}{2}\right) \approx 2.16 \end{aligned}$$

Lemma 3 (Conditional entropy) *Let e_1, \dots, e_{n+1} be discrete random variables. The conditional entropy of e_{n+1} given e_n, \dots, e_1 is the amount of uncertainty regarding the outcome of e_{n+1} once e_n, \dots, e_1 have been observed:*

$$H(e_{n+1}|e_n, \dots, e_1) = - \sum_{s_1 \in \Sigma} \cdots \sum_{s_{n+1} \in \Sigma} p_n \log p_n$$

where

$$p_n = p(e_1 = s_1, \dots, e_{n+1} = s_{n+1})$$

It is the expected value of the conditional probability of e_{n+1} given e_n, \dots, e_1 :

$$H(e_{n+1}|e_n, \dots, e_1) = \mathbb{E}[\log p(s_{n+1}|s_n, \dots, s_1)]$$

By simple rules of probability and logarithms, we can show that conditional entropy is equivalent to the difference between the joint entropy of e_{n+1}, \dots, e_1 and e_n, \dots, e_1 :

$$\begin{aligned} H(e_{n+1}|e_n, \dots, e_1) &= - \sum_{s_1} \cdots \sum_{s_{n+1}} p(s_1, \dots, s_{n+1}) \log \frac{p(s_n, \dots, s_1 | s_{n+1}) p(s_{n+1})}{p(s_n, \dots, s_1)} \\ &= - \sum_{s_1} \cdots \sum_{s_{n+1}} p(s_1, \dots, s_{n+1}) \left(\log p(s_{n+1}, \dots, s_1) - \log p(s_n, \dots, s_1) \right) \\ &= H(e_1, \dots, e_{n+1}) \\ &\quad + \sum_{s_1} \cdots \sum_{s_n} p(s_n, \dots, s_1) \log p(s_n, \dots, s_1) \\ &= H(e_1, \dots, e_{n+1}) - H(e_1, \dots, e_n) \end{aligned}$$

Example 14 For brevity, consider a new log:

$$\{\langle a, b, c \rangle, \langle a, b, d \rangle, \langle a, c, c \rangle\}$$

We will compute the conditional entropy of e_3 , given e_2 and e_1 :

$$\begin{aligned} H(e_3|e_2, e_1) &= - \sum_{s_1 \in \Sigma} \sum_{s_2 \in \Sigma} \sum_{s_3 \in \Sigma} p(s_1, s_2, s_3) \log p(s_3|s_2, s_1) \\ &= - p(a, b, c) \log(c|b, a) \\ &\quad - p(a, b, d) \log(d|b, a) \\ &\quad - p(a, c, c) \log(c|c, a) \\ &= - \frac{1}{3} \log \frac{1}{2} - \frac{1}{3} \log \frac{1}{2} - \frac{1}{3} \log 1 = \frac{2}{3} \end{aligned}$$

Lemma 4 (Per-symbol entropy rate) *Let e_1, \dots, e_n be a sequence of n variables, representing the outcomes of a stochastic process, then the per-symbol entropy rate of the process, denoted $H(\mathcal{E})$ or h , represents how the joint entropy $H(e_1, \dots, e_n)$ grows with the length n of the sequence:*

$$h = H(\mathcal{E}) = \lim_{n \rightarrow \infty} \frac{1}{n} H(e_1, \dots, e_n) \quad (4.2)$$

In the special case in which e_1, \dots, e_n are independent and identically distributed (i.i.d.), we have

$$H(\mathcal{E}) = \lim_{n \rightarrow \infty} \frac{H(e_1, \dots, e_n)}{n} = \lim_{n \rightarrow \infty} \frac{nH(e_1)}{n} = H(e_1)$$

where $H(e_1, \dots, e_n) = nH(e_1)$ for i.i.d. variables follows from Shannon's source coding theorem and the asymptotic equipartition principle [34]. Since event logs were generated by some structured process, we cannot assume independence between variables e_1, \dots, e_n . However, we must assume *stationarity*, i.e. that the statistical properties of the process do not change over time, in order to be able to use block-based entropy rate estimators.

The reason for this is illustrated by the case in which we have independent, but not identically distributed random variables: if $p(e_i = s)$ is allowed to be a function of time (i.e. the index i). From the definition of joint entropy and independence we have

$$H(e_1, \dots, e_n) = \sum_{i=1}^n H(e_i)$$

In this case, there exists a sequence of probability distributions across e_1, \dots, e_n such that

$$H(\mathcal{E}) = \lim_{n \rightarrow \infty} \frac{1}{n} \sum H(e_i)$$

does not exist.³ In such a process, $H(\mathcal{E})$ is undefined [33].

We now define an alternative interpretation of entropy rate, which is equivalent to *per-symbol entropy rate* under the assumption of stationarity.

Lemma 5 (Conditional entropy rate) *Let e_1, \dots, e_n be a sequence of n variables, representing the outcomes of a stochastic process, then the conditional entropy rate of the process, denoted $H'(\mathcal{E})$ or h' , represents the conditional entropy of the most recently observed variable given the past, in the limit:*

$$h' = H'(\mathcal{E}) = \lim_{n \rightarrow \infty} H(e_{n+1}|e_n, \dots, e_1) \quad (4.3)$$

Theorem 1 *For stationary stochastic processes, limits in $H(\mathcal{E})$ and $H'(\mathcal{E})$ exist and*

$$H(\mathcal{E}) = H'(\mathcal{E})$$

For a proof, see [33].

³ For example, if the probability of the outcome of $p(e_i = x)$ is a function of the index i , then a probability distribution exists such that the running average of $H(e_i)$ oscillates between 0 and 1.

Table 2 Flattened block-based entropy rate

	Logs	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$
entropy^{fbl}	$L_{1,2}$	2.9	3.09	3.32	3.38	3.25	3.0
	L_3	2.85	4.5	5.1	5.0	4.86	4.09
rate_k^{fd}	$L_{1,2}$	2.9	0.19	0.24	0.05	-0.12	-0.25
	L_3	2.85	1.65	0.6	-0.1	-0.42	-0.5
rate_k^{fr}	$L_{1,2}$	2.9	1.54	1.11	0.84	0.65	0.5
	L_3	2.85	2.25	1.7	1.25	0.92	0.68

The entropy rate h represents a lower bound on the compressibility of the sequence of outcomes resulting from the underlying process, in contrast to the compressibility of the outcomes of just *one* random variable represented by entropy H (i.e. the compressibility of this variable's probability distribution). This makes entropy rate a more promising guide for selecting a process mining approach.

4.1 Block-Based Estimators

We now have a formal definition of the entropy rate of a process, but they are defined as limits where the number of observed events approaches infinity. In reality, we work with event logs of finite length and must rely on estimates given the available data. From the definitions of *per-symbol* and *conditional* entropy rates, two estimators based on block entropy follow. Namely, from

$$\begin{aligned} h &= \lim_{k \rightarrow \infty} h_k = \lim_{k \rightarrow \infty} \frac{\text{entropy}_k^{bl}}{k} \\ &= \lim_{k \rightarrow \infty} \text{entropy}_{k+1}^{bl} - \text{entropy}_k^{bl} \end{aligned}$$

we obtain the following two entropy rate estimators. Each relies on making a particular choice of k , since we cannot in practice let k tend towards infinity; we will see how to choose a value for k in the next subsection (see also [35]).

Definition 21 (*Ratio-based k -block entropy rate*) Let L be a log. The *ratio-based k -block entropy rate estimator* is given by the ratio of the k -block entropy to the block's length k . The flattened k -block estimator uses the flattened k -block estimator:

$$\text{rate}_k^r(L) = \frac{\text{entropy}_k^{bl}(L)}{k} \quad \text{rate}_k^{fr}(L) = \frac{\text{entropy}_k^{fbl}(L)}{k} \quad (4.4)$$

Example 15 Consider log L_4 in Fig. 5. The flattened 2-block entropy of the log is approximately 3.03, giving

$$\text{rate}_2^{fr}(L_4) = \frac{3.03}{2} \approx 1.51$$

Definition 22 (*Difference-based k -block entropy rate*) Let L be a log. The *difference-based k -block entropy rate estimator* is based on definition (4.3) of entropy rate and is given by the difference between the $k + 1$ -block entropy and the k -block entropy:

$$\begin{aligned} \text{rate}_k^d(L) &= \text{entropy}_{k+1}^{bl}(L) - \text{entropy}_k^{bl}(L) \\ \text{rate}_k^{fd}(L) &= \text{entropy}_{k+1}^{fbl}(L) - \text{entropy}_k^{fbl}(L) \end{aligned} \quad (4.5)$$

Example 16 Consider logs L_1 and L_3 in Fig. 1. The k -block entropy and corresponding block-based entropy rates for $1 \leq k \leq 6$ are shown in Table 2.

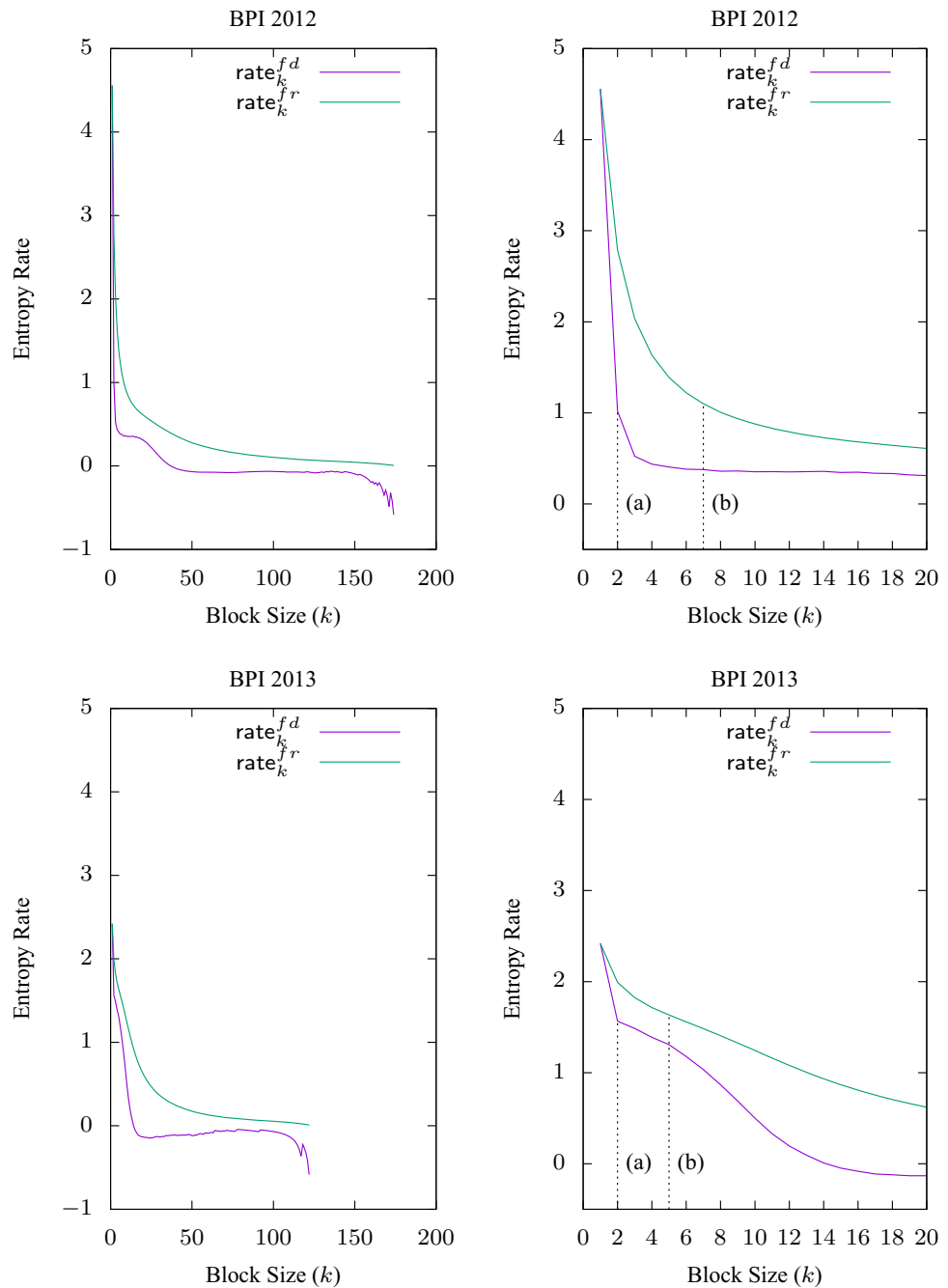
As we see blocks of increasing length, the estimators approach the true entropy rate of the underlying process from above. While they are equal in the limit, at any $k < \infty$, $\text{rate}_k^r \geq \text{rate}_k^d \geq h$ [35].

However, block-based estimators break down at longer block lengths due to the lack of sufficient samples for the frequencies to serve as a valid empirical estimate of the true probability $p(\langle s_1, \dots, s_k \rangle)$ [23,35]. This is seen clearly in Fig. 7 by the fact that as k grows the entropy rate falls to zero and even goes negative. We now discuss methods for determining a limit to block length that guarantees a sufficient number of samples.

4.1.1 Sufficient Statistics

Entropy rate is formally defined for a sequence as its length approaches infinity, but in practice we must choose a finite value of k for the estimators, (4.4), (4.5). We cannot simply choose the k corresponding to the longest trace due to *under-sampling*: there are not enough examples of k -blocks of this length and all but the longest traces will be omitted. We must therefore find a strategy for choosing a k which is small enough that a sufficient number of k -blocks are observed in the log, yet large enough that the k -blocks reach a length such that the entropy estimate begins to converge to the “true” entropy rate as defined in (4.2) and (4.3). We present five methods for choosing k based on previous research on estimating entropy of very short sequences, drawing heav-

Fig. 7 Block-based estimators of entropy rate (h), as a function of block size (k) for selected logs. The cut-off constraints, $Kh \geq k|\Sigma|^k \log |\Sigma|$ and $k < \frac{\log K}{h}$, are labelled by (a) and (b), respectively. See (4.9) and (4.6)



ily upon [35], where detailed derivations of the following constraints can be found.

Interestingly, these constraints in some cases rely on knowing upfront the “true” entropy, which one obviously does not; in practice, one implements these constraints in an iterative fashion, using in each round the previous estimate of entropy as the “true” entropy.

In a scenario of adequate sampling, the length of k -blocks is bounded above by the following function of K , the total length of a sequence, h the entropy rate, and alphabet size $|\Sigma|$:

$$k < \frac{\log K}{h} \quad \text{if } h \rightarrow \mathcal{O}(1) \quad (4.6)$$

$$k < \frac{Kh}{\log |\Sigma|} \quad \text{if } h \rightarrow 0 \quad (4.7)$$

where $\mathcal{O}(1)$ denotes some nonzero constant, meaning that the bound depends on whether the underlying process is fully predictable in the limit.

For “short” sequences (length of less than about 1200, according to [35], as will be common in a BPM context where

sequences are event traces)⁴ we are in a scenario of under-sampling meaning that the bound in (4.6), (4.7) will be too lenient. To determine stricter bounds on k , the authors of (see [35]) draw upon research showing that block-based entropy estimates tend to break down when $|\Sigma|^k \approx K$ [23] and the fact that a sequence of length K contains $\frac{K}{k}$ non-overlapping k -blocks to derive the constraint

$$K \geq k|\Sigma|^k \quad (4.8)$$

This formulation assumes sequences of i.i.d. variables, i.e. that correlations between k -blocks result exclusively from their overlap. Dropping this assumption leads to a more stringent constraint based on an adjusted sequence length scaled by the entropy rate, which serves as a measure of correlation not due to block overlap. The upper bound on entropy rate is given by $\log |\Sigma|$, which represents a scenario in which every outcome in Σ is equally probable, regardless of the preceding observations, i.e. $p(s_n | s_{n-1}, \dots, s_1) = p(s_n)$. This leads to a normalised entropy rate $\frac{h}{\log |\Sigma|}$, which can be used to define an “effective” sequence length

$$K_{\text{eff}} \approx K \frac{h}{\log |\Sigma|}$$

Substituting K with K_{eff} in (4.8), we have the new constraint

$$Kh \geq k|\Sigma|^k \log |\Sigma| \quad (4.9)$$

However, taking advantage of the asymptotic equipartition property we can relax this constraint, diminishing the influence of the size of the alphabet. Specifically, for discrete time, finite-valued, stationary, ergodic sources the Shannon–McMillan–Breiman theorem, “states that the number of k -blocks of non-negligible probability that actually contribute to entropy is not $|\Sigma|^k$, but 2^{kh} ” (see [35] and [33]), giving

$$Kh > k2^{kh} \log |\Sigma| \quad (4.10)$$

Using these constraints, we can find the highest value of k such that the estimators rate_k^r and rate_k^d remain valid, i.e. that enough examples of blocks of length k have been observed to consider their frequencies (3.3) as a reasonable estimate of the true underlying probability distribution.

We stress that these constraints are formally defined for single sequences of length K , whereas we are considering sets of sequences, which affects the number of k -blocks we will observe, depending on the distribution of trace lengths in the log: some logs may consist almost entirely of very

short traces with only a few long traces, in which case longer k -blocks will still be under-sampled. In our implementation we have used the length of the longest trace as K . Our current results use the constraints defined for single sequences, leaving the generalisation to future work.

In summary, to apply either of the entropy estimators of Definition (4.4) or (4.5), first choose one of the constraints (4.6)–(4.10); then, apply the estimator at the maximum k which satisfies that constraint.

Example 17 Consider $\log L_3$ from Fig. 1. We will compute rate_k^d up to the highest k which satisfies constraint (4.9). The longest trace is $\langle a, d, a, c, a, c, e, h, h, f, g \rangle$ with a length of 11. The alphabet has size

$$|\Sigma| = |\{a, b, c, d, e, f, g, h\}| = 8.$$

The highest value for k which satisfies constraint (4.9) is just 1; this gives $\text{rate}_1^{fd} = \text{entropy}_2^{fbl} - \text{entropy}_1^{fbl} = 4.5 - 2.85 = 1.65$.

k	rate_k^{fd}	$Kh \geq k \Sigma ^k \log \Sigma $
1	2.85	$11 \times 2.85 \geq 1 \times 8^1 \log 8 \iff 31.35 \geq 24$
2	1.65	$11 \times 1.65 \not\geq 2 \times 8^2 \log 8 \iff 18.15 \not\geq 384$

4.2 Lempel–Ziv Estimators

Lempel–Ziv entropy estimators are based on the properties of the sequence compression algorithms introduced by Abraham Lempel and Jacob Ziv in two papers from 1977 and 1978 [36,37]. These algorithms can be proven to be asymptotically optimal: in the limit they will converge to the entropy rate h when compressing a sequence resulting from a stationary, ergodic source [33].

Both versions work by parsing a sequence into unique words (blocks) and are universal coding schemes [33], meaning that they do not rely on the probability distribution of the underlying source. The cost of universality is a higher complexity of the encoder and decoder.

We will use the 1978 version of the compression scheme which moves through a sequence $\langle e_1, \dots, e_n \rangle$, parsing it into the shortest unique word not yet encountered.

Example 18 Consider the last trace from our running example $\log L_3$ in Fig. 1 (a start symbol is included for indexing purposes):

$$\langle \$, f, g, h, f, g, h, h, h \rangle$$

⁴ In the literature on entropy estimation, sequences of this length are considered relatively short since the data under consideration are often large text corpora or outputs from simulations of dynamic systems.

The first shortest word encountered is f , the second g and the third h . The fourth element f has already been encountered, so we append the fifth element to the word to arrive at a new word fg . The sixth element h has been seen, so the seventh is appended giving hh . The final element h has already been seen, so no new words are added to the dictionary. This results in the following dictionary in which each word is represented by a tuple containing the index of a previously encountered word, and a character to append, creating a new word:

$$(\$, f, g, h, fg, hh) = \langle \langle 0, f \rangle, \langle 0, g \rangle, \langle 0, h \rangle, \langle 1, g \rangle, \langle 3, h \rangle \rangle$$

Definition 23 (*Lempel–Ziv entropy rate*) Let L be a log. Let D_L be the dictionary of subsequences resulting from the parsing of all traces in L using the Lempel–Ziv compression scheme. Then the Lempel–Ziv estimate of the entropy rate of the process generating L is given by

$$\text{rate}^{LZ}(L) = \frac{|D_L| \log \sum_{\sigma \in L} |\sigma|}{\sum_{\sigma \in L} |\sigma|}$$

Definition 24 *Flattened Lempel–Ziv entropy rate*

$$\text{rate}^{fLZ} = \text{rate}^{LZ} \circ \text{flatten}(L)$$

5 Implementation and Experiments

We present in this section the results of empirically measuring entropy as defined by the various measures in the preceding sections on both synthetic and real-life logs.

5.1 Implementation

To test the various measures we implemented a command-line utility for computing the measures introduced in the preceding sections. Using this implementation, we report here the values of the discussed measures.

The implementation is not entirely trivial: in particular, use of both prefix and suffix tree data structures was necessary to compute the global block entropy in reasonable time on available memory. We note that the Hospital Log and BPI Challenge 2017 were both particularly challenging in this respect. We also used an iterative version of the Levenshtein distance algorithm with a number of heuristics (checking lower bounds and common prefixes and suffixes) to avoid unnecessary calculations. While this drastically improves performance, this estimator clearly stands out as the most computationally expensive estimator.

The implementation, along with our results in machine-readable format, is available at [38].

5.2 Real-Life Logs

We have evaluated a selection of real-life logs available at [39]. We summarise the logs in Table 3.

For those logs which include distinguishing life cycle transitions for activities such as “start”, “pending” or “complete”, we include both the log in which these are included (denoted with an asterisk) and those in which they are ignored. When life cycle transitions are included, activities like “ $a + \text{start}$ ” and “ $a + \text{pending}$ ” are considered unique events. This is arguably the only defensible approach since life cycle transitions are included to make a distinction between activities and there is no reason to throw this information away when estimating entropy. However, in the interest of completeness, we report on both versions.

There is not yet a clear agreement in the literature on which of these logs are better suited for imperative or declarative mining. However, the BPI Challenge 2012 log has been used as a use case for hybrid mining algorithms [41], showing a strong advantage over purely imperative miners. We also note that both the Sepsis Cases and Hospital Log originate from highly flexible and knowledge-intensive processes within a Dutch hospital. A recent investigation involving the BPI Challenge 2013 (incidents) log seemed to indicate that an imperative approach may be the more successful, but draws no concrete conclusions [12]. For every log of Table 3, we report measurements of entropy (Sect. 3) in Table 4, and measures of entropy rate (Sect. 4.1) in Table 5. We also present graphs of the block-based entropy rate as a function of block length (k) for two logs, to illustrate how the estimators converge differently.

5.3 Artificial Logs

In order to further illuminate the potential capability of entropy measures to distinguish declarative from imperative logs, we apply in this section the measures to a selection of artificially generated logs. We evaluate the measures on three sets of logs: one generated from Petri nets with varying degrees of noise, one generated from Declare models with varying degrees of restrictiveness, and a log with varying degrees of concurrency.

5.3.1 Petri Net-Based Logs with Noise

To evaluate the effect of noise and infrequent behaviour in imperative processes on entropy estimates, we use the set of 120 event logs from [42]. These logs, each containing 1000 traces, are generated from four typical imperative process patterns which can cause difficulty for process mining algorithms. Then, five different forms of noise are introduced in the form of an activity which is added (1) infrequently or very infrequently and (2) locally (in one position), semi-

Table 3 Overview of real-life logs measured as well as running examples. All logs but L_1 through L_4 from [39]

Log	Act.	Traces	Events	Comment
<i>Declarative indications</i>				
Hospital Event Log	624	1143	150291	Knowledge-intensive process
Road Traffic Fine	11	150370	561470	Declarative indications in [12]
Sepsis Cases	16	1050	15214	Highly flexible, knowledge-intensive process, declarative indications in [12,40]
L_3	8	40	280	Declarative by construction
<i>Hybrid indications</i>				
BPI Challenge 2012	24	13087	262200	Loan application process, successfully
BPI Challenge 2012*	36	13087	262200	Used for hybrid mining [41]
<i>Imperative indications</i>				
BPI Challenge 2013	4	7554	65533	Imperative indications in [12]
BPI Challenge 2013*	13	7554	65533	”
NASA CEV split*	94	2566	73638	Imperative miners return succinct models
L_1	8	40	280	Imperative by construction
L_2	8	40	280	Language equivalent with L_1
L_4	8	50	280	Imperative by construction
<i>No indications</i>				
BPI Challenge 2017	26	31509	1202267	
BPI Challenge 2017*	66	31509	1202267	
WABO-Receipt	27	1434	8577	
Hospital Billing	18	100000	451359	

Logs marked with an asterisk (*) include the life cycle transition of events

Table 4 Estimates of log entropy (H)

Log	entropy ^{tr}	entropy ^{fpr}	entropy ^{fbl}	entropy ^{fKL}	entropy ^{fkNN}			
					k = 1	k = 2	k = 3	k = 4
<i>Declarative indications</i>								
Hospital Event Log	9.63	16.79	24.71	7.13	7.13	6.22	5.77	5.47
Road Traffic Fine	2.48	6.51	8.73	4.64	4.64	3.76	3.4	3.18
Sepsis Cases	9.33	10.6	14.66	6.16	6.16	5.35	4.96	4.7
L ₃	3.0	5.63	7.04	2.78	1.3	1.99	1.56	1.27
<i>Hybrid indications</i>								
BPI Challenge 2012	7.75	12.53	15.99	7.28	7.28	6.47	6.07	5.81
BPI Challenge 2012*	7.75	12.54	16.28	7.34	7.34	6.51	6.11	5.85
<i>Imperative indications</i>								
BPI Challenge 2013	6.67	11.32	12.23	6.17	6.17	5.28	4.86	4.58
BPI Challenge 2013*	7.48	11.81	14.01	6.95	6.95	6.07	5.64	5.36
NASA CEV split*	11.27	10.12	14.81	6.84	6.84	5.97	5.51	5.27
L ₁	3.0	4.09	5.75	1.17	1.3	0.37	− 0.09	0.13
L ₂	2.55	4.09	5.75	1.17	1.3	0.37	− 0.09	0.13
L ₄	3.32	4.82	4.75	2.08	2.19	1.19	0.69	0.35
<i>No indications</i>								
BPI Challenge 2017	11.99	14.42	17.45	8.16	8.16	7.38	7.01	6.76
BPI Challenge 2017*	12.23	14.63	18.24	8.53	8.53	7.71	7.31	7.05
CoSeLog-Receipt	3.21	7.72	10.05	4.43	4.44	3.63	3.26	3.02
Hospital Billing	3.17	9.43	10.49	6.04	6.04	5.22	4.87	4.64

Logs marked with an asterisk (*) distinguish events by life cycle transition

Table 5 Estimates of entropy rate, h , according to the Lempel–Ziv estimator, the difference-based k -block estimator and the ratio-based k -block entropy estimator. For the latter two, block length k is determined according to five different constraints for identifying the crossover from “good statistics” to “poor statistics” [35] on short sequences

Log	rate ^{<i>flz</i>}	rate ^{<i>fd</i>} _{<i>k</i>+1} = entropy ^{<i>fl</i>} _{<i>k</i>+1} − entropy ^{<i>fl</i>} _{<i>k</i>}		rate ^{<i>fb</i>} _{<i>k</i>}		$k + 1$		Σ	K
		(4.6)	(4.7)	(4.8)	(4.9)	(4.6)	(4.7)	(4.8)	(4.9)
Declarative indications									
Hospital Event Log	2.89	≈ 0	0.08	2.2	2.2	33	18	2	5
Road Traffic Fine	1.52	−0.1	0.46	1.55	1.55	7	5	2	2
Sepsis Cases	1.89	−0.03	0.22	1.88	1.88	12	10	2	3
L_3	2.9	−0.1	0.6	1.65	1.65	4	3	2	1
Hybrid indications									
BPI Challenge 2012	0.96	−0.01	0.37	1.33	1.33	38	14	2	5
BPI Challenge 2012*	0.98	−0.01	0.35	1.02	1.02	38	12	2	5
Imperative indications									
BPI Challenge 2013	1.16	0.78	0.18	0.88	0.88	9	17	3	5
BPI Challenge 2013*	1.71	1.18	0.33	1.57	1.57	6	11	2	3
NASA CEV split*	0.84	6.1	0.5	6.1	6.1	1	4	1	1
L_1	2.18	−0.12	0.19	0.19	0.19	5	2	2	1
L_2	2.07	−0.12	0.19	0.19	0.19	5	2	2	1
L_4	2.13	2.58	0.45	2.58	2.58	1	2	1	1
No indications									
BPI Challenge 2017	0.89	≈ 0	0.39	1.33	1.33	42	15	2	5
BPI Challenge 2017*	0.97	−0.01	0.42	0.9	0.9	40	13	2	4
WABO-Receipt	2.34	−0.17	0.48	3.77	3.77	8	5	1	2
Hospital Billing	1.58	−0.11	0.06	1.42	1.42	10	9	2	4
Declarative indications						k			
Hospital Event Log	”	3.3	0.29	5.92	5.92	3	57	1	2
Road Traffic Fine	”	3.26	1.2	3.26	3.26	1	6	1	1
Sepsis Cases	”	2.28	0.49	3.22	3.22	3	22	1	2
L_3	”	2.85	1.25	2.85	2.85	1	4	1	1
Hybrid indications									
BPI Challenge 2012	”	1.05	0.58	3.56	3.56	7	21	1	2
BPI Challenge 2012*	”	1.22	0.61	4.56	4.56	6	20	1	1
Imperative indications									
BPI Challenge 2013	”	0.94	0.44	1.11	1.11	7	26	2	3
BPI Challenge 2013*	”	1.71	0.62	2.42	2.42	4	20	1	2
NASA CEV split*	”	6.1	1.22	6.1	6.1	1	8	1	1
L_1	”	2.9	1.54	2.9	2.9	1	2	1	1
L_2	”	2.9	1.54	2.9	2.9	1	2	1	1
L_4	”	2.58	1.51	2.58	2.58	1	2	1	1
No indications									
BPI Challenge 2017	”	1.41	0.61	3.66	3.66	5	23	1	2
BPI Challenge 2017*	”	1.86	0.71	5.27	5.27	4	20	1	1
WABO-Receipt	”	3.77	1.38	3.77	3.77	1	6	1	1
Hospital Billing	”	1.52	0.35	3.44	3.44	5	16	1	2

K denotes the length of the longest trace and |Σ| the size of the alphabet, i.e. number of activities. Logs marked with an asterisk (*) distinguish events by life cycle transition

locally (in one of two positions) or globally (anywhere in the process). Along with the noise-free log, this results in six sets of five logs each for each of the four process patterns. (The semi-local/very infrequent noise combination is omitted in the original logs.)

The four process patterns employed are as follows:

Parallel Five activities which can occur once, in any order.

Skip Three activities, one of which may be skipped.

Duplicates A sequence of activities with one activity occurring twice.

Non-free Choice A choice at the end of the process depends on an earlier choice.

Observations We report the results of experiments in Figs. 8 and 9.

First, we note that entropy measures tend to agree on the relative ordering of the four logs, with the exception of *Skip* and *Duplicates*. Entropy rate measures, on the other hand, tend to rank *Non-free Choice* as having the highest entropy rather than *Parallel*.

Most entropy measures are affected by noise, with the exception of nearest neighbour approaches on the *Parallel* log. Entropy rate measures perform much more poorly when faced with noise. We note in particular that entropy measures generally preserve the *relative ordering* of the four logs, whereas entropy rate measures exhibit much more “crossover”: the addition of noise switches the relative ranking of two or more of the logs.

5.3.2 Declare-Based Logs

We evaluate the output of entropy estimators on declaratively generated logs in terms of model restrictiveness versus expressiveness. Specifically, we measure not only the effect of the number of constraints, but also *types* of constraints, as well as number of activities. Each log consists of 1000 traces with lengths between 5 and 10 events, in order to ensure a comparability with the imperatively generated logs, which are of similar length.

Logs were generated from Declare models using the artificial log generator described in [43]. Insofar as possible, models were built by varying along one dimension at a time, holding others constant. That is, to measure the effect of number of constraints, the type of constraint and number of activities were held constant, and number of constraints incrementally increased. Similarly, to measure the effect of type of constraint, the number of constraints and activities was held constant, while the restrictiveness of the constraint type was increased *within its subsumption hierarchy*.

Constraint ordering Declare constraints fall into a partial ordering in terms of their restrictiveness [44]. First, they fall

into a subsumption hierarchy within the same constraint type, e.g. *AlternateSuccession* is more restrictive than *Succession*. Second, some constraint types are more restrictive than others, e.g. *Succession* is more restrictive than *Response*.

When considering the size of the model space resulting from varying these three dimensions (number of constraints, type of constraints and number of activities), we have favoured systematically adjusting individual parameters over exploring larger, more diverse models. The configuration of constraints adds a potential fourth dimension, which we have chosen to hold constant unless doing so resulted invalid models, which only was the case for some *Chain* constraints (Fig. 10).

Monotonicity Finally, we note that the effect of alphabet size in Declare models on entropy measures will always be monotonically increasing when other factors are held constant, since this increases the number of possible outcomes in the sample space, broadening and flattening the probability distribution. Recall also the conjunctive nature of Declare: adding constraints always results in a more restrictive model. **Observations** We report the results of experiments along with examples of the Declare models in Figs. 11, 12, 13 and 14.

The resulting entropy measures largely fall in line with expectations, with rate_k^{fd} and rate_k^{fr} displaying a pronounced sensitivity to the choice of k -block cut-off constraint.

The most marked trend is the clear effect of *constraint type*. A clear drop in all entropy measures occurs in models with *Alternate* and *Chain* constraint types, with the effect of increased number of constraints being more pronounced for these constraint types as well.

Notably, the number of constraints has a very diminished effect for models consisting of less restrictive constraint types so that, for example, models with five *CoExistence* constraints have a higher entropy across estimators than a model with just one *AlternateSuccession* constraint.

Furthermore, we note that the results mirror the restrictiveness between *Parallel* branches of constraint subsumption hierarchies: *Response* is somewhat more permissive than *Succession*, and this trend is also clear though less pronounced than the effect with subsumption hierarchies. Finally, the effect of adding activities is as expected: models with more activities result in higher-entropy estimates.

The effect is clear: once models approach a degree of expressiveness akin to imperative models of moderate complexity, the entropy measure suddenly begins dropping. This matches the intuition that a process resulting from a flexible declarative process will have markedly higher entropy, even if that model consists of many semantically meaningful constraints.

Artificial Logs - Imperative with Noise

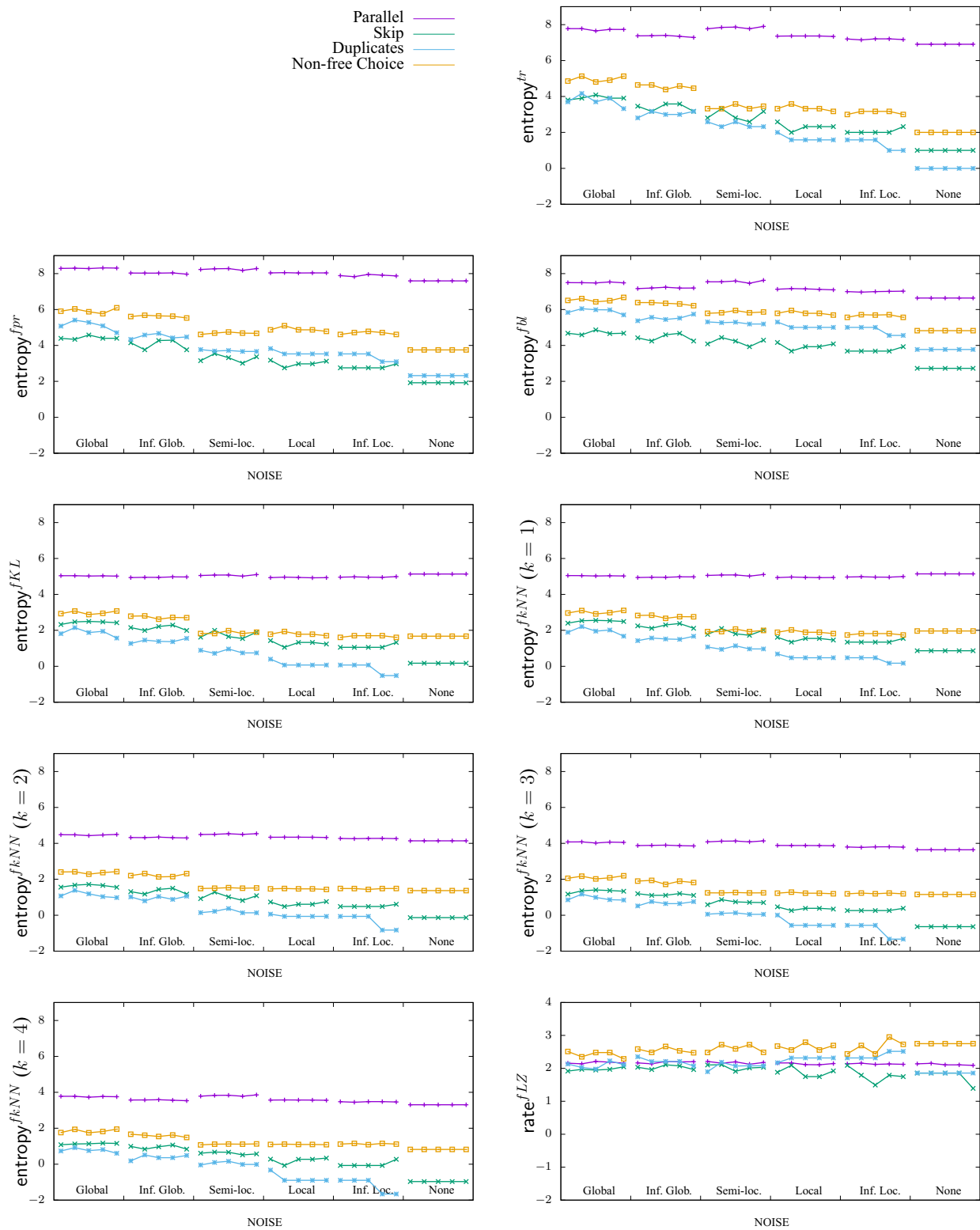


Fig. 8 Estimates of entropy (H) and entropy rate (h) for the “Testing Representational Bias” set of artificial logs. Logs have different degrees and types of noise: infrequent, very infrequent (“Inf.”); and global, semi-local and local

Artificial Logs - Imperative with Noise

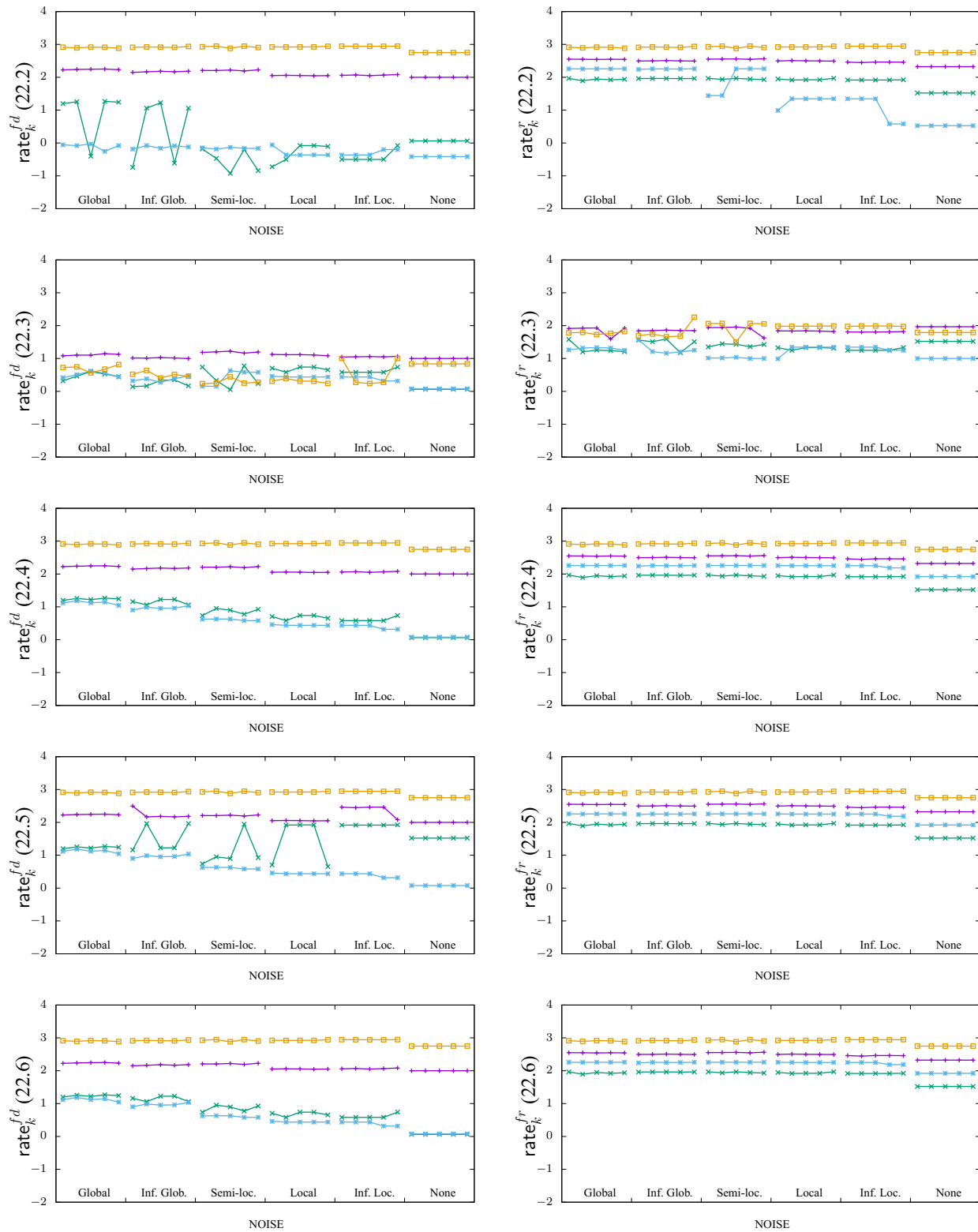


Fig. 9 Estimates of entropy (H) and entropy rate (h) for the “Testing Representational Bias” set of artificial logs. Logs have different degrees and types of noise: infrequent, very infrequent (“Inf.”); and global, semi-local and local

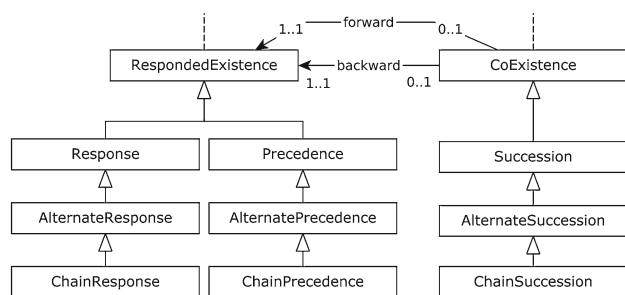


Fig. 10 Portion of the Declare constraint subsumption hierarchy from [43]

5.3.3 Concurrent Log

In order to study in detail the effect of concurrency on entropy estimates, we generated a set of logs based on models similar to the *Parallel* model above, which vary in the number of

activities in the concurrent block and are flanked by tails of sequential activities (see Fig. 15). Specifically, we consider traces with a total length ranging from 9 to 26 (with number of unique activities equal to the total trace length). For each total trace length, we generate event logs with blocks of concurrency ranging from 1 to 9 activities. Denoting the number of activities in the concurrent block by j , each event log contains $j!$ unique traces.

Observations We report the results of experiments in Fig. 16.

As the block of concurrency grows, we see that estimates of entropy (H) consistently increase. Estimates of entropy rate are less consistent: several block-based estimators level off at a high value (3–5 depending on total trace length) very quickly because the constraint on block length is violated almost immediately. Other block-based estimators grow logarithmically, appearing to converge to values between 2–3.

Most striking is the Lempel–Ziv estimator which *falls* with concurrent block size, levelling off at values between

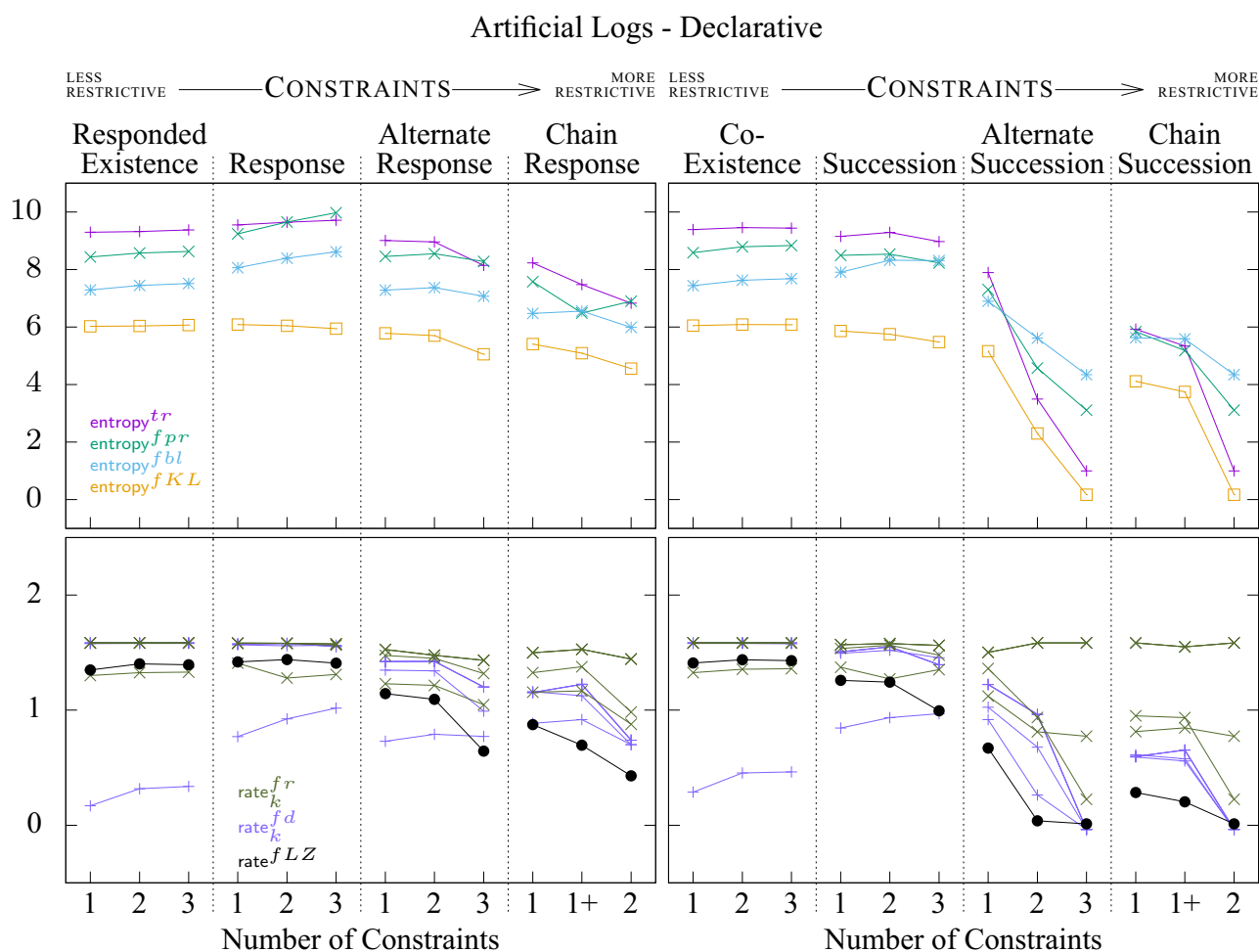
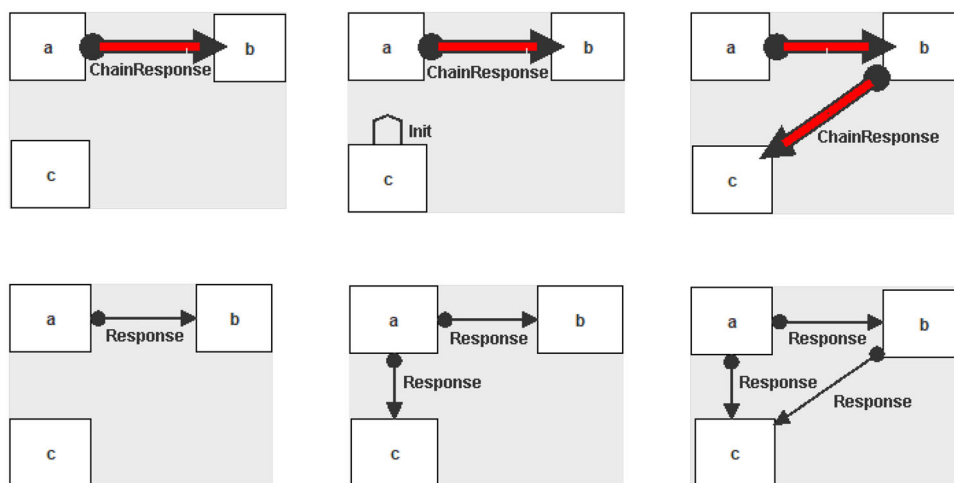


Fig. 11 Entropy measure of artificial event logs generated from Declare models with three activities and different numbers, and types, of constraints. All models consist of the same type of constraint and con-

figuration, excepting *Chain* constraints. Constraints are listed in order of restrictiveness, i.e. their subsumption ordering

Fig. 12 Examples of the Declare models used to generate artificial logs. All models follow same the configuration as the *Response* models displayed, except for *ChainResponse* since this would lead to an invalid model. In this case, the *Init* constraint has been introduced to create a corresponding number of models with a similarly increasing degree of restrictiveness



Artificial Logs - Declarative

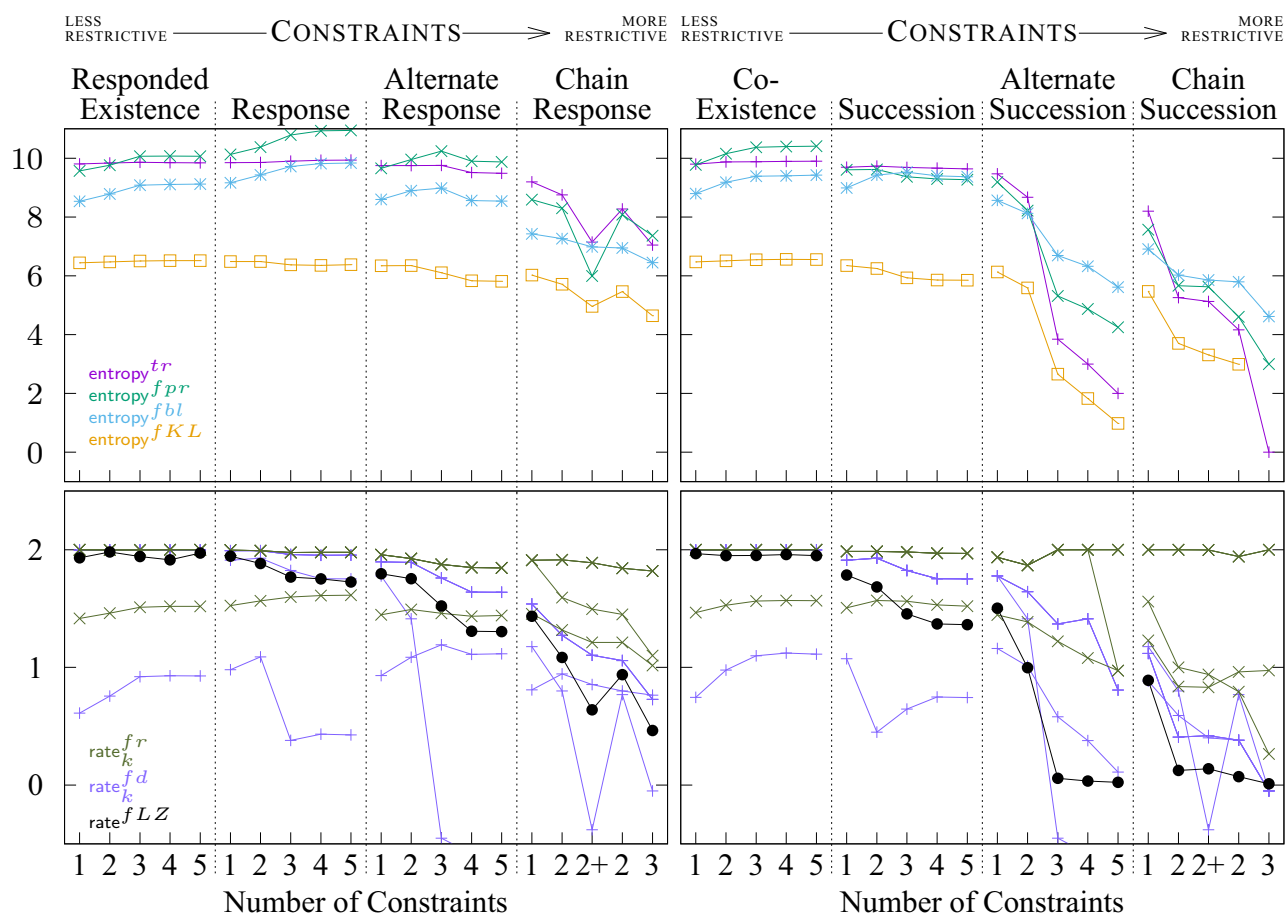


Fig. 13 Entropy measure of artificial event logs generated from Declare models with four activities and different numbers, and types, of constraints. All models consist of the same type of constraint and con-

figuration, excepting *Chain* constraints. Constraints are listed in order of restrictiveness, i.e. their subsumption ordering

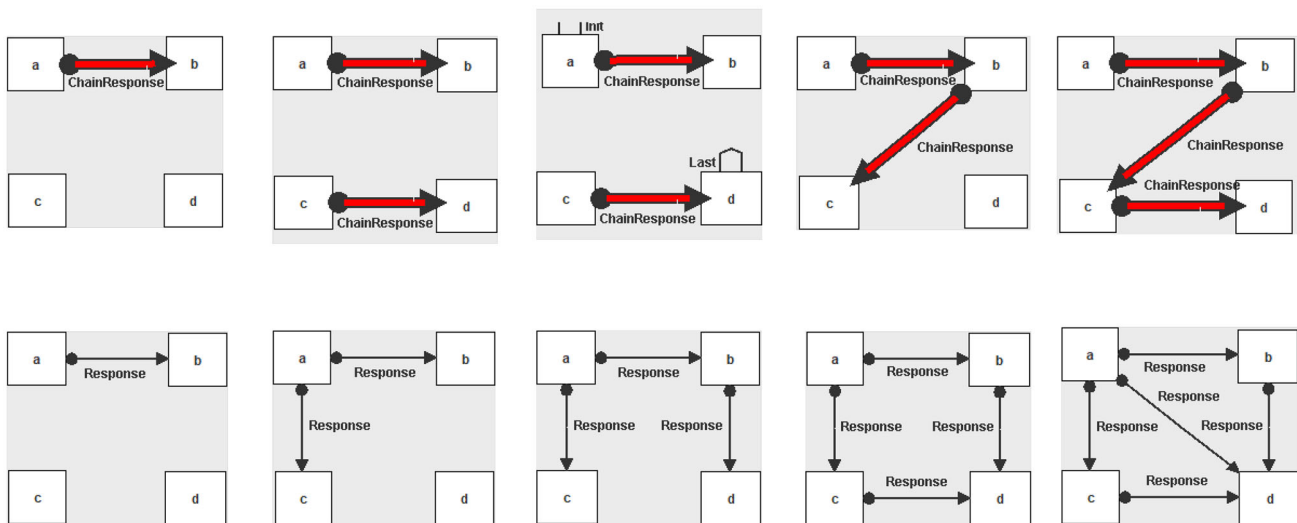


Fig. 14 Examples of the Declare models used to generate artificial logs. All models follow same the configuration as the *Response* models displayed, except for *ChainResponse* since this would lead to an

invalid model. In this case, the *Init* and *Last/End* constraints have been introduced to create a corresponding number of models with a similarly increasing degree of restrictiveness

1 and 2.5 depending on total trace length. The high estimates for blocks of length 1, 2 and 3 are most likely due primarily to smaller log sizes. Concurrent logs with many traces will nonetheless contain many shared subsequences between traces, allowing for greater compression. In this sense, Lempel–Ziv in particular and entropy *rate* estimates in general are not “tricked” by concurrency in the same way that simple entropy estimates are prone to.

The motivation for considering logs with blocks of concurrency logs is the intuition that, although this behaviour can be succinctly captured by a simple Petri net, the large degree of variation between traces would perhaps lead to inappropriately high-entropy values. However, a concurrent block can be equally, if not more, succinctly modelled using a declarative model with an *ExactlyOne* constraint on each activity in the block. This is an edge case, and in the circumstance in which a large degree of concurrency exists, but with some constraints, declarative models are capable of capturing this behaviour much more succinctly.

Those logs in which long sequential tails flank a concurrent block may indeed be more succinctly modelled imperatively, but the lower values given by several entropy rate estimators, in particular Lempel–Ziv, do in fact reflect the greater degree of structure in such logs. This is in line with the claim that low-entropy logs are better suited to be modelled imperatively. To see this, consider that for logs with a concurrent block of size 9 and no tails, the Lempel–Ziv entropy is high: about 2.5, whereas in the case of the same block flanked by sequential tails of length 9, the entropy falls to about 1.0 (see Fig. 16).

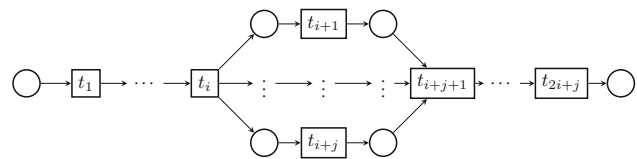


Fig. 15 A Petri net representing a block of concurrency of size j flanked by sequential tails of length i on either end. Activities in the concurrent block can be ordered in $j!$ different permutations

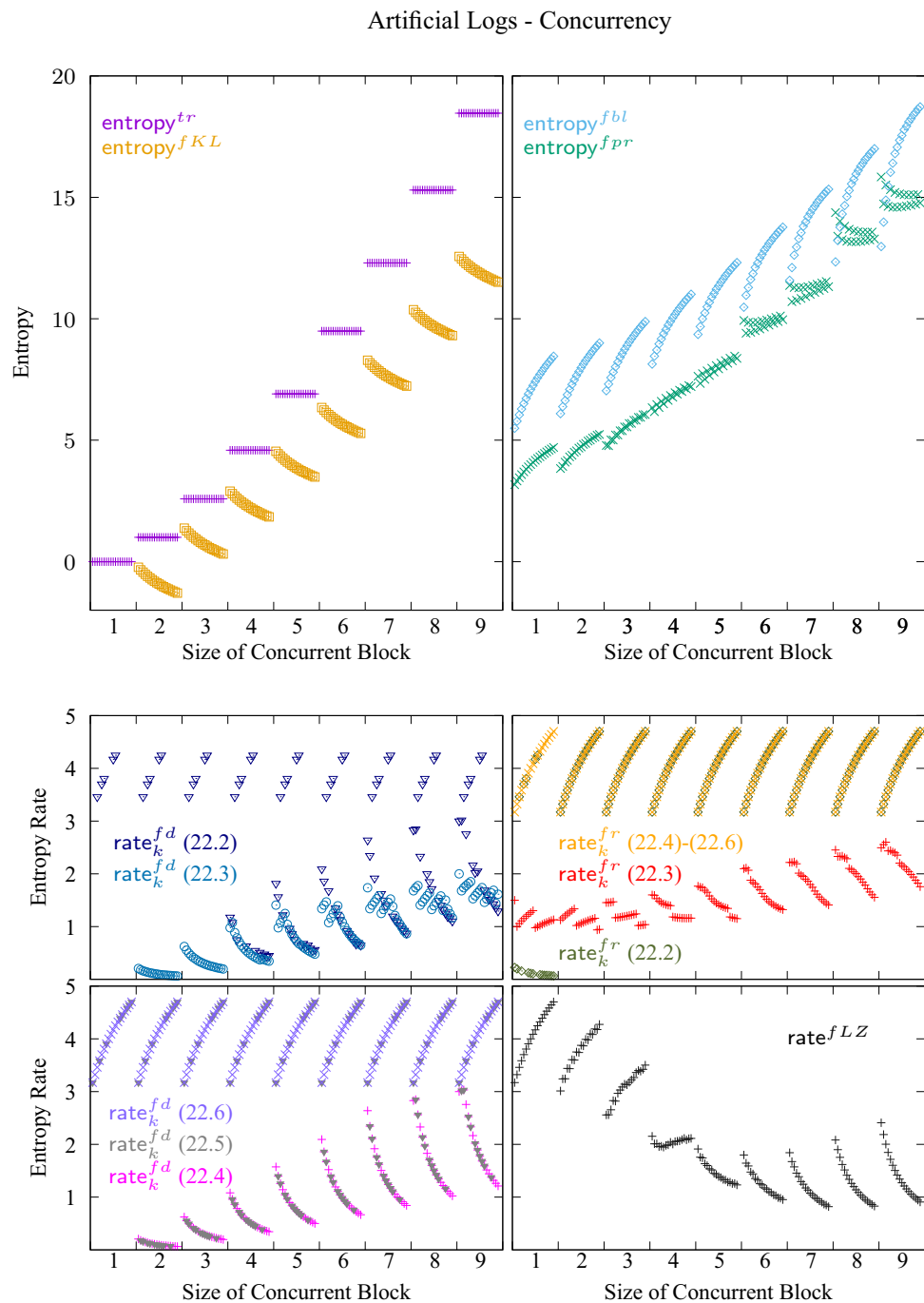
6 Discussion

In the previous section we reported entropy measurements generated by a variety of estimators. Most measures were broadly in line with expectations, especially on artificially generated logs, with some proving more robust than others.

To guide our investigation, we designed the synthetic logs L_1 , L_2 , L_4 to be suitable for imperative mining, and L_3 for declarative mining. Moreover, as indicated in Table 3, the sentiment in the community is that the BPI Challenge 2012, Hospital and Sepsis Cases logs are well suited for declarative mining. Finally, we employed several sets of artificial logs for which the generating model is known. If we take as canonical these indicators, the most promising measures for predicting suitability for imperative and declarative mining appear to be:

$$\begin{aligned} & \text{entropy}^{fbl} \\ & \text{entropy}^{fKL} \text{ and } \text{entropy}^{fKNN} \\ & \text{rate}^{fLZ} \\ & \text{rate}_k^{fd} \text{ and } \text{rate}_k^{fr} \text{ using constraint (4.9)} \end{aligned}$$

Fig. 16 Estimates of entropy (H) and entropy rate (h) of artificial concurrent logs. Data points are clustered by size of concurrent block (j) with each cluster containing 18 points, representing traces with total length from ranging from 9 to 26, from left to right within each cluster



We summarise these particular measures in Table 6. No one measure is able to classify all logs exactly as desired, which is unsurprising. In practice, several of the strongest estimators might be used in combination, or even as input to a classification algorithm along with other log attributes (Table 7).

We note that these five estimators perform quite differently when confronted with noise. In particular, $rate^{fLZ}$ is unstable in the presence of noise. The remaining estimators are either able to clearly identify noise, or are insensitive to it, in both cases maintaining a consistent relative ranking of

logs. Whether sensitivity to noise is desirable will depend on the task at hand (Table 8).

Aside from the case of artificial logs, the present evaluation suffers from a significant degree of uncertainty regarding the labelling of logs as “declarative” or “imperative”. A more thorough evaluation would involve a *quantitative* analysis of the logs to determine whether they are better suited for imperative or declarative mining according to objective criteria. One way to approach such an analysis could be to mine each log with state-of-the-art imperative and declarative miners

Table 6 Most promising measures across real-life logs and running examples

	Log	Entropy		Entropy rate		
		entropy^{fbl}	entropy^{fKL}	rate^{LZ}	$\text{rate}_k^{fd} (4.9)$	$\text{rate}_k^{fr} (4.9)$
Declarative indications	Hospital Event Log	24.71	7.25	2.89	2.2	5.92
	Road Traffic Fine	8.73	3.59	1.52	1.55	3.26
	Sepsis Cases	14.65	5.04	1.89	1.88	3.22
	L_3	7.04	2.42	2.9	1.65	2.85
Hybrid indications	BPI Challenge 2012	15.99	7.19	0.96	1.33	3.56
	BPI Challenge 2012*	16.28	7.19	0.98	1.02	4.56
Imperative indications	BPI Challenge 2013	12.22	5.34	1.16	0.88	1.11
	BPI Challenge 2013*	14.01	5.51	1.71	1.56	2.42
	NASA CEV split*	14.81	4.9	0.84	6.1	6.1
	L_1	5.75	1.15	2.18	2.9	2.9
	L_2	5.75	1.15	2.07	2.9	2.9
	L_4	4.75	1.17	2.13	2.58	2.58
	BPI Challenge 2017	17.45	7.4	0.89	1.33	3.66
No indications	BPI Challenge 2017*	18.24	7.57	0.97	0.9	5.27
	WABO-Receipt	10.05	3.74	2.34	3.77	3.77
	Hospital Billing	10.49	5.16	1.58	1.42	3.44

Logs with declarative, hybrid and imperative indications should have the highest, middle and lowest values, respectively. No estimators are able to separate logs exactly as desired. Logs marked with an asterisk (*) distinguish events by life cycle transition

Table 7 Influence of noise on estimators of entropy and entropy rate

Insensitive to noise	Correctly distinguishes noise	Partly correct	Skewed by noise
$\text{rate}_k^{fr} (4.8)$	entropy^{tr}	entropy^{fKL}	$\text{rate}_k^{fd} (4.6)$
$\text{rate}_k^{fr} (4.9)$	entropy^{fpr}	$\text{entropy}^{fkNN} (k = 1)$	$\text{rate}_k^{fd} (4.7)$
$\text{rate}_k^{fr} (4.10)$	entropy^{fbl}	$\text{rate}_k^{fd} (4.9)$	$\text{rate}_k^{fr} (4.6)$
	$\text{entropy}^{fkNN} (k = 2, 3, 4)$	$\text{rate}_k^{fr} (4.7)$	rate^{LZ}
	$\text{rate}_k^{fd} (4.8)$		
	$\text{rate}_k^{fd} (4.10)$		

and compare the resulting models according to accepted quality criteria which can be applied to models of both paradigms, e.g. [45]. The concrete outlines for such a study have been proposed in [46].

6.1 Entropy Measures (H)

We determine that entropy^{tr} and entropy^{fpr} have a number of shortcomings, which are partly addressed by entropy^{fKL} , entropy^{fkNN} and entropy^{fbl} . However, all of these estimators are sensitive to the absolute size of logs, i.e. the length of traces and number of activities. All four seem to clearly misclassify one log in particular: the Road Traffic Fines Management log which receives a low entropy, but is characterised as a declarative process in the literature.

These measures are also strongly affected by concurrency, growing steadily with the size of the concurrent block,

regardless of the presence of strictly sequential prefixes and suffixes. One aspect on which these measures perform surprisingly well is in detecting noise, clearly reflecting for most logs in “Testing Representational Bias” event log the degree and type of noise added.

6.1.1 Block Entropy Measures

Global block entropy improves on prefix entropy because it is able to capture differences between traces, even if they share the same prefix. It gives measures which partially reflect our assumptions as to which logs are more or less structured: BPI Challenge 2012, Hospital Event Log and Sepsis Cases have markedly higher global block entropy than more structured logs such as BPI Challenge 2013. However, it clearly gives lower values to smaller logs with L_1 through L_4 all having drastically lower values than the larger logs. This is unsur-

Table 8 Traces of total length 9, with blocks of concurrency of size j in the middle

j = 1	j = 2
$\langle a, b, c, d, \mathbf{e}, f, g, h, i \rangle$	$\langle a, b, c, \mathbf{d}, \mathbf{e}, f, g, h, i \rangle$
$\langle a, b, c, \mathbf{e}, \mathbf{d}, f, g, h, i \rangle$	$\langle a, b, c, \mathbf{d}, \mathbf{f}, \mathbf{e}, g, h, i \rangle$
j = 3	j = 4
$\langle a, b, c, \mathbf{d}, \mathbf{e}, \mathbf{f}, g, h, i \rangle$	$\langle a, b, c, \mathbf{d}, \mathbf{e}, \mathbf{f}, g, h, i \rangle$
$\langle a, b, c, \mathbf{e}, \mathbf{d}, \mathbf{f}, g, h, i \rangle$	$\langle a, b, c, \mathbf{e}, \mathbf{d}, \mathbf{f}, g, h, i \rangle$
$\langle a, b, c, \mathbf{e}, \mathbf{f}, \mathbf{d}, g, h, i \rangle$	$\langle a, b, c, \mathbf{e}, \mathbf{f}, \mathbf{d}, g, h, i \rangle$
$\langle a, b, c, \mathbf{f}, \mathbf{d}, \mathbf{e}, g, h, i \rangle$	$\langle a, b, c, \mathbf{f}, \mathbf{d}, \mathbf{e}, g, h, i \rangle$
$\langle a, b, c, \mathbf{f}, \mathbf{e}, \mathbf{d}, g, h, i \rangle$	$\langle a, b, c, \mathbf{f}, \mathbf{e}, \mathbf{d}, g, h, i \rangle$
	$\langle a, b, \mathbf{d}, \mathbf{c}, \mathbf{e}, \mathbf{f}, g, h, i \rangle$
	\vdots

prising, since logs with longer traces will simply have many more possible k -blocks, flattening the probability distribution over outcomes and increasing the entropy.

6.1.2 Nearest Neighbour Measures

Nearest neighbour-based measures give similar results, relatively speaking, to global block entropy on many real logs, assigning higher values to BPI Challenge 2012 and 2017, and Hospital Event Log. However, they characterise Sepsis Cases as having a moderately low entropy.

Despite the nearest neighbour measures being applied to flattened logs, they are in general close to the original trace entropy. For those logs whose nearest neighbour entropy values deviate from trace entropy, the values are significantly lower, suggesting that edit distance really does account for the trace similarity ignored by the original trace entropy measure: the nearest neighbour measures group together very similar traces which are considered distinct by the original trace entropy.

We observe that entropy^{kNN} tends to decrease with k for real logs, and note that our results confirm Singh et al.'s empirical results showing that entropy^{kNN} closely matches entropy^{KL} when $k = 1$.

The nearest neighbour approach is largely unaffected by noise in the *Parallel* artificial log. This is not surprising since the added events will only increase the normalised edit distance to nearest neighbour slightly and may have no effect on traces with similar noise.

We emphasise that this measure depends crucially on the formulation of *distance* between traces, leaving ample room for improvement by, for example, developing more sophisticated edit operation cost weightings using domain knowledge or choosing to lessen the penalty to traces of very different lengths.

This approach suffers from one very clear shortcoming: complexity. In the worst case, the distance between every pair of traces must be computed, though unnecessary computations can be avoided by using a dynamic programming approach, checking lower bounds on edit distance, as well as common prefixes and suffixes. Despite employing these improvements along with a fast iterative implementation for computing *lev*, we found this measure to be prohibitively time-consuming for some of the large artificial logs (concurrent).

6.2 Entropy Rate Measures (h)

Entropy *rate* measures are more resilient to variations in the size of logs and number of activities, as demonstrated by the fact that L_1 through L_4 are assigned entropy rate estimates within the range of the (much larger) real-life logs. However, they are in general much more affected by noise: in some cases (see Fig. 9), ranking logs by entropy is not always stable under the addition of noise. It is encouraging that Lempel–Ziv and block-based entropy rate estimators using certain cut-off constraints return relatively consistent estimates despite being based on very different approaches. This suggests that they are in fact converging towards something near to the “true” entropy rate.

6.2.1 k -Block Estimators

We observed that the constraints for “good statistics” for block-based entropy rate estimators were in line with the properties discussed in 4.1.1. Namely, constraints (4.6) and (4.7), which are asymptotic upper bounds, allow k to grow much too large. The stricter constraints give much more reasonable entropy rate estimates, but also restrict k to extremely low values.

One important question we leave for future work concerns the appropriateness of the statistical assumptions underlying k -block estimators when applied to *sets* of sequences rather than single sequences. This means we get more samples of k -blocks than would be the case for a single sequence, and the exact number of samples depends on the distribution of trace lengths in the log: a log may consist of many very short traces and one very long trace, in which case longer blocks would still be under-sampled. In our implementation we chose to define the sequence length K as the longest trace, but a more principled approach should be considered.

Furthermore, some constraints are a function of the true entropy rate h itself, for which our implementation takes the current running estimate. Another approach may be to use another estimator for this, such as Lempel–Ziv.

In general, k -block entropy rate estimators appear to be rather unstable and very sensitive to the particular combination of constraint on k and log characteristics which is clearly

illustrated by the erratic fluctuations in Fig. 9. The ratio-based formulation rate_k^r converges more slowly (see Fig. 7), which is in line with previous research [23,35]. In this sense, it is more robust than rate_k^d . We note that the unusually high value assigned to some logs, such as NASA CEV, is due to the cut-off constraints restricting the estimator to the 1-block entropy.

Finally, we note that these estimators are surprisingly poor at detecting noise in artificial logs. Only rate_k^d using constraint (4.8) is able to consistently distinguish the degree and type of noise across the “Testing Representational Bias” set of logs. On the concurrent logs these estimators also give somewhat inconsistent results: certain constraints are so strict that the estimate plateaus immediately, with the relationship of concurrent block size to tail size apparently inverted, while other constraints lead to more reasonable estimates, which grow logarithmically with the size of concurrent block.

6.2.2 Lempel–Ziv

The Lempel–Ziv estimator has the advantage over block-based estimators that it is nonparametric in contrast to rate_k^d and rate_k^r which require choosing a cut-off constraint. It also appears to be much more robust, i.e. less erratic.

We observe that for some logs, however, it returns values contradicting our assumptions regarding modelling paradigms. In particular, BPI Challenge 2012 has a very low rate^{LZ} value, lower than BPI Challenge 2013: the opposite of what we would expect if declarative processes have higher entropy.

An important detail concerns the order in which the logs are parsed. Since the Lempel–Ziv algorithm parses sequences based on the order in which symbols are observed, parsing the traces in a different order can result in a different parsing and a slightly different value for rate^{LZ} . In our implementation we parsed logs in their original ordering. A sampling-based approach for assessing the variability of the estimates could be an avenue for future work.

The Lempel–Ziv estimator clearly outshines on the concurrent artificial log. It is surprisingly effective at capturing the fact that even large blocks of concurrency should have a low entropy rate when flanked by long sequential tails, while concurrent blocks with short, or no, sequential tails show a slowly growing entropy rate.

Lempel–Ziv performs less impressively when presented with noise and appears unable to distinguish types and degree of noise, with a number of the estimates “crossing over” for different logs in the presence of noise. For example, it assigns the *Duplicates* log a higher or lower entropy than the *Parallel* log depending on the particular type of noise present: a distinction other estimators are able to make.

On the logs generated from Declare models, Lempel–Ziv performs well, showing the expected drop on very restrictive

declarative model and again proving more stable than other entropy rate estimators. Finally, this estimator proved to be one of the fastest to compute.

7 Conclusion

We studied entropy as a measure of the variability of a process log, with the intended application of classifying logs as more suitable for declarative or imperative miners. Specifically, we contributed (1) a survey of potential measures of entropy; (2) an implementation of these measures; (3) an experimental investigation of the proposed measures on both synthetic and real-life logs; and (4) based on this investigation and the community understanding of which logs are likely declarative, a qualitative evaluation of the suitability of the measures.

A more rigorous, quantitative evaluation of the proposed measures requires a clear partitioning of logs into “imperative” or “declarative” classes. More precisely, we need clear ways of evaluating whether mining a log imperatively or declaratively produces “better” models: an open research question in itself. With a clearly defined error measure in hand, one or more entropy estimators could, for example, serve as input features to a classification algorithm to determine which mining approach to apply to a log, or in the case of hybrid mining, applied to partitions of the log in order to determine the mix of mining approaches.

Acknowledgements We would like to thank Jakob Grue Simonsen for valuable discussions.

References

1. Back CO, Debois S, Slaats T (2018) Towards an entropy-based analysis of log variability. In: Teniente E, Weidlich M (eds) Business process management workshops. Lecture notes in business information processing, vol 308. Springer, Cham, pp 53–70
2. van der Aalst WMP (1998) The application of Petri nets to workflow management. *J Circuits Syst Comput* 08:21–66. <https://doi.org/10.1142/S0218126698000043>
3. Object Management Group (2011) Business process modeling notation version 2.0. Technical report, Object Management Group Final Adopted Specification
4. Pesic M, Schonenberg H, van der Aalst W (2007) Declare: full support for loosely-structured processes. In: EDOC 2007, pp 287–300
5. Debois S, Hildebrandt T, Slaats T (2015) Safety, liveness and runtime refinement for modular process-aware information systems with dynamic sub processes. In: International symposium on formal methods. Springer, Berlin, pp 143–160
6. Hull R, Damaggio E, Masellis RD, Fournier F, Gupta M, Heath F, Hobson S, Linehan M, Maradugu S, Nigam A, Noi Sukaviriya P, Vaculín R (2011) Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In: DEBS 2011, pp 51–62

7. Debois S, Slaats T (2015) The analysis of a real life declarative process. In: CIDM 2015, pp 1374–1382
8. Reijers H, Slaats T, Stahl C (2013) Declarative modeling—an academic dream or the future for BPM? In: BPM 2013, pp 307–322
9. Slaats T, Schunselaar DMM, Maggi FM, Reijers HA (2016) The semantics of hybrid process models. In: CoopIS, pp 531–551
10. Maggi FM, Slaats T, Reijers HA (2014) The automated discovery of hybrid processes. In: Business process management—12th international conference, BPM 2014, Haifa, Israel, September 7–11, 2014. Proceedings, pp 392–399
11. Smedt JD, Weerd JD, Vanthienen J (2015) Fusion miner: process discovery for mixed-paradigm models. *Decis Support Syst* 77:123–136
12. Schunselaar DMM, Slaats T, Maggi FM, Reijers HA, van der Aalst WMP (2018) Mining hybrid business process models: a quest for better precision. In: Abramowicz W, Paschke A (eds) Business information systems. Springer, Cham, pp 190–205
13. Greco G, Guzzo A, Pontieri L, Sacca D (2006) Discovering expressive process models by clustering log traces. *IEEE Trans Knowl Data Eng* 18(8):1010–1027. <https://doi.org/10.1109/TKDE.2006.123>
14. Song M, Günther CW, Aalst WM (2009) Trace clustering in process mining. In: Business process management workshops. Springer, Berlin, pp 109–120
15. Makanju AA, Zincir-Heywood AN, Milios EE (2009) Clustering event logs using iterative partitioning. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining, KDD '09. ACM, New York, NY, pp 1255–1264. <https://doi.org/10.1145/1557019.1557154>
16. Bose RJC, van der Aalst WM (2009) Context aware trace clustering: towards improving process mining results. In: Proceedings of the 2009 SIAM international conference on data mining. SIAM, pp 401–412
17. Shannon CE (1948) A mathematical theory of communication. *Bell Syst Tech J* 27(3):379–423. <https://doi.org/10.1002/j.1538-7305.1948.tb01338.x>
18. Bishop CM (2006) Pattern recognition and machine learning. Springer, Berlin
19. Breuker D, Matzner M, Delfmann P, Becker J (2016) Comprehensive predictive models for business processes. *MIS Q* 40(4):1009–1034
20. van der Aalst WMP (2011) Process mining: discovery, conformance and enhancement of business processes. Springer, Berlin. <https://doi.org/10.1007/978-3-642-19345-3>
21. van der Aalst WMP, Adriansyah A, van Dongen BF (2012) Replaying history on process models for conformance checking and performance analysis. *Wiley Interdiscip Rev Data Min Knowl Discov* 2(2):182–192. <https://doi.org/10.1002/widm.1045>
22. Li M (2008) An introduction to Kolmogorov complexity and its applications, 3rd edn. Texts in computer science. Springer, New York
23. Schürmann T, Grassberger P (1996) Entropy estimation of symbol sequences. *Chaos Interdiscip J Nonlinear Sci* 6(3):414–427
24. Cover T, King R (1978) A convergent gambling estimate of the entropy of English. *IEEE Trans Inf Theory* 24(4):413–421
25. Greco G, Guzzo A, Pontieri L, Sacca D (2006) Discovering expressive process models by clustering log traces. *IEEE Trans Knowl Data Eng* 18(8):1010–1027
26. De Medeiros A, Guzzo A, Greco G, Van Der Aalst W, Weijters A, Van Dongen B, Saccà D (2008) Process mining based on clustering: a quest for precision, pp 17–29
27. Hofmann T (1999) Probabilistic latent semantic analysis. In: Proceedings of the fifteenth conference on uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc., Los Altos, pp 289–296
28. Delias P, Doumpos M, Grigoroudis E, Matsatsinis N (2017) A non-compensatory approach for trace clustering. *Int Trans Oper Res* 26:1828–1846
29. Ha QT, Bui HN, Nguyen TT (2016) A trace clustering solution based on using the distance graph model. In: International conference on computational collective intelligence. Springer, Berlin, pp 313–322
30. Singh S, Póczos B (2016) Analysis of k-nearest neighbor distances with application to entropy estimation. [arXiv:1603.08578](https://arxiv.org/abs/1603.08578)
31. Singh H, Misra N, Hnizdo V, Fedorowicz A, Demchuk E (2003) Nearest neighbor estimates of entropy. *Am J Math Manag Sci* 23(3–4):301–321
32. Delattre S, Fournier N (2017) On the Kozachenko–Leonenko entropy estimator. *J Stat Plan Inference* 185:69–93
33. Thomas JA, Cover TM (2006) Elements of information theory. Wiley, New York
34. MacKay DJC (2003) Information theory, inference and learning algorithms, 6. print edn. Cambridge University Press, Cambridge
35. Lesne A, Blanc JL, Pezard L (2009) Entropy estimation of very short symbolic sequences. *Phys Rev E* 79(4):046208
36. Ziv J, Lempel A (1977) A universal algorithm for sequential data compression. *IEEE Trans Inf Theory* 23(3):337–343
37. Ziv J, Lempel A (1978) Compression of individual sequences via variable-rate coding. *IEEE Trans Inf Theory* 24(5):530–536
38. Back CO Eventropy—entropy estimation tool and CLI for XES event logs and other sequential data. <https://github.com/backco/eventropy>. Accessed 30 Apr 2019
39. Real life event logs. 4TU Centre for Research Data. https://data.4tu.nl/repository/collection:event_logs_real. Accessed 23 Jan 2018
40. Mannhardt F, Blinde D (2017) Analyzing the trajectories of patients with sepsis using process mining. In: RADAR+ EMISA, vol 1859, pp 72–80
41. Maggi F, Slaats T, Reijers H (2014) The automated discovery of hybrid processes. In: BPM, pp 392–399
42. Van Der Aalst WW (2017) Testing representational biases. <https://doi.org/10.4121/uuid:25d6eef5-c427-42b5-ab38-5e512cca08a9>
43. Di Ciccio C, Bernardi ML, Cimitile M, Maggi FM (2015) Generating event logs through the simulation of declare models. In: Workshop on enterprise and organizational modeling and simulation. Springer, Berlin, pp 20–36
44. Di Ciccio C, Mecella M (2015) On the discovery of declarative control flows for artful processes. *ACM Trans Manag Inf Syst* 5(4):24:1–24:37. <https://doi.org/10.1145/2629447>
45. Buijs J, Dongen B, Aalst W (2012) On the role of fitness, precision, generalization and simplicity in process discovery. In: On the move to meaningful internet systems: OTM 2012, vol 7565. Springer, Berlin, pp 305–322. https://doi.org/10.1007/978-3-642-33606-5_19. <http://www.wis.win.tue.nl/~wvdaalst/publications/p688.pdf>
46. Back CO, Debois S, Slaats T (2018) Towards an empirical evaluation of imperative and declarative process mining. In: International conference on conceptual modeling. Springer, Cham, pp 191–198

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Towards an Empirical Evaluation of Imperative and Declarative Process Mining

Christoffer Olling Back¹, Søren Debois², and Tijs Slaats¹ *

¹ Department of Computer Science, University of Copenhagen
Emil Holms Kanal 6, 2300 Copenhagen S, Denmark
{back, slaats}@di.ku.dk

² Department of Computer Science, IT University of Copenhagen
Rued Langgaards Vej 7, 2300 Copenhagen S, Denmark
{debois}@itu.dk

Abstract. Process modelling notations fall in two broad categories: declarative notations, which specify the *rules* governing a process; and imperative notations, which specify the *flows* admitted by a process. We outline an empirical approach to addressing the question of whether certain process logs are better suited for mining to imperative than declarative notations. We plan to attack this question by applying a flagship imperative and declarative miner to a standard collection of process logs, then evaluate the quality of the output models wrt. the standard model metrics of precision and generalisation. This approach requires perfect fitness of the output model, which substantially narrows the field of available miners; possible candidates include Inductive Miner and Minerful. With the metrics in hand, we propose to statistically evaluate the hypotheses that (1) one miner consistently outperforms the other on one of the metrics, and (2) there exist subsets of logs more suitable for imperative respectively declarative mining.

Keywords: Process Mining, Modelling Paradigms, Statistical Evaluation, Declarative Models, Imperative Models, Hybrid Models, Evaluation Metrics

1 Introduction

Workflow notations are commonly categorised as falling within either the *imperative* or *declarative* paradigm [16]. Imperative notations use flow-based constructs to explicitly model the *paths* through a process [1]. Declarative notations use constraint-based constructs to model the *rules* of a process. A declarative model allows all paths not forbidden by the constraints, and therefore the behaviour of the model is implicit in the rules and needs to be deduced by the system or users [7, 9, 20]. While the imperative paradigm is more mature, both paradigms have seen industrial adoption [13–15].

Regardless of notation, models have to come from somewhere. A recent trend in both academia and industry is to extract models from real-life data via *process discovery* [19], where an *output model* is automatically constructed from an *event log* of

* This work is supported by the Hybrid Business Process Management Technologies project (DFF-6111-00337) funded by the Danish Council for Independent Research, and the EcoKnow project (7050-00034A) funded by the Innovation Foundation.

observed process executions. Research into this approach has focused primarily on the discovery of imperative models, but substantial energy has been directed towards algorithms that discover declarative models as well [6, 8, 11].

Thus, when we construct process models by process discovery, we have a choice: Which paradigm should we use? Would we get better models from one than the other? Would such a difference be universal or would it depend on the particular input log? This paper outlines an approach to empirically evaluating the ramifications of the choice of paradigm on output model quality [4, 5]. We specifically propose using notation-agnostic metrics for *precision* and *generalisation* from [21], which apply equally to imperative and declarative models.

We propose using Inductive Miner [10] and MINERful [6] as representatives of the imperative and declarative paradigms, respectively. We explain this choice in detail in Section 2. In short, fitness and simplicity are held constant: we require discovered models to have perfect fitness and comparable simplicity. This limits our choice of miners, as at the time of writing the Inductive Miner is the only imperative miner guaranteeing perfect fitness [10], and MINERful is the only declarative miner that can be easily tuned to produce models of a given simplicity. Fortunately, the Inductive Miner and MINERful are widely considered to be at the cutting edge of their respective fields. Consequently, they make for reasonable representatives of their respective paradigms.

By evaluating these miners on the largest set of real-life logs available to us, we aim to test the following hypotheses:

Hypothesis 1: One miner consistently outperforms the other on one of the metrics:

- (1a) one miner outperforms the other on precision.
- (1b) one miner outperforms the other on generalisation.

Hypothesis 2: There exist subsets of logs:

- (2a) more suitable for imperative mining
- (2b) more suitable for declarative mining

That is, there exists a subset of logs which when mined either declaratively or imperatively represent a Pareto improvement over the other; *and* this deviation from the zero mean lies outside of the bounds of what can be accounted for by random chance.

A Pareto improvement simply denotes an improvement on at least one metric without sacrificing performance on any remaining metrics. The zero mean is the mean of the probability distribution associated with the null hypothesis, and represents no performance difference between models produced by different miners from the same log.

2 Methods

2.1 Log Selection

We begin by selecting a representative sample of test data, including publicly available real-life event logs in addition to one log from an industrial contact. Furthermore, we evaluate synthetic logs generated from either imperative or declarative models. The

synthetic models should comprise examples of typical control constructs in the given language. For imperative models we plan to use an existing testbed developed to test representational bias in mining algorithms, and containing a number of typical pitfalls³.

2.2 Process Discovery

We will mine the selected logs both imperatively and declaratively, selecting miners according to the following criteria:

1. Miners must be configurable to always produce perfectly fitting models.
2. Miners must be configurable to produce models of a given simplicity, save one which can serve as a benchmark.

The first criterion follows from both precision and generalisation requiring perfect fitness of the output model. It would have been an option to allow non-fitting output, and then use a model-log alignment [2, 21], but without domain knowledge or access to an expert, we cannot know which exact alignment is more appropriate for the real-world log. This means that we would be evaluating not just the mining algorithm, but the combination of mining algorithm and alignment function. In particular, we would not know whether to attribute a result in favour of one miner over the other to the miner itself, or to a fortunate choice of alignment for that particular miner.

The second criterion follows partly from a tendency of declarative miners to produce output models containing excessive numbers of constraints: for large logs, on the order of *hundreds of thousands*. More importantly, we require model simplicity to be held constant, so that the choice of mining algorithm remains the only independent variable.

The two criteria left us only two miners: The Inductive Miner and MINERful.

The Inductive Miner is an imperative miner developed by Leemans et al. [10]. Arguably the premiere imperative miner in the field, it uses a divide-and-conquer approach to generate block-structured process models output as process trees or Petri nets. With only one parameter, noise threshold, which for our purposes will be held at 1.0, ensuring perfect fitness, the model generated by Inductive Miner provides a baseline model from which to set a threshold on model simplicity for MINERful.

MINERful is a *declarative* miner developed by Di Ciccio et al. [6]. It uses a two-phase approach: in the first phase, a knowledge base of statistical information on the log is built; in the second, this knowledge is queried in order to infer the constraints of the process. The output is a Declare model, possibly including negative constraints.

MINERful has a configurable *support threshold*, an *interest factor threshold*, and a *confidence threshold*. By iteratively adjusting these settings until a model is found which has the highest possible number of constraints without exceeding complexity of the imperative model, we ensure that the imperative and declarative models are of comparable simplicity. We note that while many measures of simplicity have been proposed for imperative models, there exists no widely accepted method for comparing the

³ Retrieved from: <https://doi.org/10.4121/uuid:25d6eef5-c427-42b5-ab38-5e512cca08a9>

simplicity of imperative and declarative models. We suggest that an initial investigation begin by simply comparing the number of edge elements: transitions vs. constraints.

2.3 Computing Metrics

Defining standard measures for precision and generalisation remains an open research challenge for two main reasons. First, in process mining, data is generally not assumed to be labelled, i.e. event logs contain examples of what *did* happen, not what should *not* happen. This means that the standard definition of precision used in data mining and statistics, cannot be applied to process discovery, since it relies on defining true and false positives (type I error), and true and false negatives (type II error). Second, the prevalence of unbounded loops in process models means that they often describe an infinite set of allowed behaviour. Therefore, definitions of precision and generalisation which take into account all of the of behaviour allowed by the model are not applicable in practice. Instead most metrics aim to reduce the measured behaviour of the model to a finite set of traces.

Metric Selection To compare imperative and declarative models, we require metrics that can be applied to both equally. This means that they need to be defined on either the level of languages or transition systems. Accordingly, we have chosen to employ the metrics introduced in [21], in particular:

Precision [21, p.10] measures the degree to which a model is “underfitting” or “allowing too much behaviour” relative to the input log. This particular metric is based on the notion of *escaping edges*, which represent a point at which the model allows behaviour not seen in the log. The measured amount of additional behaviour is kept finite by only considering the first divergent activity. I.e. an escaping edge may lead to a loop representing an infinite set of traces that did not occur in the log, but only the trace ending with the first divergent activity will be counted.

Generalisation [21, p.11] , on the other hand, measures the degree to which a model is “overfitting”: is there behaviour not allowed by the model and not exhibited in the log, but that can be reasonably expected to occur in the future? This particular metric approximates generalisation by estimating for each state in the model the likelihood that a new, hitherto unseen, activity will occur. This estimation is based on the number of activities that have been observed, and how often the state was visited. Two alternatives are offered: *event-based generalisation* takes into account the number of visits to a state, *state-based generalisation* does not.

Implementation Although ProM contains a plugin for computing the metrics of [21] on Petri nets, it does not offer support for declarative models. We also find it more convenient to run our tests as a batch-process where we can easily pipeline several operations (mining, metrics computation, analysis) on a set of multiple logs. Therefore we developed our own evaluation framework⁴. The framework also makes our methods

⁴ Available at: <https://bitbucket.org/coback/qmpm>

and results straightforward to reproduce. One can inspect the code and run it on our, or their own input data. We have extensively verified our framework, in particular by testing it on the examples and results reported in [21].

Several challenges arise when computing precision and generalisation, mainly regarding time and space efficiency, but also proper handling of nondeterminism arising from silent transitions present in models produced by Inductive Miner. When assessing which activities are enabled in a given marking, a greedy algorithm will naively follow silent transitions until it encounters the first non-silent transition, potentially firing silent transitions underway. This risks associating incorrect markings with an event when the subsequent enabled activity is replayed, potentially excluding activities which should be enabled, leading to a failure to replay the log. Ensuring that the shortest path to a non-silent path is taken prevents this problem from arising.

To minimize redundancy, our framework builds a prefix tree from the event log, replaying each trace on the given model as it is added to the prefix trie. In each node (corresponding to an event in the log), the state of the model is saved, unless the node has been visited previously, in which case a counter associated with the node is incremented, recording the number of occurrences of that prefix. Finally, a map containing model states, i.e. markings, as keys and sets of nodes (events) as values, is maintained in order to facilitate the calculation of state-based generalisation. Given an event, the “enabled” activities in the log simply correspond to that node’s children, while the enabled activities in the model are obtained by querying the model using the state associated with that node. This approach minimizes redundancies and handles potential state-space explosion by computing and storing only relevant information.

3 Conclusion

We outline an approach to systematically compare the performance of imperative versus declarative process mining algorithms based on the commonly accepted quality metrics for precision and generalisation defined by [21]. We will investigate two hypotheses: first, that one miner performs better on a) precision and/or b) generalisation; second, that there exist some logs on which either miner provides a statistically significant Pareto improvement on both precision and generalisation.

To the best of our knowledge, this will be the most exhaustive study comparing imperative and declarative process discovery techniques to date. Future evaluations incorporating other aspects of the process mining life-cycle, e.g. alignment, will have this approach as a point of reference. Not least, we contribute a comprehensive software framework and tackle a number of methodological and implementation challenges, providing a foundation upon which further work can build.

Finally, we believe that the proposed study will be extremely valuable to the field of hybrid process mining [12, 17, 18], which aims to combine the strengths of the two paradigms. Ongoing research into what characteristics make a log, or sub-log, more suitable to one paradigm over the other are hampered by the fact that there exists no clear evidence from which to compare how the traditional experimental logs perform under each paradigm [3]. Providing such evidence will enable finding new characteristics of the logs, and fully evaluating previously proposed approaches.

References

1. Wil M. P. van der Aalst. Verification of workflow nets. ICATPN '97, pages 407–426, 1997.
2. Arya Adriansyah, Jorge Munoz-Gama, Josep Carmona, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Alignment Based Precision Checking. In *BPM Workshops*, Lecture Notes in Business Information Processing, pages 137–149, 2012.
3. Christoffer Olling Back, Søren Debois, and Tijs Slaats. Towards an Entropy-based Analysis of Log Variability. In *Lecture Notes in Business Information Processing*, 2017.
4. J. C. a. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst. Quality Dimensions in Process Discovery: The Importance of Fitness, Precision, Generalization and Simplicity. *International Journal of Cooperative Information Systems*, 23(01):1440001, March 2014.
5. Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. In *OTM 2012*, Lecture Notes in Computer Science, pages 305–322, 2012.
6. Claudio Di Ciccio and Massimo Mecella. On the discovery of declarative control flows for artful processes. *ACM Transactions on Management Information Systems*, 5(4):24, 2015.
7. Søren Debois, Thomas T. Hildebrandt, and Tijs Slaats. Replication, refinement & reachability: complexity in dynamic condition-response graphs. *Acta Informatica*, Sep 2017.
8. Søren Debois, Thomas T. Hildebrandt, Paw Høvsgaard Laursen, and Kenneth Ry Ulrik. Declarative Process Mining for DCR Graphs. SAC '17, pages 759–764, 2017.
9. R. Hull, E. Damaggio, R. De Masellis, F. Fournier, M. Gupta, F.T. Heath, S. Hobson, M.H. Linehan, S. Maradugu, A. Nigam, P. Noi Sukaviriya, and R. Vaculín. Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In *DEBS 2011*, pages 51–62, 2011.
10. Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. In *Application and Theory of Petri Nets and Concurrency*, pages 311–329, June 2013.
11. Fabrizio Maria Maggi, R. P. Jagadeesh Chandra Bose, and Wil M. P. van der Aalst. Efficient discovery of understandable declarative process models from event logs. In *CAiSE*, pages 270–285, 2012.
12. F.M. Maggi, T. Slaats, and H.A. Reijers. The automated discovery of hybrid processes. In *BPM*, pages 392–399, 2014.
13. M. Marquard, M. Shahzad, and T. Slaats. Web-based modelling and collaborative simulation of declarative processes. In *BPM 2015*, pages 209–225, 2015.
14. Object Management Group. Business Process Modeling Notation Version 2.0. Technical report, Object Management Group Final Adopted Specification, 2011.
15. Object Management Group. Case Management Model and Notation, version 1.0. Webpage, may 2014. <http://www.omg.org/spec/CMMN/1.0/PDF>.
16. Hajo A. Reijers, Tijs Slaats, and Christian Stahl. Declarative modeling—an academic dream or the future for bpm? In *Business Process Management*, pages 307–322, 2013.
17. Dennis Schunselaar, Tijs Slaats, Fabrizio Maggi, Hajo A. Reijers, and Wil M. P. Aalst. Mining hybrid business process models: A quest for better precision. *BIS*, pages 190–205, 2018.
18. Tijs Slaats, Dennis M. M. Schunselaar, Fabrizio Maria Maggi, and Hajo A. Reijers. The semantics of hybrid process models. In *CoopIS 2016*, pages 531–551, 2016.
19. Wil van der Aalst. *Process Mining*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.
20. Wil van der Aalst, Maja Pesic, Helen Schonenberg, Michael Westergaard, and Fabrizio M. Maggi. Declare. Webpage, 2010. <http://www.win.tue.nl/declare/>.
21. Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisc. Rev.: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.

Imperative or Declarative?

An Empirical Comparison of Event Logs

Christoffer Olling Back¹, Søren Debois², and Tijs Slaats¹ *

¹ Department of Computer Science, University of Copenhagen
Emil Holms Kanal 6, 2300 Copenhagen S, Denmark
{back, slaats}@di.ku.dk

² Department of Computer Science, IT University of Copenhagen
Rued Langgaards Vej 7, 2300 Copenhagen S, Denmark
{debois}@itu.dk

Abstract. Process modelling notations fall in two broad categories: declarative notations, which specify the *rules* governing a process; and imperative notations, which specify the *flows* admitted by a process. Process discovery algorithms can similarly be categorised on the notation of their output model. This situation begets the question: Given a log for process mining, which paradigm would produce the better output model? In this paper we study this question whether some process logs are better suited for imperative or declarative notations. We make the following contributions: (1) we provide a methodology for comparing imperative and declarative models on an equal footing based on notation-agnostic formulations of precision and generalisation; (2) we compare the results of flagship imperative and declarative miner on a standard collection of process logs; (3) we evaluate statistically the hypothesis that one miner consistently outperforms the other on one of the metrics, finding no significant difference—neither miner in general performs better than the other on neither generalisation nor precision; and (4) we identify several logs that are Pareto-optimal for either the declarative or imperative paradigm—logs that are truly “better” for one paradigm than the other.

Keywords: Process Mining, Modelling Paradigms, Statistical Evaluation, Declarative Models, Imperative Models, Hybrid Models, Evaluation Metrics

1 Introduction

Workflow notations are commonly categorised as either *imperative* or *declarative* [25]. Imperative notations use flow-based constructs to explicitly model the *paths* through a process [1]. Declarative notations use constraint-based constructs to model the *rules* of a process. A declarative model allows all paths not forbidden by the constraints, so the behaviour of the model is implicit in the rules and needs to be deduced by the system or

* This work is supported by the Hybrid Business Process Management Technologies project (DFF-6111-00337) funded by the Danish Council for Independent Research, and the EcoKnow project (7050-00034A) funded by the Innovation Foundation.

users [13, 16, 31]. While the imperative paradigm is more mature, both paradigms have seen industrial adoption [20, 23, 24].

Regardless of notation, models have to come from somewhere. A recent trend in both academia and industry is to extract models from real-life data via *process discovery* [30], where an *output model* is automatically constructed from an *event log* of observed process executions. Research into this approach has focused primarily on the discovery of imperative models, but substantial energy has also been directed towards algorithms that discover declarative models [10, 14, 18].

In work on declarative discovery, algorithms are usually tested on a limited subset of publicly available logs, usually motivated by the expectation that logs from certain domains—health-care, public governance—would be more suitable for declarative notations. Until now, this intuition has not been rigorously tested: imperative and declarative notations follow inherently different paradigms and are therefore difficult to compare on equal footing.

Contributions. In this paper we address this challenge of empirically comparing declarative and imperative miner outputs, answering the following three research questions:

- RQ1 How do we compare relative model quality of imperative and declarative output models?
- RQ2 Does one paradigm consistently outperform the other on this comparison?
- RQ3 Which are the logs that lend themselves favourably to mining with one paradigm or the other?

Methodology: For RQ1, we propose quantifying cross-paradigm quality differences by measuring the *trade-offs* in the output model metrics: *precision* and *generalisation*. That is, how the output model balances on the one hand closely modelling the log (precision) and on the other allowing a reasonable amount of additional behaviour (generalisation). This duality makes them well-suited for studying trade-offs in model quality: if switching mining paradigm improves on one without sacrificing on the other, the model has clearly improved.

The challenge then is to find realisations of the precision and generalisation metrics that apply to both imperative and declarative notations. For *precision*, the definition in [33] based on “escaping edges” in the transition system generated by a model is both widely accepted (however, do note [29]), notation agnostic and efficiently computable. For *generalisation* we take a straightforward data mining approach originally proposed for process mining in [33] and pursued recently in [6]: *k*-fold cross validation. To measure the ability of a miner to generalise a given log, we reserve $1/k$ of the log for post-mining validation; the ability of the miner to generalise that log is then the proportion of traces in that validation set that the mined model recognises.

With a method for comparing output models in place, we proceed to consider RQ2 and 3. For these, we apply flagship imperative and declarative miners to all relevant publicly available process mining logs, then (RQ2) testing statistically (Wilcoxon) whether one miner significantly outperforms the other in general; and analysing the results to determine which logs are better for each miner (weak and strong Pareto optimality).

We note two major challenges in conducting this experiment. First, the chosen measure of precision requires perfect fitness. Since we cannot use alignment [3] without

muddying our results, we must require output models to have perfect fitness outright. It follows that we are either assuming noise-free logs or accepting that output models represent also noise. Second, we noticed early on that Declarative miners achieve their maximum precision by producing models with that are order of magnitude larger than the input logs. Such models seems practically unhelpful, so to achieve a more sensible comparison we tune parameters to generate output models of size roughly comparable to their imperative counterparts. Together, these restrictions limit our choice of miners: at the time of writing the Inductive Miner is the only imperative miner guaranteeing perfect fitness [17] and MINERful [10] the only declarative miner that can be tuned to produce models of a given size bound. Fortunately, both are considered to be at the cutting edge of their respective fields and can reasonably be taken to represent their respective paradigms.

Results For RQ1, we propose studying the tradeoff between precision (escaping-edges) and generalisation (k -fold cross validation) as the yardstick on which to measure relative output model quality of imperative and declarative miners. For RQ2, we show statistically that no miner categorically outperforms the other on neither precision or generalisation. For RQ3, we confirm the common intuition in the field that the logs Nasa Crew Exploration Vehicle is better suited for imperative discovery, and the logs BPI Challenge 2012 and 2013 for declarative discovery; where “better suited” does not strictly indicate a Pareto improvement, but an improvement in one metric that significantly overshadows any decrease in the other.

We note that the logs identified as better suited for declarative mining in RQ3 encompasses logs considered “declarative” in the literature. Remarkably *no log* is better in one paradigm on both parameters³. Altogether, we take this as indicative that the two paradigms are complementary more than they are competing.

2 Related Work

The classic *output model quality* measures of fitness, precision, generalisation and simplicity [8, 9] are well-studied, e.g., [26, 30, 33]. The variants of precision and generalisation used in the present paper originate with [2, 8, 9], as a basis for evaluating alignments for conformance checking. We employ the prefix automata of [21] to avoid state-space enumeration; prefix automata themselves arise as a certain choice of parameters for the framework of [34]. Incidentally, precision and generalisation were originally formalised for Process Trees or Petri Nets, and thus did not originally apply to other models. It is precisely the later re-formulations in terms of prefix automata [21] and labelled transition systems in [33] that makes it possible to make the present comparison of imperative and declarative output model quality.

Precision and generalisation both require *perfect fitness*. There are two ways to achieve that: through cost-based alignment between log and model [4, 33] or replay techniques, e.g., [26]; or by requiring outright that the miner produces only perfectly

³ We encourage the impatient reader to skip directly ahead to Figures 2 and 1—note the *empty* second and fourth quadrants in the former!

fitting models; a feature supported (to the best of our knowledge) only by the Inductive Miner, MINERful, the Declare Maps miner [18], the DCR miner [14], and DISCO.

Several large studies tabulating miner performance exist, e.g., [5, 11]. Smaller studies comparing individual miners also abound, e.g., [10] compares the declarative MINERful and Declare Maps miner, concluding that MINERful is much faster but sensitive to parameter setting; and [12] in which imperative process discovery is applied to a process log originating from a declarative real-life process.

3 A Method for Cross-paradigm Comparison

We proceed as follows: (1) select test data (logs), (2) run imperative and declarative miners on these logs, (3) compute precision and generalisation metrics on the output models, (4) evaluate statistically the hypotheses below, as well as (5) analyse Pareto trade-offs on (3).

Hypothesis: One miner consistently outperforms the other on one of the metrics:

- (1a) one miner outperforms the other on precision.
- (1b) one miner outperforms the other on generalisation.

3.1 Log Selection

We base log selection on the criteria of public availability, drawing upon (i) the IEEE Task Force on Process Mining Real-life Event Log Collection⁴, (ii) two additional logs also published by the 4TU Center for Research Data^{5,6}, and (iii) one real-life log originating from our own industrial contacts [12]. We abstain from using synthetic logs, for which we would not be able to tell whether an observed difference in output model quality reflects a genuine difference in miner performance or bias in the generator.

The logs range over a broad range of industries and applications, including knowledge-oriented administrative processes, human behaviour tracking, manufacturing processes, and machine-generated software logs. See Table 1 for included logs, Table 2 for logs omitted for not fulfilling the following:

1. The logs must come in a supported file format. (E.g., we omitted CSV files that did not come with clear instructions on which columns represented event classes.)
2. Process discovery for the log must terminate and output a model for both the imperative and declarative mining algorithms.
3. The log must be an *event* log [32]. (E.g., we omitted “logs” that did not represent a sequence of events.)

We did not perform pre-processing except for associating lifecycle transition attributes with events using Niek Tax’ ProM plugin “Bring Lifecycle to Event Name”.

⁴ http://data.4tu.nl/repository/collection:event_logs_real

⁵ <https://doi.org/10.4121/uuid:5a9039b8-794a-4ccd-a5ef-4671f0a258a4>

⁶ <https://doi.org/10.4121/uuid:5a9039b8-794a-4ccd-a5ef-4671f0a258a4>

Log	Activities	Total Traces	Unique Traces (%)	Shortest Trace	Longest Trace	Domain
Activities of Daily Living	64	148	100.0	20	134	<i>Sensor data</i>
BPI Challenge 2012	36	13087	33.36	3	175	<i>Loan application</i>
BPI Challenge 2013	13	7554	30.16	1	123	<i>VolvoIT incidents</i>
BPI Challenge 2017	66	31509	52.98	10	180	<i>Loan application</i>
BPI Challenge 2018	41	43809	64.97	24	2973	<i>EU agr. grant app.</i>
Document Processing	18	18352	0.002	20	6	<i>Doc. processing</i>
Dreyer Foundation	31	255	15.15	1	25	<i>Grant application</i>
Electronic Invoicing	16	20135	0.002	8	20	<i>Billing/invoicing</i>
Hospital Billing	18	100000	1.02	1	217	<i>Billing/invoicing</i>
NASA Crew Explor. Veh.	94	2566	97.93	12	50	<i>Software log</i>
Production Analysis	110	225	98.22	2	350	<i>Manufacturing</i>
Road Traffic Fine Mgmt	11	150370	0.15	2	20	<i>Fine processing</i>
Sepsis Cases	16	1050	80.57	3	185	<i>Healthcare</i>
WABO/CoSeLoG - receipt	27	1434	8.09	1	25	<i>Environ. permit</i>

Table 1: Attributes of evaluated logs.

Log	Reason for Omission
Apache Commons Crypto Tests	<i>Out of memory</i>
BPI Challenge 2014	<i>Not well-formed event log</i>
BPI Challenge 2016	<i>Not well-formed event log</i>
Hospital Log	<i>MINERful out of memory</i>
Interactions w/ Lighting Interfaces	<i>Not well-formed event log</i>
JUnit 4.12 Software Event Log	<i>Both miners out of memory</i>
WABO/CoSeLoG 1-5	<i>Error reading logs</i>

Table 2: Overview of omitted logs and reasons for omission.

3.2 Process Discovery

We mine the selected logs both imperatively and declaratively. We assume that miners produce useful models in their default configuration and use them in that configuration, except as noted below.

Unfortunately, declarative miners sometimes output models with excessive numbers of constraints: for large logs, hundreds of thousands. It is difficult to imagine a use-case for process discovery where such models are helpful, much less desirable. Therefore, we modify miner configuration to force models to have a maximum size.

As we are unaware of published results that would help us define a reasonable maximum model output size, we instead take an established miner as *defining* an appropriate model output size, with the argument that the community has tacitly accepted the models produced by this miner as “of at most useful sizes”. We define model size by the number of edges in the model: edges between places and transitions in Petri nets produced by Imperative Miner and constraints between activities in the Declare model produced by MINERful. As Inductive Miner is currently accepted as *the* successful im-

perative miner, it is a natural candidate to define maximum output sizes, and we limit the declarative output for a given log to be no larger than the Inductive Miner output model size. We leave a more principled definition of appropriate maximum declarative model sizes, preferably grounded in usability studies, as future work.

Besides output model size, we must further restrict our attention to miners that can be instructed to always produce perfectly fitting models. This criterion follows from both precision and generalisation requiring perfect fitness of the output model. It would have been an option to allow non-fitting output, and then use a model-log alignment [2, 33], but without domain knowledge or access to an expert, we cannot know which exact alignment is more appropriate for the real-world log. This means that we would be evaluating not just the mining algorithm, but the combination of mining algorithm and alignment function: we would not know whether a result in favour of one miner over the other to was a real advantage or just a fortunate choice of alignment.

Altogether, we have chosen miners according to the following criteria: 1. Miners must produce perfectly fitting models. 2. Miners must produce models of a given maximum size, save one which can serve as a benchmark. These criteria leave us only two miners: The Inductive Miner and MINERful.

The Inductive Miner [17]. Arguably the premiere imperative miner, Inductive Miner uses a divide-and-conquer approach to generate block-structured process models output as process trees or Petri nets. We use version 6.8.415⁷ with stock settings, except noise threshold. We run Inductive Miner outside the ProM environment, accessing it programmatically via its Java interface. To ensure stock settings, we initialise the miner with an unmodified `MiningParametersIM()` object; we have confirmed that these settings generates models identical to those output by the Inductive Miner plugin in ProM 6.7.

MINERful [10] is a declarative miner developed by Di Ciccio et al. It uses a two-phase approach: in the first phase, a knowledge base of statistical information on the log is built; in the second, this knowledge is queried in order to infer the constraints of the process. The output is a Declare model, possibly including negative constraints.

Unfortunately, MINERful does not allow directly setting a maximum model size, rather, it allows setting *support threshold*, an *interest factor threshold*, and a *confidence threshold*. By iteratively adjusting these until a model not exceeding the size bound obtained from the imperative model is found, we ensure that the imperative and declarative models are of comparable size. Specifically, confidence is lowered by 0.05, and interest factor is adjusted at each confidence threshold using binary search, also using a resolution of 0.05. Before the size of the mined model is evaluated, inconsistency checking is applied⁸. Redundancy checking does not finish for all logs and therefore is not applied.

⁷ Retrieved from: <http://promtools.org/prom6/packages/InductiveMiner/>

⁸ HIERARCHYCONFLICT, default constraint sorting policy: ACTIVATIONTARGETBONDS, FAMILYHIERARCHY, SUPPORTCONFIDENCEINTERESTFACTOR.

3.3 Quality Metrics

In process mining, four quality metrics of models are generally considered: fitness, simplicity, precision and generalisation [32]. Presently, we focus on the trade-off between precision and generalisation: how precisely models generated by a miner describe the observed data and how well they are able to account for previously unseen data.

Precision Most precision metrics are defined on Petri nets, precluding a direct comparison to declarative models. Metrics based either on the level of languages or transition systems allow for a cross-paradigm comparison. One such metric was proposed in [33, p.10], and implemented in [6]. It is based on *escaping edges*, which represent a point at which the model allows behaviour not seen in the log. The measured amount of additional behaviour is kept finite by only considering the first divergent activity.

Let \mathcal{E} denote the set of unique events in an event log with $e \in \mathcal{E}$. Let $\text{en}_L(e)$ denote the set of activities sharing the same context (i.e., prefix) as event e in log L . Let $\text{en}_M(e)$ denote the set of activities enabled in the state of model M immediately *prior* to executing e . The precision of M w.r.t. log L is given by

$$P_L(M) = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \frac{|\text{en}_L(e)|}{|\text{en}_M(e)|} \quad (1)$$

Normalised Precision The above formulation of precision is a function of the log as well as the model. We would like to establish a *normalised* formulation by establishing a lower and upper bound for the log, giving an indication of the relative difference between miners based on worst and best case scenarios for a particular log [22].

Let P_L^u and P_L^l denote the lower and upper bound, respectively, for the precision of any model w.r.t. log L . The normalised precision w.r.t. log L of model M is given by

$$P_L^n(M) = \frac{P_L(M) - P_L^l}{1 - P_L^l} \quad (2)$$

We take as P_L^l the precision of the *flower* model, in which any activity encountered in the log is enabled in every context. The only models resulting in a lower estimate of P_L^l are those which allow *more activities than encountered in the log* to be enabled. The 1 in the denominator represents an upper limit on the precision of a fitting model w.r.t. log L , in this case a model which allows exactly the traces in the log and no others.

Generalisation (Cross-validation) Process mining is generally framed as a *descriptive* data mining task: the aim is to accurately describe training data [15], and quality metrics such as precision and fitness computed on the *in-sample* data set are most often reported. Unless we assume that an event log represents all of the behaviour we can expect to encounter, we should also investigate the expected performance on *out-of-sample* data. Cross-validation is one approach to making this estimation in which a portion of the data set is held out as a validation set, while the remaining data is used for training (mining). Traditional supervised learning tasks define an error metric, e.g. the number of correctly classified instances in classification tasks, or a distance-based metric, such as the residual sum of squares, for regression tasks.

We take as our error metric simple trace fitness: the percentage of out-of-sample traces able to be replayed on the model. If we interpret process mining as a binary classification problem (in which traces either fall into acceptable or unacceptable classes), with event logs containing only positive examples, then fitness can be interpreted as *recall*, i.e. the ratio of traces accepted by the model to the total number of traces in the validation log:

$$F_L(M) = \frac{|\text{true positives}|}{|\text{true positives}| + |\text{false negatives}|} \quad (3)$$

We employ a form of k -fold cross-validation which avoids “twinning” (including identical data points in both training and validation sets). This is done by partitioning the set of *unique* traces in a log into k equal partitions. Then, the trace multiplicities are reinserted, resulting in unevenly sized partitions. This is ameliorated by using a weighted mean of the performance across folds, weighting the performance on a given validation set by the size of the that set relative to the entire log.

Formally, letting L denote a log, L_i the i th partition of the log, k the number of partitions, $F_{L_i}(M_{L_j})$ fitness w.r.t. to log L_i of model M mined on log L_j , the weighted-mean, fitness-based, k -fold cross-validation for mining algorithm A is given by

$$G_L(A) = \sum_{i=1}^k \frac{|L_i|}{|L|} F_{L_i}(M_{L \setminus L_i}) \quad (4)$$

3.4 Implementation

To our knowledge, no implementation exists for computing the metrics defined in (2) and (4) based on XES log files and both imperative and declarative process models. We have therefore integrated this functionality to our CLI based evaluation framework⁹.

All mining and evaluation processes were performed on a Lenovo Thinkpad P50 with a 64-bit Intel Xeon E3-1535M v3 2.90GHz CPU and 32GB of RAM running Windows 10 Enterprise 64-bit edition.

⁹ Available at: <https://github.com/backco/qmpm>

Log	INDUCTIVE MINER					MINERFUL				$P_L^n(I) - P_L^n(M)$	$G_L(I) - G_L(M)$
	$P_L^n(I)$	$G_L(I)$	Places	Transitions	Edges	$P_L^n(M)$	$G_L(M)$	Activities	Constraints		
Activities	0.0136	0.858	5	67	134	0.057	0.8448	64	133	-0.043	0.013
BPIC2012	0.194	0.737	28	61	128	0.144	1.0	36	126	0.05	0.263
BPIC2013	0.0	≈ 1	13	29	58	0.234	0.998	13	31	-0.234	0.01
BPIC2017	0.07	1.0	11	71	146	0.094	1.0	66	143	-0.024	0.0
BPIC2018	0.017	1.0	10	53	106	0.006	0.999	41	105	0.011	≈ 0
Document	0.992	-	19	20	40	0.263	-	10	66	0.739	-
Dreyer	0.169	0.977	36	75	150	0.171	0.976	31	145	-0.002	≈ 0
Electronic	0.73	-	20	27	54	0.499	-	16	131	0.231	-
H. Billing	0.362	≈ 1	29	50	104	0.46	≈ 1	18	38	-0.098	0.0
NASA	0.686	0.976	70	114	228	0.251	≈ 1	94	217	0.435	-0.024
Product.	0.005	0.875	134	268	27	0.001	0.8088	110	25	0.004	0.067
Road Fine	0.72	≈ 1	17	23	52	0.788	1.0	11	44	-0.068	≈ 0
Sepsis	0.014	0.992	15	31	64	0.075	0.973	16	54	-0.061	0.02
WABO	0.288	0.994	17	48	96	0.387	0.994	27	90	-0.099	≈ 0

Table 3: Precision, generalisation and attributes of output models. Two logs were not suited for computing G_L since they had too few unique trace variants to perform cross-validation (Document Processing and Electronic Invoicing).

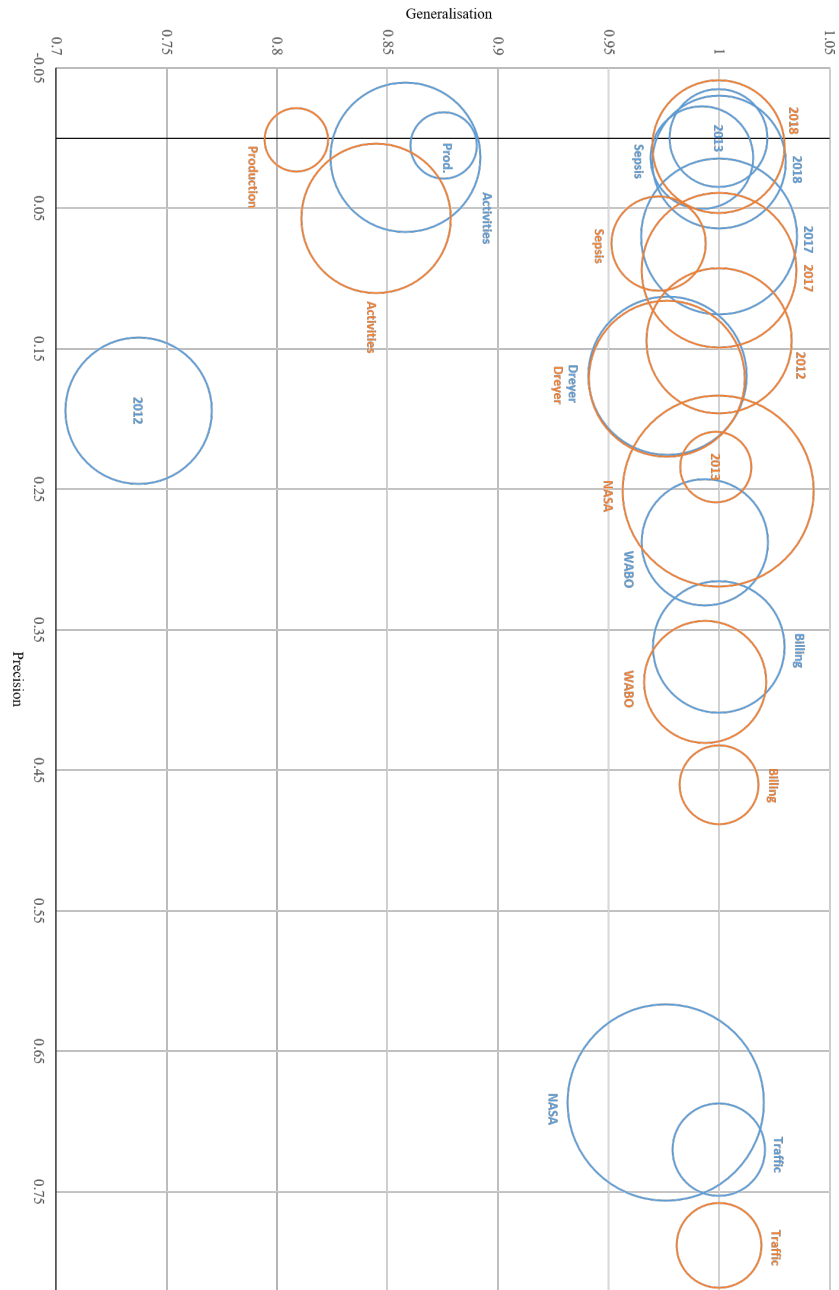


Fig. 1: Precision and generalisation of models generated by Inductive Miner and MINERful. Bubble size reflects the size of the models (edges and constraints, respectively).

3.5 Statistical analysis

To evaluate hypothesis 1a, we consider the mean difference μ_P in normalised precision for models generated by Inductive Miner and MINERful when mining the same logs. The null hypothesis H_0 states that there is no difference in miner performance across logs, and the research hypothesis H_P states that a difference exists. Formally, let \mathcal{L} denote the set of event logs, $P_L^n(I)$ the normalised precision of Inductive Miner on log L , and $P_L^n(M)$ the normalised precision of MINERful on log L ,

$$\mu_P = \frac{1}{|\mathcal{L}|} \sum_{L \in \mathcal{L}} P_L^n(I) - P_L^n(M) \quad H_0 : \mu_P = 0 \quad H_P : \mu_P \neq 0$$

To evaluate hypothesis 1b, we consider the weighted-mean, fitness-based 5-fold cross validation as a measure of generalisation and the difference of this metric between the two miners, on the same log. The null hypothesis states that no mean difference μ_G exists between the two miners across logs, while the research hypothesis states that a difference exists. Let $G_L(I)$ denote the generalisation (fitness-based cross-validation) of Inductive Miner on log L , and $G_L(M)$ the generalisation of MINERful on log L ,

$$\mu_G = \frac{1}{|\mathcal{L}|} \sum_{L \in \mathcal{L}} G_L(I) - G_L(M) \quad H_0 : \mu_G = 0 \quad H_G : \mu_G \neq 0$$

We take the included logs as a sample of all possible event logs, and assume the logs to be independent of one another, since they stem from significantly different sources and contexts. When evaluating hypothesis 1b, we have removed two logs: Document Processing and Electronic Invoicing, since they contain only 3 and 4 unique traces, respectively, making the data sets unsuitable for cross-validation.

For both hypotheses, the small sample size precludes assuming normality, requiring a nonparametric statistical test. The appropriate nonparametric test for comparing two related (paired) samples is the Wilcoxon signed rank test. Our hypothesis claims only that a difference exists between the miners, not that the difference exists to one miner's favor, so a two-tailed significance test is required. The level of significance α is set to 0.05.

Null Hypothesis	N	T	Crit. Value	Rejected?
$\mu_P = 0$	14	49	21	NO
$\mu_G = 0$	12	25	13	NO

Table 4: Wilcoxon signed rank test. Neither miner outperforms on precision or generalisation to a statistically significant degree.

3.6 Trade-off Analysis

Beyond analysing whether one miner outperforms categorically across all logs, we are interested in the trade-off between precision and generalisation and whether there exist logs that are best suited for mining imperatively or declaratively. The notion of a Pareto improvements is one approach to this analysis. If a miner outperforms on one metric without sacrificing performance on the other, it has made a Pareto improvement. The ideally, of course, it makes an improvement on both metrics.

Ideal Improvement and Trade-off Vectors Considering only strict Pareto improvements may be too limiting. We would like to capture the *degree* of improvement. Furthermore, a large improvement in one metric may be achievable at the expense of a negligible sacrifice of the other, but this will not qualify as a Pareto improvement. To this end, we anchor our analysis using the notion of an *equivalent, ideal improvement*, and its converse, an *equivalent, ideal trade-off*.

Let $\delta_P = P_L^n(I) - P_L^n(M)$ and $\delta_G = G_L(I) - G_L(M)$ and let vectors $\begin{pmatrix} \delta_P \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ \delta_G \end{pmatrix}$ form the basis of a vector space $V \subset \mathbb{R}^2$. Let

$$\mathbf{p} = \begin{pmatrix} \alpha \operatorname{argmax} \delta_P \\ \beta \operatorname{argmax} \delta_G \end{pmatrix} \quad \mathbf{t}_2 = \begin{pmatrix} \alpha \operatorname{argmin} \delta_P \\ \beta \operatorname{argmax} \delta_G \end{pmatrix} \quad \mathbf{t}_4 = \begin{pmatrix} \alpha \operatorname{argmax} \delta_P \\ \beta \operatorname{argmin} \delta_G \end{pmatrix}$$

Where α and β are weights whose relative value represents the importance of one metric to another. In the current context we simply set both to 1, giving $\mathbf{p} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, representing the scenario in which Inductive Miner achieves 1.0 on both metrics, and MINERful achieves 0.0. Orthogonal to this vector are the trade-off vectors $\mathbf{t}_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$ and $\mathbf{t}_4 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$.

Any vector $\begin{pmatrix} x \\ y \end{pmatrix}$ s.t. $\alpha x = \beta y$ falls parallel to \mathbf{p} and similarly, any vector s.t. $\alpha x = -\beta y$ falls parallel to the trade-off vectors \mathbf{t}_2 and \mathbf{t}_4 representing a decrease in one metric equal to the increase in the other, when adjusted for weighting preferences.

Projection onto Improvement Vector By projecting the data onto the ideal improvement vector \mathbf{p} , we can essentially reduce the dimensionality of the data in a manner which captures the degree of improvement, without requiring strict Pareto improvements and accounting for any preference weighting of the performance metrics. With this approach, a moderate improvement on both metrics may be collapsed to the same value as a large improvement on one metric requiring a small sacrifice of the other. In other words, both the direction and magnitude (norm) of the data vectors is taken into account.

Formally, let P be a one-dimensional subspace of \mathbb{R}^2 of which \mathbf{p} is the unit vector. Let $\operatorname{proj}_P(\mathbf{v})$ denote the projection of \mathbf{v} onto P .

$$\operatorname{proj}_P(\mathbf{v}) = \frac{\mathbf{v} \cdot \mathbf{p}}{\|\mathbf{p}\|^2} \mathbf{p} = \frac{\mathbf{v} \cdot \mathbf{p}}{\|\mathbf{p}\|} \frac{\mathbf{p}}{\|\mathbf{p}\|} \quad (5)$$

We are mainly interested in the magnitude of the projection (i.e. scalar projection):

$$\|\operatorname{proj}_P(\mathbf{v})\| = \|\mathbf{v}\| \cos \theta = \frac{\mathbf{v} \cdot \mathbf{p}}{\|\mathbf{p}\|} \quad (6)$$

In the special case that $\mathbf{p} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, the above reduces to:

$$\|\text{proj}_P\left(\begin{pmatrix} x \\ y \end{pmatrix}\right)\| = \frac{\begin{pmatrix} x \\ y \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix}}{\|\begin{pmatrix} 1 \\ 1 \end{pmatrix}\|} = \frac{1x + 1y}{\sqrt{1^2 + 1^2}} = \frac{x + y}{\sqrt{2}} \quad (7)$$

Log	$\frac{\mathbf{v} \cdot \mathbf{p}}{\ \mathbf{p}\ }$	Log	$\frac{\mathbf{v} \cdot \mathbf{p}}{\ \mathbf{p}\ }$
Activities of Daily Living	- 0.021	Hospital Billing	- 0.069
BPI Challenge 2012	- 0.15	Nasa Crew Explor. Veh.	0.291
BPI Challenge 2013	- 0.164	Production Analysis	0.05
BPI Challenge 2017	- 0.017	Sepsis Cases	- 0.029
BPI Challenge 2018	0.008	Road Traffic Fine Mgmt	- 0.048
Dreyer Foundation	- 0.001	WABO/CoSeLog - Receipt	- 0.07

Table 5: Scalar projection of 2-dimensional data vectors \mathbf{v} onto ideal vector $\mathbf{p} = (1, 1)^T$.

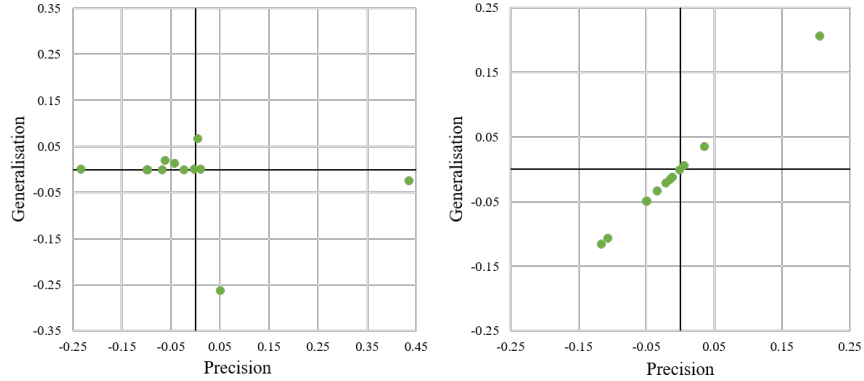


Fig. 2: *Left*: Difference between output models for each metric with boxes representing interquartile range. *Right*: Bivariate representation of the logarithm of ratio of precision and state-based generalisation. Positive values represent an advantage to Inductive Miner, negative values an advantage to MINERful. Quadrants II and IV represent a trade-off.

4 Results

We report in Table 3 the results of running the Inductive Miner and MINERful on the included logs. The *absolute* performance of the two miners on the two metrics is plotted in Figure 1, and their *relative* performance is plotted in Figure 2.

Using the Wilcoxon signed rank test, we are unable to reject the null hypotheses that $\mu_P = 0$ and $\mu_G = 0$ (see Table 4). That is, we answer RQ2 in the negative: neither paradigm outperforms consistently based on the present comparison.

Regarding RQ3, we observe that three logs stand out when using our approach: BPI Challenge 2012, BPI Challenge 2013, and NASA Crew Exploration Vehicle. After projection onto the ideal improvement vector (see Figure 2 (right)), it is clear that most logs cluster around the origin, indicating the overall improvement is small for these logs. The NASA log deviates greatly from this cluster to the advantage of Inductive Miner, while BPIC 2012/2013 also both clearly deviate to the advantage of MINERful. The magnitude (norm) of the projected vectors are listed in Table 5.

We also observe in Figure 2 that quadrants I and IV are more or less empty, indicating no logs for which an improvement is seen on both metrics. Many logs lie on the precision-axis ($P_L^n(I) - P_L^n(M)$), representing an improvement in precision with no change in generalisation. From Table 3, we see that this is a consequence of both miner achieving generalisation (G_L^n) scores of 1.0 or ≈ 1 .

A final observation can be made regarding model size from Table 3 and Figure 1. We have used the model size of Inductive Miner’s output as a strict upper bound on MINERful’s output (except in cases where no such model could be found), so it is no surprise that the latter are seldom larger than the latter. Nonetheless, we note those cases in which a much smaller model in fact provides better performance: for BPIC 2013 and Hopital Billing, MINERful generates models half the size of Inductive Miner while improving performance; while for Document Processing and Electronic Invoicing, Inductive Miner similarly produces much smaller models while improving precision.

5 Discussion

There are three interesting conclusions we can draw from our findings:

1. Neither mining paradigm can be said to perform categorically better across logs on the metrics under consideration.
2. Logs appear to fall into three categories: those for which mining paradigm makes only an insignificant difference (clustered around the origin), and those clearly best suited to one paradigm or the other.
3. Perhaps most strikingly: the formulations of precision and generalisation under consideration do not appear to be inversely correlated. Consider how few logs fall very clearly into the “trade-off” category, as might be expected - instead they usually fall close the axes.

The visualisations in Figures 1 and 2 illustrate all three points.

If one insists that generalisation and precision should be inversely correlated, then our study underlines the need to carefully consider the interplay between the different quality dimensions when developing specific metrics.

Regarding RQ3 we are able to identify at least four logs which are significantly more suited for either declarative or imperative mining, showing that there are particular circumstances in which each paradigm clearly outshines the other. Other logs either required a trade-off between precision and generalisation, or the difference between

the two miners could not be considered statistically significant. It should also be noted that for one log, “Hospital Event Log”, MINERful was unable to return a model, so by default this log falls to Inductive Miner’s favour.

A corollary of these hypotheses is that rejecting any of the null hypotheses provides evidence for the validity of the metrics themselves, i.e. that the metrics do in fact measure *something* about the mined model. If it were the case that precision and generalisation remained more or less constant for event logs, regardless of mining algorithm, this would indicate that the metrics measured an attribute of the log itself, rather than the correspondence of mined model to log.

We recognise some potential threats to the validity of the presented study: the representativeness of our sample set, the particular metrics used and the number of miners considered. Our choices and reasons are discussed in detail in Section 3. Each threat can be broken down to the current maturity of the field: there exist only a small number of publicly available real-life logs, there are only few metrics that can be equally applied to imperative and declarative models, and these metrics in turn limit our choice of miners. We therefore posit that the limitations placed on the study are reasonable given the state of the art. In addition, to the best of our knowledge, this will be the most exhaustive study comparing imperative and declarative process discovery techniques to date. Future evaluations incorporating other aspects of the process mining life-cycle, e.g. alignment, will have this approach as a point of reference. Not least, we contribute a comprehensive software framework and tackle a number of methodological and implementation challenges, providing a foundation upon which further work can build.

6 Conclusion

We gave an approach to systematically comparing the performance of imperative versus declarative process mining algorithms: studying Pareto-improvements of comparable quality metrics (RQ1). We proposed the commonly accepted quality metrics for precision and generalisation defined by [33] as forming a trade-off to study semantic model quality (RQ2); and we studied this trade-off statistically on commonly available logs (RQ3).

For the latter, we investigated two hypotheses: first, that one miner generally performs better on precision and/or generalisation; second, that there exist particular logs on which either miner provides a statistically significant Pareto improvement on both. The latter is confirmed, the former is not. While we did not consider simplicity as a dependent variable under evaluation, we did observe that in cases of comparable precision and generalisation declarative models often have significantly fewer elements.

Future Work We believe that the current study will be valuable to the field of hybrid process mining [19, 27, 28], which aims to combine the strengths of the two paradigms. Ongoing research into what characteristics make a log, or sub-log, more suitable to one paradigm over the other are hampered by the fact that there exists no established common framework for comparing miners across paradigms [7]. Moreover, the identification of logs that are significantly more suitable to one paradigm over the other shows that there are real-life scenarios for which one paradigm outperforms the other. We expect that this study will open the door to a more thorough investigation of the properties

that make a log more suitable to a paradigm, which in turn will drive the development of hybrid process discovery approaches that combine both paradigms.

We see two major avenues of research continuing the present work: (1) applying the classification of logs to hybrid mining, and (2) improving classification by including additional metrics and miners.

For (1), the present study may serve as a prerequisite step toward identifying attributes of logs or sub-logs indicating an advantage to a given mining paradigm. An obvious next step is to identify such attributes, and learn how they correlate with improved precision and generalisation.

For (2), the present study considers only the core metrics of generalisation and precision. We have argued that these together capture a meaningful core trade-off in semantic capabilities of output models. However, many other metrics exist. In particular studying the fitness/simplicity trade-off would be very interesting; however, one would need a notion of simplicity that is meaningful for and comparable between both declarative and imperative models; we are unaware of such notions in the literature. Moreover, to truly evaluate a fitness/simplicity trade-off, one would need a way to allow non-perfect fitness without biasing results through the use of a biased cost function in alignment, while still allowing the precision and generalisation metrics to be computed.

References

1. Wil M. P. van der Aalst. Verification of workflow nets. ICATPN '97, pages 407–426, 1997.
2. Arya Adriansyah, Jorge Munoz-Gama, Josep Carmona, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Alignment Based Precision Checking. In *BPM Workshops, Lecture Notes in Business Information Processing*, pages 137–149, 2012.
3. Arya Adriansyah, Jorge Munoz-Gama, Josep Carmona, Boudewijn F. van Dongen, and Wil MP van der Aalst. Alignment based precision checking. In *International conference on business process management*, pages 137–149. Springer, 2012.
4. Arya Adriansyah, Boudewijn F. van Dongen, and Wil MP van der Aalst. Conformance checking using cost-based fitness analysis. In *EDOC 2011*, pages 55–64.
5. Adriano Augusto, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, Fabrizio M Maggi, Andrea Marrella, Massimo Mecella, and Allar Soo. Automated discovery of process models from event logs: Review and benchmark. *IEEE Transactions on Knowledge and Data Engineering*, 2018.
6. Christoffer Olling Back, Søren Debois, and Tijs Slaats. Towards an empirical evaluation of imperative and declarative process mining. In *International Conference on Conceptual Modeling*, pages 191–198. Springer, 2018.
7. Christoffer Olling Back, Søren Debois, and Tijs Slaats. Towards an Entropy-based Analysis of Log Variability. In *Lecture Notes in Business Information Processing*, 2017.
8. J. C. a. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst. Quality Dimensions in Process Discovery: The Importance of Fitness, Precision, Generalization and Simplicity. *International Journal of Cooperative Information Systems*, 23(01):1440001, March 2014.
9. Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. In *OTM 2012, Lecture Notes in Computer Science*, pages 305–322, 2012.
10. Claudio Di Ciccio and Massimo Mecella. On the discovery of declarative control flows for artful processes. *ACM Transactions on Management Information Systems*, 5(4):24, 2015.

11. Jochen De Weerd, Manu De Backer, Jan Vanthienen, and Bart Baesens. A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. *Information Systems*, 37(7):654–676, November 2012.
12. S. Debois and T. Slaats. The analysis of a real life declarative process. In *CIDM 2015*, pages 1374–1382, 2015.
13. Søren Debois, Thomas T. Hildebrandt, and Tijs Slaats. Replication, refinement & reachability: complexity in dynamic condition-response graphs. *Acta Informatica*, Sep 2017.
14. Søren Debois, Thomas T. Hildebrandt, Paw Høvsgaard Laursen, and Kenneth Ry Ulrik. Declarative Process Mining for DCR Graphs. *SAC '17*, pages 759–764, 2017.
15. Stijn Goedertier, David Martens, Jan Vanthienen, and Bart Baesens. Robust process discovery with artificial negative events. *Journal of Machine Learning Research*, 10(Jun):1305–1340, 2009.
16. R. Hull, E. Damaggio, R. De Masellis, F. Fournier, M. Gupta, F.T. Heath, S. Hobson, M.H. Linehan, S. Maradugu, A. Nigam, P. Noi Sukaviriya, and R. Vaculín. Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In *DEBS 2011*, pages 51–62, 2011.
17. Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. In *Application and Theory of Petri Nets and Concurrency*, pages 311–329, June 2013.
18. Fabrizio Maria Maggi, R. P. Jagadeesh Chandra Bose, and Wil M. P. van der Aalst. Efficient discovery of understandable declarative process models from event logs. In *CAiSE*, pages 270–285, 2012.
19. Fabrizio Maria Maggi, Tijs Slaats, and Hajo A. Reijers. The automated discovery of hybrid processes. In *Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings*, pages 392–399, 2014.
20. M. Marquard, M. Shahzad, and T. Slaats. Web-based modelling and collaborative simulation of declarative processes. In *BPM 2015*, pages 209–225, 2015.
21. Jorge Munoz-Gama and Josep Carmona. Enhancing precision in process conformance: stability, confidence and severity. In *Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on*, pages 184–191. IEEE, 2011.
22. Viktorija Nekrasaite, Andrew Tristan Parli, Christoffer Olling Back, and Tijs Slaats. Discovering responsibilities with dynamic condition response graphs. In *International Conference on Advanced Information Systems Engineering*, pages 595–610. Springer, 2019.
23. Object Management Group. Business Process Modeling Notation Version 2.0. Technical report, Object Management Group Final Adopted Specification, 2011.
24. Object Management Group. Case Management Model and Notation, version 1.0. Webpage, may 2014. <http://www.omg.org/spec/CMMN/1.0/PDF>.
25. Hajo A. Reijers, Tijs Slaats, and Christian Stahl. Declarative modeling—an academic dream or the future for bpm? In *Business Process Management*, pages 307–322. 2013.
26. Anne Rozinat and Wil M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Inf. Syst.*, 33(1):64–95, 2008.
27. Dennis MM Schunselaar, Tijs Slaats, Fabrizio M Maggi, Hajo A Reijers, and Wil MP Van Der Aalst. Mining hybrid business process models: a quest for better precision. In *International Conference on Business Information Systems*, pages 190–205. Springer, 2018.
28. Tijs Slaats, Dennis M. M. Schunselaar, Fabrizio Maria Maggi, and Hajo A. Reijers. The semantics of hybrid process models. In *CoopIS 2016*, pages 531–551, 2016.
29. Niek Tax, Xixi Lu, Natalia Sidorova, Dirk Fahland, and Wil M. P. van der Aalst. The Imprecisions of Precision Measures in Process Mining. *arXiv:1705.03303 [cs]*, May 2017. arXiv: 1705.03303.
30. Wil van der Aalst. *Process Mining*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

31. Wil van der Aalst, Maja Pesic, Helen Schonenberg, Michael Westergaard, and Fabrizio M. Maggi. Declare. Webpage, 2010. <http://www.win.tue.nl/declare/>.
32. Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.
33. Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisc. Rev.: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.
34. Wil M. P. van der Aalst, Vladimir Rubin, H. M. W. Verbeek, Boudewijn F. van Dongen, Ekkart Kindler, and Christian W. Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software and System Modeling*, 9(1):87–111, 2010.

Mining Massive SAP Transaction Logs

Christoffer Olling Back

In Spring of 2019, a 4-month collaboration between GekkoBrain A/S and the University of Copenhagen was initiated in which to explore applications of process mining research in industry through a proof-of-concept project based on data and domain expert input from Vestas Wind System. Formally, my role in the project was a Research Assistant and a leave of absence was granted from my position as PhD Fellow. In other words, the project was not formally a part of the PhD, and no academic research publications have yet resulted from the work. nonetheless I include a short description of the project and insights gained since they are highly relevant to the topic of this thesis and motivated some of my later work.

GekkoBrain A/S specializes in analysis of SAP systems, the dominant enterprise resource planning (ERP) platform globally. In their collaboration with Vestas Wind Systems, they tasked with developing a tool for mining transactions recorded from Vestas' SAP systems worldwide. The ultimate goal was to gain insight into the root causes of - usually undesirable - variations in processes that sometimes span several areas of operation in multiple continents. The data available are highly granular: essentially every modification of a document in the SAP is recorded as database transaction - from the creation of a work order or invoice, to the receipt of goods in a warehouse, to the adjustment of the stock count of an item.

Many document modifications establish a connection between multiple documents, e.g. a receipt of goods fulfills a purchase order, and later that purchase order is paid in accounts payable - or sent to revision if the received goods differ from those requested. The notion of a *trace* - a fundamental component which is assumed as given by all process mining algorithms - is not well defined in the raw data. Instead an enormous graph of associations between documents exists in the raw data, indeed many highly granular transactions are of little practical interest, despite constituting a sizable portion of the full dataset. In practice, a great deal of manual work by domain experts is usually required to build heuristic-based approaches to extracting traces from this data such that classical process mining algorithms can be applied. It should be noted that a body of research does exist that addresses the mining of process related data at this low-level of granularity, and is likely to be a direction of research that will see increasing attention [1].

One recurring challenge stemmed from the fact that there exist automated processes which perform mass updates enormous numbers of documents in otherwise unrelated processes. This results in otherwise independent components in the transaction graph which suddenly become connected and appear as though they are related to the same process. Aside from being misleading, these leads to some of the necessary graph operations, such as transitive reduction, to become excessively expensive.

The lionshare of our work during this short project went into building a platform for extracting and processing this low-level and extracting patterns akin

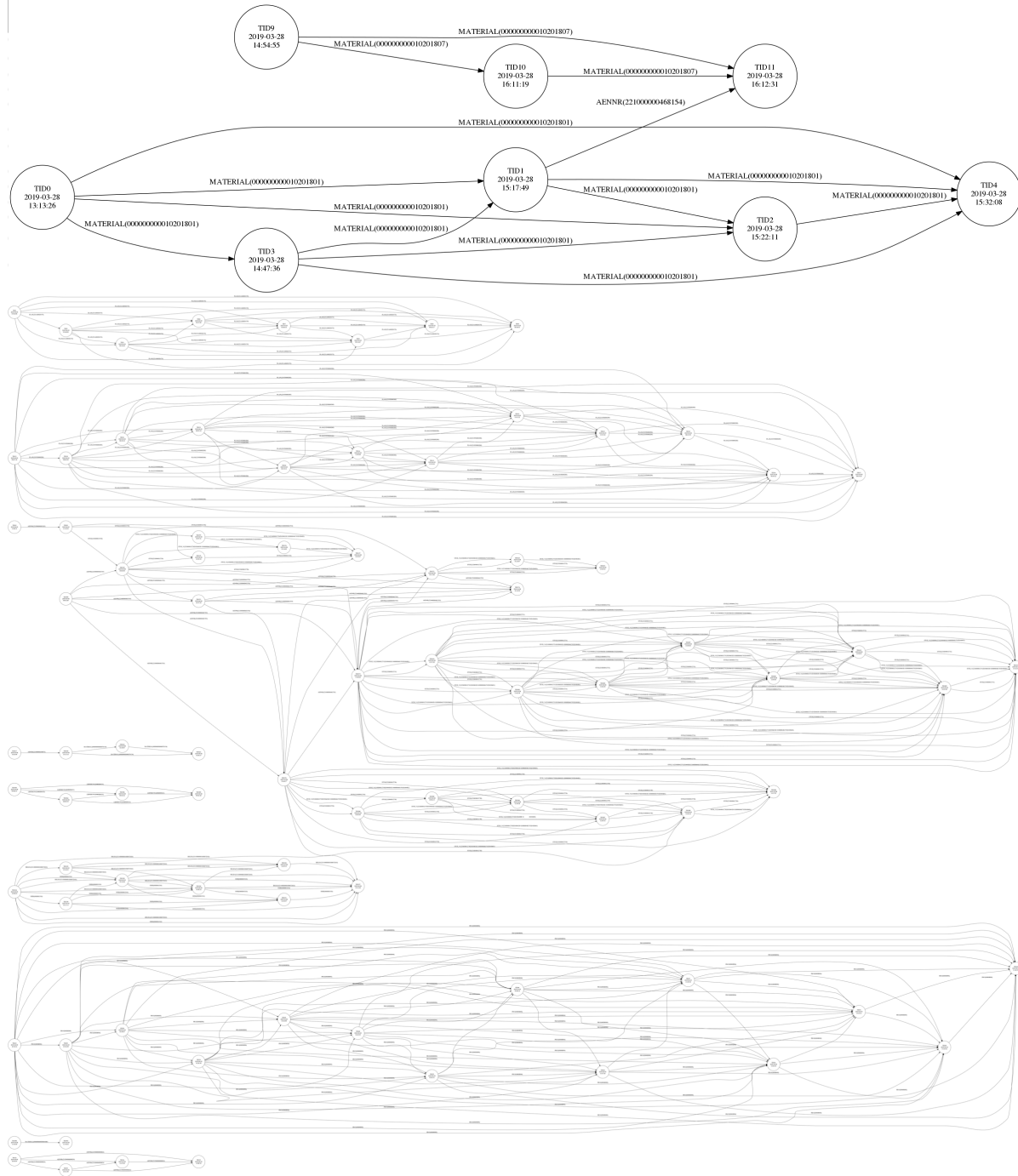


Figure 5.1: Examples of connected components in transaction graphs from a real SAP system. Edges between transactions indicate that they are related in terms of criteria set out by domain experts. Many components in the dataset were much larger than the examples here.

to traces. However Gekkobrain’s approach diverges somewhat from traditional process mining in this respect: rather than forcing the data into a simple sequential format like a trace, a more complex graph structure is maintained that is arguably more true to reality. From this, a graphical representation built for the user which does not have the formal semantics of a traditional Petri net, such that the underlying data is not filtered away to the same degree as is the case in other tools.

An interface for querying and exploring the dataset w.r.t. criteria such as timespan, starting activities, as attributes of processes, along with the ability to zoom in on data and discover similar processes, was developed. Again, the basic implementation challenges of built this search engine consumed the majority of our efforts. In the end, the challenges encountered in not only working this real-world dataset, but also the inputs from domain experts at Gekkobrain as well as from executives at Vestas Wind Systems, were incredibly informative and highlighted the real-world needs and challenges for practitioners within the process mining/intelligence space.

Bibliography

- [1] Kiarash Diba, Kimon Batoulis, Matthias Weidlich, and Mathias Weske. Extraction, correlation, and abstraction of event data for process mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 10(3):e1346, 2020.

Modelling Operating Theatre Workflows: A Brief Literature Review

Christoffer O. Back

Healthcare processes are complex and often unpredictable, requiring a high degree of flexibility. While they involve a great deal of domain knowledge on the part of practitioners, it is also an area in which safety is paramount while resources and expertise are a constraining factor. This has motivated a large body of research in modelling and optimisation of healthcare processes.

Comprehensive literature reviews of modelling processes in healthcare exist, often focusing on specific disciplines like Operations Research and Process Mining [39, 27, 29, 18]. This brief review will focus specifically on workflows in and surrounding surgery and the operating theatre. More extensive reviews focusing on operating theatres and surgical processes include [30, 42, 43, 41].

We identify two dimensions on which research can be characterised. First, the perspective taken of the process, i.e. whether the focus lies on patient pathways, workflow of personnel or resource utilisation, as well as the granularity, or scope, of the process being modelled. Second, research can usually be distinguished by discipline, i.e. the techniques used for modelling and optimisation.

1 Process Perspectives

1.1 Scope/Granularity

The level of detail in which processes are modelled is often a distinguishing feature, ranging from highly detailed analysis of surgical video, motion and activity recognition, to individual steps of the procedure, to the flow of patients within section of an operating theatre, and finally end-to-end patient flows from admission to release and even follow-up. Cross organisational and inter-ward logistics are also a common focus.

Detailed models of tool usage patterns, based on low-level sensor data collected during Cholecystectomies (a highly standardised procedure), were developed in [7, 8, 9, 35].

In [45], the workflow within an operating theatre is analysed, including anaesthesia, operation, and early recovery, without specifying details of the surgery. The authors proposed an “operating theatre of the future” in which up to three patients can proceed through the operating theatre, each in one of these three stages.

Other studies take a more encompassing view of the process surrounding surgery, from admission to recovery [16]. For example, entering the ICU after surgery can be a bottleneck in the overall process due to resource constraints [5]. Some even incorporate diagnosis and follow-up after surgery [31, 22].

1.2 Patient Pathways, Resource Allocation, Scheduling

When analysing a surgical process, an obvious view to take is that of the flow of the patient through the process, as in [49, 36].

It is also common to take the resource-oriented perspective with the aim of optimising utilisation, capacity and efficiency. Balancing between quality of care, safety and avoiding wasteful overcapacity or inefficient utilisation of expensive resources is a complex problem of constrained optimisation. Staff skill mix, facility capacity, and even organ availability need to be coordinated in as safe and efficient a manner as possible.

To this end, models for balancing bed capacity between wards, scheduling of surgeries w.r.t. staffing constraints, predicting arrival rates, and enforcement of safety/service requirements have been developed (see remainder).

2 Approaches

We identify several overarching disciplines from which researchers approach the quantitative modelling of surgical workflows. Despite a good deal of overlap, there are distinctions in modelling tools employed.

2.1 Machine Learning

A number of approaches falling under the machine learning moniker have been explored in the literature on surgical workflows. Reinforcement learning techniques were used for resource allocation in [24, 23]. Prediction for clinical pathways using latent variable models was addressed in [20, 21], and using a suite of clustering and classification techniques in [16]. Patient flows were modelled using a sophisticated Markov model in [17]. Random forests were used to detect surgical workflow phases in [46]. Bayesian belief networks were used to model several aspects of patients' stay in an emergency department in [2].

2.2 Operations Research

Usually concerned with resource allocation and network flow analysis, the discipline of operations research uses techniques like queuing theory, linear programming for solving constrained optimisation problems, as well as simulation.

The scheduling of surgeries has received a good deal of attention. It was explored in regards to optimising resource utilisation based on patient mix was investigated in [3], on operating theatre specialisation in [12], surgery duration variance in [14], on case urgency in [51], and to reduce overtime costs while

adhering to labor laws in [10]. In [4] a decomposition approach is taken to scheduling which incorporates several of the aforementioned factors.

Stochastic balancing of bed capacity based on fluctuating demand patterns was explored in [13] length of stay patterns in [5]. Resource allocation and patient admission was addressed in [25].

Simulations are another common tool for testing the effect of proposed workflow alterations [1, 38, 37, 15, 26, 32, 44]. Sophisticated simulations are even used to train personnel using virtual reality [48, 47] and augmented reality technology [33].

2.3 Process Mining and Formal Methods

Process mining approaches are usually characterised by employing proper process modelling formalisms with well defined semantics, commonly Petri nets or their equivalents, temporal logics, or process algebras.

Conformance checking was implemented in an intelligent hospital IT system in [40] for a surgical process. Modelled using the Declare language, it incorporated automated model repair and conformance checking using a cross-validation based approach.

The anaesthetic process for Endoscopic Retrograde Cholangiopancreatography procedure was investigated using the Heuristics miner in [28].

Formal process models have the advantage of being amenable to formal verification to ensure safety. In [6] Petri nets are used to model an operating theatre workflow, verified for soundness, and subsequently reengineered to optimise resource utilisation while ensuring the soundness property is retained.

Another common theme in the literature is that of formally modelling, and reasoning about, the surgical domain or medical organisations in which surgery takes place. In [34, 11] systems were developed which integrate a knowledge base (ontology) with surgical workflow models and a reasoning engine for querying. A hierarchical ontology incorporating temporal rules for clinical pathways is described in [50], while in [19] electronic medical records are integrated into clinical pathway ontologies.

References

- [1] Waleed Abo-Hamad and Amr Arisha. Simulation-based framework to improve patient experience in an emergency department. *European Journal of Operational Research*, 224(1):154–166, 2013.
- [2] Silvia Acid, Luis M de Campos, Juan M Fernández-Luna, Susana Rodriguez, José Maria Rodriguez, and José Luis Salcedo. A comparison of learning algorithms for bayesian networks: a case study based on data from an emergency medical service. *Artificial intelligence in medicine*, 30(3):215–232, 2004.

- [3] Ivo Adan, Jos Bekkers, Nico Dellaert, Jan Vissers, and Xiaoting Yu. Patient mix optimisation and stochastic resource requirements: A case study in cardiothoracic surgery planning. *Health care management science*, 12(2):129, 2009.
- [4] Alessandro Agnetis, Alberto Coppi, Matteo Corsini, Gabriella Dellino, Carlo Meloni, and Marco Pranzo. A decomposition approach for the combined master surgical schedule and surgical case assignment problems. *Health care management science*, 17(1):49–59, 2014.
- [5] Renzo Akkerman and Marrië Knip. Reallocation of beds to reduce waiting time for cardiac surgery. *Health care management science*, 7(2):119–126, 2004.
- [6] Kamel Barkaoui, Ph Dechambre, and Rim Hachicha. Verification and optimisation of an operating room workflow. In *Proceedings of the 35th annual hawaii international conference on system sciences*, pages 2581–2590. IEEE, 2002.
- [7] Tobias Blum, Nicolas Padoy, Hubertus Feußner, and Nassir Navab. Workflow mining for visualization and analysis of surgeries. *International journal of computer assisted radiology and surgery*, 3(5):379–386, 2008.
- [8] Loubna Bouarfa and Jenny Dankelman. Workflow mining and outlier detection from clinical activity logs. *Journal of biomedical informatics*, 45(6):1185–1190, 2012.
- [9] Loubna Bouarfa, Pieter P Jonker, and Jenny Dankelman. Discovery of high-level tasks in the operating room. *Journal of biomedical informatics*, 44(3):455–462, 2011.
- [10] Jens O Brunner, Jonathan F Bard, and Rainer Kolisch. Flexible shift scheduling of physicians. *Health care management science*, 12(3):285–305, 2009.
- [11] Oliver Burgert, Thomas Neumuth, Frieder Lempp, Raj Mudunuri, Juergen Meixensberger, G Strauß, Andreas Dietz, Pierre Jannin, and HU Lemke. Linking top-level ontologies and surgical workflows. *International Journal of Computer Assisted Radiology and Surgery*, 1:437, 2006.
- [12] Sangdo Choi and Wilbert E Wilhelm. An approach to optimize block surgical schedules. *European Journal of Operational Research*, 235(1):138–148, 2014.
- [13] Jeffery K Cochran and Aseem Bharti. Stochastic bed balancing of an obstetrics hospital. *Health care management science*, 9(1):31–45, 2006.
- [14] Brian Denton, James Viapiano, and Andrea Vogl. Optimization of surgery sequencing and scheduling decisions under uncertainty. *Health care management science*, 10(1):13–24, 2007.

- [15] Jeffrey E Everett. A decision support simulation model for the management of an elective surgery waiting system. *Health care management science*, 5(2):89–95, 2002.
- [16] Anastasia A Funkner, Aleksey N Yakovlev, and Sergey V Kovalchuk. Towards evolutionary discovery of typical clinical pathways in electronic health records. *Procedia computer science*, 119:234–244, 2017.
- [17] Lalit Garg, Sally McClean, Brian Meenan, and Peter Millard. A non-homogeneous discrete time markov model for admission scheduling and resource planning in a cost or capacity constrained healthcare system. *Health Care Management Science*, 13(2):155–169, 2010.
- [18] Phil Gooch and Abdul Roudsari. Computerization of workflows, guidelines, and care pathways: a review of implementation challenges for process-oriented health information systems. *Journal of the American Medical Informatics Association*, 18(6):738–748, 2011.
- [19] Zhen Hu, Jing-Song Li, Tian-Shu Zhou, Hai-Yan Yu, Muneou Suzuki, and Kenji Araki. Ontology-based clinical pathways with semantic rules. *Journal of medical systems*, 36(4):2203–2212, 2012.
- [20] Zhengxing Huang, Wei Dong, Lei Ji, and Huilong Duan. Predictive monitoring of clinical pathways. *Expert Systems with Applications*, 56:227–241, 2016.
- [21] Zhengxing Huang, Wei Dong, Lei Ji, Chenxi Gan, Xudong Lu, and Huilong Duan. Discovery of clinical pathway patterns from event logs using probabilistic topic models. *Journal of biomedical informatics*, 47:39–57, 2014.
- [22] Zhengxing Huang, Xudong Lu, Huilong Duan, and Wu Fan. Summarizing clinical pathways from event logs. *Journal of biomedical informatics*, 46(1):111–127, 2013.
- [23] Zhengxing Huang, Wil MP van der Aalst, Xudong Lu, and Huilong Duan. An adaptive work distribution mechanism based on reinforcement learning. *Expert Systems with Applications*, 37(12):7533–7541, 2010.
- [24] Zhengxing Huang, Wil MP van der Aalst, Xudong Lu, and Huilong Duan. Reinforcement learning based resource allocation in business process management. *Data & Knowledge Engineering*, 70(1):127–145, 2011.
- [25] Peter JH Hulshof, Richard J Boucherie, Erwin W Hans, and Johann L Hurink. Tactical resource allocation and elective patient admission planning in care processes. *Health care management science*, 16(2):152–166, 2013.
- [26] PE Joustra, J De Wit, VMD Struben, BJH Overbeek, P Fockens, and SG Elkhuisen. Reducing access times for an endoscopy department by an iterative combination of computer simulation and linear programming. *Health care management science*, 13(1):17–26, 2010.

- [27] Cengiz Kahraman and Y Ilker Topcu. *Operations Research Applications in Health Care Management*. Springer, 2018.
- [28] Uzay Kaymak, Ronny Mans, Tim Van de Steeg, and Meghan Dierks. On process mining in health care. In *2012 IEEE international conference on Systems, Man, and Cybernetics (SMC)*, pages 1859–1864. IEEE, 2012.
- [29] Angelina Prima Kurniati, Owen Johnson, David Hogg, and Geoff Hall. Process mining in oncology: A literature review. In *2016 6th International Conference on Information Communication and Management (ICIM)*, pages 291–297. IEEE, 2016.
- [30] Florent Lalys and Pierre Jannin. Surgical process modelling: a review. *International journal of computer assisted radiology and surgery*, 9(3):495–511, 2014.
- [31] Ronny Mans, Hajo Reijers, Michiel van Genuchten, and Daniel Wismeijer. Mining processes in dentistry. In *Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium*, pages 379–388. ACM, 2012.
- [32] Riitta A Marjamaa, Paulus M Torkki, Eero J Hirvensalo, and Olli A Kirvelä. What is the best workflow for an operating room a simulation study of five scenarios. *Health Care Management Science*, 12(2):142, 2009.
- [33] Nassir Navab, Joerg Traub, Tobias Sielhorst, Marco Feuerstein, and Christoph Bichlmeier. Action-and workflow-driven augmented reality for computer-aided medical procedures. *IEEE Computer Graphics and Applications*, 27(5):10–14, 2007.
- [34] T Neumuth, R Mudunuri, P Jannin, J Meixensberger, and O Burgert. Swan-suite: The tool landscape for surgical workflow analysis. *Computer Assisted Medical and Surgical Interventions (SURGETICA). Paris: Sauramps Medical*, pages 199–204, 2007.
- [35] Thomas Neumuth, Pierre Jannin, Julianne Schlomberg, Jürgen Meixensberger, Peter Wiedemann, and Oliver Burgert. Analysis of surgical intervention populations using generic surgical process models. *International Journal of Computer Assisted Radiology and Surgery*, 6(1):59–71, 2011.
- [36] Lua Perimal-Lewis, Denise De Vries, and Campbell H Thompson. Health intelligence: discovering the process model using process mining by constructing start-to-end patient journeys. In *Proceedings of the Seventh Australasian Workshop on Health Informatics and Knowledge Management-Volume 153*, pages 59–67. Australian Computer Society, Inc., 2014.
- [37] Marie Jeanette Persson and Jan A Persson. Analysing management policies for operating room planning using simulation. *Health care management science*, 13(2):182–191, 2010.

- [38] Julie Ratcliffe, Tracey Young, Martin Buxton, Tillal Eldabi, Ray Paul, Andrew Burroughs, George Papatheodoridis, and Keith Rolles. A simulation modelling approach to evaluating alternative policies for the management of the waiting list for liver transplantation. *Health Care Management Science*, 4(2):117–124, 2001.
- [39] Eric Rojas, Jorge Munoz-Gama, Marcos Sepúlveda, and Daniel Capurro. Process mining in healthcare: A literature review. *Journal of biomedical informatics*, 61:224–236, 2016.
- [40] Marcella Rovani, Fabrizio M Maggi, Massimiliano de Leoni, and Wil MP van der Aalst. Declarative process mining in healthcare. *Expert Systems with Applications*, 42(23):9236–9251, 2015.
- [41] Michael Samudra, Carla Van Riet, Erik Demeulemeester, Brecht Cardoen, Nancy Vansteenkiste, and Frank E Rademakers. Scheduling operating rooms: achievements, challenges and pitfalls. *Journal of Scheduling*, 19(5):493–525, 2016.
- [42] Boris G Sobolev, Victor Sanchez, and Christos Vasilakis. Systematic review of the use of computer simulation modeling of patient flow in surgical care. *Journal of medical systems*, 35(1):1–16, 2011.
- [43] KW Soh, C Walker, and M OSullivan. A literature review on validated simulations of the surgical services. *Journal of medical systems*, 41(4):61, 2017.
- [44] James E Stahl, David Rattner, Richard Wiklund, Jessica Lester, Molly Beinfeld, and G Scott Gazelle. Reorganizing the system of care surrounding laparoscopic surgery: a cost-effectiveness analysis using discrete-event simulation. *Medical decision making*, 24(5):461–471, 2004.
- [45] James E Stahl, Warren S Sandberg, Bethany Daily, Richard Wiklund, Marie T Egan, Julian M Goldman, Keith B Isaacson, Scott Gazelle, and David W Rattner. Reorganizing patient care and workflow in the operating room: a cost-effectiveness study. *Surgery*, 139(6):717–728, 2006.
- [46] Ralf Stauder, Ash Okur, Loïc Peter, Armin Schneider, Michael Kranzfelder, Hubertus Feussner, and Nassir Navab. Random forests for phase detection in surgical workflow analysis. In *International Conference on Information Processing in Computer-Assisted Interventions*, pages 148–157. Springer, 2014.
- [47] Akshay Vankipuram, Stephen Traub, and Vimla L Patel. A method for the analysis and visualization of clinical workflow in dynamic environments. *Journal of biomedical informatics*, 79:20–31, 2018.
- [48] Mithra Vankipuram, Kanav Kahol, Trevor Cohen, and Vimla L Patel. Toward automated workflow analysis and visualization in clinical environments. *Journal of biomedical informatics*, 44(3):432–440, 2011.

- [49] Hongteng Xu, Weichang Wu, Shamim Nemati, and Hongyuan Zha. Patient flow prediction via discriminative learning of mutually-correcting processes. *IEEE transactions on Knowledge and Data Engineering*, 29(1):157–171, 2016.
- [50] Yan Ye, Zhibin Jiang, Xiaodi Diao, Dong Yang, and Gang Du. An ontology-based hierarchical semantic modeling approach to clinical pathway workflows. *Computers in biology and medicine*, 39(8):722–732, 2009.
- [51] Maartje E Zonderland, Richard J Boucherie, Nelly Litvak, and Carmen LAM Vleggeert-Lankamp. Planning and scheduling of semi-urgent surgeries. *Health Care Management Science*, 13(3):256–267, 2010.

Mining Patient Flow Patterns in a Surgical Ward

Christoffer O. Back¹^a, Areti Manatakis²^b and Ewen Harrison²^c

¹*Department of Computer Science, University of Copenhagen, Denmark*

²*Usher Institute, University of Edinburgh, U.K.*

back@di.ku.dk, a.manatakis@ed.ac.uk, mail@ewenharrison.com

Keywords: Bayesian Network, Data Mining, Patient Flows, Process Mining, Surgery, Surgical Workflow.

Abstract: Surgery is a highly critical and costly procedure, and there is an imperative need to improve the efficiency in surgical wards. Analyzing surgical patient flow and predicting cycle times of different peri-operative phases can help improve the scheduling and management of surgeries. In this paper, we propose a novel approach to mining temporal patterns of surgical patient flow with the use of Bayesian belief networks. We present and compare three classes of probabilistic models and we evaluate them with respect to predicting cycle times of individual phases of patient flow. The results of this study support previous work that surgical times are log-normally distributed. We also show that the inclusion of a clustering pre-processing step improves the performance of our models considerably.

1 INTRODUCTION

Surgery is a cornerstone of the healthcare system, and critical in terms of time and resources. Ensuring efficiency, timeliness and safety are crucial for providing high quality service while controlling costs (Lalys and Jannin, 2014), (Denton, 2007). While many processes surrounding surgery are well structured, the dynamic nature of patient arrivals combined with the complexity of coordinating large numbers of specialized staff and facilities, means that delays and misalignments can have cascading effects leading to last-minute cancellations. This leads not only to an under-utilization of expensive resources, but causes stress and upheaval for patients.

The well-defined, yet dynamic; and high-cost, high-impact nature of surgical patient flows, suggests it is an area amenable to improvements via data oriented process modeling. Advances in forecasting long and short term dynamics of the surgical ward can help inform intelligent surgery sequencing, staff scheduling and workflow management systems.


This paper presents a preliminary investigation into methods for modeling patient flows in surgical wards, with outset in a data set following patients from admission to discharge at the Royal In-


firmary of Edinburgh. We focus our present investigation on *temporal* aspects of *individual* patient flows, which are key to improving efficiency. Results from this study can then inform the investigation of other aspects of patient flows such as positioning, as well as high-level dynamics between multiple patient flows competing for shared resources at the level of ward/hospital.


After an exploratory investigation of the data, we present and compare three probabilistic models describing cycle times of individual phases in patient flows prior to, during and following surgery. We evaluate these w.r.t. to predicting cycle times of individual phases of patient flows, from the time patients are sent for, through anesthesia and surgery, and until they leave recovery.

Specifically, we employ a type of probabilistic model called a Bayesian network. Aside from their capacity to easily incorporate domain knowledge, Bayesian networks have the advantage that they can be queried in complex ways even with incomplete evidence, which is invaluable in the uncertain hospital environment. Crucially, we show that by incorporating a pre-processing step based on simple clustering of flows w.r.t. cycle times, we can improve the performance of our models noticeably.

The structure of the sequel is as follows. In Section 2 we review existing literature. Our subsequent analysis of the data follows the classic data analytics workflow of *Describe* \rightarrow *Diagnose* \rightarrow *Predict*. In

^a <https://orcid.org/0000-0001-7998-7167>

^b <https://orcid.org/0000-0003-3698-8535>

^c <https://orcid.org/0000-0002-5018-3066>

Section 3 we introduce the domain, the data set, and the data cleaning process. In Section 4, we present a descriptive analysis of the data set using process mining tools and standard statistical tools to identify informative features of the data. This informs the process of building predictive models which we describe and evaluate in Section 5. In Section 6 we discuss our results and in Section 7 we conclude.

2 RELATED WORK

Improving efficiency in surgical wards, specifically improving utilization of operating rooms, has received growing interest nationally and internationally for a number of years now (Lalys and Jannin, 2014). The National Theatres Project in Scotland states as its objective, “appropriately increasing patient throughput, thereby using resources more productively and efficiently” (Scotland, 2006). The metrics for improvement include: reducing unutilized (operating room) hours; reducing over/under-runs, late-starts, cancellations and delayed discharges; and avoiding unnecessary out-of-hours and nighttime procedures. Many of these objectives are strongly related to appropriate scheduling, and would thereby benefit from more accurate, data-informed, models of patient flows.

A significant amount of research exists in modeling processes in the surgical domain. The modeling scope of much existing work tends to fall on two ends of a spectrum in terms of granularity: the level of surgical procedures at one end and broader care flows beyond the surgical ward at the other.

In (Lalys and Jannin, 2014), 46 publications on surgical process modeling are categorized into a taxonomy ranging from the level of the surgical procedure at the lowest level of granularity, to low-level physical movements at the highest. At the latter level, which is typically concerned with robot-assisted surgery or training and assessment of surgeons, we see research on phase detection (Stauder, 2014) and detailed models of individual tool usage patterns based on sensor data (Ahmadi, 2009). Individual hand motions from video data are automatically identified in (Lin, 2006) and (Haro, 2012). A number of models based on sensor data collected during Cholecystectomies (a highly standardized procedure), were developed in (Blum, 2008), (Bouarfa and Dankelman, 2012), (Bouarfa, 2011), and (Neumuth, 2011). All of these studies have the surgical procedure at the highest level of abstraction. Our present investigation lies above this level of granularity, with only the procedure name and some other basic details

being present in the data.

Above the level of individual procedures, we see work such as (Stahl, 2006) which describes the workflow within an operating room, including anesthesia, surgery, and early recovery. Other studies also address the process surrounding surgery, from admission to recovery (Funkner, 2017), which matches the scope of our data set. Taking a view beyond the operating room is important, since activities downstream from the actual surgical procedure can interrupt patient flows as shown in the case of ICU bottlenecks in (Akkerman and Knip, 2004). Some studies have also incorporated diagnosis and follow-up after surgery such as (Mans, 2012) and (Huang, 2013).

Bayesian networks were used to model several aspects of stays in an emergency department in (Acid, 2004). While overall stay duration was one attribute included in the model, the scope was at higher level of abstraction, and not focused specifically on surgical patient flows. Furthermore, the main focus was the comparison of structure learning algorithms.

Some work has looked specifically at modeling variance in surgery durations (Strum, 2000) and incorporating this into sequencing and scheduling strategies (Denton, 2007). In (Kayis, 2012), regression modeling is employed to predict surgery duration based on clinical, operational and temporal data. Stochastic balancing of bed capacity based on fluctuating demand patterns was explored in (Cochran and Bharti, 2006) and length of stay patterns in (Akkerman and Knip, 2004). Resource allocation and patient admission was addressed in (Hulshof, 2013).

In summary, the scope of patient flows ranging from admission, through surgery to recovery, is one which has been less thoroughly addressed: most work is positioned at a lower or higher level of abstraction. In regards to the distribution of surgery times, our work has the corollary contribution of confirming previous findings. In terms of the more nuanced conditional models we present of cycle times, specifically the integration of patient clusters to Bayesian networks, we believe our approach to be novel.

3 DOMAIN & DATA PREPARATION

The Royal Infirmary of Edinburgh is the largest in Scotland, housing 900 beds and with its 24-hour accident and emergency department, providing a full range of acute medical and surgical services. The hospital IT system is integrated with the Operating Room Scheduling Office System (ORSOS), a surgery management and scheduling system.

The data set analyzed stems from the ORSOS system and involves records ranging from 2010 until 2018 inclusive. Over 1700 types of procedures are recorded in the data set with about half of cases classified as emergency cases. It is oriented around individual surgical procedures, such that any time a patient receives surgical treatment, a new entry is created and each such entry has a unique case ID. This means that the same patient may have multiple unique case IDs, potentially for the same hospital stay. Unique patient IDs, as well as electronic health record identifiers, make it possible to follow patients’ overall treatment flows, though this was out of scope of this investigation.

Data regarding patient flows are entered manually by surgical support personnel, with the system requiring the entry of timestamps for each event in the patient flow. Figure 1 illustrates the proscribed sequence of events, and also shows the authors’ aggregation of activities into logical phases (pre-op, anesthesia, surgery, recovery). The system enforces a simple linear ordering of events, though it can be overridden. If users attempt to enter timestamps out of sequence, a warning is given, but can be entered upon confirmation. Summaries of cases with anomalous entries are later sent in batches to staff for review.

Aside from the 11 timestamp attributes, the data schema contains 34 other attributes, though some are empty for many cases, such as “reason for delay”. Information regarding the procedure performed is included in two different coding schemes, one providing more detail such as location on body. Other case attributes such as the case type (emergency/scheduled), its urgency classification¹, the ASA patient status rating², and whether the patient is registered as a day-case or inpatient. Staffing details include names of the main and supervising surgeon and anesthetist as well as the consultant assigned to the case. The source of admission (emergency room, etc.), the operating room number, as well as intended and actual destination following surgery (ICU, etc.) are also included. Further details include the diabetic status of the patient, types of anesthetics administered, whether antibiotics were administered, and whether pre-session briefings and surgical pauses were held.

Cleaning and Preparation. The data set contains a number of anomalous entries, comprising roughly 10% of the 38,728 entries. These entries were re-

¹NCEPOD Classification of Intervention (NCEPOD, 2019).

²American Society of Anesthesiologists physical status classification system. (Dripps, 1963)

Table 1: Anomalous cases removed prior to analysis.

ANOMALY	COUNT	% OF TOTAL
Duplicate entries	58	0.15
Missing values	31	0.08
Dates out-of-range	475	1.23
Zero timestamps	3089	7.98
Bad ordering	443	1.44
Total	4096	10.58

moved prior to further analysis. Table 1 provides an overview.

Duplicate entries may have been due to an attempt to correct a data entry error. The column `anaesthetic_start_time` was the only timestamp column to contain `<NA>` values. A larger number of cases have clearly anomalous values in the `case_date` column, e.g. dates much too far in the past (1800) or future (3206).

Process mining techniques helped quickly reveal that despite the *de-jure* linear ordering of activities, many anomalous, and decidedly implausible, event orderings exist in the data. Figure 4 shows the result of running the SIMPLE version of the Alpha miner (Van der Aalst, 2004) from the `pm4py` package (Berti, 2019) on the top 20 sequence variants. The Alpha miner takes as input an event log and outputs a *Petri net* (specifically a *workflow net*): a type of process model. Running the Alpha miner on the entire log results in a flower model³. A further analysis of the directly-follows graph indicated that nearly all possible pairwise event orderings occurred at least once in the data.

One of the aspects of the Petri net in Figure 4 that stands out is that it permits `incision_start_time` to occur before `anaesthetic_start_time`. While this was to an extent the result of anomalous timestamps in the data, upon further inquiry with surgical staff, we learned that it is indeed legal for these activities to be recorded with the same timestamp in cases where the surgeon administers a local anesthetic.

Timestamps in the data set are rounded to the minute, and that for many cases, two or more events are recorded with the same timestamp. For example, `enter_theatre_time` and `incision_start_time` are sometimes identical, and in fact `leave_theatre_time` and `enter_recovery_time` identical for all cases. This needed to be addressed prior to applying process mining techniques, since they assume sequential orderings in event traces. For this, the *de-jure* model was used as a tiebreaker in cases of simultaneity.

³A flower model is a process model which permits any event to be executed at any stage of the process.

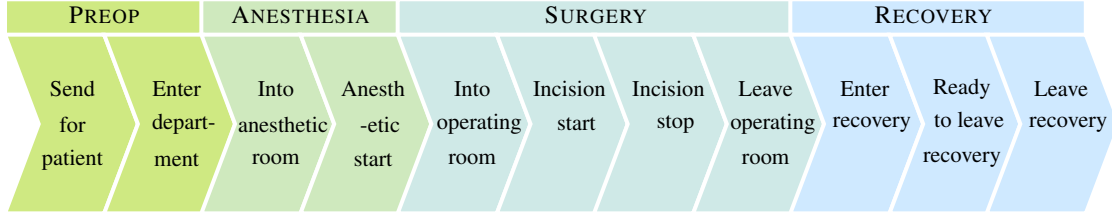


Figure 1: The patient flow proscribed by the ORSOS system. Activities are linearly ordered, but can occur “simultaneously”. That is, some activities (such as *Leave Operating Room*) can have the same timestamp as the “succeeding” activity (*Enter Recovery*), but should not occur after it.

Many of the implausible cases had zero timestamps associated with the out-of-order events (timestamps of the form YY-MM-DDT00:00:00). We suspect that these entries may be the result of users specifying only a date without a timestamp. A further 443 cases had anomalous event orderings, likely due to an incorrect entry such as failing to increment the date when a patient flow stretched from one day to the next. All cases with invalid orderings were removed prior to subsequent analysis of cycle time patterns.

Plotting event occurrences on a “dotted chart” (see Figure 2) also reveals several outlying events (occurring months or years from the rest of the flow). The dotted chart simply plots the events from an event by the case id along the y-axis and by time along the x-axis, such that events associated with the same case fall along a horizontal line.

In the remainder of the analysis, we have removed data points with cycle times in the 99th percentile of values, having observed the presence of events occurring months, even years apart, which for a single surgical case are almost certainly due to data entry mistakes. The chart also makes immediately obvious that a gap exists in the data, and gives an indication of the development in the throughput of cases over time, which remains nearly constant, perhaps increasing slightly.

The number of anomalous cases discovered despite the ORSOS system’s compliance measures, demonstrate the importance of data quality measures, especially if such data are to form the basis of prescriptive models and policies. While many of these anomalies would in principle be discoverable by manually querying the data, the use of process mining techniques helped reveal these anomalies quickly and intuitively, serving as a springboard for more detailed analysis.

4 ANALYSIS

In this section, we describe the empirical distributions of individual and aggregate cycle times, and compare how well various parametric distributions fit the data. Then, we identify the most informative features of the data set, which will be used as a starting point for model building in 5.

Marginal Cycle Time Distributions. Fitting an appropriate distribution to data can be a powerful approach to building a predictive model, despite its simplicity. These models consider only the marginal distribution, i.e. they consider outcome across all cases, without conditioning the distribution on case-specific attributes. Table 2 displays the results of fitting seven different distributions to the cycle times of both the original “low-level” events, as well as the aggregated process phases.

As indicated by goodness-of-fit statistics, aggregating individual event cycle times results in more well-formed distributions, with the one slight exception of the recovery stage. While information is clearly lost by reducing 9 cycle times to 4, this is justified by the fact that any implications of cycle times on resource utilization is captured by the aggregations. For example, an operating theatre, will have the status of being occupied and unavailable for other patients during each of the events *Into theatre*, *Incision start*, *Incision stop*, and until *Leave theatre* commences. This effect on resource availability is equivalently captured in the aggregations of these events into one *Surgery* event and its corresponding cycle time.

Previous research has indicated that surgical cycle times are log-normally distributed (Strum, 2000). Our observations are consistent with this, but it should be noted that the Kolmogorov-Smirnov goodness-of-fit does not achieve statistical significance.

Mutual Information. To get an overview of correlation between attributes, the mutual information be-

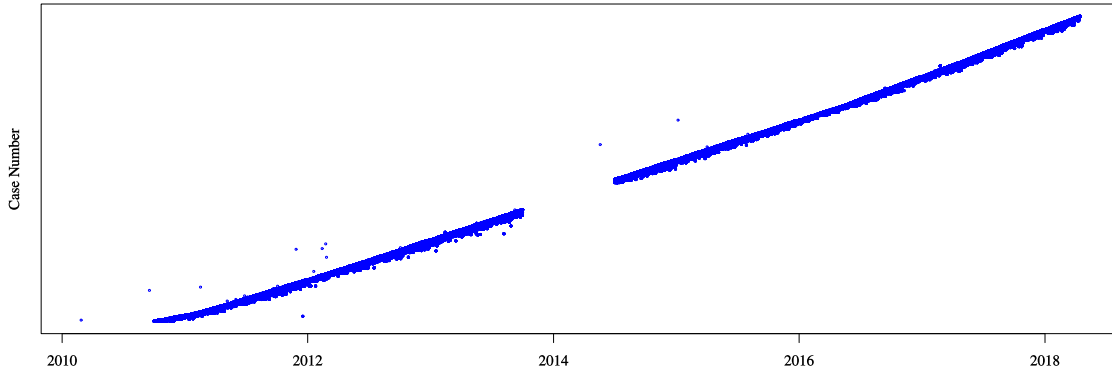


Figure 2: A “dotted chart” helps give a quick overview of the event log. Each data point represents the occurrence of an event, plotted by time on the x-axis and by a numerical case identifier on the y-axis which are incremented by time. At this level of granularity we cannot see the dynamics of individual process instances, but aspects such as arrival rate, outliers and missing values become clear. In our data set, there is a prominent gap from October 2013 to July 2014, something one would otherwise need to actively investigate, but is immediately noticeable here. Furthermore, several events lie weeks or years from the rest of the events in a case, suggesting anomalous values.

tween attributes was computed, a selection of which are visualized as heatmap in Figure 5. As a nonparametric correlation metric, mutual information is more suitable for our data than parametric estimators such as χ^2 since we cannot confidently assume normality for all attributes. Intuitively, mutual information measures the expected decrease in uncertainty regarding the outcome of y upon learning the outcome of x . Specifically, it measures the reduction in entropy of the resulting conditional probability distribution.

For continuous values, namely cycle times, it was necessary to discretize the data. This was done such that each of 7 bins contained was of equal widths such that the distribution of cases amongst bins roughly approximates their original distribution. Using this approach, correlations between attributes other than cycle times were the strongest, while attributes influencing cycle times were more weakly correlated, though still observable. Especially intended destination and source of admission stand out as informative w.r.t. cycle times. Mutual information can tend to hide important nuances since it reflects the *expected* value of the *pointwise* mutual information for individual values of a variable. Lead us to do a more detailed exploration of how different attributes influence cycle times specifically.

Conditional Cycle Time Distributions. By exploring the conditional distributions of cycle times for the individual values attributes, we were able to get a better idea of what influences cycle times. By visualizing conditional distributions on the same plot, one gets a quick impression of the whether an attribute is informative in this respect, or not. Albeit somewhat of a

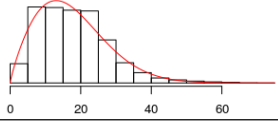
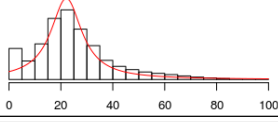
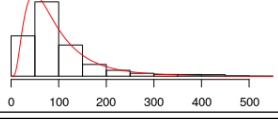
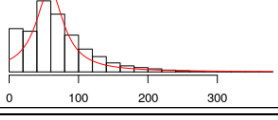
time-consuming, brute-force approach, exploring the data in this way is quite informative. This was an important factor for us in choosing which variables to include in the models we present in Section 5. See Figure 6 for examples of some of the most informative attributes.

Principle Components Analysis. Based on the intuition that cases likely fall into some sort of grouping w.r.t cycle times, we investigated the presence of clusters in the data. For example, cases with a long anesthetic cycle time may also tend to have a long surgery or recovery time - this likely being related to the procedure performed or the patient’s condition.

A visual exploration of the raw, as well as log-transformed, data gives the impression that no clear groupings exist. One method for revealing separable clusters in data that are not clearly separable in the original data is via transformation techniques such as Principle Components Analysis (PCA). PCA projects the original data onto a linear subspace which maximizes the resulting variability of the data along the resulting bases, or principle components (Bishop, 2006). It is perhaps most commonly used as a method for dimensionality reduction, by redefining the data on a subset of the principal components which capture most of the variance in the data.

Applying PCA to the log-transformed data reveals that the data does in fact fall into distinct clusters. This can be seen in Figure 3 which shows the data w.r.t to top 3 (of 4) principle components.

Table 2: Best fits for marginal distributions of cycle times. Goodness-of-fit statistic used is the Kolmogorov-Smirnov criterion.

	Gaussian	Cauchy	Logistic	Log-Normal	Gamma	Weibull	Exponential	
EVENT	GOODNESS-OF-FIT (KS)							PLOT (Best fit for aggregate)
Send for patient	0.147	0.113	0.139	0.169	0.126	0.104	0.267	
Enter department	0.161	0.147	0.197	0.205	0.171	0.157	0.184	
Pre-op	0.094	0.087	0.123	0.127	0.09	0.062	0.24	
Into anesthetic	0.226	0.166	0.168	0.153	0.133	0.15	0.19	
Anesthetic start	0.146	0.098	0.134	0.171	0.112	0.096	0.189	
Anesthetic	0.124	0.077	0.106	0.188	0.132	0.106	0.244	
Into theatre	0.16	0.094	0.143	0.111	0.093	0.114	0.298	
Incision start	0.164	0.122	0.144	0.061	0.06	0.07	0.132	
Incision stop	0.187	0.145	0.168	0.111	0.144	0.128	0.25	
Surgery	0.16	0.11	0.134	0.036	0.071	0.087	0.193	
Enter recovery	0.083	0.079	0.126	0.243	0.174	0.139	0.198	
Ready to leave	0.285	0.277	0.266	0.184	0.144	0.144	0.22	
Recovery	0.099	0.083	0.127	0.244	0.17	0.136	0.19	

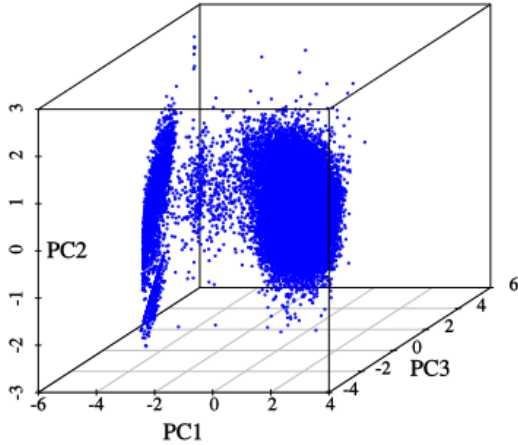


Figure 3: Top 3 principle components for (log-transformed) aggregate cycle times.

5 PREDICTION

In order to facilitate improved resource utilization through more accurate scheduling and dynamic resource allocation, we suggest using Bayesian belief networks (Koller and Friedman, 2009). The reason for this choice of model lies in its flexibility. Not

limited to one target feature, Bayesian networks can be queried on any attribute, using whatever evidence is currently available. A scheduler can pose queries concerning, for example, the probability of a surgery taking more than x minutes given the case type and condition of patient, or the likely destination of the patient given other evidence.

A Bayesian belief network is a directed acyclic graph with an associated parametrization and represents a joint probability distribution and its conditional independence relations between variables, represented as nodes. Both of these aspects, the graph structure and its parametrization need to be either hand modeled, learned automatically, or a combination of the two. We restricted this investigation to automatically learned models. This can sometimes lead to what may seem counterintuitive models, but it should be kept in mind that an edge between two nodes does not necessarily indicate a causal relationship between source and target.

We present a comparison of 3 classes of models, the latter 2 hybrid discrete/(log)-Gaussian models:

Marginal Model: an unconnected graph, equivalent to the distributions in Table 2.

10 Variable Model: 4 aggregate cycle times, ASA, CaseType, Intended Destination, Management In-

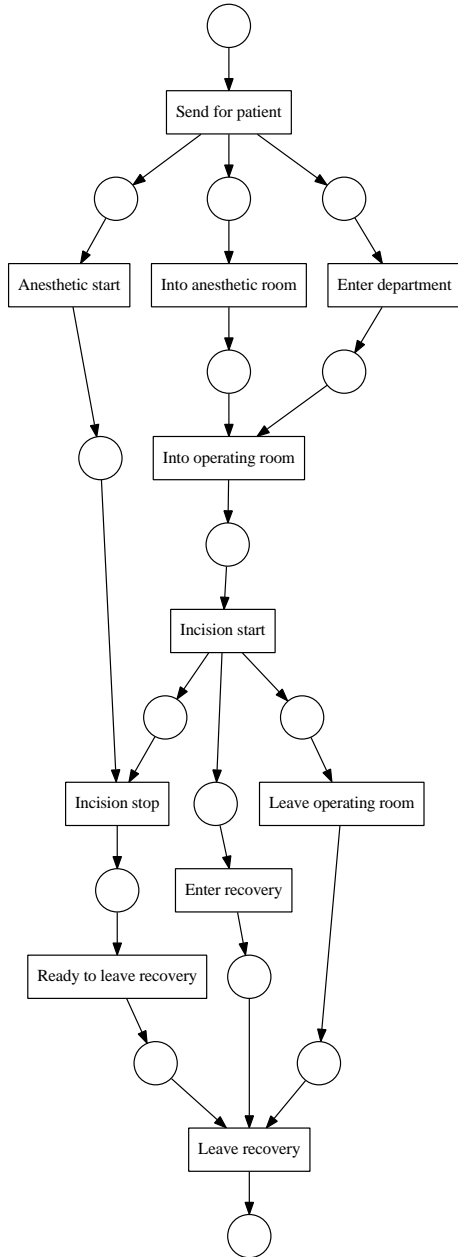


Figure 4: A Petri net generated by the Alpha miner on the top 20 trace variants observed in the event log. This model clearly allows implausible behavior, such as *Incision start* preceding *Anesthetic start*.

tent, NCEPOD Category, Source of Admission.

22 Variable Model: 4 aggregate cycle times, ASA, CaseType, Intended Destination, Management Intent, NCEPOD Category, Source of Admission, Diabetic, Operating Room, 10 Anaesthetic Type variables.

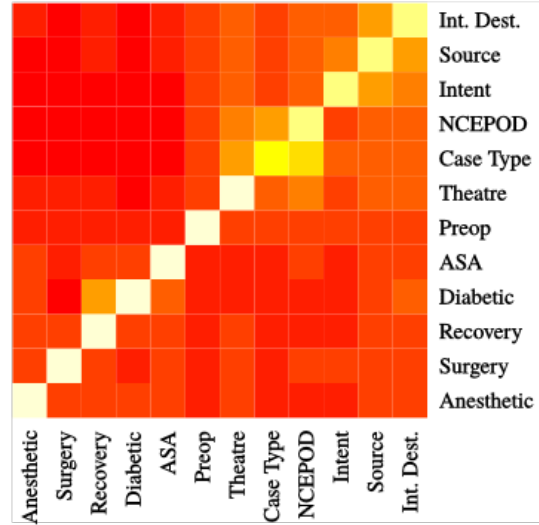


Figure 5: Heatmap of the mutual information between attributes. The “bright” spots indicate that learning the outcome of the corresponding variable on the x -axis decreases the uncertainty about the outcome of the corresponding variable on the y -axis. *Destination* denotes the intended destination following the procedure, *NCEPOD* denotes the urgency classification, and *Intent* denotes whether the case is a day-case or inpatient case.

Feature Selection. The choice of variables was based on analysis in Section 4, as well as the cardinality of variables. Variables with very large cardinality often fail to improve results due to sparse representation in the data. One solution to allow the incorporation of these is to perform dimensionality reduction on these variables prior to training the network. This is left for future work. For the largest model, we started by including all features, removing those which had no effect on performance.

Clustering. In order to explicitly incorporate the clusters observed in Section 4, we performed simple k -means clustering on the PCA transformed data and added a *Cluster* attribute to each case. This new attribute was then included as a node in some variants of the Bayesian networks. Specifically, we added 4 variants of both the 10- and 22-variable model using different numbers of clusters: 5, 10, 15, 20. For comparison a model without clusters added is evaluated as well. These values were chosen to illustrate the improvement in model performance upon adding more clusters and the eventual appearance of an elbow of diminishing improvement usually around 15 to 20 clusters. We experimented with values ranging between 2 and 40.

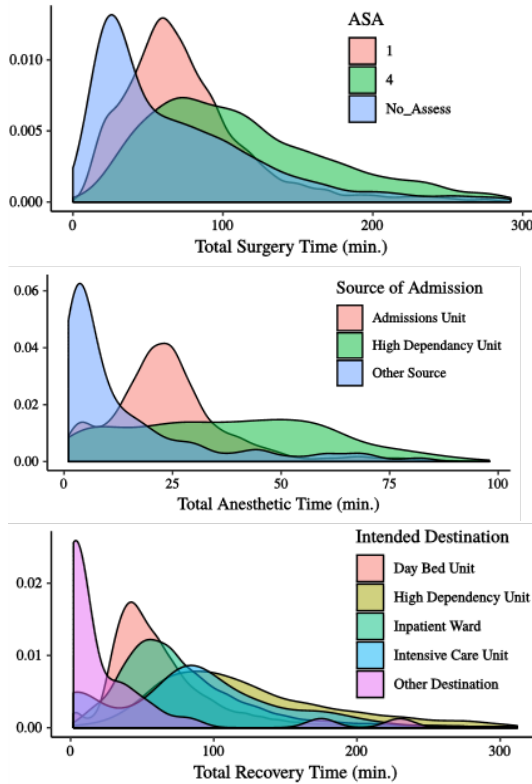


Figure 6: Examples of conditional cycle time distributions. Top: conditioned on ASA status. Middle: Source of Admission. Bottom: Intended Destination.

We found that performing clustering on the PCA transformed data gave slightly better results than clustering on the original data. Note that we used all 4 principal components, hence the data was only transformed and not reduced in dimensionality. It turns out that PCA and k -means are in fact closely linked: in (Ding and He, 2004) it is shown that PCA effectively performs clustering w.r.t. the k -means objective function. While performing PCA prior to k -means is a widespread practice, it should be noted that it does not always lead to improved results (Yeung and Ruzzo, 2001). We did observe a small improvement over performing k -means on the untransformed data.

Learning Algorithms. Structure learning was performed using score-based methods, specifically Hill Climbing and TABU search, using Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC) scores. We were unable to obtain models using log-likelihood scoring within a reasonable time. An example of the graph structure of the learned Bayesian network is shown in Figure 8. Parameter

learning was performed using the standard maximum likelihood estimation, partly due to the unavailability of Bayesian estimation techniques for hybrid models in the chosen inference library.

Smoothing & Priors. Simple smoothing was applied to avoid zero probabilities for outcomes not observed in the training data. This was done by simply adding 0.01% to all probabilities and subsequently re-normalizing. For continuous nodes with discrete parents, the marginal distribution was assigned in case a combination of the parents’ values was observed in the training data.

Evaluation. Evaluation of Bayesian networks is often based on quantifying how closely the probability distribution represented by the network matches the empirical distribution (data). Typical metrics include log likelihood, Akaike information criterion, Bayesian information criterion, and Kullback-Liebler divergence.

One can also consider a specific target variable for prediction and measure the error rate. Since we are specifically interested in predicting cycle times, we report results of the mean absolute error of predictions for these 4 target variables. We chose to report this metric rather than the more standard (root) mean squared error (RMSE), since it gives a more immediate sense of how far predictions were from actual cycle times in terms of the original time units (minutes). Results for RMSE follow very nearly the same pattern between models. Results are reported for 5 runs of 10-fold cross-validation, see Figure 7.

A crucial point regarding our approach to cross validation concerns which attributes were considered observed at each phase of the process. When predicting a given cycle time, clearly any future cycle times will not have been observed and should not be included as inputs to the model. So when predicting the *Preop* cycle time, *Anesthesia*, *Surgery*, and *Recovery* should not be including as inputs. However, once the patient has reached the *Recovery* phase, the preceding cycle times are at least *theoretically* known and can be used as inputs. This was our approach, motivated by the assumption that even if patient flow monitoring systems do not presently integrate such real-time information they will likely do so in the near future.

6 DISCUSSION

Our preliminary analysis of the ORSOS data set has demonstrated three points. First, the importance of data quality assurance, cleaning and the usefulness

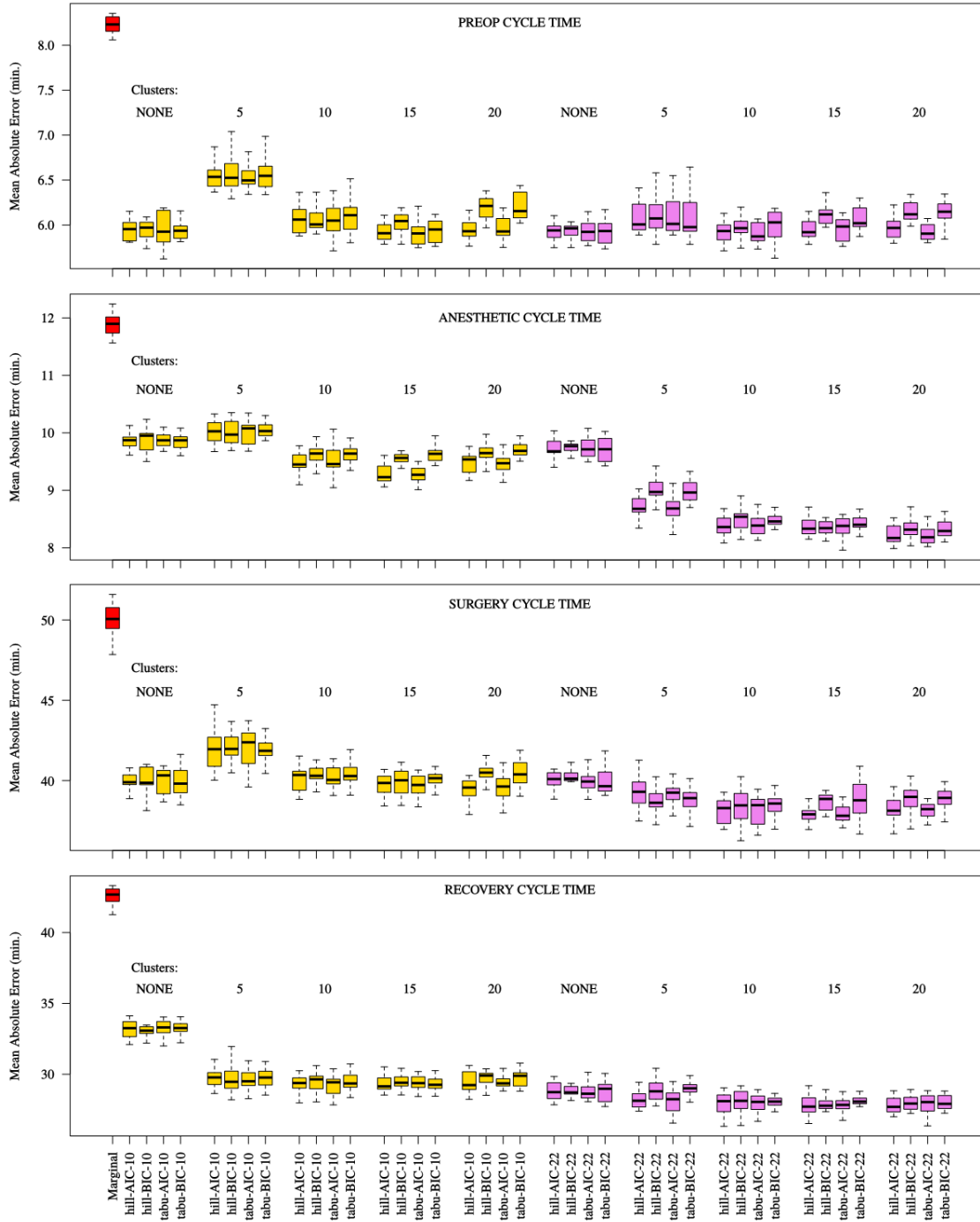


Figure 7: 5 runs of 10-fold cross-validation based on mean absolute error on the 4 cycle time target variables: preop, anesthetic, surgery, and recovery. Red: Marginal baseline model. Gold: 10-variable model. Violet: 22-variable model. Within each boxplot grouping are results for models learned with the Hill Climbing and TABU structure learning algorithms using Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) respectively.

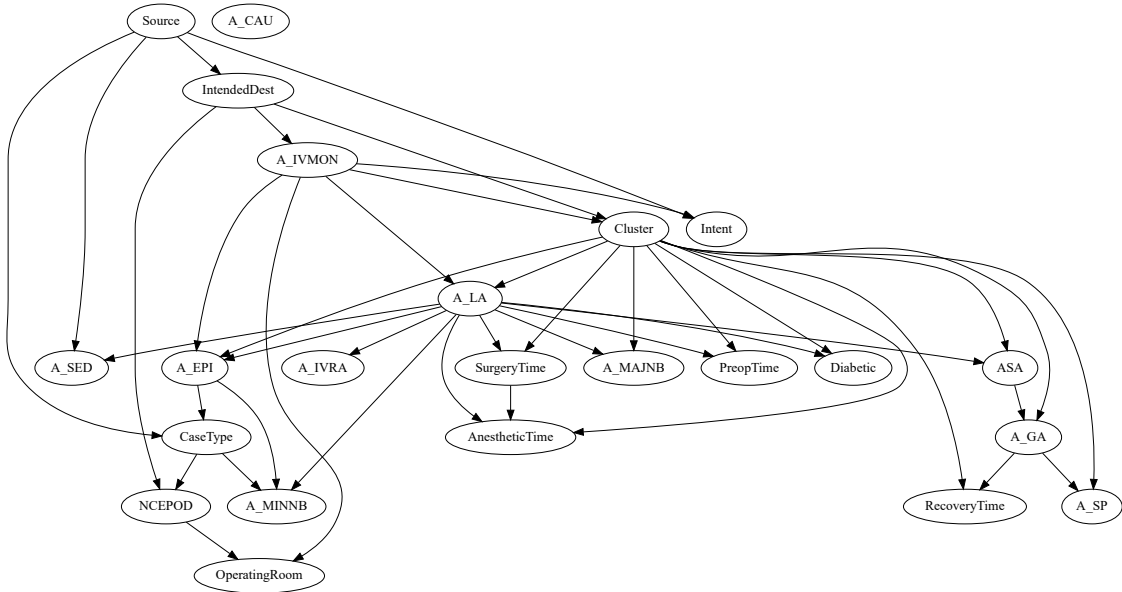


Figure 8: The overall best performing model. The 22 variable model using TABU with BIC scoring and 20 clusters. Node labels prefixed by *Ane.* denote anaesthetic types. *NCEPOD* indicates urgency classification, *ASA* patient condition and *Intent* indicates day-case/inpatient.

of process mining techniques in this respect. Second, that a reasonably accurate predictive model of event cycle times in the form of a simple Bayesian belief network can be built which significantly outperforms simple marginal distribution fitting. Third, that by clustering the target variables and including these cluster labels as attribute in the model and training data improves accuracy yet further. Furthermore, we describe how mutual information, tools for exploring conditional probability distributions and principle components analysis can not only give insight into the data, but also guide model building

The choice of Bayesian networks was motivated by their flexibility and interpretability. The fact that they can be queried in such a versatile manner, based on whatever data is available at the time, suggests they would be a strong component of a predictive model in a decision support and scheduling systems in surgery. This allows for queries of the form, “what is the probability that case c will be in surgery for more than m minutes given it has the following attributes, and took n minutes to complete anesthesia?”. Specifically, these could form the basis for probabilistic scheduling systems.

7 CONCLUSION & FUTURE WORK

We have demonstrated the utility of combining several data analysis tools, including from process mining and machine learning, to begin building a useful model of a very complex set of processes in a surgical ward. This approach would be also applicable in other areas of the healthcare system in which under-utilization of expensive resources calls for precise scheduling to avoid down-time.

In terms of the full data analytics workflow, often summarized by *Describe* \rightarrow *Diagnose* \rightarrow *Predict* \rightarrow *Prescribe*, we have only just begun the *Predict* phase. The incorporation of more aspects of the data set is a clear next step - the huge cardinality of some attributes, such as procedures and staff, should be addressed by incorporating domain knowledge and/or dimensionality reduction. Considering the notable improvements in precision we achieved with a relatively limited data set, it is likely that incorporating patient flow attributes across domains would lead to yet more precise models.

A thorough comparison with learning algorithms other than simple distribution fitting to confirm the suitability of Bayesian networks to this application is also important. Finally, while we have focused on individual patient flows here, more comprehensive

models which take into account ward level dynamics such as patient arrival rates, resource constraints, and resulting inter-patient dynamics are a natural extension. In such a system-wide model, the work presented here would serve as a component to more accurately model local event timings and subsequent downstream arrival rates.

REFERENCES

- Acid, S., e. a. (2004). A comparison of learning algorithms for bayesian networks: a case study based on data from an emergency medical service. *Artificial intelligence in medicine*, 30(3):215–232.
- Ahmadi, S.A., e. a. (2009). Motif discovery in or sensor data with application to surgical workflow analysis and activity detection. In *M2CAI workshop, MICCAI, London*. Citeseer.
- Akkerman, R. and Knip, M. (2004). Reallocation of beds to reduce waiting time for cardiac surgery. *Health care management science*, 7(2):119–126.
- Berti, A., e. a. (2019). Process Mining for Python (PM4Py): Bridging the Gap Between Process-and Data Science. In *ICPM Demo Track (CEUR 2374)*.
- Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer Science+ Business Media.
- Blum, T., e. a. (2008). Workflow mining for visualization and analysis of surgeries. *Int. journal of computer assisted radiology and surgery*, 3(5):379–386.
- Bouarfa, L., e. a. (2011). Discovery of high-level tasks in the operating room. *Journal of biomedical informatics*, 44(3):455–462.
- Bouarfa, L. and Dankelman, J. (2012). Workflow mining and outlier detection from clinical activity logs. *Journal of biomedical informatics*, 45(6):1185–1190.
- Cochran, J. K. and Bharti, A. (2006). Stochastic bed balancing of an obstetrics hospital. *Health care management science*, 9(1):31–45.
- Denton, B., e. a. (2007). Optimization of surgery sequencing and scheduling decisions under uncertainty. *Health care management science*, 10(1):13–24.
- Ding, C. and He, X. (2004). K-means clustering via principal component analysis. In *Proc. of the twenty-first int. conference on Machine learning*, page 29. ACM.
- Dripps, R. (1963). American society of anesthesiologists. *New classification of physical status*. *Anesthesiology*, 24(1):111.
- Funkner, A. A., e. a. (2017). Towards evolutionary discovery of typical clinical pathways in electronic health records. *Procedia computer science*, 119:234–244.
- Haro, B.B., e. a. (2012). Surgical gesture classification from video data. In *Int. Conf. on Medical Image Computing and Computer-Assisted Intervention*, pages 34–41. Springer.
- Huang, Z., e. a. (2013). Summarizing clinical pathways from event logs. *Journal of biomedical informatics*, 46(1):111–127.
- Hulshof, P. J. H., e. a. (2013). Tactical resource allocation and elective patient admission planning in care processes. *Health care management science*, 16(2):152–166.
- Kayis, E., e. a. (2012). Improving prediction of surgery duration using operational and temporal factors. In *AMIA Annual Symposium Proc.*, volume 2012, page 456. American Medical Informatics Association.
- Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.
- Lalys, F. and Jannin, P. (2014). Surgical process modelling: a review. *Int. journal of computer assisted radiology and surgery*, 9(3):495–511.
- Lin, H.C., e. a. (2006). Towards automatic skill evaluation: Detection and segmentation of robot-assisted surgical motions. *Computer Aided Surgery*, 11(5):220–230.
- Mans, R., e. a. (2012). Mining processes in dentistry. In *Proc. of the 2nd ACM SIGHT Int. Health Informatics Symposium*, pages 379–388. ACM.
- NCEPOD (2019). NCEPOD classification of intervention. <https://www.ncepod.org.uk/classification.html>. Accessed: 2019-11-22.
- Neumuth, T., e. a. (2011). Analysis of surgical intervention populations using generic surgical process models. *Int. Journal of Computer Assisted Radiology and Surgery*, 6(1):59–71.
- Scotland, N. (2006). National theatres project report. <https://www.isdscotland.org/Health-Topics/Quality-Indicators/National-Benchmarking-Project/National-Theatres-Project/>. Accessed: 2019-11-22.
- Stahl, J. E., e. a. (2006). Reorganizing patient care and workflow in the operating room: a cost-effectiveness study. *Surgery*, 139(6):717–728.
- Stauder, R., e. a. (2014). Random forests for phase detection in surgical workflow analysis. In *Int. Conf. on Information Processing in Computer-Assisted Interventions*, pages 148–157. Springer.
- Strum, D., e. a. (2000). Modeling the uncertainty of surgical procedure times: comparison of log-normal and normal models. *Anesthesiology*, 92(4):1160–1167.
- Van der Aalst, W. e. a. (2004). Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142.
- Yeung, K. and Ruzzo, W. L. (2001). An empirical study on principal component analysis for clustering gene expression data. *Bioinformatics*, 17(9):763–774.

Stochastic Workflow Modeling in a Surgical Ward: Towards Simulating and Predicting Patient Flow

Christoffer O. Back¹[0000–0001–7998–7167], Areti Manataki²[0000–0003–3698–8535],
Angelos Papanastasiou³[0000–0002–5649–455], and Ewen Harrison²[0000–0002–5018–3066]

¹ Department of Computer Science, University of Copenhagen, Denmark back@di.ku.dk

² Usher Institute, University of Edinburgh, U.K.
a.manataki@ed.ac.uk, mail@ewenharrison.com

³ School of Informatics, University of Edinburgh s1983122@ed.ac.uk

Abstract. Intelligent systems play an increasingly central role in healthcare systems worldwide. Nonetheless, operational friction represents an obstacle to full utilization of scarce resources and improvement of service standards. In this paper we address the challenge of developing data-driven models of complex workflow systems - a prerequisite for harnessing intelligent technologies for workflow improvement. We present a proof-of-concept model parametrized using real-world data and constructed based on domain knowledge from the Royal Infirmary of Edinburgh, demonstrating how off-the-shelf process mining, machine learning and stochastic process modeling tools can be combined to build predictive models that capture complex control flow, constraints, policies and guidelines.

Keywords: Surgery · Surgical Workflow · Bayesian Network · Petri Nets · Simulation · Data Mining · Patient Flow · Process Mining

Acknowledgements We would like to thank Cameron Fairfield and Stephen Knight for their generous feedback regarding policies and on-the-ground practices at the Royal Infirmary of Edinburgh surgical ward.

1 Introduction

Surgical care is a key component of healthcare systems worldwide, saving and improving thousands of lives every day. Over 10 million operations are performed each year in England [34], including high-risk cases and patients that require immediate life, limb or organ-saving interventions. Surgical care is also very costly, with more than \$400 billion spent each year in the United States on operative procedures [1]. The number of people requiring surgery is rising every year, often leading to long waiting times that may put patients at risk.

Ensuring efficiency, timeliness and safety are crucial for providing high-quality service while controlling costs [26], [16]. While many processes surrounding surgery are well structured, the dynamic nature of patient arrivals combined with the complexity of coordinating large numbers of specialized staff and facilities, means that delays and misalignments can have cascading effects leading to last-minute cancellations and

under-utilization of expensive resources. There is, hence, an imperative need to improve surgical workflow. Some key questions here are: How can we improve overall surgical care performance in the most cost-effective way? How can we plan surgical care in a way that it is tailored to the individual patient?

There is a wealth of data being collected through hospital IT systems, which can be used towards answering these questions. This includes operating room management and usage data, electronic health records and surgery cancellation data. By adopting a process-based approach, one can make sense of such complex and big data and inform improvements in surgical care processes, including intelligent surgery planning, staff scheduling and workflow management.

This paper extends previous work [6] by presenting a preliminary investigation into stochastic workflow modeling and verification methods in surgical wards, with outset in a data set following patients from admission to discharge at the Royal Infirmary of Edinburgh in Scotland. With the aim of gaining a comprehensive understanding of surgical workflow, we use the data to investigate both *system-wide* surgical performance and *individual* patient flow. Results from these two types of modeling can be combined to enable personalised and efficient surgical scheduling.

In particular, we discuss how process mining methods can be used to gain insights regarding control-flow and temporal patterns in the surgical ward. Focusing on system-wide performance and recognizing the high level of uncertainty in the surgical department, we demonstrate how Stochastic Time Petri Nets can be used to effectively capture complex hospital policies and constraints. The choice of Stochastic Time Petri Nets allows for simulation of different scenarios, thus enabling what-if analysis. This is key for investigating different, and often competing, workflow improvement mechanisms. Focusing on individual patient flow, we propose the use of Bayesian Networks to predict patient-specific cycle times of individual surgical phases, from the time patients are sent for, through anesthesia and surgery, and until they leave recovery. Aside from their capacity to easily incorporate domain knowledge, Bayesian networks have the advantage that they can be queried in complex ways even with incomplete evidence, which is invaluable in the uncertain hospital environment. We present and compare three probabilistic models and we evaluate them w.r.t. to prediction accuracy. Crucially, we show that by incorporating a pre-processing step based on simple clustering of flows w.r.t. cycle times, we can improve the performance of our models noticeably.

The rest of the paper is structured as follows. In Section 2 we review existing literature. Our subsequent analysis of the data follows the classic data analytics workflow of *Describe* \rightarrow *Diagnose* \rightarrow *Predict*. In Section 3 we introduce the domain, the data set, and the data cleaning process. In Section 4, we present a descriptive analysis of the data set using process mining and standard statistical tools to identify control-flow and temporal patterns in the data. This informs the process of building system-wide simulation models and individual-patient predictive models, which we describe and evaluate in Section 5. In Section 6 we discuss our results and in Section 7 we conclude and discuss directions for future work.

2 Related Work

The modeling of surgical workflows has received a significant amount of attention by researchers, motivated by the prerogative to improve efficiency and resource utilisation while ensuring adherence to service standards.

Of particular interest for the present case study is the National Theatres Project in Scotland which outlines several areas for improvement that might be addressed by workflow optimization. This includes “appropriately increasing patient throughput, thereby using resources more productively and efficiently” by reducing unutilized (operating room) hours; reducing over/under-runs, late-starts, cancellations and delayed discharges; and avoiding unnecessary out-of-hours and nighttime procedures [32].

Previous research on modeling surgical ward processes varies greatly in terms of scope: from very fine-grained models of individual procedures to high-level models of treatment pathways well beyond the context of the surgical ward itself (e.g. from visit to a GP to follow-up evaluation and treatment). In their literature review on the topic [26] Laylis and Jannin identify a range of granularity. At the finest level are low-level physical movements such as tool usage patterns based on sensor data [4], phase detection [37], automatic identification of hand motions from video in [27] and [20]. Several investigations have been made into the modelling of Cholecystectomies, a highly standardized procedure [10,11,12,31].

In [36] the authors go beyond the modeling of the surgical procedure to include anesthesia and early recovery within the operating theatre, while [19] considers the patient flow from admission to recovery. Activities downstream from surgery, namely recovery in ICU wards can present a key bottleneck, as addressed in [5]. Extending the patient pathway further, follow-up post-surgery is incorporated in [28,21]. In general, however, most research appears to have focused primarily on either very low-level procedure or high-level treatment pathways. The patient flows we consider fall in between these levels of granularity.

The use of Bayesian networks to model stays in an emergency department is evaluated in [3]. In contrast to our approach, the view of patient flows is at a higher level of abstraction, and the main focus is the comparison of structure learning algorithms.

Modeling the duration of surgical procedures was investigated in [38,24] and we are able to report findings in line with these regarding the log-normal distribution of surgical times. Surgical duration was incorporated into sequencing and scheduling strategies in [16]. Stochastic balancing of bed capacity based on fluctuating demand patterns was explored in [15] and length of stay patterns in [5] while resource allocation and patient admission was addressed in [22].

More broadly, the problem of ensuring that systems fulfill a given set of specifications has been widely addressed in model checking and process mining research. The systems under consideration can range from electronic circuits and communication protocols [33,29], to business processes [23,14] and even biological systems [17].

We can verify whether a system satisfies a property or specification by means of state-space exploration [40] or rewrite rules [8], but often realistic models reach a level of complexity that precludes closed-form verification. This leaves simulation, essentially a random sampling of the model’s state space, as a next best tool for verification and what-if analysis.

Incorporating desired constraints into a system model is straightforward [7], and allows for correct-by-construction plan generation, but also leads to state-space explosion. Advancements in seemingly unrelated areas such as robot motion [18] provide evidence that this approach to intelligent planning is feasible, even in complex domains.

3 Domain and Data Preparation

We were granted access to workflow data recorded at The Royal Infirmary of Edinburgh in connection with cases taking place from 2010 until 2018. The infirmary is Scotland’s largest, with 900 beds and a 24-hour accident and emergency department. The data at our disposal was recorded by the Operating Room Scheduling Office System (ORSOS), which is one component in the institution’s overall IT-infrastructure.

Over 1700 types of procedures are recorded in the data set, about half of which are classified as emergency cases. Each treatment procedure is given a unique case ID, meaning that the same patient may be associated with multiple case IDs, even during the same stay for inpatients. Following patients’ broader treatment patterns would be possible using this dataset, but lies beyond the scope and focus of this paper.

Data is entered manually by surgical support personnel, with the system requiring the entry of timestamps for each event in the patient flow. Figure 1 illustrates the proscribed sequence of events, along with an aggregation of activities into logical phases (pre-op, anesthesia, surgery, recovery).

The system attempts to enforce a strict linear ordering of events, though this can be overridden by personnel. If a timestamp is entered out of sequence, a warning is given, but can be entered upon confirmation. Staff are then sent a summary of anomalous cases for review at the end of the week.

Data Schema In addition to timestamps for the 11 proscribed activities in a patient flow, 34 other attributes are recorded. Attributes of note include two different procedure coding schemes, case type (emergency/scheduled, day-case/inpatient), NCEPOD urgency classification⁴, and the ASA patient condition rating.

Some staffing details are also included, such as main and supervising surgeon and anesthetist, as well as the consultant assigned to the case. The source of admission (emergency room, etc.), as well as intended and actual destination following surgery (ICU, etc.) and crucially, the operating room number, are also included. Further details include the diabetic status of the patient, types of anesthetics administered, whether antibiotics were administered, and whether pre-session briefings and surgical pauses were held.

Cleaning and Preparation A number of clearly anomalous entries are present in the dataset, comprising roughly 10% of the 38,728 entries. Due to the relatively small percentage of anomalies and the reasonably large dataset, we followed a precautionary principle and simply removed entire cases containing anomalous entries prior to further analysis and modeling. Table 1 provides an overview.

⁴ NCEPOD Classification of Intervention [30].

ANOMALY	COUNT	% OF TOTAL
Duplicate entries	58	0.15
Missing values	31	0.08
Dates out-of-range	475	1.23
Zero timestamps	3089	7.98
Bad ordering	443	1.44
Total	4096	10.58

Table 1: Anomalous cases removed prior to analysis. Originally published in [6].

Duplicate entries may have been due to an attempt to correct a data entry error, but we are unable to determine which entry is reliable. The column *anaesthetic start time* was the only timestamp column to contain NA values. A larger number of cases have clearly anomalous values in the case date column, e.g. dates much too far in the past (year 1800) or future (year 3206).

4 Analysis

4.1 Control Flow Patterns

Based on input from domain experts, we were aware of the *de jure* workflow, which follows a simple linear flow of events as illustrated in Figure 1. In addition to the anomalies discussed in Section 3, process mining techniques helped reveal further control-flow deviations, guiding the data cleaning process. In particular, we found dotted charts, directly-follows graphs and mined Petri Nets to be particularly informative.

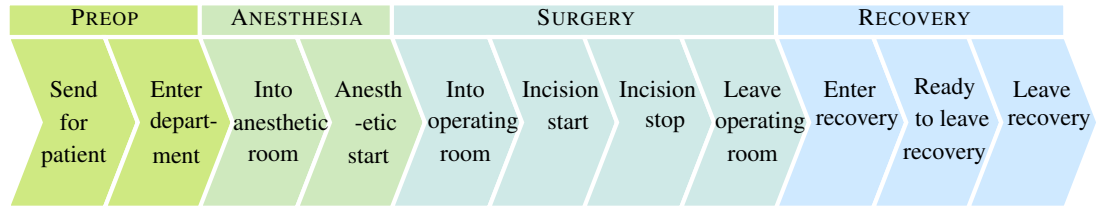


Fig. 1: The *de jure* sequence of events recorded by the ORSOS system, representing the intended patient flow. Activities are linearly ordered, but can occur “simultaneously”. That is, some activities (such as *Leave Operating Room*) can have the same timestamp as the “succeeding” activity (*Enter Recovery*), but should not occur after it. Originally published in [6].

Dotted Chart One simple yet powerful tool for getting a quick, preliminary overview of process-related data is the dotted chart, which simply charts events w.r.t. case-id across

time such that dots falling along a horizontal line represent events belonging to the same process instance (i.e. case).

Using the dotted chart in Figure 2 we immediately identified a substantial gap in the dataset. Furthermore, we can see that some cases have events occurring many months, even years apart - almost certainly evidence of anomalous entries.

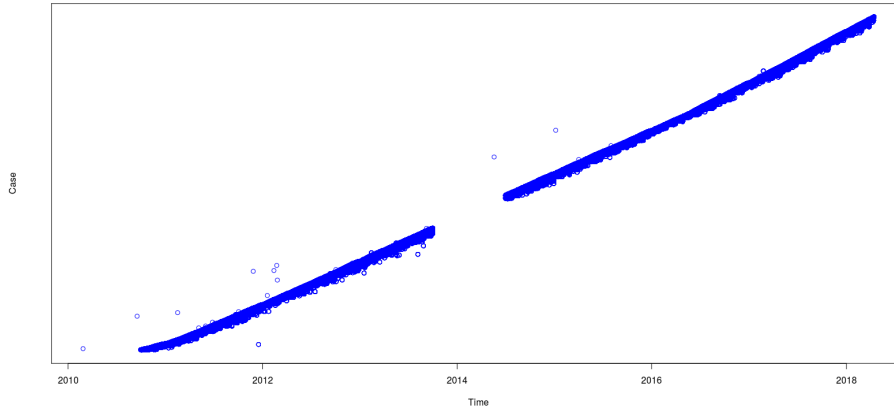


Fig. 2: A dotted chart showing all events in our dataset, arranged according to case id and timestamp. Originally published in [6].

Directly-Follows Graph Another simple visualization tool, directly-follows graphs consist of nodes and directed edges, where nodes represent the events in the log, and an edge exists between two nodes if there is at least one log trace where the source event is followed by the target event. Figure 3 shows the directly-follows graph obtained for our dataset, which includes node and edge frequencies. On one hand, the event frequencies on the graph confirmed that all events were included in each trace, in accordance with the *de jure* workflow. On the other hand, the graph indicated that nearly all possible pairwise event orderings occurred at least once in the data. This is inconsistent with the *de jure* workflow, and it includes several implausible event orderings. For example, there were a remarkable 154 traces where the last event in the *de jure* workflow, namely *leave_recovery_time*, occurs before the proscribed first event, namely *sent_for_time*.

Alpha Miner For a more nuanced view of the control-flow evidenced by the event log, proper process mining algorithms can be used. The Alpha (α) miner was one of the first process mining algorithms developed, and though it has limitations regarding the variety of control constructs it is able to identify, for our purposes it provided interesting insights into course of events as evidenced by the data.

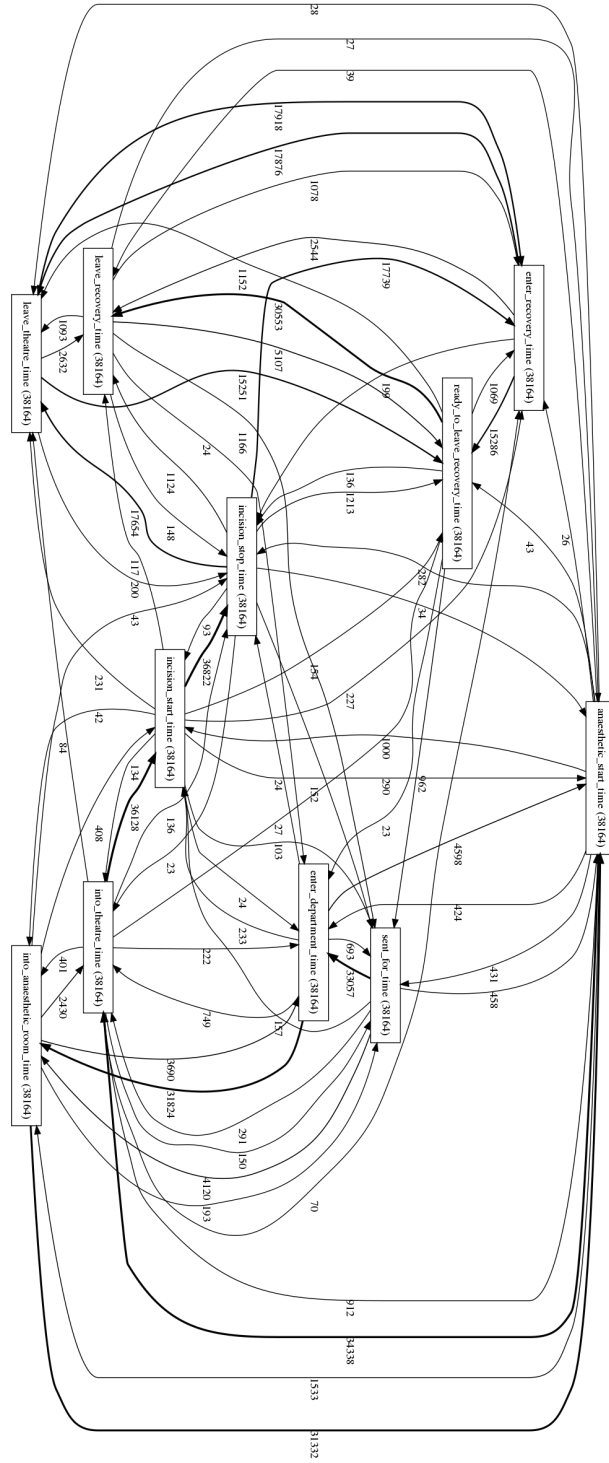


Fig. 3: Directly-follows graph indicating that nearly all possible pairwise event orderings occurred at least once in the data.

Figure 4 shows the result of running the SIMPLE version of the Alpha miner[2] from the pm4py package [9] on the top 20 sequence variants in the log. Mining on the entire log produces a flower model - a model which permits any behavior, in line with observation from the directly-follows graph in Figure 3.

According to this model, several remarkable control patterns seem to be evidenced by the most frequently occurring sequence variants. For example, according to Figure 4, *anesthetic_start* is not a precondition for *incision_start*. This observation led us to inquire with experts at the infirmary and to more closely investigate these cases in the dataset. Apparently, it was not uncommon for these two events to have exactly the same timestamp: a reflection, for example, of cases in which a surgeon administers a local anesthetic immediately prior to a minor surgery.

This observation gives rise to a further insight: nearly all process mining algorithms have a strong assumption of temporal monotonicity, i.e. events are strictly linearly ordered such that no two events share *exactly* the same timestamp. With the coarse level of temporal accuracy (1-5 minutes) in our dataset, exactly co-occurring events were common. In this sense, most process mining algorithms are unable to account for *true* concurrency in data.

4.2 Temporal Patterns

Beyond identifying anomalies in the data, there were few interesting control-flow patterns to identify at the level of patient flows, since they do in fact follow a (non-monotonic) linear process.

This left temporal patterns as the next obvious aspect to investigate, especially

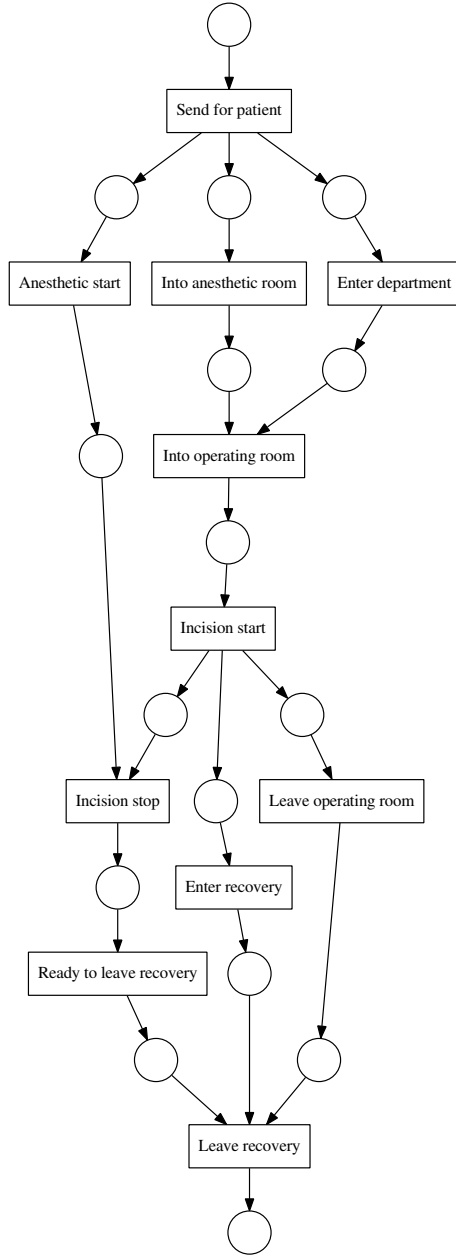


Fig. 4: Petri net generated by the Alpha miner on the top 20 trace variants. Originally published in [6].

since both resource usage and service guidelines are largely temporally focused (e.g. target time to theatre, anesthetist availability).

Event Aggregation In part due to large numbers of zero-duration cycle-times due to the phenomenon of co-occurring events, but also based on conversations with experts, we decided to group individual events into the four phases illustrated in Figure 1. Aside from clearly representing logically meaningful phases, it was also clear that this aggregation smoothed out cycle-time distributions.

On one hand, this process of aggregation arguably *removes* valuable information that could inform our model, on the other, it constitutes a form of dimensionality reduction which helps control the complexity of our model and ultimately improves performance in the end.

Marginal Distributions The simplest temporal pattern at the level of patient flows is the marginal distribution of cycle times across all patients regardless of procedure, condition, urgency, etc. Fitting a probability distribution to the empirical distributions of cycle time also constitutes the simplest possible predictive model, i.e. the maximum likelihood prediction based on the best-fit distribution.

In Table 2 we show the goodness-of-fit statistics (Kolmogorov-Smirnov criterion) for 7 types of distributions for both the individual events and events aggregated into phases. Those distributions best fit to the phase data are visually depicted.

In building the stochastic Petri net described in Section 5, we used these marginal distributions to parametrize transitions representing these phases. However, our modeling tool restricted the families of probability distributions to Exponential, Erlang and polynomials of exponentials. We illustrate our approximations to the best-fit distributions in Table 2 and give the exact parametrizations in Table 5.

Mutual Information To get an impression of which attributes might be informative independent variables in conditional distributions of cycle times, we calculated estimates of pairwise mutual information. Having an eye to identifying variables for inclusion in the Bayesian networks described in Section 5.2, we were interested in mutual information between all attributes.

As a measure of the expected decrease in uncertainty regarding the outcome of variable X upon learning the outcome of Y , mutual information is akin to standard correlation metrics, but well suited to hybrid (discrete/continuous) attributes and makes no assumption regarding normality or linearity.

Recalling the definition of the Shannon entropy of a random variable X as the its expected information content, denoted $H[X]$, we can write mutual information directly as the decrease in entropy of X upon learning the outcome of Y . Formally, $I(X;Y) = H[X] - H[X|Y]$. Two completely independent variables will have mutual information of $H[X] - H[X] = 0$, while for two perfectly correlated variables it collapses to the entropy of the dependent variable $H[X] - 0 = H[X]$.

However, as an expected value (averaged over the sample space), it can hide that specific outcomes for a variable can have a high *pointwise mutual information* - which could be harnessed by a predictive model - yet disappear amongst many uninformative

	Gaussian	Cauchy	Logistic	Log-Normal	Gamma	Weibull	Exponential	
EVENT	GOODNESS-OF-FIT (KS)							PLOT (Best fit for aggregate)
Send for patient	0.147	0.113	0.139	0.169	0.126	0.104	0.267	
Enter department	0.161	0.147	0.197	0.205	0.171	0.157	0.184	
Pre-op	0.094	0.087	0.123	0.127	0.09	0.062	0.24	
Into anesthetic	0.226	0.166	0.168	0.153	0.133	0.15	0.19	
Anesthetic start	0.146	0.098	0.134	0.171	0.112	0.096	0.189	
Anesthetic	0.124	0.077	0.106	0.188	0.132	0.106	0.244	
Into theatre	0.16	0.094	0.143	0.111	0.093	0.114	0.298	
Incision start	0.164	0.122	0.144	0.061	0.06	0.07	0.132	
Incision stop	0.187	0.145	0.168	0.111	0.144	0.128	0.25	
Surgery	0.16	0.11	0.134	0.036	0.071	0.087	0.193	
Enter recovery	0.083	0.079	0.126	0.243	0.174	0.139	0.198	
Ready to leave	0.285	0.277	0.266	0.184	0.144	0.144	0.22	
Recovery	0.099	0.083	0.127	0.244	0.17	0.136	0.19	

Table 2: *Red*: Best fits for marginal distributions of cycle times, goodness-of-fit statistic used is the Kolmogorov-Smirnov criterion. *Blue*: distributions used for modelling, in which our tool restricted the choice of distributions to Erlang (pre-op) and polynomials of exponentials (remaining). Originally published in [6].

outcomes. For this reason we also manually explored conditional distributions for cycle times.

Conditional Cycle Time Distributions Our investigation around the most informative features in the data set continued by exploring the conditional distributions of cycle times for the individual values attributes. By visualizing conditional distributions on the same plot, one gets a quick impression of whether an attribute is informative in this respect, or not. Even though this is a somewhat time-consuming, brute-force approach, exploring the data in this way turned out to be quite informative. This analysis played an important role for us in choosing which variables to include in the models we present in Section 5.2. Examples of some of the most informative attributes are presented in Figure 5. For instance, one can see that the conditional cycle time distributions for surgery differ considerably based on ASA status, i.e. for normal healthy patients (ASA status 1), for patients with severe systemic disease that is a constant threat to life (ASA status 4) and for patients with non-assessed ASA status. The anesthetic cycle time distributions

conditioned on source of admission also differ considerably, with patient cases coming from the High Dependency Unit spending longer on average in Anesthesia, compared to those coming from the Admissions Unit or some other source.

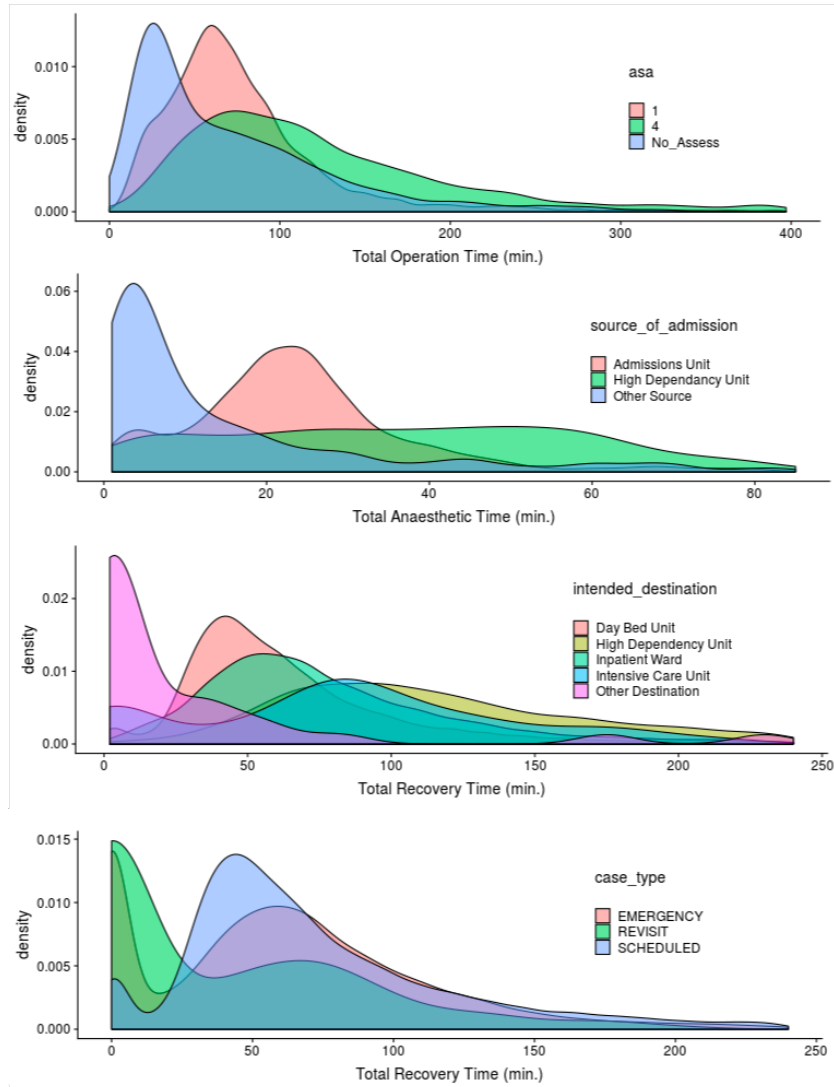


Fig. 5: Examples of conditional cycle time distributions. Top: conditioned on ASA status. Second from top: Source of Admission. Third from top: Intended Destination. Bottom: Case Type.

Principal Component Analysis & Patient Clusters We explored the presence of groupings of patients in regards to duration by a combination of visual analysis, data transformation and clustering.

Judging by the original durations of the 4 phases of a patient’s flow, there do not appear to be clear groupings of patients (Figure 6a). However, after applying principal components analysis (PCA) and plotting the data w.r.t. the four principal components, clear groupings become apparent (Figure 6b). Since PCA assumes normally distributed data, and since most durations more closely follow a log-normal distribution, the data was log-transformed prior to PCA transformation.

Afterwards, *k*-means clustering was used to discover grouping of patients. This derived attribute was added to the dataset and our predictive models, noticeably improving performance. It should be noted that we retained all 4 principal components, and thus employed PCA solely as a *transformation* rather than *dimensionality reduction* technique, as is common. This was due to the observation that removing those principal with lowest eigenvalues did not improve performance. This is not unexpected, considering the small number of dimensions.

4.3 Arrival Rates

Many of the aspects of patient flows we have considered so far concern patterns at the level of the individual patient. In order to model system-level dynamics it is crucial to consider how the system is affected by multiple processes competing for shared resources.

One key component in this analysis concerns the arrival of patients, in particular unplanned arrivals requiring immediate treatment, since this will affect and potentially interfere other patient flows. A clearly defined policy exists for the prioritisation of cases based on severity which can lead to cancellation of procedures.

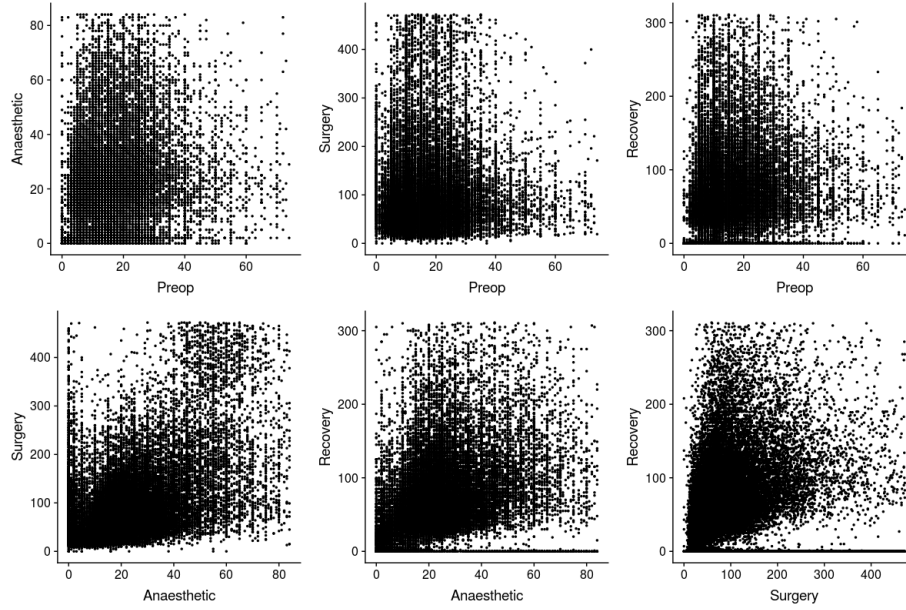
A common assumption in performance modelling and queuing theory is that the number cases arriving for service within some interval follows a Poisson distribution. We found those cases arriving via the emergency room (unscheduled) did in fact follow a Poisson distribution remarkably well (Figure 7) whereas scheduled cases did not. The latter is not so surprising since the arrival of scheduled cases is necessarily a non-random process and is adjusted to balance the arrival of emergency cases.

The close fit of daily emergency arrivals to a Poisson distribution allows us to accurately model the remaining stochastic transition in our model (the other representing marginal cycle times) using the closely related exponential distribution, which captures the corresponding distribution of inter-arrival times.

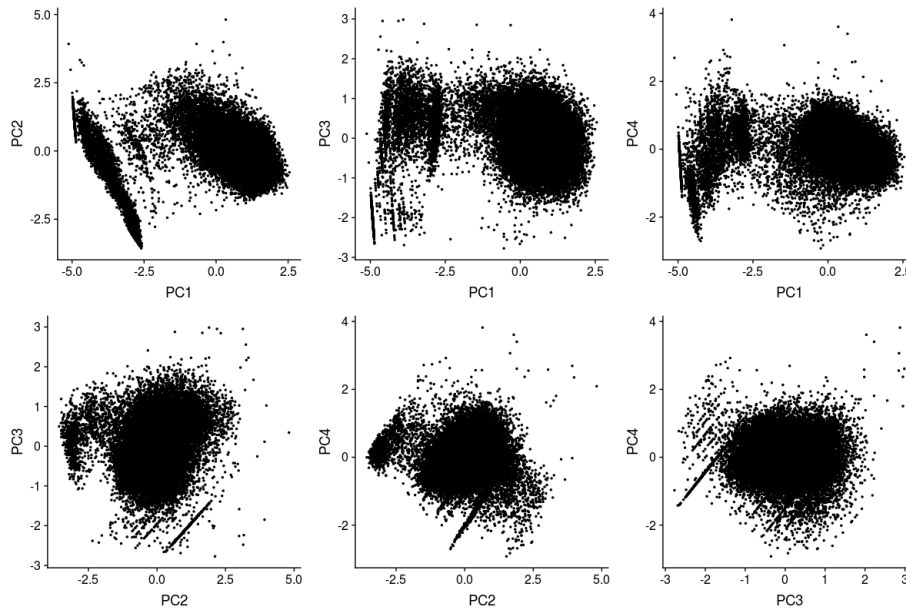
5 Modeling

5.1 Stochastic Time Petri Nets for System-Wide Simulation

To model the surgical workflow, we used Stochastic Time Petri Nets (STPN) which are essentially Petri nets in which the notion of time and uncertainty is incorporated by adding either a deterministic or a probabilistic delay for the firing of transitions. Specifically, transitions can be either immediate, deterministic or stochastic (Table 3). The



(a) Raw durations of 4 phases plotted against each other



(b) Durations w.r.t. principal components. Data was log-transformed prior to PCA transformation.

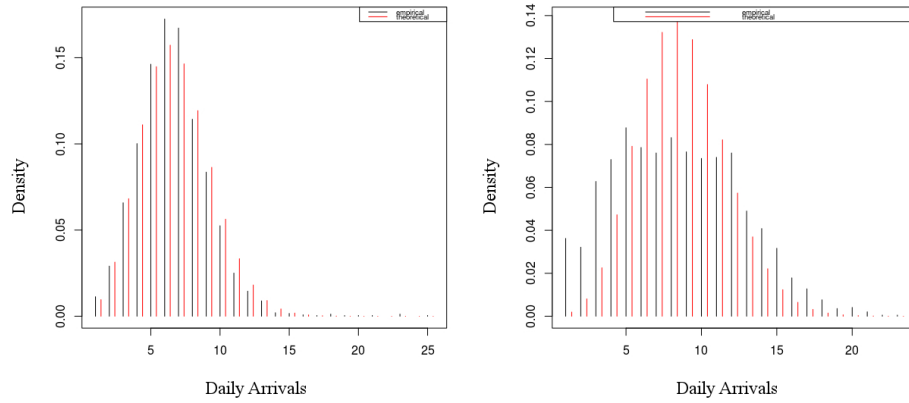


Fig. 7: Daily arrivals (black) along with best-fit Poisson distribution (red). *Left*: emergency cases. *Right*: scheduled cases.

model was implemented in ORIS API; a software tool for the modelling and evaluation of stochastic processes [13]. By using the functions provided by the ORIS tool we were able to evaluate the performance of the system by observing how different metrics change when we alternate some of its aspects.

TRANSITION	REPRESENTATION	DESCRIPTION
Immediate	? condition 	Fires immediately if enabled, conflicts between competing enabled immediate transitions are resolved using priority ranking
Deterministic	5 ■	Fires after a fixed amount of time upon becoming enabled
Stochastic	$\lambda e^{-\lambda x}$ ■	Fires after a delay sampled from a probability distribution upon becoming enabled

Table 3: Overview of transition types in Stochastic Time Petri nets.

Surgical Ward Workflow Description The scenario that is considered for this study is the following: Emergency patients arrive in the hospital at a certain rate to receive treatment throughout the entire 24 hour period (transition *arrival* in Figure 8b) while elective ones are only allowed to arrive at the hospital during the working hours (uniform transition *arrival* in Figure 8a with enabling function `?working_hours=1`). Emergency arrivals go through a checking procedure for the determination of the severity of their condition. For the current model we assume that about half of the cases do not require

immediate intervention (uniform transitions *emergency_status* and *scheduled_status*). If it is decided that the operation must be performed immediately, the patient moves to the Preop room in order to get prepared for the operation (place *preop_room*).

According to the NCEPOD urgency classification [30], target time to theater varies depending on the case. For the purposes of this study, only one type of emergency is considered, and the expected time to theater was set to 30 minutes. On the other hand, scheduled cases can be cancelled up to the time they are about to be placed under anaesthesia if there are no available resources (e.g. beds, surgeons, theaters) to continue the process (transitions *cancel_1*, *cancel_2*). This is not the case for emergency patients, however, who can move from phase to phase (pre-op, anesthesia, surgery, recovery) if the resources for the next part of the process are available. In this paper, we assume that a surgeon and an anesthetist is required for the surgery. Furthermore, the anesthetist is also required during anesthesia and recovery. Note that priority is given to emergency patients over scheduled ones when a decision has to be made regarding the entry to an operating theater or an anesthetic room. The duration of each phase (pre-op, anesthesia, surgery, recovery) is modelled using stochastic transitions with random firing rates following distributions that match our findings in Section 4 (Table 2). The properties of these transitions is shown in Table 5.

To account for cancellations and delays, two places were added in the STPN, namely *cancelled_cases_24h* and *emergencies_wait_cases*. In the former a token is added every time a scheduled case gets cancelled while in the latter a token is added whenever an emergency case is waiting for more than 30 minutes. Both places are reset to zero at the start of a working day. Prioritization and availability checking were incorporated in the model by setting the proper enabling functions and marking updates to transitions. For instance, the enabling function of the transition *enter_recovery* was set to *bed_available > 0*. This property of the model prevents emergency patients to enter the recovery phase if there are no available beds. Table 4 illustrates some examples of how our STPN captures some other policies and guidelines.

Additional simplifying assumptions made for this first iteration of modeling include: that anesthetic rooms and operating theatres are completely independent, a constant number of resources are available within and outside working hours and only one type of recovery room is present whereas in reality different sections are present, such as ICU and high-dependency unit.

Figure 8c shows the basic outline of our model. Patient flows were modeled as individual workflow nets in order to capture constraint violations *for individual patients*. These workflow nets were then programmatically duplicated during simulations. Due to space limitations we are unable to elaborate all details of the model, particularly enabling/update functions and firing priorities, but the full model can be found online ⁵.

Simulation Using our model, we investigated different resource capacity scenarios, with a focus on the ability of the system to fulfill two quality-of-service indicators:

- Number of scheduled cases cancelled in 24 timeframe
- Target time to theatre < 30 minutes for emergency cases

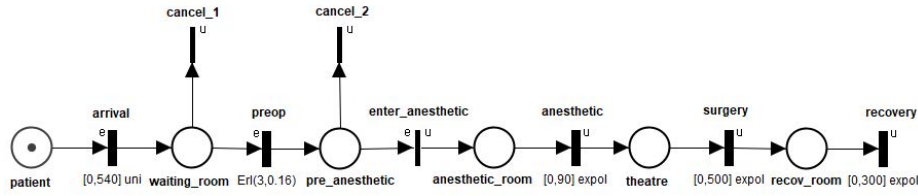
⁵ <http://www.github.com/apapan08>

POLICY/GUIDELINE	PETRI NET FRAGMENT
Normal working hours are from 8-18	
Non-emergency procedures should be handled within normal working hours	
Scheduled cases can be cancelled all the way up to entering anesthetic room if all theatres are occupied by higher priority cases	

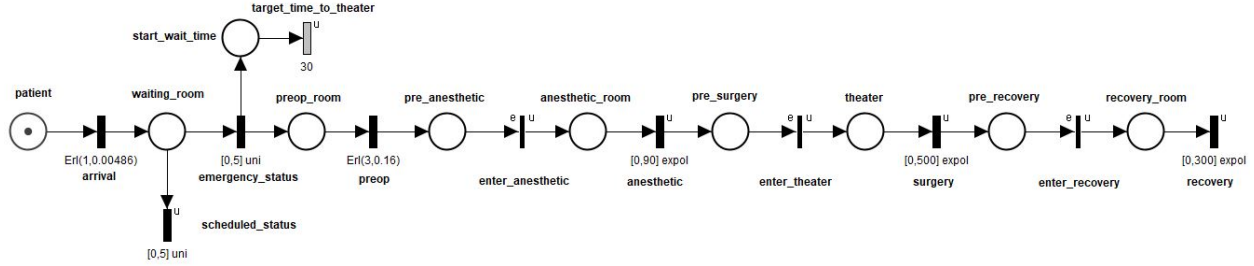
Table 4: Simplified examples of some of the policies, guidelines and constraints for patient flows captured by our model, along with the fragment of the Petri net which captures this.

TRANSITION	DISTRIBUTION	PARAMETERS
Emergency arrivals	Exponential	$\lambda = 0.00487$
Preop	Erlang	$k = 3, \lambda = 0.16$
Anesthetic	Polynomial of Exponentials	$1.5x^2e^{-0.11x} - 10xe^{-0.11x} + 30e^{-0.11x}$
Surgery	Polynomial of Exponentials	$x^2e^{-0.0385x} + xe^{-0.0085x}$
Recovery	Polynomial of Exponentials	$x^2e^{-0.05x} - 10xe^{-0.1x} + 150e^{-0.1x}$

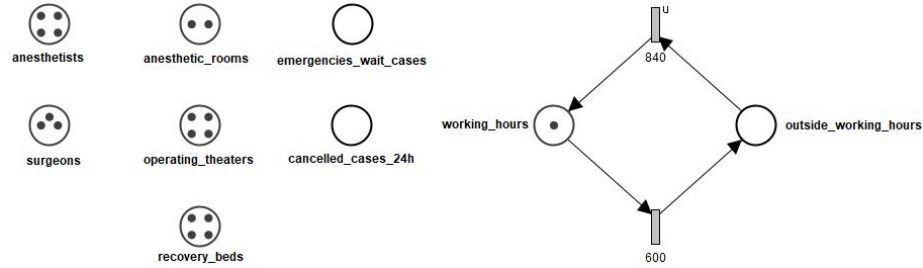
Table 5: Fitted parameters for stochastic transitions. See Table 2 for a visualization.



(a) Template for scheduled patient flow.



(b) Template for emergency patient flow.



(c) Shared resources.

Fig. 8: Stochastic Time Petri Net used to model core aspects of patients flows.

These are properties that are straightforward to formalize and evaluate. In fact, any properties that can be formalized in an appropriate temporal logic such as LTL or MITL⁶, which read similarly to natural language guidelines, can be evaluated. In addition to reporting the expected values for these QoS criteria, we included the expected resource availability over time. This helps us to identify when, and for which resources, potential bottlenecks arise. In a more sophisticated model, we would likely see more complex patterns resulting from interacting processes/resources.

We report results in terms of expected value⁷ across 100 simulation runs of each scenario. That is, the number of cancelled cases or available operating theatres at a given

⁶ Linear Temporal Logic, Metric Interval Temporal Logic.

⁷ $E[X] = \sum x p(X = x)$

	EXCESS CAPACITY	SUFFICIENT CAPACITY	INSUFFICIENT CAPACITY
scheduled cases/day	10	10	10
anesthetists	10	8	4
surgeons	8	5	3
anesthetic rooms	8	4	2
operating theatres	10	8	4
recovery beds	12	6	4

Table 6: Resource capacity scenarios explored in simulations.

time, averaged over simulation runs. With this simple model, the state of the system follows a consistent periodic fluctuation according to the working hours. Realistically, however, greater fluctuations would be likely due to uneven patient inflows and staffing availability patterns over time.

5.2 Bayesian Nets for Individualised Prediction

In modelling cycle times in patient flows, our model in Figure considers only marginal distributions, i.e. cycle time estimates are identical for all patients. However, as we illustrated in Section 4.2, there clearly exist categories of patients with significant variations in cycle times.

Bayesian Nets are probabilistic graphical models that capture the structure of complex probability distributions. By exploiting conditional independence relations between variables, inference algorithms allow us to query the belief network in a flexible manner, even with only partial information [25]. In the present context, Bayesian nets allow us to take into account multiple attributes of a patient along with the partial completion of their treatment in order to make significantly more accurate and nuanced predictions regarding cycle time and other aspects, such as destination.

We present the results of two Bayesian networks in predicting cycle time, leaving as an important avenue of future research the integration of these predictive models into a more sophisticated process model which accounts individual patient attributes. Figure 9 illustrates that Bayesian networks can be integrated with Stochastic Petri net models to more accurately model transition distributions specifically.

The models were built and trained using algorithms implemented in the `bnlearn` package for the programming language R [35]. We used our own implementation of cross-validation, in part to avoid data snooping, and added simple smoothing procedures to account for undersampling.

Feature Selection We evaluated two networks: a 10-variable model and a 22-variable extension of the first. The decision of which attributes to include was based in part on our exploration of conditional distributions in Section 4.2 and pairwise mutual information described in 4.2.

One feature of note is the *Cluster* node in both 10 and 22 variable models. This represents the patients groups identified in 4.2. Using simple *k*-means clustering, we

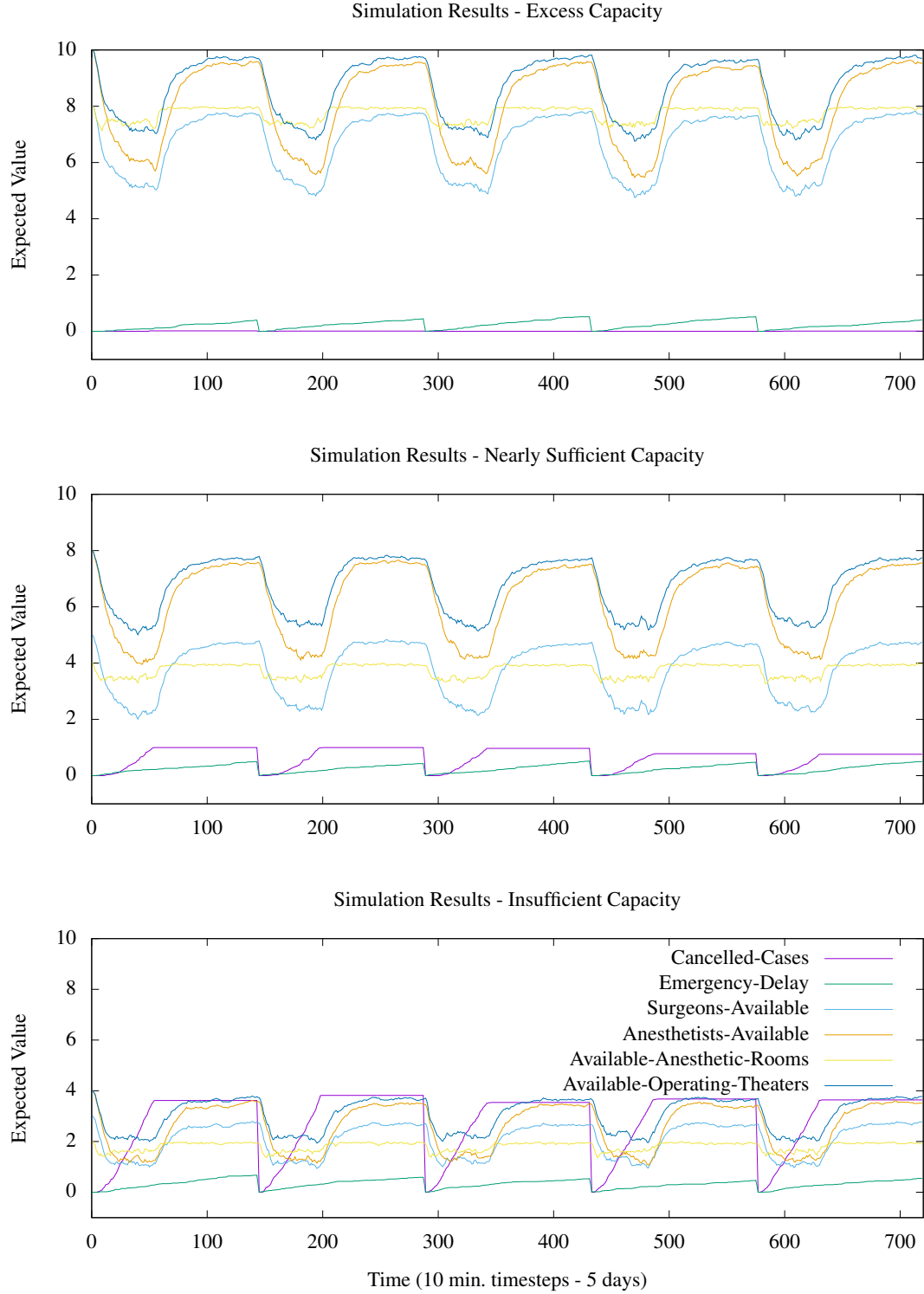


Table 7: Simulation results for scenarios with (*top*) excess capacity, (*middle*) sufficient capacity, and (*bottom*) insufficient capacity averaged over 100 runs. Cancelled-Cases and Emergency-Delay represent failures to meet quality of service guidelines when the expected value exceeds 0. In other words: these are the system properties we are interested in verifying.

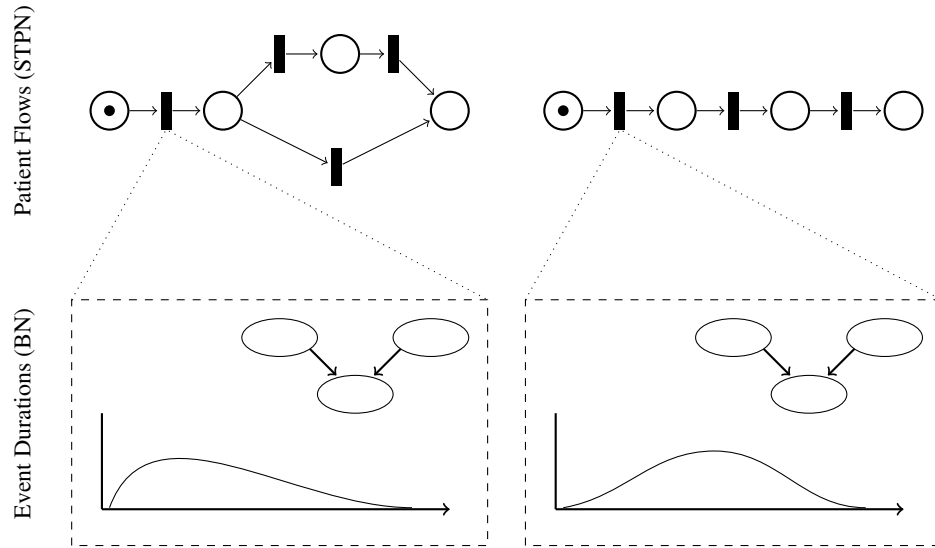


Fig. 9: Petri nets capture control-flow of a process, while Bayesian networks allow nuanced modelling of transition distributions based on case attributes. Integrating these two modelling perspectives is an important next step in developing data-driven, patient centric workflow models.

experimented with identifying 5, 10, 15, and 20 patient clusters which proved moderately helpful in improving performance - in particular for predicting anesthetic cycle times using the 22-variable model.

Structure Learning There are two methods for constructing the graph structure of a Bayesian net: manually, based on expert knowledge; and automatically using structure learning algorithms. After several attempts at building nets manually, we found that automatically generated nets outperformed, despite sometimes finding odd connections between variables.

We employed the score-based structure learning algorithms hill-climbing and TABU structure-learning algorithms, using scoring functions Akaike and Bayesian Information Criterion (AIC/BIC). This choice was due to their computational tractability and suitability to the our hybrid dataset (continuous and discrete attributes). The 22 variable graph can be seen in Figure 10.

Evaluation Models were evaluated based on prediction of cycle times for the 4 phases of the surgical patient flow using 10-fold cross validation. We present mean *absolute* error in Figure 11 for comprehensibility, but note that mean squared error result closely follow the same pattern. As a baseline for comparison, results are shown for the best-fit marginal distributions reported in Table 2.

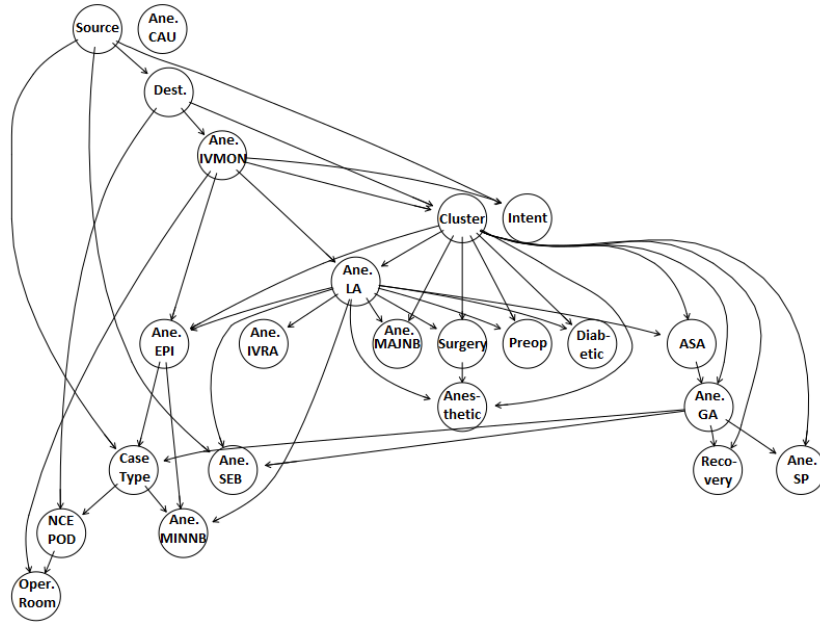


Fig. 10: A Bayesian belief network taking into account 22 attributes of a patient's treatment. Attributes prefixed by *Ane.* denote different types of anaesthetic. Note the central role of the latent *Cluster* attribute discovered in Section 4.2. Originally published in [6].

Avoiding Data Snooping One pitfall that was important to avoid, in particular when modelling *partial executions* of patient flows, was that of inadvertent data snooping by including the *Cluster* attribute. When evaluating a model's predictive power on test data, the cluster should be considered an *unobserved variable*.

While intuitively obvious, this is a crucial methodological point, as including it would constitute a form of data snooping since the cluster itself is in fact derived from the target variables (cycle times). Nonetheless, the variable is able to play a role in the Bayesian network, despite being unobserved, via conditional dependencies between it, observed variables and unobserved target variables.

6 Discussion

The case study presented in this paper has highlighted a number of challenges and lessons learnt that can be applied to other surgical workflow modeling projects, as well as wider data-driven healthcare improvement initiatives. First, data quality assurance is key. One of the most immediate observations of our analysis was the presence of a good deal of anomalous data. Process mining techniques proved to be useful for detecting outliers and identifying anomalies related to the control-flow.

Second, even though it seemed initially that there were no groupings of patients with regards to duration, PCA revealed latent patient categories in our data. Identifying

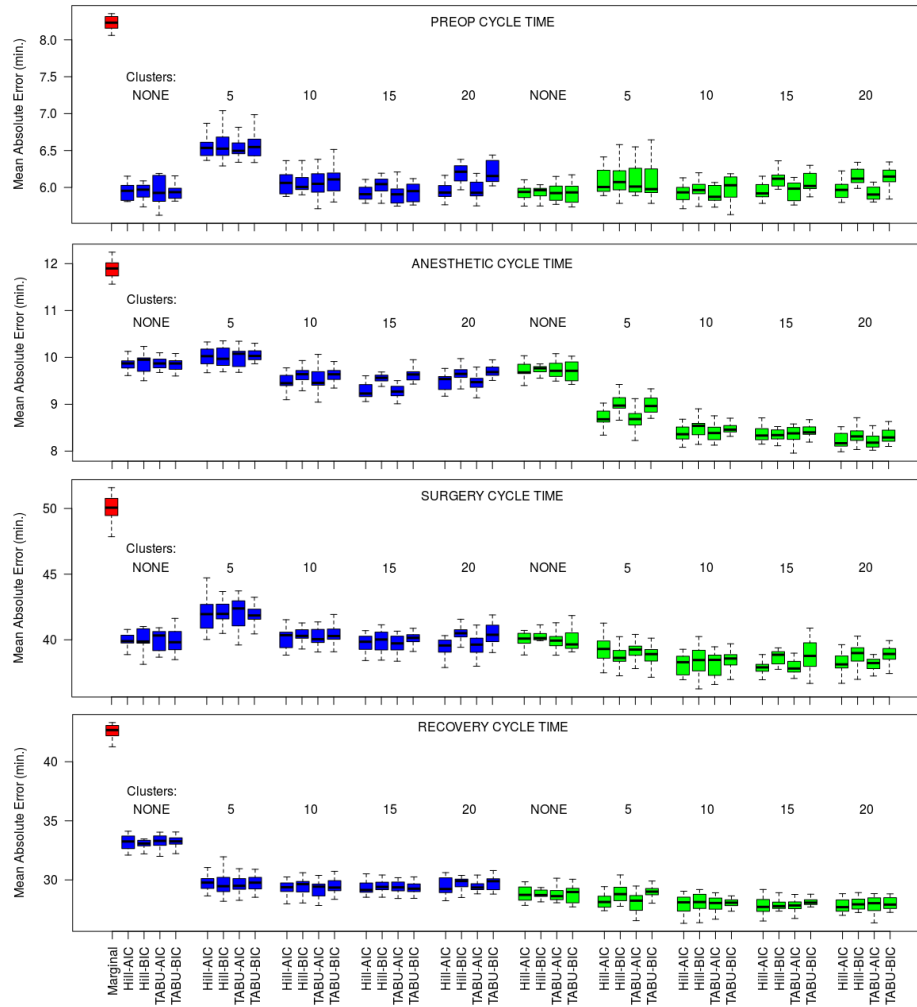


Fig. 11: Comparison of 40 different Bayesian net models using 10 (blue) and 22 (green) variables, but different structure learning algorithms. Results are based on 5 runs of 10-fold cross-validation for predicting cycle time of partially completed patient flows. As a baseline comparison, the simple best-fit marginal distribution (reported in Table 2) is shown in red. Originally published in [6].

these patient clusters improved Bayesian net prediction, even though interpreting what these categories mean is not straightforward.

Third, we showed that a reasonably accurate predictive model of event cycle times in the form of a simple Bayesian belief network can be built, which significantly outperforms simple marginal distribution fitting. The choice of Bayesian networks was motivated by their flexibility and interpretability, which is of great importance in safety-critical domains like healthcare. The ability to query these models suggests they would be a strong component of an intelligent probabilistic scheduling system in surgery.

Finally, Stochastic Time Petri Nets were found to be an appropriate formalism for capturing hospital policies and guidelines surrounding surgery, in particular regarding timing and resource requirements. Distinguishing between the workflows for emergency and scheduled cases was possible in a clear and transparent way, and incorporating case prioritization was straightforward. Even though the model presented in Section 5.1 is a simplified version of reality, it serves as proof-of-concept of how real-world data can be incorporated into a model that combines official procedures and guidelines with domain expert knowledge. Simulating different scenarios allowed us to test the limits of the system and to analyse the effect of varying resource allocation.

7 Conclusion and Future Work

In this paper, we presented a preliminary investigation into probabilistic workflow modeling, simulation and prediction methods in surgical wards. This is an important first step towards much-needed surgical care improvement. Our analysis is focused on key surgical phases, which is a level of granularity that has received less attention in existing literature.

Data-informed surgical care scheduling that takes into account individual patient characteristics, resource availability and hospital policies presents a promising approach to improving resource utilization, quality of care and, ultimately, patient and staff satisfaction. We have demonstrated the value of combining several data analysis paradigms, from mathematical modeling to process mining and machine learning, towards developing a model that effectively captures the complexity of surgical processes, while allowing for experimentation and insightful interrogation. This approach is applicable in other areas of the healthcare system, where under-utilization of expensive resources calls for precise scheduling to minimize costs and waiting times.

Our analysis considered both system-wide surgical performance (through Stochastic Time Petri Net modeling and simulation) and individual patient flow (through Bayesian Net cycle time prediction). In order to integrate the two in the future and incorporate Bayesian nets into Petri Net modeling, we propose the use of Bayesian Stochastic Petri Nets [39].

In the big data and precision medicine era, developing intelligent methods for dynamic and personalized scheduling in the surgical ward is a key research direction. Extending the work presented in this paper to incorporate more detailed information about the surgical ward is desirable, and would possibly require further domain knowledge and dimensionality reduction, so as to deal with the huge cardinality of some attributes. We also regard evaluation with domain experts as an important next step, ensuring that

the recommendations of a future surgical scheduling system are understood and deemed useful by hospital staff.

References

1. Reorganizing patient care and workflow in the operating room: a cost-effectiveness study. *Surgery* **139**(6), 717–728 (2006)
2. Van der Aalst, W.e.a.: Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering* **16**(9), 1128–1142 (2004)
3. Acid, S., e.a.: A comparison of learning algorithms for bayesian networks: a case study based on data from an emergency medical service. *Artificial intelligence in medicine* **30**(3), 215–232 (2004)
4. Ahmadi, S.A., e.a.: Motif discovery in or sensor data with application to surgical workflow analysis and activity detection. In: M2CAI workshop, MICCAI, London. Citeseer (2009)
5. Akkerman, R., Knip, M.: Reallocation of beds to reduce waiting time for cardiac surgery. *Health care management science* **7**(2), 119–126 (2004)
6. Back, C.O., Manataki, A., Harrison, E.: Mining patient flow patterns in a surgical ward. In: HEALTHINF. pp. 273–283 (2020)
7. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
8. Basin, D., Klaedtke, F., Müller, S., Zălinescu, E.: Monitoring metric first-order temporal properties. *J. ACM* **62**(2), 15:1–15:45 (May 2015)
9. Berti, A., e.a.: Process Mining for Python (PM4Py): Bridging the Gap Between Process-and Data Science. In: ICPM Demo Track (CEUR 2374) (2019)
10. Blum, T., e.a.: Workflow mining for visualization and analysis of surgeries. *Int. journal of computer assisted radiology and surgery* **3**(5), 379–386 (2008)
11. Bouarfa, L., e.a.: Discovery of high-level tasks in the operating room. *Journal of biomedical informatics* **44**(3), 455–462 (2011)
12. Bouarfa, L., Dankelman, J.: Workflow mining and outlier detection from clinical activity logs. *Journal of biomedical informatics* **45**(6), 1185–1190 (2012)
13. Bucci, G., Carnevali, L., Ridi, L., Vicario, E.: Oris: a tool for modeling, verification and evaluation of real-time systems. *International journal on software tools for technology transfer* **12**(5), 391–403 (2010)
14. Burattin, A., Maggi, F., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. *Expert Systems with Applications* **65** (03 2015). <https://doi.org/10.1016/j.eswa.2016.08.040>
15. Cochran, J.K., Bharti, A.: Stochastic bed balancing of an obstetrics hospital. *Health care management science* **9**(1), 31–45 (2006)
16. Denton, B., e.a.: Optimization of surgery sequencing and scheduling decisions under uncertainty. *Health care management science* **10**(1), 13–24 (2007)
17. Fages, F., Rizk, A.: From model-checking to temporal logic constraint solving. In: International Conference on Principles and Practice of Constraint Programming. pp. 319–334. Springer (2009)
18. Fu, J., Topcu, U.: Computational methods for stochastic control with metric interval temporal logic specifications. In: 2015 54th IEEE Conference on Decision and Control (CDC). pp. 7440–7447. IEEE (2015)
19. Funkner, A. A., e.a.: Towards evolutionary discovery of typical clinical pathways in electronic health records. *Procedia computer science* **119**, 234–244 (2017)
20. Haro, B.B., e.a.: Surgical gesture classification from video data. In: Int. Conf. on Medical Image Computing and Computer-Assisted Intervention. pp. 34–41. Springer (2012)

21. Huang, Z., e.a.: Summarizing clinical pathways from event logs. *Journal of biomedical informatics* **46**(1), 111–127 (2013)
22. Hulshof, P. J. H., e.a.: Tactical resource allocation and elective patient admission planning in care processes. *Health care management science* **16**(2), 152–166 (2013)
23. Jiménez Ramírez, A., Barba, I., Fdez-Olivares, J., Del Valle, C., Weber, B.: Time prediction on multi-perspective declarative business processes. *Knowledge and Information Systems* (03 2018). <https://doi.org/10.1007/s10115-018-1180-3>
24. Kayis, E., e.a.: Improving prediction of surgery duration using operational and temporal factors. In: *AMIA Annual Symposium Proc.* vol. 2012, p. 456. American Medical Informatics Association (2012)
25. Koller, D., Friedman, N.: *Probabilistic graphical models: principles and techniques*. MIT press (2009)
26. Lalys, F., Jannin, P.: Surgical process modelling: a review. *Int. journal of computer assisted radiology and surgery* **9**(3), 495–511 (2014)
27. Lin, H.C., e.a.: Towards automatic skill evaluation: Detection and segmentation of robot-assisted surgical motions. *Computer Aided Surgery* **11**(5), 220–230 (2006)
28. Mans, R., e.a.: Mining processes in dentistry. In: *Proc. of the 2nd ACM SIGHIT Int. Health Informatics Symposium*. pp. 379–388. ACM (2012)
29. Martina, S., Paolieri, M., Papini, T., Vicario, E.: Performance evaluation of fischer’s protocol through steady-state analysis of markov regenerative processes. In: *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. pp. 355–360. IEEE (2016)
30. NCEPOD: NCEPOD classification of intervention. <https://www.ncepod.org.uk/classification.html> (2019), accessed: 2019-11-22
31. Neumuth, T., e.a.: Analysis of surgical intervention populations using generic surgical process models. *Int. Journal of Computer Assisted Radiology and Surgery* **6**(1), 59–71 (2011)
32. NHS Scotland: National theatres project report. <https://www.isdscotland.org/Health-Topics/Quality-Indicators/National-Benchmarking-Project/National-Theatres-Project/> (2006), accessed: 2019-11-22
33. Paolieri, M., Horvath, A., Vicario, E.: Probabilistic model checking of regenerative concurrent systems. *IEEE Transactions on Software Engineering* **42**(2), 153–169 (2015)
34. Royal College of Anaesthetists: *Perioperative medicine: the pathway to better surgical care* (2015), London
35. Scutari, M.: Learning Bayesian Networks with the bnlearn R Package. *Journal of Statistical Software* **35**(3), 1–22 (2010), <http://www.jstatsoft.org/v35/i03/>
36. Stahl, J. E., e.a.: Reorganizing patient care and workflow in the operating room: a cost-effectiveness study. *Surgery* **139**(6), 717–728 (2006)
37. Stauder, R., e.a.: Random forests for phase detection in surgical workflow analysis. In: *Int. Conf. on Info. Processing in Computer-Assisted Interventions*. pp. 148–157. Springer (2014)
38. Strum, D., e.a.: Modeling the uncertainty of surgical procedure times: comparison of log-normal and normal models. *Anesthesiology* **92**(4), 1160–1167 (2000)
39. Taleb-Berrouane, M., Khan, F., Amyotte, P.: Bayesian Stochastic Petri Nets (BSPN) - a new modelling tool for dynamic safety and reliability analysis. *Reliability Engineering & System Safety* **193**, 106587 (2020)
40. Westergaard, M., Maggi, F.: Looking into the future using timed automata to provide a priori advice about timed declarative process models (2012)

Discovering Responsibilities with Dynamic Condition Response Graphs [★]

Viktorija Nekrasaite¹, Andrew Tristan Parli¹, Christoffer Olling Back¹, and
Tijs Slaats¹

Department of Computer Science, University of Copenhagen
Emil Holms Kanal 6, 2300 Copenhagen S, Denmark
viktorijanekrasaite3@gmail.com, drewparli@gmail.com, {back, slaats}@di.ku.dk

Abstract. Declarative process discovery is the art of using historical data to better understand the responsibilities of an organisation: its governing business rules and goals. These rules and goals can be described using declarative process notations, such as Dynamic Condition Response (DCR) Graphs, which has seen widespread industrial adoption within Denmark, in particular through its integration in a case management solution used by 70% of central government institutions. In this paper, we introduce PARNEK: a novel, effective, and extensible miner for the discovery of DCR Graphs. We empirically evaluate PARNEK and show that it significantly outperforms the state-of-the-art in DCR discovery and performs at least comparably to the state-of-the-art in Declare discovery. Notably, the miner can be configured to sacrifice relatively little precision in favour of significant gains in simplicity, making it the first miner able to produce understandable DCR Graphs for real-life logs.

Keywords: Declarative Process Discovery, Declarative Models, Dynamic Condition Response Graphs, DCR Graphs, DCR Discovery

1 Introduction

Automating knowledge intensive processes offers unique challenges: the problems faced by knowledge workers are highly heterogeneous and tend to require unique solutions. In municipal government, for example, no two citizens are exactly the same: when dealing with long-term unemployment or illness, solutions will be tailored to each citizen’s individual needs. These unique solutions are to a large degree determined by the case workers themselves, in sharp contrast to traditional production processes where the worker is usually expected to strictly follow the instructions of the system. At the same time, organisations and their workers do have many responsibilities: goals need to be met, laws need to be adhered to and business practices followed. Information systems should ensure

[★] This work is supported by the Hybrid Business Process Management Technologies project (DFF-6111-00337) funded by the Danish Council for Independent Research, and the EcoKnow project (7050-00034A) funded by the Innovation Foundation. First and second author listed alphabetically.

that these responsibilities are met, and therefore some form of control over the activities of the workers is still required.

Because of this high degree of flexibility, it can be problematic to focus on the *how* of knowledge work: defining all possible paths through a process becomes cumbersome as the number of variations increases, and often it is impossible to exactly predict what variations may occur in the future. Instead, it can be helpful to focus on the *what* of knowledge work: the goals we aim to achieve, and the rules to which must adhere. It can then be left to the information system to derive possible paths from these goals and rules.

Imperative process notations such as BPMN and Petri nets are used to capture the *how* of a process by using a flow-based paradigm: when executing the model, one follows the flow of the model, usually indicated by arrows, executing the activities of the model along the way. Such notations are poorly suited to capturing the *what* of the process: rules are not defined explicitly, but instead encoded as allowed paths. One rule may affect many different paths, and each path could result from many different rules interacting with each other. *Declarative* process notations such as Declare and Dynamic Condition Response (DCR) Graphs are better suited to capturing rules. They describe a process as a set of constraints which can be mapped directly to specific business rules or goals. An execution semantics is usually achieved by either mapping the declarative model to a flow-based model (e.g. transition systems), or by introducing an operational semantics that reasons over the state of the different constraints and/or activities of the model.

Instead of requiring process consultants to design all rules upfront based on requirement specifications and interviews with users, it can be helpful to use data describing the historical execution of processes, represented as event logs, to automatically derive possible rules in the form of formal constraints. Finding and proposing constraints automatically reduces the workload of the consultants and ensures the resulting models are grounded in practice, instead of an ideal – but not fully realistic – world envisioned by the users.

Such declarative process discovery algorithms can produce models in various notations. Although Declare [15], the first declarative notation introduced exclusively for describing business processes, is most prevalent in academia, there is no documented usage of the notation in industry, leaving it as a mostly academic pursuit. DCR Graphs [9] is a more recent declarative notation that has been more successful commercially. It was picked up by a Danish developer of case management systems, leading to the creation of a commercial modelling tool [18]. More recently, 70% of Danish central government institutions adopted a case management solution that uses a DCR process engine, including the police, military, tax authorities and largest public universities ¹.

DCR Graphs' success notwithstanding, process discovery based on this notation is still in its infancy: there currently exists a single algorithm, which first generates a model with all possible constraints and then removes constraints until the model allows all behaviour seen in the event log. We posit that this

¹ <http://www.kmd.dk/indsigter/fleksibilitet-og-dynamisk-sagsbehandling-i-staten>

approach is inconsistent with the declarative paradigm: if the goal of declarative models is to support flexibility, then it is counter-productive to start from an assumption that the process should be fully constrained. This intuition is supported by a closer look at the generated models: they are precise, but lack simplicity and often appear to mimic flow-based diagrams in structure.

In this paper, we introduce the PARNEK miner, a novel discovery algorithm for DCR Graphs, with a more methodologically sound basis than existing DCR Graphs miners: we start from a fully unconstrained model and, based on the event log, propose constraints that should be introduced to reasonably restrict behaviour. We show that PARNEK significantly outperforms the state-of-the-art in DCR Graphs discovery and that our results are at minimum comparable to that of MINERful, a state-of-the-art miner in declarative process discovery. We also discuss advantages of PARNEK in terms of configurability and extensibility.

The paper continues as follows: we first discuss related work in section 2 and introduce DCR Graphs in section 3. In section 4, we explain the core algorithms underlying the PARNEK miner. We empirically evaluate PARNEK against the state-of-the-art in DCR Graphs and Declare mining in section 5 and conclude in section 6.

2 Related work

Declarative notations were first introduced to the BPM community through the work on Declare [15], which proposed mapping an extensive number of control flow patterns to linear temporal logic constraints. The first process discovery algorithm for Declare was the Declare Miner [13], which suffered from being a brute-force approach, leading to poor run-time performance. A significant improvement was proposed in [12]. Another Declare-based miner, MINERful [2], has been developed as an alternative, focusing on run-time efficiency and a high degree of customisation. A multi-perspective approach to Declare discovery, including data, time and resource constraints, was introduced in [17].

Several alternative declarative notations have been proposed [11, 21]. Two stand out in particular for having led to the development of corresponding process discovery algorithms. The guard-stage-milestone model [10] provides a more artifact-centric, rather than constraint-centric, view on processes, but integrates declarative aspects and has become the basis of the recent CMMN standard [14]. A process mining approach for GSM was proposed in [16] as a multi-step process that first discovers artifact life-cycles in an event log and then maps these to a GSM schema.

DCR Graphs offers several advantages over Declare, namely stronger formal expressiveness [5], an operational semantics defined directly on the model with a corresponding visual representation and active adoption by industry [18]. Several extensions have been proposed, such as data [19], time [8] and subprocesses [3]. The work on process discovery for DCR Graphs has not advanced to the stage of considering these extensions yet, as finding a suitable base algorithm is the first priority.

3 Modelling responsibilities with DCR Graphs

In this section we describe the structure and semantics of DCR Graphs in more detail and illustrate their use for modelling casework by using a simplified running example of a municipal unemployment process.

Example 31 (Unemployment Process) *A citizen starts the unemployment process by filing a request for unemployment benefits. Until it is approved, they can update this request at any time to correct it or provide additional information. Once a request has been submitted, a municipal caseworker can approve or reject the request. It is possible to approve a request that was previously rejected, for example because additional information was provided, or a mistake was made. However, it is not possible to reject a request that has already been approved (cancellation of benefits is handled by its own separate process). Once the request is approved, the citizen should submit monthly documentation of the progress of their job-search. Once documentation is provided, the municipality is required by law to pay their benefits.*

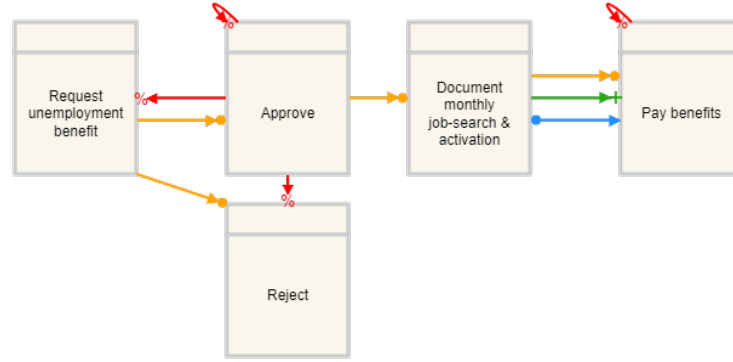


Fig. 1. DCR graph of a basic unemployment process.

Figure 1 shows the unemployment process modelled as a DCR Graph.

The *activities* of the process are represented as boxes. In formal DCR terminology these are referred to as *events*, which can represent anything that can happen, such as an activity executing, but also data changing or deadlines being reached. Note that these are different from the *log events* used in process discovery, as they can represent more than one occurrence of something happening. In process discovery terminology they are closer to *event classes*.

An unconstrained activity can happen at any time, any number of times. This means that in a graph with no constraints, anything can happen at any time, making it equivalent to the flower model often encountered in flow-based notations.

To constrain the behaviour of the process, we introduce *relations*. The *condition relation*, denoted as **Request unemployment benefit** \rightarrow_{\bullet} **Reject**, indicates that we cannot reject a request before it has been submitted. Conditions do not

alternate: after the first request, one can reject it any number of times, even if no new requests are submitted. This befits the declarative paradigm which stresses only adding constraints when absolutely required. In this case it may, for example, be convenient to update a rejection with additional reasons. Similarly, **Request unemployment benefit** is a condition for **Approve**, which is a condition for **Document monthly job-search**, which is a condition for **Pay benefits**.

The *exclusion relation*, denoted as **Approve** $\neg\%$ **Reject**, indicates that after doing **Approve**, it is no longer possible to **Reject** the request because it has been removed from the process. **Approve** similarly removes **Document monthly job-search**. Events can also exclude themselves, meaning that after they have been executed, they are no longer a part of the process. This is the case for **Approve** and **Pay benefits**.

Exclusion of events is *dynamic*: it can be undone by re-including the event through the *inclusion relation*, denoted as **Document monthly job-search** $\rightarrow\vdash$ **Pay benefits**. Including an event adds it back into the process, allowing it to be executed again. In the example, each time monthly documentation is submitted, pay benefits will be included and can be executed again.

The final *response relation* is used to indicate future requirements or goals. It is denoted as **Document monthly job-search** $\bullet\rightarrow$ **Pay benefits** and ensures that when monthly documentation is submitted, benefits will eventually be paid.

While timed constraints [8] could be used to further specify the model, for example by stating that **Document monthly job-search** can only happen once per month, we only consider control-flow constraints in this paper and leave the discovery of other perspectives to future work.

4 Discovery of dynamic condition response graphs

Given that we aim to mine flexible declarative models, we start from the least restrictive DCR Graph, containing all activities seen in the event log and no relations, allowing for any trace over these activities. PARNEK implements a number of algorithms for adding relations to this unrestricted model. These algorithms are largely orthogonal, each finding relations independently that can be combined to form the final model. According to the declarative paradigm exceptional behaviour should be embraced and therefore all algorithms have been designed to guarantee perfect fitness. The miner can be configured by the user to select which algorithms should be used. Through experimentation, we identified a number of promising configurations.

4.1 The base ParNek algorithm

The base algorithm is inspired by a subset of the constraint templates used by the MINERful [2] algorithm, however we do not always map these precisely. Instead we aim for a balance between simplicity and precision where we allow for small amounts of additional behaviour in cases where this means introducing significantly fewer relations. We consider the following constraints:

- **ATMOSTONE(A)**: All such activities can be represented as a self-excluding activity in a DCR graph, i.e., $A \neg\% A$
- **RESPONSE(A, B)**: We utilise this constraint template to add response relations to the model i.e., $A \bullet \rightarrow B$
- **PRECEDENCE(A, B)**: We utilise this constraint template to add condition relations to the model i.e., $A \rightarrow \bullet B$
- **CHAINPRECEDENCE(A, B)**: We map this constraint to A includes B and B is a self-excluding activity, i.e., $A \rightarrow + B$ and $B \neg\% B$

The base algorithm will often find redundant condition and response relations. For example, for a DCR Graph containing only the conditions $A \rightarrow \bullet B$, $B \rightarrow \bullet C$ and $A \rightarrow \bullet C$, the relation $A \rightarrow \bullet C$ is redundant because the other two conditions already ensure that C can not happen before A has happened. We use this transitive property to remove unnecessary conditions. The same form of transitivity can be applied to response relations. It should be noted that this transitivity does not always hold: when an event is dynamically excluded, its outgoing relations are also considered excluded. For example, if we remove the condition $A \rightarrow \bullet C$ and add the exclusion $D \neg\% B$ to the previous example, one would be allowed to execute D , excluding not only the event B , but also the condition $B \rightarrow \bullet C$, allowing C to be executed afterwards. However, removing conditions and responses will never lower the fitness of the model: it will at worst lower precision by allowing additional traces not seen in the log. Experiments showed that this loss of precision was heavily offset by a gain in simplicity, for example for the BPIC-2012 event log we can reduce the number of condition relations from 160 to 30 in return for a decrease in precision of 0.0003.

In the next step of the base algorithm we first build **PREDECESSOR** and **SUCCESSOR** sets for each activity. The **PREDECESSOR** set contains all the activities which can happen before the *last* occurrence of a specific activity in a trace. Similarly, the **SUCCESSOR** set contains the activities that can happen after the *first* occurrence of a target activity in a trace. We leverage these sets to discover two types of exclusions. The first type is based on the observation that an activity can exclude its predecessors which are absent in the **SUCCESSOR** set, i.e., $B \in \text{PREDECESSOR}(A, \log) \setminus \text{SUCCESSOR}(A, \log)$. Moreover, it is possible to skip activities in the model with a self exclusion, i.e., $B \notin \text{SELFEXCLUSIONS}$. This is because self exclusion relations are either the result of **ATMOSTONE** or **CHAINPRECEDENCE**, meaning this extra exclude relation would be redundant. The second type of exclusions addresses mutually exclusive activities: an activity can exclude activities not present in $\text{PREDECESSOR}(A, \log) \cup \text{SUCCESSOR}(A, \log)$.

To ensure that we keep the model simple, we also apply an optimisation step to exclusions. Each time we wish to add an exclusion $A \neg\% B$, we first check if there is already a predecessor of A that excludes B , in which case we do not add the additional exclusion. Similarly to the previous optimisation step, this approach may lead to a slight decrease in precision, but is guaranteed to keep the model fully fitting and leads to a large reduction in the number of constraints.

4.2 Mining additional conditions

The optional COND algorithm is aimed at mining more complex condition patterns. In the base algorithm we only discover a condition $A \rightarrow \bullet B$ when A always precedes B in the log. However, because of the possibility of dynamically excluding conditions, there may exist relevant conditions where this is not the case.

To discover such conditions, we follow these steps:

1. Mine a graph by using one variation of PARNEK
 2. Take a union of activities that come before the last occurrence of a specific activity (remove those which already have a condition)
 3. Examine the resulting activities in every trace using three sets:
 - POSSIBLECONDS: Activities that may have a condition relation to a target activity. In the beginning, the set consists of all the activities that occur before the last occurrence of the target activity except those which already have a condition relation to it
 - IGNORE: All the activities in POSSIBLECONDS before the first occurrence of the target activity in a current trace
 - CHECK = POSSIBLECONDS \setminus IGNORE, i.e., activities for which we need to execute the trace, keeping track of whether the activity is included or excluded from the current trace every time we want to execute the target activity. We stop the verification process if:
 - the activity gets executed
 - we reach the last target activity in the current trace
 - the activity is included when we want to execute the target activity
- If the process gets terminated because of the third condition, then we remove the activity from POSSIBLECONDS since it is not possible to have a condition from it.

By the end of step 3 we are left with the additional conditions. Our experiments showed that the COND algorithm tends to add a few conditions at most. The precision increase from these conditions varied largely per log, with some logs seeing no increase and one log seeing an increase from 0.59 to 0.87.

4.3 Improvements for long, similar sequences, repeated activities

The presence of activities in the event log that repeat themselves non-trivially or long sequences of activities that are highly similar significantly decreases the precision of the models that the base PARNEK algorithm is capable of mining. To increase precision, we developed an optional CHECKFOLLOW algorithm which discovers dynamic include and exclude relations that help guide the local execution of the model.

We define NOTDIRECTLYFOLLOWS(A) as the set of activities that occur after, but not directly after A and not including A . Next we add exclusion relations from A to each activity in NOTDIRECTLYFOLLOWS. We repeat this for all activities in the log. To make sure that these activities are included in the graph again, we need to define the set DIRECTLYPRECEDES(B, A). This is the set of all activities

that directly precede B in the event after the execution of A and not including B. Here we can equate B to one of the activities the A dynamically excluded above. This means that adding inclusion relations to B from all the activities in its corresponding $\text{DIRECTLYPRECEDES}(B, A)$ set will ensure the perfect fitness of the mined model.

In order to balance between model simplicity and precision, **CHECKFOLLOW** takes a threshold parameter that is compared against the cardinality of **DIRECTLYPRECEDES** before introducing any new constraints. By keeping the threshold low, one lowers the number of constraints found, prioritising simplicity over precision. Through experimentation, we noticed that the effect of the threshold is most noticeable when set to either 1 or ∞ . In general, we observed that the **CHECKFOLLOW** algorithm greatly improved the precision of the model, but at the cost of adding a large amount of additional constraints. We discuss this difference in more detail in section 5.

4.4 Running Example

Let us consider our running example and abbreviate the five activities as follows: **Request unemployment benefit** (U), **Approve** (A), **Reject** (R), **Document monthly job-search** (D), **Pay benefits** (P).

U	U, R	U, R, R, A	U, R, A, D, P
U, U	U, R, U	U, A, D, P	U, A, D, P, D, P

Table 1. A log which consists of 8 traces.

If we provide **PARNEK**, in particular the base algorithm, with the event log in Table 1 then we observe that A happens at most once in every trace, i.e., it excludes itself. Moreover, every time D happens, it is followed by P, i.e., we have a response relation. Also, D and P follow the **CHAINPRECEDENCE** pattern, where P is always directly preceded by D. Therefore we discover an inclusion from D to P and a self-exclusion for P. Furthermore, U is the first activity in every trace. As a result, it has conditions to A and R. Additionally, we have $A \rightarrow \bullet D$ since D can only happen after A was executed. In other words, in order to proceed with an unemployment process, first an application needs to be approved. On top of that, there is another condition $D \rightarrow \bullet P$ because P never occurs before the first D in any trace. Finally, if we look at activity A, its **PREDECESSOR** set contains {U, R} while **SUCCESSOR** set consists of D and P. If we take the difference of these two sets, we are left with {U, R}. Since neither U nor R has a self-exclusion, we can add exclusion relations from A to them. This results in the DCR Graph shown in Figure 1.

5 Evaluation

We evaluate the PARNEK miner by comparing its performance against two declarative miners: the DCR miner from [4], referred to as baseline, and MINERful, which returns models based on the Declare modelling language. Aside from running time, we evaluate the degree of under- and overfitting using a notation-agnostic precision metric, and fitness-based cross-validation, respectively.

5.1 Methodology

Metrics Process mining is generally framed as a *descriptive* data mining task: the aim is to accurately describe training data [7]. To this end, quality metrics such as precision and fitness computed on the *in-sample* data set are most often reported. In regards to precision we follow this paradigm, since the formulation of precision we employ cannot be computed on traces which cannot be replayed on the model (as are likely to be present in out-of-sample data). We also present an estimate of *out-of-sample* performance in terms of fitness using cross-validation.

Precision The most widely adopted precision metrics in the process mining literature are defined on Petri nets and cannot be applied to declarative models. A formulation of precision defined on the underlying state-transition system has been proposed in [20] and was implemented for the evaluation proposed in [1]. This formulation allows for comparison across all modelling notations.

Let \mathcal{E} denote the set of unique events in an event log with $e \in \mathcal{E}$. Let $\text{en}_L(e)$ denote the set of activities sharing the same context (i.e., prefix) as event e in log L . Let $\text{en}_M(e)$ denote the set of activities enabled in the state of model M immediately *prior* to executing e . The precision of M w.r.t. log L is given by

$$P_L(M) = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \frac{|\text{en}_L(e)|}{|\text{en}_M(e)|} \quad (5.1)$$

This metric captures the degree to which more behaviour is allowed by the model than has been observed in the log.

Normalised Precision The above formulation of precision is a function of the log as well as the model, and we would like to establish a normalised formulation by establishing a lower and upper bound for the log. This will allow us to better understand the relative difference between miners based on what the worst and best case results are for a particular log.

Let P_L^l and P_L^u denote the lower and upper bound for precision, respectively. The normalised precision is given by

$$P_L^n(M) = \frac{P_L(M) - P_L^l(M)}{P_L^u - P_L^l} \quad (5.2)$$

We take as P_L^l the precision of the *flower* model, in which any activity encountered in the log is enabled in every context. The only models resulting in

a lower estimate of P_L^l are those which allow *more activities than encountered in the log* to be enabled. We base P_{upper} on the maximum formal expressiveness of the modelling notations used. We know that DCR Graphs capture all regular languages and therefore for each event log there exists a DCR Graph representing it with a precision of 1. Thus $P_L^u = 1$.

Cross-validation Precision gives an indication of the degree to which a model may be *underfitting* the data. Unless we assume that an event log represents all of the behaviour we can expect to encounter, we should also investigate the generalisability of our model: the degree to which it may be *overfitting* the training data.

Traditional supervised learning tasks define an error metric, e.g. the number of correctly classified instances in classification tasks, or a distance-based metric, such as the residual sum of squares, for regression tasks. Cross-validation is one approach to estimating the error of a given model on *out-of-sample* data.

We take as our error metric simple trace fitness: the percentage of out-of-sample traces able to be replayed on the model. If we interpret process mining as a binary classification problem (in which traces either fall into acceptable or unacceptable classes), with event logs containing only positive examples, then fitness can be interpreted as *recall*, i.e. the ratio of traces accepted by the model to the total number of traces in the validation log:

$$\text{fitness} = \frac{|\text{true positives}|}{|\text{true positives}| + |\text{false negatives}|} \quad (5.3)$$

We employ k -fold cross-validation, splitting the original logs into k partitions of equal size. The n^{th} partition is then held out as a validation set, the miner run on the remaining $k - 1$ partitions, and the resulting model evaluated on the validation set. This is repeated for $n \in \{1, \dots, k\}$. We report the mean fitness across all k validation sets. Traces are ordered chronologically by the timestamp of the first event prior to partitioning.

We report results for lower k than the standard $k = 10$, having observed that lower k are more informative in that most miners produce almost perfectly fitting models when trained on 90% of the log. Note that computing (normalised) precision of a model mined from the training data and evaluated on the validation data is possible, but problematic since these metrics become skewed by the fact that precision can only be computed for fitting traces. For this reason, we report normalised precision for the whole log.

Logs We base log selection on the criterion of public availability, drawing upon (i) the IEEE Task Force on Process Mining Real-life Event Log Collection², (ii) two additional logs also published by the 4TU Center for Research Data³⁴, and (iii) one real-life log originating from our own industrial contacts [6].

² http://data.4tu.nl/repository/collection:event_logs_real

³ <https://doi.org/10.4121/uuid:6df27e59-6221-4ca2-9cc4-65c66588c6eb>

⁴ <https://doi.org/10.4121/uuid:5a9039b8-794a-4ccd-a5ef-4671f0a258a4>

For logs containing activities with more than one *lifecycle transition* such as `A.start` and `A.complete`, we present results both for the log in which this distinction is ignored, and a modified log in which these are considered distinct activities. No other preprocessing has been performed.

Miner Configuration We evaluate two variants of the PARNEK miner: the base algorithm described in section 4.1 with the improvement for mining additional conditions described in 4.2; and a variant which includes all of the improvements described in section 4.3. Internal experiments showed that the first variant produced simpler models with a reasonable number of constraints. The second variant produced more precise, but more complex models.

MINERful has three parameters: a minimum threshold for *support*, *confidence*, and *interest factor*. We set support to 1.0 to ensure perfect fitness on training data. The remaining two parameters are set such that the resulting model is as close in size to the corresponding PARNEK variant. Parameters are determined automatically by a divide and conquer parameter tuning algorithm, with a step resolution of 0.1. MINERful is allowed to consider all constraints in the Declare language, including negative constraints. Finally, before the size of the mined model is evaluated, inconsistency checking is applied using the setting HIERARCHYCONFLICT with constraint sorting policy: ACTIVATIONTARGETBONDS, FAMILYHIERARCHY, SUPPORTCONFIDENCEINTERESTFACTOR. Redundancy checking does not finish for all logs and therefore not applied.

5.2 Implementation

The aforementioned evaluation metrics have been implemented in a stand-alone CLI application⁵. All mining and evaluation processes were performed on a Lenovo Thinkpad P50 with a 64-bit Intel Xeon E3-1535M v3 2.90GHz CPU and 32GB of RAM running Windows 10 Enterprise 64-bit edition. The PARNEK miner and its source code are available publicly for research purposes⁶.

5.3 Results and Discussion

The bottom half of table 3 shows aggregated results across logs, we make the following observations:

1. PARNEK is able to find significantly simpler models than the baseline (ratio of 0.14), while sacrificing relatively little precision (ratio of 0.58).
2. An extended version of PARNEK is able to find equally precise models (ratio of 1.00), using noticeably fewer constraints (ratio of 0.68).
3. When aiming for comparable model sizes, PARNEK finds more constraints than MINERful (ratio of 1.58 and 1.47 on respectively small and large models), but leads to a relatively larger increase in precision (ratio of 2.12 and 1.53 on respectively small and large models).

	PARNEK MINER		$\frac{\text{Conf.}}{\text{Int. Fac.}}$	PARNEK MINER		$\frac{\text{Conf.}}{\text{Int. Fac.}}$	Base line
	(COND)	-ful		(FULL)	-ful		
BPI Challenge 2012	1281	5108	0.5/0.2	2433	4614	0.2/0.1	1975
BPI Challenge 2012 LT	2089	6713	0.5/0.3	4951	6238	0.2/0.0	2864
BPI Challenge 2013	65	1410	0.0/0.9	151	1107	0.0/0.0	173
BPI Challenge 2013 LT	124	1992	0.1/0.0	340	1296	0.0/0.0	350
BPI Challenge 2017	5164	25409	1.0/0.5	9964	25657	0.1/0.0	7807
Activities Daily Living	72	1185	0.3/0.2	224	872	0.0/0.0	432
Activities Daily Liv. LT	115	2515	0.7/0.1	1127	2094	0.1/0.0	1497
Document Processing	189	2999	1.0/0.2	253	2478	0.7/0.6	220
Document Proc. LT	272	3558	1.0/0.2	274	3558	1.0/0.2	166
Dreyer Foundation	34	917	0.4/0.2	114	655	0.1/0.0	145
Electronic Invoicing	163	3345	1.0/0.8	294	2983	1.0/0.5	202
Electronic Invoicing LT	177	4048	1.0/0.8	301	3461	1.0/0.5	218
Hospital Billing	307	7662	0.4/0.0	507	6652	0.0/0.0	425
NASA CEV	497	3919	0.5/0.2	3370	3276	0.0/0.0	1832
NASA CEV LT	1310	7444	0.5/0.4	11381	6978	0.1/0.0	7948
Production	34	1809	0.1/0.0	359	1741	0.0/0.0	1205
Production LT	76	4931	0.1/0.0	1656	5349	0.0/0.0	10075
Sepsis Cases	112	887	0.6/0.0	225	562	0.1/0.0	229
Traffic Fines Mgmt.	623	7289	0.3/0.3	742	6270	0.2/0.0	647
WABO Receipt Phase	32	483	0.8/0.0	90	963	0.8/0.0	95

Table 2. Median mining time in milliseconds across 20 runs. The MINERful values to the right of either PARNEK variant are the result of confidence and interest factor thresholds (adjacent) which produce models of a size comparable to the PARNEK variant. Log names suffixed by LT have the lifecycle transition included in the event name, so that they will be interpreted as instances of distinct activities.

Looking at individual logs gives further insights. Some of the most notable logs include (multiples are w.r.t. to corresponding MINERful or PARNEK variant, respectively, unless otherwise stated):

BPIC 2012 PARNEK(COND) almost doubles precision by adding only 3 relations.

BPIC 2013 MINERful achieves the same precision with only 2 constraints.

BPIC 2017 PARNEK(COND) more than doubles precision with *fewer* relations.

Document Proc. PARNEK(COND) almost triples precision with just 2 extra relations.

D. Proc. (LT) PARNEK(FULL) gets 0.99 prec. with 150 relations vs. Baseline’s 558.

Elec. Inv. (LT) For both versions of this log, PARNEK makes significant improvements, often with *fewer* relations.

Traffic PARNEK outperforms, the COND variant doing so with one extra relation.

WABO PARNEK(COND) increases precision by 56.4% with just 3 extra relations.

PARNEK is often able to improve precision at the expense of only a small number of additional relations, and in some cases even *fewer* relations than MINERful. The converse is also true for MINERful for a handful of logs, implying that the combination of modelling language and mining algorithm are best suited for certain types of processes, being able to capture behaviour which is more *meaningful* in the given

⁵ <https://github.com/backco/qmpm>

⁶ <https://github.com/viktorija-nek/ParNek>

context. The aggregated results across logs, however, indicate that PARNEK has an overall advantage.

From table 2 we can see that PARNEK outperforms both the Baseline algorithm and MINERful on several, though not all, of the logs we evaluated. Notably, its running time is often markedly faster, the only exceptions being for the most computationally intensive, i.e. FULL, variant of PARNEK which is somewhat slower on a few logs.

Finally, PARNEK(COND) returns models with higher or near equal generalisability, as measured by 5-fold cross-validation, compared to all other miners.

6 Conclusion

In this paper we introduced the PARNEK miner, a novel discovery algorithm for DCR Graphs. We compare the miner to the state-of-the-art in DCR (baseline) and Declare (MINERful) discovery and show that PARNEK significantly outperforms the state-of-the-art in DCR discovery. In addition, it performs at least comparably to the state-of-the-art in Declare discovery. While a precise comparison between DCR Graphs and Declare is difficult to make, it can be argued that Declare relations carry more semantic meaning, making them more complex to reason about, leading to an argument in favour of PARNEK outperforming the state-of-the-art in declarative mining in general. However, this would require a more thorough study on the relative complexity of the different notations.

In addition to these empirical observations, we note that because of the compositional architecture of PARNEK it allows for better configurability and leaves room for future extensions, features that are not clearly present in the baseline algorithm. We believe that this makes PARNEK much more suitable for practical applications, a notion that is supported by expressed stakeholder interest in the integration of the algorithm into commercial DCR tools.

Future work In the future, we hope to take advantage of the fact that PARNEK is both easily configurable and extensible by developing an interactive user-guided discovery approach. This will require research into the use of negative sample data, ranking of constraints based on relevance and the use of partial domain models to guide the discovery. In addition, we are considering further extensions to the algorithm for finding models of even higher quality, for example by supporting more complex uses of the response relation.

References

1. Christoffer Olling Back, Søren Debois, and Tijs Slaats. Towards an empirical evaluation of imperative and declarative process mining. In *International Conference on Conceptual Modeling*, pages 191–198. Springer, 2018.
2. Claudio Di Ciccio and Massimo Mecella. On the discovery of declarative control flows for artful processes. *ACM Trans. Manage. Inf. Syst.*, 5(4):24:1–24:37, January 2015.
3. Søren Debois, Thomas Hildebrandt, and Tijs Slaats. Hierarchical declarative modelling with refinement and sub-processes. In *Business Process Management*, volume 8659 of *Lecture Notes in Computer Science*, pages 18–33. Springer, 2014.

	PARNEK(COND)			MINERful			PARNEK(FULL)			MINERful			Baseline		
LOGS	SIZE	PN	GEN	SIZE	PN	GEN	SIZE	PN	GEN	SIZE	PN	GEN	SIZE	PN	GEN
BPIC 2012	68	0.28	≈ 1	65	0.15	1.0	343	0.36	≈ 1	235	0.22	≈ 1	762	0.42	≈ 1
BPIC 2012 LT	99	0.26	≈ 1	83	0.14	≈ 1	1066	0.5	≈ 1	838	0.27	≈ 1	1668	0.51	≈ 1
BPIC 2013	7	0.49	1.0	0	0.51	1.0	11	0.51	1.0	9	0.51	1.0	17	0.51	1.0
BPIC 2013 LT	32	0.22	≈ 1	31	0.23	≈ 1	138	0.29	≈ 1	101	0.28	≈ 1	170	0.29	≈ 1
BPIC 2017	75	0.18	≈ 1	87	0.08	1.0	594	0.32	≈ 1	566	0.22	≈ 1	865	0.34	≈ 1
Activities	90	0.02	0.75	78	≈ 0	0.62	994	0.08	0.44	803	0.07	0.48	1084	0.07	0.48
Activities LT	228	0.04	0.75	226	0.16	0.77	3288	0.45	0.44	3247	0.42	0.51	4594	0.46	0.46
Doc. Proc.	34	0.44	1.0	32	0.16	1.0	101	0.51	1.0	70	0.23	1.0	156	0.51	1.0
Doc. Proc. LT	83	0.88	1.0	72	0.26	1.0	132	0.99	1.0	72	0.26	1.0	540	0.99	1.0
Dreyer Found.	123	0.12	0.98	120	0.17	0.98	887	0.54	0.94	431	0.3	0.96	1148	0.56	0.95
Elec. Inv.	24	0.36	1.0	49	0.2	1.0	83	0.49	1.0	65	0.2	1.0	130	0.49	1.0
Elec. Inv. LT	57	0.61	1.0	131	0.5	1.0	134	0.82	1.0	187	0.51	1.0	440	0.98	1.0
Hosp. Billing	56	0.45	≈ 1	38	0.46	≈ 1	297	0.72	≈ 1	209	0.71	≈ 1	337	0.72	≈ 1
NASA CEV	137	0.14	≈ 1	125	0.04	≈ 1	3144	0.54	0.86	2157	0.39	0.71	2415	0.43	0.86
NASA CEV LT	298	0.31	0.98	281	0.25	≈ 1	7018	0.82	0.86	6451	0.71	0.84	9996	0.88	0.86
Production	720	0.02	0.87	123	≈ 0	0.78	2916	0.09	0.63	2765	0.11	0.57	3120	0.08	0.65
Production LT	1123	0.03	0.90	763	0.01	0.66	9287	0.12	0.54	11967	0.14	0.47*	13363	0.12	0.54
Sepsis Cases	83	0.21	0.99	77	0.19	0.98	194	0.25	0.96	106	0.2	0.97	284	0.21	0.97
Traffic Fines	22	0.79	1.0	21	0.62	1.0	70	0.93	≈ 1	44	0.79	1.0	150	0.93	≈ 1
WABO Receipt	93	0.61	0.99	90	0.39	0.99	388	0.7	0.99	90	0.39	0.99	873	0.81	0.99
$\frac{1}{N} \sum(\cdot)$	$\frac{\text{PARNEK(COND)}}{\text{MINERful(COND)}}$			$\frac{\text{PARNEK(FULL)}}{\text{MINERful(FULL)}}$			$\frac{\text{PARNEK(FULL)}}{\text{PARNEK(COND)}}$			$\frac{\text{MINERful(FULL)}}{\text{MINERful(COND)}}$			$\frac{\text{PARNEK(COND)}}{\text{Baseline}}$		
SIZE	1.579			1.474			7.324			7.751			0.144		
PRECISION	2.117			1.526			2.610			5.142			0.576		

Table 3. TOP: Comparison of miners based on normalised precision (PN), generalisation (GEN), and model size. Generalisation to *out-of-sample* data is estimated as mean trace fitness on hold-out data in 5-fold cross-validation. Normalised precision and model size are based on the complete log. Two variants of the PARNEK miner are reported: the base algorithm with the additional conditions improvement from section 4.2 (COND), and the algorithm with all improvements (FULL). To the right of the results for each PARNEK variant are results for models produced by MINERful when restricted to a model size comparable to that produced by PARNEK. Baseline refers to the DCR miner from [4]. Size refers to the number of constraints/relations. Log names suffixed by LT have the lifecycle transition included in the event name, so that they will be interpreted as instances of distinct activities. (*) Due to computational limitations, this value was computed using 2-fold cross-validation.

BOTTOM: Relative improvement of miners against other miners or their own variant. Each number represents the mean increase in size and precision, respectively.

4. Søren Debois, Thomas T. Hildebrandt, Paw Høvsgaard Laursen, and Kenneth Ry Ulrik. Declarative process mining for DCR graphs. In *SAC 2017*, pages 759–764, 2017.
5. Søren Debois, Thomas T. Hildebrandt, and Tijs Slaats. Replication, refinement & reachability: complexity in dynamic condition-response graphs. *Acta Informatica*, 55(6):489–520, Sep 2018.
6. Søren Debois and Tijs Slaats. The analysis of a real life declarative process. In *Computational Intelligence, 2015 IEEE Symposium Series on*, pages 1374–1382, 2015.
7. Stijn Goedertier, David Martens, Jan Vanthienen, and Bart Baesens. Robust process discovery with artificial negative events. *Journal of Machine Learning Research*, 10(Jun):1305–1340, 2009.
8. Thomas Hildebrandt, Raghava Rao Mukkamala, Tijs Slaats, and Francesco Zanitti. Contracts for cross-organizational workflows as timed dynamic condition response graphs. *Journal of Logic and Algebraic Programming (JLAP)*, may 2013.
9. Thomas T. Hildebrandt and Raghava Rao Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In *PLACES*, 2010.
10. Richard Hull et al. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *Web Services and Formal Methods*, pages 1–24, 2011.
11. Linh Thao Ly, Stefanie Rinderle-Ma, and Peter Dadam. Design and verification of instantiable compliance rule graphs in process-aware information systems. In *Advanced Information Systems Engineering*, pages 9–23, 2010.
12. Fabrizio M. Maggi, R. P. Jagadeesh Chandra Bose, and Wil M. P. van der Aalst. Efficient discovery of understandable declarative process models from event logs. In *Advanced Information Systems Engineering*, pages 270–285, 2012.
13. F.M. Maggi, A.J. Mooij, and W.M.P. van der Aalst. User-Guided Discovery of Declarative Process Models. In *2011 IEEE Symposium on Computational Intelligence and Data Mining*. IEEE, 2011.
14. Object Management Group. Case Management Model and Notation, version 1.0. Webpage, may 2014. <http://www.omg.org/spec/CMMN/1.0/PDF>.
15. M. Pesic and W. M. P. van der Aalst. A declarative approach for flexible business processes management. In *Proc. of the 2006 international conference on Business Process Management Workshops, BPM’06*, pages 169–180. Springer-Verlag, 2006.
16. Viara Popova, Dirk Fahland, and Marlon Dumas. Artifact lifecycle discovery. *Int. J. Cooperative Inf. Syst.*, 24, 2015.
17. Stefan Schöning, Claudio Di Ciccio, Fabrizio Maria Maggi, and Jan Mendling. Discovery of multi-perspective declarative process models. In *ICSOC*, pages 87–103, 2016.
18. Tijs Slaats. *Flexible Process Notations for Cross-organizational Case Management Systems*. PhD thesis, IT University of Copenhagen, January 2015.
19. Tijs Slaats, Raghava Rao Mukkamala, Thomas Hildebrandt, and Morten Marquard. Exformatics declarative case management workflows as dcr graphs. In *Business Process Management*, volume 8094 of *Lecture Notes in Computer Science*, pages 339–354. Springer, 2013.
20. Wil Van der Aalst, Arya Adriansyah, and Boudewijn van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Know. Disc.*, 2(2):182–192, 2012.
21. M. Zeising, S. Schnig, and S. Jablonski. Towards a common platform for the support of routine and agile business processes. In *10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 94–103, Oct 2014.

DisCoveR: Accurate & Efficient Discovery of Declarative Process Models

Christoffer Olling Back¹ · Tijs Slaats¹ · Thomas Troels Hildebrandt¹ · Morten Marquard²

Received: date / Accepted: date


Abstract *Declarative* process modeling formalisms - which capture high-level process constraints - have seen growing interest, especially for modeling flexible processes. This paper presents DisCoveR, an efficient and accurate declarative miner for learning Dynamic Condition Response (DCR) Graphs from event logs. We present a precise formalization of the algorithm, describe a highly efficient bit vector implementation and present a preliminary evaluation against five other miners, representing the state-of-the-art in declarative and imperative mining. DisCoveR performs competitively with each of these w.r.t. a fully automated binary classification task, achieving an average accuracy of 96.1% in the Process Discovery Contest 2019¹. We appeal to computa-


tional learning theory to gain insight into its performance as a classifier. Due to its linear time complexity, DisCoveR also achieves much faster runtimes than other declarative miners.


Finally, we show how the miner has been integrated in a state-of-the-art declarative process modeling framework as a model recommendation tool, discuss how discovery can play an integral part of the modeling task and report on how the integration has improved the modeling experience of end-users.

Keywords Process Discovery · Declarative Process Models · Process Mining · DCR Graphs

Work supported by the Innovation Fund Denmark project *EcoKnow* (7050-00034A) and the Danish Council for Independent Research project *Hybrid Business Process Management Technologies* (DFF-6111-00337)

✉ Christoffer Olling Back
back@di.ku.dk
 <https://orcid.org/0000-0001-7998-7167>

Tijs Slaats
slaats@di.ku.dk
 <https://orcid.org/0000-0002-7435-5563>

Thomas Troels Hildebrandt
hilde@di.ku.dk
 <https://orcid.org/0000-0001-6244-6970>

Morten Marquard
mm@dcrsolutions.net

¹ Department of Computer Science, University of Copenhagen
Copenhagen, Denmark

² DCR Solutions, Copenhagen, Denmark

¹ Results are available at <https://icpmconference.org/2019/process-discovery-contest>

1 Introduction

Technologies for business process management have matured significantly since the early proposals of office automation systems and business process definition languages in the late 1970s [4, 34, 89]. Today, BPMN [33, 66, 88] has become a stable, de-facto standard notation for describing business processes. Users can choose from a number of commercial design tools and business process management systems, supporting the design and enactment of business processes. In recent years, we have even seen commercial process mining tools [48] that support the automated discovery of BPMN models from event logs [2, 86].

With the increased need to accommodate flexible, knowledge-intensive processes, notations focusing on essential rules, rather than detailed procedures have seen increased attention from researchers [32, 59, 69, 74, 79]. This approach is often characterized as *declarative* and juxtaposed with *imperative* notations like BPMN [43, 56, 70].

Highly regulated workflows, for example governmental case work processes, are particularly challenging examples, since constantly changing legislation gives rise

to changes in rules, and often an increase in complexity [28, 40]. Declarative notations are, by design, well suited to translation from natural language rules while avoiding over-specification, making them suited to capturing regulatory constraints in workflows requiring some degree of flexibility.

As part of a broader national digitalization initiative, the challenge of modeling knowledge-intensive workflows has been tackled in several collaborative projects involving Danish universities, government institutions and firms in the private sector. One such project is the EcoKnow² project which builds upon the DCR Graphs formalism [44, 59, 79].

To support the local development and maintenance of the declarative DCR models, several modeling tools have been developed [20, 24, 56], supported by formal understandability studies [5–7]. Along with the tools, a methodology for modeling with DCR has been developed, advocating an iterative and incremental, scenario-driven approach with three main tasks. First, to identify key activities and roles. Second, to perform simulations of wanted and unwanted scenarios. Finally, the modeler may either go back to add missing activities and roles or forward to the task of identifying rules that supports the wanted scenarios and forbid the unwanted scenarios.

The iterative approach lends itself extremely well to being supported by process discovery: after the users define wanted and unwanted scenarios, discovery algorithms can be used to automatically make suggestions for which rules should be added. Such a discovery algorithm needs to be both efficient and accurate. On the one hand users expect their modeling experience to be continuous without long interruptions waiting for a discovery algorithm to compute possible rules. On the other hand, they are only helped by rule suggestions that are relevant and correct in terms of the suggested scenarios: poor suggestions will only confuse the users and reduce the quality of their modeling experience.

Recently, an efficient and accurate discovery algorithm was developed for DCR Graphs and implemented in a commercial design tool [63]. One advantage of the algorithm is that it can provide accurate suggestions even with small training sets, facilitating rule discovery from large historical event logs as well as fast recommendations based on few simulated scenarios carried out as part of the scenario-driven modeling approach.

This paper is part of a special issue of the journal in connection with the Process Discovery Contest 2019, which frames process discovery as a binary classification task. The DisCoveR algorithm secured a second place in that year's contest in terms of classification accuracy. The algorithm itself was first introduced by Nekrasaite et al. in [63], the current paper expands on this initial introduction with: a com-

plete and thorough formalization of the algorithm that provides all details required for its implementation (Section 4); a novel, open source and more efficient implementation based on bit vector operations (Section 5); a novel evaluation of the algorithm based on the classification task provided by the Process Discovery Contest 2019, and a runtime comparison with flagship academic miners suggesting the miner is competitive with its peers along with a framing of process discovery in terms of computational learning theory which helps explain the key to its effectiveness in terms of regularization (Section 6); a case study showing how the algorithm has been swiftly transferred to industry through its integration in the `dcrgraphs.net` process modeling portal, leading to an enhanced modeling experience by its users (Section 7).

After surveying related work in Section 2 and introducing preliminaries in Section 3, we proceed as sketched above, concluding and proposing future directions of research in Section 8.

2 Related Work

Many declarative process notations have been developed, several with corresponding discovery algorithms [80]. One of the first of these was Declare [1, 3, 70], which was inspired by property specification patterns for linear temporal logic (LTL) [35]. Declare identified a particular set of patterns relevant for business processes and gave them semantics through a mapping to LTL formulae relevant for describing the rules governing a business process. A Declare model is therefore a collection of such patterns, and the semantics of a model is defined as the traces that satisfy the conjunction of the formulae underlying the patterns. More recently the same patterns have been formalized using colored automata [52], SCIFF [57, 58], and regular expressions [92]. Extensions to Declare include timed [90] and data [22] constraints, which were combined in MP-Declare [14] (Multi Perspective Declare), and hierarchy [95]. The first miner for Declare was the Declare Maps Miner [53], while initially using a brute-force approach, it was extended with several improvements [50] inspired by the Apriori algorithm for association rule mining [10]. More recently the miner was extended to allow for parallelization [51]. The second Declare miner to be developed was Minerful [18] which provided significant gains in efficiency. Since its introduction it has been extended with support for target-branched constraints [31], removal of redundancies and inconsistencies [16] and removal of vacuously satisfied constraints [17].

Another prominent declarative approach is the Guard-Stage-Milestone (GSM) notation [45], inspired by earlier work on artifact-centric business processes [13]. GSM aims to effectively model case management and has been a primary contributor to the development of the Case Management Model And Notation (CMMN) [65]. CMMN has seen

² (Effective, co-created and compliant adaptive case management for Knowledge workers

a relatively fast industrial and academic adoption through the development of tools and case studies [39, 47, 93]. Work on process discovery for GSM or CMMN on the other hand is still rather sparse, only one discovery algorithm has been proposed to date [71] with no working implementation.

Process discovery has also been considered for the Declarative Process Intermediate Language (DPIL) [76, 94], which is a textual, multi-perspective, declarative modeling language. Process discovery for DPIL is supported through the DPIL Miner³. In comparison to other Declarative miners, which tend to focus on the control-flow perspective of processes, the DPIL Miner instead focuses more on mining the organizational perspective [75]. Interestingly the miner has never been made publicly available and its effectiveness or accuracy can not be independently ascertained.

In more recent work it has been proposed to combine declarative and imperative discovery to produce so-called hybrid [11, 25, 73, 83] or mixed [21, 23, 91] models that combine both paradigms. Hybrid miners include the Fusion miner [84], which produces an inter-mixed Petri net and Declare model, the Hybrid Miner [54] which produces a hierarchical Petri net and Declare model, and the Precision Optimization Hybrid Miner [77] which produces a process tree in which some nodes may be Declare models.

Approaches to workflow formalization based on Classical Linear Logic, a resource-aware logic, were implemented in WorkflowFM [67, 68] which guarantees concurrent, correct-by-construction processes. The framework was applied to intra-hospital patient transfers in [55].

Temporal logics have also been used to model phenomena which would not be considered workflows, such as robot motion [36], naval traffic and train network monitoring [46].

Process mining is often framed as an inherently *descriptive* rather than *predictive* data mining problem, which precludes the use of standard evaluation metrics familiar in classification and regression tasks. This is largely due to the assumption that an event log represents only positive examples [37]. Some authors have addressed this by developing techniques to generate artificial negative examples [38].

Finally, DCR Graphs were inspired by event structures [64] and developed after Declare was shown to not be sufficiently expressive in modeling industrial cases [61]. In contrast to Declare, the semantics of DCR Graphs are defined as transformations on the markings of the events. This allows modellers to straightforwardly reason about the execution semantics of a model by simulating it and observing the changes to the markings as events are executed. [56] Since their inception, DCR Graphs have been extended with nesting [41], time [42], data [20, 60, 82], and hierarchy [26].

3 Preliminaries

We present here the formal definitions of processes and event logs, necessary to give a formal presentation of the task of process discovery in terms of computational learning theory, as well as the DCR Graphs formalism.

Definition 1 (Processes and Event Logs)

- An *alphabet* Σ is a finite set of symbols denoting activities. We denote by Σ_L activities present in log L .
- Σ^* and Σ^ω denote countably infinite sets of finite, respectively infinite, sequences over Σ .
- A *process* is a pair (P, \mathbb{P}_P) where P is a set of allowable sequences of activities along with an associated probability distribution \mathbb{P}_P over P . The probabilistic framing is required for consistency with the statistical metrics (e.g. accuracy) used for evaluation in Section 6.
- An *event*, denoted ς , is a particular occurrence of an activity.
- A *trace* $\sigma \in \Sigma^* \cup \Sigma^\omega = \langle \varsigma_1, \dots, \varsigma_i, \dots \rangle$ represents a sequence of activities, with $i \in \mathbb{N}$. A trace can be seen as a partial mapping:

$$\sigma(i) : \mathbb{N} \hookrightarrow \Sigma$$

- A *process model* h defines a semantics such that the *language* ℓ of h denotes the set of traces accepted by h . That is,

$$\ell(h) \subseteq \Sigma^* \cup \Sigma^\omega$$

and for some process (P, \mathbb{P}_P) we have $P = \ell(h)$ if h is a perfect model of the process. Note that h may be agnostic regarding \mathbb{P}_P .

- Finally, a *log* L is a multiset representing the number of occurrences of different traces:

$$L = \left\{ \sigma_1^{m(\sigma_1)}, \dots, \sigma_n^{m(\sigma_n)} \right\}$$

where $m(\sigma_k) \in \mathbb{N}$ denotes the multiplicity of σ_k . As L is essentially a *sample* from (P, \mathbb{P}_P) , it is necessary to consider trace multiplicities rather than collapsing the log to a set.

Note the assumption of strict monotonicity implied by this definition of traces. That is, for all $i, j \in \mathbb{N}$ we have that

$$i < j \implies \sigma(i) \prec \sigma(j)$$

where \prec denotes “precedes”, and also that

$$i = j \implies \sigma(i) = \sigma(j).$$

The definition of a trace as a function mapping from a timestamp domain to the codomain of individual activities implies that no two events can share the exact same timestamp (otherwise σ would not be a function). We note this,

³ <http://www.kppq.de/miner.html>

in part, due to the observation that shared timestamps are not uncommon in real data sets. Nonetheless, the present formalization of traces is widely accepted and sufficient for the study at hand.

Definition 2 (Process Discovery) *Process discovery* refers to a procedure that derives a process model from an event log. Let \mathcal{L} denote the set of all valid event logs and \mathcal{H}_F the set of process models encodable by some process modeling formalism F . A process discovery algorithm γ is a mapping from logs to models:

$$\gamma : \mathcal{L} \rightarrow \mathcal{H}_F$$

Examples of F include Petri nets, sound Petri nets, Work-Flow nets, R/I-nets, Declare maps, and of course DCR Graphs. In other words, \mathcal{H}_F is our *hypothesis space* to which our learning algorithm is restricted.

By extension, we can view the overall task as a mapping from a log to a language, i.e. a subset of all possible traces:

$$\ell(\gamma) : \mathcal{L} \rightarrow 2^{\Sigma^* \cup \Sigma^\omega}$$

Where $2^{\mathcal{X}}$ denotes the *powerset* of set \mathcal{X} . To see this, consider that for some $L \in \mathcal{L}$, we have $\gamma(L) = h$ and $\ell(h) \subseteq 2^{\Sigma^* \cup \Sigma^\omega}$. That is, $\ell(\gamma(L)) \subset \Sigma^* \cup \Sigma^\omega$. This view of process discovery will lead naturally to the classification task, and reduce the choice of modeling formalism F to an intermediate step w.r.t. classification.

Definition 3 (DCR Graphs) DCR Graphs consist of a set of events with three associated unary predicates: *executed*, *pending*, and *included* which together constitute the marking (i.e. state) of a DCR Graph. Moreover, four binary relations are defined between events. In order to be executed, an event must be included and satisfy any relevant relations.

Formally, a *dynamic condition response graph* is a tuple

$$g = (\mathcal{E}, m, A, \bullet \rightarrow, \rightarrow \bullet, \rightarrow +, \rightarrow \%, l)$$

where

- \mathcal{E} is a set of “events” (analogous to transitions in a Petri net, and not to be confused with events in a trace, see l).
- $m \in 2^{\mathcal{E}} \times 2^{\mathcal{E}} \times 2^{\mathcal{E}}$ is the *marking*
- A is the set of *activities*.
- $\rightarrow \bullet \in \mathcal{E} \times \mathcal{E}$ is the set of *condition* relations.
- $\bullet \rightarrow \in \mathcal{E} \times \mathcal{E}$ is the set of *response* relations.
- $\rightarrow + \in \mathcal{E} \times \mathcal{E}$ is the set of *includes* relations.
- $\rightarrow \% \in \mathcal{E} \times \mathcal{E}$ is the set of *excludes* relations.
- $\rightarrow + \cap \rightarrow \% = \emptyset$.
- $l : \mathcal{E} \rightarrow A$ is a labeling function mapping every “event” to an activity.

A DCR Graph marking $m = (\text{Ex}, \text{Pe}, \text{In})$ represents events which have previously been *executed*, *pending* events to be executed or excluded, and events currently *included*. For finite traces, a DCR Graph is defined to be *accepting* when $\text{Pe} \cap \text{In} = \emptyset$, i.e. no pending events are currently included. For infinite traces, accepting states are defined in the limit as with Büchi automata, to which DCR graphs can be translated [62].

The execution semantics of DCR Graphs requires that for an event e to be executed, it must fulfill the following criteria:

- e must be *included*, i.e. $e \in \text{In}$
- If any condition relations exist s.t. $e' \rightarrow \bullet e$, then all such e' must have been executed, *or excluded*, i.e. $e' \in \text{Ex}$ or $e' \notin \text{In}$. In this way, conditions can be nullified by excluding the source event. The latter is the “dynamic” aspect of DCR Graphs.

Furthermore, if e is executed, the marking m will change as follows:

- If any response relations exist s.t. $e \bullet \rightarrow e'$, then all such e' will become *pending*, i.e. $e' \in \text{Pe}$
- If any excludes relations exist s.t. $e \rightarrow \% e'$ then any included e' will become *excluded*, i.e. $e' \notin \text{In}$.
- If any includes relations exist s.t. $e \rightarrow + e'$ then any excluded e' will become *included*, i.e. $e' \in \text{In}$.

An important point to note regards the labeling function l , which may map more than one event to the same activity (analogous to Petri nets with duplicate transitions). This can potentially result in a non-deterministic model. In the algorithm presented here, only bijective labeling functions are considered, so each event is mapped to exactly one activity and vice versa.

Example Consider a DCR Graph consisting of 4 events with a one-to-one mapping to activities: a, b, c, d :

- Initial marking:
 - Executed: \emptyset
 - Pending: a
 - Included: a, c, d
- Relations
 - $a \rightarrow \bullet b$
 - $a \bullet \rightarrow b$
 - $b \rightarrow \bullet a$
 - $b \bullet \rightarrow a$
 - $c \rightarrow + b$
 - $d \rightarrow \% b$
 - $d \rightarrow \% d$

Accepting run 1: $\langle a \rangle$. The model begins in a non-accepting state since a is both pending and included. Since b is not included, $b \rightarrow \bullet a$ does not come into effect. After a is executed,

b becomes pending, but since it is not included, the model is in an accepting state.

Accepting run 2: $\langle a, c, b, d, a \rangle$. After a is executed, b becomes pending. Executing c causes b to be included as well. Now the model is in a non-accepting state. Executing b causes a to become pending. Executing d excludes b and d itself. Finally, a , which is still pending and included is executed, which causes b to become pending, but since it is not included, the model is in an accepting state.

Non-accepting run: $\langle c, d, c \rangle$. When c is executed, b becomes included, but cannot be executed due to the condition relation $a \rightarrow \bullet b$. Likewise, a is unable to execute due to $b \rightarrow \bullet a$. Executing d excludes b and d itself, releasing a from $b \rightarrow \bullet a$. At this point, executing a will lead to an accepting state. However, if instead c is executed again, b is included again and $b \rightarrow \bullet a$ comes into effect and now d cannot be executed to exclude b . The graph is now locked in a permanently non-accepting state as neither a , nor b can ever be executed because of their mutual conditions, yet a remains forever included and pending.

4 Algorithm

In this section we formally describe the *ParNek* algorithm underlying DisCoveR. Note the distinction we draw between the fundamental algorithm, ParNek, and the specific implementation, DisCoveR, presented in Section 5. This distinction is also reflected in the formal, functional description in this section which remains agnostic to concrete implementation details, e.g. for extracting the sets of relations defined in Table 2.

The algorithm always produces perfectly fitting models, i.e. all traces in the log will be replayable on the generated model. The algorithm proceeds in the following steps:

1. A set of candidates for four relation patterns is constructed.
2. Additional excludes relations are added based on predecessor and successor relations.
3. Additional includes/excludes patterns are added analogous to NOTCHAINSUCCESSION relations.
4. Redundant excludes relations are removed.
5. Redundant condition and response relations are removed via transitive reduction.
6. Additional condition relations are discovered using a limited replay strategy.
7. A final transitive reduction is performed for condition relations.

We will refer to seven relation templates from the LTL-based modeling language *Declare*. The relations are described in words in Table 1 with analogous DCR relations. These particular Declare constraints have been selected based

on their ability to be mapped to DCR Graph relations that can be composed orthogonally (thereby ensuring the perfect fitness requirement of the miner), the possibility to detect them in linear time and extensive experimentation to determine which combination of constraints yielded the best balance between precision and simplicity on real-life logs. Formal specifications of functions for identifying relations satisfied by the log are given in Table 2 (again, these are only specifications, not implementations). In the description that follows, we refer to lines in the high-level control flow pseudocode in Algorithm 1.

The first step of the ParNek algorithm is the initialization of a DCR Graph, after which we begin adding relations using a number of strategies.

Initialization (lines: 2-5) We begin by defining a set of events

$$E \equiv \{1, \dots, |\Sigma_L|\}$$

containing the same number of events as distinct activities present in the log, the latter defining our set of activities

$$A \equiv \Sigma_L.$$

The labeling function

$$l : E \rightarrow \Sigma_L; i \mapsto s_i$$

is a bijective mapping between events and activities. So for all intents, events and activities are equivalent. Finally we assign an initial marking

$$m \equiv (\emptyset, \emptyset, E)$$

in which all events are included, none are pending, and none are executed. This marking does not change, and is returned in the final graph.

Self-Exclusions - *ATMOSTONE* (line: 11): We begin with activities for which the log satisfies the *ATMOSTONE* relation. Any activity s satisfying this unary relation are mapped onto the binary self-exclusion relation $s \rightarrow \% s$.

Responses - *RESPONSE* (line: 13): All pairs of *distinct* activities s and t for which the log satisfies the *RESPONSE* relation, are mapped directly onto the response relation $s \bullet \rightarrow t$.

Conditions - *PRECEDENCE* (line: 12): All pairs of *distinct* activities s and t for which the log satisfies the *PRECEDENCE* relation, are mapped directly onto the condition relation $s \rightarrow \bullet t$. While this forms the basis of the condition relation, more will be added in lines 32-34.

Declare	DCR Graphs	Description
ATMOSTONE(a)	$a \rightarrow\% a$	Activity a can occur 0 or 1 time
RESPONSE(a, b)	$a \bullet \rightarrow b$	After a occurs, b must eventually occur
PRECEDENCE(a, b)	$a \rightarrow\bullet b$	Before b can occur, a must have occurred
ALTERNATEPRECEDENCE(a, b)	$a \rightarrow+ b$ and $b \rightarrow\% b$	For b to occur, a must occur exactly once prior
CHAINPRECEDENCE(a, b)	<i>See caption</i>	For b to occur, a must occur immediately prior
NOTCHAINSUCCESION(a, b)	$a \rightarrow\% b$	Activity b may not occur immediately after a
NOTCOEXISTENCE(a, b)	$a \rightarrow\% b \wedge b \rightarrow\% a$	Activities a and b may not co-occur in the same trace

Table 1 Relevant constraint templates from Declare. The CHAINPRECEDENCE relation is not straightforward to encode in DCR Graphs relations and in fact, ParNek looks for evidence of CHAINPRECEDENCE relations, but encodes them as $a \rightarrow+ b, b \rightarrow\% b$, which is not exactly equivalent.

Includes/Excludes - CHAINPRECEDENCE (line: 14-15): The first step in populating $\rightarrow+$ and adding further self-exclusions to $\rightarrow\%$, is based on identifying CHAINPRECEDENCE relations. However, encoding CHAINPRECEDENCE in DCR Graphs is less straightforward than ALTERNATEPRECEDENCE, which is (nearly⁴) captured by an include and self-excludes. Since ALTERNATEPRECEDENCE *subsumes* CHAINPRECEDENCE, it is safe to check for evidence of the more restricted CHAINPRECEDENCE, yet add ALTERNATEPRECEDENCE to the model.

Excludes - Predecessor/Successor (lines: 17-21): Further excludes relations are found by defining two relations:

$$Predecessor(L) \text{ and } Successor(L)$$

which return the sets of *all possible* predecessors and successors of an activity, respectively.

Based on the observation that a log in which activities s and t never co-occur in the same trace satisfies the NOTCOEXISTENCE(s, t) relation, we add $s \rightarrow\% t$ and $t \rightarrow\% s$ (lines: 17-18). However, due to the subsequent removal of redundant exclusions (lines: 26-27), the NOTCOEXISTENCE relation cannot be guaranteed to hold since one or both of the exclusions may be removed.

Furthermore, if s is observed to precede, but never succeed t , and if no self-exclusion $s \rightarrow\% s$ has been found, we add $t \rightarrow\% s$ (lines: 19-21).

In order to restrain model complexity, only one exclusion relation is included for each target activity by means of the *ChooseOneRelation* function. At present, this function is implemented in a naive (but fast and deterministic), first-come manner with a more sophisticated approach being left for future work.

Includes & Excludes - NOTCHAINSUCCESION (lines: 23-24): To identify further includes and excludes relations, we rely on

⁴ In order to completely capture ALTERNATEPRECEDENCE, the target activity needs to be excluded in the initial marking. This can lead to complications w.r.t. other relations in which the target is source, and is therefore omitted.

NotChainSuccession(L) as well as *Between(L)*, which simply identifies activities occurring between two other activities in a log.

Put simply, if we never observe s followed immediately by t , we add an exclusion $s \rightarrow\% t$ (NOTCHAINSUCCESION). If however, t occurs after s , with some sequence of intermediate activities s.t. we have $\langle \dots, s, u_1, \dots, u_n, t, \dots \rangle$, then we allow all intermediate events to re-include t . That is, for all $1 \leq i \leq n$, we add $u_i \rightarrow+ t$.

Remove Redundant Excludes (lines: 26-27): Here we remove redundant excludes relations based on the observation that if activity r always precedes s , and if $r \rightarrow\% t$, then adding $s \rightarrow\% t$ is redundant. It should be noted that this redundancy does not hold if some u occurs between r and s and $u \rightarrow+ t$. Presently, this caveat is ignored, potentially leading to a decrease in model precision, but allowing for an enormous reduction in model complexity.

Limited Transitive Reduction (lines: 29-30 and 36): The condition and response relations satisfy the transitive property when seen in isolation. That is, if we have $s \rightarrow\bullet t$ and $t \rightarrow\bullet u$, then $s \rightarrow\bullet u$. In this case, $s \rightarrow\bullet u$ is superfluous. The caveat, *seen in isolation*, is crucial however, since if the same model has $v \rightarrow\% t$ for some v , then t may become excluded, annulling the implicit $s \rightarrow\bullet u$. Formally,

$$s \rightarrow\bullet t \wedge t \rightarrow\bullet u \wedge \nexists v. v \rightarrow\% t \models s \rightarrow\bullet u$$

In fact, we can safely remove redundant $s \rightarrow\bullet u$ despite the presence of an interfering excludes relation (that is, we ignore $\nexists v. v \rightarrow\% t$). The removal is safe in the sense that this can only result in a more permissive model, i.e. we do not risk arriving at a model on which the log cannot be replayed. The downside is a less precise model, which may permit behavior which ought to be forbidden.

A limited-horizon transitive reduction is performed which considers only relations between and activity and its neighbors' neighbors, but not further, in order to constrain computational complexity. This is applied to all condition and

response relations prior to the final step of discovering additional condition relations, and once again on condition relations afterwards. In many models the reduction in relations is very substantial. See Figure 1 for a graphical illustration.

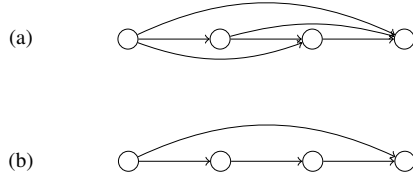


Fig. 1 Transitive reduction with a limited horizon: graph (a) has the same reachability/transitive closure as the reduced graph (b), but redundant edges within a 2-edge horizon have been removed.

Additional Conditions (lines: 32-34): The first set of conditions we added based on the PRECEDENCE relations were conservative in that this relation was observed to hold unconditionally across traces. We can now add less obvious condition relations, taking advantage of semantics added to our model by inclusion and exclusion relations.

We start by adding $s \rightarrow \bullet t$ if s occurs before the *first* occurrence of t in *some* trace. For those traces in which s does not precede the first t , it may be the case that at the time of executing t , that s is currently excluded, e.g. if the relation $u \rightarrow \% s$ is present and u is observed prior to t , and s has not been re-included. Recall that DCR Graphs semantics dictate that a relation does not apply when the source activity is excluded.

Since only includes and excludes relations are determinative for the validity of these candidate relations, we can utilize a limited replay strategy based on these relations alone. This approach is less computationally demanding than using the full model.

5 The DisCoveR Miner

In the previous section we provided a formal, functional characterization of the ParNek algorithm. In the current section we show how the algorithm was operationally implemented as the DisCoveR miner. The whole JAVA source code is provided as open source (licensed under LGPL-3.0) at [85]. As the full source code is too large to include in this paper we will at various times provide a skeleton of the code and refer to the repository for the full details, note that this means that the listings below may at times obfuscate some details from the actual source code, or include additional comments, when class names and line numbers are mentioned, they refer specifically to the release version 1.0.1.

The primary contribution of the implementation is its run-time complexity, expressed in terms of the size of the event log (L) and in terms of the number of unique activities in the log (A). This is achieved through two primary means. First of all, instead of computing the various functions of Table 2 naively by continuously re-parsing the log, we first build an abstraction of the log, which allows us to afterwards compute these functions in $\mathcal{O}(A^2)$, which in turn makes the main Algorithm 1 independent to the size of the log, except for the computation of additional conditions. Secondly, by using bit vector operations for 1) the building of the abstraction, 2) the computation of additional conditions and 3) the DCR Graph semantics, we reduce their complexity to be respectively $\mathcal{O}(L * A)$, $\mathcal{O}(L)$, and $\mathcal{O}(1)$. This means that the combined complexity of the miner is $\mathcal{O}((L * A) + A^2)$, with the log size usually dominating. The bitvector implementation of DCR Graphs was inspired by earlier work by Debois et al [24,49].

Why bit vectors? A bit vector (also bit array or bit set) is an array of bits (i.e. Booleans) that exposes bitwise operations. This allows the compiler to map the data structure directly to bitwise machine instructions, making computations on them extremely fast.

5.1 DCR Graph semantics

We first show how we used bit vectors to improve the efficiency of replaying DCR Graphs. Note that BitSets are JAVA's implementation of bit vectors, the marking of a DCR Graph can then be represented as such:

```
public BitSet executed = new BitSet();
public BitSet included = new BitSet();
public BitSet pending = new BitSet();
```

Listing 1 [85]: BitDCRMarking, ln. 7-9

For example, let us assume that we have three activities with respective indices A (1), B (2), and C (3). If A and C have been previously executed, then their executed states can be represented as the bit vector:

```
executed = [true, false, true];
```

We can similarly represent relations as matrices, encoded in practice as hashmaps of bit vectors to allow fast lookup of the relations of a particular activity:

```
public HashMap<Integer, BitSet>
conditionsFor = new HashMap<>();
public HashMap<Integer, BitSet>
responsesTo = new HashMap<>();
public HashMap<Integer, BitSet>
excludesTo = new HashMap<>();
public HashMap<Integer, BitSet>
includesTo = new HashMap<>();
```

Listing 2 [85]: BitDCRGraph, ln. 28-33

```

input : A log  $L$ 
output: A DCR Graph  $G$ 

1                                     // INITIALIZATION
2    $E \equiv \{1, \dots, |\Sigma_L|\}$                                      // set of events
3    $A \equiv \Sigma_L$                                                  // activities in log
4    $l \equiv i \in E \mapsto s_i \in \Sigma_L$                          // bijective labeling (events and activities)
5    $m \equiv (\emptyset, \emptyset, E)$                                    // initial marking
6    $\rightarrow+ \equiv \emptyset$                                                // set of includes relations
7    $\rightarrow\% \equiv \emptyset$                                            // set of excludes relations
8    $\rightarrow\bullet \equiv \emptyset$                                          // set of condition relations
9    $\bullet\rightarrow \equiv \emptyset$                                            // set of response relations

10                                     // ADD 'DECLARE' TEMPLATES
11  $\rightarrow\% := \rightarrow\% \cup \{ (s, s) \mid s \in AtMostOne(L) \}$        // self exclusions
12  $\rightarrow\bullet := \rightarrow\bullet \cup Precedence(L)$                        // condition relations
13  $\bullet\rightarrow := \bullet\rightarrow \cup Response(L)$                            // response relations
14  $\rightarrow+ := \rightarrow+ \cup \{ (s, t) \mid s \neq t \wedge (s, t) \in ChainPrecedence(L) \}$  // alternate precedence
15  $\rightarrow\% := \rightarrow\% \cup \{ (t, t) \mid \exists s, s \neq t. (s, t) \in ChainPrecedence(L) \}$  // alternate precedence

16                                     // ADD ADDITIONAL EXCLUDES
17  $\rightarrow\% := \rightarrow\% \cup ChooseOneRelation( \{ (s, t) \mid (s, t) \notin Predecessors(L) \wedge$  // not coexistence
18                                      $(s, t) \notin Successors(L) \wedge s \neq t \} )$ 
19  $\rightarrow\% := \rightarrow\% \cup ChooseOneRelation( \{ (t, s) \mid (s, t) \in Predecessors(L) \wedge$  // not succession
20                                      $(t, s) \notin Successors(L) \wedge$ 
21                                      $(s, s) \notin \rightarrow\% \wedge s \neq t \} )$ 

22                                     // ADDITIONAL INCLUDES/EXCLUDES
23  $\rightarrow\% := \rightarrow\% \cup NotChainSuccession(L)$                        // not chain succession
24  $\rightarrow+ := \rightarrow+ \cup \{ (u, t) \mid \exists s. (s, t) \in NotChainSuccession(L) \wedge (s, u, t) \in Between(L) \}$ 

25                                     // REMOVE 'REDUNDANT' EXCLUSIONS
26  $\rightarrow\% := \rightarrow\% \setminus \{ (s, t) \mid \exists u. (u, t) \in \rightarrow\% \wedge$ 
27                                      $(u, s) \in AlternatePrecedence(L) \}$ 

28                                     // REMOVE 'REDUNDANT' CONDITIONS/RESPONSES
29  $\bullet\rightarrow := \bullet\rightarrow \setminus \{ (s, t) \mid (s, u) \in \bullet\rightarrow \wedge (u, t) \in \bullet\rightarrow \}$ 
30  $\rightarrow\bullet := \rightarrow\bullet \setminus \{ (s, t) \mid (s, u) \in \rightarrow\bullet \wedge (u, t) \in \rightarrow\bullet \}$ 

31                                     // ADD ADDITIONAL CONDITIONS
32  $\rightarrow\bullet := \rightarrow\bullet \cup \{ (s, t) \mid (\exists \sigma \in L. \forall k. s = \sigma(i) \wedge t = \sigma(j) = \sigma(k) \wedge i < j \leq k) \wedge$ 
33                                      $(\forall \sigma \in L. \forall i > j. s = \sigma(i) \wedge t = \sigma(j) \wedge \exists h < j. (\sigma(h), s) \in \rightarrow\% \wedge$ 
34                                      $\nexists g < j. g > h \wedge (\sigma(g), s) \in \rightarrow+ ) \}$ 

35                                     // REMOVE 'REDUNDANT' CONDITIONS
36  $\bullet\rightarrow := \bullet\rightarrow \setminus \{ (s, t) \mid (s, u) \in \bullet\rightarrow \wedge (u, t) \in \bullet\rightarrow \}$ 

37 return  $(E, M, A, \bullet\rightarrow, \rightarrow\bullet, \rightarrow+, \rightarrow\%, l)$  // RETURN DCR GRAPH

```

Algorithm 1: High-level control flow of the mining algorithm.

AtMostOne :			
$L \mapsto \left\{ s \mid \begin{array}{l} s \in \Sigma_L \wedge i, j \in \mathbb{N} \\ \wedge \forall \sigma \in L \forall i, j. \quad s = \sigma(i) = \sigma(j) \implies i = j \end{array} \right\}$			
Response :			
$L \mapsto \left\{ (s, t) \mid \begin{array}{l} s, t \in \Sigma_L \wedge i, j \in \mathbb{N} \\ \wedge \forall \sigma \in L \forall i. \quad s = \sigma(i) \implies \exists j \left(t = \sigma(j) \wedge i < j \right) \end{array} \right\}$			
Precedence :			
$L \mapsto \left\{ (s, t) \mid \begin{array}{l} s, t \in \Sigma_L \wedge i, j \in \mathbb{N} \\ \wedge \forall \sigma \in L \forall i. \quad t = \sigma(i) \implies \exists j \left(s = \sigma(j) \wedge j < i \right) \end{array} \right\}$			
AlternatePrecedence :			
$L \mapsto \left\{ (s, t) \mid \begin{array}{l} s, t \in \Sigma_L \wedge i, j, k \in \mathbb{N} \\ \wedge \forall \sigma \in L \forall i. \quad t = \sigma(i) \implies \exists j \left(s = \sigma(j) \wedge j < i \wedge \right. \right. \\ \left. \left. \nexists k \left(t = \sigma(k) \wedge j < k < i \right) \right) \end{array} \right\}$			
ChainPrecedence :			
$L \mapsto \left\{ (s, t) \mid \begin{array}{l} s, t \in \Sigma_L \wedge i \in \mathbb{N} \\ \wedge \forall \sigma \in L \forall i. \quad t = \sigma(i) \implies s = \sigma(i-1) \end{array} \right\}$			
NotChainSuccession :			
$L \mapsto \left\{ (s, t) \mid \begin{array}{l} s, t \in \Sigma_L \wedge i \in \mathbb{N} \\ \wedge \nexists \sigma \in L \forall i. \quad \left(s = \sigma(i) \wedge t = \sigma(i+1) \right) \end{array} \right\}$			
Predecessors :			
$L \mapsto \left\{ (s, t) \mid \begin{array}{l} s, t \in \Sigma_L \wedge i, j \in \mathbb{N} \\ \wedge \exists \sigma \in L \exists i, j. \quad \left(s = \sigma(i) \wedge t = \sigma(j) \wedge i < j \right) \end{array} \right\}$			
Successors :			
$L \mapsto \left\{ (s, t) \mid \begin{array}{l} s, t \in \Sigma_L \wedge i, j \in \mathbb{N} \\ \wedge \exists \sigma \in L \exists i, j. \quad \left(s = \sigma(i) \wedge t = \sigma(j) \wedge i > j \right) \end{array} \right\}$			
Between :			
$L \mapsto \left\{ (s, u, t) \mid \begin{array}{l} s, u, t \in \Sigma_L \wedge i, j, k \in \mathbb{N} \\ \wedge \exists \sigma \in L \exists i, k, j. \quad \left(s = \sigma(i) \wedge u = \sigma(k) \wedge t = \sigma(j) \right. \right. \\ \left. \left. \wedge i < k < j \right) \end{array} \right\}$			
ChooseOneRelation :			
$R \in 2^{\Sigma \times \Sigma} \mapsto (s, t) \in R$			

Table 2 Formal definitions of helper functions which return sets of relevant relations. All functions have event logs as their domain (\mathcal{L}), except *ChooseOneRelation*.

Continuing on the previous example, if we have a condition from A to B and from B to C, the data structure `conditionsFor` would be constructed as follows:

```
conditionsFor.put(1, [false, false, false]);
conditionsFor.put(2, [true, false, false]);
conditionsFor.put(3, [false, true, false]);
```

Given these definitions, the semantics of DCR Graphs can be expressed as a short list of bitvector operations. Note that the `get()` method retrieves the bit at a given index and that the `intersects` method first applies an AND operation on two vectors and afterwards checks if the result is 0. Enabledness of events can be computed as follows:

```
public Boolean enabled(final BitDCRMarking
    marking, final int event) {
    // The event is not included.
    if (!marking.included.get(event))
        return false;
    // Any of the conditions for the event are
    // included and have not been executed.
    if (conditionsFor.get(event).intersects(
        marking.blockCond()))
        return false;
    return true;
}
// Method on the class BitDCRMarking
public BitSet blockCond() {
    return included.clone().andNot(executed);
}
```

Listing 3 [85]: BitDCRGraph, ln. 96-116 & BitDCRMarking, ln. 11-21

First we check the index of the included bit vector corresponding to the event, after we check if any of the conditions for the event are current included and not executed. The latter requires two bitwise operations: first we subtract the executed from the included events, giving us a bit vector representing those events that are currently included, but have not yet been executed, after we check if this bit vector intersects (i.e. checking if the bitwise AND is greater than 0) with the bitvector representing the conditions for the event. Note that a more straightforward, but less efficient implementation of DCR Graphs would loop over a data structure containing all conditions to achieve a similar result.

Likewise, the execution of an event can be computed as follows:

```
public BitDCRMarking execute(final
    BitDCRMarking marking, final int event) {
    // Copy the previous marking
    BitDCRMarking result = marking.clone();
    // Set the event as executed
    result.executed.set(event);
    // Clear the event as no longer pending
    result.pending.clear(event);
    // Add all new pending responses
    result.pending.or(responsesTo.get(event));
    // Exclude excluded events
    result.included.andNot(excludesTo.get(
        event));
    // Include included events
    result.included.or(includesTo.get(event));
    return result;
}
```

Listing 4 [85]: BitDCRGraph, ln. 153-174

Here, we first set the bit that corresponds to the executed event in the executed bit vector to true. We then set the bit that corresponds to the event in the pending bit vector to false. Afterwards we add any new pending responses through the bitwise OR operation on the pending bitvector and the responseTo bitvector for the executed event (which represents those events that are a response to the event). Then we remove excluded events from the included bitvector by subtracting the excludesTo bitvector for the executed event. Finally we add included events to the included bitvector through a bitwise OR with the includesTo bitvector.

As before, because hashmap lookup and bitvector operations are constant, a function that would usually loop over the sets of relations becomes a short list of constant operations.

This implementation of DCR Graphs allows for extremely fast replay of logs, which significantly reduces the duration of the *Additional Conditions* part of the algorithm, which requires a replay of the log on the graph that has been found up to that point. We'll further address how we reduced the computation of *Additional Conditions* to linear time later in this section.

MARKING AT TIMESTEP t		
	10000000	executed
	01001000	pending
	01001001	included

EXECUTE EVENT 4		
	10000000	executed
OR	00001000	event 4
	10001000	executed'

	01001000	pending
AND	11110111	not event 4
	01000000	
OR	00100000	responsesTo event 4
	01100000	pending'

	01001001	included
AND	11111110	not excludesTo event 4
	00001000	
OR	00100000	includesTo event 4
	01101000	included'

MARKING AT TIMESTEP $t + 1$		
	10001000	executed
	01100000	pending
	01101001	included

Table 3 Example of bit operations involved in execute method (see Listing 4).

5.2 Abstracting the log

To avoid repeating computations, we separate the mining process into two steps: first we build a number of relevant abstractions of the log, which we then use afterwards during the actual model building steps as described in Section 4. This separation of concerns ensures that there is a central part of the code where we parse the log, with all other parts of the algorithm working only on these abstractions, which are bounded by the number of activities ($\mathcal{O}(A^2)$) and not the log size. To increase the efficiency of the log abstraction mechanism, we also store and compute these abstractions through bit vector operations. The listing below shows their definition:

```
public HashMap<Integer, BitSet>
    chainPrecedenceFor = new HashMap<>();
public HashMap<Integer, BitSet> precedenceFor
    = new HashMap<>();
public HashMap<Integer, BitSet> responseTo =
    new HashMap<>();
public HashMap<Integer, BitSet> predecessor =
    new HashMap<>();
public HashMap<Integer, BitSet> successor =
    new HashMap<>();
public BitSet atMostOnce = new BitSet();
```

Listing 5 [85]: BitParNekLogAbstractions, ln. 37-50

Below we show how the abstractions are computed. The `parseTrace` method is called once for each trace in the log. Note that the method does not require a nested iteration over the log or current trace, only a single nested iteration over the activities to compute responses. Therefore the complexity of computing the abstractions is $\mathcal{O}(L * A)$. For convenience, logs are transformed into lists of integers, this allows for straightforward mapping of activities to the indices of the bit vectors and efficient storage of the log for later reuse.

```
public void parseTrace(List<Integer> t) {
    // We keep track of which activities were
    // seen at least once before
    BitSet localAtLeastOnce = new BitSet();
    // We keep track of which activities were
    // seen only before another activity
    HashMap<Integer, BitSet> seenOnlyBefore =
        new HashMap<>();
    // We keep track of previously seen
    // activity
    int last_i = -1;

    // For each event in the trace:
    for (int i : t) {
        // Predecessors: any activities seen at
        // least once before i
        this.predecessor.get(i).or(
            localAtLeastOnce);
        // i occurs more than once
        if (localAtLeastOnce.get(i))
            this.atMostOnce.clear(i);
        localAtLeastOnce.set(i);
        // Precedence for (i): any activity that
        // occurred before the first instance
        this.precedenceFor.get(i).and(
            localAtLeastOnce);
        // ChainPrecedence for (i): any activity
        // that occurred before i in every
        // instance.
        if (last_i != -1) {
            BitSet bs = new BitSet();
            bs.set(last_i);
            this.chainPrecedenceFor.get(i).and(bs);
        } else {
            this.chainPrecedenceFor.get(i).and(
                new BitSet());
        }
        // To later compute responses we track
        // which events were seen before i and
        // not after.
        if (this.responseTo.get(i).cardinality()
            > 0) {
            seenOnlyBefore.put(i, (BitSet)
                localAtLeastOnce.clone());
        }
        for (int j : seenOnlyBefore.keySet()) {
            seenOnlyBefore.get(j).clear(i);
        }
        last_i = i;
    }

    // Responses: those events that always
    // occur after j
    for (int j : seenOnlyBefore.keySet()) {
```

```
        this.responseTo.get(j).and(
            localAtLeastOnce);
        this.responseTo.get(j).andNot(
            seenOnlyBefore.get(j));
    }
}
```

Listing 6 [85]: BitParNekLogAbstractions, ln. 156-217

To avoid unnecessary computations embedded in the main parsing of the log, we exploit the fact that the predecessor and successor functions are each other's dual and compute the successor function after the log has been parsed:

```
public void finish() {
    for (int i : this.predecessor.keySet()) {
        for (int j : this.predecessor.keySet()) {
            {
                if (this.predecessor.get(i).get(j)) {
                    this.successor.get(j).set(i);
                }
            }
        }
    }
}
```

Listing 7 [85]: BitParNekLogAbstractions, ln. 219-230

5.3 Mining from log abstractions

After creating the log abstractions we start the discovery task. For the sake of brevity we will not show source code here, but refer to [85]: BitParNeks, ln. 75-228. In short, the implementation follows largely the steps described in Algorithm 1. The key difference is in the additional condition step, where we avoid having nested loops over the traces by implementing this function as follows (we only show the most relevant parts, for the full method we refer to [85]):

```
public void findAdditionalConditions(
    BitParNekLogAbstractions h, BitDCRGraph
    g) {
    // Possible additional conditions:
    // predecessors - current conditions
    HashMap<Integer, BitSet>
        possibleConditions = new HashMap<>();
    ;
    for (final Entry<Integer, BitSet> kvp :
        h.predecessor.entrySet()) {
        BitSet pc = (BitSet) kvp.getValue().
            clone();
        pc.andNot(g.conditionsFor.get(kvp.
            getKey()));
        possibleConditions.put(kvp.getKey(),
            pc);
    }
    // Go through the log once.
    for (final Entry<List<Integer>, Integer>
        kvp : h.traces.entrySet()) {
        List<Integer> trace = kvp.getKey();
        BitDCRMarking m = g.
            defaultInitialMarking();
```



```

// for each trace we track which
// activities have been seen at
// least once before.
BitSet seen = new BitSet();
for (int event : trace) {
    // possible activities that may
    // be a condition for the
    // current activity are those
    // that are not included, or
    // have been seen before.
    BitSet ok = new BitSet();
    ok.set(0, h.ActivityToID.size());
    ok.andNot(m.included);
    ok.or(seen);
    // valid additional conditions
    // are those for which this
    // applies in each instance.

    possibleConditions.get(event).and(
        ok);
    // execute the current activity in
    // the current DCR graph to get
    // a new marking.
    m = g.execute(m, event);
    // add the current activity to
    // those we have seen.
    seen.set(event);
}
}
}

```

Listing 8 [85]: BitParNek, ln. 255-296

Altogether, these optimizations provide us with an extremely efficient implementation of the ParNek algorithm. In the following section we will show through experimentation that it is in fact nearly one order of magnitude faster than any other miner and two orders of magnitude faster than most of the state-of-the-art Declare miners.

6 Evaluation

To evaluate the performance of our algorithm, we frame the process discovery task as a binary classification task of identifying legal/illegal traces. For this, we take advantage of a labeled data set from the Process Discovery Contest 2019⁵, in which DisCoveR was among the top performing submissions, classifying 96.1% of traces correctly. This result was achieved despite the fact that DisCoveR considers only control-flow, ignoring auxiliary data associated with events. Nevertheless, the present evaluation should not be interpreted as a comprehensive benchmarking, but rather a preliminary, proof-of-concept evaluation.

For comparison, we report results for: 1) a miner based on the same formalism (DCR Graphs) developed by Debois, et.al. [29]; 2) two leading miners also based on the declarative paradigm: MINERful [19] and Declare Miner [51]; 3) the

well-established Petri net miner, Inductive Miner; and finally 4) the winning miner for the PDC 2019, the Log Skeleton miner [87]. Note that the reason DisCoveR achieves a higher accuracy than the Log Skeleton miner in our evaluation is due to the fact that we report the results of the algorithm's classification alone, whereas the winning submission to the process discovery contest was a manually augmented model based on the output of the Log Skeleton miner.

Framing process discovery as a binary classification task is arguably an oversimplification of the aim of process discovery, since it does not capture the *degree* to which a model fails to capture an event log. Error measures that aim to capture this are usually based on model-log *alignment* techniques [9], or model specific measures such as *token replay* metrics for Petri nets [72]. The advantage of classification-based evaluation lies in the ease of interpretability and comparability. In a model-agnostic manner, we gain a view of the algorithm's bias towards committing different classes of statistical errors (e.g. Type I/II) by analyzing *true/false positives/negatives*, and the corresponding *precision, recall, F₁-score* and *MCC* measures.

Before presenting the results, we briefly formalize the task of process discovery as binary classification in terms of computational learning theory. This clarifies our formulation of processes in probabilistic terms, a property which is implied by the statistical evaluation metrics we present, a subset of which were the basis for evaluation in the PDC 2019.

Through the appeal to learning theory we aim to illustrate that a key reason our algorithm performs well is due to the - albeit heuristic - regularization (i.e. restriction on model complexity) performed at several steps in the algorithm.

6.1 The Learning Task

The goal of a supervised learning task is to learn an approximation h of a target function f which is assumed to generate the observed data [8]. The training data L is an i.i.d.⁶ sample from the true probability distribution (\mathbb{P}_P) associated with f . The aim is to maximize performance (e.g. minimize an error function) on *out-of-sample* data by means of optimizing performance on *in-sample* training data in such a way that the learned model avoids overfitting.

Formally, a learning algorithm γ is a mapping from a sampling L from the process (P, \mathbb{P}_P) to a hypothesis space \mathcal{H} s.t. the *out-of-sample* error E_{out} is minimized:

$$\gamma : \mathcal{L} \rightarrow \mathcal{H}; L \mapsto \arg \min_{h \in \mathcal{H}} E_{out}(h)$$

To define our error function E , we can frame process discovery-based binary classification as the task of predicting the outcome of a random Bernoulli variable defined by

⁵ <https://icpmconference.org/2019/process-discovery-contest>

⁶ Independent and identically distributed

$$\mathbb{1}(\sigma \in P)$$

which returns 1 when a trace σ is a member of P (the set of traces associated with the true process) and 0 otherwise

The most straightforward way of defining the *in-sample* error measure, is simply the proportion of “successes” in this Bernoulli trial. If L contains only positive examples (i.e. $L \subseteq P$), the in-sample error can be formulated as the proportion of traces accepted by the learned model h (i.e. *recall*):

$$E_{in}^r(h) = \sum_{\sigma \in L} \frac{\mathbb{1}(\sigma \in \ell(h))}{|L|}$$

If L contains both positive and negative examples, the in-sample error can be written as the proportion of examples on which the learned model and example agree (i.e. *accuracy*):

$$E_{in}^a(h) = \sum_{\sigma \in L} \frac{\mathbb{1}(\sigma \in \ell(h) \iff \sigma \in P)}{|L|}$$

We include this formulation (E_{in}^a) for clarity, but note that it is at odds with our formalization of a log L as a *sample* from P . For it to be consistent, we would need to consider L a sample of traces in P as well as not in P . In the evaluation based on PDC 2019 data, all training logs contain only positive examples.

In most learning tasks, minimizing $E_{in}(h)$ is trivial if the hypothesis set \mathcal{H} is large enough. Indeed, E_{in}^r can be trivially minimized by a flower process model which permits all behavior. The true challenge of the learning task lies in ensuring not only that in-sample error is small, but simultaneously that in-sample error is close to out-of-sample error.

Formally,

$$|E(h)_{in} - E(h)_{out}| < \epsilon$$

for some tolerance threshold ϵ .

While a large enough hypothesis space \mathcal{H} may indeed contain the target function f , the likelihood of our learning algorithm choosing f in such a large hypothesis space is vanishingly small. It is much more *likely* to settle on some other, very complex, function $g \in \mathcal{H}$, leading to a high E_{out} . We therefore seek an approach to ensuring that

$$\mathbb{P}[|E(h)_{in} - E(h)_{out}| < \epsilon] > 1 - \delta$$

where δ is a desired confidence threshold. This is known as a “probably (δ), approximately (ϵ), correct” (PAC) bound.

While somewhat counter-intuitive, this formulation helps us understand why restricting \mathcal{H} to a smaller set which *does not include the target function* f will often lead to a lower E_{out} .

Regularization Thus, a key component in the learning process is that of *regularization*: a process for controlling the complexity of a learned model, i.e. restricting the size of the hypothesis space, to improve generalization. This gives rise to the formulation of the learning process as a trade-off between inductive *bias*⁷ of a hypothesis set and a penalty for the *complexity* of a hypothesis [78]. The sum of these terms gives an *estimate* of the out-of-sample error:

$$\hat{E}_{out} = E_{in} + \Omega(N, \mathcal{H}, \delta).$$

Where N denotes sample size, \mathcal{H} the hypothesis space and δ the desired confidence that $E_{out} \leq \hat{E}_{out}$.

So although we can achieve a very low in-sample error using a rich hypothesis set, we penalize complex models using a *regularization* function Ω . Explicitly incorporating this function into learning algorithms s.t. it minimizes \hat{E}_{out} rather than E_{in} , can greatly improve results.

ParNek does not currently attempt to explicitly minimize \hat{E}_{out} , and Ω is likewise not explicitly formulated. However, some form of regularization is achieved by effectively restricting the size of \mathcal{H} . This is done via a set of heuristics attempting to control model complexity, removing those which are redundant w.r.t. training data or add little to the precision of its semantics. Indeed, ParNek cannot discover the entire set of DCR Graphs, thus

$$\mathcal{H}_{ParNek} \subset \mathcal{H}_{DCR} = \omega\text{-regular languages}$$

Restricting the available hypothesis set is analogous to limiting a linear regression algorithm to third-order polynomials, for example, which corresponds to an Ω which assigns a zero weight to all higher-order coefficients.

While heuristic in nature, the approach is effective, as is seen in comparison to miners which do little to control model complexity, such as Debois, et al’s miner. We intend to pursue more well-defined regularization procedures for DCR Graph mining algorithms in future work.

Metrics Aggregate evaluation metrics, such as *precision*, *recall* and F_1 -score are commonly reported for classification tasks. Given a confusion matrix, we define precision(prec.) and recall as follows:

PRED- ICTION	DATA		
	+	−	
+	True Pos.(TP)	False Pos.(FP)	prec. $\equiv \frac{TP}{TP+FP}$
−	False Neg.(FN)	True Neg.(TN)	
	recall $\equiv \frac{TP}{TP+FN}$		acc. $\equiv \frac{TP+TN}{TP+TN+FP+FN}$

⁷ The minimal in-sample error achievable for hypothesis $h \in \mathcal{H}$.

The F_β -score is then the harmonic mean of precision and recall, where β determines a weighting of precision relative to recall:

$$F_\beta = \frac{(1 + \beta^2) \cdot \text{precision} \cdot \text{recall}}{\beta \cdot \text{precision} + \text{recall}}$$

Originally stemming from information retrieval, these metrics have been criticized for giving weight to true positives and ignoring true negatives [15], and other metrics such as Matthews Correlation Coefficient (MCC) avoid assumptions regarding the target class.

Arguably, process mining *can* be seen as an information retrieval task, if the tool is used to “query” an event log for compliant/noncompliant traces. For completeness, we report precision, recall and F_1 -score for both the situation in which the target class is compliant behavior (true positive) and non-compliance (true negative), as well as Matthews Correlation Coefficient (MCC).

6.2 Results

In addition to case studies, we present a controlled evaluation of the algorithm based on a labeled data set from the Process Discovery Contest 2019. The evaluation is bolstered by the truly blind nature of the process. After being presented with a training set with positive examples only, and submitting results for a partially blind validation round, the predictions on a separate test set were sent in to the contest administrators who independently evaluated their accuracy. This removes any potential for accidental data snooping.

See Table 5 for the complete results.

Dataset The data set essentially consists of 10 independent data sets stemming from 10 different processes. Participants were presented with an unlabeled training set from each process. Then, two validation sets were provided for which participants could submit their algorithm’s classification results. The organizers then returned a confusion matrix - but no details regarding which traces specifically were misclassified and how. Two rounds of submission for validation were permitted, though we only took advantage of the first.

Event logs for processes 1, 5, 7, 8, 9, and 10 contained auxiliary data associated with each event, sometimes more than one attribute. The version of our algorithm presented here considers only control-flow and is unable to take advantage of additional attributes, and neither do the miners we present in the following comparison.

Comparison For comparison, we present the performance of five relevant mining algorithms: the first, another DCR Graph mining algorithm designed by Debois, et al [29]; second, two miners based on Declare constraints, MINERful [19] and

Declare Miner [51]; third, Inductive Miner, a flagship imperative miner which returns Petri net models; and finally Log Skeleton Miner, the winning submission to PDC 2019 [87].

Debois, et al’s DCR Graph miner takes a very greedy approach to identifying DCR relations which hold for an event log. Essentially, the algorithm begins with a fully constrained model over the set of activities in the log (mapped one-to-one to DCR events), then goes through the log and removes any constraints which are violated by observed behavior.

Due to the greedy strategy, the algorithm often finds thousands of constraints and clearly overfits the training data, leading to poor performance on test data.

MINERful is a miner for the Declare language which uses a number of user-defined parameters to determine which constraints to include in a model after mining the event log. The three core parameters are:

Support	The fraction of traces in which the constraints must hold.
Confidence	Support scaled by the fraction of traces in which a constraint is activated.
Interest Factor	Confidence scaled by the fraction of traces in which target of a constraint is also present.

A constraint is considered to be *activated* when it becomes relevant in a trace. So, a succession constraint between s and t will only become activated in traces in which s is present. In addition, to count towards interest factor, the target t must also be present. Defined as scalings, these parameters are dependent on one another and result in the bounds: support > confidence > interest factor.

MINERful also performs subsumption checks to eliminate redundant or meaningless constraints. For example, wherever a CHAINSUCCESSION constraint is found to hold, SUCCESSION will necessarily hold and adds no information. This procedure is akin to DisCover’s strategy of removing transitively redundant constraints in order to avoid unnecessarily complex models.

We employed an automated parametrization procedure originally developed for the evaluation in [12]. The procedure employs a binary search strategy to find values for confidence and threshold which result in a model with a number of constraints as close to, but not exceeding, some limit. We present results for models with between 89 and 200 constraints. Allowing larger models did not improve accuracy further.

Declare Miner was the first miner developed for the Declare language and uses a frequent itemset mining approach using the Apriori algorithm combined with subsequent pruning techniques. The user can set two threshold parameters: *support*, which measures the fraction of traces in which the constraints hold and *alpha* which measures the how often a constraint is activated (same as *confidence* for Minerful).

Furthermore the user can specify which constraint templates should be considered.

We consider models generated by Declare Miner with thresholds $support = 100$ and $alpha = 100$ and with either all constraint templates or only positive constraint templates (no *Not*-constraints). The parameter settings were settled upon after testing numerous settings from the range of thresholds, with 100/100 performing best.

Inductive Miner uses a divide-and-conquer approach to recursively partition the directly-follows graph (eventually-follows in the IMi variant of the miner) of a log such that the partitions correspond to one of four *process tree* operators: exclusive choice, sequential composition, parallel composition and redo loop. The resulting process tree can be transformed into a corresponding Petri net.

We tested Inductive Miner (IMf) using a range of noise thresholds from 0.0 to 1.0, where a setting of 0.0 ensures perfectly fitting models w.r.t. to the mined event log (training set). A noise threshold of 0.0 is equivalent to the original Inductive Miner (IM). We also investigated the variants known as IM-EKS, IMc, IMcpt, IMlc and IMflc, whose performance was nearly identical to standard IM (noise threshold 0.0). The largest difference was IMflc with 2 fewer correct classifications. We only report detailed results for settings 0.0, 0.5, 1.0 for readability, but note that intermediate noise threshold between these values followed the same, roughly linear, relationship with the accuracy of the resulting model.

Log Skeleton miner was the basis for the winning submission to the PDC 2019 and builds on some basic Declare constraint templates: PRECEDENCE, RESPONSE, NOTCOEXISTENCE, and adds NOTPRECEDENCE, and NOTRESPONSE. Furthermore, it employs the notion of equivalence classes for co-occurring activities. We report the results for the fully automated miner, but as noted, the final submission was manually extended, which is why the results we report are lower than the 99.78% accuracy achieved by the creator of Log Skeleton.

Results We report results for the classification task in a confusion matrix for each of the 10 processes, as well as aggregate across processes in Table 5. Keep in mind, that a user-defined error measure may choose to weigh false positives and false negatives differently (α and β in our formalization).

Additionally, we report Matthews Correlation Coefficient (MCC) in addition to precision, recall, and F_1 -score, both in the case of the target class being permissible traces, as well as forbidden traces. The appropriate framing would depend on the application.

6.2.1 Run-time

We compared runtime performance to the same miners as in our classification evaluation, finding that DisCoveR performs comparably with the fastest miners, and much faster

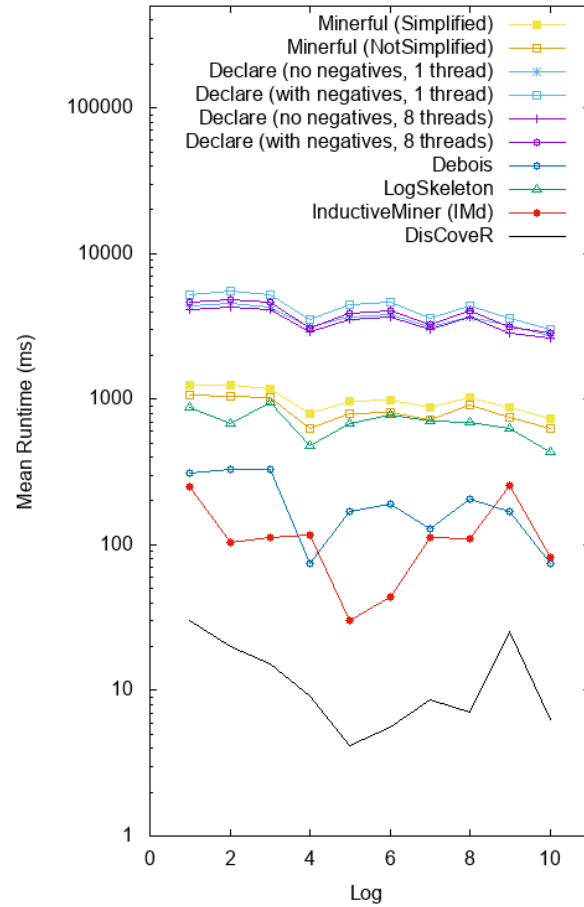


Fig. 2 Mean run-times in milliseconds across 100 runs on Process Discovery Contest 2019 training logs. MINERful was run with the thresholds: $support = 1.0$, $confidence = 1.0$, $interest\ factor = 1.0$, with and without a post-processing step to simplify models. Declare Miner was with and without multithreading, and with $alpha = 1.0$, $support = 1.0$, with all constraint templates and with all but the negative constraint templates. For runtimes, the IMd variant of Inductive Miner from the py4pm platform for Python, as this variant is significantly faster than other IM variants. Log Skeleton Miner was the winning submission in terms of classification accuracy, DisCoveR was the runner-up.

than Declare-based miners, MINERful and Declare miner, even when multithreading is enabled. Note that for runtime comparison, the linear-time IMd variant of Inductive Miner from the pm4py⁸ Python module was used.

Experimental setup Experiments were conducted on the set of 10 test logs from the Process Discovery Contest 2019, and were run on a Lenovo Thinkpad P50 with an Intel Xeon E3-1535M v5 2.90 GHz quad-core processor and 32G of RAM. We present mean run-times over 100 runs of mining each log.

⁸ <http://pm4py.org>

MINERful was parametrized with support threshold of 1.0, a confidence threshold of 1.0 and interest factor threshold of 1.0. Declare miner was parametrized with support = 1.0 and alpha = 1.0. The parameters for MINERful were chosen due to being the most “generous” in terms of runtime. The parameters for Declare miner stem from the best performance in classification. We also report notable variants: for MINERful with and without an additional model simplification step, for Declare miner with/without negative constraints and with/without multithreading. We note that changes in parametrizations do not significantly alter performance - certainly not relative to other miners. Note that we did *not* employ the parameter tuning procedure used to achieve the results for MINERful in Table 5 which requires re-running the miner many times.

The 10 logs all consist of 700 traces. Run-time results can be seen in Figure 2 as well as Table 4, where details regarding number of activities and mean trace length are also included.

Note that these results should be taken as a rough indication of performance subject to some variance. A number of factors that are out of our control may affect runtimes, especially for very low runtimes. These include Java Virtual Machine’s garbage collection strategies, just-in-time compilation and optimization strategies, as well as background operating system processes. To determine a reasonable number of runs, we observed the convergence of runtime estimates w.r.t. increases in runs, finding that estimates stabilized by 100 runs and clearly so by 1000 and 10000 runs. We present results for 100 runs in part because higher numbers of runs for some miners was not feasible.

6.2.2 Mined Model

Finally, we show an example of what a mined DCR Graph actually looks like. In the listing below we show the mined model for log 10 of the PDC 2019 data set. DCR Graphs can be represented either graphically (as nodes representing activities and edges representing relations) or as a language [30]. Here we opted for the language format as it allows for a more concise representation of large models. The relations are written as `-->*`, `*-->`, `-->%` and `-->+`, respectively, the condition, response, exclusion and inclusion. By `d -->*` `ae` we denote that `d` is a condition for `ae`. Activities can be grouped together as a shorthand for denoting multiple relations, e.g. `i -->*` `(p, ai)` denotes a condition from `i` to `p` and an additional condition from `i` to `ai`.

```
d -->* ae
i -->* (p, ai)
t -->* q
q -->* w
w -->* (p, ab, ak, e)
p -->* ai
b -->* (ab, ak, e)
k -->* ar
x -->* ap
```

```
j -->* (g, m, v, o, l, r, ab, ak, e)
ap -->* j
ad -->* (j, u, aa, ao)
ao -->* (v, l, r)
ae -->* (l, r, ab, ak, e)
ab -->* aq
ak -->* ai
aq -->* ai
e -->* ai
k *--> ar
u *--> j
aa *--> i
ab *--> (ae, aq, ai)
ak *--> (ae, ai)
aq *--> (l, r, ao)
e *--> (ae, aq, ai)
ai *--> (l, r, ao)
d -->% d
t -->% t
q -->% q
w -->% w
p -->% p
b -->% (b, h)
a -->% (d, a)
k -->% (d, k, ap, ad, o, l, r, ao, ae, ab, ak,
      aq, e, ai)
x -->% x
ar -->% ar
j -->% j
g -->% g
h -->% (i, b, h)
ap -->% ap
u -->% (g, ad, u, l, r, aa, ao, ab, ak, aq, e,
      ai)
m -->% m
v -->% (v, ae)
o -->% (d, g, o, ao, ae)
l -->% l
r -->% r
aa -->% (ad, u, v, l, r, aa, ao, ab, ak, aq, e,
      ai)
c -->% (i, t, q, w, p, b, h, c)
ao -->% ao
ab -->% ab
ak -->% ak
e -->% e
ai -->% ai
```

Listing 9 Mined DCR Graph for log 10 of PDC2019

7 Case Study: Interactive Model Recommendation

In this section we discuss how DisCoveR has been integrated in the `dcrgraphs.net` process portal as a means to provide modeling recommendations for the interactive modeling of declarative knowledge-intensive processes. We start by briefly describing the portal and its main functionalities. We then show how process discovery has been integrated in the portal and end with a discussion on how the model recommendation functionality is used in practice.

Log	RUN-TIME (ms)										LOG ATTRIBUTES	
	Minerful Simplified	Minerful NotSimpl.	Declare Negatives	Declare NoNeg.	8-thr. Declare Negatives	8-thr. Declare NoNeg.	Debois	Log Skeleton Miner	Inductive Miner (IMd)	DisCoveR	Number of Activities	Mean Trace Length
1	1243.3	1071.8	5197.2	4395.8	4652.0	4116.6	312.4	872.9	250.3	30.1	45	17.21
2	1236.1	1048.6	5494.5	4564.2	4854.3	4250.3	329.0	686.3	104.0	20.0	46	18.99
3	1183.7	1033.9	5253.9	4310.4	4638.3	4094.2	329.2	944.6	112.6	15.2	48	12.0
4	791.2	633.1	3556.0	3146.9	3098.3	2875.7	74.9	482.4	117.7	9.2	34	10.09
5	969.5	797.7	4449.5	3676.4	3891.0	3520.7	167.9	680.5	30.3	4.2	44	5.33
6	979.9	810.6	4632.3	3829.3	4006.6	3649.2	191.4	773.8	43.8	5.6	43	8.62
7	869.6	723.5	3574.3	3118.1	3285.7	3010.8	129.2	706.0	112.2	8.6	35	12.58
8	1030.2	920.2	4338.5	3672.3	4040.3	3657.5	205.6	692.5	110.9	7.1	44	9.04
9	880.5	747.9	3594.0	3188.3	3123.0	2862.4	170.5	623.7	256.1	25.3	29	26.41
10	742.4	625.1	3033.5	2721.1	2864.7	2633.1	74.8	432.3	82.8	6.3	32	9.33

Table 4 Mean run-times in milliseconds across 100 runs on Process Discovery Contest 2019 training logs, along with log statistics. See Figure 2 for descriptions of miner parametrizations. Log Skeleton Miner was the winning submission to the contest in terms of classification accuracy, DisCoveR was the runner-up.

7.1 The DCR Process Portal

The `dcrgraphs.net` process portal is a cloud-based commercial modeling solution for declarative process models, offering an extensive range of functions including process modeling, simulation, analysis, maintenance, and a wide variety of collaboration features. The portal has been created and is maintained by DCR Solutions, in close collaboration with researchers from the University of Copenhagen, IT University of Copenhagen and Danish Technical University. The DCR notation, portal and DCR process engine have been applied in a range of application domains. Most notably the engine was integrated into Workzone, a case management product used by over 70% of Danish central government institutions⁹ and the portal has become a cornerstone of the Ecomknow research project¹⁰, which proposes a novel digitalization strategy for Danish municipalities grounded in the declarative modeling of knowledge-intensive citizen processes.

The key component of the portal is the DCR modeling tool, shown in Figure 3, which allows users to model and simulate DCR graphs. At the center of the screen is the modeling pane with the graphical representation of the DCR Graph, where activities are drawn as boxes and relations as colored arrows in a style similar to the formal syntax. Users can add and manipulate activities and relations between them directly in the modeling pane and change their details in an option panel on the right. The simulation screen is shown in Figure 4. The upper right of the screen shows the current task list, here the user can select which task to execute next. The middle of the screen shows recommendations for next steps and a simulation log. On the left we have a number of

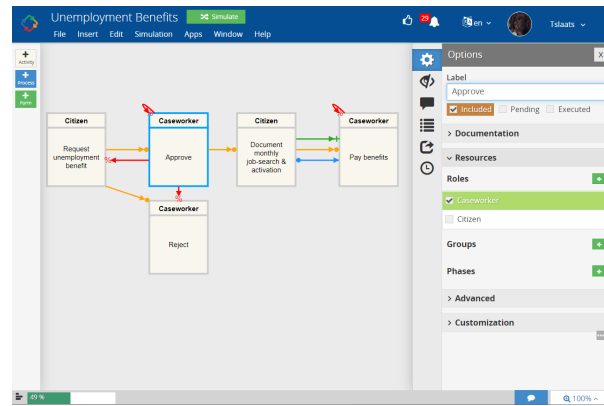


Fig. 3 DCR Graphs Modeling

advanced features, such as making time steps and a list of all users involved in the simulation (collaborative simulations are supported). In the bottom of the screen the user can see a step-by-step flowchart representation of the current simulation, divided into swimlanes.

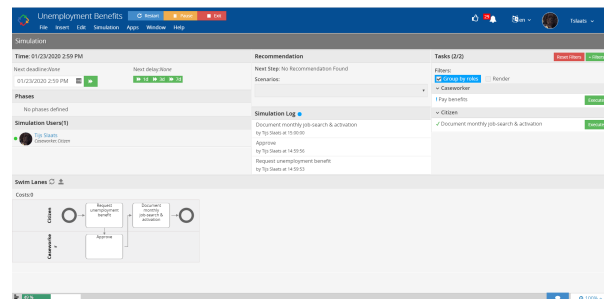


Fig. 4 DCR Graphs Simulation

⁹ <http://www.kmd.dk/indsigter/fleksibilitet-og-dynamisk-sagsbehandling-i-staten>

¹⁰ <https://ecoknow.org/>

		OBSERVED										Aggregate		TARGET TRACES Posi- Nega- tive tive													
		P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}																
DisCoveR		+	-	+	-	+	-	+	-	+	-	+	-	+	-	MCC	0.92										
PRED-ICTED	+	45	0	45	0	45	0	47	6	45	3	45	0	44	8	448	30	Prec.	0.94	0.99							
	-	0	45	0	45	0	45	1	36	0	42	0	45	1	37						2	43	0	42	1	37	5
Model size		142		189		271		182		447		412		143		284		171		136		Acc.:	96.1%		F_1	0.96	
Log Skeleton		+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	MCC	0.85								
PRED-ICTED	+	36	1	42	0	38	2	47	1	45	3	45	0	39	8	44	1	413	27	Prec.	0.94	0.91					
	-	9	44	3	45	7	43	1	41	0	42	0	45	6	37	7	40						1	44	6	39	40
Model size		235		240		250		180		230		225		185		230		155		170		Acc.:	92.6%		F_1	0.92	
MINERful ²		+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	MCC	0.85								
PRED-ICTED	+	43	2	45	1	44	4	48	1	45	19	45	4	45	8	45	7	447	67	Prec.	0.87	0.98					
	-	2	43	0	44	1	41	0	41	0	26	0	41	0	37	0	38						0	31	3	38	6
Model size		182		188		186		194		198		199		199		189		174		183		Acc.:	91.9%		F_1	0.92	
MINERful ¹		+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	MCC	0.79								
PRED-ICTED	+	45	7	45	2	45	6	48	5	45	21	45	3	45	13	45	15	452	96	Prec.	0.82	0.98					
	-	0	38	0	43	0	39	0	37	0	24	0	42	0	32	0	30						0	28	1	38	1
Model size		99		99		92		96		89		99		99		94		94		97		Acc.:	89.9%		F_1	0.90	
DeclarePos		+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	MCC	0.50								
PRED-ICTED	+	45	25	45	11	45	19	48	24	45	35	45	37	44	33	45	38	451	267	Prec.	0.63	0.99					
	-	0	20	0	34	0	26	0	18	0	10	0	8	1	12	0	7						0	37	1	8	2
Model size		464		759		269		623		338		778		259		242		885		379		Acc.:	70.1%		F_1	0.77	
Inductive ^{0.0}		+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	MCC	0.49								
PRED-ICTED	+	45	35	45	0	45	36	48	34	45	7	45	2	45	44	45	43	44	29	45	45	452	275	Prec.	0.62	0.99	
	-	0	10	0	45	0	9	0	8	0	38	0	43	0	1	0	2										1
Model size		110		174		122		170		156		154		92		112		102		86		Acc.:	69.3%		F_1	0.77	0.55
DeclareAll		+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	MCC	0.45								
PRED-ICTED	+	45	25	42	8	43	21	48	25	45	35	45	39	44	33	45	38	44	7	43	36	439	267	Prec.	0.62	0.93	
	-	0	20	3	37	2	24	0	17	0	10	0	6	1	12	0	7										1
Model size		1870		3542		3244		2395		4469		4248		1369		2687		1507		1720		Acc.:	68.8%		F_1	0.76	0.56
Inductive ^{0.5}		+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	MCC	0.26								
PRED-ICTED	+	45	35	0	0	17	8	0	0	31	2	43	0	35	31	38	33	0	0	0	1	209	110	Prec.	0.66	0.58	
	-	0	10	45	45	28	37	48	42	14	43	2	45	10	14	7	12										45
Model size		110		172		154		152		120		116		92		132		80		148		Acc.:	60.6%		F_1	0.54	0.66
Inductive ^{1.0}		+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	MCC	0.26								
PRED-ICTED	+	0	0	0	0	0	0	31	2	30	0	0	0	0	0	0	0	0	0	0	0	61	2	Prec.	0.97	0.53	
	-	45	45	45	45	45	45	48	42	14	43	15	45	45	45	45	45										45
Model size		110		150		110		100		120		108		84		118		100		88		Acc.:	56.2%		F_1	0.24	0.69
Debois, et al		+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	MCC	0.03								
PRED-ICTED	+	0	0	0	0	1	0	0	0	12	7	0	0	6	4	1	1	1	0	6	8	27	20	Prec.	0.57	0.50	
	-	45	45	45	45	45	44	42	48	38	33	45	45	41	39	44	44										45
Model size		1821		2293		2376		641		1557		1515		1268		1716		984		775		Acc.:	50.4%		F_1	0.11	0.66

Table 5 Confusion matrices for individual data sets, each generated by separate ground truth model, in our formulation referred to as (P_i, \mathbb{P}_{P_i}) . Precision(Prec.), Recall and F_1 -scores are reported for which the target class is legal and illegal traces, respectively. Matthews Correlation Coefficient (MCC) is also reported. MINERfulⁿ refers to a parametrization which results in a model with fewer than $n \cdot 100$ constraints. DeclarePos refers to the Declare Miner excluding negative constraints, Declare includes all constraint templates, both with parameterisations: alpha=100, support=100. Inductiveⁿ refers to Inductive Miner with a noise threshold of n . Model size refers to “edges” in all models, i.e. binary relations in declarative models (including Log Skeleton) and edges between places and transitions in Petri nets from the Inductive Miner.

7.2 Interactive process modelling through model recommendation

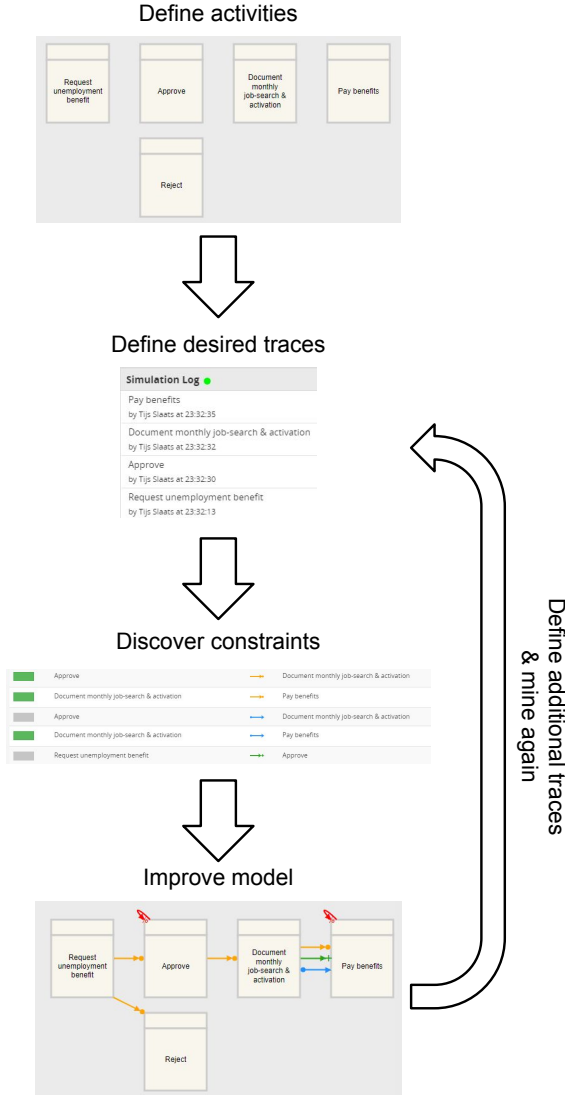


Fig. 5 Overview of the model recommendation approach

In the declarative modeling approach advocated by DCR Solutions modelers are encouraged to 1) identify the activities and roles of the process, 2) think about what common and uncommon scenarios (i.e. traces) should be supported by the process, 3) based on the scenarios determine what reasonable constraints for the process would be, and 4) ensure that the constraints do not conflict with any desired paths through the use of simulation and test-cases [81]. The identification of constraints in step 3 has been identified as the

most challenging for users because it requires a firm grasp of the semantics of DCR Graphs. While test cases and simulation can be used to retroactively check that no conflicting constraints have been introduced, they are not helpful for identifying suitable constraints directly. As a result, novice users often use a fairly inefficient trial-and-error approach where they try a constraint, check how it behaves under simulation and then update their model accordingly.

We introduced process discovery as an alternative to this trial-and-error approach. In this new setting, the portal supports the user by having an algorithm automatically propose suitable relations based either on an existing event log, and/or the traces that were identified during step 2 of the previously sketched modeling method.

Figure 5 provides an overview of the adapted approach: we start by identifying the activities of the process and modeling these directly in the portal. In the next step we run simulations on these activities (recall that following the declarative paradigm, these simulations are entirely unconstrained and any trace can be generated). We store the traces generated during the simulation and use these as input for the following step, where we use DisCoveR to identify constraints based on the generated traces. Finally the user can improve on their model and potentially run more simulations which can be used for additional process discovery, possibly finding additional constraints that were not found for the initial traces.

Source	Type	Target	Explanation
Request unemployment benefit	→	Approve	
Request unemployment benefit	→	Reject	
Approve	→	Document monthly job-search & activation	
Document monthly job-search & activation	→	Pay benefits	
Approve	→	Document monthly job-search & activation	
Document monthly job-search & activation	→	Pay benefits	
Request unemployment benefit	→	Approve	
Request unemployment benefit	→	Reject	
Document monthly job-search & activation	→	Pay benefits	
Approve	→	Pay benefits	
Pay benefits	→	Pay benefits	
Reject	→	Reject	

Relations found: 12 New relations: 12

Add Relations

Fig. 6 Model Recommendation

The model recommendation screen is shown in Figure 6 and fairly straightforward: the user is shown which relations were found between which activities and can select those they wish to add through the box on the left. The user can also enter an explanation for the relation (i.e. why was it added or left out), this enables rationale management of the model and allows other users to follow the modeler's reasoning. In addition, we plan to use this information in the future to improve upon the discovery algorithm. By clicking *Add Relations*, all selected relations are added to the model.

7.3 Discussion

Since the integration of DisCoveR into the DCR Graphs portal, DCR Solutions has been actively conducting workshops with users where the new methodology is demonstrated and used. The inclusion of process mining in the modeling task was embraced enthusiastically by users and has been (informally) observed to lower the complexity of the modeling task.

In the traditional modeling exercise, users that are more familiar with BPMN and/or flow charts are often hampered by the novelty of the notation, e.g. they will be unclear on what the different relations mean and how to use them. In particular, the fact that arrows do not indicate flow, but logical relations between the activities can lead to confusion. Using model recommendations, on the other hand, has allowed DCR Solutions to ask the users questions based on the recommended relations such as, “Is it true that approval is a condition for providing documentation?” or, “Is it true approval removes the ability to reject?”.

In essence, model recommendation has managed to bridge an important gap between the consultant and user: in the past, the users were new to the notation, the consultants to the process. This made building a common understanding about the process a time intensive task. Model recommendation closes this divide by, on the one hand, helping the consultant better understand the process and, on the other, providing the user with examples of the notation that are uniquely fitting to their own domain.

The high accuracy of the algorithm has also been noted in practice: even for processes that include other perspectives than just control-flow (e.g. decisions depending on contextual data), the algorithm has been noted to be highly successful in recommending relevant relations that improved the users’ understanding of the process.

The integration of the algorithm in the commercial tools was relatively effortless: the front-end of the model recommendation was developed rapidly at DCR Solutions through existing plugin support for the portal. The algorithm itself was simply deployed as a cloud service by the researchers. Because of a long history of close collaboration between the two parties, the details of the interface between these two components and a general understanding of how the system should work was fleshed out quickly over two meetings and a few emails.

It should be noted that two variations of DisCoveR exist: the regular version used in the Process Discovery Contest prioritizes accuracy, whereas there also exists a light version that skips the step of finding additional inclusions and exclusions, thereby returning a less accurate but simpler model. It is this light version that is used within the DCR Graphs portal.

8 Conclusion

In this paper we presented DisCoveR, a declarative miner for DCR Graphs based on the ParNek algorithm. We formally defined the underlying algorithm and how it has been implemented using an acute mapping to bit vector operations, yielding a highly efficient process discovery tool. We then preface the evaluation by framing process-discovery-as-classification in terms of computational learning theory in order to gain insight into the convincing performance of the algorithm on out-of-sample data. We evaluated the miner using a traditional classification task and computed the standard machine learning measures of accuracy (96.1%), precision (0.94 on positive traces, 0.99 on negative traces), recall (0.99 on positive traces, 0.93 on negative traces), F1 (0.96 on each) and MCC (0.92).

The present evaluation suggests that DisCoveR is competitive with its peers, but should not be seen as a comprehensive benchmarking: this would require a more extensive evaluation on a larger variety of data sets, and against a more representative collection of miners. Where DisCoveR does appear to excel - in particular in comparison to other declarative miners - is in terms of run-time, performing one order of magnitude faster than the state-of-the-art in DCR Graphs discovery and nearly two orders of magnitude faster than the state-of-the-art in Declare discovery. Finally, we showed how the tool has been integrated in a commercial modeling tool and discuss how its integration has significantly improved the modeling experiences of its users.

8.1 Future Work

Several avenues exist for future work in mining DCR Graphs from event logs. So far, we have considered only the control flow of processes. Incorporating timing, data, and resource perspectives is very relevant for many real-world scenarios and one of the primary requests made by DCR Solutions. Furthermore, accounting for noisy data is an important point to address since this is common in real world applications.

We restricted our hypothesis space to graphs with the same simple initial marking in which all events are enabled. This is due to the complicated interactions arising with other relations when excluding a source event. Considering different initial markings would enable the discovery of more complex models, but also enlarge the hypothesis space and increase the danger of overfitting.

In order to control more explicitly for overfitting and quantify the tradeoff between inductive bias and complexity, a formulation of regularization functions for classes of DCR Graphs is an important next step. This is not entirely straightforward due to the non-monotonic nature of DCR Graphs [27], rendering simple relation counting more or less meaningless for regularization purposes.

As described in the case study, users of the *dcrgraphs*.net portal are not only able to define positive scenarios, but also undesired scenarios. The use of negative input data in process discovery has so far been mostly ignored based on the assumption that such data is not available. Having negative scenarios provided by the portal offers a unique opportunity to develop new algorithms that take negative examples as input and thereby produce more relevant models. We observe that DisCoveR has a noticeably lower recall on negative than positive traces and hypothesize that the ability to analyze negative examples of traces will help us improve on this aspect of the accuracy of the tool.

Finally, there remain certain points in the ParNek algorithm in which choices are currently taken in a naive manner (e.g. *ChooseOneRelation*). This decision point should be framed as a proper optimization problem. In fact, framing DCR Graph mining properly as an optimization task would open a powerful set of tools from the general optimization literature.

References

1. van der Aalst, W., Pesic, M., Schonenberg, H., Westergaard, M., Maggi, F.M.: Declare. Webpage (2010). <http://www.win.tue.nl/declare/>
2. Van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering* **16**(9), 1128–1142 (2004)
3. van der Aalst, W.M., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: M. Bravetti, M. Nunez, G. Zavattaro (eds.) *Proceedings of Web Services and Formal Methods (WS-FM 2006)*, LNCS, vol. 4184, pp. 1–23. Springer Verlag (2006)
4. van der Aalst, W.M.P., van Hee, K.M.: *Workflow Management: Models, Methods, and Systems*. MIT Press (2002)
5. Abbad Andaloussi, A., Buch-Lorentsen, J., López, H.A., Slaats, T., Weber, B.: Exploring the modeling of declarative processes using a hybrid approach. In: A.H.F. Laender, B. Pernici, E.P. Lim, J.P.M. de Oliveira (eds.) *Conceptual Modeling*, pp. 162–170. Springer International Publishing, Cham (2019)
6. Abbad Andaloussi, A., Burattin, A., Slaats, T., Petersen, A.C.M., Hildebrandt, T.T., Weber, B.: Exploring the understandability of a hybrid process design artifact based on dcr graphs. In: I. Reinhartz-Berger, J. Zdravkovic, J. Gulden, R. Schmidt (eds.) *Enterprise, Business-Process and Information Systems Modeling*, pp. 69–84. Springer International Publishing, Cham (2019)
7. Abbad Andaloussi, A., Slaats, T., Burattin, A., Hildebrandt, T.T., Weber, B.: Evaluating the understandability of hybrid process model representations using eye tracking: First insights. In: F. Daniel, Q.Z. Sheng, H. Motahari (eds.) *Business Process Management Workshops*, pp. 475–481. Springer International Publishing, Cham (2019)
8. Abu-Mostafa, Y.S., Magdon-Ismail, M., Lin, H.: *Learning from Data: A Short Course*. AMLBook.com (2012). URL <https://books.google.co.uk/books?id=iZUzMWECAAJ>
9. Adriansyah, A., Muñoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.: Alignment based precision checking. In: *International Conference on Business Process Management*, pp. 137–149. Springer (2012)
10. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94*, pp. 487–499. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1994). URL <http://dl.acm.org/citation.cfm?id=645920.672836>
11. Andaloussi, A.A., Burattin, A., Slaats, T., Kindler, E., Weber, B.: On the declarative paradigm in hybrid business process representations: A conceptual framework and a systematic literature study. *Inf. Syst.* **91**, 101505 (2020). DOI [10.1016/j.is.2020.101505](https://doi.org/10.1016/j.is.2020.101505). URL <https://doi.org/10.1016/j.is.2020.101505>
12. Back, C.O., Debois, S., Slaats, T.: Towards an empirical evaluation of imperative and declarative process mining. In: *International Conference on Conceptual Modeling*, pp. 191–198. Springer (2018)
13. Bhattacharya, K., Gereade, C., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: *In preparation*, pp. 288–304 (2007)
14. Burattin, A., Maggi, F.M., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. *Expert Systems with Applications* **65**, 194 – 211 (2016). DOI <https://doi.org/10.1016/j.eswa.2016.08.040>. URL <http://www.sciencedirect.com/science/article/pii/S0957417416304390>
15. Chicco, D., Jurman, G.: The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics* **21**(1), 6 (2020)
16. Ciccio, C.D., Maggi, F.M., Montali, M., Mendling, J.: Resolving inconsistencies and redundancies in declarative process models. *Information Systems* **64**, 425 – 446 (2017). DOI <https://doi.org/10.1016/j.is.2016.09.005>. URL <http://www.sciencedirect.com/science/article/pii/S0306437915302052>
17. Ciccio, C.D., Maggi, F.M., Montali, M., Mendling, J.: On the relevance of a business constraint to an event log. *Information Systems* **78**, 144 – 161 (2018). DOI <https://doi.org/10.1016/j.is.2018.01.011>. URL <http://www.sciencedirect.com/science/article/pii/S0306437916306457>
18. Ciccio, C.D., Mecella, M.: On the discovery of declarative control flows for artful processes. *ACM Trans. Manage. Inf. Syst.* **5**(4), 24:1–24:37 (2015). DOI [10.1145/2629447](https://doi.org/10.1145/2629447). URL <http://doi.acm.org/10.1145/2629447>
19. Ciccio, C.D., Mecella, M.: On the discovery of declarative control flows for artful processes. *ACM Transactions on Management Information Systems (TMIS)* **5**(4), 1–37 (2015)
20. Costa Seco, J., Debois, S., Hildebrandt, T., Slaats, T.: Reseda: Declaring live event-driven computations as reactive semi-structured data. In: *2018 IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC)*, pp. 75–84 (2018). DOI [10.1109/EDOC.2018.00020](https://doi.org/10.1109/EDOC.2018.00020)
21. De Giacomo, G., Dumas, M., Maggi, F.M., Montali, M.: Declarative process modeling in bpmn. In: J. Zdravkovic, M. Kirikova, P. Johannesson (eds.) *Advanced Information Systems Engineering*, pp. 84–100. Springer International Publishing, Cham (2015)
22. De Masellis, R., Maggi, F.M., Montali, M.: Monitoring data-aware business constraints with finite state automata. In: *Proceedings of the 2014 International Conference on Software and System Process, ICSSP 2014*, pp. 134–143. ACM, New York, NY, USA (2014). DOI [10.1145/2600821.2600835](https://doi.org/10.1145/2600821.2600835). URL <http://doi.acm.org/10.1145/2600821.2600835>
23. De Smedt, J., De Weerd, J., Vanthienen, J., Poels, G.: Mixed-paradigm process modeling with intertwined state spaces. *Business & Information Systems Engineering* **58**(1), 19–29 (2016). DOI [10.1007/s12599-015-0416-y](https://doi.org/10.1007/s12599-015-0416-y). URL <https://doi.org/10.1007/s12599-015-0416-y>
24. Debois, S., Hildebrandt, T.: The DCR Workbench: Declarative Choreographies for Collaborative Processes. In: S. Gay, A. Ravara (eds.) *Behavioural Types: from Theory to Tools*, River Publishers

- Series in Automation, Control and Robotics, pp. 99–124. River Publishers (2017). URL https://www.riverpublishers.com/pdf/ebook/chapter/RP_9788793519817C5.pdf
25. Debois, S., Hildebrandt, T., Marquard, M., Slaats, T.: Hybrid Process Technologies in the Financial Sector: The Case of BRFKredit, pp. 397–412. Springer International Publishing, Cham (2018)
 26. Debois, S., Hildebrandt, T., Slaats, T.: Hierarchical declarative modelling with refinement and sub-processes. In: S. Sadiq, P. Soffer, H. Völzer (eds.) *Business Process Management*, pp. 18–33. Springer International Publishing, Cham (2014)
 27. Debois, S., Hildebrandt, T., Slaats, T.: Safety, liveness and run-time refinement for modular process-aware information systems with dynamic sub processes. In: *International Symposium on Formal Methods*, pp. 143–160. Springer (2015)
 28. Debois, S., Hildebrandt, T., Slaats, T., Marquard, M.: A case for declarative process modelling: Agile development of a grant application system. In: *2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations*, pp. 126–133 (2014). DOI 10.1109/EDOCW.2014.27
 29. Debois, S., Hildebrandt, T.T., Laursen, P.H., Ulrik, K.R.: Declarative process mining for dcr graphs. In: *Proceedings of the Symposium on Applied Computing*, pp. 759–764 (2017)
 30. Debois, S., Hildebrandt, T.T., Slaats, T.: Replication, refinement & reachability: complexity in dynamic condition-response graphs. *Acta Informatica* **55**(6), 489–520 (2018). DOI 10.1007/s00236-017-0303-8. URL <https://doi.org/10.1007/s00236-017-0303-8>
 31. Di Ciccio, C., Maggi, F.M., Mendling, J.: Efficient discovery of target-branched declare constraints. *Inf. Syst.* **56**(C), 258–283 (2016). DOI 10.1016/j.is.2015.06.009. URL <https://doi.org/10.1016/j.is.2015.06.009>
 32. Di Ciccio, C., Marrella, A., Russo, A.: Knowledge-intensive processes: Characteristics, requirements and analysis of contemporary approaches. *Journal on Data Semantics* **4**(1), 29–57 (2015). DOI 10.1007/s13740-014-0038-4. URL <https://doi.org/10.1007/s13740-014-0038-4>
 33. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in bpmn. *Information and Software Technology* **50**(12), 1281–1294 (2008)
 34. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: *Fundamentals of Business Process Management*. Springer (2013). DOI 10.1007/978-3-642-33143-5. URL <http://dx.doi.org/10.1007/978-3-642-33143-5>
 35. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: *Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No. 99CB37002)*, pp. 411–420. IEEE (1999)
 36. Fu, J., Topcu, U.: Computational methods for stochastic control with metric interval temporal logic specifications. In: *2015 54th IEEE Conference on Decision and Control (CDC)*, pp. 7440–7447. IEEE (2015)
 37. Goedertier, S., Martens, D., Baesens, B., Haesen, R., Vanthienen, J.: Process mining as first-order classification learning on logs with negative events. In: *International Conference on Business Process Management*, pp. 42–53. Springer (2007)
 38. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events. *Journal of Machine Learning Research* **10**(Jun), 1305–1340 (2009)
 39. Herzberg, N., Kirchner, K., Weske, M.: Modeling and monitoring variability in hospital treatments: a scenario using cmmn. In: *International Conference on Business Process Management*, pp. 3–15. Springer (2014)
 40. Hildebrandt, T., Mukkamala, R.R., Slaats, T.: Designing a cross-organizational case management system using dynamic condition response graphs. In: *2011 IEEE 15th International Enterprise Distributed Object Computing Conference*, pp. 161–170 (2011). DOI 10.1109/EDOC.2011.35
 41. Hildebrandt, T., Mukkamala, R.R., Slaats, T.: Nested dynamic condition response graphs. In: *Proceedings of Fundamentals of Software Engineering (FSEN)* (2011). URL http://www.itu.dk/people/rao/pubs_accepted/fsenpaper.pdf
 42. Hildebrandt, T., Mukkamala, R.R., Slaats, T., Zanitti, F.: Contracts for cross-organizational workflows as timed dynamic condition response graphs. *Journal of Logic and Algebraic Programming (JLAP)* (2013). <http://dx.doi.org/10.1016/j.jlap.2013.05.005>
 43. Hildebrandt, T.T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs. In: *Proceedings Third Workshop on Programming Language Approaches to Concurrency and communication-cEntric Software, PLACES 2010, Paphos, Cyprus, 21st March 2010.*, pp. 59–73 (2010). DOI 10.4204/EPTCS.69.5. URL <http://dx.doi.org/10.4204/EPTCS.69.5>
 44. Hildebrandt, T.T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs. *arXiv preprint arXiv:1110.4161* (2011)
 45. Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath III, F.T., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculin, R.: Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In: *Proc. of WS-FM’10*, pp. 1–24. Springer-Verlag, Berlin, Heidelberg (2011)
 46. Kong, Z., Jones, A., Belta, C.: Temporal logics for learning and detection of anomalous behavior. *IEEE Transactions on Automatic Control* **62**(3), 1210–1222 (2016)
 47. Kurz, M., Schmidt, W., Fleischmann, A., Lederer, M.: Leveraging cmmn for acm: examining the applicability of a new omg standard for adaptive case management. In: *Proceedings of the 7th International Conference on Subject-Oriented Business Process Management*, p. 4. ACM (2015)
 48. La Rosa, M., Reijers, H.A., Van Der Aalst, W.M., Dijkman, R.M., Mendling, J., Dumas, M., García-Bañuelos, L.: *Apromore: An advanced process model repository*. *Expert Systems with Applications* **38**(6), 7029–7040 (2011)
 49. Madsen, M.F., Gaub, M., Høgnason, T., Kirkbro, M.E., Slaats, T., Debois, S.: Collaboration among adversaries: distributed workflow execution on a blockchain. In: *Symposium on Foundations and Applications of Blockchain*, p. 8 (2018)
 50. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient discovery of understandable declarative process models from event logs. In: *Advanced Information Systems Engineering*, pp. 270–285 (2012)
 51. Maggi, F.M., Ciccio, C.D., Francescomarino, C.D., Kala, T.: Parallel algorithms for the automated discovery of declarative process models. *Information Systems* **74**, 136 – 152 (2018). DOI <https://doi.org/10.1016/j.is.2017.12.002>. URL <http://www.sciencedirect.com/science/article/pii/S0306437916306615>. Special Issue on papers presented in the 20th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2016
 52. Maggi, F.M., Montali, M., Westergaard, M., van der Aalst, W.M.P.: Monitoring business constraints with linear temporal logic: An approach based on colored automata. In: *Business Process Management (BPM) 2011, Lecture Notes in Computer Science*, vol. 6896, pp. 32–147 (2011). DOI 10.1007/978-3-642-23059-13
 53. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-guided discovery of declarative process models. In: *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pp. 192–199 (2011). DOI 10.1109/CIDM.2011.5949297
 54. Maggi, F.M., Slaats, T., Reijers, H.A.: The automated discovery of hybrid processes. In: S. Sadiq, P. Soffer, H. Völzer (eds.) *Business Process Management*, pp. 392–399. Springer International Publishing, Cham (2014)
 55. Manataki, A., Fleuriot, J., Papapanagiotou, P.: A workflow-driven formal methods approach to the generation of structured checklists

- for intrahospital patient transfers. *IEEE journal of biomedical and health informatics* **21**(4), 1156–1162 (2016)
56. Marquard, M., Shahzad, M., Slaats, T.: Web-based modelling and collaborative simulation of declarative processes. In: H.R. Motahari-Nezhad, J. Recker, M. Weidlich (eds.) *Business Process Management*, pp. 209–225. Springer International Publishing, Cham (2015)
 57. Montali, M.: Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach, *Lecture Notes in Business Information Processing*, vol. 56. Springer (2010)
 58. Montali, M., Pesic, M., van der Aalst, W.M., Chesani, F., Mello, P., Storari, S.: Declarative specification and verification of service choreographiess. *ACM Transactions on the Web (TWEB)* **4**(1), 3 (2010)
 59. Mukkamala, R.: A formal model for declarative workflows: dynamic condition response graphs. it university of copenhagen. Ph.D. thesis, IT University of Copenhagen (2012)
 60. Mukkamala, R.R.: A formal model for declarative workflows - dynamic condition response graphs. Ph.D. thesis, IT University of Copenhagen (2012)
 61. Mukkamala, R.R., Hildebrandt, T., Tøth, J.B.: The resultmaker online consultant: From declarative workflow management in practice to Itl. In: *Proceedings of the 2008 12th Enterprise Distributed Object Computing Conference Workshops, EDOCW '08*, pp. 135–142. IEEE Computer Society, Washington, DC, USA (2008). DOI 10.1109/EDOCW.2008.57. URL <http://dx.doi.org/10.1109/EDOCW.2008.57>
 62. Mukkamala, R.R., Hildebrandt, T.T.: From dynamic condition response structures to büchi automata. In: *2010 4th IEEE International Symposium on Theoretical Aspects of Software Engineering*, pp. 187–190. IEEE (2010)
 63. Nekrasaite, V., Parli, A.T., Back, C.O., Slaats, T.: Discovering responsibilities with dynamic condition response graphs. In: *Accepted for Proceedings of 31st International Conference on Advanced Information Systems Engineering (CAiSE 2019)* (2019)
 64. Nielsen, M., Plotkin, G., Winskel, G.: Petri nets, event structures and domains. In: G. Kahn (ed.) *Semantics of Concurrent Computation, Lecture Notes in Computer Science*, vol. 70, pp. 266–284. Springer Berlin / Heidelberg (1979). URL <http://dx.doi.org/10.1007/BFb0022474>. 10.1007/BFb0022474
 65. Object Management Group: Case Management Model and Notation, version 1.0. Webpage (2014). <http://www.omg.org/spec/CMMN/1.0/PDF>
 66. Object Management Group BPMN Technical Committee: Business Process Model and Notation, version 2.0. Webpage (2011). <http://www.omg.org/spec/BPMN/2.0/PDF>
 67. Papapanagiotou, P., Fleuriot, J.: Workflowm: a logic-based framework for formal process specification and composition. In: *International Conference on Automated Deduction*, pp. 357–370. Springer (2017)
 68. Papapanagiotou, P., Fleuriot, J.: A pragmatic, scalable approach to correct-by-construction process composition using classical linear logic inference. In: *International Symposium on Logic-Based Program Synthesis and Transformation*, pp. 77–93. Springer (2018)
 69. Pesic, M., Schonenberg, H., Van der Aalst, W.M.: Declare: Full support for loosely-structured processes. In: *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, pp. 287–287. IEEE (2007)
 70. 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)*, 15–19 October 2007, Annapolis, Maryland, USA, pp. 287–300 (2007). DOI 10.1109/EDOC.2007.25. URL <http://doi.ieeecomputersociety.org/10.1109/EDOC.2007.25>
 71. Popova, V., Fahland, D., Dumas, M.: Artifact lifecycle discovery. *International Journal of Cooperative Information Systems* **24**(01), 1550001 (2015). DOI 10.1142/S021884301550001X. URL <https://doi.org/10.1142/S021884301550001X>
 72. Rozinat, A., Van der Aalst, W.M.: Conformance checking of processes based on monitoring real behavior. *Information Systems* **33**(1), 64–95 (2008)
 73. Sadiq, S., Sadiq, W., Orlowska, M.: Pockets of flexibility in workflow specification. In: H. S.Kunii, S. Jajodia, A. Sølvberg (eds.) *Conceptual Modeling — ER 2001, Lecture Notes in Computer Science*, vol. 2224, pp. 513–526. Springer Berlin Heidelberg (2001)
 74. Santos França, J.B.d., Netto, J.M., do E. S. Carvalho, J., Santoro, F.M., Baião, F.A., Pimentel, M.: Kipo: the knowledge-intensive process ontology. *Software & Systems Modeling* **14**(3), 1127–1157 (2015). DOI 10.1007/s10270-014-0397-1. URL <https://doi.org/10.1007/s10270-014-0397-1>
 75. Schöning, S., Cabanillas, C., Jablonski, S., Mendling, J.: A framework for efficiently mining the organisational perspective of business processes. *Decision Support Systems* **89**, 87–97 (2016)
 76. Schöning, S., Zeising, M.: The dpil framework: Tool support for agile and resource-aware business processes. *BPM (Demos)* **1418**, 125–129 (2015)
 77. Schunselaar, D.M.M., Slaats, T., Maggi, F.M., Reijers, H.A., van der Aalst, W.M.P.: Mining hybrid business process models: A quest for better precision. In: W. Abramowicz, A. Paschke (eds.) *Business Information Systems*, pp. 190–205. Springer International Publishing, Cham (2018)
 78. Shalev-Shwartz, S., Ben-David, S., Press, C.U.: *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press (2015). URL <https://books.google.co.uk/books?id=tBVctAEACAAJ>
 79. Slaats, T.: Flexible process notations for cross-organizational case management systems. Ph.D. thesis, IT University of Copenhagen (2015)
 80. Slaats, T.: Declarative and Hybrid Process Discovery: Recent Advances and Open Challenges. *Journal on Data Semantics* **9**(1), 3–20 (2020). DOI 10.1007/s13740-020-00112-9. URL <https://doi.org/10.1007/s13740-020-00112-9>
 81. Slaats, T., Debois, S., Hildebrandt, T.: Open to change: A theory for iterative test-driven modelling. In: M. Weske, M. Montali, I. Weber, J. vom Brocke (eds.) *Business Process Management*, pp. 31–47. Springer International Publishing, Cham (2018)
 82. Slaats, T., Mukkamala, R.R., Hildebrandt, T., Marquard, M.: Ex-formatics declarative case management workflows as dcr graphs. In: F. Daniel, J. Wang, B. Weber (eds.) *Business Process Management*, pp. 339–354. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
 83. Slaats, T., Schunselaar, D.M.M., Maggi, F.M., Reijers, H.A.: The semantics of hybrid process models. In: C. Debruyne, H. Panetto, R. Meersman, T. Dillon, e. Kühn, D. O’Sullivan, C.A. Ardagna (eds.) *On the Move to Meaningful Internet Systems: OTM 2016 Conferences*, pp. 531–551. Springer International Publishing, Cham (2016)
 84. Smedt, J.D., Weerdt, J.D., Vanthienen, J.: Fusion miner: Process discovery for mixed-paradigm models. *Decision Support Systems* **77**, 123 – 136 (2015). DOI <https://doi.org/10.1016/j.dss.2015.06.002>. URL <http://www.sciencedirect.com/science/article/pii/S0167923615001165>
 85. Tjies Slaats: DisCoveR. <https://github.com/tslaats/DisCoveR> (2020)
 86. Van Der Aalst, W.: *Process mining: discovery, conformance and enhancement of business processes*, vol. 2. Springer (2011)
 87. Verbeek, H., de Carvalho, R.M.: Log skeletons: A classification approach to process discovery. *arXiv preprint arXiv:1806.08247* (2018)
 88. Völzer, H.: An overview of bpmn 2.0 and its potential use. In: J. Mendling, M. Weidlich, M. Weske (eds.) *Business Process Modeling Notation, Lecture Notes in Business Information Processing*,

- vol. 67, pp. 14–15. Springer Berlin Heidelberg (2010). DOI 10.1007/978-3-642-16298-5_3. URL http://dx.doi.org/10.1007/978-3-642-16298-5_3
89. Weske, M.: Business Process Management - Concepts, Languages, Architectures, 2nd Edition. Springer (2012). DOI 10.1007/978-3-642-28616-2. URL <http://dx.doi.org/10.1007/978-3-642-28616-2>
 90. Westergaard, M., Maggi, F.M.: Looking into the future. In: R. Meersman, H. Panetto, T. Dillon, S. Rinderle-Ma, P. Dadam, X. Zhou, S. Pearson, A. Ferscha, S. Bergamaschi, I.F. Cruz (eds.) On the Move to Meaningful Internet Systems: OTM 2012, pp. 250–267. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
 91. Westergaard, M., Slaats, T.: Mixing paradigms for more comprehensible models. In: F. Daniel, J. Wang, B. Weber (eds.) Business Process Management, pp. 283–290. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
 92. Westergaard, M., Stahl, C., Reijers, H.A.: Unconstrainedminer: Efficient discovery of generalized declarative process models (2013)
 93. Wiemuth, M., Junger, D., Leitritz, M., Neumann, J., Neumuth, T., Burgert, O.: Application fields for the new object management group (omg) standards case management model and notation (cmmn) and decision management notation (dmn) in the perioperative field. *International journal of computer assisted radiology and surgery* **12**(8), 1439–1449 (2017)
 94. Zeising, M., Schonig, S., Jablonski, S.: Towards a common platform for the support of routine and agile business processes. In: Collaborative Computing: Networking, Applications and Work-sharing (CollaborateCom), 2014 International Conference on, pp. 94–103. IEEE (2014)
 95. Zugal, S., Soffer, P., Haisjackl, C., Pinggera, J., Reichert, M., Weber, B.: Investigating expressiveness and understandability of hierarchy in declarative business process models. *Software & Systems Modeling* **14**(3), 1081–1103 (2015). DOI 10.1007/s10270-013-0356-2. URL <https://doi.org/10.1007/s10270-013-0356-2>

Weighing the Pros and Cons: Process Discovery with Negative Examples

Tijs Slaats¹, Søren Debois^{2,3}, and Christoffer Olling Back¹

¹ Department of Computer Science, University of Copenhagen, Denmark
slaats@di.ku.dk, back@di.ku.dk

² IT University of Copenhagen, Denmark
debois@itu.dk

³ DCR Solutions A/S, Denmark

Abstract. Contemporary process discovery methods take as inputs only *positive* examples of process executions, and so they are *one-class classification* algorithms. However, we have found *negative* examples to also be available in industry, hence we propose to treat process discovery as a *binary classification* problem. This approach opens the door to many well-established methods and metrics from machine learning, in particular to improve the distinction between what should and should not be allowed by the output model. Concretely, we (1) present a formalisation of process discovery as a binary classification problem; (2) provide cases with negative examples from industry, including real-life logs; (3) propose the Rejection Miner binary classification procedure, applicable to any process notation that has a suitable syntactic composition operator; and (4) apply this miner to the real world logs obtained from our industry partner, showing increased output model quality in terms of accuracy and model size.

Keywords: Process mining, binary classification, negative examples, labelled event logs

1 Introduction

From the perspective of machine learning, process discovery [28] sits uneasily in the gap between unary and binary classification problems [16,25]. Popular contemporary miners, e.g. [4,18], approach process discovery as unary classification: given only positive examples (the input log) they generate a classifier (the output model) which recognizes traces (adhering to the output model) that resemble the training data. However, a process model is really a binary classifier: it classifies traces into those it accepts (desired executions of the process) and those it does not (undesired executions of the process).

Binary classification in machine learning relies on having access to examples of both classes. For process discovery, this means having not only positive examples of desired behaviour to be accepted by the output model, but also negative examples of undesired behaviour that should be rejected.

Negative examples also underpin a substantial part of the mechanics and theory of machine learning, in particular on model evaluation. Output models are evaluated on measures comparing ratios of true and false positives and negatives; however, absent negative examples, it is impossible to apply such measures. Accordingly, in process discovery, we use measures based only on true positive answers, such as *recall*; we are deprived of more fine-grained measures involving true negative or false positive answers such as *accuracy*.

In practical process discovery, negative examples would help distinguish between incidental correlation and actual rules. For instance, suppose that in some log, whenever we see an activity B , that B is preceded by an activity A . Does that mean that we can infer the declarative rule $A \rightarrow \bullet B$, that A is required before B may happen? In general, no. E.g., if A is “call taxi” and B is “file minutes from weekly status meeting”; by coincidence, we always call a taxi in the morning the day we file minutes, but clearly there is no rule that we must call a taxi before filing minutes. Conversely, if A is “approve payment” and B is “execute payment”, very likely it is a rule that B must be preceded by A . Negative examples potentially help here: If BA is in the set of negative examples, adding the rule $A \rightarrow \bullet B$ is justified, as it rejects both those traces. Conversely, if a rule rejects no trace from the negative examples, it is not necessary but *discretionary* for the miner to leave out or keep in.

As shown by [22] negative examples *do* exist in practice, some mining algorithms that include negative examples have been proposed, e.g. [22,17], and interestingly recent editions of the process discovery contest⁴ have moved towards using labelled test logs (but not training logs!) to rank submissions. In this paper we add to these developments with the following contributions:

1. We formalize process discovery as a binary classification problem, and show that not all process notations can express complete solutions to this problem (sec. 3).
2. We propose the *Rejection Miner*, a notation-agnostic binary mining procedure applicable to *any* process notation with a syntactic composition operator sec. 4.
3. We describe two cases where negative examples were encountered in industry and provide data sets [27] (sec. 5).
4. We implement a concrete Rejection Miner and apply it to these data sets, comparing exploratively to contemporary unary miners (sec. 6). The miner has been integrated in the commercial dcrgraphs.net modelling tool.

For the latter experiments, do note that the contemporary unary miners with which we compare do not take into account the negative examples. They must guess from the positive examples which traces to reject, whereas the Rejection Miner has the negative examples to guide it. We find that the Rejection Miner achieves noticeably better accuracy, in particular on out-of-sample tests, and produces models that are orders-of-magnitude smaller than the unary miners. We also note that we chose not to compare to other binary miners, as we did not

⁴ PDC 2019, PDC 2020

aim to show the merits of the rejection miner in particular, but of binary mining in general. We chose the rejection miner as representative for binary mining as it allows us to build DCR Graphs, which were requested by the industry partner. The implementation of the Rejection Miner is available on-line [26].

Related work There have been several earlier works framing process mining as a binary classification task. [17] formulates constraints as Horn clauses and uses the ICL learning algorithm to successively find constraints which remove negative examples, stopping when there are no negative examples left. They translate these generated clauses to DECLARE. The rejection miner generalises this approach in that (a) it replaces the horn clauses with a generic notion of "model" for notations with composition (or synchronous product of models), and thus applies directly to a plethora of languages such as Delcare and DCR Graphs, (b) the rejection miner leaves the choice of which clauses to prune until after a set of constraints ruling out all negative constraints is found, opening the door to non-greedy minimisation, and most importantly (c) we proof correctness for the rejection miner. [22] proposes an approach where traces are represented as points in an n -dimensional space (n being the number of unique event classes of the log), each point representing the multiplicity of the event classes in that trace. Finding a model is then reduced to the problem of finding a convex hull for the points such that positive points are included and negative points excluded. Whereas the work only considers the multiplicity of event classes in negative traces, the rejection miner is able to also consider the temporal ordering of individual events, while the former works well for the generation of Petri net models, it is less suitable for declarative notations. In [13], the authors artificially generate negative labels, but at the level of individual events rather than traces. The authors also defined process mining oriented metrics based on the resulting true positive/negative labels at the level of events. In [23] the development of binary process discovery algorithms was identified as a key open challenge for the field of declarative process discovery.

2 Process Notations and Unary Discovery

We recall the traditional definitions of event logs etc. [28].

Definition 1 (Events, traces, logs). *Assume a countably infinite universe \mathcal{A} of all possible activities. As usual, an alphabet $\Sigma \subseteq \mathcal{A}$ is a set of activities, and the Kleene-star Σ^* denotes the countably infinite set of finite strings or sequences over Σ ; we call such a string a trace. A log L is a multiset of occurrences of traces $L = \{t_1^{m_1}, \dots, t_n^{m_n}\}$ where $m_k > 0$ is the multiplicity of the trace $t_k \in \Sigma$. We write \mathcal{L}_Σ for the set of all event logs over alphabet Σ .*

When convenient, we treat an event log L also as simply a set of traces by ignoring multiplicities.

When we discuss unary and binary process discovery in the abstract in later sections, we will be interested in applying discovery to a variety of process notations; and we shall propose a miner which can be instantiated to any notation

with a suitable composition operator. To make such statements formally, we need a formal notion of process notation.

Definition 2 (Process notation). *A process notation for an alphabet Σ comprises a set of models \mathbf{M} and an interpretation function $\llbracket - \rrbracket : \mathbf{M} \rightarrow \mathcal{P}(\Sigma^*)$ assigning to each individual model m the set of traces $\llbracket m \rrbracket$ accepted by that model. For a set $S \subseteq \Sigma^*$, we write $m \models S$ iff $S \subseteq \llbracket m \rrbracket$.*

While a process notation comprises the three components Σ , \mathbf{M} , and $\llbracket - \rrbracket$, when no confusion is possible we shall allow ourselves to say “consider a process notation \mathbf{M} ”, understanding the remaining two components to be implicit.

Example 3. Here is a toy declarative formalism which allow exactly the condition constraint of DECLARE [1,21] or DCR [14,10] over a countably infinite alphabet $\Sigma = \{A, B, C, \dots\}$. A “model” is any finite set of pairs $(x, y) \in \Sigma \times \Sigma$, and we interpret each such pair as a condition from x to y . Formally:

$$\begin{aligned} \mathbf{M}_{\text{cond}} &= \{C \subseteq \Sigma \times \Sigma \mid C \text{ finite}\} \\ \llbracket C \rrbracket &= \{t \in \Sigma^* \mid \forall (x, y) \in C. \text{ each } y \text{ in } t \text{ is preceded by } x\} \end{aligned}$$

For instance, $\{(A, B)\} \in \mathbf{M}_{\text{cond}}$ is a model consisting of a single condition from A to B . In DECLARE or DCR, we would write this model “ $A \rightarrow \bullet B$ ”. Just as in DECLARE or DCR, this model admits all traces in which any occurrence of B is preceded by an occurrence of A . That is, this model admits the trace AB , but not B or $BABA$. Formally, we write

$$\begin{aligned} AB \in \llbracket \{(A, B)\} \rrbracket & \quad \text{or} \quad \{(A, B)\} \models \{AB\} \\ \{B, BABA\} \not\subseteq \llbracket \{(A, B)\} \rrbracket & \quad \text{or} \quad \{(A, B)\} \not\models \{B, BABA\} \end{aligned}$$

Any process modelling formalism with trace semantics is a process notation in the above sense; such formalisms include DECLARE, DCR, and Workflow Nets [2] (see also [28]).

We conclude this Section by pinning down process discovery: a procedure which given an event log produces a process model which admits that log. Assume a fixed alphabet Σ , and write \mathcal{L}_Σ for the set of all valid event logs over Σ .

Definition 4 (Unary process discovery). *A unary process discovery algorithm γ for a process notation $(\mathbf{M}, \llbracket - \rrbracket)$ over Σ is a function $\gamma : \mathcal{L}_\Sigma \rightarrow \mathbf{M}$. We say that γ has perfect fitness iff for all $L \in \mathcal{L}_\Sigma$ we have $\gamma(L) \models L$.*

Anticipating our binary miners, we shall refer to “perfect fitness” also as *positive soundness* of the miner.

3 Process Discovery as Binary Classification

We proceed to consider process discovery a binary classification problem. This approach presumes that we have not only positive examples (the set L in Def. unary-mining), which the output model must accept, but also a set of negative examples, which the output model must reject.

Example 5. Consider again the condition models \mathbf{M}_{cond} of Ex. 3. Take as positive set of examples the singleton set $\{AB\}$, and take as negative examples the set $\{BA, B\}$. One model which accepts the positive example and rejects the negative ones is the singleton condition $\{(A, B)\}$. This model admits the positive example AB , because B is preceded by A ; and it rejects the negative examples, because in both of the traces B and BA , the initial B is *not* preceded by A .

The negative examples here help solve the relevancy problem that plagues unary miners for declarative formalisms: The positive example AB clearly supports the constraint “ A is a condition for B ”, however, as we saw in the introduction, with only positive examples and without domain knowledge, we cannot know whether this is a coincidence or a hard requirement. In the present example, the negative examples tell us that our model must somehow reject the trace BA , encouraging us to include the condition $A \rightarrow \bullet B$. Negative examples still leave a degree of freedom for the discovery procedure, though: a condition $C \rightarrow \bullet B$ would *also* accept the positive example and reject the negative ones.

Unfortunately, a model accepting a given set P of positive examples and rejecting a given set N of negative ones does not necessarily exist: At the very least, we must have P and N disjoint. To cater to such ambiguous inputs, we allow a binary miner to refuse to produce a model.

Definition 6 (Binary process discovery). *Let \mathbf{M} be a process notation for an alphabet Σ . A binary-classification process discovery algorithm (“binary miner”) is a partial function $\eta : \mathcal{L}_\Sigma \times \mathcal{L}_\Sigma \rightarrow \mathbf{M}$, taking sets of positive and negative examples P, N to a model $\gamma(P, N)$. We require that $\eta(P, N)$ is defined whenever P, N are disjoint.*

In the rest of this paper, unless otherwise stated, we shall implicitly assume that examples P, N are disjoint. We proceed to generalise the notion of fitness from unary mining.

Definition 7 (Soundness, perfection). *Let $P, N \subseteq \mathcal{L}_\Sigma$ be positive and negative examples. We say that a binary miner η is positively sound at N, P iff $\eta(P, N) \models P$. Similarly, we say that η has negatively sound at N, P iff $N \cap \llbracket \eta(P, N) \rrbracket = \emptyset$. We say that η is perfect iff it any disjoint N, P it is defined and both positively and negatively sound.*

In other words: A perfect binary miner produces an output whenever its positive and negative examples are not in direct conflict, and that output admits neither false positives nor false negatives.

Over- and underfitting of out-of-sample data. A perfect binary miner has no choice in how it treats the elements of P and N : it must admit its positive examples P and reject its negative examples N . It is the remaining *undecided* traces where it has a choice. In the limits, we have the overfitting “maximally rejecting miner”, whose output always accepts exactly P and nothing else; and the underfitting “maximally accepting miner”, whose output rejects exactly N and nothing else.

However, unlike the unary case, where perfect fitness miners are generally quite easy to come by, **perfect binary miners do not necessarily exist**, and helpful ones may in practice be quite hard to come by.

First, let us try to use a unary miner as a binary one. We do so by simply ignoring the negative examples and applying our unary miner to the positive ones. In this case, it is easy to show that for any unary miner (for any notation!) which never returns exactly its input log, we can construct a negative example which will be accepted by the output model of that miner for those positive examples:

Proposition 8. *Let γ be a unary miner for a notation \mathbf{M} over alphabet Σ , and assume that γ for all L we have $\llbracket \gamma(L) \rrbracket \neq L$. Then for all $P \in \mathcal{L}_\Sigma$ there exists a $N \in \mathcal{L}_\Sigma$ s.t. N and P are disjoint, yet N is accepted by the output model $\gamma(L)$.*

So in this sense, **non-trivial unary miners never generalise to binary ones**. This is perhaps not entirely surprising. Much less obvious, and a core difference between binary and unary mining, we find that some *notations* cannot express distinctions fine enough to distinguish between positive and negative examples. This is in stark contrast to the unary case, where essentially all notations have a model accepting all traces (the “flower model”); moreover, all commonly accepted notations are able to express any finite language, and so for any input log (finite language), a perfectly fitting model must exist.

However, in the binary case, even though our example notation admits the “flower model”, it is still too coarse to admit a perfect binary miner.

Lemma 9. *In \mathbf{M}_{cond} , take positive examples $P = \{ABC\}$, and negative examples $N = \{AB\}$. Then no model $m \in \mathbf{M}_{\text{cond}}$ exists such that $m \models P$ yet $m \not\models N$.*

Proof. Suppose m is a model with $m \not\models \{AB\}$. Then m requires something preceding either A or B , something which is apparently not there. But then that something is missing also from ABC .

In fact, we prove below that **no perfect binary miner can exist in any notation that has only finitely many possible models**. To understand the ramifications of this Theorem, consider again DCR and DECLARE. For DCR or DECLARE models over a fixed finite alphabet (e.g., the set of tasks present in a given log), DCR has infinitely many such models (with distinct semantics), whereas DECLARE has only finitely many. To see this, note that in DCR, because labels and events are not one-one, we can keep adding events that do affect behaviour, while remaining within a finite set of observable tasks. In DECLARE, if there are n activities to choose from, you can populate only finitely many DECLARE templates with those finitely many tasks. Since the arity of DECLARE templates is bounded, you are left with only finitely many models.

Note the following consequence for DECLARE: **any binary miner for DECLARE has inputs P, N for which the output a model has either false positives or false negatives**.

Theorem 10. *No perfect binary miner exists for any finite process notation \mathbf{M} over any non-empty finite alphabet Σ .*

Proof. We construct finite positive and negative examples P and N such that no model accepts P and rejects N .

First, we construct N . Let I^+ as the subset of models that accepts infinitely many traces, i.e., $I^+ = \{m \in \mathbf{M} \mid \llbracket m \rrbracket \text{ infinite}\}$. Since there are only finitely many models, I^+ is finite, and wlog write it $I^+ = \{m_1, \dots, m_n\}$. For each m_i , choose a $t_i \in \llbracket m_i \rrbracket$, and define $N = \{t_1, \dots, t_n\}$.

Next, we construct P . Let I^- the subset of models which reject infinitely many traces, i.e., $I^- = \{m \in \mathbf{M} \mid \mathbb{C}(\llbracket m \rrbracket) \text{ infinite}\}$. Again I^- is finite and we write it wlog $I^- = \{p_1, \dots, p_k\}$. For each of p_j , pick a trace s_j such that $s_j \notin \llbracket p_j \rrbracket$ and $s_j \notin N$ —this is always possible because $\mathbb{C}(\llbracket m_j \rrbracket)$ is infinite and N finite. Then define $P = \{s_1, \dots, s_k\}$. Note that by construction P and N are disjoint.

Finally, let $m \in \mathbf{M}$ be a model. At least one of $\llbracket m \rrbracket$ and $\mathbb{C}(\llbracket m \rrbracket)$ must be infinite; we show that in neither case can m be the output of a perfect binary miner applied to P, N . If $\llbracket m \rrbracket$ is infinite, then $m \in I^+$, say $m = m_i$, and it follows that $m \models t_i \in N$; hence m fails to reject all negative examples. If on the other hand $\mathbb{C}(\llbracket m \rrbracket)$ is infinite, then $m \in I^-$, say $m = p_j$ and it follows that $s_j \notin \llbracket p_j \rrbracket = \llbracket m \rrbracket$; hence m fails to accept the positive example $s_j \in P$.

Alternatively, the above proof can possibly be used to show that there are infinitely many problems N, P with pairwise distinct solutions; the Theorem then follows from the Vapnik-Chervonenkis dimension [3] of the set of interpretations of the finite set of models being necessarily finite, and so unable to shatter this infinite set of distinct solutions.

In unary mining, we may construct a perfect fitness miner like this: As notation, pick simply finite sets of traces, and let the semantics of the notation be that a model (set of traces) T accepts a trace t iff $t \in T$. Then the function $\eta(P) = P$ is a perfect fitness miner. This generalises to any notation strong enough to characterise exactly a given set of T of traces. Obviously, this unary miner has little practical relevance.

It is interesting to note that a similar perfect binary miner exist. Pick as notation pairs of sets of traces T, U , with semantics that T, U accepts t iff $t \in T$ and $t \notin U$. Clearly the function $\eta(P, N) = (P, N)$ is a perfect binary miner, although again, not a particularly helpful one. However, the construction shows that a perfect binary miner exists for any notation strong enough to exactly characterise membership resp. non-membership of finite sets of traces. Notable examples here are Petri-nets and BPMN (through an exclusive choice over the set of positive traces); so it follows that a (trivial) binary miner exists for these notations.

4 Rejection Miners

We proceed to construct a family of binary miners we call “Rejection miners”, defined for *any* process notation which has a behaviour-preserving syntactic

model composition. Rejection miners are parametric in a “pattern oracle” which selects a set of patterns for consideration; if the patterns selected allow it, the output of the rejection miner is perfect. When they do not, the miner does a greedy approximation to optimise for accuracy (i.e., maximising the ratio of true positives and negatives to all inputs).

Definition 11 (Additive process notation). *We say that a process formalism \mathbf{M} over Σ is additive if it comes equipped with a commutative monoid $(\oplus, \mathbf{1})$ on \mathbf{M} such that*

$$\llbracket \mathbf{1} \rrbracket = \Sigma^* \quad (1)$$

$$\llbracket m \oplus n \rrbracket = \llbracket m \rrbracket \cap \llbracket n \rrbracket \quad (2)$$

We lift the monoid operator to sequences and write $\bigoplus_{i < n} m_i = m_1 \oplus \dots \oplus m_{n-1}$,

That is, an additive formalism has a flower model $\mathbf{1}$ and a model combination operator \oplus . This operator combines two models into a compound one, such that this compound model accepts *exactly* the traces accepted by both of the two original models. DECLARE is an additive formalism: A DECLARE model is a finite set of constraints; the empty such set accepts all traces ($\mathbf{1}$), and the union of two such sets is again such a set, with exactly the desired semantics (\oplus). DCR also has a model composition, where the composite model is the union of events, markings, and constraints [15,9]. However, this composition does not preserve semantics in the general case.

In practice, any process notation can be considered additive by forming the synchronous product of models: To check whether a given trace t conforms to a composite model $m \oplus n$, we simply check whether $m \models t$ and $n \models t$. Incidentally, this is a popular implementation mechanism for DECLARE constraints (see, e.g., [12,8]): Each individual instantiated template is implemented as a finite automaton; a set of constraints is then checked by the synchronous product of that automaton.

The key property of additive process notations used for rejection miners is that in such a notation, we can think about models as being the sum of their parts, and the problem of mining can then be reduced to finding suitable such parts. For this approach to be able to generate all models, we would also need to know a subset $\mathbf{S} \subseteq \mathbf{M}$ which generates \mathbf{M} under the model composition operator $-\oplus-$. DECLARE and DCR clearly has such subsets. In keeping with declarative notations and nomenclature, we will refer to such part models as “constraints” in the sequel, however, we emphasise that there is nothing special about them: A constraint m is just another model $m \in \mathbf{M}$.

A rejection miner is parametric in two sub-components: A *pattern oracle*, which given positive and negative examples produces a finite set of (hopefully) relevant constraints; and a *constraint minimiser*, which given a sequence of constraints known to fully reject a set of negative examples selects a subset still fully rejecting those examples.

Definition 12 (Rejection miner components). Let \mathbf{M} be a process notation over an alphabet Σ . A pattern oracle is a function $\text{patterns} : \mathcal{L}_\Sigma \times \mathcal{L}_\Sigma \rightarrow \mathbf{M}^*$. A minimiser is a function $\text{minimise} : \mathbf{M}^* \times \mathcal{L}_\Sigma \rightarrow \mathbf{M}^*$ satisfying:

1. if $\sigma \in \mathbf{M}^*$ fully rejects L , then also $\text{minimise}(\sigma, L)$ fully rejects L ; and
2. $\text{minimise}(\sigma, L)$ contains only elements from the input sequence σ .

An example pattern oracle for DECLARE would be the function that produces all possible instantiations of all templates with activities observed in either of its input logs. An example minimiser is the *greedy minimiser* which, starting from the left of the list of constraints, removes those constraints which reject only traces in N that are already rejected by preceding constraints.

Algorithm 13 (Rejection miner). Let \mathbf{M} be an additive notation over Σ , let patterns be a pattern oracle and let minimise be a minimiser.

```

1: procedure REJECTIONMINER( $P, N$ )
2:    $[m_1, \dots, m_n] \leftarrow \text{patterns}(P, N)$ 
3:    $\sigma \leftarrow [m_i \mid m_i \models P]$  ▷ remove  $m_j$  where  $m_j \not\models P$ 
4:    $\sigma \leftarrow \bigoplus \text{minimise}(\sigma, N)$ 
5:   if  $[\bigoplus \sigma] \cap N \neq \emptyset$  then ▷ are any negative examples not rejected?
6:      $\delta \leftarrow 1$ 
7:     while  $\delta > 0$  and  $|\sigma| < n$  do
8:        $N' \leftarrow \{n \in N \mid \bigoplus \sigma \models n\}$  ▷ negative examples not yet rejected
9:        $P' \leftarrow \{p \in P \mid \bigoplus \sigma \models p\}$  ▷ positive examples currently accepted
10:       $m, \delta \leftarrow \max_{m_j \notin \sigma} (|\{n \in N' \mid m_j \not\models n\}| - |\{p \in P' \mid m_j \not\models p\}|)$ 
11:      if  $\delta > 0$  then
12:         $\sigma \leftarrow \sigma, m$ 
13:      end if
14:    end while
15:  end if
16: end procedure

```

A brief explanation: On line 2, the pattern oracle is invoked to produce a finite list $[m_1, \dots, m_n]$ of relevant constraints. On Line 3, those constraints *not* modelling the positive examples P are filtered out; only the constraints m_i which *do* model P are retained; we assign the resulting list to σ . We then apply the minimiser in Line 4, which by Def. 12 at most removes constraints. On Line 5, we check whether all negative examples are rejected; if so, we have found a perfect model and return it.

Otherwise, we turn to approximation. In the loop in Line 7 to 13, we repeatedly compute the set N' of negative examples not yet rejected and P' of positive examples currently accepted. In Line 10, we iterate over the constraint m_j of the original pattern oracle and compute for each the difference δ_j between how many additional negative examples m_j rejects (wins) and how many already accepted positive examples m_j rejects (losses); we then pick the m_j with the maximum δ_j . If $\delta > 0$, adding the constraint m_j will improve accuracy, and we add it to

the set of output constraints. If $\delta \leq 0$, we cannot improve accuracy by including any more constraints, and the loop terminates.

Recall from the previous section the notions of maximally accepting or maximally rejecting perfect binary miners. The minimiser provides a handle for pushing the Rejection Miner towards either of these extremes. Using the identity function as the minimiser will retain all constraints, and so reject the most undecided traces. Conversely, using a minimiser which finds a least subset of constraints rejecting N will remove more constraints, accepting more undecided traces.

The rejection miner is not in general a perfect binary miner: The patterns σ provided to it by the `patterns` might not, even if all of them were retained, be strong enough to fully reject the set N of negative examples while retaining the positive ones. Moreover, while the rejection miner in practice produces decent results, its approximation phase does not find a subset of patterns with optimal accuracy because of its greedy nature.

However, the rejection miner will *always* accept all the positive examples; and if the selected patterns σ has any subset σ' which accepts P and rejects N , the rejection miner will find such a subset.

Proposition 14. *Let `patterns` be a pattern oracle, let `minimise` be a minimiser, and let N, P be disjoint sets of negative and positive examples. Then the rejection miner for this oracle and minimiser has positive soundness at N, P . Moreover if there exists $\sigma \subseteq \text{patterns}(N, P)$ such that σ accepts P and fully rejects N , then the rejection miner also has negative soundness at N, P .*

Proof (sketch). The former is immediate from line 4; the latter is immediate by the requirements 1 and 2 of Definition 12.

That is: On all inputs where the pattern oracle produces patterns strong enough to make the distinction, the rejection miner will exhibit neither false negatives nor positives.

5 Cases with Negative Examples

The development of the rejection miner was not just motivated by academic, but also industrial interest. When pursuing process mining activities in practice we regularly see opportunities to label data and in some cases we have even been asked directly by commercial partners to include counter examples in the construction of models. In this section we discuss the two most developed cases we've encountered, where we both had the opportunity to extract labelled data and publish it in an anonymized format. The negative examples in these cases arise from test-driven development and as failures in process engineering.

5.1 DCR Solutions: Test-driven Modelling

Danish vendor of adaptive case-management systems, *DCR Solutions*, offers the on-line process modelling portal `dcrgraphs.net`. In this tool, modellers define *required* (positive) resp. *forbidden* (negative) test cases (traces), expected to be

accepted resp. rejected by the model under development. The test cases are also used as input to a process discovery algorithm, which dynamically recommends new constraints to modellers [5]. However, the algorithm used only the positive test-cases, ignoring the negative ones. The extension to consider also those negative ones has been repeatedly requested by the developers of the portal and was implemented as part of this paper. DCR Solutions has kindly allowed us to make the entire data set of test-cases produced in the portal available in an anonymized form [27].

5.2 Dreyer Foundation: Process Engineering

The Danish *Dreyer Foundation* supports budding lawyers and architects, and has previously released an anonymised log of casework [11]. This log documents also testing and early stages of deployment of the system. In a number of cases, process instances that had gone astray were reset to their starting state and partially replayed. The log contains reset markers, and so provides clear negative examples: those prefixes that ended in a reset. We make available here also this partitioning into positive and negative examples [27].

6 Experimental Results

We report on exploratory experiments applying an instantiation of the Rejection Miner to the data sets of Sec. 5, comparing results to current major unary miners.

Data sets The DCR Solutions case (Sec. 5.1) comprises 215 logs, each containing at least one negative example, and each produced by users of the portal to codify what a single model should or should not do. The logs contain 7030 events, 1681 unique activities, 589 negative and 705 positive traces. Logs vary enormously in size: the largest log contains 1162 events, 19 activities, 98 negative and 14 positive traces; the smallest log contain but one negative trace of 3 events. Log size distribution is visualised in fig. 1. The Dreyer case (Sec. 5.2) comprises a single log of 10177 events, 33 unique activities, 492 positive and 208 negative traces. The mean trace length is 15 (1–46), and the mean number of activities per trace is 12 (1–24). Both data sets are available on-line [27].

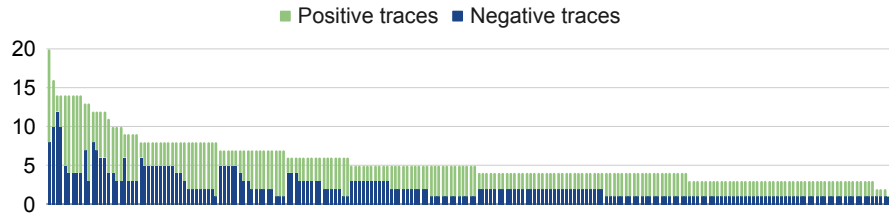


Fig. 1. DCR Solutions data set log size distribution. The largest log of 98 negative and 14 positive traces has been omitted from the diagram.

Metrics Binary classification mining allows us to rely on traditional machine learning metrics [29] of relative misclassification (true and false positives, TP and FP, and true and false negatives, TN and FN). We use in particular the true positive rate (TPR), true negative rate (TNR), accuracy (ACC), balanced accuracy (BAC), positive predictive value (PPV), and F1-score (F1). We recall their definitions in table 1. These particular measures demonstrate the difference between what can be measured in the unary and binary settings. In the setting of unary-classification miners, where we do not have negative examples, we can count only TP and FN. In that setting, we can only measure the true positive rate (TPR)-known as “fitness” in the process mining community-but none of the other measures⁵. But in the setting of binary-classification miners, we can measure also how well the output model recognizes negatives (TNR), how reliable a positive classification is (PPV), and generally how accurately both positive and negative traces are classified (ACC, which counts each trace equally and BAC, which balances between positive and negative traces).

Finally, one goal particular to process discovery is to produce output models that are understandable by humans: Output models are not mere devices for classification; they are vehicles for humans to understand the reasons and structure behind that classification. To this end, smaller models are more helpful, so we calculate also the size of the models, dependent on their notation. For the pattern-based notations such as DECLARE, we use the number of such patterns; for DCR models the number of relations; and for workflow nets the number of edges and places.

Log classification			
Model class.	Pos.	Neg.	ACC = $\frac{TP+TN}{TP+FP+TN+FN}$
Pos.	TP	FP	PPV = $\frac{TP}{TP+FP}$
Neg.	FN	TN	BAC = $\frac{TPR+TNR}{2}$
$TPR = \frac{TP}{TP+FN}$ $TNR = \frac{TN}{FP+TN}$ $F1 = 2 \cdot \frac{PPV \cdot TPR}{PPV+TPR}$			

Table 1. Confusion matrix for binary mining

Rejection Miner We provide a JavaScript implementation of the Rejection Miner, available at [26]. We use a pattern oracle which simply instantiates the following list of DECLARE-like patterns at all activities seen in the log: *Existence(x)*, *Absence(x)*, *Absence2(x)*, *Absence3(x)*, *Condition(x, y)*, *Response(x, y)*, *NotSuccession(x, y)*, *AlternatePrecedence(x, y)*, *DisjunctiveResponse(x, (y, z))*, and *ConjunctiveResponse((x, y), z)*. The oracle outputs patterns sorted by how many negative examples they exclude. Ties are broken by sorting the disjunctive and conjunctive responses last, to de-emphasise these relatively more complex patterns. We then use a greedy

⁵ The name “F1” is used for a metric of unary miners defined like F1 here, except using the escaping-edges notion of precision [6] *en lieu* of the PPV.

minimizer picking patterns from this sorted list of patterns as long as they exclude at least one negative example not already excluded by a previous pattern; in effect, the miner prefers fewer constraints without having to solve the NP-complete problem of finding a minimal subset.

We emphasise the flexibility of the oracle and minimizer selection: if one wants to include more patterns, one simply extends the oracle; if one wants to have a more restrictive model, or a different prioritization of constraints, one simply replaces the minimizer. One can also produce models that sacrifice TPR for accuracy by creating a minimizer that accepts constraints excluding some positive examples, but also excluding many negative examples.

Other miners We compare the Rejection Miner (RM) to flagship miners for three major process notations. For DCR graphs [10,14], we use DisCoveR [20]. DisCoveR is used commercially for model recommendation by DCR solutions. We consider DisCoveR with two settings, the default one (intended to emphasise precision, denoted D), and a “light” version intended to emphasise simplicity (DL). For DECLARE [1,21], we use MINERful [7] and consider three settings, (M1) the most restrictive setting where support=1.0, confidence=0.0, and interest factor=0.0; (M2) a less restrictive setting (likely outputting smaller models) with support=1.0, confidence=0.5, and interest factor=0.25; and (M3) with support=1.0, confidence=0.75, and interest factor=0.5. Finally, for Workflow Nets [2], we use the Inductive Miner [18,28], with a noise threshold of 0.0 (IM) and 0.2 (IMf) respectively.

6.1 Results

We performed both in-sample and out-of-sample testing. For the latter we performed 10-fold validation [24] and calculated our measures as the mean values across 10 randomized attempts. The results are shown in table 2. For the DCR Solutions data set each value is calculated as the mean over all 215 logs. Because of the limited size of most of the logs, we only tested on in-sample data for this case, however, since the primary goal for the company is to find models that accurately fit the training data, in-sample accuracy is highly relevant.

DCR Solutions First, on in-sample test data, the Rejection Miner mines perfectly accurate models on every log. This is a small, but meaningful, improvement over the 0.967 accuracy achieved by DisCoveR light, which is currently used for this task. In practice this means that, given a mapping from the Declarative patterns to DCR Graphs, the Rejection Miner will allow the portal to recommend perfectly accurate models for all test cases that have been defined to-date. Secondly, there is an order-of-magnitude gain in simplicity for the Rejection Miner compared to all other miners: the Rejection Miner requires only 1.5 constraints on average per model. We conjecture that this gain is achieved because knowing what behaviour should be forbidden allows the miner to find precisely the constraints we need, instead of having to propose many constraints to forbid all behaviour that was not explicitly seen in the positive samples. This gain in

Miner	TPR	TNR	ACC	BAC	PPV	F1	Size
DCR Solutions Data set (sec. 5.1) In-sample							
Rejection (RM)	1.000	1.000	1.000	1.000	1.000	1.000	1.5
DisCoveR (D)	1.000	0.927	0.976	0.964	0.971	0.983	24.8
- light (DL)	1.000	0.921	0.974	0.948	0.967	0.981	19.6
MINERful (M1)	1.000	0.881	0.958	0.941	0.949	0.970	120.5
- 0.5/0.25 (M2)	0.997	0.841	0.942	0.919	0.930	0.957	77.6
- 0.75/0.5 (M3)	0.961	0.657	0.848	0.809	0.850	0.877	37.8
Inductive (IM)	1.000	0.860	0.946	0.930	0.932	0.960	22.1
- 0.2 noise (IMf)	1.000	0.860	0.946	0.930	0.932	0.960	22.1
Dreyer Foundation Data set (sec. 5.2) In-sample							
Rejection	1.000	0.928	0.979	0.964	0.970	0.985	6.0
DisCoveR	1.000	0.048	0.717	0.524	0.713	0.832	125.0
- light	1.000	0.048	0.717	0.524	0.713	0.832	71.0
MINERful	1.000	0.067	0.723	0.534	0.717	0.835	1124.0
- 0.5/0.25	1.000	0.0288	0.711	0.514	0.709	0.830	174.0
- 0.75/0.5	1.000	0.005	0.704	0.502	0.704	0.826	102.0
Inductive	1.000	0.019	0.709	0.510	0.707	0.828	160.0
- 0.2 noise	1.000	0.019	0.709	0.510	0.707	0.828	160.0
Dreyer Foundation Data set (sec. 5.2) Out-of-sample							
Rejection	0.985	0.914	0.964	0.950	0.965	0.975	6.2
DisCoveR	0.962	0.362	0.692	0.662	0.706	0.814	127.6
- light	0.968	0.447	0.697	0.707	0.708	0.817	72.9
MINERful	0.906	0.231	0.659	0.569	0.698	0.787	1128.4
- 0.5/0.25	0.962	0.270	0.685	0.616	0.701	0.810	176.4
- 0.75/0.5	0.970	0.081	0.684	0.525	0.698	0.810	104.9
Inductive	0.981	0.339	0.696	0.660	0.703	0.818	158.9
- 0.2 noise	0.983	0.359	0.698	0.671	0.704	0.819	157.8

Table 2. Experiment results

simplicity also directly benefits the business case, as the industry partners have repeatedly voiced a strong preference for fewer, but more relevant, recommended relations. As a result, the Rejection Miner has already been integrated into the portal by the company.

Dreyers Foundation The results show that the Rejection Miner once again provides high levels of accuracy while requiring only a small model. Of most interest are the out-of-sample results, shown in more detail in the boxplots of Figure 2, which indicate that the models found by the Rejection Miner are not only accurate for the training data, but also for unseen test data. In other words, providing the miner with *some* negative examples allows it to accurately predict what *other* negative examples may be seen in the future. In addition there is very little variance in the results of the Rejection Miner, with model size and accuracy scores remaining close to the mean for each randomized run of the 10-fold validation. We also included measures of the run-time performance in Figure 2, showing that the Rejection Miner is several orders of magnitude slower

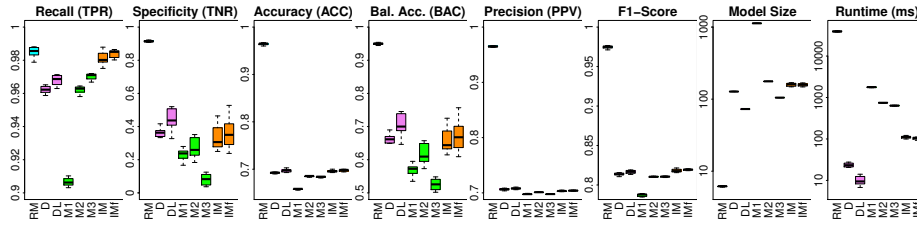


Fig. 2. Boxplots illustrating the distribution of mean performance of various miners across 10 runs of 10-fold cross validation on the Dreyers log.

than the other miners (requiring on average 39.3 seconds to mine the Dreyers log). We stress however that good run-time performance was never a goal for the current prototype, that there are known methods for improving the run-time performance through a more intelligent initial selection of relevant patterns by the oracle [5,19], and that the results do show that the miner is computationally viable for the experimental data.

7 Conclusion

We propose approaching process discovery as a binary classification problem. We provided a formal account of when binary miners exist; proposed the Rejection Miner; introduced real-world cases of negative examples; and compared Rejection Miner to contemporary miners for various notations, finding an increase in accuracy and, in particular, output model simplicity.

References

1. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a Truly Declarative Service Flow Language. In: Web Services and Formal Methods. LNCS, vol. 4184, pp. 1–23. Springer (Sep 2006). https://doi.org/10.1007/11841197_1
2. van der Aalst, W.M.P.: Verification of Workflow Nets. In: ICATPN '97. pp. 407–426. Springer (1997)
3. Abu-Mostafa, Y.S., Magdon-Ismael, M., Lin, H.: Learning from Data: A Short Course. AML (2012)
4. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Polyvyanyy, A.: Split miner: Automated discovery of accurate and simple business process models from event logs. *Knowl. Inf. Syst.* **59**(2), 251–284 (May 2019)
5. Back, C.O., Slaats, T., Hildebrandt, T.T., Marquard, M.: Discover: Accurate & efficient discovery of declarative process models (2020)
6. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. In: OTM '12. LNCS, vol. 7565, pp. 305–322. Springer (Sep 2012)
7. Ciccio, C.D., Mecella, M.: A two-step fast algorithm for the automated discovery of declarative workflows. In: CIDM '13. pp. 135–142 (Apr 2013)
8. de Leoni, M., Maggi, F.M., van der Aalst, W.M.P.: An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. *Inf. Sys.* **47**, 258–277 (Jan 2015). <https://doi.org/10.1016/j.is.2013.12.005>

9. Debois, S., Hildebrandt, T.T., Slaats, T.: Hierarchical Declarative Modelling with Refinement and Sub-processes. In: BPM. LNCS, vol. 8659, pp. 18–33 (2014)
10. Debois, S., Hildebrandt, T.T., Slaats, T.: Replication, refinement & reachability: Complexity in dynamic condition-response graphs. *Acta Informatica* **55**(6), 489–520 (Sep 2018). <https://doi.org/10.1007/s00236-017-0303-8>
11. Debois, S., Slaats, T.: The Analysis of a Real Life Declarative Process. In: SSCI/CIDM '15. pp. 1374–1382. IEEE (2015)
12. Di Ciccio, C., Bernardi, M.L., Cimitile, M., Maggi, F.M.: Generating event logs through the simulation of declare models. In: *Work. on Ent. and Org. Modeling and Simulation*. pp. 20–36. Springer (2015)
13. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events. *Journal of Machine Learning Research* **10**, 1305–1340 (2009)
14. Hildebrandt, T., Mukkamala, R.R.: Declarative Event-Based Workflow as Distributed Dynamic Condition Response Graphs. In: PLACES '10. EPTCS, vol. 69, pp. 59–73 (2010). <https://doi.org/10.4204/EPTCS.69.5>
15. Hildebrandt, T.T., Mukkamala, R.R., Slaats, T.: Safe distribution of declarative processes. In: SEFM 2011. LNCS, vol. 7041, pp. 237–252. Springer (2011)
16. Khan, S.S., Madden, M.G.: A survey of recent trends in one class classification. In: *Irish conference on artificial intelligence and cognitive science*. pp. 188–197. Springer (2009)
17. Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Inducing declarative logic-based models from labeled traces. In: *Business Process Management*. pp. 344–359. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
18. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. In: *Application and Theory of Petri Nets and Concurrency*. pp. 311–329. LNCS, Springer (Jun 2013)
19. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient discovery of understandable declarative process models from event logs. In: *Advanced Information Systems Engineering*. pp. 270–285 (2012)
20. Nekrasaite, V., Parli, A.T., Back, C.O., Slaats, T.: Discovering Responsibilities with Dynamic Condition Response Graphs. In: *Advanced Information Systems Engineering*. pp. 595–610. LNCS, Springer (2019)
21. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: Full Support for Loosely-Structured Processes. In: EDOC '07. pp. 287–287 (Oct 2007)
22. Ponce de León, H., Nardelli, L., Carmona, J., vanden Broucke, S.K.: Incorporating negative information to process discovery of complex systems. *Information Sciences* **422**, 480–496 (2018)
23. Slaats, T.: Declarative and hybrid process discovery: Recent advances and open challenges. *Journal on Data Semantics* pp. 1–18 (2020)
24. Stone, M.: Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society: Series B (Methodological)* **36**(2), 111–133 (1974)
25. Tax, D.M.J.: One-class classification: Concept learning in the absence of counter-examples. (2002)
26. Tijs Slaats, Søren Debois: The Rejection Miner. <https://github.com/tslaats/RejectionMiner> (Jul 2020)
27. Tijs Slaats, Søren Debois, Christoffer Olling Back: Data Sets: DCR Solutions and Dreyers Foundation logs. <https://github.com/tslaats/EventLogs> (Jul 2020)
28. van der Aalst, W.: *Process Mining*. Springer Berlin Heidelberg (2016)
29. Witten, I., Frank, E., Hall, M., Pal, C.: *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 4th edn. (2016)

Process Mining via Hidden Markov Models: A Feasibility Study

Christoffer Olling Back

1 Introduction

Modern organizations increasingly generate and store enormous amounts of data related to their business processes via IT systems, in particular Enterprise Resource Planning (ERP) systems which not only document, but also automate many processes. This has given rise to so-called *process mining* techniques and affiliated research communities in both academia and industry. The field is closely related to data mining and machine learning, but differs somewhat, not only in its specific focus on process related data, but also its approach and historical roots in theoretical computer science and formal methods.

My aim here is to present a very preliminary investigation into the feasibility of building a bridge between probabilistic graphical models and process mining approaches. The motivation is, on the one hand, to bring a more principled probabilistic perspective to process mining, while incorporating the interpretability/explainability of high-level process modeling languages.

2 Approach

Process mining is usually characterized as having three main facets: process discovery, conformance checking, and process enhancement. The distinguishing aspect being the degree to which the process is understood/modeled a priori, and the goal of the process owner. The focus here will be primarily on process discovery, which is essentially an unsupervised learning task aimed at finding a process model which accurately describes the data.

Nearly all process mining algorithms assume a given process formalism, and usually deterministically map an event log to a model in that formalism. By far the most widespread modeling formalism are *imperative* languages which model specific flows a process can follow. These include Petri nets, BPMN, event-driven process chains, and process trees. *Declarative* modeling languages, such as Dynamic Condition Response Graphs, Linear Temporal Logic (LTL), Declare (a subset of LTL), π -calculus and other process algebras refrain from describing specific flows, instead describing a process by the rules (*relations*) that must be satisfied in a process - allowing any flow which satisfies these rules. Certain processes (or aspects of a process) are arguably best modeled with a certain

formalism, according to its degree of complexity/variability, and formalisms can also be combined into hybrid languages.

There are two key aspects of the approach presented here. First the observation that, regardless of modeling paradigm, any process model has an underlying state space with associated transitions between states. Second, the claim that maintaining a set of candidate hypotheses regarding process generating observed data, with associated confidence rankings, is a more powerful approach than deterministically generating one model as the presumed “true model”. If a process manager wants to visualize the process in a given formalism, it can then be generated from the underlying probabilistic graphical model over process model states. This aspect will be discussed further in the Future Work section.

3 Evaluation

As a first step in assessing the viability of this approach, we ask a very simple question: can a dynamic Bayesian network, specifically a hidden Markov model (HMM), learn the process model generated from a simple process model which allows loops and concurrency? To evaluate this, data was generated from a known model, and an HMM was trained and subsequently evaluated on an out-of-sample validation set.

Process models essentially describe which activities are able to be executed at any point in the execution of a process. That is, after some (potentially empty) prefix of activity occurrences, the process model will be in some state $s \in S$ where S denotes the model’s (finite, nonempty) state space. The resulting state after executing an activity $a \in \Sigma$ is determined by the transition function $\delta : S \times \Sigma \mapsto S$.

Most process modeling formalisms are not inherently probabilistic: they simply describe which sequences of events are *legal*. Arguably, this does nonetheless establish probabilistic bounds on legal sequences since illegal activities implicitly have a zero probability, and the probability of the legal outgoing transitions must sum to 1 unless the model is in an accepting state.

3.1 Data Generation

Consider the one-bounded Petri net in Figure 1. It only permits sequences beginning with a and ending with e . In between, b and c must be executed, but this can happen concurrently. Executing d is optional, but must subsequently be followed by c before the process can proceed to e .

We can also model the labeled transition system for this Petri net as in Figure 2. The initial state s_1 represents the marking shown in Figure 1 with one token in the first *place* node. State s_3 , for example represents the marking in which one token is in the place immediately following the b transition as well as one token in the place immediately prior to the c transition.

If we insist on viewing Figure 2 as a stochastic automaton, and denote by element A_{ij} the probability of moving from state i to state j , then its transition

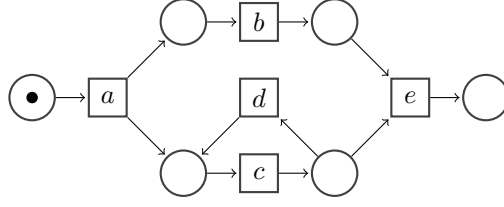


Figure 1: The Petri net used to generate artificial event logs

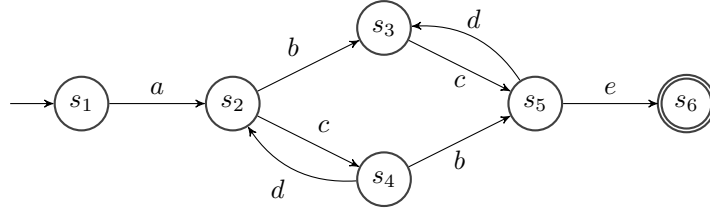


Figure 2: Labeled transition system (DFA) of Petri net in Figure 1

matrix is given by

$$\mathbf{A} = \begin{bmatrix} 0 & A_{12} & 0 & 0 & 0 & 0 \\ 0 & 0 & A_{23} & A_{24} & 0 & 0 \\ 0 & 0 & 0 & 0 & A_{35} & 0 \\ 0 & A_{42} & 0 & 0 & A_{45} & 0 \\ 0 & 0 & A_{53} & 0 & 0 & A_{56} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

where $A_{12}, A_{23}, A_{24}, A_{35}, A_{42}, A_{45}, A_{53}, A_{56}$ are all nonzero and less than or equal to 1. Of course, we also have the constraints that $\sum_j A_{ij} = 1$ which implies that $A_{12} = 1$ and imposes bounds for values sharing the same row, so that $A_{23} + A_{24} = 1$, for example.

Artificial event logs were generated using the model illustrated above by sampling transition probabilities for a given row of \mathbf{A} from a Beta distribution with $\alpha = \beta = 2$. Transitions were then followed by randomly selecting an outgoing transition according to the parametrization of \mathbf{A} . The initial state probability distribution, $\boldsymbol{\pi}^T = (1 \ 0 \ 0 \ 0 \ 0 \ 0)$, meant that the initial state was always s_1 . The generated models and sample of the event logs are included in the results tables.

3.2 Model Training and Selection

Several first-order hidden Markov model (HMM) were used for the present evaluation. A more sophisticated approach will be discussed in the Future Work

section, but due to technical and time limitations, an out-of-the-box solution from the `hmmlearn` Python package (formerly part of `pgmpy`) was used, which did not allow for easily building more structured HMMs. Models were fit using the Viterbi algorithm.

During the training phase, the model was fit to 100 sequences from the artificial event log, and evaluated on a separate set of 100 sequences from the same log. This was performed 10 times, with the model performing best on out-of-sample data being kept for final evaluation. This repeated restart is to account for the fact that the Expectation Maximization algorithm tends to get stuck in local optima. Other relevant model selection criteria include Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC), since they explicitly punish overly complex models.

This process was repeated for models with 3, 6, and 9 hidden states to evaluate the effect of the number of states on model performance. Ideally the model with same number of states as the true generating model would perform best, which it comes close to doing. In real-world data, this parameter would obviously need to be set automatically, since the true model is not known.

3.3 Negative examples

For the final evaluation, 100 negative examples were generated as well. This was done by a similar process as that used for generating positive examples, except that whenever an event was generated, the label was changed to an invalid label with a 10% probability. Any resulting legal sequences resulting from this process were thrown out. Note that simply generating random sequences is less useful, since by far most random sequences have a near zero likelihood (the space of random noise is vastly larger than the space of true “signals” or patterns). This approach ensured that the sequences generated were largely *feasible* according to model and not particularly dissimilar to the training data, despite being illegal.

In order to build a confusion matrix and evaluate the HMM as a binary classifier, a threshold was set on the likelihood of a sequence given the model, below which a sequence would be considered illegal. Results for several threshold values are reported. Any sequence with a likelihood below these values is classified as illegal.

4 Conclusion

Preliminary results seem to support the usefulness of the proposed approach. Even the shallow, first-order HMMs are able to fit the underlying distribution quite well, which can be seen in the comparison of the likelihood of sequences based on the true model with that based on the fitted model, as well as good results in several confusion matrices.

The fact that 3-state HMMs clearly perform more poorly is encouraging if the end goal is to uncover the actual state transition system of the generating process. The 9-state models perform slightly better in some cases, but the

improved performance would be diminished by a punishing factor for model complexity as in BIC or AIC.

5 Future Work

There are several aspects in which this approach can be extended, one of which is in using more sophisticated HMMs. Various data is often associated with events in event logs, such as who or what executed the activity. These should clearly be incorporated into a more sophisticated model. Also the ability to handle previously unseen activities would be an important addition. Fortunately, techniques from previous research on language modeling exist for this purpose.

One very important simplification that was made was to allow only one transition/label pair between states in the generating model. So if the process transitions from s_i to s_j , we can be sure of exactly which activity caused the transition. This is why the emission matrix for the true model is not reported in the results. In many process models this is not the case, and several activities can cause the same state transition. A probability distribution must therefore also be inferred, not just over state-transitions, but state-transition/label pairs. This might be done by adding a variable Y'_t between the hidden state X_t and the observed state Y_t , and adding an edge from Y'_t to X_{t+1} . This also allows the model to capture “noise”, or incorrectly recorded events, via the Y' to Y edge.

To achieve the original aim of eventually translating the state-transition system learned by an HMM into one or more candidate process models in a traditional formalism, requires “disentangling” the most likely paths through the transition system and translating this into a model. For example, if we consider a simple two state system, with just two activities, there are eight possible state-transition/activity combinations, as shown in Figure 3. This model can be interpreted as representing 16 different deterministic systems if we consider all of the different ways state-transition/label combinations can be made when determinism is enforced (see Figure 4). In order to give preference to (nearly) deterministic models, one can for example use MAP estimation with a minimum entropy prior.

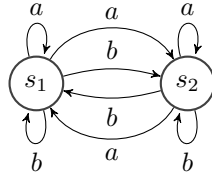


Figure 3: A fully connected state-transition system with two activities. In this form, it either represents a nondeterministic system, since executing one activity leads to two different states, or a probabilistic system if we enforce a probability measure on outgoing transitions with the same label.

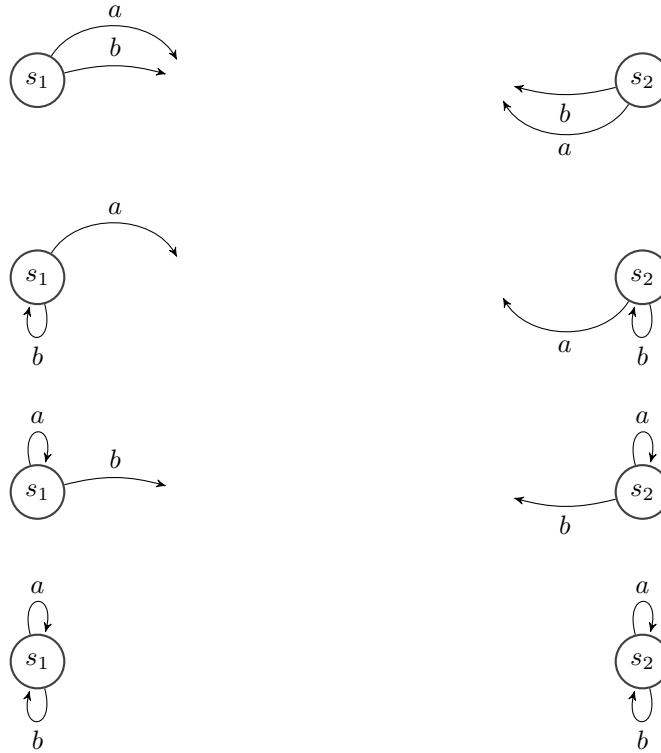


Figure 4: All of the different outgoing transition/label combinations for each state if determinism is enforced. The different combinations of the two states result in 16 different possible systems.

TRUE MODEL (Run 1)

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & .75 & .25 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & .16 & 0 & 0 & .84 & 0 \\ 0 & 0 & .2 & 0 & 0 & .8 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Sample of Event Log

acbdce
abce
acdcdcbdcddce
abcdce
acbe
abcdcdce

3-state HMM

6-state HMM

9-state HMM

$$\boldsymbol{\pi}^T = (0 \quad 0 \quad 1)$$

$$\text{Start Priors} \\ (0 \quad 0 \quad 0 \quad .04 \quad 0 \quad .96)$$

$$(0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0)$$

Transition Matrices

$$\mathbf{A} = \begin{pmatrix} 0 & .36 & .64 \\ .93 & .07 & 0 \\ .26 & .74 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ .56 & 0 & 0 & 0 & .44 & 0 \\ 0 & .13 & 0 & 0 & .87 & 0 \\ 0 & .55 & 0 & 0 & .45 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & .97 & 0 & 0 & .03 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ .79 & 0 & 0 & 0 & .21 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & .3 & 0 & .7 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .2 & .8 & 0 & 0 & 0 & 0 & 0 \\ 0 & .24 & 0 & .74 & 0 & 0 & 0 & .02 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ .99 & 0 & 0 & 0 & .01 & 0 & 0 & 0 & 0 \\ 0 & 0 & .97 & 0 & 0 & 0 & 0 & .03 & 0 \end{pmatrix}$$

Emission Matrices

$$\boldsymbol{\theta} = \begin{pmatrix} 0 & 0 & .91 & 0 & .09 \\ 0 & .74 & 0 & .26 & 0 \\ .53 & 0 & 0 & .01 & .45 \end{pmatrix}$$

$$\begin{pmatrix} 0 & .39 & .61 & 0 & 0 \\ 0 & .78 & .22 & 0 & 0 \\ 0 & 0 & 0 & .28 & .72 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & .33 & .67 \end{pmatrix}$$

LIKELIHOOD COMPARISON



CONFUSION MATRICES (Legal/Illegal)

$p < 0.001$		L	I	$p < 0.001$		L	I	$p < 0.001$		L	I
	L	100	54		L	100	13		L	100	2
	I	0	46		I	0	87		I	0	98
$p < 0.01$		L	I	$p < 0.01$		L	I	$p < 0.01$		L	I
	L	91	29		L	96	13		L	100	2
	I	9	71		I	4	87		I	0	98
$p < 0.1$		L	I	$p < 0.1$		L	I	$p < 0.1$		L	I
	L	70	10		L	77	8		L	94	0
	I	30	90		I	23	92		I	6	100

TRUE MODEL (Run 2)

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & .51 & .49 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & .34 & 0 & 0 & .66 & 0 \\ 0 & 0 & .74 & 0 & 0 & .26 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Sample of Event Log

abce
acdcdbcdbdce
abcdcdcdcdcdce
acdcdbcdbdcdcdce
abcdcdcdcdcdcdcdcdcdcdcdce
acbdce
abcdce

3-state HMM

6-state HMM

9-state HMM

$$\pi^T = (0 \quad 0 \quad 1)$$

$$\text{Start Priors} \\ (0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0)$$

$$(0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0)$$

Transition Matrices

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 \\ .87 & .13 & 0 \\ .49 & .51 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & .62 & 0 & .38 & 0 \\ 0 & 0 & .51 & 0 & .49 & 0 \\ 0 & 0 & 0 & .67 & 0 & .33 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ .44 & 0 & .56 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & .52 & .48 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & .23 & 0 & 0 & .77 & 0 & 0 \\ 0 & 0 & 0 & .17 & 0 & 0 & .83 & 0 & 0 \\ 0 & .49 & .51 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & .5 & 0 & 0 & .49 \\ 0 & 0 & 0 & .06 & 0 & 0 & .94 & 0 & 0 \\ 0 & .52 & .48 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & .62 & 0 & 0 & .38 \\ .56 & 0 & 0 & 0 & 0 & 0 & 0 & .44 & 0 \end{pmatrix}$$

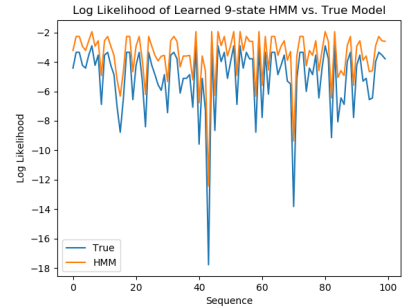
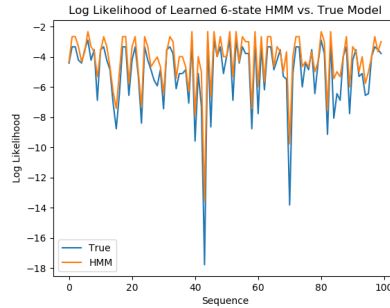
Emission Matrices

$$\theta = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & .21 & 0 & .58 & .21 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & .72 & .28 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & .85 & .15 \\ 0 & 0 & 0 & .58 & .42 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

LIKELIHOOD COMPARISON



CONFUSION MATRICES (Legal/Illegal)

$p < 0.001$		L	I
	L	87	15
	I	13	85
$p < 0.01$		L	I
	L	52	6
	I	48	94
$p < 0.1$		L	I
	L	0	1
	I	100	99

		L	I
	L	99	5
	I	1	95
		L	I
	L	92	4
	I	8	96
		L	I
	L	31	0
	I	69	100

		L	I
	L	99	5
	I	1	95
		L	I
	L	97	4
	I	3	96
		L	I
	L	56	0
	I	44	100

TRUE MODEL (Run 3)

Sample of Event Log

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & .94 & .06 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & .46 & 0 & 0 & .54 & 0 \\ 0 & 0 & .87 & 0 & 0 & .13 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

abcbcdcdcdcdcdcdcdcdcdcdcdcdcdcdce
acdbce
abcbcdcdcdcdcdcdcdcdcdcdcdcdcdce
abcbcdcdcdcdcdcdcdcdcdcdcdcdcdce
abcbcdcdce
abcbcdcdcdcdcdcdcdcdcdcdcdcdcdce
abcbcdcdcdcdcdcdcdcdcdcdcdcdcdcdce

3-state HMM

6-state HMM

9-state HMM

$$\pi^T = (0 \quad 1 \quad 0)$$

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

$$(0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0)$$

Transition Matrices

$$A = \begin{pmatrix} .01 & 0 & .99 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ .72 & 0 & 0 & 0 & .01 & .27 \\ 0 & .97 & 0 & .03 & 0 & 0 \\ 0 & .18 & 0 & 0 & .82 & 0 \\ .69 & 0 & 0 & 0 & 0 & .3 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & .3 & 0 & 0 & 0 & .3 & 0 & 0 & .4 \\ .26 & 0 & 0 & .13 & 0 & 0 & .61 & 0 & 0 \\ .26 & 0 & 0 & .13 & 0 & .01 & .6 & 0 & 0 \\ 0 & .31 & 0 & 0 & 0 & .3 & 0 & 0 & .39 \\ 0 & 0 & .97 & 0 & 0 & 0 & 0 & .03 & 0 \\ .21 & 0 & 0 & .1 & 0 & 0 & .69 & 0 & 0 \\ 0 & .26 & 0 & 0 & 0 & .26 & 0 & 0 & .48 \\ 0 & 0 & .6 & 0 & 0 & 0 & 0 & .4 & 0 \\ .16 & 0 & 0 & .07 & 0 & 0 & .77 & 0 & 0 \end{pmatrix}$$

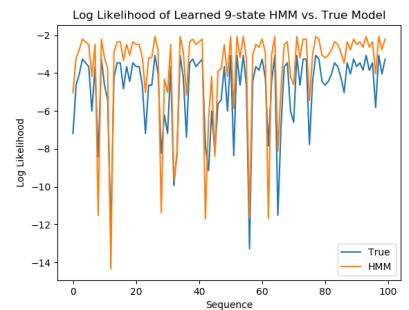
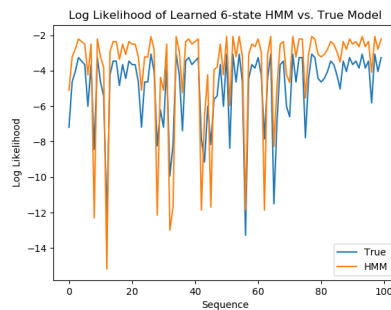
Emission Matrices

$$\theta = \begin{pmatrix} 0 & .97 & .03 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & .5 & .44 & .07 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & .87 & .13 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & .77 & .23 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & .96 & .04 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & .6 & .4 & 0 \\ 0 & 0 & 0 & .87 & .13 \end{pmatrix}$$

LIKELIHOOD COMPARISON



CONFUSION MATRICES (Legal/Illegal)

$p < 0.001$		L	I
	L	46	6
	I	54	94
$p < 0.01$		L	I
	L	31	1
	I	69	99
$p < 0.1$		L	I
	L	0	0
	I	100	100

		L	I
	L	91	11
	I	9	89
		L	I
	L	90	5
	I	10	95
		L	I
	L	70	0
	I	30	100

		L	I
	L	94	2
	I	6	98
		L	I
	L	90	2
	I	10	98
		L	I
	L	70	0
	I	30	100

Towards Inference of Resource Dependency Grammars from Event Streams^{*}

Christoffer Olling Back¹[0000–0001–7998–7167]

Dept. of Computer Science, University of Copenhagen, Denmark
back@di.ku.dk

Abstract. Many approaches to mining process-related data focus on discovering temporal control-flow patterns between actions, usually assuming a given trace identifier. In some cases, trace identifiers may be unavailable and in any case, event logs often generated by the multiple interconnected processes. This paper presents a framework for discovering such interactions based on the notion of a conditional control-flow. We present a simple, context-free resource dependency grammar coupled with corresponding temporal control-flow models. We outline methods to ground these formula as transition systems and how these can be integrated into factorial hidden Markov models to facilitate probabilistic reasoning. We present some preliminary approaches to grammar and parameter inference for a restricted fragment of the presented grammar and outline approaches to building more sophisticated models.

Keywords: Hybrid Process Mining · Resource Dependency · Factorial Hidden Markov Model

1 Introduction

Recent years have seen growing interest in mining process-related data, for example from enterprise resource planning (ERP) and workflow/case management systems. This has led to the development of so-called process mining techniques for the discovery from event logs of generative models that are specifically process-oriented and have proper execution semantics. One nearly universal assumption of such approaches is the existence of a trace identifier that partitions events into sequences representing instances of a process. Furthermore, nearly all process mining algorithms focus narrowly on an end-to-end control flow based on temporal orderings of events. Real-world data sometimes lacks such a trace identifier, and in any case, may contain other attributes that provide another informative partitioning of events: the relations between these attributes is often just as interesting as temporal ordering. In many cases event logs will contain overlapping processes, for example patients which follow a certain flow and are

^{*} This work is supported by the Hybrid Business Process Management Technologies project (DFF-6111-00337) funded by the Danish Council for Independent Research

treated by hospital staff who follow their own workflow. Exploring how different views of a process overlap can reveal interactions and dependencies among different resources/attributes related to a process.

We present a framework for discovering and parametrizing models of subprocess interaction from event streams without trace identifiers. The main thrust of the approach is the notion of a *conditional control-flow*: a temporal structure which is induced events by constraints on the relations between event attributes. We show how these models can then be integrated with a probabilistic model, to which standard parameter inference techniques can be applied.

The general idea of conditional control flows is broad and amenable to more solutions. In Section 3, we present a straightforward, proof-of-concept solutions based on a simple context-free grammar for capturing constraints on attributes, along with control-flow models based on simple directly follows graphs. In Section 4.1 we describe the structure learning and parameter inference problem and present a simple, clustering-based method for the former, and outline standard techniques for the latter. In Section 6 we conclude and present a several avenues for extended the framework with more sophisticated modeling constructs and inference techniques. We begin by first summarizing related work in Section 2.

2 Related Work

Process mining centers around *events* and traditionally assumes a well-formed event log partitioning the set of events, first into separate processes and second into separate process instances (i.e. a trace, associated with a case identifier). The assumption is that a process instance represents an enactment of a repeatable process template with a clear beginning and completion, with the case identifier defining the most meaningful perspective, e.g. a patient's treatment or a customers purchase [11].

Aside from events not always being organized into well-formed event logs, events may not eve be given a central role in traditional data schemas, and sometimes produced only as a by-product. Extracting and correlating events is not always straightforward, and often system specific. A recent literature review summarizes various approaches to this challenge [5]. In [10], for example, the authors use database redo logs to do so in a system-independent manner. Events may even need to be extracted from disparate and incongruous auxiliary sources lacking a common case identifier [8, 3] or even buried in digital communications [4]. Adding to this challenge is the fact that data increasingly needs to be handled as a constant stream rather than a static, complete dataset [12].

Combining formal grammars, such as stochastic context-free grammar, with Markov models for analysis of sequential data has received a good deal of attention, largely in the context of text and language processing. The more structured models resulting from integrating a formal grammar with simple Markov models (e.g. bigram model) significantly lowers the entropy (complexity) of the model while adding only a marginal number of additional parameters [6, 9].

3 Modeling

The fundamental principle underlying our approach is that event attributes can be used to induce temporal orderings on an event stream. In other words, by selecting events according to a criteria, for example those events which share the same value for an attribute, we can view the temporal orderings of the selected events independently of all other events.

An important notion in our approach is that of a control-flow model coupled with attribute constraints defining when the control-flow restrictions are applicable. The control-flow could be modeled by any modeling formalism, such as Petri nets or even LTL¹ formula.

For simplicity, we will restrict our scope to simple partial orderings, denoted by $a \succ b$ (i.e. a precedes and is succeeded by b).

Example For illustration, we introduce the following running example. Consider a simple scenario in which two agents, *Alice* and *Bob*, share one resource, *Machine*. In this case, *Alice* and *Bob* have the same restrictions on the control-flow of actions: $a \succ b$ and $c \succ d$. For *Machine* and a given agent, the restriction $b \succ c$ must hold. See Table 2 for an example event log fulfilling these conditions and Figure 1 for a graphical representation.

We can present the model described above as shown in Table 1. We will refer to these control-flow models subjected to attribute constraints as *conditional control-flow* formulas.

Table 1. Example model

Control-flow	Subject to
$a \succ b$ and $c \succ d$	$a.agent = b.agent \wedge$ $b.agent = c.agent \wedge$ $c.agent = d.agent \wedge agent \in \{Alice, Bob\}$
$b \succ c$	$b.agent = c.agent \wedge$ $b.machine = c.machine \wedge machine \in \{Machine\}$

Interleaving and Handshaking Clearly the two processes described in our example are not fully independent since they both impose constraints on actions b and c . To capture this, we introduce two composition operators for conditional control-flow formulas: the *interleaving* operator $|||$ and the *handshake* operator $||_H$.

The interleaving operator simply indicates that two formulas capture completely independent processes that can be executed asynchronously. The handshake operator indicates that two processes must be synchronized over a set of shared actions, H . This is the case in our example where $H = \{b, c\}$. In this

¹ Linear Temporal Logic

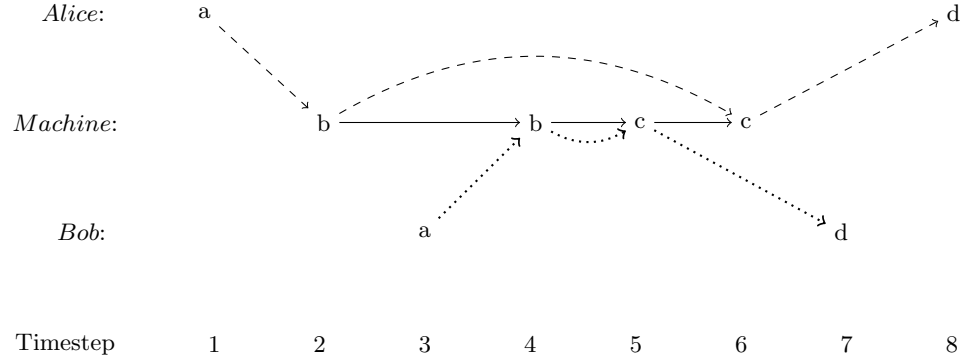


Fig. 1. Informal representation of event log in Table 2 which is an instance of the process described in Table 1. Workers *Alice* and *Bob* must perform activity *a* before *b*, and activity *c* before *d*. The *Machine* resource must perform activity *b* before *c* when *b* and *c* are associated with the same *Agent*. Via the handshaking operator for shared events, the full set of constraints on events w.r.t. to resources/attributes is derived.

Table 2. Event log.

Timestamp	Action	Agent	Resource
1	<i>a</i>	<i>Alice</i>	-
2	<i>b</i>	<i>Alice</i>	<i>Machine</i>
3	<i>a</i>	<i>Bob</i>	-
4	<i>b</i>	<i>Bob</i>	<i>Machine</i>
5	<i>c</i>	<i>Bob</i>	<i>Machine</i>
6	<i>c</i>	<i>Alice</i>	<i>Machine</i>
7	<i>d</i>	<i>Alice</i>	-
8	<i>d</i>	<i>Bob</i>	-

case, the two processes must execute actions in H *simultaneously*, meaning one process may need to wait for the other to be ready before executing an action.

3.1 Formal Grammar

Our modeling approach can be captured as a context-free grammar. Recall that a context free grammar is a 4-tuple $G = (V, \Sigma, R, V_0)$ consisting of a set of *nonterminal* symbols V , the set of *terminal* symbols Σ , the set of rewrite/production rules R (below) and the start symbol (in V).

The production rules R for the proposed grammar are as follows:

1. $P \rightarrow P \parallel P$
2. $P \rightarrow P \parallel_H P$
3. $P \rightarrow F \wedge C$
4. $F \rightarrow F \vee F \mid A \succ A$
5. $C \rightarrow C \wedge C \mid C \wedge E \mid E \wedge E$
6. $E \rightarrow A.T = A.T \wedge T \in D^T$
7. $A \rightarrow action_1 \mid action_2 \mid \dots \mid action_n$
8. $T \rightarrow attribute_1 \mid attribute_2 \mid \dots \mid attribute_m$
9. $D^T \rightarrow D_1^T \mid D_2^T \mid \dots \mid D_k^T$

Where $D_i^T \in 2^{Dom(T)}$, i.e. the powerset of values for attribute T .

Fig. 2. Production rules R for the resource dependency grammar.

As rules 1-3 indicate, at the root of the grammar's parse tree is either an interleaving (1) of parallel processes, a handshake synchronization (2) of interacting processes, or a process (3) consisting of the conjunction of a control-flow (F) component conditioned on attribute criteria (C).

Rule 4 represents the control-flow aspect of our model. In this case, it is restricted to conjunctions of simple follows relations. For more elaborate control-flow models, this is the production rule which should be modified.

Rules 5-9 define the constraints on attributes. As rule 5 indicates, these are conjunctions of the equality relations defined by rule 6. Rule 6 states that two actions must share the same value for some attribute, and that that value should be a member of D^T (some subset of the domain of attribute T). Rules 7-9 map the nonterminal action, attribute, and domain symbols to terminal symbols in the set of actions, attributes, and attribute domains subsets, respectively.

In summary, our grammar is given by

$$G = (\{P, F, C, E, A, T, 2^{Dom(T)}\}, \Sigma \cup A \cup D, R, P)$$

with R as defined in Figure 2.

Example We will now formalize our running example using the grammar as laid out above. Add to the example a second resource, *truck*, and two new actions *E* and *F*. Note that now $Dom(agent) = \{Alice, Bob\}$ and $Dom(resource) = \{Machine, Truck\}$. We will build the full formula from the bottom up.

For simplicity we start with *Truck* which is associated with control-flow: $E \succ F$. The powerset of the resource domain is

$$2^{Dom(resource)} = \{\emptyset, \{Machine\}, \{Truck\}, \{Machine, Truck\}\}$$

. In this case, $D^{resource}$ will yield $\{Truck\}$ via rule 9. Using rules 3-9 we can capture control-flow subject to the corresponding attribute constraint as:

$$P_1 : (e \succ f) \wedge (e.resource = f.resource \wedge resource \in \{Truck\})$$

Similarly we can capture the control-flow associated with *Machine* as

$$P_2 : (b \succ c) \wedge (b.resource = c.resource \wedge resource \in \{Machine\})$$

Alice and *Bob* are associated with the control flow: $A \succ B \vee C \succ D$. The powerset of the agent domain is $2^{Dom(agent)} = \{\emptyset, \{Alice\}, \{Bob\}, \{Alice, Bob\}\}$. In this case, $D^{agent} = \{Alice, Bob\}$ (rule 9). Using rules 3-9 we can capture the constraint on

$$P_3 : (a \succ b \vee c \succ d) \wedge (a.agent = b.agent \wedge \\ b.agent = c.agent \wedge \\ c.agent = d.agent \wedge agent \in \{Alice, Bob\})$$

Using rules 1 and 2, we can combine the subprocesses defined above into a single model:

$$P_1 \parallel (P_2 \parallel_H P_3)$$

Where $H = \{b, c\}$. Since P_1 does not share any actions with other subprocesses, it relates to them in parallel via the interleaving operator, whereas P_2 and P_3 must interact via the handshake operator for shared actions *b* and *c*.

3.2 Operational Semantics

In the previous section, we outlined the semantics of the proposed approach. Now we will illustrate how the grammar described above can be grounded and translated to a concrete transition system. We restrict our focus to the simple control-flow model presented earlier, but more complex models can be translated in a similar manner.

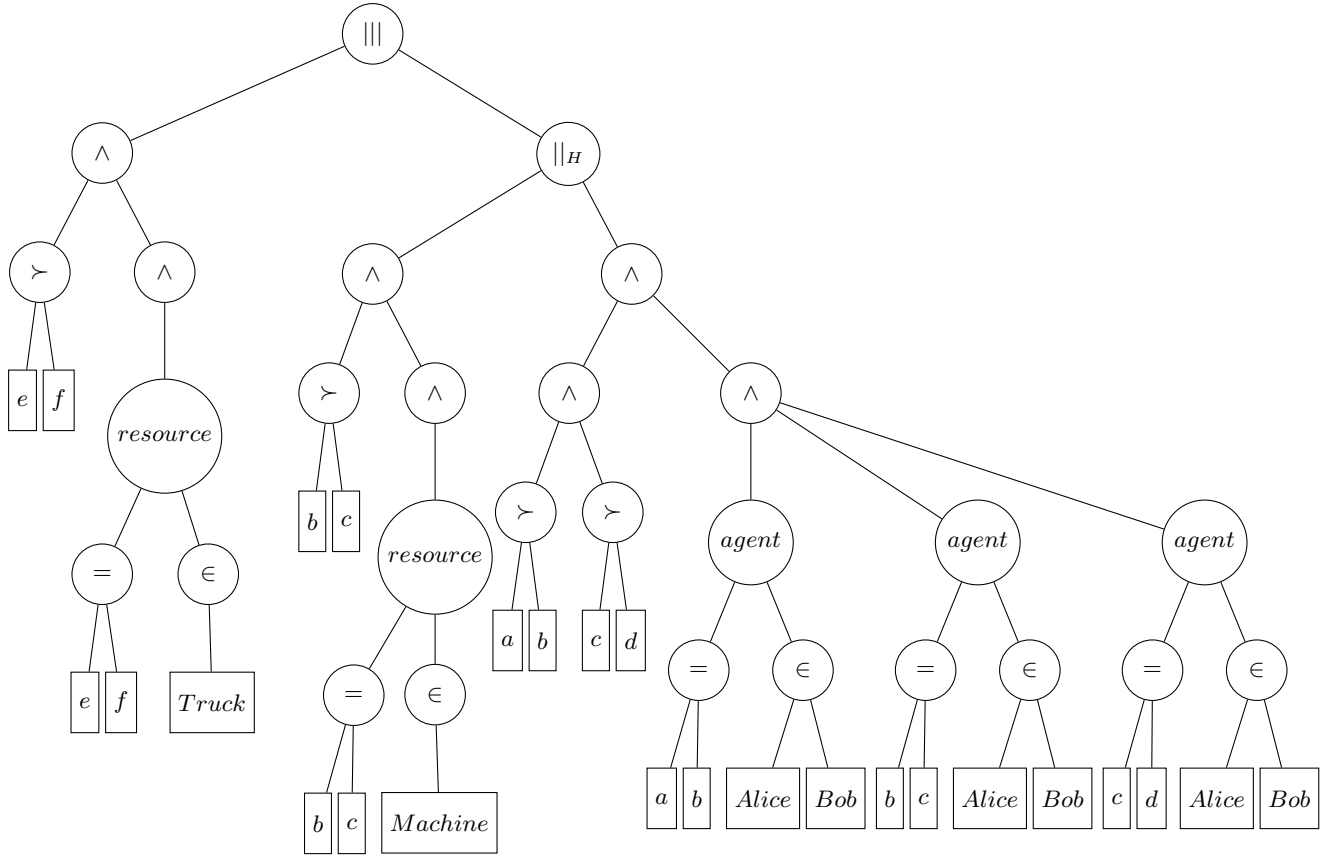


Fig. 3. Parse tree of $P_1 ||| (P_2 ||_H P_3)$

Recall the definition of a transition system TS as a tuple $(S, A, \rightarrow, I, AP, L)$ for a set of states S , actions A , transitions $\rightarrow \subseteq S \times A \times S$, atomic propositions AP and labeling function $L : S \rightarrow 2^{AP}$. The process of grounding interpretations of models and subsequently applying handshake and interleaving operators will result in several independent transition systems.

Start with the conjunction of control-flow F and attribute constraint C (rule 3). For every interpretation $I(D_i^T)$ s.t. the attribute constraint C is satisfied, i.e. $C \models I(D_i^T)$, construct a simple transition system as follows:

- Create a state S_a for every action a in A
- For every follows relation $a \prec b$ in F , add a transition (S_a, b, S_b)
- For attribute constraints $a.p = b.p$ and $c.q = d.q$ (rule 6) in C where p and q are assigned according to interpretation $I(D_i^T)$, add atomic proposition $P = p$ to states S_a, S_b and $Q = q$ to states S_c, S_d .
- To those states with no incoming edges, add a start state and corresponding edge. If all states have incoming edges, add one start state with an outgoing edge to all other states.
- If the resulting transition system consists of more than one connected component, perform a parallel operation ($|||$) between all components.

Example To illustrate, consider a modified version of P_2 where the constraint $B.agent = C.agent$ has been removed. This allows interpretations involving both agent values. The possible interpretations are:

$A.agent = Alice, B.agent = Alice, C.agent = Alice, D.agent = Alice$
 $A.agent = Alice, B.agent = Alice, C.agent = Bob, D.agent = Bob$
 $A.agent = Bob, B.agent = Bob, C.agent = Alice, D.agent = Alice$
 $A.agent = Bob, B.agent = Bob, C.agent = Bob, D.agent = Bob$

The transition system for the second interpretation is shown in Figure 3.2.

Handshaking The handshaking operator $P_1 ||_H P_2$ requires that both processes execute the actions in H synchronously. This can be captured as a single transition system according to the following transition rule in (2). Let $s_1 \xrightarrow{\alpha} s'_1$ denote the transition from state s_1 to s'_1 via action α in P_1 , and (s_1, s_2) denote the state resulting from the conjunction of s_1 and s_2 .

For $\alpha \in H$

$$\frac{s_1 \xrightarrow{\alpha} s'_1 \wedge s_2 \xrightarrow{\alpha} s'_2}{(s_1, s_2) \xrightarrow{\alpha} (s'_1, s'_2)} \quad (1)$$

This operation can be extended to multiple processes $P_1 || \dots || P_n$ as follows, where $H_{i,j}$ denotes the shared actions between any two processes:

For $\alpha \in H_{i,j}$ and $0 < i < j \leq n$:

$$\frac{s_i \xrightarrow{\alpha} s'_i \wedge s_j \xrightarrow{\alpha} s'_j}{(s_1, \dots, s_i, \dots, s_j, \dots, s_n) \xrightarrow{\alpha} (s'_1, \dots, s'_i, \dots, s'_j, \dots, s'_n)} \quad (2)$$

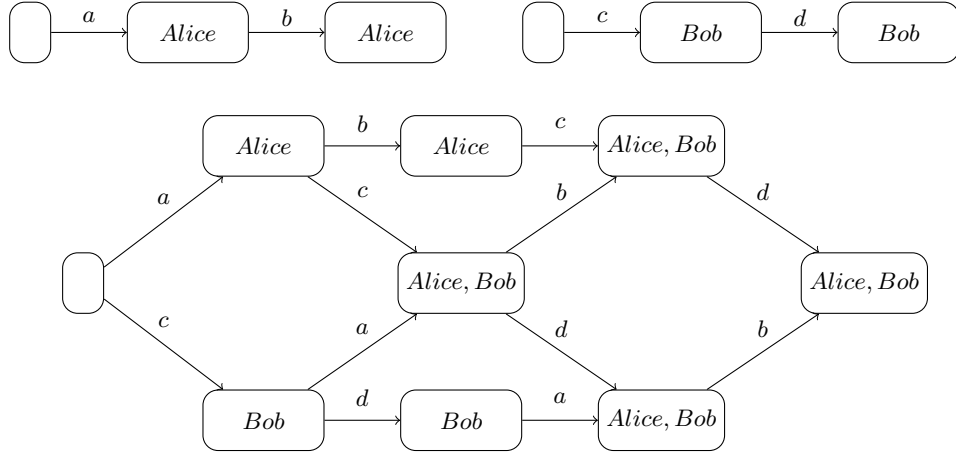


Fig. 4. Grounded transition system for the second interpretation of P_3 in Table 3.2. The formula consists of two non-overlapping parallel control flows ($a \succ b$ and $c \succ d$), resulting in two transition systems that can be executed in parallel (above). The equivalent system (below) resulting from applying the interleaving operator $|||$. The latter illustrates the issue arising with the standard definition of the labeling function after interleaving: atomic propositions *Alice* and *Bob* remain associated with states in a counter-intuitive fashion. We address this in Section 3.4

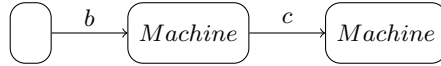


Fig. 5. Grounded transition system for the only possible interpretation of P_2

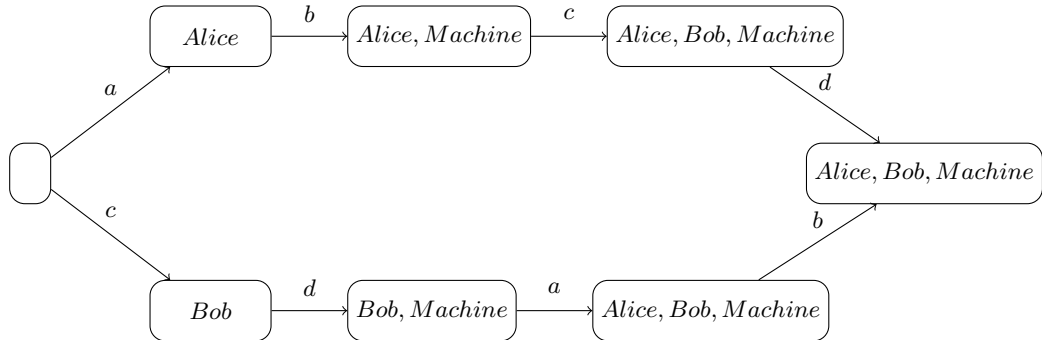


Fig. 6. Grounded transition system for $P_2 ||_H P_3$ under the interpretation shown above. Some states have become unreachable due to the synchronization criteria embodied in handshake transition dynamics (see (2)). As in Figure 3.2 we see the unintended propagation of atomic propositions.

Interleaving For the interleaving operator $P_1 ||| \dots ||| P_n$, as well as actions outside of H for the handshake operator, the resulting transitions obviously occur independently.

That is, for $\alpha \notin H$ for $||_H$, or $\alpha \in \bigcup Act(P_i)$ for $|||$

$$\frac{s_i \xrightarrow{\alpha}_i s'_i}{(s_1, \dots, s'_i, \dots, s_n) \xrightarrow{\alpha} (s'_1, \dots, s'_i, \dots, s'_j, \dots, s'_n)} \quad (3)$$

3.3 Labeling of Atomic Propositions

When applying the $||_H$ and $|||$ operators, the standard approach is simply to associate the atomic propositions of the new state with the union of the atomic propositions of the constituent states. That is,

$$L(s^{(1)}, \dots, s^{(M)}) = \bigcup_m L(s^{(m)})$$

where L is the labeling function from states to the powerset of atomic propositions.

For our purposes, using this approach directly leads to a slightly undesired result as illustrated in Figure 3.2 and 6. To see this, consider the following execution trace in our example where we indicate the state of the process associated with the *agent* and *resource* as s^a and s^r , respectively:

<i>action</i>	<i>agent</i>	<i>resource</i>	<i>state</i>
<i>a</i>	<i>Alice</i>	-	(s_1^a, s_0^r)
<i>b</i>	<i>Alice</i>	<i>Machine</i>	(s_2^a, s_1^r)
<i>c</i>	<i>Alice, Bob</i>	<i>Machine</i>	(s_3^a, s_2^r)
<i>d</i>	<i>Alice, Bob</i>	<i>Machine</i>	(s_4^a, s_2^r)

The problem here is in the third and fourth steps. First, *Alice* continues to be affiliated with the process, even though she is only involved in actions *a* and *b* according to our grammar. Then in the fourth step, *Machine* ends up being included in the execution of *d* as a result of the interleaving of states s_4^a and s_2^r . This seems misleading since the two processes (P_2 and P_3) have “parted ways” at this point. We can address this by applying a kind of “filter” that takes into account how a state was reached. This corresponds to expanding each state into multiple states, one for each incoming action:

As we will see in the following section, we can capture this in a straightforward manner by expanding the state transition matrix as well as the emission matrix in a corresponding manner.

3.4 Translation to Factorial Hidden Markov Model

Using the methods above for grounding formulas in the proposed grammar, we obtain a set of independent transition systems. The independence stems either

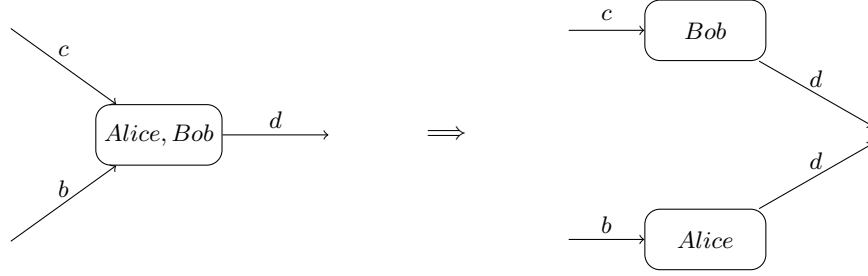


Fig. 7. Adjusted atomic proposition labeling by means state expansion w.r.t incoming actions and the corresponding grammar.

from separate interpretations as described in 3.2, or from the parallel operator $|||$.

These transition systems can now be easily integrated into a probabilistic model as the hidden state chains of a *factorial hidden Markov model*: a natural model for sequential data like event streams. Hidden Markov models (HMM) assume that a sequence of observed variables $\mathbf{o}_1, \dots, \mathbf{o}_t, \dots, \mathbf{o}_T$ are conditionally independent given a corresponding set of $s_1, \dots, s_t, \dots, s_T$ as depicted in Figure 8. Formally, the joint probability of the observed sequence is given by:

$$\mathbb{P}(\mathbf{o}_1, \dots, \mathbf{o}_t, \dots, \mathbf{o}_T) = \mathbb{P}(\mathbf{o}_1 | s_1) \mathbb{P}(s_1) \prod_{t=2}^T \mathbb{P}(\mathbf{o}_t | s_t) \mathbb{P}(s_t | s_{t-1}) \quad (4)$$

The temporal dynamics are thus captured entirely by $\mathbb{P}(s_t | s_{t-1})$, i.e. the probability of transitioning to state s_t from state s_{t-1} . Thus, the chain of states fulfills the Markov property and observed variable \mathbf{o}_t is solely a function of the state of the model at time t .

Factorial hidden Markov models (FHMM) extend the HMM model with multiple chains of hidden states, rather than just one. A FHMM with M state chains can be modeled by a HMM by setting the domain of the state variable to the Cartesian product of the domains of all M state variable in the FHMM such that $\mathbf{s} = (s^{(1)}, \dots, s^{(m)}, \dots, s^{(M)})$. However, this results in a unnecessarily complex model that fails to take advantage of the independencies between the state variables.

To see this, let $K^{(m)}$ denote the number of states of state variable $s^{(m)}$. If we let $K^{(m)} = K$ for all $s^{(m)}$, the number of possible state combinations is K^M or $\prod_{m=1}^M K^{(m)}$ for variable $K^{(m)}$ and the size of the transition matrix will be $K^M \times K^M$. By separating state chains, we can capture state transitions with M separate $K \times K$ matrices. This reduces the complexity of the inference process and avoids finding spurious correlations between states.

In a FHMM, the outcome of the observed variables is determined according to a function which aggregates the influence of each hidden state variable to the outcome of the observed variable. In a continuous valued model, this might

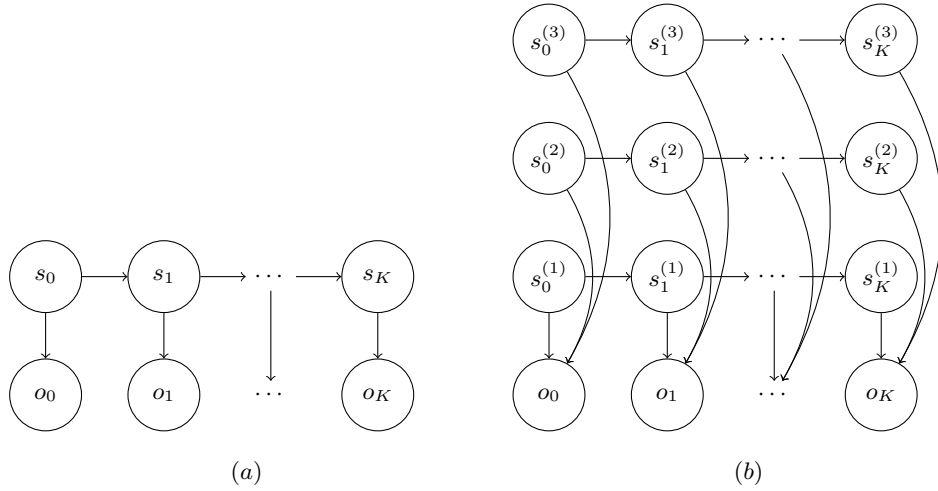


Fig. 8. (a) Hidden Markov model (HMM) and (b) a factorial hidden Markov model (FHMM)

simply be a weighted linear combination of the chosen distributions (usually Gaussian). In our model, we can simply marginalize over all states that may have generated \mathbf{o}_t .

It also makes sense here to include a weighting, $\mathbb{P}(m)$ - i.e. a prior - on the state variables to express a belief regarding which subprocess - i.e. which state chain - is *a priori* likely to be in play. In the case of many candidate processes, the prior distribution can also serve as ranking mechanism to aid in the presentation and understandability of the inferred model. Assuming $\mathbb{P}(m)$ is stationary, we have:

$$\mathbb{P}(\mathbf{o}_t | s_t) = \sum_m \mathbb{P}(\mathbf{o}_t | s_t^{(m)}) \mathbb{P}(s_t^{(m)} | s_{t-1}^{(m)}) \mathbb{P}(m) \quad (5)$$

Emission Probabilities We noted in the previous section that we have an issue with states being associated with unexpected atomic propositions and that this can be addressed by expanding states w.r.t incoming edges (see Figure 7). Practically, we can capture this using an extended-width transition matrix and a corresponding extended-height emission matrix - extended by the number of possible actions. This will give a $K \times K|\Sigma|$ transition matrix, a $K|\Sigma| \times |\Sigma|$ emission matrix for observed actions and $K|\Sigma| \times |Dom(T)|$ emission matrices for the various attributes T . For example, a simple system with 3 states, 2 actions, and one attribute with 4 values would be given by:

$$\mathbb{P}(s_t^{(m)'} | s_{t-1}^{(m)}) = \begin{matrix} & s_1^a & s_1^b & s_2^a & s_2^b & s_3^a & s_3^b \\ \begin{matrix} s_1 \\ s_2 \\ s_3 \end{matrix} & \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} & p_{15} & p_{16} \\ p_{21} & p_{22} & p_{23} & p_{24} & p_{25} & p_{26} \\ p_{31} & p_{32} & p_{33} & p_{34} & p_{35} & p_{36} \end{pmatrix} \end{matrix}$$

$$\mathbb{P}(o_t^{act} | s_t^{(m)'}) = \begin{matrix} & a & b \\ \begin{matrix} s_1^a \\ s_1^b \\ s_2^a \\ s_2^b \\ s_3^a \\ s_3^b \end{matrix} & \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \end{matrix}$$

$$\mathbb{P}(o_t^{att} | s_t^{(m)'}) = \begin{matrix} & \alpha & \beta & \gamma & \delta \\ \begin{matrix} s_1^a \\ s_1^b \\ s_2^a \\ s_2^b \\ s_3^a \\ s_3^b \end{matrix} & \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44} \\ p_{51} & p_{52} & p_{53} & p_{54} \\ p_{61} & p_{62} & p_{63} & p_{64} \end{pmatrix} \end{matrix}$$

The emission matrix associated with the action leading to state, $\mathbb{P}(o_t^{act} | s_t^{(m)'})$, can reasonably be set to be deterministic as shown.

4 Inference

In Section 3 we outlined the proposed model, and how to translate the proposed grammar into a set of independent transition systems which can then be incorporated into a factorial hidden Markov model. Now we must address how to discover such a model from data. Assuming no input from a domain expert or modeler, the following aspects of the model are unknown:

- The grammatical formulas describing control flows conditioned on resource constraints
- The parameters of the probabilistic model:
 - $\theta_T^{(m)}$: the transition matrix defining state transition probabilities for state chain (m) , that is $\mathbb{P}(s_t | s_{t-1})$.
 - $\theta_E^{(m,x)}$: the emission matrix for defining the probability of observing values of attribute x for each state in state chain m .
 - $\mathbb{P}(s_1^{(m)})$: the probability over initial states in state-chain m .
 - $\pi^{(m)}$: the prior probability of state-chain m being the source of an observation.

A standard HMM shares the first three of these parameters in the probabilistic model. These are learned using a version of the Expectation-Maximization (EM) algorithm called the Baum-Welch or *forward-backward* algorithm. This exact inference approach quickly becomes intractable, however for FHMMs, since the several state chains become conditionally dependent via the observed variable, making the computation of posterior probabilities in the E step infeasible.

To overcome this, techniques for approximating posterior probabilities can be applied, such as sampling schemes (e.g. Gibb’s sampling) or variational inference, in which a simpler distribution is solved for, which sufficiently approximates the true distribution and lacks the conditional dependencies that make exact inferences intractable.

Inference of the probabilistic parameters assumes a given model structure, or at least the number of hidden states in each chain. However our approach also requires the learning of that structure - this is the first point in the list above. This step could be integrated into an end-to-end expectation-maximization approach, for example by defining a probability distribution over possible parse trees for the grammar. Another approach akin to branch splitting based on information gain in decision trees could use the sequential entropy of an induced sequence as described in [1] - or alternatively, the entropy of a random walk on the *directly-follows graph* - to build a parse tree branch by branch. We leave a more rigorous investigation of these approaches for future work and instead present a heuristic approach based on clustering of directly follows graphs.

4.1 Structure Learning: Constructing Formulas from Clusters

Our aim is to find a set of formulas that are entailed by the event log such that the formulas are as *informative* as possible, specifically regarding resource dependencies and subprocess interplay. While one could imagine enumerating all possible formulas and allowing the inference algorithm to dismiss useless candidates with extremely low probabilities, this will likely slow and mislead the inference procedure unnecessarily.

The approach we propose essentially works by clustering similar subprocess as defined by their attribute-value groundings. So, returning to our example, we would likely to discover that *agent = Alice*, *agent = Bob*, *resource = Machine*, and *resource = Truck* are associated with similar sequences of actions, therefore likely involved in the same processes.

To accomplish this we propose using clustering techniques based on some distance measure defined between the sequences of actions associated with the difference attribute values. One candidate for defining a distance between sequence that comes to mind is sequence alignment or an edit-distance measure such as Levenshtein distance. These can be very expensive to compute and we leave an investigation of these for future work.

Clustering One straightforward method to define a distance based on the sequential patterns in two strings is based on the well-known *directly follows graph* (DFG). The adjacency matrix for the DFG counts the number of times an action represented by row i is followed by the action represented by column j .

By “unfolding” the rows of this matrix into a single vector (analogous to simple image processing), we can compute standard distance metrics defined in a vector space, such as the Euclidian distance. To capture that we are interested in the sequential *structure* rather than raw frequencies, we can normalize the DFG to form a proper probability distribution by marginalizing over rows.

Denote by \mathbf{D} the directly-follows matrix as described above and let \mathbf{J} denote the unit matrix (each element is 1) with the same dimensions as \mathbf{D} . Then the normalized directly follows graph can be written formally as

$$\mathbf{D}_{\text{norm}} = \mathbf{D} \oslash \mathbf{D}\mathbf{J} \quad (6)$$

Where \oslash denotes Hadamard (element-wise) division. Again, this simply normalizes the matrix across rows. we can then define distance measure between directly follows matrices and cluster attribute values, such as *agent = Alice* and *agent = Bob*, based on these. One distance metric which is straightforward to define is the cosine distance²:

$$d^{\cos}(\mathbf{A}, \mathbf{B}) = 1.0 - \cos \theta = 1.0 - \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = 1.0 - \frac{\sum_i \sum_j \mathbf{A}_{i,j} \mathbf{B}_{i,j}}{\sqrt{\sum_i \sum_j \mathbf{A}_{i,j}^2} \sqrt{\sum_i \sum_j \mathbf{B}_{i,j}^2}} \quad (7)$$

Since our aim is to find overlapping and interacting subprocesses, we want to be able to find a connection between, for example, process P_1 which overlaps with process P_2 which in turn overlaps with process P_3 even if P_1 and P_3 do not overlap at all. This makes linkage-based clustering algorithms natural candidates.

Crucially, in order to capture that attributes co-occur *in the same events*, we define the distance between two attributes α and β as the sum of the distance from the DFG for *alpha* alone to the DFG for α and β combined, and the distance from the DFG for *alpha* alone to the DFG for α and β combined:

$$d(\alpha, \beta) = d^{\cos}(\mathbf{D}_{\text{norm}}^{\alpha}, \mathbf{D}_{\text{norm}}^{\alpha\beta}) + d^{\cos}(\mathbf{D}_{\text{norm}}^{\alpha\beta}, \mathbf{D}_{\text{norm}}^{\beta})$$

One natural choice of clustering algorithm is DBSCAN, which is non-parametric and well suited to finding non linearly separable clusters. In contrast to most clustering approaches, it does not require us to choose a fixed number of clusters, but finds them automatically according to a threshold ϵ and a minimum number of elements per cluster. Roughly speaking, it looks for groups of points within an ϵ neighborhood and assigns remaining points to “noise”, i.e. singleton clusters.

Translation After clusters of directly-follows graph have been identified, translating these to condition control-flow formulas is relatively straightforward. For simplicity, in our implementation we have chosen to group all values (e.g. *Alice*, *Bob*) of the same attribute together. So for each attribute, consider the combined, normalized directly follows graph for these values. For every follows relation above some noise threshold η , add a follows relations associated with the relevant attribute constraint. The attribute constraint is simply set as

² The cosine distance is technically not a true distance metric since it does not fulfill the triangle equality.

$$\bigwedge_{1 \leq i < j \leq K} action_i.attribute = action_j.attribute$$

Next, combine the generated conditional control-flow formulas by means of handshake operators (\parallel_H). These will be used to generate the final factorial Markov model by grounding each interpretation and assigning to each a state-chain.

5 Application

We present some examples of conditional control-flow models discovered using the clustering approach described in Section 4.1. We mined an event log from a machining shop³, which has a rich amount of information associated with events, such as *WorkerID*, *Resource*, *Part Description*, *Order Type* and more. We have displayed the control-flow model graphically: again, a simple directly follows graph in this case, but potentially a richer model for a more sophisticated approach. Below this are examples of the raw textual output of the algorithm: In Figure 9 the full description is included, whereas in Figure 10 most of the textual description is omitted for readability.

The fully expanded model - including each state groundings for each state-chain in the FHMM model, along with the parameters (probability distributions) associated with it - would be overwhelming and unhelpful to present to the user. Therefore we envision display these handshaking subprocesses according to an importance ranking based on the state-chain prior, i.e. the final component $\mathbb{P}(m)$ in (7). Alternatively, a user could explore the model by querying certain attributes, searching by model complexity, or even relevance at a certain position in an execution trace. If a model were used for anomaly detection, when the system could present those models that are most explanatory for the low probability of an anomalous observation.

6 Future Work & Conclusion

We have presented a framework for 1) framing processes in terms of conditional control-flow models subject to constraints on event attributes and 2) an integration of models formulated in this manner with standard probabilistic models. Several details of this paper should be seen as simple proof-of-concept approaches to be elaborated on in future work. Specifically, more sophisticated control-flow models and attribute grammars, and different approaches to inference of models.

In terms of inference, a number of avenues exist. One obvious approach would be a maximum likelihood estimation (MLE) approach: building models from end-to-end that maximizes the probability of the observed data - one challenges

³ 10.4121/uuid:68726926-5ac5-4fab-b873-ee76ea412399

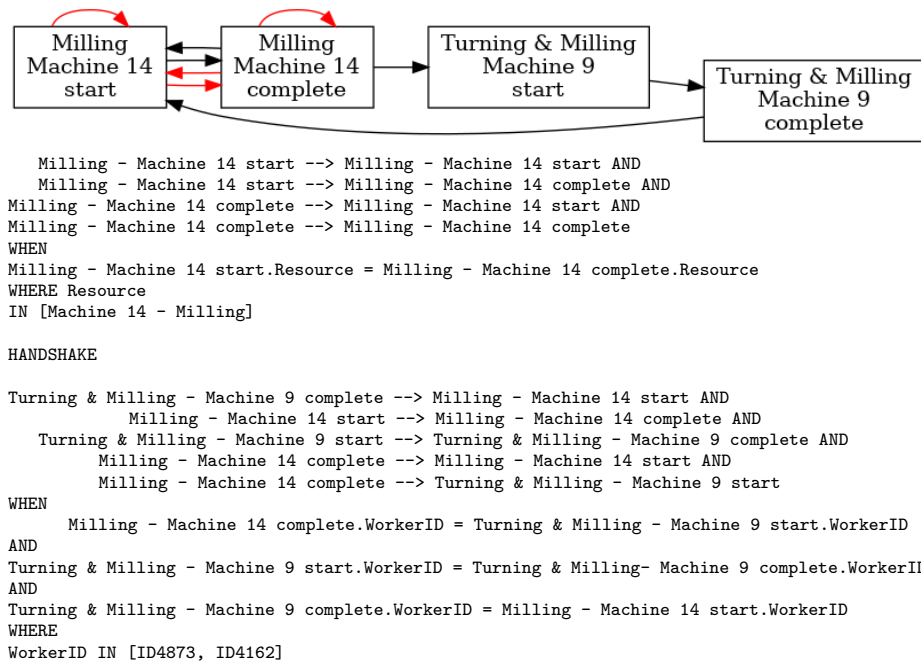


Fig. 9. Caption

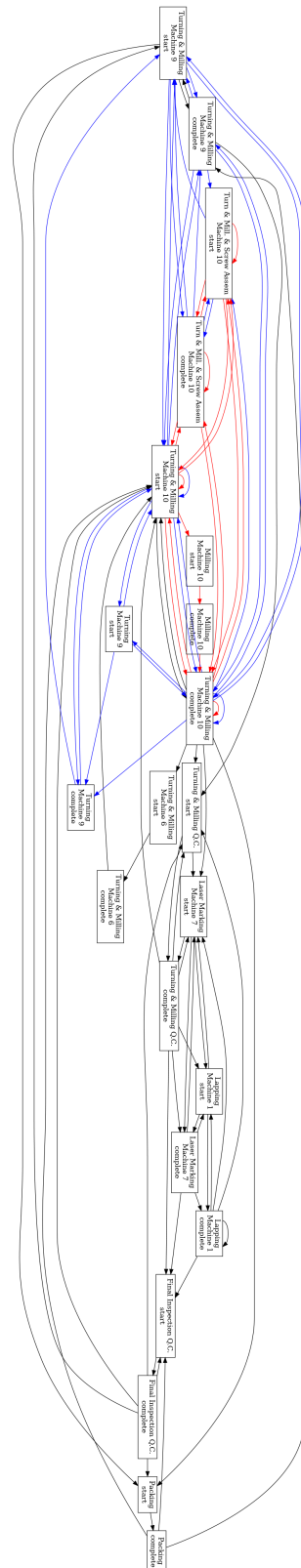


Fig. 10. Conditional follows graph for three resource conditions:
 Black: Part Desc. IN [Cylinder, Drill33]
 Red: Resource IN [Machine 10 - Grinding]
 Blue: WorkerID IN [ID4132]

here becomes building the grammar. A naive approach would require learning a probability distribution over all possible parse trees.

This highlights the distinction between learning 1) the structure and 2) the parameters of the model: a distinction we recognize from the probabilistic graphical model (PGM) literature. In the course of the present study we did explore similar approaches - the distinction here being that rather than determining whether to add an edge between two nodes representing *variables*, we are determining which rewrite rules two apply at a given point in a parse tree, and our statistical metrics for doing so are not simply defined between variables but on the resulting ordering induced on the event stream.

One potential approach akin to the notion of *information gain* in building decision trees, is to choose, at any point in the parse tree, the operation resulting in the greatest decrease in entropy (with some penalty for model complexity to avoid overfitting). One straightforward way to quantify this formulation of information gain could be based on the entropy of a random walk on the directly follows graph resulting from the grammar of a parse tree.

Other measure of information gain could be based on entropy estimator specifically intended for symbolic sequences, such as those explored in [1]. Some of these estimators requires very long sequences to converge to valid estimates. This might not be problem for in event streams in larger systems and would clearly more accurately capture long-range dependencies than DFGs. Entropy estimates can also take into account the regularity of temporal intervals, rather just symbolic sequences, if this is relevant for the application at hand.

Alternatively, continuing with the clustering approach presented here, a variety of alternative approaches to 1) sequence embedding and/or 2) alternative distance metrics could be explored. For example, various string/sequence embedding approaches exist, such as Parikh, spectrum, subsequence kernels [7] and the patterns described in [2], or sequence sampling based approaches which could be relevant for sufficiently long streams. Alternative distance metrics include, for example, edit distances which don't require embedding in a vector space.

The grammar presented here was intentionally kept as simple as possible, using only an equivalence relation. Even without adding new relations/operators, more expressive rewrite rules can be formulated. For example, it is not possible to express For example, $a.Agent = b.Resource$ in the current set of rewrite rules. Adding logical operators (\neg , \rightarrow) and other relations - perhaps domain specific and based on available data schema - would be able to capture more complex relations between attributes.

Finally, a proper evaluation of the proposed framework is a crucial next step. This could be approached in different ways: by using the model as a classifier and comparing with other approach on standard datasets, by evaluation on a dataset for which attribute dependencies are known and determine whether these are discovered, and finally usability studies to determine whether the format proposed is informative and easy to interpret.

References

1. Back, C.O., Debois, S., Slaats, T.: Entropy as a measure of log variability. *Journal on Data Semantics* **8**(2), 129–156 (2019)
2. Bose, R.J.C., van der Aalst, W.M.: Trace clustering based on conserved patterns: Towards achieving better process models. In: *International Conference on Business Process Management*. pp. 170–181. Springer (2009)
3. Bose, R.J.C., Mans, R.S., van der Aalst, W.M.: Wanna improve process mining results? In: *2013 IEEE symposium on computational intelligence and data mining (CIDM)*. pp. 127–134. IEEE (2013)
4. Di Ciccio, C., Mecella, M.: Mining artful processes from knowledge workers’ emails. *IEEE Internet Computing* **17**(5), 10–20 (2013)
5. Diba, K., Batoulis, K., Weidlich, M., Weske, M.: Extraction, correlation, and abstraction of event data for process mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **10**(3), e1346 (2020)
6. Grenander, U., Miller, M.I., Miller, M., et al.: *Pattern theory: from representation to inference*. Oxford university press (2007)
7. De la Higuera, C.: *Grammatical inference: learning automata and grammars*. Cambridge University Press (2010)
8. Mannhardt, F., De Leoni, M., Reijers, H.A.: Extending process logs with events from supplementary sources. In: *International Conference on Business Process Management*. pp. 235–247. Springer (2014)
9. Mark, K., Miller, M., Grenander, U., Abney, S.: Parameter estimation for constrained context-free language models. In: *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992* (1992)
10. de Murillas, E.G.L., van der Aalst, W.M., Reijers, H.A.: Process mining on databases: Unearthing historical data from redo logs. In: *International Conference on Business Process Management*. pp. 367–385. Springer (2016)
11. Van Der Aalst, W.M., Van Hee, K.M., Ter Hofstede, A.H., Sidorova, N., Verbeek, H., Voorhoeve, M., Wynn, M.T.: Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing* **23**(3), 333–363 (2011)
12. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.: Event stream-based process discovery using abstract representations. *Knowledge and Information Systems* **54**(2), 407–435 (2018)