International Journal on Software Tools for Technology Transfer (2022) 24:563–587
https://doi.org/10.1007/s10009-021-00616-0

COMPETITIONS AND CHALLENGES

Regular

DisCoveR: accurate and efficient discovery of declarative process models

Christoffer Olling Back[1] · Tijs Slaats[1] · Thomas Troels Hildebrandt[1] · Morten Marquard[2]

Accepted: 6 May 2021 / Published online: 28 June 2021

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 2019 (Results are available at https://icpmconference.org/2019/process-discovery-contest). We appeal to computational learning theory to gain insight into its performance as a classifier. Due to its linear time complexity, DisCoveR also achieves much faster run times 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 and 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

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 [30,84,89]. Today, BPMN [29,62,88] 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 [44] that support the automated discovery of BPMN models from event logs [83,85].

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 [28,55,65,70,75]. This approach is often characterized as *declarative* and juxtaposed with *imperative* notations like BPMN [39,52,66].

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 [24,36]. 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.

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

Tijs Slaats
slaats@di.ku.dk

Thomas Troels Hildebrandt
hilde@di.ku.dk

Morten Marquard
mm@dcrsolutions.net

1 Department of Computer Science, University of Copenhagen, Copenhagen, Denmark

2 DCR Solutions, Copenhagen, Denmark

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 EcoKnow[1] project which builds upon the DCR Graphs formalism [40,55,75].

To support the local development and maintenance of the declarative DCR models, several modeling tools have been developed [16,20,52], supported by formal understandability studies [1–3]. 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 [59]. 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 [59], and the current paper expands on this initial introduction with: a complete and thorough formalization of the algorithm that provides all details required for its implementation (Sect. 4); a novel, open source and more efficient implementation based on bit vector operations (Sect. 5); an evaluation of the algorithm against flagship academic miners based on the classification task provided by the Process Discovery Contest 2019 and a run time comparison 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 (Sect. 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 (Sect. 7).

After surveying related work in Sect. 2 and introducing preliminaries in Sect. 3, we proceed as sketched above, concluding and proposing future directions of research in Sect. 8.

2 Related work

Many declarative process notations have been developed, several with corresponding discovery algorithms [76]. One of the first of these was Declare [66,82,86], which was inspired by property specification patterns for linear temporal logic (LTL) [31]. 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 [48], SCIFF [53,54], and regular expressions [92]. Extensions to Declare include timed [90] and data [18] constraints, which were combined in MP-Declare [10] (Multi-Perspective Declare), and hierarchy [95]. The first miner for Declare was the Declare Maps Miner [49], while initially using a brute-force approach, it was extended with several improvements [46] inspired by the Apriori algorithm for association rule mining [6]. More recently, the miner was extended to allow for parallelization [47]. The second Declare miner to be developed was Minerful [14], which provided significant gains in efficiency. Since its introduction, it has been extended with support for target-branched constraints [27], removal of redundancies and inconsistencies [12] and removal of vacuously satisfied constraints [13].

Another prominent declarative approach is the Guard-Stage-Milestone (GSM) notation [41], inspired by earlier work on artifact-centric business processes [9]. GSM aims to effectively model case management and has been a primary contributor to the development of the Case Management Model And Notation (CMMN) [61]. CMMN has seen a relatively fast industrial and academic adoption through the development of tools and case studies [35,43,93]. Work on process discovery for GSM or CMMN on the other hand is still rather sparse, and only one discovery algorithm has been proposed to date [67] with no working implementation.

[1] Effective, co-created and compliant adaptive case management for Knowledge workers.

Process discovery has also been considered for the Declarative Process Intermediate Language (DPIL) [72,94], which is a textual, multi-perspective, declarative modeling language. Process discovery for DPIL is supported through the DPIL Miner.[2] In comparison with 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 [71]. Interestingly, the miner has never been made publicly available and its effectiveness or accuracy cannot be independently ascertained.

In more recent work, it has been proposed to combine declarative and imperative discovery to produce the so-called hybrid [7,21,69,79] or mixed [17,19,91] models that combine both paradigms. Hybrid miners include the Fusion miner [80], which produces an inter-mixed Petri net and Declare model, the Hybrid Miner [50] which produces a hierarchical Petri net and Declare model, and the Precision Optimization Hybrid Miner [73] 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 [63,64] which guarantees optimally concurrent, correct-by-construction processes. The framework was applied to intra-hospital patient transfers in [51].

Temporal logics have also been used to model phenomena which would not be considered workflows, such as robot motion [32], naval traffic, and train network monitoring [42].

Finally, DCR Graphs were inspired by event structures [60] and developed after Declare was shown to not be sufficiently expressive in modeling industrial cases [57]. In contrast to Declare, the semantics of DCR Graphs are defined as transformations on the markings of the events. This allows modelers to straightforwardly reason about the execution semantics of a model by simulating it and observing the changes to the markings as events are executed. [52] Since their inception, DCR Graphs have been extended with nesting [37], time [38], data [16,56,78], and hierarchy [22].

Regarding evaluation, process mining has traditionally been 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 [33]. Some authors have addressed this by developing techniques to generate artificial negative examples [34].

3 Preliminaries

We present here the 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 Sect. 6.
– An *event*, denoted ς, is a particular occurrence of an activity.
– A *trace* $\sigma \in \Sigma^* \cup \Sigma^\omega = \langle \varsigma_1, \ldots, \varsigma_i, \ldots \rangle$ 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:

$$L = \left\{\sigma_1^{m(\sigma_1)}, \ldots, \sigma_n^{m(\sigma_n)}\right\}$$

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,

[2] 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, WorkFlow 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.
- $\rightarrow\!+ \bigcap \rightarrow\!\% = \emptyset$.
- $l : \mathcal{E} \rightarrow A$ is a labeling function mapping every "event" to an activity.

A DCR Graph marking $m = (\mathsf{Ex}, \mathsf{Pe}, \mathsf{In})$ 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 $\mathsf{Pe} \cap \mathsf{In} = \emptyset$, 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 [58].

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., $e \in \mathsf{In}$
- If any condition relations exist s.t. $e' \rightarrow\!\!\bullet e$, then all such e' must have been executed, *or excluded*, i.e., $e' \in \mathsf{Ex}$ or $e' \notin \mathsf{In}$. 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., $e' \in \mathsf{Pe}$
- If any excludes relations exist s.t. $e \rightarrow\!\% e'$, then any included e' will become *excluded*, i.e., $e' \notin \mathsf{In}$.
- If any includes relations exist s.t. $e \rightarrow\!+ e'$, then any excluded e' will become *included*, i.e., $e' \in \mathsf{In}$.

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 Sect. 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

$$E \equiv \{1, \ldots, |\Sigma_L|\}$$

Table 1 Relevant constraint templates from Declare

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)	*See caption*	For b to occur, a must occur immediately prior
NotChainSuccession(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

The ChainPrecedencerelation is not straightforward to encode in DCR Graphs relations and in fact, ParNek looks for evidence of ChainPrecedencerelations, but encodes them as $a \rightarrow+\ b, b \rightarrow\%\ b$, which is not exactly equivalent

```
input  : A log $L$
output: A DCR Graph $G$

                                                                        // INITIALIZATION
  $E \equiv \{1, \ldots, |\Sigma_L|\}$                                               // set of events
  $A \equiv \Sigma_L$                                                        // activities in log
  $l \equiv i \in E \mapsto s_i \in \Sigma_L$                  // bijective labeling (events and activities)
  $m \equiv (\emptyset, \emptyset, E)$                                                   // initial marking
$\rightarrow+ \equiv \emptyset$                                               // set of includes relations
$\rightarrow\% \equiv \emptyset$                                               // set of excludes relations
$\rightarrow\bullet \equiv \emptyset$                                              // set of condition relations
$\bullet\rightarrow \equiv \emptyset$                                               // set of response relations

                                                              // ADD 'DECLARE' TEMPLATES
$\rightarrow\% := \rightarrow\% \bigcup \{ (s, s) \mid s \in AtMostOne(L) \}$                   // self exclusions
$\rightarrow\bullet := \rightarrow\bullet \bigcup Precedence(L)$                                 // condition relations
$\bullet\rightarrow := \bullet\rightarrow \bigcup Response(L)$                                    // response relations
$\rightarrow+ := \rightarrow+ \bigcup \{ (s, t) \mid s \neq t \bigwedge (s, t) \in ChainPrecedence(L) \}$     // alternate precedence
$\rightarrow\% := \rightarrow\% \bigcup \{ (t, t) \mid \exists s, s \neq t. (s, t) \in ChainPrecedence(L) \}$    // alternate precedence


                                                              // ADD ADDITIONAL EXCLUDES
$\rightarrow\% := \rightarrow\% \cup \bigcup_t ChooseOneRelation\big( \{ (s, t) \mid (s, t) \notin Predecessors(L) \bigwedge$     // not coexistence
                                          $(s, t) \notin Successors(L) \bigwedge s \neq t \}$
                                   $\cup \{ (s, t) \mid (t, s) \in Predecessors(L) \bigwedge$          // not succession
                                          $(t, s) \notin Successors(L) \bigwedge$
                                          $(t, t) \notin \rightarrow\% \bigwedge s \neq t \} \big)$

                                                     // ADDITIONAL INCLUDES/EXCLUDES
$\rightarrow\% := \rightarrow\% \bigcup NotChainSuccession(L)$                           // not chain succession
$\rightarrow+ := \rightarrow+ \bigcup \{ (u, t) \mid \exists s. (s, t) \in NotChainSuccession(L) \bigwedge (s, u, t) \in Between(L) \}$

                                                        // REMOVE 'REDUNDANT' EXCLUSIONS
$\rightarrow\% := \rightarrow\% \setminus \{ (s, t) \mid \exists u. (u, t) \in \rightarrow\% \bigwedge$
                    $(u, s) \in AlternatePrecedence(L) \}$

                                          // REMOVE 'REDUNDANT' CONDITIONS/RESPONSES
$\bullet\rightarrow := \bullet\rightarrow \setminus \{ (s, t) \mid (s, u) \in \bullet\rightarrow \bigwedge (u, t) \in \bullet\rightarrow \}$
$\rightarrow\bullet := \rightarrow\bullet \setminus \{ (s, t) \mid (s, u) \in \rightarrow\bullet \bigwedge (u, t) \in \rightarrow\bullet \}$

                                                          // ADD ADDITIONAL CONDITIONS
$\rightarrow\bullet := \rightarrow\bullet \bigcup \{ (s, t) \mid (\exists \sigma \in L.\forall k. s = \sigma(i) \bigwedge t = \sigma(j) = \sigma(k) \bigwedge i < j \leq k ) \bigwedge$
                    $(\forall \sigma \in L.\forall i > j. s = \sigma(i) \bigwedge t = \sigma(j) \bigwedge \exists h < j. (\sigma(h), s) \in \rightarrow\% \bigwedge$
                    $\nexists g < j. g > h \wedge (\sigma(g), s) \in \rightarrow+ )\}$

                                                        // REMOVE 'REDUNDANT' CONDITIONS
$\bullet\rightarrow := \bullet\rightarrow \setminus \{ (s, t) \mid (s, u) \in \bullet\rightarrow \bigwedge (u, t) \in \bullet\rightarrow \}$

return $(E, M, A, \bullet\rightarrow, \rightarrow\bullet, \rightarrow+, \rightarrow\%, l)$                         // RETURN DCR GRAPH
```

Algorithm 1: High-level control flow of the mining algorithm.

Table 2 Formal definitions of helper functions which return sets of relevant relations

$\boldsymbol{AtMostOne}$:				
$L \mapsto \Big\{ s$	$\mid$	$s \in \Sigma_L \bigwedge i, j \in \mathbb{N}$	$\bigwedge \forall \sigma \in L \, \forall i, j.$	$s = \sigma(i) = \sigma(j) \implies i = j \Big\}$
$\boldsymbol{Response} : L \mapsto \Big\{ (s, t)$	$\mid$	$s, t \in \Sigma_L \bigwedge i, j \in \mathbb{N}$	$\bigwedge \forall \sigma \in L \, \forall i.$	$s = \sigma(i) \implies \exists j \Big(t = \sigma(j) \bigwedge i < j \Big) \Big\}$
$\boldsymbol{Precedence} : L \mapsto \Big\{ (s, t)$	$\mid$	$s, t \in \Sigma_L \bigwedge i, j \in \mathbb{N}$	$\bigwedge \forall \sigma \in L \, \forall i.$	$t = \sigma(i) \implies \exists j \Big(s = \sigma(j) \bigwedge j < i \Big) \Big\}$
$\boldsymbol{AlternatePrecedence} : L \mapsto \Big\{ (s, t)$	$\mid$	$s, t \in \Sigma_L \bigwedge i, j, k \in \mathbb{N}$	$\bigwedge \forall \sigma \in L \, \forall i.$	$t = \sigma(i) \implies \exists j \Big(s = \sigma(j) \bigwedge j < i \bigwedge$
$\boldsymbol{ChainPrecedence}$:				$\nexists k \Big(t = \sigma(k) \bigwedge j < k < i \Big) \Big) \Big\}$
$L \mapsto \Big\{ (s, t)$	$\mid$	$s, t \in \Sigma_L \bigwedge i \in \mathbb{N}$	$\bigwedge \forall \sigma \in L \, \forall i.$	$t = \sigma(i) \implies s = \sigma(i - 1) \Big\}$
$\boldsymbol{NotChainSuccession} : L \mapsto \Big\{ (s, t)$	$\mid$	$s, t \in \Sigma_L \bigwedge i \in \mathbb{N}$	$\bigwedge \nexists \sigma \in L \, \forall i.$	$\Big(s = \sigma(i) \bigwedge t = \sigma(i + 1) \Big) \Big\}$
$\boldsymbol{Predecessors} : L \mapsto \Big\{ (s, t)$	$\mid$	$s, t \in \Sigma_L \bigwedge i, j \in \mathbb{N}$	$\bigwedge \exists \sigma \in L \, \exists i, j.$	$\Big(s = \sigma(i) \bigwedge t = \sigma(j) \bigwedge i < j \Big) \Big\}$
$\boldsymbol{Successors} : L \mapsto \Big\{ (s, t)$	$\mid$	$s, t \in \Sigma_L \bigwedge i, j \in \mathbb{N}$	$\bigwedge \exists \sigma \in L \, \exists i, j.$	$\Big(s = \sigma(i) \bigwedge t = \sigma(j) \bigwedge i > j \Big) \Big\}$
$\boldsymbol{Between} : L \mapsto \Big\{ (s, u, t)$	$\mid$	$s, u, t \in \Sigma_L \bigwedge i, j, k \in \mathbb{N}$	$\bigwedge \exists \sigma \in L \, \exists i, k, j.$	$\Big(s = \sigma(i) \bigwedge u = \sigma(k) \bigwedge t = \sigma(j)$
$\boldsymbol{ChooseOneRelation}$:				$\bigwedge i < k < j \Big) \Big\}$
$R \in 2^{\Sigma \times \Sigma}$	$\mapsto$	$(s, t) \in R$		

All functions have event logs as their domain ($\mathcal{L}$), except $ChooseOneRelation$

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 is 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.

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 (nearly[3]) 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. Note that these relations are, in fact, each other's dual:

$$(a, b) \in Predecessor(L) \implies (b, a) \in Successor$$

Nonetheless, to maintain consistency between the implementation described in Sect. 5, we distinguish between the two.

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 determinstic), first-come manner with a more sophisticated approach being left for future work.

Includes and Excludes—NOTCHAINSUCCESSION (lines: 23–24): To identify further includes and excludes relations, we rely on $NotChainSuccession(L)$ 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 $s \rightarrow\!\% \ t$ (NOTCHAINSUCCESSION). If, however, t occurs after s, with some sequence of intermediate activities s.t. we have $\langle \ldots, s, u_1, \ldots, u_n, t, \ldots \rangle$, 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

[3] 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.

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 an 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 afterward. In many models, the reduction in relations is very substantial. See Fig. 1 for a graphical illustration.

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 full JAVA source code is provided as open source (licensed under LGPL-3.0) at [81]. 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.

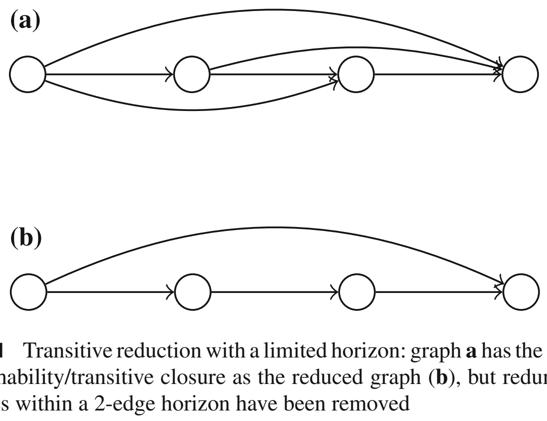


Fig. 1 Transitive reduction with a limited horizon: graph **a** has the same reachability/transitive closure as the reduced graph (**b**), but redundant edges within a 2-edge horizon have been removed

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 afterward 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. [20,45].

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 [81]: 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 [81]: BitDCRGraph, ln. 28-33

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 afterward 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 [81]: 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 [81]: 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. Afterward, 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 (Table 4).

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 will further address how we reduced the computation of *Additional Conditions* to linear time later in this section.

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 afterward during the actual model building steps as described in Sect. 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

Table 3 Example of bit operations involved in `execute` method (see Listing 4)

	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

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 [81]: 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 occured
            before the first instance
        this.precedenceFor.get(i).and(
            localAtLeastOnce);
        // ChainPrecedence for (i): any
             activity that occured
            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 [81]: 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 [81]: 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 [81]: 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 [81]):

```
    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 see
                    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 [81]: 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 2019,[4] 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. [25]; (2) two leading miners also based on the declarative paradigm: MINERful[15] and Declare Miner [47]; (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 [5], or model specific measures such as *token replay* metrics for Petri nets [68]. 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 toward committing different classes of statistical errors (e.g., Type I/II) by analyzing *true/false positives/negatives*, and the corresponding *precision*, *recall*, F_1-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.

[4] https://icpmconference.org/2019/process-discovery-contest

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 [4]. The training data L are an i.i.d.[5] 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:

$$\gamma : \mathcal{L} \to \mathcal{H};\; L \mapsto \underset{h \in \mathcal{H}}{\arg\min}\, E_{\text{out}}(h)$$

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

$$\mathbb{1}(\sigma \in P)$$

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*):

$$E^r_{\text{in}}(h) = \sum_{\sigma \in L} \frac{\mathbb{1}(\sigma \in \ell(h))}{|L|}$$

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*):

$$E^a_{\text{in}}(h) = \sum_{\sigma \in L} \frac{\mathbb{1}(\sigma \in \ell(h) \iff \sigma \in P)}{|L|}$$

We include this formulation (E^a_{in}) 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 $E_{\text{in}}(h)$ is trivial if the hypothesis set $\mathcal{H}$ is large enough. Indeed, E^r_{in} can be trivially minimized by a flower process model which permits all

[5] Independent and identically distributed.

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,

$$|E(h)_{\text{in}} - E(h)_{\text{out}}| < \epsilon$$

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

$$\mathbb{P}[\ |E(h)_{\text{in}} - E(h)_{\text{out}}| < \epsilon\ ] > 1 - \delta$$

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*[6] of a hypothesis set and a penalty for the *complexity* of a hypothesis [74]. The sum of these terms gives an *estimate* of the out-of-sample error:

$$\hat{E}_{\text{out}} = E_{\text{in}} + \Omega(N, \mathcal{H}, \delta).$$

Where N denotes sample size, $\mathcal{H}$ the hypothesis space and δ the desired confidence that $E_{\text{out}} \leq \hat{E}_{\text{out}}$.

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 $\hat{E}_{\text{out}}$ rather than E_{in}, can greatly improve results.

ParNek does not currently attempt to explicitly minimize $\hat{E}_{\text{out}}$, 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

$$\mathcal{H}_{\text{ParNek}} \subset \mathcal{H}_{\text{DCR}} = \omega\text{-regular languages}$$

[6] The minimal in-sample error achievable for hypothesis $h \in \mathcal{H}$.

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 with 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:

PREDICTION	DATA +	−	
+	(TP)	False Pos. (FP)	prec. $\equiv \frac{TP}{TP+FP}$
−	False Neg. (FN)	True Neg. (TN)	
	recall $\equiv \frac{TP}{TP+FN}$	acc.$\equiv \frac{TP+TN}{TP+TN+FP+FN}$	

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 [11], 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 noncompliance (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 4 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. [25]; second, two miners based on Declare constraints, MINERful[15] and Declare Miner [47]; 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 toward 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 [8]. 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, NOTCOEX-

ISTENCE, 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 4. 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 run time performance to the same miners as in our classification evaluation, finding that DisCoveR performs comparably with the fastest miners, and much faster than Declare-based miners, MINERful and Declare miner, even when multithreading is enabled. Note that for run time comparison, the linear-time IMD variant of Inductive Miner from the `pm4py`[7] 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.

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 run time. 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 4 which requires re-running the miner many times.

The 10 logs all consist of 700 traces. Run time results can be seen in Fig. 2 as well as Table 5, where details regarding number of activities and mean trace length are also included.

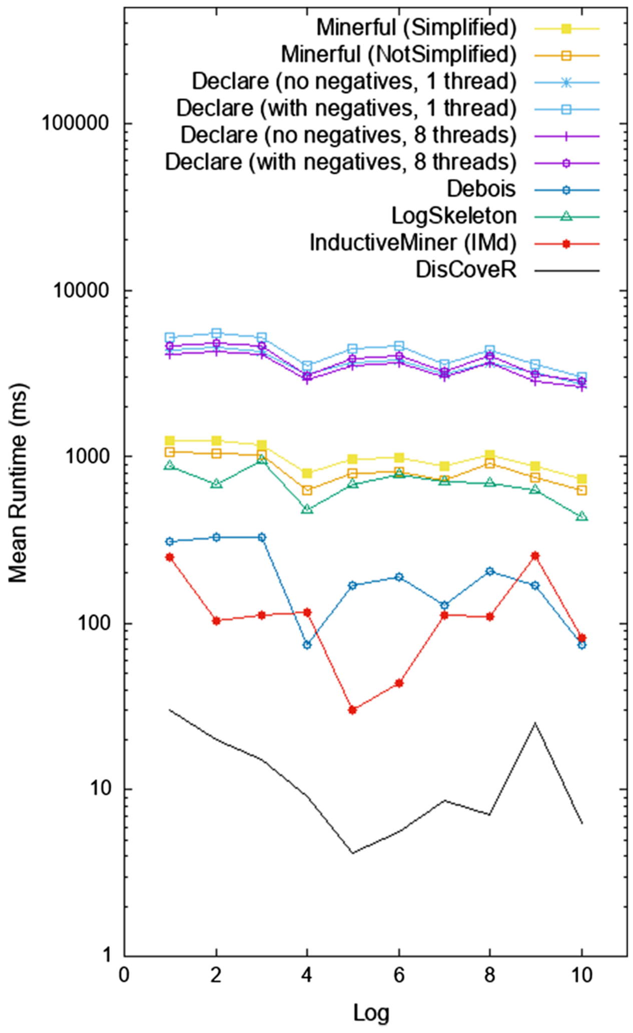


Fig. 2 Mean run times in milliseconds across 100 runs on Process Discovery Contest 2019 training logs. MINERful was run with the thresholds: *support* = 1.0, *confidence* = 1.0, *interest factor* = 1.0, with and without a post-processing step to simplify models. Declare Miner was with and without multithreading, and with *alpha* = 1.0, *support* = 1.0, with all constraint templates and with all but the negative constraint templates. For run times, 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

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 run times, especially for very low run times. 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 run time estimates w.r.t. increases in runs, finding that estimates stabilized by 100 runs and clearly so by 1000 and 10,000 runs. We

[7] http://pm4py.org

Table 4 Confusion matrices for individual data sets, each generated by separate ground truth model, in our formulation referred to as $(P_i, \mathbb{P}_{P_i})$

	OBSERVED																				Aggregate			TARGET TRACES	
	P_1		P_2		P_3		P_4		P_5		P_6		P_7		P_8		P_9		P_{10}					Positive	Negative
DisCoveR	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	MCC	*0.92*	
PRED- +	45	0	45	0	45	0	47	6	45	3	45	0	44	8	43	2	45	3	44	8	**448**	**30**	Prec.	0.94	0.99
ICTED −	0	45	0	45	0	45	1	36	0	42	0	45	1	37	2	43	0	42	1	37	**5**	**417**	Recall	0.99	0.93
Model size	*142*		*189*		*271*		*182*		*447*		*412*		*143*		*284*		*171*		*136*		Acc.:	*96.1%*	F_1	0.96	0.96
Log Skeleton	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	MCC	*0.85*	
PRED- +	36	1	42	0	38	2	47	1	45	3	45	0	39	8	38	5	44	1	39	6	**413**	**27**	Prec.	0.94	0.91
ICTED −	9	44	3	45	7	43	1	41	0	42	0	45	6	37	7	40	1	44	6	39	**40**	**420**	Recall	0.91	0.94
Model size	*235*		*240*		*250*		*180*		*230*		*225*		*185*		*230*		*155*		*170*		Acc.:	*92.6%*	F_1	0.92	0.93
MINERful[2]	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	MCC	*0.85*	
PRED- +	43	2	45	1	44	4	48	1	45	19	45	4	45	8	45	7	45	14	42	7	**447**	**67**	Prec.	0.87	0.98
ICTED −	2	43	0	44	1	41	0	41	0	26	0	41	0	37	0	38	0	31	3	38	**6**	**380**	Recall	0.99	0.85
Model size	*182*		*188*		*186*		*194*		*198*		*199*		*199*		*189*		*174*		*183*		Acc.:	*91.9%*	F_1	0.92	0.91
MINERful[1]	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	MCC	*0.79*	
PRED- +	45	7	45	2	45	6	48	5	45	21	45	3	45	13	45	15	45	17	44	7	**452**	**96**	Prec.	0.82	0.98
ICTED −	0	38	0	43	0	39	0	37	0	24	0	42	0	32	0	30	0	28	1	38	**1**	**351**	Recall	0.98	0.79
Model size	*99*		*99*		*92*		*96*		*89*		*99*		*99*		*94*		*94*		*97*		Acc.:	*89.9%*	F_1	0.90	0.87
DeclarePos	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	MCC	*0.50*	
PRED- +	45	25	45	11	45	19	48	24	45	35	45	37	44	33	45	38	45	8	44	37	**451**	**267**	Prec.	0.63	0.99
ICTED −	0	20	0	34	0	26	0	18	0	10	0	8	1	12	0	7	0	37	1	8	**2**	**180**	Recall	0.996	0.40
Model size	*464*		*759*		*269*		*623*		*338*		*778*		*259*		*242*		*885*		*379*		Acc.:	*70.1%*	F_1	0.77	0.57

Table 4 continued

		OBSERVED																				Aggregate			TARGET TRACES	
		P_1		P_2		P_3		P_4		P_5		P_6		P_7		P_8		P_9		P_{10}					Positive	Negative
Inductive$^{0.0}$		+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	MCC	*0.49*	
PRED-	+	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
ICTED	−	0	10	0	45	0	9	0	8	0	38	0	43	0	1	0	2	1	16	0	0	**1**	**172**	Recall	0.997	0.38
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-	+	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
ICTED	−	0	20	3	37	2	24	0	17	0	10	0	6	1	12	0	7	1	38	2	9	**14**	**180**	Recall	0.97	0.40
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-	+	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
ICTED	−	0	10	45	45	28	37	48	42	14	43	2	45	10	14	7	12	45	45	45	44	**244**	**337**	Recall	0.46	0.75
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-	+	0	0	0	0	0	0	0	0	31	2	30	0	0	0	0	0	0	0	0	0	**61**	**2**	Prec.	0.97	0.53
ICTED	−	45	45	45	45	45	45	48	42	14	43	15	45	45	45	45	45	45	45	45	45	**392**	**445**	Recall	0.13	0.996
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-	+	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
ICTED	−	45	45	45	45	45	44	42	48	38	33	45	45	41	39	44	44	45	44	37	39	**427**	**426**	Recall	0.06	0.96
Model size		*1821*		*2293*		*2376*		*641*		*1557*		*1515*		*1268*		*1716*		*984*		*775*		Acc.:	50.4%	F_1	0.11	0.66

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. MINERful^n 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. Inductive^n 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 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 [26]. 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
```

Table 5 Mean run times in milliseconds across 100 runs on Process Discovery Contest 2019 training logs, along with log statistics

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

See Fig. 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

```
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.

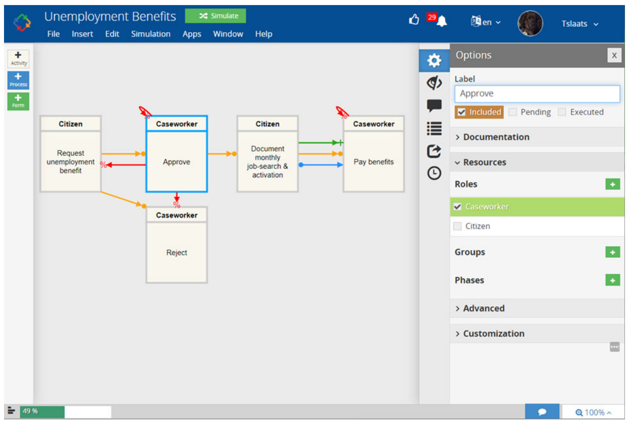
7.1 The DCR process portal

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 institutions[8] and the portal has become a cornerstone of the Ecoknow research project,[9] 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 Fig. 3, which allows users to model and simulate

[8] http://www.kmd.dk/indsigter/fleksibilitet-og-dynamisk-sagsbehandling-i-staten

[9] https://ecoknow.org/



Label: Approve
[✓ Included] Pending Executed
Roles: [✓ Caseworker] Citizen

Fig. 3 DCR graphs modeling

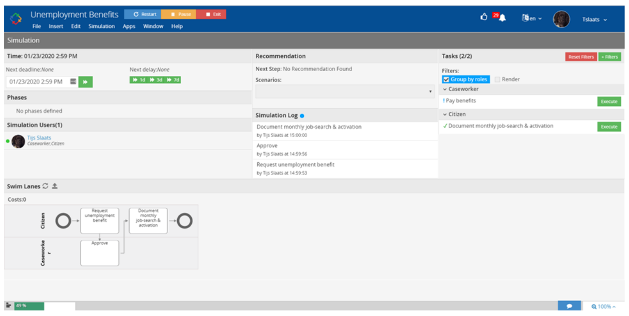


Fig. 4 DCR graphs simulation

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 Fig. 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 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.

7.2 Interactive process modeling through model recommendation

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 [77]. 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.

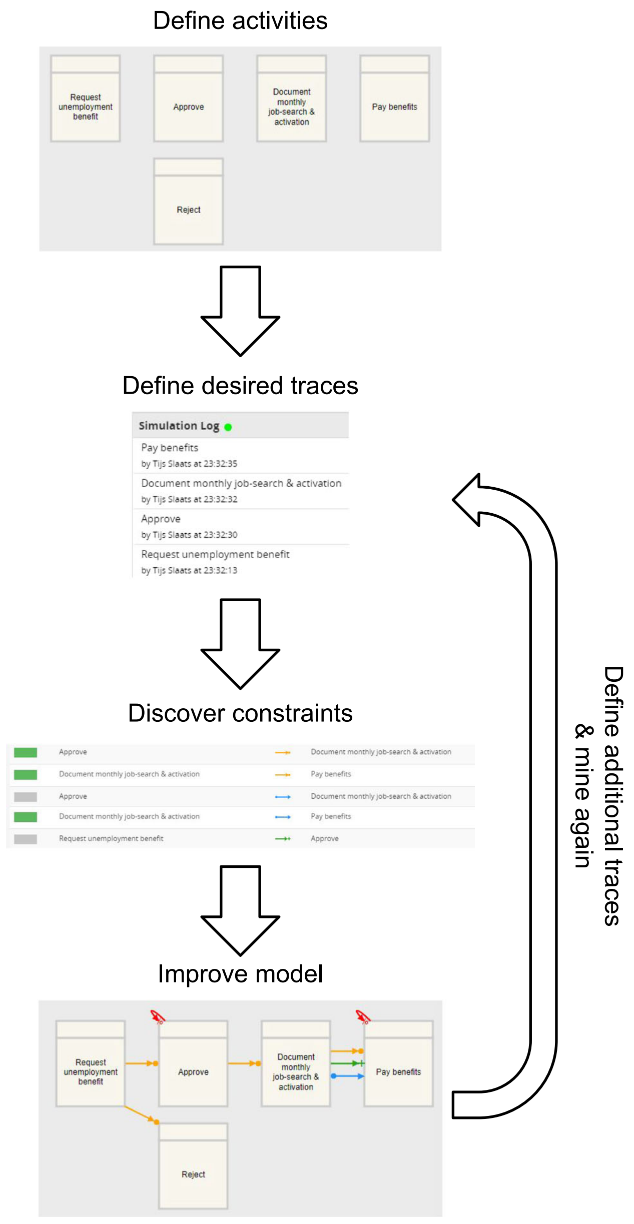
Figure 5 provides an overview of the adapted approach: we start by identifying the activities of the process and modeling these directly in the portal. In the next step, we run simulations on these activities (recall that following the declarative paradigm, these simulations are entirely unconstrained 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.

The model recommendation screen is shown in Fig. 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.

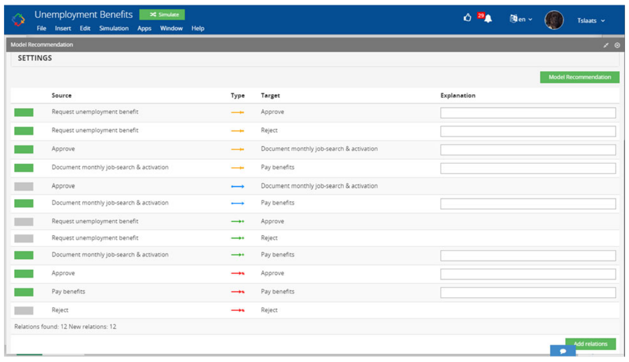
In the traditional modeling exercise, users that are more familiar with BPMN and/or flowcharts are often hampered by the novelty of the notation, e.g., they will be unclear


Define activities
Request unemployment benefit
Approve
Document monthly job-search & activation
Pay benefits
Reject
Define desired traces
Discover constraints
Improve model
Define additional traces & mine again

Simulation Log
Pay benefits
by Tijs Slaats at 23:32:35
Document monthly job-search & activation
by Tijs Slaats at 23:32:32
Approve
by Tijs Slaats at 23:32:30
Request unemployment benefit
by Tijs Slaats at 23:32:13

🟩	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

Fig. 5 Overview of the model recommendation approach



	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	→%	Approve	
🟩	Pay benefits	→%	Pay benefits	
⬜	Reject	→%	Reject	

Relations found: 12 New relations: 12

Fig. 6 Model recommendation

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 (0.961), 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. However, this 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 with 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.

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 [23], 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 are 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. Abbad Andaloussi, A., Buch-Lorentsen, J., López, H.A., Slaats, T., Weber, B.: Exploring the modeling of declarative processes using a hybrid approach. In: Laender, A.H.F., Pernici, B., Lim, E.P., de Oliveira, J.P.M. (eds.) Conceptual Modeling, pp. 162–170. Springer, Cham (2019)
2. 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: Reinhartz-Berger, I., Zdravkovic, J., Gulden, J., Schmidt, R. (eds.) Enterprise, Business-Process and Information Systems Modeling, pp. 69–84. Springer, Cham (2019)
3. 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: Daniel, F., Sheng, Q.Z., Motahari, H. (eds.) Business Process Management Workshops, pp. 475–481. Springer, Cham (2019)
4. Abu-Mostafa, Y.S., Magdon-Ismail, M., Lin, H.: Learning from data: a short course. AMLBook.com (2012). https://books.google.co.uk/books?id=iZUzMwEACAAJ
5. 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)
6. 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 (1994). http://dl.acm.org/citation.cfm?id=645920.672836
7. 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). https://doi.org/10.1016/j.is.2020.101505
8. 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)
9. Bhattacharya, K., Gerede, C., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: In preparation, pp. 288–304 (2007)
10. Burattin, A., Maggi, F.M., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. Expert Syst. Appl. **65**, 194–211 (2016)
11. Chicco, D., Jurman, G.: The advantages of the Matthews correlation coefficient (MCC) over f1 score and accuracy in binary classification evaluation. BMC Genom. **21**(1), 6 (2020)
12. Ciccio, C.D., Maggi, F.M., Montali, M., Mendling, J.: Resolving inconsistencies and redundancies in declarative process models. Inf. Syst. **64**, 425–446 (2017)
13. Ciccio, C.D., Maggi, F.M., Montali, M., Mendling, J.: On the relevance of a business constraint to an event log. Inf. Syst. **78**, 144–161 (2018)
14. Ciccio, C.D., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM Trans. Manag. Inf. Syst. **5**(4), 24:1–24:37 (2015). https://doi.org/10.1145/2629447
15. Ciccio, C.D., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM Trans. Manag. Inf. Syst. **5**(4), 1–37 (2015)
16. 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). https://doi.org/10.1109/EDOC.2018.00020
17. De Giacomo, G., Dumas, M., Maggi, F.M., Montali, M.: Declarative process modeling in BPMN. In: Zdravkovic, J., Kirikova, M., Johannesson, P. (eds.) Advanced Information Systems Engineering, pp. 84–100. Springer, Cham (2015)
18. 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 (2014). https://doi.org/10.1145/2600821.2600835
19. De Smedt, J., De Weerdt, J., Vanthienen, J., Poels, G.: Mixed-paradigm process modeling with intertwined state spaces. Bus. Inf. Syst. Eng. **58**(1), 19–29 (2016). https://doi.org/10.1007/s12599-015-0416-y
20. 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). https://www.riverpublishers.com/pdf/ebook/chapter/RP_9788793519817C5.pdf
21. Debois, S., Hildebrandt, T., Marquard, M., Slaats, T.: Hybrid Process Technologies in the Financial Sector: The Case of BRFkredit, pp. 397–412. Springer, Cham (2018)
22. Debois, S., Hildebrandt, T., Slaats, T.: Hierarchical declarative modelling with refinement and sub-processes. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) Business Process Management, pp. 18–33. Springer, Cham (2014)
23. 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)
24. 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). https://doi.org/10.1109/EDOCW.2014.27
25. 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)
26. Debois, S., Hildebrandt, T.T., Slaats, T.: Replication, refinement & reachability: complexity in dynamic condition-response graphs. Acta Inform. **55**(6), 489–520 (2018). https://doi.org/10.1007/s00236-017-0303-8
27. Di Ciccio, C., Maggi, F.M., Mendling, J.: Efficient discovery of target-branched declare constraints. Inf. Syst. **56**(C), 258–283 (2016). https://doi.org/10.1016/j.is.2015.06.009
28. Di Ciccio, C., Marrella, A., Russo, A.: Knowledge-intensive processes: characteristics, requirements and analysis of contemporary approaches. J. Data Semant. **4**(1), 29–57 (2015). https://doi.org/10.1007/s13740-014-0038-4
29. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Inf. Softw. Technol. **50**(12), 1281–1294 (2008)

30. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer, Cham (2013). https://doi.org/10.1007/978-3-642-33143-5
31. 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)
32. 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)
33. 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)
34. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events. J. Mach. Learn. Res. **10**, 1305–1340 (2009)
35. 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)
36. 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). https://doi.org/10.1109/EDOC.2011.35
37. Hildebrandt, T., Mukkamala, R.R., Slaats, T.: Nested dynamic condition response graphs. In: Proceedings of Fundamentals of Software Engineering (FSEN) (2011). http://www.itu.dk/people/rao/pubs_accepted/fsenpaper.pdf
38. Hildebrandt, T., Mukkamala, R.R., Slaats, T., Zanitti, F.: Contracts for cross-organizational workflows as timed dynamic condition response graphs. J. Log. Algebr. Program. **12**, 12 (2013). https://doi.org/10.1016/j.jlap.2013.05.005
39. 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). https://doi.org/10.4204/EPTCS.69.5
40. Hildebrandt, T.T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs. arXiv preprint arXiv:1110.4161 (2011)
41. 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: Proceedings of WS-FM'10, pp. 1–24. Springer, Berlin (2011)
42. Kong, Z., Jones, A., Belta, C.: Temporal logics for learning and detection of anomalous behavior. IEEE Trans. Autom. Control **62**(3), 1210–1222 (2016)
43. 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)
44. La Rosa, M., Reijers, H.A., Van Der Aalst, W.M., Dijkman, R.M., Mendling, J., Dumas, M., García-Ba nuelos, L.: Apromore: an advanced process model repository. Expert Syst. Appl. **38**(6), 7029–7040 (2011)
45. 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)
46. 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)
47. Maggi, F.M., Ciccio, C.D., Francescomarino, C.D., Kala, T.: Parallel algorithms for the automated discovery of declarative process models. Inf. Syst. **74**, 136–152 (2018)
48. 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). https://doi.org/10.1007/978-3-642-23059-13
49. 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). https://doi.org/10.1109/CIDM.2011.5949297
50. Maggi, F.M., Slaats, T., Reijers, H.A.: The automated discovery of hybrid processes. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) Business Process Management, pp. 392–399. Springer, Cham (2014)
51. Manataki, A., Fleuriot, J., Papapanagiotou, P.: A workflow-driven formal methods approach to the generation of structured checklists for intrahospital patient transfers. IEEE J. Biomed. Health Inf. **21**(4), 1156–1162 (2016)
52. Marquard, M., Shahzad, M., Slaats, T.: Web-based modelling and collaborative simulation of declarative processes. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) Business Process Management, pp. 209–225. Springer, Cham (2015)
53. Montali, M.: Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach. Lecture Notes in Business Information Processing, vol. 56. Springer (2010)
54. Montali, M., Pesic, M., van der Aalst, W.M., Chesani, F., Mello, P., Storari, S.: Declarative specification and verification of service choreographiess. ACM Trans. Web **4**(1), 3 (2010)
55. Mukkamala, R.: A formal model for declarative workflows: dynamic condition response graphs. it University of Copenhagen. Ph.D. thesis, IT University of Copenhagen (2012)
56. Mukkamala, R.R.: A formal model for declarative workflows—dynamic condition response graphs. Ph.D. thesis, IT University of Copenhagen (2012)
57. Mukkamala, R.R., Hildebrandt, T., Tøth, J.B.: The resultmaker online consultant: From declarative workflow management in practice to LTL. In: Proceedings of the 2008 12th Enterprise Distributed Object Computing Conference Workshops, EDOCW '08, pp. 135–142. IEEE Computer Society, Washington, DC, USA (2008). https://doi.org/10.1109/EDOCW.2008.57
58. 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)
59. 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)
60. Nielsen, M., Plotkin, G., Winskel, G.: Petri nets, event structures and domains. In: Kahn, G. (ed.) Semantics of Concurrent Computation. Lecture Notes in Computer Science, vol. 70, pp. 266–284. Springer, Berlin (1979). https://doi.org/10.1007/BFb0022474
61. Object Management Group: Case Management Model and Notation, version 1.0. Webpage (2014). http://www.omg.org/spec/CMMN/1.0/PDF
62. 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
63. Papapanagiotou, P., Fleuriot, J.: Workflowfm: a logic-based framework for formal process specification and composition. In: Interna-

tional Conference on Automated Deduction, pp. 357–370. Springer (2017)
64. 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)
65. 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), p. 287. IEEE (2007)
66. 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). https://doi.org/10.1109/EDOC.2007.25
67. Popova, V., Fahland, D., Dumas, M.: Artifact lifecycle discovery. Int. J. Coop. Inf. Syst. **24**(01), 1550001 (2015). https://doi.org/10.1142/S021884301550001X
68. Rozinat, A., Van der Aalst, W.M.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1), 64–95 (2008)
69. Sadiq, S., Sadiq, W., Orlowska, M.: Pockets of flexibility in workflow specification. In: Kunii, H.S., Jajodia, S., Sølvberg, A. (eds.) Conceptual Modeling—ER 2001. Lecture Notes in Computer Science, vol. 2224, pp. 513–526. Springer, Berlin (2001)
70. 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. Softw. Syst. Model. **14**(3), 1127–1157 (2015). https://doi.org/10.1007/s10270-014-0397-1
71. Schönig, S., Cabanillas, C., Jablonski, S., Mendling, J.: A framework for efficiently mining the organisational perspective of business processes. Decis. Support Syst. **89**, 87–97 (2016)
72. Schönig, S., Zeising, M.: The DPIL framework: tool support for agile and resource-aware business processes. BPM (Demos) **1418**, 125–129 (2015)
73. 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: Abramowicz, W., Paschke, A. (eds.) Business Information Systems, pp. 190–205. Springer, Cham (2018)
74. Shalev-Shwartz, S., Ben-David, S., Press, C.U.: Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press (2015). https://books.google.co.uk/books?id=tBVCtAEACAAJ
75. Slaats, T.: Flexible process notations for cross-organizational case management systems. Ph.D. thesis, IT University of Copenhagen (2015)
76. Slaats, T.: Declarative and hybrid process discovery: recent advances and open challenges. J. Data Semant. **9**(1), 3–20 (2020). https://doi.org/10.1007/s13740-020-00112-9
77. Slaats, T., Debois, S., Hildebrandt, T.: Open to change: a theory for iterative test-driven modelling. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) Business Process Management, pp. 31–47. Springer, Cham (2018)
78. Slaats, T., Mukkamala, R.R., Hildebrandt, T., Marquard, M.: Exformatics declarative case management workflows as DCR graphs. In: Daniel, F., Wang, J., Weber, B. (eds.) Business Process Management, pp. 339–354. Springer, Berlin (2013)
79. Slaats, T., Schunselaar, D.M.M., Maggi, F.M., Reijers, H.A.: The semantics of hybrid process models. In: Debruyne, C., Panetto, H., Meersman, R., Dillon, T., Kühn, E., O'Sullivan, D., Ardagna, C.A. (eds.) On the Move to Meaningful Internet Systems: OTM 2016 Conferences, pp. 531–551. Springer, Cham (2016)
80. Smedt, J.D., Weerdt, J.D., Vanthienen, J.: Fusion miner: process discovery for mixed-paradigm models. Decis. Support Syst. **77**, 123–136 (2015)
81. Tijs Slaats: DisCoveR. https://github.com/tslaats/DisCoveR (2020)
82. van der Aalst, W., Pesic, M., Schonenberg, H., Westergaard, M., Maggi, F.M.: Declare. Webpage (2010). http://www.win.tue.nl/declare/
83. Van Der Aalst, W.: Process Mining: Discovery, Conformance and Enhancement of Business Processes, vol. 2. Springer (2011)
84. van der Aalst, W.M.P., van Hee, K.M.: Workflow Management: Models, Methods, and Systems. MIT Press (2002)
85. Van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004)
86. 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 (2006)
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: Mendling, J., Weidlich, M., Weske, M. (eds.) Business Process Modeling Notation, Lecture Notes in Business Information Processing, vol. 67, pp. 14–15. Springer, Berlin (2010). https://doi.org/10.1007/978-3-642-16298-5_3
89. Weske, M.: Business Process Management—Concepts, Languages, Architectures, 2nd edn. Springer (2012). https://doi.org/10.1007/978-3-642-28616-2
90. Westergaard, M., Maggi, F.M.: Looking into the future. In: Meersman, R., Panetto, H., Dillon, T., Rinderle-Ma, S., Dadam, P., Zhou, X., Pearson, S., Ferscha, A., Bergamaschi, S., Cruz, I.F. (eds.) On the Move to Meaningful Internet Systems: OTM 2012, pp. 250–267. Springer, Berlin (2012)
91. Westergaard, M., Slaats, T.: Mixing paradigms for more comprehensible models. In: Daniel, F., Wang, J., Weber, B. (eds.) Business Process Management, pp. 283–290. Springer, Berlin (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. Int. J. Comput. Assist. Radiol. Surg. **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: 2014 International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 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. Softw. Syst. Model. **14**(3), 1081–1103 (2015). https://doi.org/10.1007/s10270-013-0356-2