DCR4Py: A PM4Py Library extension for Declarative Process Mining in Python

Simon V.H. Hermansen[1], Ragnar Jónsson[1], Jonas L. Kjeldsen[1], Tijs Slaats[2], Vlad Paul Cosma[2] and Hugo A. López[1]

[1]*Technical University of Denmark, Richard Petersens Plads, 321, 2800 Kgs. Lyngby, Denmark*
[2]*University of Copenhagen, Copenhagen, Denmark*

Abstract

DCR4Py is the first open-source library to offer a broad range of features and up-to-date algorithms for Dynamic Condition Response Graphs. It extends the popular PM4Py library with a new declarative language and matches PM4Py's GPL3 license, design, installation steps, maturity, and performance. Our key contribution consists of an open-source implementation in Python of all existing discovery and conformance-checking algorithms for DCR graphs, as well as import and export capabilities, visualization, and conversion, all in a single, well-documented, open-access, easy-to-use library that closely follows the research literature definitions and nomenclature.

Keywords

DCR Graphs, Declarative process mining, PM4Py, Open-source software

Metadata description	Value
Tool name	DCR4Py
Current version	1.0 (forked from PM4Py 2.7.11)
Legal code license	GPLv3
Languages, tools and services used	Python 3.1X, GraphViz 0.20
Supported operating environment	Microsoft Windows, GNU/Linux
Source code repository	https://github.com/paul-cvp/pm4py-dcr/tree/feature/dcr_in_pm4py_revised
Download/Demo URL	(Source code repository)/notebooks/dcr_tutorial.ipynb
Documentation URL	https://paul-cvp.github.io/dcr4pydocs/
Screencast video	https://youtu.be/fGZw2X-Wfc0

1. Introduction

Process mining [1] aims to optimize business processes by using event data to discover and analyze process models. As processes become more digitized, they include not only structured production applications but also flexible knowledge-intensive fields [2], diversifying their event data. Therefore it is not enough to analyze only the imperative, procedural, perspective of your business process, one needs to also consider the declarative, rule-based, perspective. Declarative

ICPM 2024 Tool Demonstration Track, October 14-18, 2024, Kongens Lyngby, Denmark
slaats@di.ku.dk (T. Slaats); vco@di.ku.dk (V. P. Cosma); hulo@dtu.dk (H. A. López)
http://lopezacosta.net (H. A. López)
0000-0001-6244-6970 (T. Slaats); 0000-0001-8022-6402 (V. P. Cosma); 0000-0001-5162-7936 (H. A. López)


process models can capture unstructured processes more concisely than imperative models and have been proven to capture better process insights for knowledge-intensive processes [2], achieving greater levels of optimization via process mining. Process mining tools[1] aim to easily onboard practitioners and researchers, increase the dissemination of new ideas, and speed up the adoption of emerging techniques. Furthermore, as process mining positions itself as the bridge between data science and process science it also increases demand for tool interoperability concerning process modeling, verification, machine learning, and statistical analysis.

Dynamic Condition Response (DCR) Graphs [3] are a relatively new modeling notation that belongs to the declarative paradigm. From its core definition, DCR Graphs have evolved to include six relation types as well as hierarchy, through nestings [4], and to capture other process dimensions, such as time [5] and roles [3]. In addition, recent work on process discovery [6, 4], conformance checking [7], and transformation to Petri Nets [8] has made the DCR notation practically interesting. However, tool support for DCR either covers broad functionality with closed source software [9] or provides open source code showcasing narrow functionality coded in a variety of programming languages and lacking continuous maintenance and support.

This paper proposes DCR4Py as the first broad functionality open source Python library developed as a seamless extension to the popular PM4Py (Process Mining for Python) project [10] using the same code structure and licenses. DCR4Py is an up-to-date implementation of DCR Graphs complete with process discovery, rule and alignment-based conformance checking, and a transformation to Petri Nets. Additional functionality includes execution semantics, visualization, and import/export capabilities via XML files compatible with both the DCR Solutions Portal [9] and the DCR-js modeling tool [11]. By extending a mature well-established library we enable users already familiar with PM4Py to get started with process mining with DCR graphs. As the library is open source it encourages the community to develop new extensions and algorithms for DCR Graphs. As DCR4Py is an extension of PM4Py, it allows for the reuse of the same installation requirements, automatic documentation generation, and testing.

Related work Tool support for declarative notations already exists in PM4Py for DECLARE [2, 12] and Log-Skeletons [13]. In addition DECLARE benefits from dedicated software which exposes a wide range of features, such as RUM [12] and Declare4Py [14]. PromTools [15] offers a no-code alternative to PM4Py where plug-ins for DECLARE [16] and Log-Skeletons [13] exist. Unlike DECLARE, DCR Graphs have been adopted commercially through the DCR Portal [9]. However, while freely available for academic use, the portal is closed source. Open-source repositories exist for all notable DCR Graphs algorithms [6, 7, 4, 8], but each repository is narrow in scope, and not all are actively maintained. For example, the original DisCoveR repository written in Java lacks the newer extensions for nestings and roles. DCR-js [11] allows users to model DCR Graphs.

The remainder of the paper is structured as follows: section 2 briefly defines DCR Graphs. In section 3 we present the new features DCR4Py brings into PM4Py. Next in section 4 we discuss the tool's maturity, before concluding in section 5.

[1] processmining.org/software.html/

2. What is a DCR graph?

The DCR Graphs [3] modeling notation represents events as nodes and relations as edges in a directed graph. Events can be atomic, or part of a hierarchy (nestings or subprocesses), and can be associated with roles. Events have a notion of state through a three-tuple binary marking (included, executed, pending). Directed relations either constrain or affect events in the direction of the edge. The constraining condition ($\rightarrow\!\bullet$) and milestone ($\rightarrow\!\diamond$) relations between two events A and B limit the execution of B based on the marking of A. The include ($\rightarrow\!+$), exclude ($\rightarrow\!\%$), response ($\bullet\!\rightarrow$) and no-response ($\bullet\!\rightarrow\!\!\!\!\times$) effect relations change the marking of B after the execution of A. An event is enabled if it is included, all conditions have been executed and all milestones are not pending. The effect of executing an enabled event is that its marking becomes executed and all effect relations from that event apply their changes to the marking of the target event. Finally, we say that a DCR Graph is accepting when none of its events are both included and pending. The condition and response relations can also be timed with delays on conditions ($\overset{k}{\rightarrow\!\bullet}$) and deadlines on responses ($\bullet\!\overset{d}{\rightarrow}$), resulting in Timed DCR Graphs. Figure 2 shows a prototypical DCR graph.

Event logs are defined as a set of traces and a trace is a finite sequence of events. A trace is a run of a DCR Graph if only enabled events are executed, and at the end of the trace, the DCR Graph is accepting. If all log traces are accepting then the DCR Graph perfectly fits the log.

3. Overview of DCR4Py Features

A brief overview of the library can be seen in Figure 1. DCR4Py supports the main process mining tasks of conformance checking and process discovery for DCR Graphs. This includes a Python implementation of both alignment [7], a novel rule-based conformance algorithm, and of the DisCoveR algorithm [6]. The DisCoveR algorithm is extended with the mining of nestings [4], roles, time [5], and initially pending events. The latter 3 are new contributions. The `objects` component exposes the existing conversion of DCR Graphs to Petri Nets [8] and new Python object definitions of DCR Graphs, execution semantics, and import/export capabilities. Finally `visualizations` for DCR Graphs are implemented through reuse of the existing PM4Py functionality.

DCR Graph class variants. The *core* DCR Graph class, based on the original definition from [3], consists of events, markings, and the include, exclude, condition, and response relations. Four main variants inherit from the core object. The *Distributed* variant covers the distributed

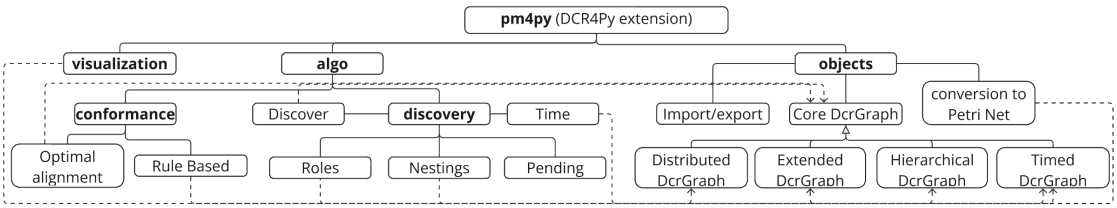


Figure 1: DCR4Py features and dependencies in PM4Py (classes in bold are PM4Py classes)

definition [3], the *Extended* variant covers new relation extensions [17], the *Hierarchical* one covers nestings [4], and the *Timed* variant covers the time perspective [5]. Execution semantics are also extended for the *Distributed*, *Extended*, and *Timed* DCR Graphs classes. *Hierarchical* DCR Graphs map to *Extended* DCR Graphs, therefore do not have execution semantics.

Process Discovery. Process discovery for core DCR Graphs in DCR4Py consists of the DisCoveR algorithm [6]. The algorithm is called via the `pm4py.discover_dcr` method. Additional post-processing can be called using the `post_process` parameter. *Roles*, *pending*, and *time* enhance the richness of the discovered graphs, while *nestings* improve their simplicity. Roles are mined by associating events with the `org:resource` attribute of event logs. Pending events are mined to discover graphs with a non-accepting initial marking. Time is mined as delays and deadlines, according to the execution semantics of conditions and responses respectively.

Rule and Alignment-based Conformance checking. DCR4Py implements both rule-based and alignment-based conformance [7]. Rule-based conformance checking for DCR Graphs performs a replay of traces in a given event log. The algorithm utilizes the execution semantics to determine deviations. If any deviations occur, the algorithm notes the deviation and what caused it. The implementation utilizes a `handleChecker`, that determines the kind of DCR Graphs, to allow for correct notation of deviations, and currently supports core and distributed DCR Graphs. The algorithm is called via the `pm4py.conformance_dcr` method. DCR4Py also features the optimal alignment algorithm proposed in [7]. Alignment-based conformance checking directly connects deviations between the expected process flow, as per the DCR Graph, and the actual execution, as recorded in the logs. The `TraceAlignment` class aligns each trace in the log with the paths in the DCR graph. The computation of the alignment is performed by the `Alignment` class. The `Performance` class, calculates fitness as one minus the ratio of the optimal alignment cost to the worst-case alignment cost. The alignment algorithm is called via the `pm4py.optimal_alignment_dcr` method.

Conversion, visualization, import/export. The conversion to Petri Nets [8] is exposed via the `pmpy.convert_to_petri_net` function when passing a DCR Graph object. Visualizations of DCR Graphs, as shown in fig. 2, are provided via the `pm4py.view_dcr` and `pm4py.save_vis_dcr` functions. The functions make use of Graphviz, which is a graph visualization library already used in PM4Py. Finally, DCR4Py provides import/export capabilities for DCR Graphs to maintain interoperability with the existing DCR-js [11] and DCR Portal [9] modeling tools. Detailed instructions can be found in the notebook `dcr_tutorial`.

4. Maturity of the tool

Benchmarks. We compare the runtime of DCR4Py against the existing DECLARE, Log Skeletons, Petri Net, and DFG implementations in PM4Py. We selected Sepsis, Road Traffic, BPIC13i from [18] and Dreyers from [7]. For process discovery, we see in fig. 3a that DCR4Py is the fastest overall, with DECLARE being the slowest on Road traffic. For conformance checking we take 10% of the log to discover a model and run the full log through the checker. For the rule checker, we see in fig. 3b that DCR4Py is, on average the slowest, and slowest overall on the Road traffic log. A similar pattern is seen in fig. 3c for the alignment checker.

Installation, testability, and reuse. DCR4Py uses the same installation steps and require-

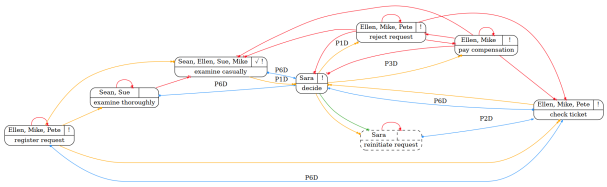
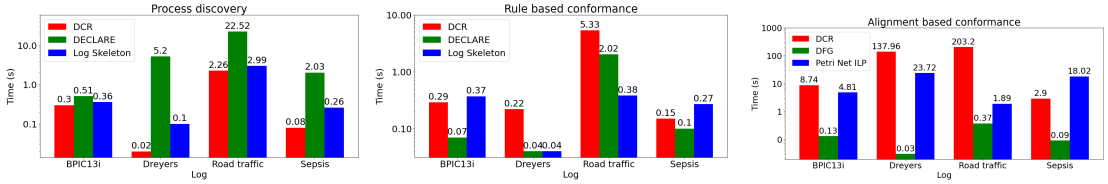


Figure 2: DCR4Py Visualization



(a) Discovery (b) Rule conformance (c) Alignment conformance

Figure 3: Runtime performance (log scale) of DCR4Py compared to existing PM4Py algorithms.

ments as PM4Py. Unit tests for the newly added components are integrated into the PM4Py library, thus guaranteeing the testability of the new features and the integrity of the existing functionality. Automatic documentation generation is supported through the reuse of existing PM4Py functionality. As an academic tool, subsets of DCR4Py were successfully used to showcase new research [19], support student projects, and in the classroom setting (see `notebooks/exercise-sepsis` in the repository).

5. Conclusion

We introduced DCR4Py, an extension of the process mining library PM4Py supporting process discovery, conformance-checking, visualization, conversion and I/O capabilities for DCR Graphs. By extending PM4Py's API we lower the entry barrier for declarative process mining for researchers and practitioners.

Limitations and future work. We will look at adding subprocesses, at multi-threaded conformance checking to improve performance and object-centric mining of DCR Graphs. As a forked PM4Py project, we periodically synchronize DCR4Py to ensure it can be merged into PM4Py via a pull request. We are in dialogue with the PM4Py owners to achieve this.

Acknowledgments

We thank Axel Christfort and Thomas Hildebrandt for their contribution to the codebase and the underlying theory. This work was supported by the research grant "Center for Digital ComplianceE (DICE)" (VIL57420) from VILLUM FONDEN.

References

[1] W. van der Aalst, Process Mining, 2016.

[2] W. M. van Der Aalst, M. Pesic, H. Schonenberg, Declarative workflows: Balancing between flexibility and support, Computer Science-Research and Development 23 (2009) 99–113.
[3] T. Hildebrandt, R. Mukkamala, Declarative event-based workflow as distributed dynamic condition response graphs, EPTCS, 2010, pp. 59–73.
[4] V. P. Cosma, A. K. F. Christfort, T. T. Hildebrandt, X. Lu, H. A. Reijers, T. Slaats, Improving simplicity by discovering nested groups in declarative models, in: Advanced Information Systems Engineering, 2024, pp. 440–455.
[5] T. Hildebrandt, R. R. Mukkamala, T. Slaats, F. Zanitti, Contracts for cross-organizational workflows as timed dynamic condition response graphs, The Journal of Logic and Algebraic Programming 82 (2013) 164–185.
[6] C. O. Back, T. Slaats, T. T. Hildebrandt, M. Marquard, Discover: Accurate & efficient discovery of declarative process models, 2020. arXiv:2005.10085.
[7] A. K. F. Christfort, T. Slaats, Efficient optimal alignment between dynamic condition response graphs and traces, in: Business Process Management, 2023, pp. 3–19.
[8] V. P. Cosma, T. T. Hildebrandt, T. Slaats, Transforming dynamic condition response graphs to safe petri nets, in: Application and Theory of Petri Nets and Concurrency, 2023.
[9] M. Marquard, M. Shahzad, T. Slaats, Web-based modelling and collaborative simulation of declarative processes, in: BPM, 2015, pp. 209–225.
[10] A. Berti, S. van Zelst, D. Schuster, Pm4py: A process mining library for python, Software Impacts 17 (2023) 100556.
[11] L. Tamo, A. Abbad-Andaloussi, D. Trinh, H.-A. López, An open-source modeling editor for declarative process models, in: CoopIS Demos, 2023.
[12] A. Alman, I. Donadello, F. M. Maggi, M. Montali, Declarative process mining for software processes: The rum toolkit and the declare4py python library, in: International Conference on Product-Focused Software Process Improvement, Springer, 2023, pp. 13–19.
[13] H. Verbeek, The log skeleton visualizer in prom 6.9: the winning contribution to the process discovery contest 2019, International Journal on STTT 24 (2022) 549–561.
[14] I. Donadello, F. Riva, F. M. Maggi, A. Shikhizada, Declare4py: A python library for declarative process mining, in: CEUR Workshop Proceedings, 2022, pp. 117–121.
[15] B. F. Van Dongen, A. K. A. de Medeiros, H. M. Verbeek, A. Weijters, W. M. van Der Aalst, The prom framework: A new era in process mining tool support, in: ICATPN, 2005, pp. 444–454.
[16] F. M. Maggi, Declarative process mining with the declare component of prom., BPM (Demos) 1021 (2013).
[17] T. T. Hildebrandt, H. Normann, M. Marquard, S. Debois, T. Slaats, Decision modelling in timed dynamic condition response graphs with data, in: International Conference on Business Process Management, Springer, 2021, pp. 362–374.
[18] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, F. Maggi, A. Marrella, M. Mecella, A. Soo, Data underlying the paper: Automated discovery of process models from event logs: Review and benchmark, 2019. URL: https://data.4tu.nl/datasets/a24f253c-722d-4a3e-9e92-f1a46dbb7473/1.
[19] P. Cosma, Declarative process models as explainable and verifiable Artificial Intelligence, Ph.D. thesis, 2024.