The DyLoPro Library: Comprehensively Profiling the Dynamics of Event Logs by Means of Visual Analytics

Brecht Wuyts[1], Hans Weytjens[1], Seppe vanden Broucke[1,2] and Jochen De Weerdt[1]

[1]LIRIS, Faculty of Economics and Business, KU Leuven, Leuven, Belgium
[2]Department of Business Informatics and Operations Management, Ghent University

Abstract

In Process Mining, a notable issue arises due to the discrepancy between prevailing techniques, which assume constancy in business processes, and the actuality of modern business processes characterized by frequent changes. As this discrepancy can lead to biased results, it is crucial that such drifts are detected and understood, prior to applying other PM techniques on the corresponding event logs. However, such drifts in event logs can manifest in different forms (sudden, gradual, recurring, incremental) and occur in many different process perspectives (control-flow, data, performance). This necessitates approaches that holistically and efficiently delve into the temporal dynamics present in event logs. Therefore, in this paper, we present the *DyLoPro* Python library, a visual analytics tool that enables PM practitioners to efficiently and comprehensively explore event log dynamics over time, and which caters to all kinds of event logs. This demo paper aims to familiarize Process Mining practitioners with the DyLoPro library, showcasing its main capabilities, and encouraging the Process Mining field to take the time dimension into greater consideration in all stages of their projects.

Keywords

Process Mining, Event Logs, EDA, DyLoPro, Visual Analytics, Python

1. Introduction

Event logs violating Process Mining [1] techniques' ubiquitous assumption of stationary processes, will induce a significant, yet oftentimes unnoticed bias in their results, potentially resulting in flawed conclusions. Therefore, it is imperative that the dynamics are analyzed and potential drifts are identified prior to the application of PM techniques. However, many different forms of drift exist, and drifts can occur in different process perspectives (control-flow, resource, data, performance). This multi-perspective property, together with the sequential nature of event logs, complicates the process of studying event log dynamics over time. One intuitive and powerful manner in which event log dynamics over time could be analyzed, is by means of visualizations. Dotted charts [2] allow for such a multi-dimensional exploration of the dynamics over time, and are offered by several open-source PM tools such as ProM [3] and PM4PY [4]. However, dotted charts visualize every event, which makes it hard and cumbersome to entirely visualize the dynamics of large real-life event logs. Another interesting tool that

brecht.wuyts@kuleuven.be (B. Wuyts)
0000-0001-6079-7515 (B. Wuyts); 0000-0003-4985-0367 (H. Weytjens); 0000-0002-8781-3906 (S. v. Broucke); 0000-0001-6151-0504 (J. D. Weerdt)

provides functionality to visualize dynamics over time, is the Performance Spectrum Miner (PSM) [5]. The PSM focuses solely on visualizing the evolution of the time elapsed between a pair of two consecutive activities. Nevertheless, to the best of our knowledge, no comprehensive tool has been developed to efficiently explore the dynamics in an event log over time.
This gap is addressed by the recently introduced *Dynamic Log Profiling* (***DyLoPro***) framework [6], and its complementary Python library, which we present in this paper. On the one hand, the *visual analytics* [6] framework enables PM practitioners to conduct a *comprehensive* analysis of the dynamics of event logs over time, from various process perspectives, both individually and combined with the performance perspective. The complementary *DyLoPro* library, on the other hand, provides users with a ready-to-use and user-friendly Python implementation of this framework, thereby empowering users to leverage the framework's *comprehensive* visualization capabilities in an *efficient* way, too.

2. *DyLoPro* - the Python Library

2.1. Architecture and Intended Use

The *DyLoPro* library implements and extends the eponymous framework [6], whose comprehensiveness is achieved through the incorporation of the main process perspectives - the control-flow, data (including resources) and performance, along two orthogonal dimensions of *log concepts* and *representation types*. Accordingly, the *DyLoPro* library provides functionality to construct and visualize time series, i.e. the log dynamics, for a variety of *log concepts*, while for each log concept, the dynamics can be represented using several different *representation types*. All these visualization capabilities can be accessed and customized by invoking the appropriate methods on the initialized `DynamicLogPlots` instance, and specifying the desired values for their parameters, respectively. Accordingly, first of all, a `DynamicLogPlots` instance has to be initialized. The `DynamicLogPlots` class provides one single source of access to the library's functionalities, and thereby serves as the interface between the users' Python environment and *DyLoPro*'s underlying computational logic. Initializing a `DynamicLogPlots` instance involves specifying a number of required arguments, including the event log that should be loaded into a *pandas* [7] DataFrame object, and a number of optional arguments. *Secondly*, after specifying the arguments, the software will first validate the arguments. If an error was found, the software will raise an adequate error, with a dedicated message describing the error and how to resolve it. If no errors are found, the log is preprocessed in an internal format that allows *DyLoPro* to efficiently compute and visualize all aggregations on an on-demand basis, as illustrated in Fig. 1[1]. For a more elaborate explanation detailing the way in which the framework is implemented into the library and consequently how to access *DyLoPro*'s extensive capabilities, please refer to the '**Get Started**' section of the *project description*[2]. A brief demonstration video can be found at https://youtu.be/Z9hqpkGbta0.

[1]Data used: https://doi.org/10.4121/uuid:d06aff4b-79f0-45e6-8ec8-e19730c248f1
[2]See Footnote 5 or 6.

2.2. Functionality

After having successfully initialized a `DynamicLogPlots` instance, all functionality can easily be accessed by invoking the corresponding methods on this instance *(e.g. Fig. 1.* As already mentioned, the DyLoPro library implements the identically named framework, proposed in [6], which consists of three stages, (1) *log discretization*, (2) *domain definition*, and (3) *time series construction & visualization*.
Log discretization boils down to subdividing the event log into equal length[3] sublogs. *Domain definition* amounts to defining how to capture and represent event log dynamics over time. This involves defining *log concepts*, which refer to the primary dimensions used to capture the dynamics of event logs, and *representation types*, which determine how the dynamics of the event logs should be represented and analyzed for each log concept. The *domain definition*, i.e. the resulting *log concept - representation type combination*, translates into a unique domain-specific mapping function. Finally, *time series construction & visualization* basically comes down to applying that mapping function to each of the (chronologically ordered) sublogs of the *log discretization*, thereby yielding several time series.
The functionality provided by the framework is implemented as follows. Each of the six *log concepts* defined in [6] is assigned one or two dedicated plotting methods *(see Table 1)*. Given

Table 1
Note that the functionality can be continuously extended. The reader is referred to the documentation page for the most recent version of *DyLoPro*.

	Log Concept	*Method 1*	*Method 2*
1	Variants	`topK_variants_evol()`	`variants_evol()`
2	Directly-Follows Relations	`topK_dfr_evol()`	`dfr_evol()`
3	Categorical Case Feature	`topK_categorical_caseftr_evol()`	/
4	Numerical Case Features	`num_casefts_evol()`	/
5	Categorical Event Feature	`topK_categorical_eventftr_evol()`	/
6	Numerical Event Features	`num_eventfts_evol()`	/

a certain *log concept* and an associated plotting method, both the *log discretization* and the *concept*'s *representation type* can simply be configured by customizing the method's arguments. More specifically, the *log discretization* can be configured by specifying the `frequency` parameter *(e.g. 'weekly')*, which determines the frequency by which cases are grouped together, and the `case_assignment` parameter *(e.g. 'first_event')*, which determines the condition upon which each case is assigned to one of the time periods. The *representation type* can be chosen by specifying each method's `plt_type` parameter. To illustrate, the third command in Fig. 1 generates, for the 10 most frequently occurring variants (`max_k=10`), time series according to the 'throughput time' *representation type* (`plt_type='type_tt'`), while each consecutive value in each of the time series is calculated by aggregating[4] over a sublog covering one week (`frequency='weekly'`), and containing all cases of which the first event occurred during that specific week (`case_assignment='first_event'`). The third and final stage, *time series con-*

[3]'Equal length' means that each sublog covers an equal-length time period.
[4]In this case, the aggregation corresponds to computing the mean for each sublog (`numeric_agg='mean'`).

struction & visualization, simply corresponds to the execution of the specified plotting method. Table 1 lists all six concepts introduced in [6], together with their visualization methods. In addition to the common hyperparameters just discussed, each of the plotting methods is also equipped with additional configuration parameters, providing users with even more flexibility on the fly. For more information about *DyLoPro*'s methods and their parameters, the reader is referred to the documentation page *(see Section 3)*. The *DyLoPro* library not only implements the framework *(see Table 1)*, but also extends it, with new functionality added over time when needed. Several auxiliary methods are also provided to, e.g., enable the user to adjust the initialized `DynamicLogPlots` instance after initialization, or for example to query DataFrames containing the string representations of the most frequently occurring variants or directly-follows relations. Again, please refer to the extensive documentation page for an up-to-date overview of all of *DyLoPro*'s functionality.

```python
# Import the DyLoPro library
import DyLoPro as dlp
# Initializing a DynamicLogPlots object
plot_object = dlp.DynamicLogPlots(event_log,
                                  case_id_key,
                                  activity_key,
                                  timestamp_key,
                                  categorical_casefeatures,
                                  numerical_casefeatures,
                                  categorical_eventfeatures,
                                  numerical_eventfeatures,
                                  start_date,
                                  end_date,
                                  outcome)
# Invoking a visualization method on plot_object
plot_object.topK_variants_evol(time_unit='days',
                               frequency='weekly',
                               case_assignment='first_event',
                               plt_type = 'type_tt',
                               numeric_agg='mean',
                               max_k = 10,
                               xtr_outlier_rem = False)
```

Python

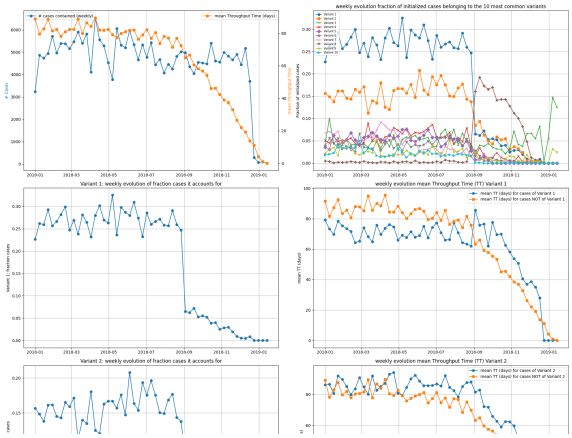

Figure 1: Example DyLoPro code: after having correctly initialized a `DynamicLogPlots` object, all of *DyLoPro*'s capabilities can easily be accessed by invoking the desired methods on the initialized object. All arguments used for initializing the object are assumed to be already specified. The invoked visualization method generates, i.a., for each of the *10* most frequently occurring variants, the *weekly mean throughput time* of cases pertaining to that particular variant, vs. cases pertaining to all other variants (and of which the *first event* occurred in that week).

3. *DyLoPro*'s Maturity

DyLoPro was first released on PyPI[5] on 18/06/2023. As we move forward, it will be continuously monitored, and new releases will be launched whenever necessary to ensure the best possible performance and features for its users. Moreover, *DyLoPro* is equipped with an encompassing GitHub repository[6]. This repository serves as a centralized hub for collaborative development and houses essential elements to foster a thriving open-source PM community. Additionally, *DyLoPro* offers an extensive Read the Docs page[7], which documents, i.a., how to initialize a `DynamicLogPlots` instance, and its visualization methods.

Furthermore, within a separate repository[8], we present case studies that leveraged *DyLoPro* to examine the dynamics of various public real-life event logs frequently referenced in PM literature. These case studies identify and further examine various remarkable patterns and problems in a number of these event logs. By increasing transparency and understanding of these intricate datasets, our case studies aim to improve the validity and accuracy of research relying on these event logs. As part of an ongoing effort, the repository will continue to grow with additional case studies of additional public event logs over time. Moreover, these case studies also contribute to an improved understanding of the use and accessibility of *DyLoPro*'s extensive array of plotting methods.

References

[1] W. M. P. van der Aalst, Process Mining: Data Science in Action, 2 ed., Springer, Heidelberg, 2016. doi:10.1007/978-3-662-49851-4.

[2] M. Song, W. Aalst, Supporting process mining by showing events at a glance, Proceedings of 17th Annual Workshop on Information Technologies and Systems (WITS 2007) (2007) 139–145.

[3] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, W. M. P. van der Aalst, The prom framework: A new era in process mining tool support, in: G. Ciardo, P. Darondeau (Eds.), Applications and Theory of Petri Nets 2005, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 444–454.

[4] A. Berti, S. van Zelst, W. Aalst, Process mining for python (pm4py): Bridging the gap between process- and data science, 2019.

[5] V. Denisov, E. Belkina, D. Fahland, W. Aalst, The performance spectrum miner: Visual analytics for fine-grained performance analysis of processes, 2018.

[6] B. Wuyts, H. Weytjens, S. vanden Broucke, J. De Weerdt, DyLoPro: Profiling the dynamics of event logs, in: C. Di Francescomarino, A. Burattin, C. Janiesch, S. Sadiq (Eds.), Business Process Management, Springer Nature Switzerland, Cham, 2023, pp. 146–162.

[7] Wes McKinney, Data Structures for Statistical Computing in Python, in: Stéfan van der Walt, Jarrod Millman (Eds.), Proceedings of the 9th Python in Science Conference, 2010, pp. 56 – 61. doi:10.25080/Majora-92bf1922-00a.

[5] https://pypi.org/project/DyLoPro

[6] https://github.com/BrechtWts/DyLoPro

[7] https://dylopro.readthedocs.io

[8] https://github.com/BrechtWts/DyLoPro_CaseStudies