PLG2: Multiperspective Process Randomization with Online and Offline Simulations

Andrea Burattin
University of Innsbruck, Austria
andrea.burattin@uibk.ac.at

Abstract. The evaluation of process mining algorithms requires, as any other data mining task, the availability of large amount of (real-world) data. Despite the increasing availability of such datasets, they are affected by many limitations: *in primis*, the absence of a "gold standard" (i.e., the reference model). This work extends an approach already available in the literature for the generation of random processes. Novelties have been introduced throughout the work which, in particular, involve the complete support for multiperspective models and logs (i.e., the control-flow perspective is enriched with time and data information) and for online settings (i.e., generation of multiperspective event streams and concept drifts). The proposed new framework is able to cover the spectrum of possible scenarios that can be observed in the real-world.

Keywords: Process mining; log generation; event stream; concept drift.

1 Introduction

Process mining [1] gained a lot of attention and is now considered an important field of research, bridging data mining and business process modeling/analysis. Particularly, the aim of process mining is to extract useful information from business process executions. In data mining, the term *gold standard* (also referred to as *ground truth*) typically indicates the "correct" answer to a mining task (i.e., the reference model). For example, in data clustering, the gold standard may represent the right (i.e., the target) assignment of elements to their corresponding clusters. Many times, referring to a gold standard is fundamental in order to properly evaluate the quality of mining algorithms. Several concepts, like *precision* or *recall*, are actually grounded on this idea. However in the context of business processes, companies are usually reluctant to publicly share their data for analysis purposes. Moreover, detailed information on their running processes (i.e., the reference models) are considered as company assets and, therefore, are kept even more private. An annual event, called *BPI challenge*, releases real world event logs. Despite the importance of this data, the logs are not accompanied with their corresponding gold standards. Moreover, they do not provide examples of all possible real world situations: many times, researchers and practitioners would like to test their algorithms and systems against specific conditions and, to this purpose, those event logs may not be enough.

In this paper we propose the new version of a tool already available [5] (PLG). The new tool, Processes and Logs Generator 2 (PLG2), can be used to randomly generate multiperspective process models and to simulate them, with the purpose of synthesizing multiperspective event logs. Moreover, the approach is tailored to the simulation of online settings, since it is possible to generate local evolutions of processes and event streams. Another feature allows to dynamically change the model underlying the stream, in order to produce concept drifts. PLG just allowed the generation of random control-flows and their simulation as static logs. PLG2 is implemented in a standalone Java application which is also accompanied by a set of APIs, useful for the programmatic definition of custom experiments.

The implemented components are reported in Fig. 1. The central part is responsible for the representation of a process model. In order to create a process, the user can import a file; randomly generate a new process; or evolve an existing process into a different one. Processes can also be exported to file. Starting from a process model, PLG2 can simulate it in order to create an event log or an event stream. Both these last components use a noise generator in order to add more realism to the generated data.

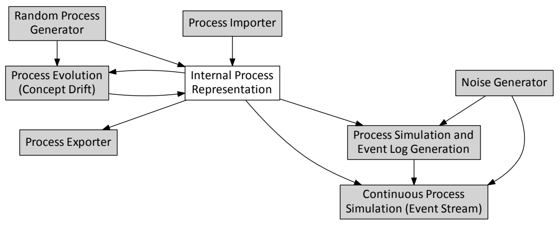

Random Process Generator
Process Importer
Process Evolution (Concept Drift)
Internal Process Representation
Noise Generator
Process Exporter
Process Simulation and Event Log Generation
Continuous Process Simulation (Event Stream)

Fig. 1: Components of PLG2.

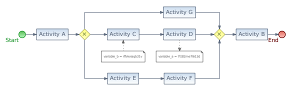
2 PLG2: Technical Details

Process Models. In PLG2, the internal structure of a process model is intuitively derived from the definition of a BPMN process model. A process is essentially an aggregation of *components*. Each component can be either a flow object, a sequence or a data object. Flow objects are: events (*start* or *end*); gateways (*exclusive* or *parallel*); and tasks. Sequences connect two flow objects. Data objects are associated with activities and can be *plain* or *dynamic*. A plain data object is a name-value pair. A dynamic data object has a value that can change every time it is required (i.e., it is dynamically generated by a script). Data objects can be *generated* or *required* by activities. This internal structure allows more flexibility. For example, it is now possible to load BPMN models generated with external tools, as long as the modeled components are available also in PLG2. Also, since we are restricting to non-ambiguous components, we can convert our processes into Petri nets. The generation of random processes is based on some workflow control-flow patters [2]: *(a)* WCP-1: direct succession of two activities; *(b)* WCP-2: parallel execution; *(c)* WCP-3: synchronization of parallel branches; *(d)* WCP-4: mutual execution; *(e)* WCP-5: convergence of branches; *(f)* WCP-21: ability to execute sub-processes repeatedly. By progressively combining these patterns we build a complete process model. The combination of these patterns is performed according to a predefined set of probabilistic rules.

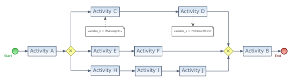
Detailed description of the generation rules is reported in the technical report [3]. Random data objects are generated and randomly connected to activities as well.

Process Simulation. The procedure for the generation of logs starting from business processes, basically, consists of a simulation engine running the "*play-out game*": the algorithm keeps a set of "enabled activities" and randomly picks one for the simulation (different probabilities are not supported yet). After that, it checks for new enabled activities, puts them into the enabled activities set, and repeats itself. To determine activities times and durations, the system checks for any of these parameters. If these are not reported, the activity is assumed to be instantaneous and to execute just after the previous. However, the user can specify these parameters as Python functions: `time_after(caseId)` and `time_lasted(caseId)`. Both functions are called by the simulator with the `caseId` parameter valued with the actual case id: this allows the functions to be case-dependent. `time_after(caseId)` returns the number of seconds to wait before the following activity starts. `time_ lasted(caseId)` returns the number of seconds that the activity lasts. This approach is extremely flexible: it is possible to define, for example, different durations for the same activity depending on which flow the current trace has followed. Regarding data objects, *generated* data objects are stored as attribute of the current activity, *required* are written as attributes for the preceding activity. *Plain* data objects are treated as fixed values (i.e., the simulation generates always the same value); *dynamic* data objects are actually Python scripts whose values are determined by the execution of the script itself. These scripts implement a `generate(caseId)` function which returns an integer or a string value. Please note that, also in this case, the function is called with the `caseId` parameter valued with the actual instance case id, providing the user with an in-depth, and case dependent, control over the generated values. There is no particular limit on the number of plain and dynamic data objects that a task can have, both required and generated. It is also possible to introduce simulated noise. Noise can be at different "levels": *(i)* at the trace level (involving trace organization); *(ii)* at the event level (involving events on the control-flow); *(iii)* at the data object level (involving the data perspective). The actual noise generation is driven by the parameters set by the user. The noise details for the trace and the event level have already been discussed in the literature and reported in details in [7]. The remaining technical details are described in [3].

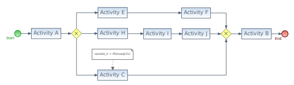
PLG2 is explicitly design for the simulation of online event streams [6]: the *play-out game* should continue infinitely and the underlying process model is allow to change. The stream definitions used are reported in [4, 6]. An event stream is just a sequence of events, each of them potentially belonging to different traces. From an implementation point of view, the idea is to create a socket, which is accepting connections from external clients. PLG2, then, "emits" (i.e., writes on the socket) the generated events. In order to generate a continuous stream, the user has to set two parameters: the maximum number of parallel instances running at the same time and the "time scale". The first parameter is



(a) Starting process model.



(b) First evolution.



(c) Second evolution.

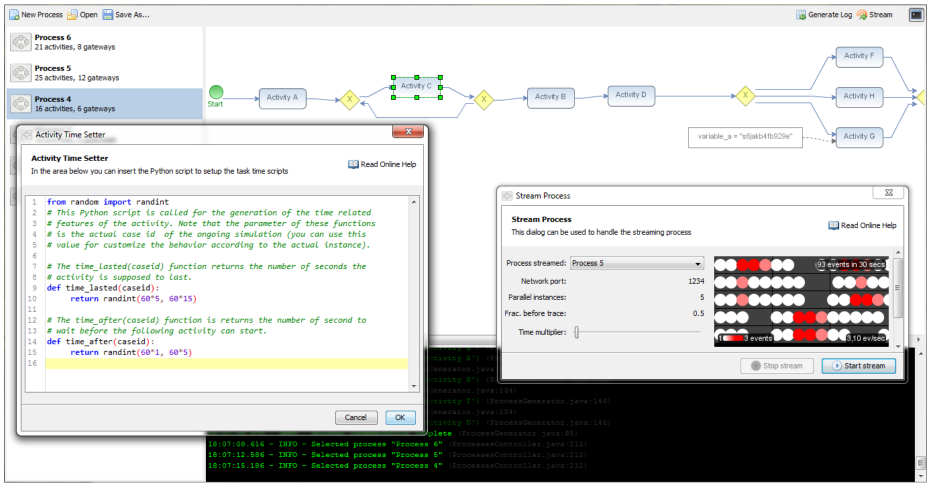
Fig. 2: A process model randomly generated with two sequential evolutions.

used to populate the data structures required for the generation of the stream. Then, since the event emission is performed in "the actual time" (opposed to the "simulated time"), it might be necessary to scale the simulation time in order to have the desired events emission rate. To this end we need a time multiplier, which is expected to be defined in $(0, \infty]$. This time multiplier is used to transform the duration of a trace (and the time position of all the contained events), from the simulation time to the real time.

One typical aspect of online settings is the presence of concept drifts. The tool is able to dynamically switch the source generating the events but to change the stream source, a second model is required. To create another model, two options are available: one is to load or generate from scratch a model; the other is to "evolve" an existing one: this is an important feature of PLG2. To evolve an existing model, PLG2 replaces an activity with a subprocess generated using the random procedure already used for the process randomization. The new process could be very similar to the originating one or very different, and this basically depends on the probability configured by the user. For example, Fig. 2 reports two evolutions of the process model, which has been randomly generated. Please note that evolutions can involve the creation or the deletion of data objects as well.

Implementation The tool is implemented as a Java application. It is available as open source project and also binary files are provided.[2] The project APIs can also be easily used to randomly generate processes or logs. The current implementation is able to load BPMN files generated with Signavio or PLG2. Model can be exported as PNML or PLG2 file, or as graphical representation (as BPMN and Petri net) using the Graphviz file format. The simulation of log files generates a XES-compliant objects, which can be exported both as XES or MXML (for compatibility with ProM 5). From Fig. 3 it is possible to see the main structure of the GUI: there is a list of generated processes on the left. The selected process is shown on the main area. Right clicking on activities allows the user to set up activity-specific properties (such as times, or data objects). On the bottom part of the main application it is possible to see the PLG2 console. Here the application reports all log information, useful for debugging purposes. The application dialog in the foreground is used for the configuration of the Python script which will be used to determine the time properties. As shown, specific syntax highlighting and other typing hints (such as automatic

[2] See http://plg.processmining.it and https://github.com/delas/plg.



Activity Time Setter
In the area below you can insert the Python script to setup the task time scripts

```python
from random import randint
# This Python script is called for the generation of the time related
# features of the activity. Note that the parameter of these functions
# is the actual case id  of the ongoing simulation (you can use this
# value for customize the behavior according to the actual instance).

# The time_lasted(caseid) function returns the number of seconds the
# activity is supposed to last.
def time_lasted(caseid):
    return randint(60*5, 60*15)

# The time_after(caseid) function is returns the number of second to
# wait before the following activity can start.
def time_after(caseid):
    return randint(60*1, 60*5)

```

Stream Process
This dialog can be used to handle the streaming process

Process streamed: Process 5
Network port: 1234
Parallel instances: 5
Frac. before trace: 0.5
Time multiplier:

```
ctivity R') (P
Generator.java
ctivity S') (P
Generator.java:134)
ctivity T') (ProcessGenerator.java:146)
Generator.java:134)
ctivity U') (ProcessGenerator.java:146)
plete (ProcessGenerator.java:85)
18:07:08.616 - INFO - Selected process "Process 6" (ProcessesController.java:212)
18:07:12.586 - INFO - Selected process "Process 5" (ProcessesController.java:212)
18:07:15.186 - INFO - Selected process "Process 4" (ProcessesController.java:212)
```

Fig. 3: Screenshot of PLG2 with several models in the workspace; the time setting dialog for “Activity C”, and the log console. The stream dialog is also displayed.

indentation) helps the user in writing Python code. The stream dialog is also displayed in foreground: it is possible to dynamically change the streamed process (using the “Process streamed” combo box) and the time multiplier. The right hand side of such dialog (in the rectangle with black background) reports “a preview” of the stream: 30 seconds of the stream are reported and each filled circle represents, in this case, up to 3 events.

3 Conclusion

This paper describes PLG2, which is the evolution of an already available tool. While that tool was able to randomly generate process models and simulate them, PLG2 extends the support to multiperspective models (by adding detailed control of time perspective and introducing data objects) and has full support for the simulation of offline and online settings (generating drifting models and simulating event streams). A screencast showing PLG2 features is available at http://youtu.be/t-GMV4hU_vs.

References

1. van der Aalst, W.M.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer Berlin / Heidelberg (2011)
2. van der Aalst, W.M., ter Hofstede, A.H., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
3. Burattin, A.: PLG2: multiperspective processes randomization and simulation for online and offline settings. CoRR abs/1506.08415 (2015)
4. Burattin, A.: Process Mining Techniques in Business Environments. Springer (2015)

5. Burattin, A., Sperduti, A.: PLG: a Framework for the Generation of Business Process Models and their Execution Logs. In: Proceedings of BPI. Springer (2010)
6. Burattin, A., Sperduti, A., van der Aalst, W.M.: Control-flow Discovery from Event Streams. In: Proceedings of the IEEE CEC (2014)
7. Günther, C.W.: Process mining in Flexible Environments. TU Eindhoven (2009)