PTandLogGenerator: a Generator for Artificial Event Data

Toon Jouck and Benoît Depaire

Hasselt University, Agoralaan Bldg D, 3590 Diepenbeek, Belgium
toon.jouck@uhasselt.be; benoit.depaire@uhasselt.be

Abstract. The empirical analysis of process discovery algorithms has recently gained more attention. An important step within such an analysis is the acquisition of the appropriate test event data, i.e. event logs and reference models. This requires an implemented framework that supports the random and automated generation of event data based on user specifications. This paper presents a tool for generating artificial process trees and event logs that can be used to study and compare the empirical workings of process discovery algorithms. It extends current tools by giving users full control over an extensive set of process control-flow constructs included in the final models and event logs. Additionally, it is integrated within the ProM framework that offers a plethora of process discovery algorithms and evaluation metrics which are required during empirical analysis.

Keywords: artificial event logs; process simulation; process discovery

1 Generating Event Data for Algorithm Evaluation

Process discovery techniques are concerned with discovering the control-flow of a process directly from an event log. During the last decade many process discovery techniques have been developed (see [2] for an overview). Currently new techniques are developed to outperform others in term of model quality measures. This has led to an increasing importance of evaluating and comparing existing algorithms empirically [2]. In order to perform such an evaluation, a set of appropriate test event data, i.e. event logs and reference models, is required.

Generally, three requirements with regard to test data must hold while doing empirical analysis of process discovery algorithms. Firstly, a researcher should have full control over the control-flow characteristics of the event data generated. A second requirement is randomness to prevent wrong generalizations based on non-random event data. Finally, the final event logs and reference models should be in the standard format[1] to ensure their compatibility with tools that implement process discovery algorithms and evaluation metrics.

[1] Event logs should conform to the XES standard: http://www.xes-standard.org

The presented tool *PTandLogGenerator* fulfills all the requirements stated above as it enables the random and automated generation of process trees and event logs based on user-defined control-flow specifications. It applies a generic two-step approach: generate a process tree, then simulate this tree into an event log described in [4]. Firstly, the user specifies the control-flow constructs, defined as meaningful process blocks that will be included in the generated process trees. In the second step, the trees are simulated into event logs.

The idea of implementing an artificial data generator is not new. However, the existing tools still have some limitations with regard to the requirements stated above. The most advanced tool, PLG2 [1], allows for control of the basic workflow patterns, but does not allow for more complex constructs such as duplicate activity labels or long-term dependencies. Moreover, PLG2 is not directly integrated within the ProM framework, which is a disadvantage when doing empirical process discovery evaluation.

The remainder of this paper describes how to use the *PTandLogGenerator* tool in the case of comparing two process discovery algorithms.

2 Walkthrough of the Process Tree and Log Generator

The *PTandLogGenerator* tool is available as a package in the open-source framework ProM[2]. This section will describe the different steps of generating a sample of event logs needed to evaluate two process discovery algorithms. Consider the comparison of the α++ miner [6] and the Inductive Miner [5] on logs including long-term dependencies, i.e. causal dependencies between tasks in different exclusive choice constructs. Consequently, one needs a set of event logs containing such long-term dependencies, while controlling for other control-flow constructs. The following paragraphs show how these can be created using the *PTandLogGenerator*, see `https://drive.google.com/file/d/0B9nT40tWjscV0V94VDEwb3I4V2s/view?usp=sharing` for a screencast.

A Population of Process Trees The starting point is the definition of a process tree population. These process trees contain a combination of control-flow constructs (CFC), i.e. process tree building blocks, that are used as population parameters to describe the population. The user can assign probabilities to each of the CFC to express the probability that these constructs are added to a tree within the population. We distinguish between three types of constructs: activities, workflow patterns and complex constructs:

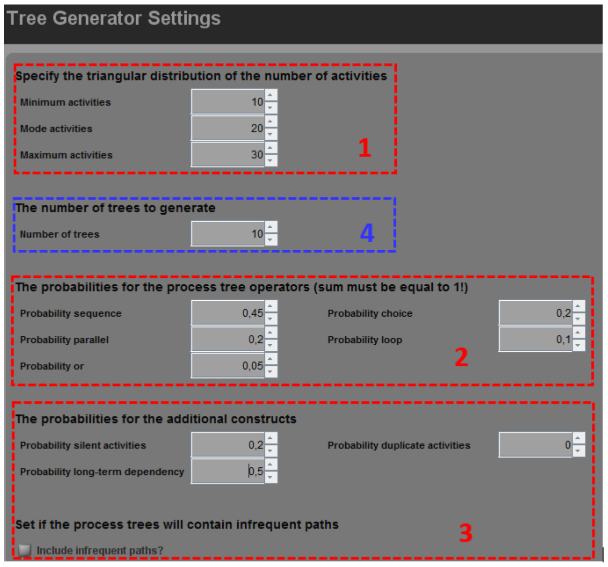
– Activities (see area 1 in Fig. 1): users can influence the size of the trees included in the population by specifying the triangular distribution of the number of activities by assigning a minimum, a mode and a maximum. Each time a process tree is generated, i.e. drawn from the population, a random number for the number of activities is taken from that triangular distribution.

[2] Available in the ProM nightly builds at `http://www.promtools.org/`

- Workflow control-flow patterns (see area 2 in Fig. 1): basic fundamental patterns common to all business processes. These patterns include sequence, exclusive choice, multi-choice, concurrent behavior and loops, represented by the following operator nodes in Process trees: $\rightarrow, \times, \vee, \wedge$ and $\circlearrowleft$.
- Complex constructs (see area 3 in Fig. 1): more complex control-flow constructs include silent activities, reoccurring activities (i.e. duplicate labels), long-term dependencies and infrequent paths. The last construct assigns unequal branch probabilities to each of the outgoing branches of an exclusive choice in order to make some paths less frequent in the process.

In our example use case, we want to evaluate two algorithms on long-term dependencies. Therefore we define a population of process trees with 50% probability of inserting long-term probabilities. The definition of this population can be configured using the settings wizard shown in Fig. 1.

Generating a Random Sample of Trees The next step involves drawing a random sample, i.e. generating random process trees, from the previously specified population. The size of this sample can be specified in the tree generator settings as can be seen in area 4 of Fig. 1. For the example use case we generate a random sample of size 10. The algorithm described in [4] is implemented to build each process tree in a stepwise manner. It uses the probabilities specified in the population as input parameters to randomly add nodes to the tree.


Tree Generator Settings

Specify the triangular distribution of the number of activities

Minimum activities	10
Mode activities	20
Maximum activities	30

The number of trees to generate

Number of trees	10

The probabilities for the process tree operators (sum must be equal to 1!)

Probability sequence	0,45	Probability choice	0,2
Probability parallel	0,2	Probability loop	0,1
Probability or	0,05		

The probabilities for the additional constructs

Probability silent activities	0,2	Probability duplicate activities	0
Probability long-term dependency	0,5		

Set if the process trees will contain infrequent paths

☐ Include infrequent paths?

Fig. 1: The Population of Process Trees in the Example

The output of this step is a set of process trees in the standard PTML-format. In this way the complete toolbox for import/export, analysis and visualization of process trees integrated into ProM is available to the user. The screenshot in Fig. 2 shows the visualization pane for tree number 3 in the sample. The annotations on the branches represent their execution probabilities. The tree shown in Fig. 2 contains a long-term dependency between the activity h and c expressing the causal relationship that if h is executed, c can never follow later on. For further information, the reader is referred to the tree generating algorithm described in [4].

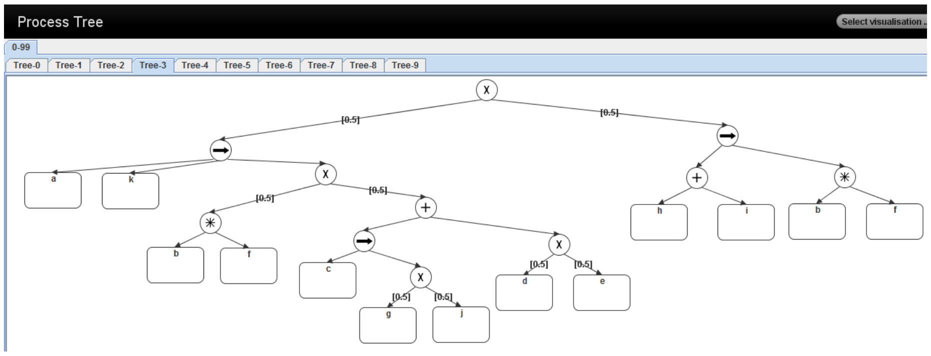
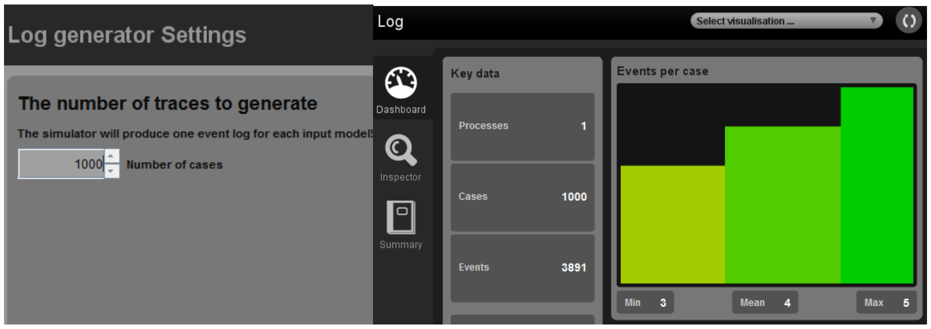


Fig. 2: A process tree from the random sample visualized

Simulating Trees Into Event Logs Then in the third step the tool enables users to generate (an) event log(s) for each process tree in the sample. Each process tree can be seen as a population of event logs: an event log is a multiset of traces simulated from that process tree. The tool allows users to specify the number of traces that the final event log(s) will contain as shown in Fig. 3a. In the example case we generate one event log with 1000 traces for each tree. The resulting event logs are in the standard XES-format providing all the log functionalities provided in ProM. Fig. 3b shows the log view for the event log generated from the tree in Fig. 2.


Log generator Settings

The number of traces to generate

The simulator will produce one event log for each input model!

1000 Number of cases

Log

Key data

Processes 1

Cases 1000

Events 3891

Events per case

Min 3 Mean 4 Max 5

(a) The Number of Traces in the Generated Event Logs (b) Log View of the Generated Event Log

Fig. 3: Log Generator

Evaluating Process Discovery Techniques Once the event logs are generated, the empirical analysis of the evaluation of process discovery technique is enabled. In the running case we apply the α++ miner [6] and the Inductive Miner [5] in ProM and calculate quality metrics for each discovered model. In the running case the discovered models of the inductive miner have an average fitness value of 100% and a precision value of 58.5%, whereas the models discovered by α++ miner have a lower average fitness value of 32.1% and a higher average precision value of 77.9%. These results are used to demonstrate the possible use cases of the tool. A more thorough empirical analysis would need more observations which is outside the scope of this paper.

3 Maturity and Use Cases

The tool has reached a high level of maturity which enables its use in large scientific experiments. This has been proven by the successful application of the tool in the large scale empirical assessments in [3]. Furthermore, the organizers of the first process discovery contest[3] have chosen this tool to create the benchmark event logs as it allows users the full control over an extensive range of control-flow constructs.

Acknowledgements

Special thanks to Alfredo Bolt and dr. Massimiliano de Leoni for their help on implementing and improving this tool.

References

1. Burattin, A.: PLG2: Multiperspective Processes Randomization and Simulation for Online and Offline Settings. ArXiv e-prints (1506.08415) (Jun 2015)
2. De Weerdt, J., De Backer, M., Vanthienen, J., Baesens, B.: A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. Information Systems 37(7), 654–676 (Nov 2012), `http://www.sciencedirect.com/science/article/pii/S0306437912000464`
3. Janssenswillen, G., Jouck, T., Creemers, M., Depaire, B.: Measuring the quality of models with respect to the underlying system: An empirical study. In: Business Process Management 2016. Springer (accepted)
4. Jouck, T., Depaire, B.: Generating Artificial Data for Empirical Analysis of Process Discovery Algorithms: a Process Tree and Log Generator. Technical Report, Universiteit Hasselt, Universiteit Hasselt (Mar 2016), `http://hdl.handle.net/1942/20818`
5. Leemans, S.J., Fahland, D., van der Aalst, W.M.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Business Process Management Workshops. pp. 66–78. Springer (2014), `http://link.springer.com/chapter/10.1007/978-3-319-06257-0_6`
6. Wen, L., van der Aalst, W.M., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Mining and Knowledge Discovery 15(2), 145–180 (2007), `http://link.springer.com/article/10.1007/s10618-007-0065-y`

[3] `http://www.win.tue.nl/ieeetfpm/doku.php?id=shared:process_discovery_contest`