### Concept Drift Log Generator Tool (CDLG)

A Tool for the Generation of Event Logs with Concept Drifts

### TUTORIAL

This project comes as an extension of the initial project CDLG tool that is described in the paper:\_CDLG: A tool for the Generation of Event Logs with concept Drifts\_by Justus Grimm, Alexander Kraus, and Han van der Aa. This tutorial provides a step-by-step demonstration of how to use the CDLG tool. This tutorial provides a step-by-step demonstration of how to use the CDLG tool. The tool itself can be accessed through our repository: [https://github.com/akprojectshub/cdlg\_tool\_dev.](https://gitlab.uni-mannheim.de/processanalytics/cdlg_tool) Before using the tool, clone the project and follow the provided installation instructions.

Reference: “*CDLG: A Tool for the Generation of Event Logs with Concept Drifts”* by Justus Grimm, Alexander Kraus, and Han van der Aa, submitted to the demo track of BPM 2022 and “An Experimental Evaluation of Process Concept Drift Detection” by Jan Niklas Adams, Cameron Pitsch, Tobias Brockhoff, Will M.P. van der Aalst

# The Tool’s Purpose

Overall the tool includes two fundamental features:

1. Generation of a collection of logs with noise and multiple drifts
2. The evaluation of drifts in actual logs by comparing it to tool generated drifts

# Generation of a Collection of Logs

Unlike the previous version, here we opted for a parameter-file mode to generate a collection of logs or single event logs. This method allows to generate a collection of logs with each containing a single or multiple drifts. A predefined file has already been specified to generate collections with different drift scenarios. The parameter files can be found in “***src\input\_parameters”.***

A screenshot of a computer code

Description automatically generated

Figure 1 Paramter file

# As we can see in this sample parameter file (Figure 1), a new important feature compared to the previous version is the inclusion of multiple drifts per log as depicted by the *Drift\_types* parameter.

# To execute the parameter-file mode user should:

# Specify the parameters in the corresponding text files placed in “Data/parameters”, if needed.

# Execute the file **“generate\_collection\_of\_logs.py”**.

# The collection of logs generated can be found in **“output”** folder. The folder contains a CSV file named **“drift\_info.CSV”** which is a data frame that summarizes all the logs and drifts generated.

# A screenshot of a computer Description automatically generated

Figure 3 Generated logs

# The folder also contains multiple XES files each one of them stores the information (drift, attributes, traces…) of a single log generated by the tool as illustrated by Figure 3.

# Evaluation of actual logs

# The evaluation of actual logs refers to the process through which the user compares a collection of logs that he owns with a tool generated one. Overall, the evaluation process can be divided into three steps. The first step consists of generating a collection of logs using the tool (this step can be executed by referring to section 1). Second step consists of “reformatting” the collection of logs owned by the user to a structure that is recognized by the tool. The last step consists of running the evaluation which will compare the two collection of logs and return evaluation metrics. It is important to note that the tool can only evaluate two collection logs if they are in the format of a class **“Collection”** that is generated by the tool. When loading logs used for the automated evaluation a step that converts the data uploaded by the user to a class **“Collection”** is needed as we will see in the more detailed explanation below.

# Reformatting the actual collection of logs

# The user can proceed in two ways to convert his actual collection of logs to a structure that is recognized by the tool.

# First the user can either provide a CSV file with the same structure as the one generated by the tool. This file as illustrated in Figure 4.

# A screenshot of a computer Description automatically generated

Figure 4 Files storing collection of logs information

# In this case the user proceeds as described below and illustrated by Figure 5. First, generate two empty collections of logs. Use the first one to import the user collection from a csv file using the function **“import\_drift\_and\_noise\_info\_from\_flat\_file\_csv(path\_of\_the\_file)”**. This function is a method of the collection class which can be found in **“src/data\_classes/class\_collection”.**

# A screen shot of a computer program Description automatically generated

Figure 5 Compare two collection of logs using CSV files

# Next the user should provide a second collection of logs by again either providing a CSV file or a set of XES files, each containing log information.

# Second the user can provide a folder containing a set of XES files. In this case to run the automated evaluation the user proceeds as described in Figure 6.

# A screenshot of a computer program Description automatically generated

Figure 6 Compare two collection of logs using XES files

# The user uses the **Collection** class method **Extract\_collection\_of\_drifts** which extracts log data from a set of XES files to a Collection class.

# Evaluation

# To run the evaluation the user can use the function **Automated\_evaluation**, which needs an actual collection class and a detected collection class, as well as a lag parameter. As a result of this function, an evaluation report will be generated which after a matching of the two collections will generate the level of accuracy, recall, precision and F1 score in a XLS file.

# 

Figure 7 Evaluation Report

Figure 7 illustrates an example of an evaluation report after running the Automated\_evaluation function.

# Real Example

# So first let’s generate a collection of logs.

# For that we should first specify the parameters to be used for the generation process in the file “**\src\input\_parameters\default.txt”**. Then we should execute the “**generate\_collection\_of\_logs.py”**. Upon that a new folder, located in **“\output”** will be created. The folder will contain different XES files each one representing a generated log as well as a CSV file that aggregates all the information related to all the logs that have been generated.

# After that let’s assume, the user wants to compare the tool generated collection of logs with an actual collection of logs at hand.

# In that case a preprocessing step is required. In fact, the user has to convert both collections of logs into a class collection that the tool can recognize. Thus the user will first, instantiate two empty collection classes by running the lines of code:

**Col\_act = Collection()**

# **Col\_det = Collection()**

# Next the user will load the collection of logs into the empty class instances by running the methods. For each method the user has to specify the path leading to the csv files and XES files storing the collection of logs information.

**Col\_act.import\_drift\_and\_noise\_info\_from\_flat\_file\_csv(path\_actual\_csv\_fil)**

**Col\_act.Extract\_collection\_of\_drifts(path\_actual\_collection)**

# In this case we assumed that the collection of logs the user has at hand is stored in csv files.

# Finally, both collection classes can be compared using the function:

**Automated\_evaluation(Col\_act, Col\_det,lag= 100)**

# The automated evaluation generates a report which stores the comparison results in the folded **“\output\Evaluation reports”**