# AIR-BAGEL: An Interactive Root cause-Based Anomaly Generator for Event Logs

Jonghyeon Ko, Jongyup Lee, Marco Comuzzi
*Department of Industrial Engineering*
*Ulsan National Institute of Science and Technology (UNIST)*
Ulsan, Republic of Korea
{whd1gus2, belllight, mcomuzzi}@unist.ac.kr

***Abstract*—We describe AIR-BAGEL, a tool[1][2] to generate pseudo-real trace-level anomalies in event logs. Anomalies to be injected are defined by their root cause, i.e., resource behaviour or system malfunctioning. For each root cause, several anomaly types can be specified, e.g., deleting, replacing or moving events in a trace. Root causes and anomalies have been modelled based on existing literature on event log cleaning and data quality analysis. AIR-BAGEL addresses the issue of unavailability of labelled real world event logs for developing and evaluating event log cleaning and reconstruction techniques and it represents a step forward compared to current approaches in the literature that simply inject different types of anomalies randomly in event logs.**

***Index Terms*—event log, data quality, anomaly, process mining, event log cleaning.**

## I. Introduction

Event logs captured in the real world are prone to errors [1]. These can stem from a variety of root causes, such as system malfunctioning or human mistakes, resulting in different types of errors, such as abnormal activity labels, or missing, duplicated, and swapped events [1]–[8]. Errors in event logs hamper the possibility of extracting useful insights from event log analysis. For instance, they clearly influence the models obtained through process discovery and the fitness computation in conformance checking. Therefore, one should aim at removing these errors before running event log analytics.

If a process model is available or can be discovered from clean traces, then errors can be detected using traditional conformance checking techniques. However, in many cases a process model is not available. Event log anomaly detection or *cleaning* has emerged recently as a new field in process mining developing techniques to identify anomalies in event logs without assuming the existence of a process model [1].

To be evaluated, event log cleaning techniques require event logs in which traces and events are labelled, i.e., whether normal or anomalous. When such techniques exploit machine learning algorithms, labelled logs are also necessary for training/testing during model development. Unfortunately, labelled real world event logs are not available among the ones[3] usually considered within the process mining community. Therefore, researchers in this field have relied on artificially injecting anomalies into real or simulated event logs.

[1]Github (including tutorial): https://github.com/jonghyeonk/AIR-BAGEL
[2]Screencast video: https://shorturl.at/wRY04
[3]e.g., at https://data.4tu.nl/repository/collection:event_logs_real

Anomalies of different types are normally injected randomly into event logs at different ratios, i.e., until a target ratio of existing traces and/or events have been modified to become anomalous [1]. As far as existing tools are concerned, event log generation tools, e.g., PLG2 [9] and PTandLogGenerator [10] allow to perturb an event log obtained through simulation of a process model by randomly changing the order of events in a trace or randomly deleting/adding events in a trace, but they do not allow injecting anomalies generated by more complex patterns or into already existing event logs.

In the real world, anomalies are generated by complex organisational situations. Anomalies in an event log, in fact, normally are linked to specific root causes [2], [3]. These can be classified, at least at a first high level of analysis, into two categories: *resource* and *system*. The former refers to human resources involved in a process being the source of anomalies. Resource A, for instance, may be sloppier at their job than resource B, and therefore cases in which A is involved may have more events skipped or wrongly recorded than the ones involving B. The latter refers to anomalies associated with malfunctioning of systems supporting the execution of processes. System A, for instance, may have malfunctioned for a specific amount of time on a specific day; as a consequence, depending on the type of malfunctioning, the events happening in that specific time window may be missing from the log or have been erroneously recorded, e.g., moved to a different case.

This paper describes AIR-BAGEL, an interactive tool for anomaly injection in event logs that aims at simulating pseudo-real anomalies. The idea behind AIR-BAGEL is to associate the injected anomalies with specific root causes, i.e., the behaviour of resources or system malfunctioning. AIR-BAGEL injects anomalies at the level of order and occurrence of activities in a case, e.g., skipping or replacing events in a trace in an event log. The tool outputs event logs with injected anomalies augmented with case-level anomaly binary labels and other attributes required to reconstruct the type of anomaly that was injected. The main objective of AIR-BAGEL is to provide the process mining research community with a simple tool to generate pseudo-real anomalies in existing event logs, so as to enable the development and evaluation of event log cleaning approaches using event log anomalies that better resemble the ones that could be encountered in the real world.

The next section gives an overview of the tool, while Section III and IV describe more in detail the features and the maturity of AIR-BAGEL, respectively. How to access and install the tool is described in Section V, while conclusions are drawn in Section VI.

## II. Overview

AIR-BAGEL generates pseudo-real anomalies applying two types of root causes, i.e., *resource* and *system*. It is assumed that each event in a log has two attributes, one identifying the (human) resource in charge of executing the task to which an event refers and one identifying the system used for recording the task. Since this information, particularly the *system* attribute, is often not available in an event log, AIR-BAGEL also allows to generate these two attributes. The user can specify the existence of a given number of resources/systems and associate them with specific classes of events in the process.

As far as generating anomalies is concerned, when *resource* is the root, since each resource may have different job competency, experience, and behaviour, e.g., new employees are more prone to do mistakes than longer tenured ones, AIR-BAGEL firstly creates a distribution of probabilities for each resource in the log to commit an error while logging an event. This distribution may also be set manually if needed. Then, it allows to specify what type(s) of error the resources are likely to commit, e.g., forgetting to record an event or recording an event as a different one. When *system* is the root, the time and occurrence of one or more malfunctioning of each system may be specified by a Poisson random process or manually-specified. Then, the type of malfunctioning must be specified, e.g., whether it concerned skipping the registration of events, i.e., similarly to a system down, or recording a wrong label for events. Specifically, AIR-BAGEL considers 8 pseudo-real patterns of anomaly types caused by the *resource* root and 3 caused by the *system* root. These are discussed in the detail in the next section.

The output of AIR-BAGEL is an event log having the same attributes of the input event log in which (i) additional resource and/or system columns are created if required, (ii) anomalies have been injected according to specific patterns and (iii) events and cases are augmented with additional labels to support evaluation of event log cleaning and reconstruction techniques. In particular, for each event, one label recording whether the event belongs to a case that has become anomalous is created. This label supports the evaluation of event log cleaning techniques. Other attributes are created to record the detail of the type of anomaly, such as the position at which an event was skipped in a case. This type of information supports the evaluation of event log reconstruction techniques aiming at explaining the type and cause of trace anomalies.

## III. Architecture and Features

The usage workflow of AIR-BAGEL includes 3 steps (see Figure 1): (i) data import and preprocessing, (ii) root parameter setting and anomaly injection, and (iii) output evaluation and data export.

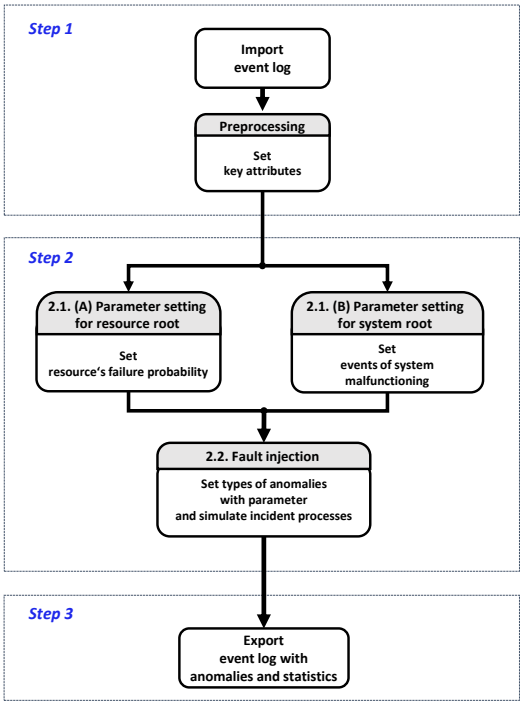


Fig. 1. AIR-BAGEL usage workflow

***Data import and preprocessing.*** The tool provides basic import functionality.[4] In particular, the tool tries to guess the meaning of columns and the format of timestamps in an event log using basic pattern matching. The user, however, can override the matching if incorrect. After an event log has been imported, if necessary AIR-BAGEL supports a user to configure and generate artificial attribute values for *resource* and/or *system*.

***Root parameter setting and anomaly injection.*** The next step for the user is to specify the root of anomaly injection (resource or system). It is also possible to inject anomalies considering both roots at the same time.

Then, for the *resource* root cause, the probability of a specific resource to inject an anomaly into an event has to be specified. This can be done by selecting and parameterizing one of three possible distributions: exponential(parameter $\lambda$), normal($\mu, \sigma$), and uniform($a, b$). When injecting anomalies, a value is drawn from the specified distribution for each value of the resource attribute in the log. Alternatively, the user can specify the probabilities for each resource manually. This value represents the probability that an event involving this resource will be affected by an anomaly.

For the *system* root, the user must specify the time of occurrence and duration of system malfunctioning occurrences. This can be done manually, or by specifying the value of the parameter $\lambda$ of a Poisson process that models the distribution of inter-arrival times of system malfunctioning (which sets the

[4]Only import/export of event logs in csv format is supported at the moment.

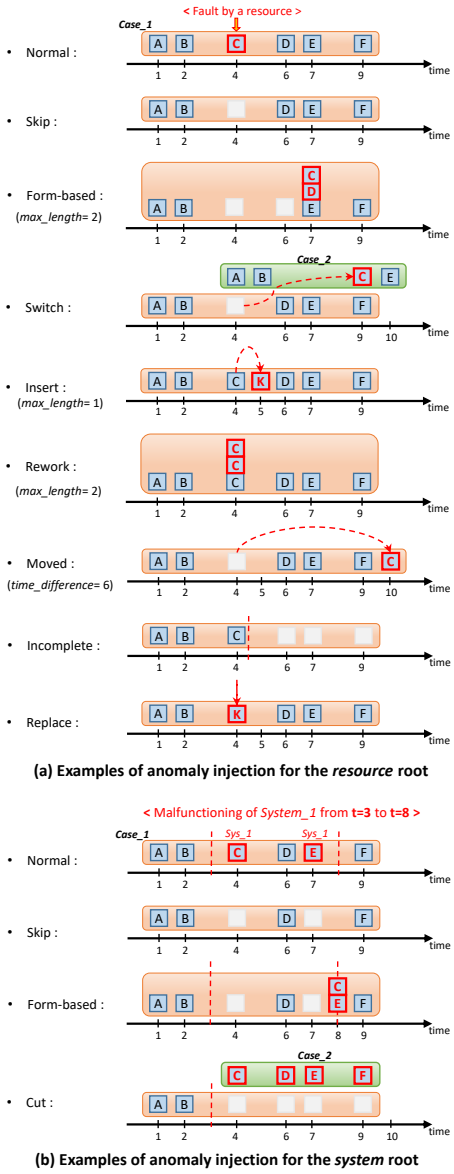


Fig. 2. Examples of anomaly injection by root causes

start timestamp of malfunctioning occurrences) and the values of the parameters $a$ and $b$ of a uniform distribution from which the duration of the malfunctioning will be sampled (which sets the end timestamp of malfunctioning occurrences).

Next, the user decides which type of anomalies will be applied for each root cause to events that have been selected as being anomalous, because the corresponding resource behaviour fault probability or because they fall within the occurrence of a system malfunctioning. Multiple types of anomalies may be defined. If multiple types are defined for a given root, then a *strength* parameter determines how likely is each type to be chosen in respect of others when injecting an anomaly for a given event.

Specifically, anomalies are injected to events ordered in ascending timestamp order. In particular, once a case becomes anomalous, i.e., because one of its events has been modified according to an anomaly type, no other anomalies of the same root will be applied to events of the same case. Multiple anomalies implied by different root cases can be applied to the same trace. Therefore, an individual trace will be injected with at most two types of anomalies, one for the resource root and one for the system root cause.

As stated earlier, we consider a total of 11 anomaly types, 8 for the *resource* root and 3 for the *system* root. Anomaly types have been modelled based on previous research on event log cleaning and event log data quality analysis [1]–[8]. They are exemplified in Fig. 2 and described in detail next:

### A. Resource root anomalies

These anomalies are defined by replacing the target event that has been designated as anomalous (event C in Fig. 2) with other events, skipping it or moving it to a different case.

- $Skip$: The target event is deleted from the log [3]–[6]. This may occur when a resource forgets to record a task that they executed;
- $Form-based(max\_length)$: A group of $max\_length$ consequent events including the target event is assigned the timestamp of the latest event in the group. This typically occurs when information about these events is collected in a single form [7]. Note that, when events have the same timestamp, traces are likely to be defined by the lexicographic order of event activity labels, which may introduce anomalies;
- $Switch$: The target event is moved to a different case chosen randomly;
- $Insert(max\_length)$: $max\_length$ event(s) are randomly generated by controlling their timestamp so that they are inserted right after the target event. For instance, in a hospital, a resource may guide a patient to visit a wrong department to receive treatment. Events such as registration at this wrong department may be logged before the patient is redirected to the correct place to receive treatment [4], [5];
- $Rework(max\_length)$: The target event is recorded multiple times. For instance, a resource might click the 'enter' button twice by mistake, recording 2 events of the same type [3], [4], [6];
- $Moved(time\_difference)$: The timestamp of the target event is modified of a quantity $time\_difference$. For instance, a resource might record a wrong time for a task in the future or the past by mistake [3]–[5], [7];
- $Incomplete$: All the events after the target one are deleted from the case. For instance, a resource may have an emergency and leave their workplace, therefore failing to record the last events of a case [6], [8];
- $Replace$: The activity of the target event is replaced by a different one chosen randomly. For instance, a resource may record by mistake the name of the customer or another activity as the activity label in one or more events [1].

### B. System root anomalies

The target event(s) in this case are the ones executed by a specific system during its malfunctioning.

- $Skip$: The target events are deleted. This is the typical example of system down malfunctioning [3]–[6];
- $Form-based$: Similarly to the same anomaly type for the resource root, the target events are assigned the timestamp corresponding to the end of the malfunctioning. This may occur because of system lagging: the recording of one or more events is delayed until a system that has become unavailable goes back online [7];
- $Cut$: All the events after the start of the malfunctioning are cut to form a new case. For instance, after a system reboot, the case ID of the active cases may be reset, leading to the creation of a new case ID [7].

***Output evaluation and data export.*** After having configured the root causes and anomaly types, AIR-BAGEL now can start to generate anomalies in the event log. When the process is completed, the tool provides a statistical summary of the anomalies injected. The tool also allows to show or download the process models discovered using the inductive miner [11] with standard parameter settings from the original log and the log with injected anomalies. These process models give a first visual cue on the impact of injecting anomalies into an event log. Finally, the tool allows to export the event log injected with anomalies and augmented with additional anomaly label. Details about the attributes added by the tool are described in detail in the companion tutorial document.

## IV. Maturity

The tool is at an early development stage and far from being mature, particularly from a user experience standpoint. Yet, it provides a complete implementation of several pseudo-real anomaly injection patterns. The user experience can be improved by providing a better look-and-feel, a more streamlined workflow, possibly including feedback from extensive testing with students, and migrating to a Web-based interface.

As far as the functional requirements are concerned, we are currently working on the following extensions: (i) supporting the XES event standard format for import/export of event logs, (ii) implementing a finer-grained control of anomaly types, supporting users in specifying anomaly types for individual or groups of resource/system roots, and (iii) improving the summary information shown in the last step. We are also developing pre-defined anomaly injection *scenarios* to support non-expert users. Finally, by design the tool does not allow to control the overall injected case anomaly ratio, i.e., the ratio of cases that become anomalous as a result of anomaly injection. We are extending the tool to give an early estimate while setting the anomaly injection parameters of the number of cases that will be affected. This feature is particularly important when evaluating new techniques for event log cleaning and reconstruction in a systematic way.

## V. Access and Installation

The tool is a stand alone Python application. The source code is available at https://github.com/jonghyeonk/AIR-BAGEL. A tutorial document explaining the installation procedure and discussing more details about the parameters required to configure the anomaly injection process is available at the same location. AIR-BAGEL has been tested for Python 3.6. It requires the installation of `tk/tkinter`, the Python Imaging Library, and `PM4Py` [12]. For convenience, a pre-configured Ubuntu 20.04-based virtual machine image that could be imported by popular hosted hypervisors has also been made available and can be downloaded at: shorturl.at/kmHN2.

## VI. Conclusions

We have presented AIR-BAGEL, a tool to inject pseudo-real trace-level anomalies in event logs. The tool contributes to the process mining community by providing a means to generate event log anomalies modelled around the concepts of root cause and pseudo-real anomaly types, rather than simply injected randomly. It can be used to generate logs for evaluating event log cleaning and reconstruction approaches and to support the training/testing of those approaches that use machine learning algorithms.

## References


[1] H. T. C. Nguyen, S. Lee, J. Kim, J. Ko, and M. Comuzzi, "Autoencoders for improving quality of process event logs," *Expert Systems with Applications*, vol. 131, pp. 132–147, 2019.

[2] F. Bezerra and J. Wainer, "Algorithms for anomaly detection of traces in logs of process aware information systems," *Information Systems*, vol. 38, no. 1, pp. 33–44, 2013.

[3] K. Böhmer and S. Rinderle-Ma, "Multi-perspective anomaly detection in business process execution events," in *OTM Confederated International Conferences*. Springer, 2016, pp. 80–98.

[4] T. Nolle, S. Luettgen, A. Seeliger, and M. Mühlhäuser, "Binet: Multi-perspective business process anomaly classification," *Information Systems*, p. 101458, 2019.

[5] S. Budalakoti, A. N. Srivastava, and M. E. Otey, "Anomaly detection and diagnosis algorithms for discrete symbol sequences with applications to airline safety," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 39, no. 1, pp. 101–113, 2008.

[6] R. P. J. C. Bose, R. S. Mans, and W. M. P. van der Aalst, "Wanna improve process mining results?" in *2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, 2013, pp. 127–134.

[7] S. Suriadi, R. Andrews, A. H. ter Hofstede, and M. T. Wynn, "Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs," *Information Systems*, vol. 64, pp. 132–150, 2017.

[8] K. Diba, S. Remy, and L. Pufahl, "Compliance and performance analysis of procurement processes using process mining," in *International Conference on Process Mining*, 2019.

[9] A. Burattin, "Plg2: Multiperspective process randomization with online and offline simulations." in *BPM (Demos)*, 2016, pp. 1–6.

[10] T. Jouck and B. Depaire, "Ptandloggenerator: A generator for artificial event data." *BPM (Demos)*, vol. 1789, pp. 23–27, 2016.

[11] S. J. Leemans, D. Fahland, and W. M. van der Aalst, "Scalable process discovery with guarantees," in *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2015, pp. 85–101.

[12] A. Berti, S. J. van Zelst, and W. van der Aalst, "Process mining for python (PM4PY): bridging the gap between process-and data science," *arXiv preprint arXiv:1905.06169*, 2019.