

BIG GUI: a tool for building and analysing instance graphs

Claudia Diamantini¹, Laura Genga^{2,*}, Alex Mircoli¹ and Domenico Potena¹

¹ *Università Politecnica delle Marche, Ancona, Italy*

² *Eindhoven University of Technology, Eindhoven, The Netherlands*

Abstract

In this paper, we present BIG GUI, a novel tool supporting the generation and visualization of so-called *instance graphs* from an event log and a process model. Instance graphs are directed acyclic graphs representing both sequential and concurrent relations of process executions stored in the event log. The tool implements an IG generation algorithm robust to infrequent and non-compliant behaviours and provides a graphical interface for visualising the generated instance graph set.

Keywords

Process mining, Instance graphs, Process Discovery

1. Introduction

Modern organizations increasingly rely on information systems (e.g., SAP, or ERP) to carry out their business processes. As a result, *event logs* tracking business process executions have become readily available in several domains, boosting the development of *Process Mining* techniques, whose goal consists in extracting from such logs valuable knowledge regarding the corresponding processes [1]. Most existing PM techniques assume events generated during process executions to be stored in a *totally ordered* fashion, usually according to the timestamp of the logged events. However, this representation may lead to misleading insights in the presence of, e.g., uncertainty on the correctness of the logging entries. Furthermore, it hides potential concurrency between process activities. A recent survey from Leemans et al. [2] highlights an increasing interest towards techniques able to leverage *partial orders* to represent the recorded process behaviours to overcome these limitations and to support the development of process mining algorithms aware of concurrency and able to deal with logging uncertainty. Nevertheless, tool support for generating and analysing partially ordered traces is still limited.

In the present paper, we aim to fill this gap by introducing our *Building Instance Graphs GUI* (BIG GUI) tool. The BIG GUI tool supports the creation and exploration of so-called *instance graphs* (IGs). An IG is a directed acyclic graph where each node represents an activity instance. Two nodes may only be connected using an edge if there is a causal relationship between the

BPM: Demo & Resource Track


*Corresponding author.

†These authors contributed equally.

✉ c.diamantini@staff.univpm.it (C. Diamantini); l.genga@tue.nl (L. Genga); a.mircoli@staff.univpm.it (A. Mircoli); d.potena@staff.univpm.it (D. Potena)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

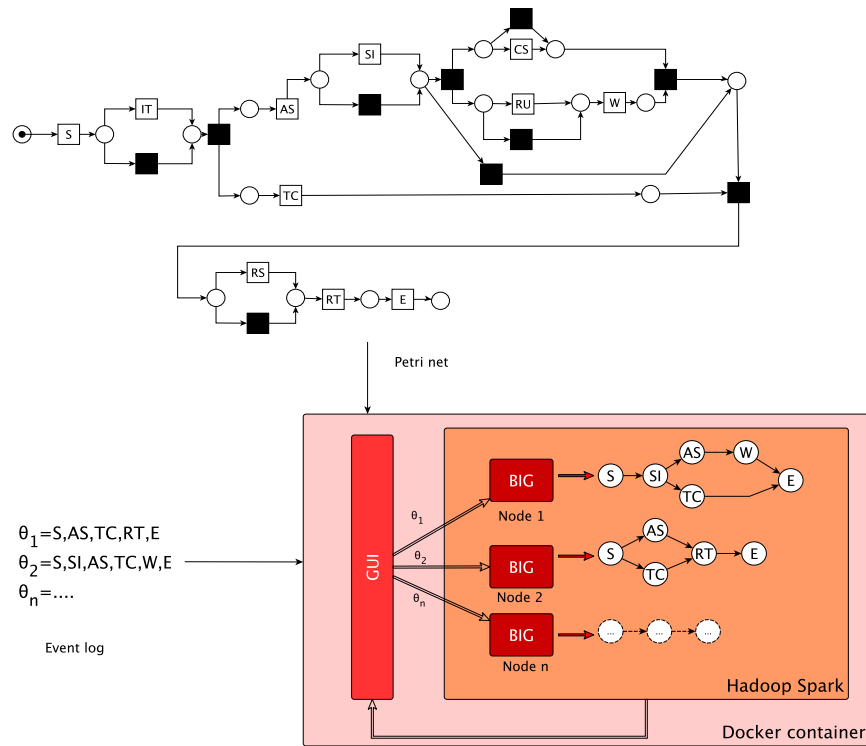


Figure 1: Architecture of the BIG GUI tool

source and target node, meaning that the activity corresponding to the target cannot be executed before the activity corresponding to the source is completed. Causal relations can be determined, for instance, by using prior domain knowledge or extracted from an event log recording process executions. Instance graphs resemble a transitive closure of a partial order relation among process activities and allow explicitly representing concurrent relations. The current version of our tool implements the algorithm introduced by Diamantini et al. [3], which is robust to infrequent and non-compliant behaviours. It implements a set of functionalities to visualize the generated IGs, which resemble what is commonly offered by PM tools for sequential traces (e.g., ProM, or Disco). Furthermore, it is integrated with the Spark architecture, thus enabling the use of cluster resources to parallelize and speed up the IG generation phase.

In the remainder of this manuscript, we introduce the main characteristics of the BIG GUI tool.

2. Innovation and characteristics

Figure 1 shows the architecture of the tool, which implements two main sets of functionalities. The first one is dedicated to the generation of a set of instance graphs, while the second one is dedicated to the visualization of the generated IGs, together with a set of simple insights and statistics to support the user in investigating the recorded behaviours. Note that the Hadoop

suite¹ is used to parallelize the generation of the IGs. In particular, each trace is converted independently and in parallel from the other on an independent Spark node. The Docker² suite is used to ensure interoperability and make the tool installation as transparent to the users as possible. The following subsections delve into the tool functionalities.

2.1. IG generation

The tool takes as input an event log and converts each sequential trace in the event log into an Instance Graph. The process model can be provided by a domain expert or extracted by a process discovery algorithm. The tool implements the Building Instance Graph (BIG) algorithm proposed in [3], which is able to handle traces that do not conform to the model. BIG is a two-step algorithm. First, an IG is built for each trace by extracting pair-wise ordering relations among process activities from the input Petri net.

However, in the presence of non-compliant events this procedure generates anomalous IGs, where connections among nodes do not reflect the temporal order of occurrence of the events and which over-generalize the process behaviour. An *IG repairing* procedure is applied to deal with these issues. First, anomalous traces (and, hence, IGs) are recognized in the event log using a conformance checking technique [4]. Then, tailored rules are applied for repairing IGs with deleted and inserted events. For deleted events, the repairing consists of identifying and connecting the nodes that should have been connected to the deleted activity. For the insertion repairing, we have to change the edges connecting the nodes corresponding to the event(s) before and the event(s) after the inserted event to connect such nodes with the node corresponding to the inserted event in the graph, taking into account the causal relations among its predecessors and successors in the trace.

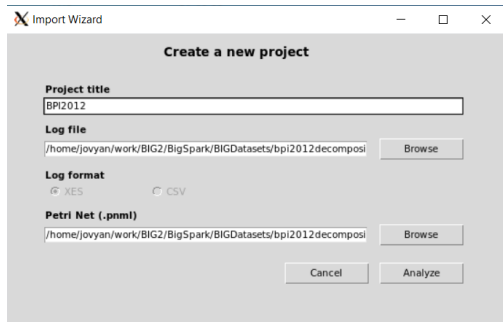
Figure 1 shows the IGs generated by the tool for the traces and the Petri net provided in input. While θ_1 is compliant with respect to the Petri net and, hence, no repairing is needed, θ_2 is not compliant since the activity *SI* is executed before *AS* and the activity *RT* is missing. In the corresponding IG, we can observe that the repairing of the deletion of *RT* has been realized by connecting its predecessors *W* and *TC* with its successor *E* while the inserted event *SI* has been connected to the events occurring before/after the anomalous event in the event log. Two main forces are driving the repairing procedures. On the one hand, we want to obtain a representation as precise as possible of the occurred anomaly. On the other, we want to preserve the concurrency relations described by the model. For this reason, the insertion of the event *SI* after *S* is repaired by connecting *SI* to both the causal successors of *S*³.

Figure 2.a shows the screenshots related to the *import* functionalities that allow the user to select the event log she wishes to convert and a process model in Petri net notation that she intends to use for deriving the causal relations. Every time the user imports an event log and a process model, a new *project* is created. The user can visualize previously created projects in her *home*, shown in Figure 2.b.

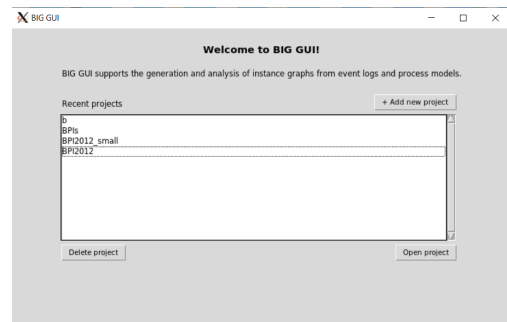
¹<https://spark.apache.org/>

²<https://www.docker.com/>

³Note that for deviations occurring within parallel constructs, other repair configurations are available, e.g., by adding parallel branch involving the inserted activities. Refer to [3] for additional details.



(a)



(b)

Figure 2: Import wizard (a) and Home window (b)

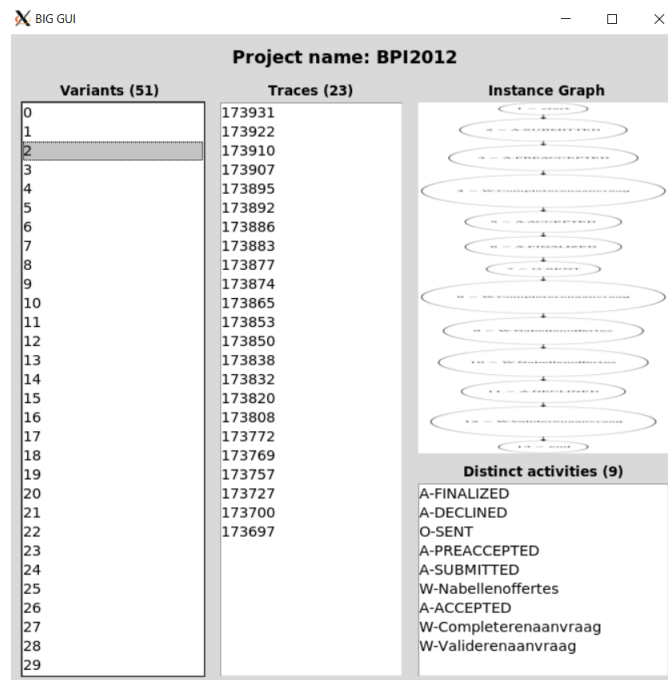


Figure 3: IG visualization

2.2. IG set exploration

BIG GUI supports both the visualization of single IGs and the exploration of the overall set of converted traces. Figure 3 shows the visualization canvas shown to the user after the event log has been imported and converted. On the left, the tool lists the different *variants* that have been detected in the log. Note that while a variant in a sequential event log corresponds to a unique sequence of events, a variant in the BIG GUI tool corresponds to a unique IG, which may correspond to several different sequential traces, depending on the degree of concurrency of the corresponding process execution. The panel in the centre lists all the case ids of the

traces corresponding to the variant. The user can explore a single variant by clicking on the corresponding case id. Both the graph and the original sequence of events are shown in the panel on the right to provide a complete visualization of the process execution.

The tool also implements *export* functionalities, allowing the user to export the set of instance graphs to be analyzed with other tools dedicated to, e.g., graph mining.

3. Tool maturity and evaluation

The stand-alone BIG GUI tool can be downloaded at <https://github.com/a-mircoli/big-gui>. At the same address can also be found a video showcasing the usage of the tool and a manual describing a use case, together with the corresponding event log and Petri net. The current version of the tool is still a prototype. Nevertheless, the BIG algorithm has been tested on synthetic and real-life event logs, as described in [3].

4. Conclusion

In this paper, we introduced the BIG GUI tool, which supports practitioners in converting sequential event logs into Instance Graphs, thus being able to take into account concurrent ordering relations among process activities in the analysis. The tools are available for download. In future work, we plan to further extend the tool by a) including additional strategies to convert sequential traces into partially ordered traces to support, e.g., techniques aimed to deal with uncertainty and b) extending the set of functionalities for graph-based analysis, e.g., by incorporating graph mining techniques.

References

- [1] W. Van Der Aalst, Process mining: data science in action, volume 2, Springer, 2016.
- [2] S. J. Leemans, S. J. van Zelst, X. Lu, Partial-order-based process mining: a survey and outlook, Knowledge and Information Systems 65 (2023) 1–29.
- [3] C. Diamantini, L. Genga, D. Potena, W. van der Aalst, Building instance graphs for highly variable processes, Expert Systems with Applications 59 (2016) 101–118.
- [4] W. Van der Aalst, A. Adriansyah, B. Van Dongen, Replaying history on process models for conformance checking and performance analysis, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2 (2012) 182–192.