Truncated Trace Classifier. Removal of Incomplete Traces from Event Logs

Gaël Bernard[1(✉)] and Periklis Andritsos[2]

[1] Faculty of Business and Economics (HEC), University of Lausanne, Lausanne, Switzerland
gael.bernard@unil.ch
[2] Faculty of Information, University of Toronto, Toronto, Canada
periklis.andritsos@utoronto.ca

Abstract. We consider truncated traces, which are incomplete sequences of events. This typically happens when dealing with streaming data or when the event log extraction process cuts the end of the trace. The existence of truncated traces in event logs and their negative impacts on process mining outcomes have been widely acknowledged in the literature. Still, there is a lack of research on algorithms to detect them. We propose the Truncated Trace Classifier (TTC), an algorithm that distinguishes truncated traces from the ones that are not truncated. We benchmark 5 TTC implementations that use either LSTM or XGBOOST on 13 real-life event logs. Accurate TTCs have great potential. In fact, filtering truncated traces before applying a process discovery algorithm greatly improves the precision of the discovered process models, by 9.1%. Moreover, we show that TTCs increase the accuracy of a next event prediction algorithm by up to 7.5%.

Keywords: Process mining · Predictive process monitoring · Predictive analytics · Truncated trace classifier

1 Introduction

The execution of a business process often leaves trails in information systems called event logs. Using these event logs, process mining techniques can extract data-driven insights about business processes. For example, it is possible to discover process models from event logs [1], to predict the next event [2,3], or to assess whether an ongoing process will fulfill a time constraint [4].

A truncated trace is a trace where the last events are missing. In fact, these events happened or will happen, but they are not available at the time of the analysis. The presence of truncated traces in event logs is acknowledged by researchers [5–8]. Interestingly, organizers of the latest edition of the Process Discovery Contest [9]–a contest where participants have to infer process models from event logs–have included truncated traces to make the synthetic event logs more realistic.

S. Nurcan et al. (Eds.): BPMDS 2020/EMMSAD 2020, LNBIP 387, pp. 150–165, 2020.
https://doi.org/10.1007/978-3-030-49418-6_10

We propose a Truncated Trace Classifier (TTC) that distinguishes truncated traces from the ones that are not truncated. We refer to the latter as the 'complete trace'. We foresee three benefits of using a TTC. First, a TTC can filter truncated traces. This is important because the success of process mining depends on the quality of the input event logs [10–13]. Second, a TTC helps to increase operational efficiency. For instance, a ticket in a call center might stay open for several days because an agent forgot to close it or because the customer did not follow up on a requested action. Using a TTC, we could avoid the manual task of closing them by doing it in an automated manner. Third, a TTC has potential that goes beyond filtering techniques. For instance, we show in this work that a TTC can improve the accuracy of predicting the next event.

It is not uncommon to read that truncated traces can be filtered out by looking at the very last event [5–8]. For example, a ticket is complete only when the activity 'closing the ticket' happens. However, such a closing event might not exist. Instead, a recurring one might occur [14]. For instance, the event 'delivering package' might be a good indicator to predict that an order is fulfilled, but it might reoccur if some items are being shipped separately. In such a case, relying on the very last event will result in a poorly performing TTC. This observation is in line with the conclusion drawn by Conforti et al. that existing techniques to filter traces are often simplistic [11].

To the best of our knowledge, we are the first work focusing on building an accurate TTC. Our contributions are the following: (1) We propose five machine learning-based TTC implementations. (2) We benchmark these five implementations and a baseline approach using 13 event logs. (3) We highlight the benefit of using a TTC for two process mining tasks, that of process discovery and next event prediction.

The rest of this paper is organized as follows. In Sect. 2, we provide an overview of process mining and define truncated traces. In Sect. 3, we propose several approaches to building a TTC, which we benchmark in Sect. 4. Sections 5 and 6 demonstrate the value of a TTC by showing how it can increase the process model precision and next event prediction, respectively. In Sect. 7, we discuss related work and we conclude the paper in Sect. 8.

2 Preliminaries

In this section, we briefly introduce the process mining discipline. Then, we define truncated traces.

2.1 Process Mining

Process mining brings data science and business process management closer together [6]. As stated in the process mining manifesto, the starting point of process mining is an event log [12]. An event log contains traces, which are sequences of events. Event logs often contain additional information such as a timestamp or the resource. We will use the simple event log definition introduced

in [6]. A simple trace σ is a sequence of events. A simple event log L is a multi-set of traces. For example, $L = [\langle abc \rangle^3, \langle ab \rangle^2]$ is an event log containing 5 traces and 13 events. Taking an event log as input, several process mining techniques are available. Typically, a process discovery algorithm such as the inductive miner, [15], can infer the most likely business process model behind an event log. To ensure that process mining works well, event logs need to be noise-free [10–12]. Truncation is one type of noise [16], which we introduce in the next section.

2.2 Truncated Traces

A truncated trace is an ongoing trace where the end of the process is missing. Truncated traces are sometimes referred to as 'incomplete cases' [7,8], 'incomplete traces' [5], or 'missing heads' [10]. We favor the term truncated over the term incomplete as the latter is often used for the concept of 'event log incompleteness', referring to the fact that an event log will most likely not contain all the combinations of behaviors that are possible because there are too many of them [12]. For instance, when there is a loop in the process model, the number of unique combinations is infinite. Event logs will most likely be incomplete while they may not contain truncated traces.

There are several reasons to explain the existence of incomplete traces. They might exist because of a flawed event log extraction process that cuts the traces at a fixed date, leaving the traces that finish after truncated. This issue–named 'the snapshots challenge'–has been identified by van der Aalst as one of the five challenges that occurs when extracting event logs [6, chapter 5.3]. This type of truncated trace could be avoided by extracting only the traces where no event happens after the extraction date. However, once the data is extracted, we cannot know which traces are truncated. As another example, incomplete traces can exist because the events have not happened yet. This is especially relevant when working with streaming data. Finally, truncated traces can result from a wrong execution (e.g., the ticket was supposed to be closed but the agent forgot to do it) or when the information system fails. In the next section, we introduce a classifier to automatically detect truncated traces.

3 Truncated Trace Classifier

A TTC inputs the current execution of a trace and predicts whether it is truncated. As shown in Table 1, we generate one input sample and one target for each prefix length of each trace. The input sample represents the current state of the process on which we apply a TTC. The target is a binary label that is 'true' when the trace is truncated or 'false' otherwise.

This setting implies that 'real' truncated traces that we would like to identify as such using the TTC would be labeled as complete. However, our intuition is that the model will also learn from similar complete traces where the truncated parts will be labeled as 'truncated'. For illustration purposes, let us define the following event logs: $\langle abc^3, ab \rangle$. During the training phase, the sequence ab

appears three times as 'truncated' and once as 'complete'. Hence, during the prediction phase, the sequence *ab* would most likely be predicted as 'truncated'.

Trace	#	Input Sample	Target: (Truncated?)
<abc>	1	a	true
	2	ab	true
	3	abc	false
<acadd>	4	a	true
	5	ac	true
	6	aca	true
	7	acad	true
	8	acadd	false

Fig. 1. The traces $\langle abc \rangle$ and $\langle acadd \rangle$ result in eight samples.

To build a classifier, we need to map the input sample to a feature space. There are several options to do so, covered in depth in [8,17–19]. We provide non-exhaustive examples that are illustrated in Fig. 2.

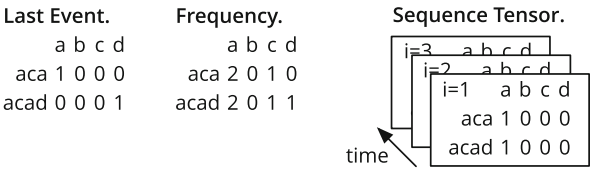

Fig. 2. Illustration of the feature spaces for the input samples $\langle aca \rangle$ and $\langle acad \rangle$.

Last Event. Relying only on the last event to predict that a trace is truncated is one option often mentioned in the literature [5–8]. For example, the input for sample #3 from Fig. 1 would be 'c'.

Frequency. The 'frequency' feature space counts the occurrences of each event. As shown in Fig. 2, $\langle aca \rangle$ becomes {a:2,b:0,c:1,d:0}. This feature space does not record the order in which the events appear.

Sequence Tensor. A sequence tensor contains an extra 'timestep' dimension. Each timestep is a matrix similar to the 'last event'; i.e., it describes which event happens. The extra dimension allows to describe the full sequence of timesteps in a lossless way. The number of timesteps is equal to the longest sequence in the event logs.

Once the input samples have been mapped into a feature space, it can be fed together with the target to a classifier. We propose five TTC implementations depicted in Fig. 3. As can be seen in Fig. 3, We also add a few base features: (1) the number of activities in the prefix, (2) the number of seconds since the first event in the trace, and (3) the number of seconds since the previous event in

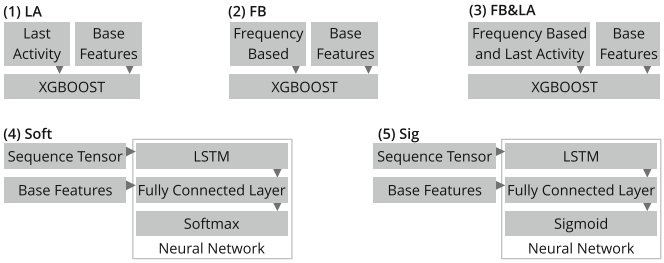

Fig. 3. Five implemented TTCs.

the trace. Such extra features were also added in the predictive business process monitoring proposed by Tax et al. [3]. The five TTCs are described below.

1 LA (Last Activity). This TTC relies on the last activity to predict that the trace is truncated.

2 FB (Frequency Based). This TTC uses the 'frequency' feature space described in Sect. 3.

3 FB&LA. This TTC concatenates the TTCs '1 LA' and '2 FB' because they both convey complementary information.

4 Soft (Softmax). This TTC corresponds to a next event prediction algorithm. In fact, the implementation is similar to the predictive business process monitoring from Tax et al. [3]. Predicting which event will occur is a multi-class prediction problem. Thus, we rely on the Softmax function because it transforms the output to a probability distribution. The end of the process is treated as any other event. If the latter is predicted as the most likely next event, we predict that the trace is complete. If not, we predict that the trace is truncated.

5 Sig (Sigmoid). The TTC '5 Sig' turns the multi-class problem into a binary one by using a one-vs-all strategy with the special 'end' event. We implemented both TTCs to compare the accuracy when the neural network is specially trained to recognize truncated traces ('5 Sig') or when the task is to predict the next event ('4 Soft').

TTCs 1 to 3 use XGBoost[1] [20], which stands for eXtreme Gradient Boosting. It relies on an ensemble of decision trees to predict the target. This technique is widely used among the winning solutions in machine learning challenges [20]. For the main settings, we set the number of trees to 200 and the maximum depth of the trees to 8. The last two TTCs rely on a neural network implemented in Keras [21]. As shown in Fig. 3, the architecture has two inputs. First, the sequence tensor is passed to a Long Short-Term Memory (LSTM) network. LSTM is a special type of Recurrent Neural Network (RNN) introduced in [22]. Compared to RNN, LSTM possesses a more advanced memory cell that gives LSTM powerful modeling capabilities for long-term dependencies [3]. The output of the LSTM network and the base features are provided to a fully connected

[1] Available at https://github.com/dmlc/xgboost/tree/master/python-package.

layer. Both the LSTM network and the fully connected layer have 16 cells. We use Adam [23] as an optimizer and we set the number of epochs to 100.

4 Benchmark

In this section, we benchmark the five TTCs described in the previous section, in addition to a baseline approach.

4.1 Datasets

We used 13 event logs[2] well known in the process mining literature. The event logs come from "real-life" systems, offering the advantage of containing complex traces and a wide range of characteristics visible in Table 1.

To the best of our knowledge, these event logs do not contain truncated traces. However, this is difficult to confirm. For instance, exceptional events might happen several months after the event log extraction date. In general, without having a deep expertise of the domain under analysis and direct access to the person in charge of the dataset extraction, it is not possible to guarantee that all traces are complete. We use the term 'false complete' to refer to traces that we wrongly consider complete during the training phase but that are in fact truncated because more events will happen. We claim that a TTC should be resilient to 'false complete'. In other words, a TTC should not overfit on a single 'false complete' and wrongly classify all similar traces as complete.

To test the resilience of the TTCs, we generated 0%, 10%, and 20% of 'false complete' traces by randomly cutting them. The setting with 0% of 'false complete' reflects how the TTC should be used with a real dataset, i.e., considering all the traces as complete. For the two other settings, we kept track of the traces that are truncated and refer to them as 'ground truth'. To benchmark the various TTCs, we use the ground truth. For instance, let us define that $\langle abc \rangle$ is a complete trace that we randomly cut to become the following 'false complete': $\langle ab \rangle$. During the training phase, we train the classifier to consider $\langle ab \rangle$ as a complete trace, while during the evaluation $\langle ab \rangle$ should be classified as truncated to be well classified.

4.2 Baseline: Decreasing Factor

Standard process mining tools and libraries such as the plugin 'Filter Log using Simple Heuristics' in ProM[3], the software Disco[4] or the Python library PM4Py [24] offer some options to remove truncated traces. Typically, a set of ending activities is selected by the end-user and the traces that do terminate with the ending activities are considered truncated and removed. It is also possible to automatically determine the set of ending activities. Let S be the set of

[2] Downloaded from https://data.4tu.nl/repository/collection:event_logs_real.
[3] http://www.promtools.org.
[4] https://fluxicon.com/disco/.

Table 1. Characteristics of the 13 datasets.

Dataset	#σ	# activities	Unique activities	Max length (σ)	Min length (σ)	Mean length (σ)
BPI_12	13.1K	164.5k	23	96	3	12.6
BPI_13_CP	1.5K	6.7K	7	35	1	4.5
BPI_13_i	7.6K	65.5k	13	123	1	8.7
BPI_15_1	1.2K	52.2K	398	101	2	43.6
BPI_15_2	0.8K	44.4K	410	132	1	53.3
BPI_15_3	1.4K	59.7K	383	124	3	42.4
BPI_15_4	1.1K	47.3K	356	116	1	44.9
BPI_15_5	1.2K	59.1K	389	154	5	51.1
BPI_17	31.5K	561.7K	26	61	8	17.8
BPI_18	2.0K	123.3K	129	680	35	61.9
BPI_19	251.7K	1.6M	42	990	1	6.3
Env_permit	0.9K	38.9K	381	95	2	41.6
Helpdesk	3.8K	13.7K	9	14	1	3.6

ending activities that we will use to filter the truncated traces. As a baseline, we use the method implemented in PM4Py which works as follows: First, the number of occurrences of each activity as a last activity is counted. Let C_i be the count of the i^{th} most frequent end activity. We start by adding the most frequent end activity, C_1, to S. Then, we calculate the decreasing factor of the next most frequent activity using the following formula: C_i/C_{i-1}. If the decreasing factor is above a defined threshold we add C_i to S and move to next most frequent activity. If the threshold is not met, we stop the process. We tried the following thresholds: 0.40, 0.45, 0.50, 0.55, 0.60, 0.65, and 0.70. We report the results obtained using a threshold of 0.60 as it is the one that yields the best accuracy to detect truncated traces. Interestingly, it is also the default value in PM4Py.

4.3 Evaluation

The first 80% of the traces were used to train the model, and the evaluation was done on the remaining 20%. Out of the 80% of training data, 20% was used to validate the parameters. To compare the ground truth with the output of the TTC, we used the Matthews Correlation Coefficient (MCC) [25]. The MCC has the nice property of using all the quantities of the confusion matrix, i.e., True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). Its value lies between −1 and 1, where 0 represents a random prediction, 1 a perfect score, and −1 an inverse prediction. It is defined as:

$$MCC(\sigma) = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FN)(FP + FN)(TN + FP)(TN + FN)}}$$

Figure 4 aggregates the results per TTC, while Fig. 5 contains the detailed results. In Fig. 5, we can see a large MCC score gap per dataset. This gap highlights the various levels of complexity involved in detecting truncated traces. Also, none of the techniques always outperforms the others. This is in line with a similar conclusion that was drawn from large predictive business process monitoring experiments [26]. Nonetheless, when looking at Fig. 4, we observe that the TTC '3 FB&LA' has the highest median MCC score. Interestingly, the performance of the baseline is comparable to the best implementations for the following five datasets: BPI_13_CP, BPI_13_i, BPI_18, Env_permit, Helpdesk (see Fig. 5). For the other eight datasets, there is a clear drop in performance between the baseline and more sophisticated methods. Looking at Table 1, we do not see any clear dataset characteristics to explain the performance gap. We conclude that looking at the last activity might work well, but for some datasets it is better to use a more sophisticated TTC.

We also tested the null hypothesis that the results from different TTCs come from the same distribution. To do this, we ran a permutation test with 100,000 random permutations and a p-value of 0.05. The results are visible in Fig. 6. As can be seen, '3 FB&LA' outperforms the baseline approach with strong statistical significance. We also observe that transforming a multi-class problem into a binary classifier–using the '4 Soft' instead of the '5 Sig'–does not seem to improve the ability of the TTC to detect truncated traces, as the MCC scores of Fig. 4 are comparable.

Figure 7 compares the execution time per TTC. The baseline takes in the order of milliseconds to run. TTCs that are based on XGboost take in the order of seconds or minutes to run, while approaches that rely on neural networks take from minutes to hours to run. In fact, the '4 Soft' and '5 Sig' are on average 112 times slower than the other TTCs that rely on XGBoost.

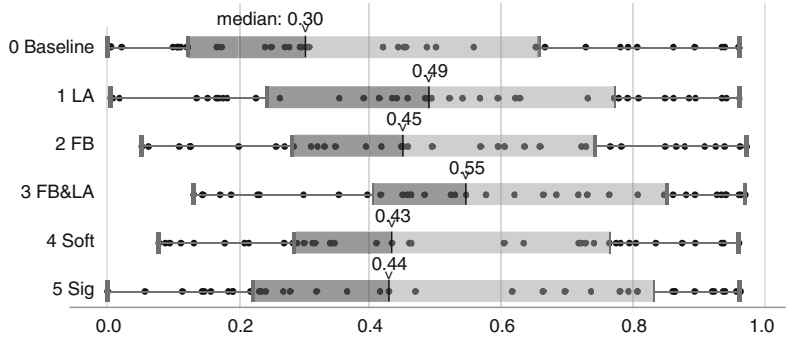

Fig. 4. Boxplot showing the MCC scores per technique. Each dot depicts an individual value. The median is written on top.

	Noise: 0						Noise: 0.1						Noise: 0.2					
	0 Baseline	1 LA	2 FB	3 FB&LA	4 Soft	5 Sig	0 Baseline	1 LA	2 FB	3 FB&LA	4 Soft	5 Sig	0 Baseline	1 LA	2 FB	3 FB&LA	4 Soft	5 Sig
BPI_12	.28	.89	.97	.97	.88	.92	.27	.85	.93	.93	.72	.86	.25	.78	.85	.85	.77	.78
BPI_13_CP	.96	.96	.13	.96	.96	.96	.94	.94	.13	.94	.94	.94	.87	.87	.13	.87	.84	.87
BPI_13_i	.80	.52	.33	.53	.80	.81	.78	.62	.61	.62	.78	.79	.73	.57	.57	.58	.73	.74
BPI_15_1	.22	.49	.45	.46	.22	.00	.09	.23	.35	.46	.13	.15	.13	.27	.31	.40	.11	.06
BPI_15_2	.26	.49	.72	.72	.28	.32	.25	.41	.63	.69	.30	.15	.21	.35	.60	.55	.21	.18
BPI_15_3	.12	.15	.11	.19	.09	.22	.12	.18	.06	.23	.10	.16	.11	.14	.05	.23	.08	.12
BPI_15_4	.00	.01	.28	.15	.46	.28	.00	.01	.27	.13	.35	.27	.00	.02	.26	.15	.27	.19
BPI_15_5	.13	.19	.29	.30	.34	.23	.13	.17	.20	.18	.32	.23	.12	.17	.32	.36	.32	.19
BPI_17	.68	.79	.97	.97	.78	.96	.67	.77	.94	.94	.72	.92	.64	.73	.88	.88	.74	.86
BPI_18	.50	.50	.50	.52	.18	.23	.49	.49	.45	.49	.47	.24	.46	.39	.46	.47	.34	.37
BPI_19	.32	.63	.77	.77	.61	.70	.31	.60	.73	.73	.64	.66	.30	.54	.66	.67	.41	.62
Env_permit	.45	.46	.42	.46	.34	.43	.44	.44	.45	.45	.44	.47	.42	.44	.40	.42	.29	.41
Helpdesk	.94	.94	.91	.94	.94	.94	.90	.90	.87	.90	.90	.90	.81	.81	.78	.81	.81	.81

Fig. 5. Detailed MCC score.

	0 Baseline	1 LA	2 FB	3 LA&FB	4 Sig	5 Soft
0 Baseline		.147	.141	.008	.146	.151
1 LA	.147		.986	.204	.929	.974
2 FB	.141	.986		.206	.941	.959
3 LA&FB	.008	.204	.206		.268	.188
4 Sig	.146	.929	.941	.268		.906
5 Soft	.151	.974	.959	.188	.906	

Fig. 6. P-values of the permutation tests between the MCC scores per techniques.

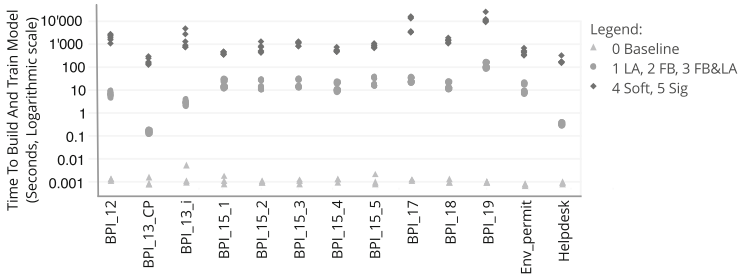

Fig. 7. Execution time for the five TTCs. The vertical axis uses a logarithmic scale.

The full benchmark implementations, the parameters, and the event logs, as well as the results are available online[5]. The machine used for the experiment has 61GB of RAM, 4 CPUs, and a GPU that speeds up the neural network training phase.

5 Improving Discovered Process Models with a TTC

A process discovery algorithm discovers a process model from an event log [6]. Because the discovered process model is based on event logs, it offers the advantage of being a data-driven approach that shows how the process is really executed. However, discovering a process model from an event log is a challenging task. Typically, process discovery algorithms are sensitive to noise [10–13]. Applying process mining techniques on traces that must supposedly be complete but are instead truncated is no exception. The quality of a process model is commonly measured using four competing metrics [12]: (1) The *precision* measures to what extent behaviors that were not observed can be replayed on the process model. (2) The *fitness* measures to what extent the traces from the event logs can be replayed on the model. (3) The *generalization* ensures that the model does not overfit. Finally, (4) the *simplicity* measures the complexity involved to read the process model. When facing truncated traces, a process discovery algorithm will wrongly infer that the process can be stopped in the middle. This will negatively impact the precision of the discovered process model. To solve this issue, researchers advocate removing truncated traces [5,6,8]. As highlighted by Conforti et al., "[t]he presence of noise tends to lower precision as this noise introduces spurious connections between event labels" [11].

We ran an experiment to measure the impact of removing truncated traces on the quality of the process models using PM4Py [24], a process mining library in Python. We used the default metrics in PM4Py which are described in the following papers: precision [27], fitness [6, p. 250], generalization [28], and simplicity [29]. To start, we randomly generated 100 process models with the PM4py

[5] https://github.com/gaelbernard/truncated-trace-classifier.

implementation of PTandLogGenerator [30], using the default parameters[6]. For each process model, we produced an event log containing 1,000 traces. We produced 20 variations of each event log with a level of noise ranging from 0 to 1, where 0 means that no traces were truncated and 0.05 means that 5% of the traces were truncated, and so on. Altogether, this process produced 20,000 event logs. For each of the 20,000 event logs, we ran the following experiment: (1) We applied the Inductive Miner [15] on the event logs to discover a process model, and (2) then we replayed the original event log–that did not contain the truncated traces–on the process model to measure the quality of the discovered process models. This was the experiment without a TTC. For the experiment with a TTC, we applied the exact same steps, but, beforehand, we automatically removed the truncated traces with the TTC '3 FB&LA' described in Sect. 3.

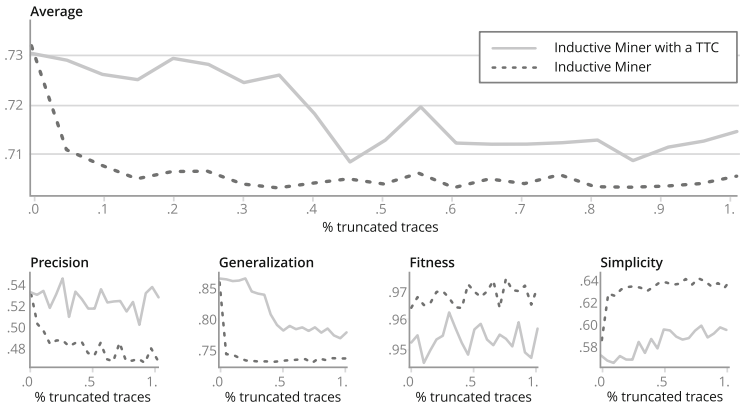

Fig. 8. Impact of the truncated traces on the process model qualities.

In Fig. 8, the results are averaged. As can be seen, the precision and the generalization metrics are greatly improved, while the simplicity and the fitness dimensions are negatively impacted. Table 2 shows that the average process quality is improved by 1.7%. Figure 8 shows the link between the ratio of truncated traces in the event logs and the quality of the resulting process models. The average process quality visible at the top of Fig. 8 shows that when there are some truncated traces in the event logs, applying the TTC is always beneficial. In the next section, we show that a TTC can also increase the prediction of the next events.

Table 2. Mean process quality of the discovered process models with and without the TTC.

	Without ttc	With ttc	Increase
Precision	0.4829	0.5267	9.10%
Generalization	0.7413	0.8112	9.40%
Fitness	0.9687	0.9523	−1.70%
Simplicity	0.6328	0.5833	−7.80%
Average	**0.7064**	**0.7184**	**1.70%**

[6] Visible at: https://pm4py.fit.fraunhofer.de/documentation#process-tree-generation.

6 Improving Next Event Prediction with a TTC

The goal of a next event algorithm is to predict the most likely event that will follow a truncated trace. As shown with the '4 Soft' TTC, we can turn a next event algorithm into a TTC. Looking at the benchmark in Fig. 4, we show that the TTC '3 FB&LA' outperforms the TTC '4 Soft'. We have the intuition that combining the best TTC with a next event algorithm will increase the prediction accuracy. The rationale is that we will more accurately predict the end of the process with a TTC because it has been trained for this purpose. Hence, we first rely on the TTC to predict if more events are expected. If not, we do not need to call the next event algorithm. Overall, we should improve the results as we avoid predicting a next event when the trace is not truncated. The goal of this section is to validate this hypothesis.

As it is initially a next event prediction algorithm, we use the TTC '4 Soft' for the prediction of the next event. In the setting without a TTC, it is the only algorithm involved. In the version with a TTC, we complement the architecture with the TTC '3 FB&LA' in the following way. First, we assess if the trace is truncated using the TTC. If the trace is truncated, we predict the next event. Conversely, if the trace is already complete, we do not need to predict the next event. The results are visible in Table 3. Including a TTC improves the accuracy by up to 7.5% and on average by 1.4%. In the experiment, building the TTC took an extra 2.4% of duration. We claim that including a TTC is beneficial for the accuracy while having a limited negative execution time impact. A TTC solves one problem noted by Tax et al.: "We found that LSTMs have problems [...] to predict overly long sequences of the same activity, resulting in predicted suffixes that are much longer than the ground truth suffixes" [3].

7 Related Work

To the best of our knowledge we are the first to focus on the task of distinguishing truncated from complete traces. Still, existing works—especially in the area of predictive process monitoring—are relevant to uncover truncated traces.

Predictive process monitoring anticipates whether a running process instance will comply with a predicate [4]. For instance, a predicate might be about the process execution time, the execution of a specific event, or the total amount of sales. As highlighted by Verenich et al., techniques in this space differ according to their object of prediction [18]. A TTC is a specific type of predictive process monitoring task where the predicate is whether we will observe more events. In [31], Maggi et al. propose a generic predictive process monitoring approach. Once the predicate is set, the most similar prefixes are selected based on the edit distance. Finally, a classifier is used to correlate the goal with the data associated with the process execution. Insights are then provided to the end-user to optimize the fulfillment of the goal while the process is being executed. It was later extended with a clustering step to decrease the prediction time [4]. Tax et al. propose a neural network that leverages LSTM that could serve as

Table 3. Comparing the accuracy of predicting the next event and the execution time, without and with a TTC.

Dataset	Accuracy			Execution time (in seconds)		
	Without ttc	With ttc	Increase	Without ttc	With ttc	Increase
BPI_12	0.6899	0.6896	–	1060	1066	0.6%
BPI_13_CP	0.5559	0.5559	–	172	172	–
BPI_13_i	0.6478	0.6486	0.1%	519	521	0.4%
BPI_15_1	0.0851	0.0915	7.5%	324	347	6.6%
BPI_15_2	0.1509	0.1584	5.0%	606	628	3.5%
BPI_15_3	0.1364	0.1364	–	851	873	2.5%
BPI_15_4	0.1385	0.1394	0.6%	366	383	4.4%
BPI_15_5	0.0873	0.0884	1.3%	634	663	4.4%
BPI_17	0.7642	0.7689	0.6%	2318	2342	1.0%
BPI_18	0.6277	0.6323	0.7%	786	804	2.2%
BPI_19	0.4755	0.4855	2.1%	8674	8797	1.4%
Env_permit	0.2325	0.2324	–	373	388	3.9%
Helpdesk	0.8226	0.8226	–	116	116	–

another generic predictive process monitoring algorithm capable of fitting different predicates [3]. In our work, we use the approach from Tax et al. as a baseline (i.e., TTC '4 Soft'). Despite the advantage of being generic, we show that a tailor-made algorithm to detect a TTC outperforms such an approach.

The goal of a business process deviance mining algorithm is to assign a binary class–normal or deviant–to a trace. In this sense, it shares similarities with predictive process monitoring. This is especially true because of their overlapping inputs and feature extraction methods [32]. However, deviance mining works on completed instances and focuses on the why [32].

Finally, Bertoli et al. propose a reasoning-based approach to recover missing information from event logs [33]. Ultimately, it would allow us to turn a truncated trace into a complete one. To work, this technique requires a reference process model as input. Therefore, it is not applicable if the task at hand is to discover a process model.

8 Conclusion

Event logs are often noisy, which makes the application of process mining sometimes difficult in a real setting [13]. Typically, the existence of truncated traces is known. Still, there is a research gap in systematically detecting them. In this work, we treat the identification of truncated traces as a predictive process monitoring task and we benchmark several TTCs using 13 complex event logs. We show that building a TTC that consistently achieves high accuracy is challenging. This finding highlights the importance of conducting further research to

build an efficient TTC. Typically, for some event logs, using a baseline approach that relies solely on the last activity works well. Still, we show that the TTC '3 FB&LA' outperforms such baseline approach with strong statistical significance.

We also measure the process model quality impact when a process discovery algorithm is run on event logs that contain truncated traces. We show that only a few truncated traces can greatly decrease the process model quality and that a TTC can alleviate this problem by automatically removing truncated traces. Finally, we highlight the unexplored potential of a TTC to increase the accuracy of predicting the next event. We expect that more benefits of TTCs are yet to be discovered, especially in the predictive business process monitoring area.

In this work, we use the sequence of activities as well as some timing information. Using more information such as the name of the resource, the day of the week or any other event attributes could further improve the accuracy of the TTCs. Higher accuracy could also be achieved by using different classifiers, trying new neural network architecture, or implementing alternative feature spaces. This is an area for future research where our work can serve as a baseline.

References

1. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from incomplete event logs. In: Ciardo, G., Kindler, E. (eds.) PETRI NETS 2014. LNCS, vol. 8489, pp. 91–110. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07734-5_6
2. Bernard, G., Andritsos, P.: Accurate and transparent path prediction using process mining. In: Welzer, T., Eder, J., Podgorelec, V., Kamišalić Latifić, A. (eds.) ADBIS 2019. LNCS, vol. 11695, pp. 235–250. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-28730-6_15
3. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 477–492. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_30
4. Di Francescomarino, C., Dumas, M., Maggi, F.M., Teinemaa, I.: Clustering-based predictive process monitoring. IEEE Trans. Serv. Comput. **12**, 896–909 (2016)
5. Bezerra, F., Wainer, J., van der Aalst, W.M.P.: Anomaly detection using process mining. In: Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Soffer, P., Ukor, R. (eds.) BPMDS/EMMSAD -2009. LNBIP, vol. 29, pp. 149–161. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01862-6_13
6. Aalst, W.: Data science in action. Process Mining, pp. 3–23. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4_1
7. Fluxicon: Deal with incomplete cases, process mining in practice, disco 2.1 documentation. http://processminingbook.com/incompletecases.html
8. Verenich, I.: Explainable predictive monitoring of temporal measures of business processes. Ph.D. thesis, Queensland University of Technology (2018)
9. Carmona, J., de Leoni, M., Depaire, B.: Process discovery contest. In: International Conference on Process Mining 2019 (2019)
10. Suriadi, S., Andrews, R., ter Hofstede, A.H., Wynn, M.T.: Event log imperfection patterns for process mining: towards a systematic approach to cleaning event logs. Inf. Syst. **64**, 132–150 (2017)

11. Conforti, R., La Rosa, M., ter Hofstede, A.H.: Filtering out infrequent behavior from business process event logs. IEEE Trans. Knowl. Data Eng. **29**(2), 300–314 (2016)
12. van der Aalst, W., et al.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM 2011. LNBIP, vol. 99, pp. 169–194. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28108-2_19
13. Bose, R.J.C., Mans, R.S., van der Aalst, W.M.: Wanna improve process mining results? In: 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), pp. 127–134. IEEE (2013)
14. Selig, H.: Continuous event log extraction for process mining (2017)
15. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Lohmann, N., Song, M., Wohed, P. (eds.) BPM 2013. LNBIP, vol. 171, pp. 66–78. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06257-0_6
16. de Medeiros, A.K.A., Weijters, A.J., van der Aalst, W.M.: Genetic process mining: an experimental evaluation. Data Min. Knowl. Disc. **14**(2), 245–304 (2007). https://doi.org/10.1007/s10618-006-0061-7
17. Verenich, I., Dumas, M., La Rosa, M., Maggi, F.M., Di Francescomarino, C.: Complex symbolic sequence clustering and multiple classifiers for predictive process monitoring. In: Reichert, M., Reijers, H.A. (eds.) BPM 2015. LNBIP, vol. 256, pp. 218–229. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42887-1_18
18. Verenich, I., Dumas, M., La Rosa, M., Maggi, F.M., Chasovskyi, D., Rozumnyi, A.: Tell me what's ahead? Predicting remaining activity sequences of business process instances (2016)
19. Leontjeva, A., Conforti, R., Di Francescomarino, C., Dumas, M., Maggi, F.M.: Complex symbolic sequence encodings for predictive monitoring of business processes. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) BPM 2015. LNCS, vol. 9253, pp. 297–313. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23063-4_21
20. Chen, T., Guestrin, C.: Xgboost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794. ACM (2016)
21. Chollet, F.: Keras (2015). https://github.com/fchollet/keras
22. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
23. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
24. Berti, A., van Zelst, S.J., van der Aalst, W.: Process mining for python (PM4PY): bridging the gap between process-and data science. arXiv preprint arXiv:1905.06169 (2019)
25. Matthews, B.W.: Comparison of the predicted and observed secondary structure of t4 phage lysozyme. Biochimica et Biophysica (BBA)-Acta Protein Struct. **405**(2), 442–451 (1975)
26. Di Francescomarino, C., et al.: Genetic algorithms for hyperparameter optimization in predictive business process monitoring. Inf. Syst. **74**, 67–83 (2018)
27. Muñoz-Gama, J., Carmona, J.: A fresh look at precision in process conformance. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 211–226. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15618-2_16
28. Buijs, J.C., van Dongen, B.F., van der Aalst, W.M.: Quality dimensions in process discovery: the importance of fitness, precision, generalization and simplicity. Int. J. Coop. Inf. Syst. **23**(01), 1440001 (2014)

29. Blum, F.: Metrics in process discovery. Technical report, TR/DCC. 1–21 (2015)
30. Jouck, T., Depaire, B.: Ptandloggenerator: a generator for artificial event data (2016)
31. Maggi, F.M., Di Francescomarino, C., Dumas, M., Ghidini, C.: Predictive monitoring of business processes. In: Jarke, M., et al. (eds.) CAiSE 2014. LNCS, vol. 8484, pp. 457–472. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07881-6_31
32. Nguyen, H., Dumas, M., La Rosa, M., Maggi, F.M., Suriadi, S.: Business process deviance mining: review and evaluation. arXiv preprint arXiv:1608.08252 (2016)
33. Bertoli, P., Di Francescomarino, C., Dragoni, M., Ghidini, C.: Reasoning-based techniques for dealing with incomplete business process execution traces. In: Baldoni, M., Baroglio, C., Boella, G., Micalizio, R. (eds.) AI*IA 2013. LNCS (LNAI), vol. 8249, pp. 469–480. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-03524-6_40