Detection and Interactive Repair of Event Ordering Imperfection in Process Logs

Prabhakar M. Dixit[1,2(✉)], Suriadi Suriadi[2], Robert Andrews[2], Moe T. Wynn[2], Arthur H. M. ter Hofstede[2], Joos C. A. M. Buijs[1], and Wil M. P. van der Aalst[1,2]

[1] Eindhoven University of Technology, Eindhoven, The Netherlands
{p.m.dixit,j.c.a.m.buijs,w.m.p.v.d.aalst}@tue.nl
[2] Queensland University of Technology, Brisbane, Australia
{s.suriadi,r.andrews,m.wynn,a.terhofstede}@qut.edu.au

Abstract. Many forms of data analysis require timestamp information to order the occurrences of events. The *process mining* discipline uses historical records of process executions, called event logs, to derive insights into business process behaviours and performance. Events in event logs must be ordered, typically achieved using timestamps. The importance of timestamp information means that it needs to be of high quality. To the best of our knowledge, no (semi-)automated support exists for detecting and repairing ordering-related imperfection issues in event logs. We describe a set of timestamp-based indicators for detecting event ordering imperfection issues in a log and our approach to repairing identified issues using domain knowledge. Lastly, we evaluate our approach implemented in the open-source process mining framework, ProM, using two publicly available logs.

Keywords: Event log imperfection · Event ordering · Data quality

1 Introduction

Process mining [1], a form of data mining, utilises historical records of process executions to derive insights into business process performance and behaviour. Central to all process mining techniques is the "event log". An event log is a set of records of processing steps (events) carried out during multiple process instances (cases). Each event is minimally characterised by attributes identifying the process instance to which it belongs *(case id)*, the process step that was carried out *(activity)* and an attribute, typically a *timestamp*, that allows the events in the case to be ordered. Event records may optionally contain other attributes such as the person that carried out the process step.

The saying *garbage in - garbage out* alludes to the fact that, for all forms of data-to-information transformations, process mining included, poor quality input data

J. Krogstie and H. A. Reijers (Eds.): CAiSE 2018, LNCS 10816, pp. 274–290, 2018.
https://doi.org/10.1007/978-3-319-91563-0_17

leads to poor quality information. Many forms of data analyses, especially time-series analyses, e.g., stock market prediction and weather forecasting, need quality timestamp information. Likewise, within the domain of process mining, timestamps are the principal means for ordering activities within cases. Correctly ordering activities within a case is critical to deriving accurate process models (discovery), to determining whether activities were carried out in accordance with organisational expectations (conformance) and to determine whether activities and cases are executed efficiently (performance). Thus, it is clear that the quality of process mining results is contingent on the quality of timestamp data. For instance, consider a situation, where activities in an event log are drawn from two different information systems, one of which records timestamps at only day level granularity, while the other records timestamps to the milli-second. The effects on process mining analysis (see Fig. 1 for an example), include discovered process models which do not reflect the actual ordering of events.

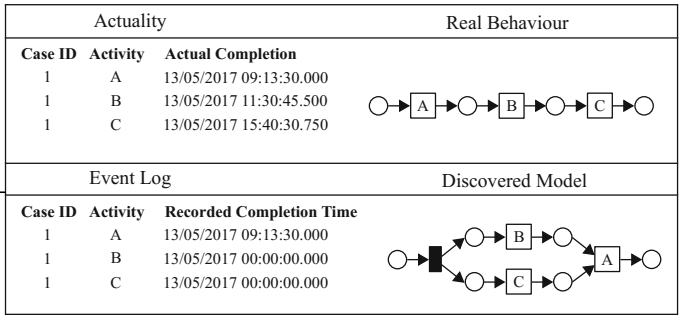

Fig. 1. Effect of inaccurate timestamps on event ordering.

The importance of timestamps in process mining analysis is well recognised [2]. A variety of authors have identified timestamp-related data quality issues and their impact on process mining [1,3–6]. As incorrect ordering of events can have adverse effects on the outcomes of process mining analysis, we provide a set of relevant indicators to detect ordering related problems in an event log to pinpoint those activities that *might* be incorrectly ordered.[1] Next, we let the user repair the event log by interactively injecting domain knowledge directly into the event log, with the help of process fragments, as well as analyze the impact of the repaired log. While tools for automated and user-driven detection and repair of time-oriented data exist, e.g. TimeCleanser [7], to our knowledge, this is the first technique that provides consolidated detection and repair mechanisms to deal with data quality-related ordering issues in event logs.

In this paper, Sect. 2 describes the *indicators* of event ordering imperfection data quality issues, Sect. 3 outlines our approach to detect, repair, and analyze the impact of repair of the event log, and Sect. 4 describes our implementation. Section 5 evaluates the tool against two publicly-accessible data sets. Related work is provided in Sect. 6, followed by conclusion in Sect. 7.

[1] These indicators may be specific to event logs; however, some of the indicators that we propose are generic enough to be applicable to other time-ordered data.

2 Indicators of Event Ordering Imperfection

The approach proposed in this paper aims to enable domain experts to detect and repair event-ordering imperfection that arises due to timestamp-related issues. As a first step, we need to be able to *locate exactly where timestamp issues may exist in a log* by recognising characteristics commonly associated with event ordering imperfection. From existing literature that describe data quality issues in event logs [1,2,4,5,8] in particular, and time-oriented data in general [3], we abstract three classes of indicators that may be used to locate event ordering issues. It is not the intention of this paper to provide a comprehensive list of indicators of event ordering imperfection; rather, our aim here is to highlight the importance of recognising the various indicators of event ordering imperfection from an event log as a starting point for our log repair approach. Note that the existence of these indicators does not automatically mean that the log has event ordering issues. However, the ability to highlight those activities will assist domain experts in selecting the best repair actions, if necessary.

Granularity. One of the indicators of event ordering imperfection is the existence of either coarse timestamp granularity (e.g. *imprecise timestamp* [1,8]) or mixed timestamp value granularity (e.g. event log includes events from multiple systems where each system records timestamps differently [2,3]). The mixture of varying timestamp granularity may result in events being ordered incorrectly. For example, an event *'Seen by Doctor'* may be recorded at day-level granularity, e.g. '05 Dec 2017 00:00:00'. Within the same case, another event *'Register Patient'* may have second-level granularity, e.g. '05 Dec 2017 19:45:23'. The ordering of these two events will be *'Seen by Doctor'* followed by *'Register Patient'*, which is incorrect as it should have been the other way around.

Order Anomaly. Locating events affected by ordering imperfection can also be performed by identifying events exhibiting unusual temporal ordering, e.g. duplicate entry of exactly the same event [3]. Given an activity a_1 that was recorded twice (due to a user mistakenly double-clicking a button on his/her screen), we may observe an unusual (infrequent) directly-followed temporal order $a_1 \rightarrow a_1$, highlighting a potential event ordering imperfection. Issues related to missing events [1,4,8] and incorrect timestamps due to events being recorded post-mortem [1] or due to manual entry [8] can also be detected by learning if there exist other forms of unusual temporal orderings between activities.

Statistical Anomaly. Extracting more generic statistical anomalies, such as learning the temporal position of a particular activity in the context of other activities, or the distribution of timestamp values of all events in a log, may indicate the existence of timestamp-related problems. For example, when a log

is comprised of events from multiple systems, there may be more than one way in which timestamps are formatted, which may lead to the 'misfielded' or 'unanchored' timestamp problem [3,5] whereby timestamp values are interpreted incorrectly. A common situation is when timestamp values formatted as `DD/MM/YYYY` are interpreted as `MM/DD/YYYY`. In this situation, one may see imbalance distribution of the 'date' values of all events in the log whereby all events will have the date value between 1–12 only, and none for 13–31. There are other indicators of event ordering problems that can be detected through statistical anomaly, such as the use of batch processing [3,5] and multiple timezone problem [3].

3 Approach

We start with an event log and adopt an iterative approach (see Fig. 2) to addressing event order imperfection, cycling through (i) automated, indicator-based issue detection, (ii) user-driven repair, (iii) impact analysis, and (iv) log update. In this section we show, through examples, how our indicators are used to detect potential event-order issues, how identified issues are repaired using domain knowledge infused process-fragments and an alignment strategy. Lastly we describe a set of metrics for assessing the impact of the repair actions on the log which can guide the user in accepting or rejecting the repairs.

3.1 Detection

In this section, we discuss the three detection strategies employed in our tool to detect the three indicators presented earlier in Sect. 2. The output of detection is a consolidated list of possible time-oriented data quality issues, containing the actual problem and description, the affected activity(ies) and the number of instances affected. Our technique doesn't require any input, other than the event log, for detection of the issues. It should be noted that the aim of our detection technique is to provide hints on the probable ordering related issues in the event log, and the user may choose to ignore, even *whitelist*, some of the detected items from the list. These whitelisted items will not be shown in the subsequent detection cycles.

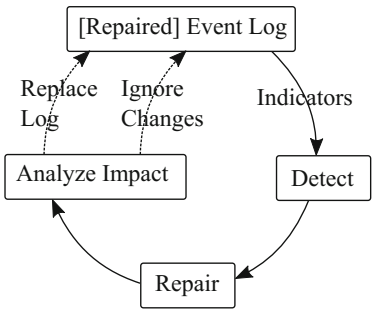

Fig. 2. Detect/repair approach

Table 1. Sample granularity table populated by scanning an event log.

Activity	Year	Month	Day	Hour	Minute	Second
a	0	0	0	1	2	500
b	0	0	0	450	20	0
c	0	0	0	0	5	350
d	0	0	0	2	7	448
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Granularity based detection makes use of a table where rows correspond to activities and columns correspond to the granularity of timestamps. Each cell represents the frequency (by finest recording granularity) of a particular activity in the event log. Table 1 gives an example of such a table, which clearly indicates that activity b is usually recorded at a coarser granularity (Hour) compared to the other activities. Every such b is added to the list of detected issues.

Ordering based detection is conceptually similar to the challenge faced by the process discovery techniques, which try to deduce the correct order of activities using an event log. Here we use pairwise causal relations between activities to *guess* the correct order of activities based on frequency thresholds. In order to achieve this, we populate two tables: one containing the directly *follows* relations and the other containing the directly *precedes* relations between the activities in the event log.

Table 2. An example *follows relations* table snapshot (left) and *precedes relations* table snapshot (right).

$\rightarrow$	a	b	c	d	e	f
⋮	⋮	⋮	⋮	⋮	⋮	⋮
c	3	0	0	150	200	25
⋮	⋮	⋮	⋮	⋮	⋮	⋮

$\leftarrow$	a	b	c	d	e	f
a	0	3	3	450	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮
f	0	0	25	0	0	0

Each cell contains the number of times an activity from a corresponding row, was directly followed/preceded by an activity from the corresponding column in the event log. For example, in the *follows* relations of Table 2, activity c is followed by activities a, d, e and f: 3, 150, 200 and 25 times respectively. The discrepancies in the ordering w.r.t. each activity are analyzed. For example, consider the activity c in Table 2 and a (user defined) threshold value of 0.8. The first step is to filter out the smallest set of activities which are directly followed by c resulting in at least 80% of the total occurrences of c.

In the *follows* table (Table 2), these are activities d and e. Next, for every remaining activity with non-zero frequency (a and f) not within the threshold, we check the corresponding *precedes* relations table with the threshold values for infrequent non zero activity relations. In the case of a, these would be activities b and c. Since both the directly follows and precedes relations ($c \rightarrow a$ and $a \leftarrow c$) are infrequent, we conclude that there is an issue in ordering between activities a and c, and thus these activities are added to the detected issues list. However, for activity f, activity c is within the threshold, i.e., all occurrences of f are preceded by c. Since activity f is highly infrequent (compared to c), the ordering between c and f is assumed to be correct. Hence by considering both the directly *follows* and *precedes* relations, we *detect* only those infrequencies

that are ordering related. A similar approach is followed for eventually *follows* and *precedes* relations to explore long term infrequent relations.

Statistical anomaly can be used to detect event ordering imperfection. In this paper, we investigated one type of statistical anomaly useful in detecting event ordering issues due to *mis-fielding* (see Sect. 2). For example, data extracted as `MM/DD/YY` from the system might be incorrectly processed as `DD/MM/YY` format in the event log. In this scenario, one would only witness days in the range 1–12 and not in the range 13–31. In order to evaluate such issues, we employ a directional statistical technique - Kuiper's test. Kuiper's test is especially useful in the scenarios when the data is circular. That is, while analyzing the hourly distribution of activity distribution, timestamp values 23:59 h, and 00:01 would be considered close. For more details about the Kuiper's test we refer the reader to [9] (p. 99). We employ Kuiper's test to compare the distribution of each activity with all the other activities in the event log, at 6 levels: hour of the day, minute of the hour, second of the minute, day of the week, day of the month and month of the year. The activity whose distribution is statistically significantly different ($p = 0.05$) compared to all the other activities, would then show up in the detection list.

3.2 Repair

Insights from the detection phase are coupled with domain knowledge to repair the event log. A single activity is repaired at a time. The repair workflow consists of four steps as shown in Fig. 3 and explained in the subsections that follow.

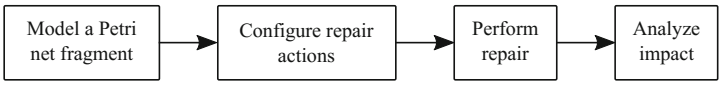

Fig. 3. The four steps of the process repair workflow

Modeling process fragments lets the user specify domain knowledge as a free choice Petri net fragment via an interactive Petri net editor. Minimally, the Petri net contains only the activity which should be repaired. The user can additionally include "other activities" in the Petri net with respect to which the activity should be repaired. For example, consider that the original expected model of Fig. 1 is unavailable. However, the user is aware that activity C should happen only after activity A. From the detection phase, the user would know that there is a problem with the granularity of activity C. Hence the user can decide to repair the activity C in relation to activity A, by modeling a simple sequential Petri net. It should be noted that in many real life scenarios, the event log contains multiple activities and the user may not be aware of relationship between *all* the activities. Hence we would like to support the user in modeling partial fragments of Petri

nets, explicitly specifying the relations between activities that the user is aware of. Modeling via Petri nets allows the possibility to include complex graphical fragments such as concurrency, choices, loops, duplications and introducing *silent* activities. Duplicate activities in the Petri net are uniquely numbered. After modeling the Petri net fragment, the next step is configuration of the repair options.

Table 3. The repair activity configuration options.

Activity	Description
Activity to repair	Specify the activity to be repaired from the Petri net fragment. For duplicate occurrences in the Petri net, select the (uniquely numbered) activity instance
Add events?	Specify if new activities should be artificially inserted in the event log. If set to true, the technique may insert artificial events (corresponding to the activity to be repaired) in the event log
Remove events?	Specify if activities should be removed from the event log. If set to true, the technique may remove existing events (corresponding to the activity to be repaired) from the event log

Repair Configuration. The repair configuration allows the user to specify the settings for correcting the ordering of a particular activity, and consists of: (i) the repair activity configuration, and (ii) the repair context configuration. The repair activity configuration deals with the activity to be repaired and is shown in Table 3. The repair context configuration provides the contextual information of how to correct an activity w.r.t. another activity, using the so-called repair actions. Each *repair action* consists of:

(i) **Anchor**: The timestamp of the activity to be repaired would be corrected based on a so-called anchor activity. Anchor activity is an activity from the Petri net which cannot be the same as the activity to be repaired. Alternatively, the anchor could be the start of a case (first activity) or the end of a case (last activity).
(ii) **Position**: Select whether the activity to be repaired should occur *before* the anchor or *after* the anchor activity.
(iii) **Value**: Specify the value of the new timestamp of the activity to be repaired in relation to the anchor. This could be an absolute value, e.g. 4 h, or a mean/median value of all the conforming relations from the event log between the anchor activity and the activity to be repaired per the modeled Petri net.

Multiple repair actions can be specified for repairing an activity. The order of repair actions determines the sequence in which repairs are performed in the event log, such that if the first action fails, the second one is applied and so on.

Perform Repairs. The actual repair is performed using the outcome from the alignments based conformance technique [10]. We demonstrate the use of alignments strategy to perform the actual repair of the event log with an example. Consider the ordering of an activity b is incorrect in an event log and needs to be fixed. Lets assume that the correct ordering of activity b in relation to two activities a and c is a sequential relation: $a \rightarrow b \rightarrow c$, i.e. activity a should be followed by b, which should be followed by c. This relation can easily be modeled using a Petri net. As a first step, the original event log is duplicated. Let's call this duplicated event log L_D. This duplicated event log and the Petri net fragment are used as input for the alignments. The alignments based conformance technique finds an optimal navigation path in a Petri net model for a particular sequence of activities. It does so by assigning a *cost* to each activity, which is then used for determining the correct ordering with an objective of minimizing the overall cost. All the activities which are not present in the Petri net fragment are assigned a cost of zero. In our example, as b is the activity to be repaired, it is assumed that the positioning of activity b is inaccurate compared to a and c. Hence a and c are assigned higher costs compared to b, i.e. it is preferred to align a or c at the expense of b. Now consider there is a case in the event log L_D, such that all the three activities (a, b and c) happen once and on the same day, however b happens at 08:00, a happens at 09:00 and c happens at 10:00. It is easy to see that this case does not fit the *expected* sequential behavior ($a \rightarrow b \rightarrow c$). The outcome of alignments for such a case would thus be:

log sequence	b	a	$>>$	c
model sequence	$>>$	a	b	c

In the alignments example above, activities a and c exhibit *synchronous* behavior per the Petri net fragment and the event log. The first occurrence of b (red) is a *move on log*, indicating a behaviour in the event log that cannot be replayed in the net. The second b (gray) is a *move on model*, indicating an expected occurrence of b as per the Petri net fragment which did not occur in the event log. Now, the ordering of b is corrected using each repair action from the context configuration box as follows:

1. For every *move on model* of b in the alignment of a particular case, change the timestamp of an event corresponding to a *move on log* of b for that case. The new timestamp is set based on the current repair action. A timestamp of an event can be updated only once for a particular repair action. It should be noted that, in case of Petri net patterns such as *loops*, the nearest *synchronous* anchor activity is chosen (in case the anchor is not case start or end). In our example above, the timestamp of the event corresponding to the first b is changed based on the configured repair action.
2. If *add events* is set to true: Let n be the difference between the number of *move on model* of b and the number of *move on log* of b for a particular case. If n is non-zero positive number, then add n number of artificial events of activity b, based on the chosen repair action.
3. If *remove events* is set to true: Let n be the difference between the number of *move on log* of b and the number of *move on model* of b for a case. If n is

non-zero positive number, then remove n number of events of activity b whose timestamp values were not changed (in step (a)), and whose alignment step corresponds to *move on log*.

4. Perform alignments on the repaired event log, and retrace and replicate all those cases with correct ordering of activity b in the event log L_D. Create a new event log using L_D containing only those cases which have incorrect ordering of activity b. Perform alignments on this newly created event log, and repeat the steps (a)–(d) until all the repair actions are performed. Alternatively, stop if the positioning of activity b for all the cases is correct.

3.3 Impact Analysis

Having repaired the event log, the user is presented with information about the impact of the repair on the event log. The user can choose to replace the current working copy of the event log with the repaired log (L_D), or choose to ignore the repaired event log. The metrics presented to the user to analyze the impact are: (i) edit distance: Levenshtein distance between the repaired event log and the original event log, (ii) fitness [10] of the original event log and the repaired event log for the activities from the Petri net fragment, (iii) the total number of cases impacted, (iv) the total number of events added, (v) the total number of events removed, and (vi) the total number of events with a changed timestamp value.

4 Implementation

The proposed techniques have been implemented in ProM[2], and can be used from the "Interactive Process Mining" package in the nightly build version of ProM. Figure 4 shows the detection screen of our tool. The user is presented with all the detected quality issues (along with their description, frequencies etc.) in a tabular format (panel A). On selecting an issue from the detection table, the user is provided with the filtered event log (panel B) which contains only the events relevant for the detected issue. Furthermore, the user is also provided with an aggregated histogram (panel C) showing the distribution of all the affected events based on the type of the detected issue.

Figure 5 shows the repair screen of our tool. The user models a Petri net interactively by adding one activity at a time (panel A). This Petri net specifies the relation between the activity to be repaired with other activities. The user specifies the repair strategy in repair configuration view (panel B). Upon performing the repair the impact is presented to the user (panel C) which can be used for the decision making of either keeping or discarding the changes.

[2] http://www.processmining.org/prom/start.

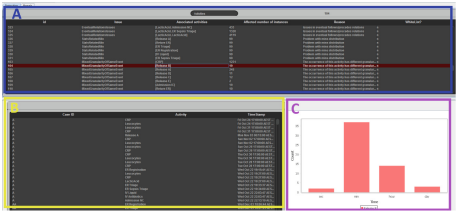

Fig. 4. The detection tab showing the three detection panels: (A) the list of detected issues, (B) the event log view, and (C) the graph view showing the distributions for the selected issue.

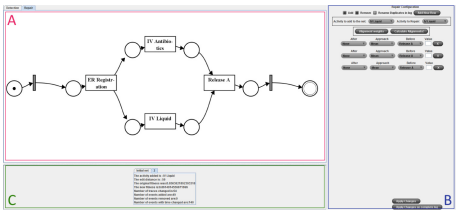

Fig. 5. The repair tab showing the three repair panels: (A) the Petri net modeled interactively by the user, (B) the repair configuration view, and (C) the impact of a (sequence of) repair(s).

5 Evaluation

Detecting and repairing event ordering anomalies involve re-structuring the event log to correct the control flow aspect. Hence for each event log, we demonstrate the detectability of ordering related issues in the event log, followed by repair of the event logs using the insights from detection along with domain knowledge. The final outcome is evaluated based on the process models discovered using the repaired event log. The event logs used are: (i) Sepsis event log - wherein the process model discovered from a repaired event log is compared with the ground truth process model, (ii) BPIC 2015 event log - wherein the outcomes of process discovery techniques are compared before and after the event log repair. From [11] and the reports submitted to BPIC 2015, we know that both the event logs have quality issues pertaining to event ordering. However, the exact problems and repairs were unknown and/or unavailable.

As detailed in Sect. 3, access to *domain knowledge* is essential in our approach. For the evaluation of our approach using the Sepsis log, domain knowledge was acquired through a process analyst who had extensive consultation with relevant domain experts [11]. For the evaluation using the BPIC 2015 log, domain knowledge was acquired by consulting the reports submitted to BPIC 2015. We acknowledge that our evaluation may be limited by the absence of *domain experts* who would be able to confirm the validity of the repaired log and to shed light on any peculiar, but still valid, phenomenon within the process being analysed that may be lost due to the repair action(s) applied. Nevertheless, even without access to domain experts, our approach is capable of improving the quality of the log as evidenced by the ability to generate a process model that is closer to the known ground-truth process model (for the Sepsis log) and of better process model quality (for the BPIC 2015 challenge log). The details of the evaluation of our approach are provided below.

5.1 Sepsis

The Sepsis log[3] [11] (1,050 patients/cases and 16 activities) contains the Sepsis treatment process of patients in a Dutch hospital. The "ground truth" process model (see Fig. 6) was hand-drawn in consultation with domain experts (see [11] for details).

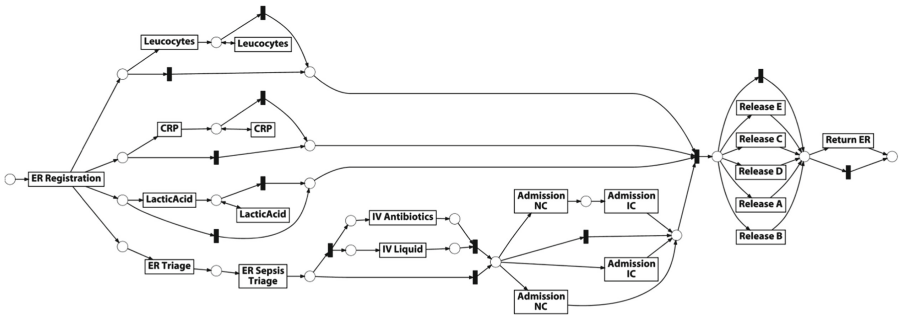

Fig. 6. Original Sepsis model for the event log as modeled in [11].

The detection outcome of our tool indicated that 96% of the issues were granularity and ordering related and the remaining 4% were issues arising due to statisical anomalies. Among the ordering related issues, almost 50% (14 out of 28) of the *direct* ordering problems had at least one of the three activities: *IV Antibiotics, IV Liquid* and/or *Admissions NC*, which hinted at a possible ordering problem with regards to these activities.

Here we give an example of the steps followed to repair one of the activities (*Admissions NC*). The Petri net fragment shown in Fig. 7 was modeled based on the domain knowledge about Sepsis protocols from the literature. In order to repair the activity *Admissions NC*, the position of *Admissions NC* was configured to be after (the mean duration of all the fitting cases) *IV Antibiotics* or *IV Liquids* for the mis-aligned cases. In total, the timestamp ordering of six activities was corrected (see Table 4). After repairing the time values (and removing duplicates) of the 6 activities, Inductive miner-incomplete [12] was used to automatically discover a process model from the final repaired event log (Fig. 9).

It should be noted that the process model (see Fig. 8) discovered from the original event log using inductive miner-incomplete includes much parallelism and allows self-loops for almost all the activities. Upon comparing the process models in Figs. 6 and 9, it is quite evident that by using our tool to repair the event log in just 6 steps, we were able to come very close to the process model discovered manually by the authors of [11]. We were able to achieve this despite the assumption of *absence* of a complete normative model to begin with. Instead, we looked at the *problem* activities from the detection phase, and used

[3] https://data.4tu.nl/repository/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460.

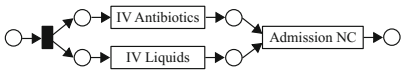

Fig. 7. *Admission NC* should be optional (see Fig. 6), but according to the available domain knowledge it is only known that if *Admission NC* occurs, it should always occur (only once) after *IV Antibiotics* and *IV Liquid*.

Table 4. Activities repaired on Sepsis log.

Activity repaired	Edit distance	% traces impacted
ER Triage	15	0.9
ER Sepsis Triage	46	2.3
IV Liquid	124	6.2
IV Antibiotics	22	1.1
Admission NC	407	32.3
Admission IC	17	1.2

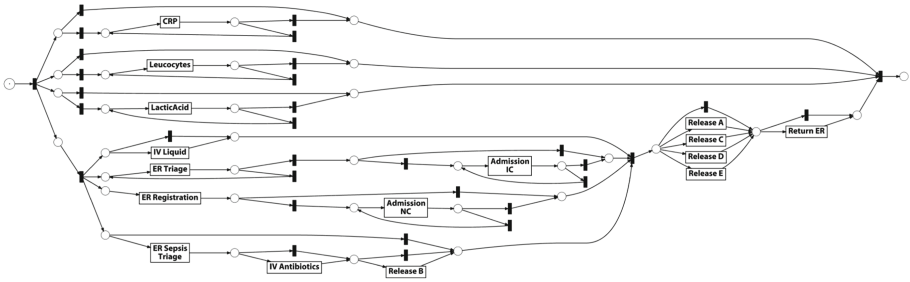

Fig. 8. Sepsis model discovered from the original event log using the Inductive miner-incomplete discovery algorithm.

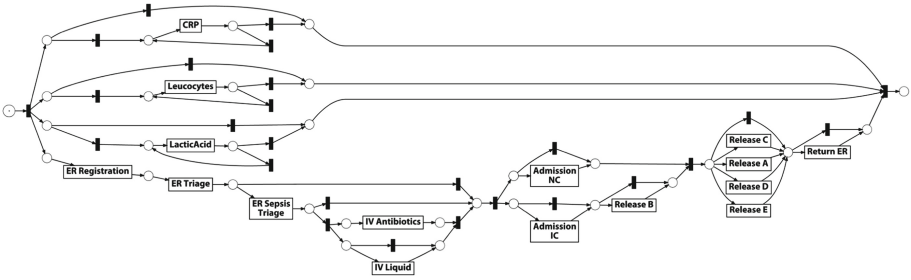

Fig. 9. Sepsis model discovered from the repaired event log using the Inductive miner-incomplete discovery algorithm.

the clinical protocols pertaining to such activities as-is, specified by Petri net fragments, in order to repair the event log.

5.2 BPIC 2015

We also evaluated our approach against the BPI Challenge 2015[4] event logs which deal with building permit applications of five municipalities in the Netherlands. We use the event log from one of the municipalities (municipality 1) which

[4] http://www.win.tue.nl/bpi/2015/challenge.

contains almost 400 activities. To make the results comprehensible, we filter the log to select the top-25 frequent activities belonging to the main application process and focus on almost 700 *closed* cases.

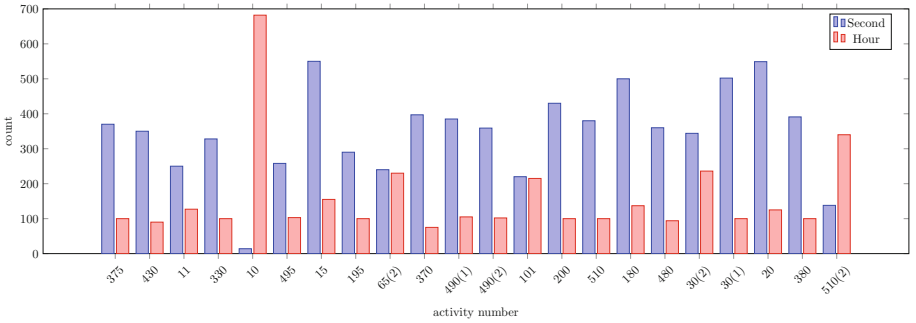

Fig. 10. Number of events per activity with hourly vs. second-granularity.

Our tool's detection technique indicated a likely granularity problem (50% of all the issues) It is clear that granularity problems can influence ordering, and hence there were a high number of ordering related issues too (47%), compared to 3% of statistical anomaly issues.

Among granularity issues, more than 10 activities had multi-granular timestamp values (see Fig. 10) and many activities had high number of events registered at *hourly* granularity resulting in the loss of ordering information in the event log. Therefore, we repair the event log in order to manually correct the ordering of top 10 activities with high number of hourly granularity. The activities in the filtered event log consists of the following naming structure: 01_HOOFD_### (a few activities have an additional _## at the end). From the available domain knowledge, it is known that typically the last three digits from 01_HOOFD_### denote the ordering sequence of activities (however this may not always be the case). We use this information to *repair* the positioning of activities which have a high number of events with hourly granularity, in relation to the activities which have high number of events with seconds granularity. Upon doing this repeatedly, we increased the granularity of almost 45% of the events which initially had hourly granularity. Because we do not have *ground truth* for the BPIC 2015 log, we use the process mining quality dimensions of fitness (i.e., the faithfulness of the

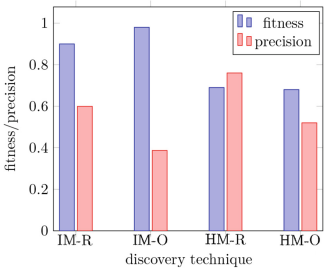

Fig. 11. Fitness and precision (original log). IM/HM: Inductive miner infrequent/Heuristic miner algorithms. Leading -R/-O denote repaired and original logs used for discovery.

model to the log) and precision (i.e., the extent to which behaviours not seen in the log are allowed by the model) to evaluate the results. Generally, a process model with higher fitness and precision is desirable [1]. We discover process models using two state of the art process discovery algorithms (Heuristic miner [13] and Inductive miner [12]) using the original (non-repaired) event log at default settings. Next, we use the same algorithms to discover process models using the repaired event log. All the four discovered process models are evaluated against the original (non-repaired) event log to assess using the fitness [10] and precision [14] dimensions (see Fig. 11). From the detection phase it was clear that there were granularity issues (and thus event ordering issues) in the event log, by repairing the event log using the domain knowledge available, we significantly improved the precision of the models thereby restricting the model from allowing too much unobserved behavior, with little or no impact on the fitness of the model with the event log.

6 Related Work

In this section we consider notions of data quality in general, data quality for event logs (timestamps in particular) and discuss work related to detection and repair of event order-related data quality issues. Data quality has a long history of active research (see [15–17]). Data quality is generally described as being a multi-dimensional concept with dimensions such as accuracy/correctness, completeness, understandability, currentness and precision [18,19] being frequently mentioned. Data quality for event logs, its impact on process mining and the particular importance of timestamps for event data was first considered in [2]. In line with general data quality discussion, consideration of event log quality also uses multi-dimensional frameworks and adopts similar dimensions. For instance, Mans et al. [6] describe event log quality as a two-dimensional spectrum with the first dimension concerned with the level of abstraction of the events and the second concerned with the accuracy of the timestamp (in terms of its (i) *granularity*, (ii) *directness of registration* (i.e., the currency of the timestamp recording) and (iii) *correctness*) thus making *explicit* the importance of these quality dimensions for proper temporal ordering of events.

We note that many process mining techniques *implicitly* detect quality issues in the event log. For example, conformance checking technique by [10] matches expectation (modeled by a Petri net) with reality (as per event logs) to detect conforming and deviating behavior. Thus the order related data quality issues would surface as deviating behavior. However such techniques *require* a process model to do the analysis. Our approach does not require ground truth during the detection phase. Many process discovery algorithms (for e.g. [12,13,20]) implicitly try to detect noise in the event log, which can sometimes be attributed to event ordering imperfections. However, the decisions and detection's made during the discovery phase are implicitly incorporated in the discovered process model, but not explicitly presented to the user. Our approach explicitly presents the user with the individual anomalies related to event ordering imperfections.

Some techniques from the literature automatically quantify the quality of an event log. [21] is a package in R which provides aggregated information about the structuredness and behavioralness of an event log. Similarly [22] discusses various metrics to quantify the overall quality of the event log. However, these techniques typically provide a global measure of data quality and do not pin point the exact list of issues within the event log. Our approach focuses on providing the user with a list of time related quality issues in the event log.

Unlike process discovery and performance analysis, repair of data quality issues in event logs is almost unexplored territory in the field of process mining. The authors of [23,24] describe techniques to correct the positioning of events based on (timed) Petri nets and probabilities derived from alignments. In [25] the authors describe a Petri net decomposition based heuristic repair strategy for efficient event log repair. In reality however, a de-facto standard process model is seldom available. Compared to these approaches which assume presence of a process model, our approach does not require an end-to-end process model for event log repair. Our approach only requires sufficient domain knowledge to recognise the sources of timestamps problems from the detected timestamp quality indicators and to then construct appropriate process fragments.

In [26,27], the authors describe a denial constraints based approach and a temporal constraints based approach resp. to automatically repair the timestamps of events in the event log. Again, these approaches require the users to pre-specify the constraints, whereas in our case the user can first analyze the detected issues, and then choose to act upon them. In [7] a visual guided approach for repairing time related issues in event log is discussed. Contrary to both constraints based approaches and visualizations driven approaches to event log repair, our approach allows the user to flexibly specify the domain knowledge using graphical process fragments, which typically are more intuitive and allow specification of complex process behavior such as concurrency, loops, choices, duplication of activities etc. In [28] a technique is proposed to automatically remove the "noisy" behavior from the event log without any user involvement. However, in our case we present such probable issues to the user, instead of automatically removing them from the event log. The user can temporarily repair the issues, analyze the impact and optionally make the changes permanent.

7 Conclusion

We have presented an integrated technique for detecting and repairing event ordering imperfections in an event log. The indicators of event ordering imperfections were discussed followed by a comprehensive strategy to detect such indicators. The proposed repair approach allows the user to develop a global truth (designed as a process fragment), which can be enforced locally on each case in the event log. This makes our repair approach far easier compared to a case by case repair strategy. Furthermore, intuitive graphical process fragments can be interactively designed by users for easy incorporation of domain knowledge. We applied our detection and repair techniques on two real life event logs.

We were able to show that we can detect anomalies and repaired them, leading to better process models being discovered. This work mostly focused on the data quality issues from a control flow perspective of process mining. In the future, we would like to address the performance and compliance perspective, along with exploring other data quality issues in process mining.

Acknowledgment. The contributions to this paper of R. Andrews, M.T. Wynn and A.H.M. ter Hofstede were supported through ARC Discovery Grant DP150103356.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, 2nd edn. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
2. van der Aalst, W., Adriansyah, A., de Medeiros, A.K.A., Arcieri, F., Baier, T., Blickle, T., Bose, J.C., van den Brand, P., 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
3. Gschwandtner, T., Gärtner, J., Aigner, W., Miksch, S.: A taxonomy of dirty time-oriented data. In: Quirchmayr, G., Basl, J., You, I., Xu, L., Weippl, E. (eds.) CD-ARES 2012. LNCS, vol. 7465, pp. 58–72. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32498-7_5
4. Bose, J.C., Mans, R.S., van der Aalst, W.M.P.: Wanna improve process mining results? IEEE CIDM **2013**, 127–134 (2013)
5. Suriadi, S., Andrews, R., ter Hofstede, A.H.M., Wynn, M.T.: Event log imperfection patterns for process mining: towards a systematic approach to cleaning event logs. Inf. Syst. **64**, 132–150 (2017)
6. Mans, R.S., van der Aalst, W.M.P., Vanwersch, R.J.B., Moleman, A.J.: Process mining in healthcare: data challenges when answering frequently posed questions. In: Lenz, R., Miksch, S., Peleg, M., Reichert, M., Riaño, D., ten Teije, A. (eds.) KR4HC/ProHealth -2012. LNCS (LNAI), vol. 7738, pp. 140–153. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36438-9_10
7. Gschwandtner, T., Aigner, W., Miksch, S., Gärtner, J., Kriglstein, S., Pohl, M., Suchy, N.: TimeCleanser: a visual analytics approach for data cleansing of time-oriented data. In: i-KNOW, p. 18. ACM (2014)
8. Mans, R.S., van der Aalst, W.M.P., Vanwersch, R.J.B.: Data quality issues. Process Mining in Healthcare. SBPM, pp. 79–88. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16071-9_6
9. Mardia, K.V., Jupp, P.E.: Directional Statistics. Wiley, Chichester (1999)
10. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Towards robust conformance checking. In: zur Muehlen, M., Su, J. (eds.) BPM 2010. LNBIP, vol. 66, pp. 122–133. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20511-8_11
11. Mannhardt, F., Blinde, D.: Analyzing the trajectories of patients with sepsis using process mining. RADAR+EMISA **1859**, 72–80 (2017)
12. 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

13. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004)
14. 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
15. Wang, R.Y., Storey, V., Firth, C.: A framework for analysis of data quality research. IEEE Trans. Knowl. and Data Eng. **7**(4), 623–640 (1995)
16. Batini, C., Palmonari, M., Viscusi, G.: Opening the closed world: a survey of information quality research in the wild. In: Floridi, L., Illari, P. (eds.) The Philosophy of Information Quality. SL, vol. 358, pp. 43–73. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07121-3_4
17. Laranjeiro, N., Soydemir, S.N., Bernardino, J.: A survey on data quality: classifying poor data. In: PRDC, pp. 179–188. IEEE (2015)
18. Wand, Y., Wang, R.Y.: Anchoring data quality dimensions in ontological foundations. Commun. ACM **39**(11), 86–95 (1996)
19. ISO/IEC FDIS 25012: Software engineering - software product quality requirements and evaluation - data quality model (2008)
20. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining – adaptive process simplification based on multi-perspective metrics. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 328–343. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75183-0_24
21. Swennen, M., Janssenswillen, G., Jans, M., Depaire, B., Vanhoof, K.: Capturing process behavior with log-based process metrics. In: SIMPDA (2015)
22. Kherbouche, M.O., Laga, N., Masse, P.A.: Towards a better assessment of event logs quality. In: 2016 IEEE SSCI, pp. 1–8, December 2016
23. Rogge-Solti, A., Mans, R.S., van der Aalst, W.M.P., Weske, M.: Improving documentation by repairing event logs. In: Grabis, J., Kirikova, M., Zdravkovic, J., Stirna, J. (eds.) PoEM 2013. LNBIP, vol. 165, pp. 129–144. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41641-5_10
24. Rogge-Solti, A., Mans, R.S., van der Aalst, W.M.P., Weske, M.: Repairing event logs using timed process models. In: Demey, Y.T., Panetto, H. (eds.) OTM 2013. LNCS, vol. 8186, pp. 705–708. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41033-8_89
25. Song, W., Xia, X., Jacobsen, H.A., Zhang, P., Hu, H.: Heuristic recovery of missing events in process logs. In: ICWS, pp. 105–112, June 2015
26. Chu, X., Ilyas, I.F., Papotti, P.: Holistic data cleaning: putting violations into context. In: IEEE ICDE, pp. 458–469, April 2013
27. Song, S., Cao, Y., Wang, J.: Cleaning timestamps with temporal constraints. Proc. VLDB Endow. **9**(10), 708–719 (2016)
28. Conforti, R., Rosa, M.L., ter Hofstede, A.H.M.: Filtering out infrequent behavior from business process event logs. IEEE Trans. Knowl. Data Eng. **29**(2), 300–314 (2017)