Why Am I Waiting? Data-Driven Analysis of Waiting Times in Business Processes

Katsiaryna Lashkevich, Fredrik Milani, David Chapela-Campa, Ihar Suvorau, and Marlon Dumas(✉)

University of Tartu, Tartu, Estonia
{katsiaryna.lashkevich,fredrik.milani,david.chapela-campa,
ihar.suvorau,marlon.dumas}@ut.ee

Abstract. Waiting times in a business process often arise when a case transitions from one activity to another. Accordingly, analyzing the causes of waiting times of activity transitions can help analysts to identify opportunities for reducing the cycle time of a process. This paper proposes a process mining approach to decompose the waiting time observed in each activity transition into multiple direct causes and to analyze the impact of each identified cause on the cycle time efficiency of the process. An empirical evaluation shows that the proposed approach is able to discover different direct causes of waiting times. The applicability of the proposed approach is demonstrated in a real-life process.

Keywords: Process mining · Waiting time · Cycle time efficiency

1 Introduction

Waiting time is a common source of waste in business processes [6]. Although it is impractical to completely eliminate waiting times in business processes, there are various approaches to reduce them [11]. Waiting times typically arise during transitions between activities, i.e. when the execution of a case moves from one activity to another. There are different reasons why waiting times occur during activity transitions. For instance, when two consecutive activities in a case are executed by different resources (a.k.a. a handoff [22]), the processing of the case is put on hold until the next resource becomes available to execute it. In this scenario, the cause of the waiting time is resource contention, i.e. a resource is not available to execute an activity instance because they are busy executing other activity instances [7]. Waiting times may also be caused by data exchanges [17], coordination issues [18], or synchronization points [18].

Process mining techniques allow us to analyze data generated by business process executions, a.k.a. *event logs*, to unveil performance and conformance issues, and associated improvement opportunities [1]. In particular, process mining techniques support the discovery of sources of waste, including waiting times [1]. However, while existing process mining techniques enable analysts to visualize

Work funded by the European Research Council (PIX Project).

M. Indulska et al. (Eds.): CAiSE 2023, LNCS 13901, pp. 174–190, 2023.
https://doi.org/10.1007/978-3-031-34560-9_11

activity transitions with high waiting time (i.e. bottlenecks), they provide limited support for identifying the causes of waiting times and how to reduce them.

To tackle this gap, this paper addresses the following research questions: (RQ1) *"What causes of waiting times between pairs of activity instances can be identified from event logs?"*, (RQ2) *"How can these causes of waiting time be identified from event logs?"*, and (RQ3) *"How can improvement opportunities, expressed as inefficiencies due to waiting times, be identified from event logs?"*

The contribution of the paper is two-fold. First, we conceptualize the causes of waiting time arising from activity transitions. Second, we propose a process mining approach to (1) discover waiting times associated with activity transitions; (2) identify their causes; and (3) analyze their impact w.r.t. a well-known measure of temporal efficiency called Cycle Time Efficiency (CTE): the ratio of processing time to cycle time in a process [7].

The proposed approach has been implemented as a software tool and empirically evaluated to verify its ability to discover different causes of waiting time using synthetic event logs. In addition, the applicability of the approach has been tested by applying it to a real-life event log.

The rest of the paper is structured as follows. Section 2 introduces background and related work. Section 3 presents the proposed approach. Section 4 describes the empirical evaluation, and Sect. 5 concludes the paper.

2 Background and Related Work

In this section, we introduce relevant conceptual foundations and notations from the fields of business process management and process mining, and position our contribution w.r.t. existing approaches to discover and analyze waiting times.

2.1 Business Processes and Temporal Performance Measures

A business process is a collection of events, activities, and decisions that lead from a customer need to an outcome that is of value to this customer [7]. Each execution of a business process is called a *case*. A common measure of temporal performance of a business process is its *cycle time*: the time between the moment a case of the process starts, and the case ends, aggregated to the level of the set of cases of a process observed during a period of time. The cycle time of a process consists of *processing time* (the time during which a case is being processed) and *waiting time* (the time when a case waits to be processed) [7].

Waiting time may be caused by resource contention, i.e. no suitable resource is available to execute an activity instance [11]. Other causes of waiting time include: synchronization between resources within a process [18] or across multiple processes [7], coordination between resources executing different activities [18], data transfer [7], batching [12], handoffs [18], and external inputs [2].

The temporal efficiency of a process can be measured by its CTE: the ratio of processing time to cycle time. When CTE is close to 1, there is relatively little waiting time. Conversely, if the CTE is close to 0, the waiting times are longer relative to processing times and there are opportunities to improve the CTE by reducing waiting times [7].

2.2 Event Logs and Activity Instance Logs

Modern IT systems record and store process execution data in *event logs*, i.e. sets of timestamped events capturing the execution of the activities in a process [7]. An event log contains information about state changes of each activity instance (e.g. enablement, start, end, or cancellations of activity instances). In this paper, we use the concept of *activity instance log* to represent the execution of a set of cases in a process. An activity instance log is an event log in which each entry contains information about the start time and end time of an activity instance [16]. Below, we introduce several notations used in the paper, leading to a definition of an activity instance log.

We consider a business process that involves a set of *activities* A. We denote each of these activities with α. An *activity instance* $\varepsilon = (\varphi, \alpha, \tau_e, \tau_s, \tau_c, \rho)$ denotes one execution of activity α, where φ identifies the *process case* to which this execution belongs to, τ_e, τ_s, and τ_c denote, respectively, the instants in time in which this activity instance was *enabled*, *started*, and *completed*, and ρ identifies the *resource* that processed the activity. Accordingly, we use $\varphi(\varepsilon_i)$, $\alpha(\varepsilon_i)$, $\tau_e(\varepsilon_i)$, $\tau_s(\varepsilon_i)$, $\tau_c(\varepsilon_i)$ and $\rho(\varepsilon_i)$ to denote, respectively, the process case, the activity, the enablement time, the start time, the completion time, and the resource associated with the activity instance ε_i. We use (τ_i, τ_j) to denote the time interval between τ_i and τ_j. We write $\omega(\varepsilon_i) = (\tau_e(\varepsilon_i), \tau_s(\varepsilon_i))$ to denote the *waiting time* of ε_i, representing the interval since ε_i became available for processing $(\tau_e(\varepsilon_i))$, until its recorded start $(\tau_s(\varepsilon_i))$. The processing time of ε_i is $pt(\varepsilon_i) = (\tau_s(\varepsilon_i), \tau_c(\varepsilon_i))$. We use $(\tau_i, \tau_j) \in (\tau_k, \tau_l)$ to denote that the interval (τ_i, τ_j) is contained in (τ_k, τ_l), i.e. $\tau_i \geq \tau_k$ and $\tau_j \leq \tau_l$. With $(\tau_i, \tau_j) \perp (\tau_k, \tau_l)$ we denote that both intervals (partially or fully) overlap, i.e. $\exists (\tau_m, \tau_n) \in (\tau_i, \tau_j) \mid (\tau_m, \tau_n) \in (\tau_k, \tau_l)$.

Given the above, an *activity instance log* L is a collection of activity instances recording the data of the execution of a set of cases of a business process. Table 1 shows an example of 10 activity instances from an activity instance log, whereas Fig. 1 depicts the corresponding process model.

Table 1. Fragment of an activity instance log composed of 10 activity instances.

Case ID	Activity	Enabled Time	Start Time	End Time	Resource
...					
510	Register invoice	08:30:12	08:30:12	11:30:00	Jack
511	Notify acceptance	09:10:11	09:10:11	10:01:01	Carolyn
511	Post invoice	09:10:11	09:10:11	10:14:15	Sarah
512	Post invoice	10:25:45	10:25:45	10:30:00	Sarah
513	Post invoice	10:34:15	10:34:15	12:00:00	Sarah
514	Post invoice	09:10:11	11:30:00	13:00:00	Jack
515	Register invoice	12:08:10	12:08:10	13:00:00	Sarah
512	Pay invoice	10:30:00	15:55:50	17:00:11	Jack
513	Pay invoice	12:00:00	17:00:11	17:55:40	Jack
511	Pay invoice	10:14:15	17:55:40	18:30:15	Jack
...					

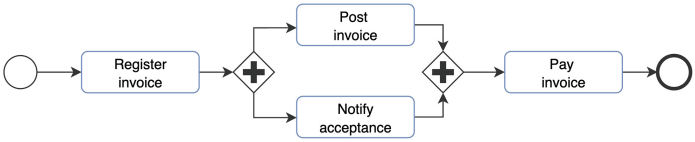

Fig. 1. Process model example corresponding to the event log of Table 1.

2.3 Related Work

Process mining is a family of techniques that support the discovery of process models from data as well as the analysis of business processes with respect to efficiency, quality, and compliance. Some of these techniques address the question of how to discover and analyze waiting times in specific application domains. For instance, Uysal et al. [21] present a case study where process mining is used to identify bottlenecks and reduce the cycle time in a production process. Similarly, Erdogan et al. [8] apply process mining techniques to identify waiting times in a hospital emergency process, while Yampaka & Chongstitvatana [24] describe an application of process mining combined with a queuing system to analyze and improve temporal performance in a healthcare process. Similarly, Antunes et al. [3] combine process mining with discrete event simulation to optimize waiting time in an emergency department. However, none of these domain-specific studies considers the question of how to attribute waiting times to their causes, which is the focus of this paper.

Ferreira & Vasilyev [9] present a technique to identify why some cases in a process take longer time to complete. They identify case characteristics correlating with higher delays, e.g., when a given activity occurs in a case, or when a given resource is involved, the case is likely to have higher waiting time. Likewise, De Leoni & van der Aalst [5] combine some of the existing correlation analysis techniques to identify how different process characteristics correlate with the process performance, e.g. if process deviations cause delays. Similarly, Hompes et al. [10] propose an approach based on time series analysis to detect cause-effect relations between process characteristics and performance indicators, e.g., if the waiting time for the receipt of payment depends on the time of day. Toosinezhad et al. [20] introduce an approach to detect event patterns that frequently precede, i.e. lead to, dynamic bottlenecks. While these studies take a correlation-based approach to analyze waiting times, we classify the causes of waiting times and consider their impact on process performance.

Some process mining techniques support the identification of waiting times caused by queuing effects, i.e. when activity instances wait in a queue until a resource becomes available [19]. Similarly, in [12], the authors present an approach to discover waiting times caused by batch processing. However, these techniques focus on identifying a singular cause of waiting time. In contrast, in the present paper, we seek to decompose the observed waiting time into multiple causes.

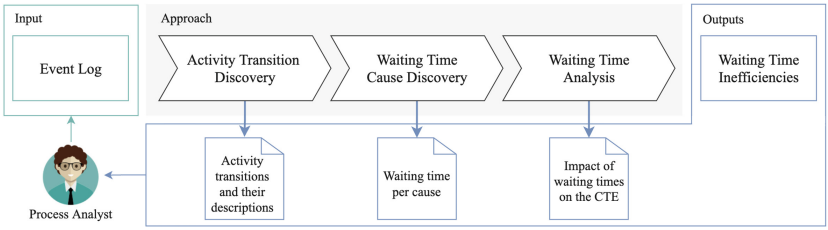

Fig. 2. Overview of the proposed approach.

3 Waiting Time Discovery and Analysis

In this section, we describe our approach to discover and analyze waiting times in a business process. The approach takes an event log as input and, as a result, produces a report comprising the causes of waiting times and their impact on the CTE. Figure 2 depicts an overview of the three main steps of the approach.

In the first step, we discover the transitions between activities and their characteristics – total frequency, case frequency, and total waiting time – from the event log. In the second step, we identify the causes of waiting time of each transition. In the third and final step, we analyze the impact of each cause on the temporal efficiency of the process.

3.1 Activity Transition Discovery

The first step of our approach is to discover transitions between activities and their waiting times. We define an *activity transition instance* as a pair of activity instances $\langle a_1, b_1 \rangle$ in a single case, such that the completion of a_1 enables b_1, i.e. b_1 cannot be executed before a_1 is completed. We call the first element of an activity transition instance the *source activity instance*, while the latter is the *target activity instance*. An *activity transition* is a set of activity transition instances with the same source and target activities, where the *source activity* is the activity executed in all its source activity instances, and the *target activity* is the activity executed in all its target activity instances. For example, the activity transition $\langle a, b \rangle$ is composed of the set of activity transition instances $\{\langle a_1, b_1 \rangle, \langle a_2, b_2 \rangle, \ldots \langle a_n, b_n \rangle\}$. For simplicity, we refer to activity transitions as *transitions* and to activity transition instances as *transition instances*.

As input, we require an activity instance log as defined in Sec. 2.2, where the resources sharing the same role are considered separately, and the enabled time of each activity instance is optional. If this latter element is missing, we estimate it as follows. In a sequential process, each activity instance of a case is enabled by the completion of the preceding activity instance. However, concurrency is common in real-life processes. For example, Fig. 3 shows the execution of a case with concurrency between two activities. The order of the activity instances is "Register invoice", "Post invoice", "Notify acceptance", and "Pay invoice". However, "Post invoice" and "Notify acceptance" are enabled when the activity

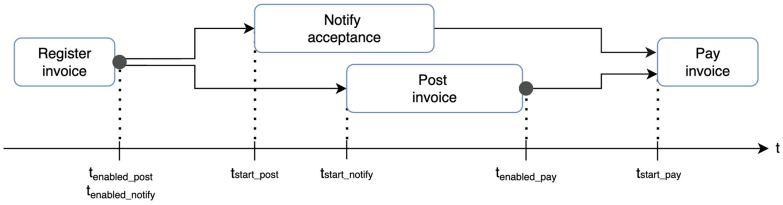

Fig. 3. Waiting time in a case with concurrent activity instances.

instance "Register invoice" completes. In the same way, "Pay invoice" is enabled only when "Post invoice" is completed. Accordingly, there are only three activity transitions in this example, namely $\langle RegisterInvoice, NotifyAcceptance\rangle$, $\langle RegisterInvoice, PostInvoice\rangle$, and $\langle PostInvoice, PayInvoice\rangle$. In this way, we consider each activity instance to be enabled by its closest non-concurrent preceding activity instance. We assume that once an activity instance is enabled, it remains enabled until its processing starts.

To detect concurrent activities (e.g. if activities "Post invoice" and "Notify acceptance" are concurrent), we use the concurrency oracle of the Heuristics Miner [23]. This method computes, for each pair of activities related via a directly-follows relation, a coefficient indicating whether these activities are in a concurrent relation or not. This coefficient is computed based on the percentage of times the two activities directly-follow each other in either order, e.g. "Post invoice" followed by "Notify acceptance" versus "Notify acceptance" followed by "Post invoice". The concurrency oracle then determines which pairs of activities are concurrent based on certain thresholds.

Once the transition instances are discovered, we calculate their *duration*, i.e. the waiting times they induce. The waiting time of a target activity instance in a transition instance is the interval between its enablement and its start time. For example, in Fig. 3, the waiting time in the transition $\langle PostInvoice, PayInvoice\rangle$ corresponds to the interval $(t_{enabled\_pay}, t_{start\_pay})$. In this way, we identify the waiting time of each target activity instance per transition.

Finally, we compute the following characteristics for each identified transition (composed of all its transition instances): *Case frequency* illustrates the proportion of process cases from the total number of cases where this transition is observed. *Total frequency* indicates the number of occurrences of this transition in the process. *Total duration* is the sum of the waiting times of all transition instances. The output of this step is a report depicting all identified transitions and their characteristics, sorted by total duration in descending order. Based on this information, the analysts can see what transitions cause the highest waiting times and how frequently they are executed.

3.2 Waiting Time Cause Discovery

Once the activity transitions and their characteristics are discovered, we analyze the waiting time of each transition instance and identify their causes. In this

section, we define the proposed causes of waiting times (RQ1) and describe how they can be identified from an event log (RQ2).

Given the event log information, we consider that an enabled activity instance can wait for *i)* other activity instances to be enabled (so they are processed together) or *ii)* the assigned resource to become available. Below, we analyze each of these situations and we relate them to direct causes of waiting time. *i)* When an activity instance waits for another activity instance to be enabled, we observe a batch processing behavior, and thus, *waiting time due to batching*. *ii)* If an activity instance is not waiting for this reason, it might wait for the assigned resource. The assigned resource might be busy processing other activity instances, enabled before or after the waiting activity instance. Thus, we observe *waiting times due to resource contention* or *due to prioritization*, respectively. If the resource is not busy, they might be unavailable due to working schedules, causing *waiting time due to resource unavailability*. Finally, if there is waiting time that cannot be explained by any of the above causes, we consider the cause to be *due to extraneous factors*, i.e. causes that cannot be identified from the log. Accordingly, we propose to target five causes of waiting time: batching, resource contention, prioritization, resource unavailability, and extraneous factors.

The waiting time within a given transition instance may stem from one or multiple causes –e.g. the resource was busy performing another activity instance during half of this waiting time, while the resource was off-duty (outside their working hours) during the other half. If there are multiple waiting time causes for a given transition instance, we decompose this waiting time into non-overlapping time intervals and attribute each interval to one cause using the decision procedure in Fig. 4. According to this decision procedure, we first identify if any intervals of the transition duration are caused by batching. Then we look for intervals of waiting time caused by resource contention and prioritization, followed by resource unavailability and extraneous factors. This order is determined by the dominance relations between these causes. Batching dominates resource contention, prioritization, and unavailability, because regardless of the availabil-

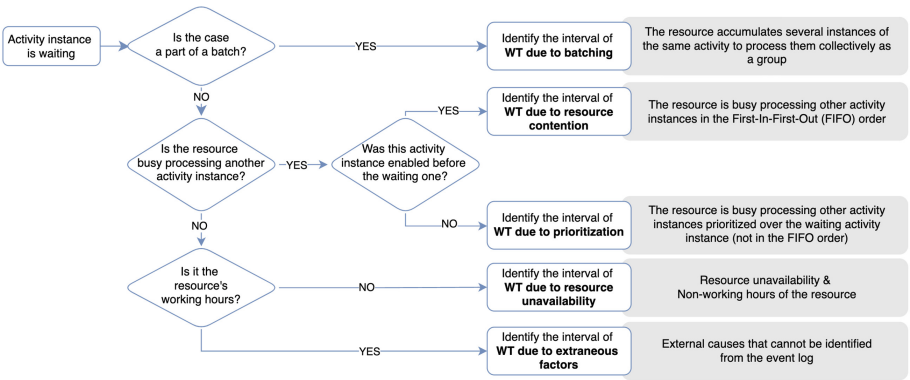

Fig. 4. Overview of the waiting time cause discovery process and their definitions.

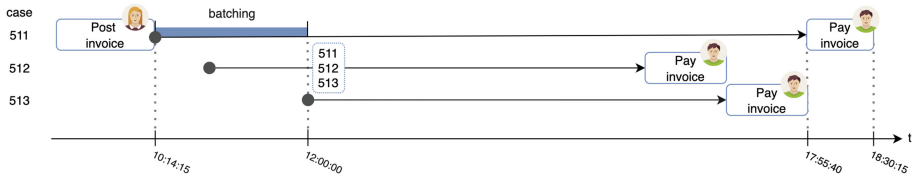

Fig. 5. Waiting time due to batching.

ity status of any given resource, an activity instance that is part of a batch is not ready to be assigned (and started) until the batch is ready. Resource contention and prioritization dominate resource availability, because if a resource has a work queue, they cannot start an activity instance until the latter reaches the front of the queue, or until this activity instance has the highest priority among all activity instances in the queue, regardless of the resource's availability status. Extraneous factors are dominated by all other causes, as they act as a "catch-all" cause for any waiting time that cannot be attributed to other causes.

Note that since each interval of waiting time is attributed to a single cause, this approach ensures the identified waiting time are additive, i.e. the sum of the waiting time causes is equal to the total waiting time of the transition instance.

Below, we define each waiting time cause in turn, and we specify how it is discovered within an activity instance log.

Waiting Time Due to Batching. The first cause that we identify is *waiting time due to batching*. Batch processing occurs when a set of instances of the same activity are accumulated to be processed together (either simultaneously or one after the other) [12]. In this context, a batch is a set of activity instances $\mathcal{E}_b \subseteq L$, such that all $\varepsilon_i \in \mathcal{E}_b$ record the execution of the same activity, performed by the same resource, and processed as a batch (i.e. all of them were enabled before any of them started, and they were processed as a group). We use the technique proposed in [12] to identify batch processing. In batch processing, when an activity instance is enabled and ready to be processed, it can wait for other instances of the same activity until the batch is accumulated, i.e. until all instances that are part of a batch are collected. Accordingly, when the target activity instance of a transition instance is detected as part of a batch, the waiting time interval from its enablement time to the batch accumulation time is classified as *waiting time due to batching*. The waiting time due to batching of an activity instance is then defined as follows:

Definition 1 (Waiting Time Due to Batching). Given an activity instance log L, a batch $\mathcal{E}_b \subseteq L$, and an activity instance $\varepsilon_i \in \mathcal{E}_b$, the waiting time due to batching of ε_i is $\omega_{ba}(\varepsilon_i) = (\tau_e(\varepsilon_i), \tau_{bc}) \mid \tau_{bc} = max(\{\tau_e(\varepsilon_j) \mid \varepsilon_j \in \mathcal{E}_b\})$, i.e. the interval of time between the enablement of ε_i and the last enablement of the activities in the batch.

Consider the transition instance between "Post invoice" and "Pay invoice" of case `511` in the running example (Fig. 5) with a waiting time between 10:14:15

and 17:55:40. The activity "Pay invoice" is a batch-processed activity where the resource accumulates instances and then processes them one by one (sequential batch processing) [16]. The batch is accumulated until the last activity instance of "Pay invoice" is enabled, i.e. case 513 is ready to be processed ($t_{enabled} = 12{:}00{:}00$). Therefore, if we analyze the transition instance $\langle PostInvoice, PayInvoice \rangle$ of case `511`, its waiting time due to batching corresponds to the interval between 10:14:15 and 12:00:00.

Waiting Time Due to Resource Contention. There are situations where the resources that have to process a certain activity are busy processing other activity instances that were enabled earlier than the waiting one, and thus, it's understood that they start processing them before the current one (following a first-in-first-out order).[1] When this situation occurs, we classify as *waiting time due to resource contention* those intervals in which the resource that performed the activity instance was working in other activity instances enabled before it. Therefore, the waiting time due to resource contention of an activity instance is defined as follows:

Definition 2 (Waiting Time Due to Resource Contention). Given an activity instance log L, and an activity instance $\varepsilon_i \in L$, the waiting time due to resource contention of ε_i is $\Omega_{rc}(\varepsilon_i) = \{(\tau_i, \tau_j) \mid \tau_i = max(\tau_e(\varepsilon_i), \tau_s(\varepsilon_j)) \wedge \tau_j = min(\tau_s(\varepsilon_i), \tau_c(\varepsilon_j)) \wedge \varepsilon_j \in L \wedge \varepsilon_j \neq \varepsilon_i \wedge \rho(\varepsilon_j) = \rho(\varepsilon_i) \wedge \tau_e(\varepsilon_j) \leq \tau_e(\varepsilon_i) \wedge pt(\varepsilon_j) \perp \omega(\varepsilon_i)\}$, i.e. the set of intervals of processing time of all ε_j of L (executed by the same resource as ε_i, and enabled before it) overlapping with the waiting time of ε_i.

Coming back to the running example, during the transition instance between "Post invoice" and "Pay invoice" of case `511`, the resource works on case `514` that has an earlier enabled instance of "Post invoice" (see Fig. 6). Therefore, there is waiting time due to resource contention between 12:00:00 and 13:00:00.

Waiting Time Due to Prioritization. However, the resources might not always follow the FIFO policy. In some situations, the resources might give priority to certain activity instances over others. We call this behavior *prioritization*, meaning that an activity instance is processed out of turn w.r.t. a FIFO policy, thus causing other activity instances to wait longer. When this situation occurs, we classify as *waiting time due to prioritization* those intervals in which the resource that performed the activity instance was working in other activity instances enabled after it. Therefore, the waiting time due to prioritization of an activity instance is defined as follows:

Definition 3 (Waiting Time Due to Prioritization). Given an activity instance log L, and an activity instance $\varepsilon_i \in L$, the waiting time due to prioritization of ε_i is $\Omega_{prior}(\varepsilon_i) = \{(\tau_i, \tau_j) \mid \tau_i = \tau_s(\varepsilon_j) \wedge \tau_j = min(\tau_s(\varepsilon_i), \tau_c(\varepsilon_j)) \wedge \varepsilon_j \in$

[1] We assume that resources work only on one activity at a time, i.e. there is no multitasking. Thus, we foresee that the proposed estimation technique will not be suitable for event logs with a high proportion of multitask activity instances.

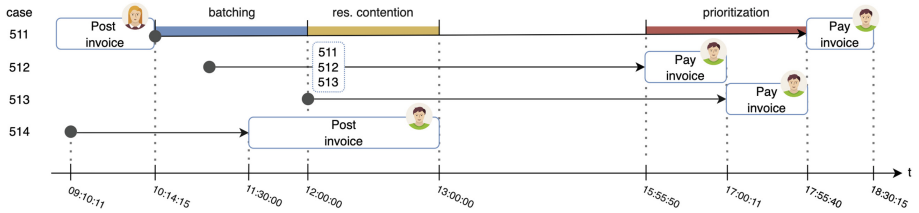

Fig. 6. Waiting time due to resource contention and due to prioritization.

$L \wedge \varepsilon_j \neq \varepsilon_i \wedge \rho(\varepsilon_j) = \rho(\varepsilon_i) \wedge \tau_e(\varepsilon_j) > \tau_e(\varepsilon_i) \wedge pt(\varepsilon_j) \perp \omega(\varepsilon_i)\}$, i.e. the set of intervals of processing time of all ε_j of L (executed by the same resource as ε_i, and enabled after it) overlapping with the waiting time of ε_i.

In Fig. 6, when the resource starts processing the batch of cases `511-513`, instead of processing cases in their order of enablement, he prioritizes cases `512` and `513` over `511`. Waiting time due to prioritization then corresponds to the interval between 15:55:50 and 17:55:40.

Waiting Time Due to Resource Unavailability. The fourth cause of waiting time that we propose to consider is resource unavailability, which corresponds to the intervals in time in which the resource is not available to work due to their working schedules. To identify this waiting time, we need to first discover the working schedules of the resources. We propose to use the technique presented in [14] to discover calendars over time granules not fully described by the input data. This technique analyzes the instants in time when each resource interacted with the system (i.e. the start and end of each activity instance) to build a weekly working calendar composed of time intervals in which there was enough evidence, based on given support and confidence values, that the resource is working.[2] Given the weekly calendars of each resource, we transform them to absolute time intervals to compare them with the waiting times observed in the log. Then, we classify as *waiting time due to resource unavailability* those intervals where the resource is not available for working. Therefore, the waiting time due to resource unavailability of an activity instance is defined as follows:

Definition 4 (Waiting Time Due to Resource Unavailability). Given an activity instance log L, an activity instance $\varepsilon_i \in L$, and being $cal_{av}(\rho) = \{(\tau_{avs}, \tau_{avc})\}$ a resource availability calendar with the set of time intervals in which the resource is available to work, the waiting time due to resource unavailability of ε_i is $\Omega_{unav}(\varepsilon_i) = \{(\tau_i, \tau_j) \mid (\tau_i, \tau_j) \in \omega(\varepsilon_i) \wedge \nexists(\tau_k, \tau_l) \in cal_{av}(\rho(\varepsilon_i)) \mid (\tau_i, \tau_j) \perp (\tau_k, \tau_l)\}$, i.e. the set of intervals of the waiting time of ε_i that do not overlap with the availability calendar of the resource $\rho(\varepsilon_i)$ that executed ε_i.

[2] Although we use this resource calendar discovery algorithm, the approach can be applied with other calendar discovery algorithms or with manually defined calendars.

In the running example (Fig. 7), the resource executing "Pay invoice" does not work between 13:00:00 and 15:00:00. Thus, in the transition instance of case `511`, the respective waiting time interval is due to resource unavailability.

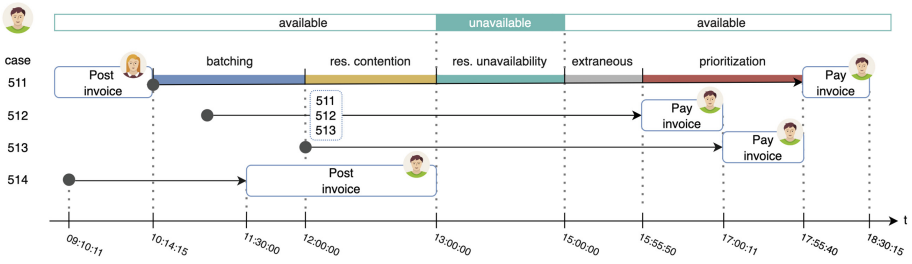

Fig. 7. Waiting time due to resource unavailability and due to extraneous factors.

Waiting Time Due to Extraneous Factors. The last cause of waiting time that we propose to consider is "extraneous factors". We propose to classify as *waiting time due to extraneous factors* the waiting time intervals caused by the external effects that cannot be identified from the event log – e.g. the resource is working on another process, the activity instance cannot start until some unrecorded event has occurred, fatigue effects, or context switch In Fig. 7, the interval between 15:00:00 and 15:55:50 cannot be explained by the data available in the log and is considered due to extraneous factors. Thus, the waiting time of an activity instance due to extraneous factors is defined as follows:

Definition 5 (Waiting Time Due to Extraneous Factors). Given an activity instance log L and an activity instance $\varepsilon_i \in L$, the waiting time due to extraneous factors of ε_i is $\Omega_{extr}(\varepsilon_i) = \{(\tau_i, \tau_j) \mid (\tau_i, \tau_j) \in \omega(\varepsilon_i) \wedge \nexists(\tau_k, \tau_l) \in (\{\omega_{ba}(\varepsilon_i)\} \bigcup \Omega_{rc}(\varepsilon_i) \bigcup \Omega_{prior}(\varepsilon_i) \bigcup \Omega_{unav}(\varepsilon_i)) \mid (\tau_i, \tau_j) \perp (\tau_k, \tau_l)\}$, i.e. the set of intervals within the waiting time of ε_i that does not overlap with waiting times due to batching, resource contention, prioritization, or resource unavailability.

3.3 Waiting Time Analysis

The final step is to analyze how much each waiting time cause contributes to the temporal performance of the process and their impact on the CTE. With that purpose, we propose to compute the percentage of time that each cause induces in the CTE of the process. In this context, we measure the CTE as the processing time divided by the sum of the processing and the waiting time (PT + WT), where PT is the sum of the processing time of all activity instances in the process, and WT is the sum of their waiting time. The impact of waiting times per cause is calculated as the difference between the original process CTE and the CTE if a particular waiting time is eliminated. In this way, we can measure

(1) the impact that each waiting time cause has on the process CTE, (2) the impact that each transition has on the process CTE, and (3) the impact that each waiting time cause has in each transition. These metrics can indicate the potential CTE improvement if a particular cause of waiting time is addressed.

As a result, we can analyze the discovered waiting time causes and their impact on CTE. With this information, process analysts can identify where the inefficiencies due to waiting times are localized, choose which transitions and/or waiting time causes to address, and which redesign alternatives to apply.

4 Evaluation

In this section, we present the evaluation of our approach. We evaluate the approach by addressing two evaluation questions: (EQ1) *To which extent is the technique able to detect the presence or absence of certain waiting time causes?*, (EQ2) *To what extent is the technique able to correctly quantify the amount of waiting time waste per each cause?* In this experimentation, we use synthetic data to validate the ability of the technique to accurately discover transitions, their waiting times and the waiting time causes known to be present in the event logs. Then, we demonstrate the approach's applicability on a real-life event log. The approach implementation, the event logs and experiment results are all available on GitHub.[3]

4.1 Evaluation on Synthetic Data

To answer EQ1, we used a business process simulation model (BPS model) of a loan application process to simulate a set of event logs with different combinations of waiting time causes. To simulate waiting time due to resource contention, we set a low number of available resources in the BPS model, so that in some cases, there were no resources available to process an enabled activity instance. To create waiting time due to resource unavailability, we set resource working calendars so that some resources worked from Monday to Wednesday, and others from Thursday to Friday. To simulate waiting time due to extraneous factors, we added timer events before some of the activities in the BPS model, thus delaying their start. Waiting time due to batching and prioritization cannot be injected by modifying the simulation parameters, as current BPS engines do not support them. Therefore, in batching, we added a set of new cases delaying the start of some activity instances so that they are processed as a batch. To simulate prioritization, we added a set of new cases changing the order of execution of some activity instances, so they are processed following a prioritization order. Combining these modifications, we simulated a set of 32 event logs with all the combinations of causes of waiting time and measured the performance using precision and recall. True positives and false positives stand for the discovery of

[3] https://github.com/AutomatedProcessImprovement/waiting-time-analysis/tree/caise2023.

a waiting time cause that, respectively, was and was not injected in the event log. True and false negatives denote an undiscovered waiting time cause that, respectively, was not and was injected.

Table 2. Results for the simulated event logs with all waiting time causes, depicting the true positives with '✓', the false positives with '✗', and the true negatives with an empty cell (there are no false negatives).

	S01	S02	S03	S04	S05	S06	S07	S08	S09	S10	S11	S12	S13	S14	S15	S16
Batching		✓					✓	✓	✓	✓				✗		
Prioritization			✓				✓				✓	✓	✓	✗	✗	
Res. Contention				✓				✓			✓			✓	✓	
Res. Unavailability					✓	✗			✓	✗		✓	✗	✓	✗	✓
Extraneous factors						✓				✓			✓		✓	✓
	S17	S18	S19	S20	S21	S22	S23	S24	S25	S26	S27	S28	S29	S30	S31	S32
Batching	✓	✓	✓	✓	✓	✓	✗			✗	✓	✓	✓	✓	✗	✓
Prioritization	✓	✓	✓		✗		✓	✓	✓	✗	✓	✓	✓	✗	✓	✓
Res. Contention	✓			✓	✓		✓	✓		✓	✓	✓		✓	✓	✓
Res. Unavailability		✓	✗	✓	✗	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓
Extraneous factors			✓		✓	✓		✓	✓	✓		✓	✓	✓	✓	✓

Table 2 depicts the results for the simulated event logs used to evaluate EQ1. Our approach discovered the injected waiting time causes in all the event logs (no false negatives) resulting in a recall of 100%. However, due to the influence that some waiting time causes have between them, the results contain 17 false positives (precision of 83%). False positives in waiting time due to resource unavailability (S06, S10, S13, S15, S19, S21, S24, and S28) are caused by the presence of extraneous waiting time, combined with limited data for the discovery of resources' working calendars. To simulate these logs, we set 24/7 working calendar for all resources (high availability). However, when a resource has low occupation (executes few events), there is not enough data for the calendar discovery to identify a 24/7 calendar and some intervals are interpreted as non-working time. When these non-working intervals overlap with extraneous waiting time, the tool classifies them as waiting time due to resource unavailability. This limitation is inherent to the discovery of the resource calendar, if there is no data showing that a resource was active during a period of time, it cannot be assumed that they were working.

The injection of waiting time due to extraneous factors also induced false positives of prioritization (S15, S21, S26, S30). When an activity instance is enabled but waiting due to extraneous factors, the resource might execute other activities enabled after the waiting one, being detected as waiting time due to prioritization. These false positives are due to the absence of an explicit indicator of the extraneous factors (i.e. extraneous waiting time is only detected when no other causes are identified). False positives due to batching (S14, S23, S26, S31) are

caused by the appearance of a batch processing behavior that was not intentionally added. To create waiting time due to resource contention and unavailability, in some scenarios, we assigned working calendars with low resource availability. In such cases, while instances were waiting until the resource became available, they were collected and processed by the same resource. This resulted in an unassigned but correctly discovered batch processing behavior.

EQ2 aimed at assessing if our approach is able to correctly identify waiting time intervals and their causes. We could not use the set of event logs used for EQ1 to answer EQ2, as the logs were created through stochastic business process simulation – we know we have introduced a certain cause of waiting time, but we don't know to what extent. Therefore, we manually created a set of 5 event logs with activity transitions having different waiting time causes. Due to the low number of events per resource, we manually defined the resource working calendars for this experiment. We ran our technique over these logs and obtained accurate waiting time intervals and their causes. The input and results for the EQ2 experiment are available on GitHub.

4.2 Evaluation on a Real-Life Log

To evaluate the applicability of the proposed approach in a real-life scenario, we used an event log of a manufacturing production process [13]. The event log has 225 cases recording the execution of 24 activities in a total of 4,503 activity instances, executed by 46 resources.

First, we discovered the transition instances – 91 transitions with 3,421 transition instances (i.e. executions of each transition) –, and their characteristics – case frequency, total frequency, and total duration (i.e. total waiting time).

Then, we discovered the waiting time causes, producing a report that captures to what extent each cause (per transition) contributes to the total waiting time of the process. The highest waiting times originated from the self-loop transitions, i.e. when the same activity is executed twice in a row. For instance, the self-loop of "Turning & Milling Q.C." induced a total waiting time of 1,101 days, 2 h, and 54 min (see Fig. 8), being the greatest contributor to the total waiting time. The largest portion of the waiting time in this transition (54.74%) was caused by resource unavailability (602 days, 19 h, and 59 min).

Finally, we analyzed to what extent each waiting time cause contributes to the total waiting time of the process, and how they affect the CTE. Resource unavailability was the primary source of waiting time (as it caused 57% of the total waiting time), followed by batching (22%), prioritization (9%), extraneous (8%), and resource contention (4%). Then, we measured the impact of the waiting times by cause on the process CTE (CTE = 6.81%). The highest CTE increase (up to 14.62%) can be achieved if the waiting time due to resource unavailability is eliminated. Regarding individual transitions, addressing the "Turning & Milling Q.C" self-loop is the highest improvement opportunity. If the waiting time in this transition is addressed, the CTE could increase to 7.69%.

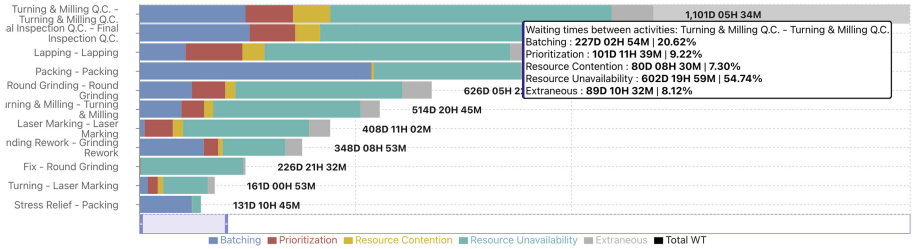

Fig. 8. Waiting time causes in activity transitions of the production process.

5 Conclusion

This paper outlines a process mining-based approach for identifying causes of waiting times and their impact. To address research question RQ1, we propose a method to attribute the waiting time of activity transitions to one of five causes: *batching*, *resource contention*, *prioritization*, *resource unavailability*, and *extraneous factors*. To address RQ2, we outline how these five causes of waiting time can be discovered, for each activity transition, from an event log. Finally, we propose to measure the impact of each waiting time cause on the CTE of a process, to help analysts to prioritize opportunities to reduce these waiting times (RQ3). The empirical evaluation shows that our approach can accurately classify the waiting times in a process into the five causes, in (synthetic) event logs where the causes of waiting time are known. Finally, we illustrated the applicability of our approach using an event log of a production process.

In its current form, the proposed approach only considers waiting times in transitions between activity instances. Yet waiting times may also arise in at least two other settings: (i) between case creation and start of the first activity instance; and (ii) within an activity instance due to interruptions (e.g. the resource interrupts their work and resumes it later). The first of these waiting times could be analyzed by applying methods that estimate the inter-arrival time of each case [4,15]. The second requires new methods for modeling and inferring interruptions, possibly using additional attributes of the log or other additional data. Another limitation of the approach is that it does not consider multitasking. This could be addressed by inferring multitasking patterns from the log, and using this data to estimate at what point in time a resource would normally have started an activity instance, given their past multitasking behavior.

In future work, we plan to develop a method for discovering business process simulation models from event logs, which considers the causes of waiting times considered in this paper. Such simulation models could be used to support analysts in identifying combinations of redesign options to optimize CTE, while also considering other performance dimensions (e.g. cost).

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. Andrews, R., Wynn, M.: Shelf time analysis in CTP insurance claims processing. In: Kang, U., Lim, E.-P., Yu, J.X., Moon, Y.-S. (eds.) PAKDD 2017. LNCS (LNAI), vol. 10526, pp. 151–162. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67274-8_14
3. Antunes, B.B.P., Manresa, A., Bastos, L.S.L., Marchesi, J.F., Hamacher, S.: A solution framework based on process mining, optimization, and discrete-event simulation to improve queue performance in an emergency department. In: Di Francescomarino, C., Dijkman, R., Zdun, U. (eds.) BPM 2019. LNBIP, vol. 362, pp. 583–594. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-37453-2_47
4. Berkenstadt, G., Gal, A., Senderovich, A., Shraga, R., Weidlich, M.: Queueing inference for process performance analysis with missing life-cycle data. In: Proceedings of the 2nd International Conference on Process Mining (ICPM 2020), pp. 57–64. IEEE (2020)
5. De Leoni, M., van der Aalst, W.M., Dees, M.: A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. Inf. Syst. **56**, 235–257 (2016)
6. Delias, P.: A positive deviance approach to eliminate wastes in business processes: the case of a public organization. Ind. Manag. Data. Syst. **117**, 1323–1339 (2017)
7. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A., et al.: Fundamentals of Business Process Management, 2nd edn. Springer, Heidelberg. (2018). https://doi.org/10.1007/978-3-662-56509-4
8. Erdogan, T.G., Tarhan, A.K.: Multi-perspective process mining for emergency process. Health Inform. J. **28**(1), 14604582221077196 (2022)
9. Ferreira, D.R., Vasilyev, E.: Using logical decision trees to discover the cause of process delays from event logs. Comput. Ind. **70**, 194–207 (2015)
10. Hompes, B.F., Maaradji, A., La Rosa, M., Dumas, M., Buijs, J.C., van der Aalst, W.M.: Discovering causal factors explaining business process performance variation. In: Dubois, E., Pohl, K. (eds.) Advanced Information Systems Engineering. CAiSE 2017. LNCS, vol. 10253, pp. 177–192. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_12
11. Jansen-Vullers, M., Reijers, H.: Business process redesign in healthcare: towards a structured approach. Information **43**(4), 321–339 (2005)
12. Lashkevich, K., Milani, F., Chapela-Campa, D., Dumas, M.: Data-Driven Analysis of Batch Processing Inefficiencies in Business Processes. In: Guizzardi, R., Ralyté, J., Franch, X. (eds.) RCIS, pp. 231–247. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-05760-1_14
13. Levy, D.: Production analysis with process mining technology (2014). https://doi.org/10.4121/uuid:68726926-5ac5-4fab-b873-ee76ea412399
14. López-Pintado, O., Dumas, M.: Business process simulation with differentiated resources: does it make a difference? In: Di Ciccio, C., Dijkman, R., del Río Ortega, A., Rinderle-Ma, S. (eds.) Business Process Management. BPM 2022. LNCS, vol. 13420, pp. 361–378. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-16103-2_24
15. Martin, N., Depaire, B., Caris, A.: Using event logs to model interarrival times in business process simulation. In: Reichert, M., Reijers, H.A. (eds.) BPM 2015. LNBIP, vol. 256, pp. 255–267. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42887-1_21

16. Martin, N., Pufahl, L., Mannhardt, F.: Detection of batch activities from event logs. Inf. Syst. **95**, 101642 (2021)
17. Ramakrishnan, S., Kumaran, S., Chang, H., Kulkarni, N., Srihari, K.: Defining and categorizing handoff points for the service domain. In: Proceedings of the 29th Annual Conference of ASEM, pp. 12–15 (2008)
18. Rummel, J.L., Walter, Z., Dewan, R., Seidmann, A.: Activity consolidation to improve responsiveness. Eur. J. Oper. Res. **161**(3), 683–703 (2005)
19. Senderovich, A., Weidlich, M., Gal, A., Mandelbaum, A.: Queue mining for delay prediction in multi-class service processes. Inf. Syst. **53**, 278–295 (2015)
20. Toosinezhad, Z., Fahland, D., Köroğlu, Ö., Van Der Aalst, W.M.: Detecting system-level behavior leading to dynamic bottlenecks. In: 2020 2nd ICPM, pp. 17–24. IEEE (2020)
21. Uysal, M.S., et al.: Process mining for production processes in the automotive industry. In: Industry Forum at BPM, vol. 20 (2020)
22. Van Der Aalst, W.M., et al.: Business process mining: an industrial application. Inf. Syst. **32**(5), 713–732 (2007)
23. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible heuristics miner (FHM). In: Proceedings of the IEEE Symposium on CIDM. IEEE (2011)
24. Yampaka, T., Chongstitvatana, P.: An application of process mining for queueing system in health service. In: 13th International JCSSE, pp. 1–6. IEEE (2016)