Retrieving batch organisation of work insights from event logs

Peer-reviewed author version

MARTIN, Niels; SWENNEN, Marijke; DEPAIRE, Benoit; JANS, Mieke; CARIS, An & VANHOOF, Koen (2017) Retrieving batch organisation of work insights from event logs. In: DECISION SUPPORT SYSTEMS, 100, p. 119-128.

DOI: 10.1016/j.dss.2017.02.012
Handle: http://hdl.handle.net/1942/24049

Retrieving batch organisation of work insights from event logs

Niels Martin, Marijke Swennen, Benoît Depaire, Mieke Jans, An Caris and Koen Vanhoof

Hasselt University, Agoralaan Building D, 3590 Diepenbeek, Belgium

Abstract

Resources can organise their work in batches, i.e. perform activities on multiple cases simultaneously, concurrently or intentionally defer activity execution to handle multiple cases (quasi-) sequentially. As batching behaviour influences process performance, efforts to gain insight on this matter are valuable. In this respect, this paper uses event logs, data files containing process execution information, as an information source. More specifically, this work (i) identifies and formalises three batch processing types, (ii) presents a resource-activity centered approach to identify batching behaviour in an event log and (iii) introduces batch processing metrics to acquire knowledge on batch characteristics and its influence on process execution. These contributions are integrated in the Batch Organisation of Work Identification algorithm (BOWI), which is evaluated on both artificial and real-life data.

Keywords: Batch processing, Event log, Event log insights, Process mining, Business process management, Process metrics

Accepted manuscript version. The final paper is published in Decision Support Systems (volume 100, pages 119-128) with DOI:10.1016/j.dss.2017.02.012.

1. Introduction

Business processes are composed of a series of connected activities executed by resources. Resources, such as process participants or equipment [1], are assigned to activities and typically carry these out on multiple cases such as files or products. Assuming that arriving cases are handled immediately when the resource becomes available is an undue simplification of reality. Employees might deem it more efficient to accumulate files and treat the entire stack later or machines can process multiple products at the same time. This type of resource behaviour is referred to as batch processing.

While the occurrence of batch processing might be readily observable for *passive resources* such as machines, it is typically less straightforward to determine how human resources, or *active resources* [1], organise their work. The latter is especially the case for processes in which staff members have a lot of freedom to arrange their tasks as they desire. Direct observation of staff members‘ behaviour has limitations as it is both time-consuming and the Hawthorne effect can cause observed behaviour to deviate from real behaviour when humans know they are being observed [2]. Consequently, investigating the use of more readily available information sources is valuable. In this respect, event logs containing process execution information can be analysed, which belongs to the process mining domain. While batch processing is studied widely in the operations management, operations research literature and, to a lesser extent, the process modelling domain, limited attention is attributed to this topic in process mining.

This paper is the first paper to systematically analyse batching behaviour using an event log. More specifically, the key contributions of this paper are threefold. Firstly, three types of batch processing are distinguished and

formalised. Secondly, a resource-activity centered approach is presented to identify these batch processing types from an event log. Finally, batch processing metrics are defined to describe the identified batches and the implications of batching on process execution. These contributions are included in the Batch Organisation of Work Identification algorithm (BOWI). Even though the contributions of this paper are of general interest, they are especially useful for business processes in which human resources have significant freedom in their work organisation. As extensive observations would, for instance, be required in such contexts, using the proposed technique allows companies to gain insight in batching behaviour from event data.

The current paper is situated at the intersection between Business Process Management and process mining, which corresponds to one of the focal points of this special issue. More specifically, the generated insights in batching behaviour will support process modelling activities and decision-making within the BPM life cycle [1]. Process modelling is facilitated as event log analysis can provide suitable values for parameters such as the batch size. Integrating batching behaviour will lead to more realistic process models which can, for instance, be used for simulation purposes. Simulation models serve as a decision support tool as it enables organisations to evaluate policy alternatives prior to implementation [3]. Besides shouldering process modelling, BOWI also provides direct support for decision-making. The algorithm allows companies to judge the desirability of batching behaviour by showing its influence on process performance. Consequently, a company can determine whether batching behaviour should be encouraged or discouraged. Besides positioning it in the BPM life cycle, this work can also be framed within the BPM use case *extend model* as it leverages resource information in the event log, which is useful to extend a process model [4].

The paper is structured as follows. Section 2 presents a running example and defines the three types of batch processing. The BOWI-algorithm is outlined in Section 3, after which it is evaluated on both artificial and real-life data in Section 4. Finally, related work, the algorithm's limitations and conclusions are presented in Sections 5, 6 and 7, respectively.

2. Preliminaries

This section presents some preliminary topics that will be used in the remainder of this paper. A running example is introduced in Section 2.1 and three batch processing types are distinguished in Section 2.2.

2.1. Running example

Throughout this paper, a simplified process at the emergency department of a hospital will serve as a running example. The process model, annotated with all assumed parameters, is visualised in Figure 1. After a patient registers at the reception (R), an initial triage and assessment by a doctor follows (T). Next, a patient either (i) undergoes an X-ray examination (X) or (ii) has laboratory tests performed on his blood samples (L) and is subjected to an MRI scan (M). When the required tests are completed, the patient discusses the further treatment with a medical specialist (S) after which the patient checks out (C). All time units are expressed in minutes and patient interarrival times and activity durations are assumed to follow an exponential and triangular distribution, respectively.

2.2. Batch processing type definition

Batch processing is defined as a type of work organisation in which a resource executes a particular activity on multiple cases simultaneously or

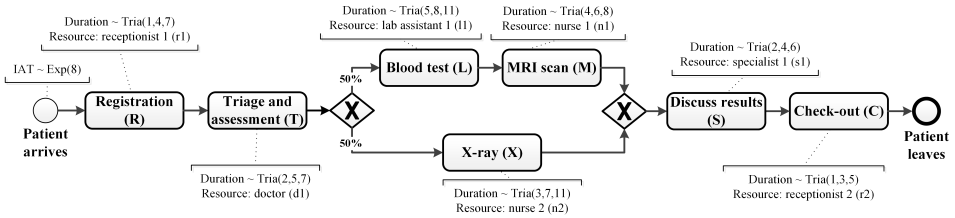

Figure 1: Process model running example

concurrently, or intentionally defers activity execution to handle multiple cases (quasi-) sequentially. As in [5], three batch types are distinguished: simultaneous, concurrent and sequential batch processing. To exemplify the difference between these types, Figure 2 shows the activities executed for six patients, where an activity is always executed by the same resource across all cases.

- **Sequential batch processing.** Activity instances are in a sequential batch when a resource intentionally defers the execution of this activity such that multiple cases can be handled (almost) immediately after each other. Consequently, all cases included in a batch need to be present at the activity before the resource starts processing the batch's first case. The latter distinguishes sequential batch processing from mere queue handling, stressing its intentional nature. In Figure 2, the initial assessment by a doctor takes place in sequential batches. Given the doctor's busy schedule, he occupies himself with other tasks until several patients need to undergo an initial assessment, after which they are handled sequentially.

- **Simultaneous batch processing.** Activity instances are in a simultaneous batch when they are executed by the same resource for distinct cases at exactly the same time. For example: blood samples

from several patients can be analysed in the same run, as shown in Figure 2.

- **Concurrent batch processing.** Activity instances are in a concurrent batch when they are executed by the same resource for distinct cases partially overlapping in time. This indicates that the resource can handle multiple cases at the same time, but is flexible as it is not required that processing starts and ends at the same time for all cases. In Figure 2, registrations and check-outs illustrate different types of concurrent batch processing, because, e.g., the receptionist already starts registering the next patient while the current patient is filling out a drug allergy form which is required to finish his registration.

The above batch processing types are largely consistent with [6], where simultaneous and sequential batch processing correspond to the concepts of parallel and serial process batches, respectively. Concurrent batch processing is not included in [6]. In operations management literature, batch processing is commonly referred to as the intermittent production of a particular type of product [7, 8], where production volumes are situated between a job shop setting with small volumes and mass production [7].

3. Batch Organisation of Work Identification Algorithm

This section proposes the Batch Organisation of Work Identification Algorithm (BOWI), which generates insights in batching behaviour from an event log. A general overview is presented in Section 3.1. Afterwards, Section 3.2-3.6 present the algorithm in more detail. In Section 3.7, the implementation of the algorithm is briefly discussed.

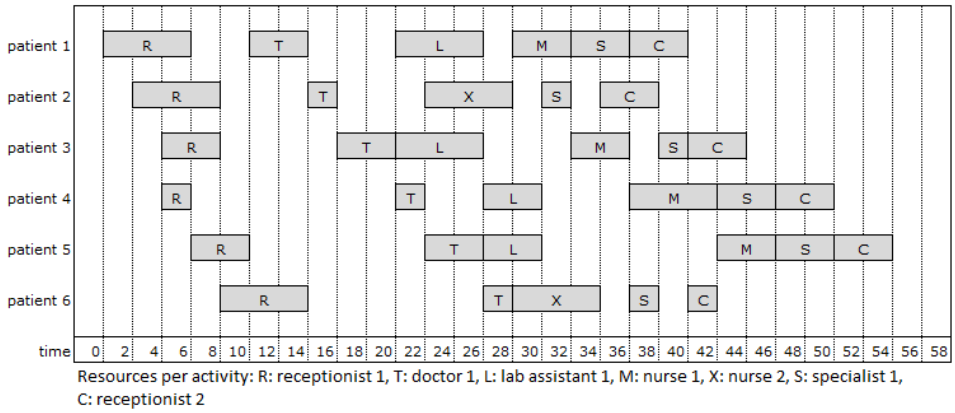

Figure 2: Conceptual representation of six cases executed in the running example process

3.1. General overview

As shown in Figure 3, BOWI's input is an event log. This event log, consisting of atomic events, is converted to an activity log by mapping start events to their corresponding complete events. This activity log is restructured in a resource-activity matrix (RAM), where each cell contains activity instances of a particular resource-activity combination. Using the RAM as an input, a batching matrix (BM) is created for each batch processing type. A BM mimics the structure of the RAM, but groups activity instances that are executed in the type of batch under consideration. This information is used to calculate batch processing metrics such as the batch size.

3.2. Event log requirements

BOWI requires an event log, composed of ordered events related to a particular case and activity, as input. For each event, the timestamp, executing resource and transaction type needs to be recorded. Two transaction types have to be registered for BOWI: start and complete. Moreover, each start event should have an accompanying complete event with the same resource being associated to both events.

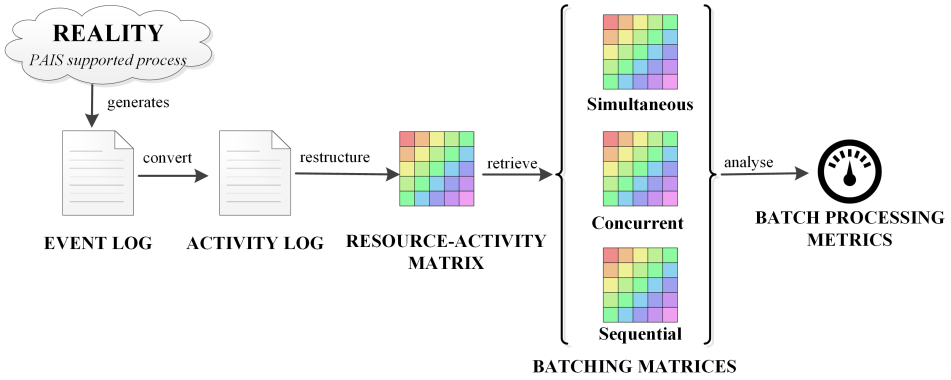

Figure 3: Overview of BOWI

While Table 1 illustrates the event log structure, its key characteristics can be formalised as follows:

Definition 1 (Event log characteristics)**.** *Let $\mathscr{E}$ be the set of all events included in event log E. Moreover, let $\forall e \in \mathscr{E}$:*

- $\#_{case}(e)$ *represents the case associated to event* e
- $\#_{activity}(e)$ *represents the activity associated to event* e
- $\#_{resource}(e)$ *represents the resource associated to event* e
- $\#_{time}(e)$ *represents the timestamp associated to event* e
- $\#_{trans}(e)$ *represents the transaction type associated to event* e

Then, in this paper, $\forall e \in \mathscr{E} : \#_{case}(e) \neq \perp \wedge \#_{activity}(e) \neq \perp \wedge \#_{resource}(e) \neq \perp \wedge \#_{time}(e) \neq \perp \wedge \#_{trans}(e) \in \{start, complete\}$*, where* $\perp$ *represents a null value. Moreover, every start event should have an accompanying complete event, i.e.* $\forall e_1 \in \mathscr{E}, \exists e_2 \in \mathscr{E} : \#_{case}(e_1) = \#_{case}(e_2) \wedge \#_{activity}(e_1) = \#_{activity}(e_2) \wedge \#_{resource}(e_1) = \#_{resource}(e_2) \wedge \#_{time}(e_1) \leq \#_{time}(e_2) \wedge \#_{trans}(e_1) = start \wedge \#_{trans}(e_2) = complete$*. When* $|e_2| > 1$*,*

Table 1: Illustration of event log structure

case id	timestamp	activity	transaction type	resource
...	...	...	...	...
patient 22	03/01/2016 11:14:41	x-ray	start	nurse 2
patient 22	03/01/2016 11:22:37	x-ray	complete	nurse 2
patient 25	03/01/2016 11:22:37	x-ray	start	nurse 2
patient 34	03/01/2016 11:22:54	blood test	start	lab assistant 1
patient 42	03/01/2016 11:25:17	registration	start	receptionist 1
patient 34	03/01/2016 11:28:02	blood test	complete	lab assistant 1
patient 42	03/01/2016 11:31:58	registration	complete	receptionist 1
patient 25	03/01/2016 11:32:18	x-ray	complete	nurse 2
...	...	...	...	...

i.e. when more than one event satisfies the conditions outlined for e_2, it is required that $|e_1| = |e_2|$.

3.3. Activity log creation

To retrieve batch processing insights, the atomic events in the event log are converted to activity instances, i.e. the execution of a particular activity by a particular resource on a particular case. To this end, each start event is mapped on its corresponding complete event, i.e. the complete event that is associated to the same case, activity and resource in the event log. When multiple start and complete events are present for a particular case, activity and resource combination, the first occurring unmapped start event will iteratively be mapped to the first occurring unmapped complete event. The activity log obtained from the event log in Table 1 is shown in Table 2.

Definition 2 (Activity log)**.** *Let L be an activity log based on event log E. Then A is composed of a set of activity instances $\mathscr{A}$. Each activity instance depicts the execution of an activity a by resource r on case c, started at time τ_{start} and completed at time $\tau_{complete}$. An activity instance*

Table 2: Illustration of an activity log

case id	**activity**	**resource**	$\tau_{\textbf{start}}$	$\tau_{\textbf{complete}}$
...	...	...	...	...
patient 22	x-ray	nurse 2	03/01/2016 11:14:41	03/01/2016 11:22:37
patient 25	x-ray	nurse 2	03/01/2016 11:22:37	03/01/2016 11:32:18
patient 34	blood test	lab assistant 1	03/01/2016 11:22:54	03/01/2016 11:28:02
patient 42	registration	receptionist 1	03/01/2016 11:25:17	03/01/2016 11:31:58
...	...	...	...	...

is represented by $\eta = (c, a, r, \tau_{start}, \tau_{complete})$*, where* $\#_k(\eta)$ *represents the value of attribute* k *for activity instance* η *as suggested for events in Definition 1. All activity instances* $i \in L$ *are sorted according to* τ_{start}*, i.e.* $\forall \eta_i, \eta_{i+1} \in L : \#_{\tau_{start}}(\eta_i) \leq \#_{\tau_{start}}(\eta_{i+1})$.

3.4. Resource-activity matrix

As the batch organisation of work reflects how resources execute an activity, batching behaviour will be identified at the resource-activity level. To this end, the activity log is restructured in a RAM, where each cell contains activity instances associated to a particular resource-activity combination. An excerpt of the *nurse 2 - x-ray RAM* cell is shown in Table 3.

Definition 3 (Resource-activity matrix)**.** *Let RAM represent the resource-activity matrix and let* $RAM(a, r)$ *be the cell of RAM related to activity* a *and resource* r*. Then* $RAM(a, r) = \{\eta \in L \mid \#_{activity}(\eta) = a \wedge \#_{resource}(\eta) = r\}$.

To prepare the RAM for analysis, immediate rework is removed, which refers to the repeated execution of a particular activity by a resource on the same case (almost) immediately after each other. Immediate rework is not consistent with the definition of batch processing as batching focuses on activity execution on distinct cases. Consequently, for these instances immediate rework is replaced by a single activity instance with τ_{start} the

Table 3: Illustration of RAM cell *nurse 2 - x-ray*

case id	$\tau_{\mathbf{start}}$	$\tau_{\mathbf{complete}}$
...	...	...
patient 22	03/01/2016 11:14:41	03/01/2016 11:22:37
patient 25	03/01/2016 11:22:37	03/01/2016 11:32:18
patient 27	03/01/2016 11:52:03	03/01/2016 12:03:01
patient 28	03/01/2016 12:03:01	03/01/2016 12:11:51
...	...	...

start timestamp of the first immediate rework instance and $\tau_{complete}$ the complete timestamp of the last immediate rework instance.

3.5. Batching matrices

In general, a batch is a set of activity instances. To identify batches, BOWI creates a batching matrix (BM) for each batch processing type. The structure of these BMs mimics the RAM, i.e. each cell focuses on one resource-activity combination. Taking a RAM cell as input, activity instances are grouped based on the conditions of the batch type under consideration. These instance sets are recorded in the corresponding BM cell, where instances in a singleton set could not be grouped based on the type of batch under consideration.

Definition 4 (Batch)**.** *A batch b is a set of activity instances $\eta \in L$, for which $\forall \eta_i, \eta_j \in b : \#_{activity}(\eta_i) = \#_{activty}(\eta_j) \wedge \#_{resource}(\eta_i) = \#_{resource}(\eta_j)$, i.e. all instances in b originate from a particular cell $RAM(a, r)$ in the RAM.*

The definitions of the three BMs can, consistent with Section 2.2, be formalised as follows:

Definition 5 (Simultaneous batching matrix)**.** *Let BM_{sim} represent the simultaneous batching matrix and let $BM_{sim}(a, r)$ be the cell of BM_{sim} related to activity a and resource r. Then $BM_{sim}(a, r)$ consists of a set of batches $\mathscr{B}$. When b represents a batch in $\mathscr{B}$, then $\forall \eta_i, \eta_j \in b : \#_{\tau_{start}}(\eta_i) = \#_{\tau_{start}}(\eta_j) \wedge \#_{\tau_{complete}}(\eta_i) = \#_{\tau_{complete}}(\eta_j)$ $(\forall b \in \mathscr{B})$. Moreover, $\forall b_i, b_j \in \mathscr{B} : b_i \cup b_j \notin \mathscr{B}$, i.e. any combination of batches in $BM_{sim}(a, r)$ does not fulfill the aforementioned conditions.*

Definition 6 (Concurrent batching matrix)**.** *Let BM_{conc} represent the concurrent batching matrix and let $BM_{conc}(a, r)$ be the cell of BM_{conc} related to activity a and resource r. Then $BM_{conc}(a, r)$ consists of a set of batches $\mathscr{B}$. When b represents a batch in $\mathscr{B}$, then $\forall \eta_i, \eta_{i+1} \in b : \#_{\tau_{start}}(\eta_i) \leq \#_{\tau_{start}}(\eta_{i+1}) < \#_{\tau_{complete}}(\eta_i) \wedge (\#_{\tau_{start}}(\eta_i) \neq \#_{\tau_{start}}(\eta_{i+1}) \vee \#_{\tau_{complete}}(\eta_i) \neq \#_{\tau_{complete}}(\eta_{i+1}))$ $(\forall b \in \mathscr{B})$. Moreover, $\forall b_i, b_j \in \mathscr{B} : b_i \cup b_j \notin \mathscr{B}$, i.e. any combination of batches in $BM_{conc}(a, r)$ does not fulfill the aforementioned conditions.*

While the formalisation of the simultaneous and concurrent BM directly follows from the definition of the respective batch type in Section 2.2, the specification of the sequential BM is subject to more restrictions. Firstly, the time between the complete timestamp of a case and the start timestamp of the next should be lower than γ. This parameter can be set to a strictly positive value to accommodate, e.g., some set-up time required to open a new file after the previous one is processed. However, γ should be small to remain consistent with the idea of batch processing and no other resource activity can be recorded while processing cases in a sequential batch.

Secondly, to integrate the distinction between sequential batch processing and regular queue handling, a function ϕ is introduced. This function

returns the time at which a case arrives at the activity under consideration. Case arrival is proxied by the completion time of the prior activity executed on this case. Consequently, the preceding activity needs to be known. This control-flow notion can be retrieved using domain knowledge or by applying a control-flow discovery algorithm on the event log. Given the large body of research on the latter [9], the operationalisation of ϕ is not treated here.

Thirdly, none of the cases can be included in a simultaneous or concurrent batch for the activity under consideration. This way, it can be avoided that sequences of these two batch types are treated as a sequential batch.

Finally, if multiple cases arrive at the same time at the activity under consideration, they can only form a sequential batch when the first case in this batch is not processed (quasi-)immediately upon arrival. This can, for example, be relevant when the activity preceding the activity under consideration is executed in a simultaneous batch.

Definition 7 (Sequential batching matrix)**.** *Let BM_{seq} represent the sequential batching matrix and let $BM_{seq}(a, r)$ be the cell of BM_{seq} related to activity a and resource r. Then $BM_{seq}(a, r)$ consists of a set of batches $\mathscr{B}$. When b represents a batch in $\mathscr{B}$, then $\forall \eta_i, \eta_{i+1} \in b$, the following conditions cumulatively hold:*

- $(\#_{\tau_{start}}(\eta_{i+1}) - \#_{\tau_{complete}}(\eta_i)) \in [0, \gamma]$, *with* $\gamma \geq 0$
- $\nexists\, e \in E : \#_{resource}(e) = \#_{resource}(\eta_i) \wedge \#_{\tau_{complete}}(\eta_i) \leq \#_{time}(e) \leq \#_{\tau_{start}}(\eta_{i+1})$
- $\phi(\eta_i) \leq \#_{\tau_{start}}(\eta_1) \wedge \phi(\eta_{i+1}) \leq \#_{\tau_{start}}(\eta_1)$, *where η_1 represents the first processed case in b and $\phi(\eta_x)$ is a function returning the arrival of case $\#_{case}(\eta_x)$ at activity a*

- $\eta_i, \eta_{i+1} \notin \{b' \,|\, (b' \in BM_{sim} \vee b' \in BM_{conc}) \wedge |b'| > 1\}$, *with* $|b'|$ *expressing the number of activity instances included in batch* b'
- *when* $\phi(\eta_i) = \phi(\eta_{i+1}) = \phi(\eta_1)$, *then* $\#_{\tau_{start}}(\eta_1) > \phi(\eta_i) + \gamma$, *with* $\gamma \geq 0$

$(\forall b \in \mathscr{B})$. *Moreover,* $\forall b_i, b_j \in \mathscr{B} : b_i \cup b_j \notin \mathscr{B}$, *i.e. any combination of batches in* $BM_{seq}(a, r)$ *does not fulfill the conditions outlined above.*

An appropriate value of γ can be determined using domain knowledge. When such knowledge is unavailable, the event log can support its specification by studying time differences between complete and start timestamps of subsequent non-overlapping activity instances.

To illustrate Definitions 5-7, they are applied to all activity instances in *RAM* cell *nurse 2 - x-ray*, depicted in Table 3. Two sequential batches are found, causing $\{c_{22}, c_{25}\}$ and $\{c_{27}, c_{28}\}$ to be added to the $BM_{seq,nurse2-X-ray}$ cell. In the other BMs, these instances are added as singletons.

The three batching matrices contain all batch identification information. It should be noted that batch identification is independent of the complexity of the process control-flow as it is situated on the resource-activity level. Control-flow complexity can render the imputation of arrival events more difficult as this requires knowledge on the prior activity for a particular case. Arrival times are used in Definition 7 in an effort to distinguish sequential batching from regular queue handling.

3.6. Batch organisation of work metrics

Using the information in the batching matrices as input, batch processing metrics are defined, which describe batching behaviour and provide insight in its business value. Table 4 provides an overview of the metrics, which

Table 4: Batch organisation of work metrics

Metric	Description
Frequency of batch processing	The absolute and relative number of times that a set of $BM_x(a, r)$ contains two or more activity instances.
Batch size	Summary statistics of the number of activity instances in each set of $BM_x(a, r)$, both including and excluding sets of size one.
Number of cases included in a batch	The absolute and relative number of cases that appear in each set of $BM_x(a, r)$.
Duration of activity instances in a batch	Summary statistics of the difference between the duration of the activity instances in each set of size two or more in $BM_x(a, r)$ compared to the duration for sets of size one.
Waiting time of activity instances in a batch	Summary statistics of the difference between the waiting time of the activity instances in each set of $BM_x(a, r)$ compared to activity instances not in this set.
Overlap in concurrent batches	Summary statistics of the amount of time that the activities in a concurrent batch are actually concurrent.

are briefly explained in the remainder of this subsection. All metrics are defined on the resource-activity level. However, aggregations to other levels of analysis such as the activity level, the resource level and the level of the complete event log can be derived.

3.6.1. Frequency of batch processing

The batch processing frequency expresses how often a particular type of batching takes place in an absolute sense and relative to the number of sets in the corresponding BM cell. In Figure 2, for instance, the laboratory test (L) is performed by the lab assistant in two simultaneous batches, which accounts for 100 % of the sets in the associated BM_{sim} cell.

3.6.2. Batch size

Besides knowing how frequent a particular batch type occurs, the batch size is another valuable metric. As BMs also include singleton sets, indicating that an instance is not part of a batch of this type, batch size

summary statistics (such as the mean, median, standard deviation, etc.) can be calculated both including and excluding singleton sets. In Figure 2, two simultaneous batches of laboratory tests (L) of size two are observed. This generates, for instance, a mean batch size of two and an associated standard deviation of zero.

3.6.3. Number of cases included in a batch

This metric combines insights from the prior two metrics by determining the number of cases that are included in a particular type of batch. For instance, 100 % of the patients receiving a blood test have this test executed in the laboratory (L) as part of a simultaneous batch in Figure 2.

For concurrent batch processing, an additional calculation can be performed. According to Definition 6, concurrent batching requires that subsequent instances have an overlap in time. However, this does not imply that all instances in a batch overlap. Consequently, summary statistics on the number of cases that a resource actually handles concurrently can be determined. The registration (R) of patient 1 in Figure 2 is for example not overlapping in time with the registration of patients 5 and 6, while they are still in the same concurrent batch.

3.6.4. Duration of activity instances in a batch

The effect of batching on activity duration can be determined as resources may become more efficient when they have to perform a similar task on multiple cases. To this end, the duration of instances included in a batch are compared to the duration for instances that are not part of a batch. A nurse might need, e.g., 20 minutes to perform an X-ray for three patients sequentially, while she needs 10 minutes to conduct an X-ray separately as she needs to get accustomed to the settings of the machine.

3.6.5. Waiting time of activity instances in a batch

The global efficiency gain of batch processing should be weighed against the increase in waiting time for individual cases, which is the elapsed time between case arrival and the start of activity execution. In Figure 2, patient 1 has to wait until t_{10} before triage and assessment (T) starts, even though the patient is registered at t_6 and the doctor could have assessed him at t_6. As is the case for activity duration, waiting time summary statistics are provided for batched cases and non-batched cases.

3.6.6. Overlap in concurrent batches

For concurrent batching, the time overlap between batched cases can also be calculated. This enables the organisation to determine whether an employee starts working on the next case right before he finishes the previous one, or immediately after its start. Summary statistics are provided for the percentage overlap in time. In Figure 2, the mean overlap between concurrently batched cases for the check-out (C) activity equals 62.5%.

3.7. Implementation

BOWI is fully implemented using R[1], a programming language for which a large set of packages is available which can be used to create application-specific functions. The key packages that are used are *dplyr* for data manipulations such as sorting and data summarisations, *lubridate* to work with timestamps and *reshape* for converting the event log to an activity log.

Algorithm 1 provides the pseudocode for BOWI's batch identification component. It directly follows from the formalisation introduced in this section and shows that batches are identified from an activity log by parsing

[1] https://www.r-project.org/

it once and comparing each line in this log with the prior one. In this way, the algorithm enriches the activity log with batch information by adding two columns: (i) a batch number, grouping activity instances that belong to the same batch and (ii) the batch type, indicating which of the three batching types prevails. This is all information required to create BMs and calculate the metrics. Each metric is implemented as a separate function, which makes the framework easily extendable with additional metrics.

4. Evaluation

A twofold approach is used to evaluate the algorithm: Section 4.1 focuses on BOWI's ability to correctly rediscover batches in artificial event logs and Section 4.2 discusses the application of the algorithm on real-life logs.

4.1. Artificial event logs

4.1.1. Experimental design

BOWI's performance is evaluated by investigating its ability to rediscover known batches solely using an artificial event log. To this end, an artificial log is generated based on a generalised version of the process model in Figure 1. For each of the seven resource-activity combinations, it is randomly determined whether no, simultaneous, sequential or concurrent batching prevails with all options having the same probability. In the latter three cases, an integer batch size is randomly drawn from the set {2,3,4,5}. Given these inputs, the event log generator autonomously determines which cases are batched for each activity and generates a log considering 500 cases that enter the process. The data file also indicates which cases are grouped as a batch of a particular type. This information is only used for evaluation purposes and is removed from the event log that is provided to BOWI.

Algorithm 1 Batch organisation of work identification

Input: *eventLog*: an event log (list of complex objects representing events), *controlFlowNotion*: knowledge on the prior activity that is executed for a case to support (when required) arrival event imputation, *tolerances*: time tolerances for sequential batch processing

Output: a: activity log with batching information (list of complex objects representing activity instances)

1: $eventLog \leftarrow$ addArrivalEvents($eventLog$, $controlFlowNotion$)
▷*imputes (when required) arrival events using knowledge on the prior activity executed for a case*
2: $a \leftarrow$ convertToActivityLog($eventLog$)
▷*creates activity instances by mapping corresponding events*
3: $a \leftarrow$ sortActivityLog(a)
▷*sort rows in activity log based on variables in following order: activity, resource, start timestamp and complete timestamp*
4: $a \leftarrow$ removeImmediateRework(a)
▷*removes immediate rework from activity log*
5: $batchNumber \leftarrow 1$
▷*initialise value - instances in a batch will have the same batchNumber*
6: $a[1].batchNr \leftarrow batchNumber$ ▷*initialise batchNumber value for first instance in activity log*
7: $firstCaseStart \leftarrow a[1].start$
▷*initialise value representing the start timestamp of the first case of a potential batch*
8: $tol \leftarrow$ getTolerance($tolerances$, $a[1].activity$, $a[1].resource$)
▷*determines sequential batch proc.time tolerance for particular resource-activity combination*
9: $n \leftarrow$ numberOfRows(a) ▷*number of rows in activity log*
10: **for** $i = 2$ to n **do**
11: $currentActivity \leftarrow a[i].activity$ ▷*activity of instance under analysis*
12: $priorActivity \leftarrow a[i-1].activity$ ▷*activity of prior instance in a*
13: $currentResource \leftarrow a[i].resource$
14: $priorResource \leftarrow a[i-1].resource$
15: $currentArrival \leftarrow a[i].arrival$
16: $currentStart \leftarrow a[i].start$
17: $priorStart \leftarrow a[i-1].start$
18: $currentComplete \leftarrow a[i].complete$
19: $priorComplete \leftarrow a[i-1].complete$
20: $priorBatchType \leftarrow a[i-1].batchType$ ▷*batch type to which the prior case belongs*
21: **if** $currentActivity == priorActivity$ **and**
22: $currentResource == priorResource$ **then**
23: **if** $currentStart == priorStart$ **and** ▷*simultaneous batch processing*
24: $currentComplete == priorComplete$ **and**

```
25:             priorBatchType is empty or simultaneous then
26:             a[i].batchNumber ← batchNumber
27:             a[i].batchType ← simultaneous
28:             if a[i − 1].batchType is empty then
29:                 a[i − 1].batchType ← simultaneous
30:             end if
31:         else if currentStart ≥ priorStart and                    ▷concurrent batch processing
32:             currentStart < priorComplete and
33:             currentComplete ≠ priorComplete and
34:             priorBatchType is empty or concurrent then
35:             a[i].batchNumber ← batchNumber
36:             a[i].batchType ← concurrent
37:             if a[i − 1].batchType is empty then
38:                 a[i − 1].batchType ← concurrent
39:             end if
40:         else if currentStart ≥ priorComplete and                 ▷sequential batch processing
41:             currentStart ≤ priorComplete + tol and
42:             currentArrival ≤ firstCaseStart and
43:             !RESOURCEACTIVE(a, currentResource, priorComplete, currentStart) and
44:             priorBatchType is empty or sequential then
45:             a[i].batchNumber ← batchNumber
46:             a[i].batchType ← sequential
47:             if a[i − 1].batchType is empty then
48:                 a[i − 1].batchType ← sequential
49:             end if
50:         else                                                     ▷start a new batch
51:             batchNumber ← batchNumber + 1
52:             a[i].batchNumber ← batchNumber
53:             firstCaseStart ← currentStart
54:         end if
55:     else          ▷subsequent instances belong to different resource-activity combination
56:         batchNumber ← batchNumber + 1
57:         a[i].batchNumber ← batchNumber
58:         firstCaseStart ← currentStart
59:         tol ← GETTOLERANCE(tolerances, currentActivity, currentResource)
    ▷adjust tolerance
60:     end if
61: end for
62: return a                          ▷returns activity log enriched with batching information
```

After executing BOWI on the event log, the algorithm's output is compared to the real batch composition. For a particular resource-activity combination, a case is correctly classified by BOWI when it is (i) contained in its correct batch in the BM of the batch type prevailing in reality and (ii) included as a singleton in the BMs of the other two batch types. Consequently, the evaluation centers around the detection of errors, which are (i) cases that are included in a batch of the correct type but in the wrong composition and (ii) cases being included in a batch of a particular type while they are not included in such a batch in reality. Using these conditions, the number of errors is calculated for each resource-activity combination. The first condition is defined rather rigorously as the composition of discovered batches has to be completely correct. For instance: when BOWI rediscovers a batch for all but one case, all cases in this batch are reported as errors because they are not part of the exact same batch prevailing in reality.

The aforementioned constitutes one experiment. To determine the number of experiments, an a priori power analysis for a one-sample Wilcoxon singed-rank test is conducted. To achieve a power value (i.e. the probability of rejecting the null hypothesis when it is false) of 0.80 [10] and given a family-wise significance level to 0.05 and effect size of 0.20 (the value proposed by [11] for the detection of small effects), the power analysis shows that at least 185 event logs need to be generated. Consequently, the number of artificial event logs is set to 200, which surpasses this lower bound.

4.1.2. Results

The application of the experimental design calculates, for each resource-activity combination in an event log, the number of errors. These results are aggregated by grouping resource-activity combinations in 12 classes, ex-

Table 5: Summary statistics on the error proportion of BOWI's output

Event log input - BOWI output	**Error proportion**				
	mean	**sd**	**median**	**min**	**max**
seq - seq	0.08	0.15	0.01	0.00	1.00
no batch - seq	0.54	0.16	0.57	0.13	0.82
all 10 other classes	0.00	0.00	0.00	0.00	0.00

pressing a combination of the real batch type in the event log (no batching, simultaneous, concurrent or sequential batching) and BOWI's output (simultaneous, concurrent or sequential BMs). For each of them, a decimal error proportion is calculated by dividing the number of errors by the number of cases that are included in the real batches for that class.

Table 5 reports summary statistics on the error proportions detected for the 12 classes over all 200 event logs. With *'seq - seq'* and *'no batch - seq'* as an exception, all classes show that BOWI's output is free from errors. This confirms that BOWI can rediscover existing batches solely using the event log. Moreover, the algorithm does, e.g., not detect sequential batch processing when concurrent batch processing prevails.

Regarding BOWI's detection of sequential batch processing, errors are detected when either sequential batch processing prevails in reality or no batch processing takes place. For an event log in which sequential batch processing is introduced, BOWI does not rediscover the exact composition of these batches for, on average, 7.62 % of batched cases, with a standard deviation of 15.41 % point. These errors are fairly concentrated as an exact match, i.e. an error proportion of zero is present for 243 of the 352 observations (69.03 %). For the remaining 109 observations, several explanations for the observed deviations can be identified. When sequential batch processing is inserted in the event log for the first activity, no arrival proxy will be available in the resulting event log as no prior activity is present. Consequently,

conditions related to the arrival proxy in Definition 7 cannot be checked, leading to a less stringent definition. This can cause multiple batches of a particular size that are executed one after another to be included as a single batch in BOWI's output. The same holds when the activity under analysis is preceded by an activity where simultaneous batch processing prevails with a higher batch size than the batch size for the activity under analysis. When the arriving simultaneous batch is processed immediately upon arrival, BOWI will detect, e.g., a batch of size four instead of two batches of size two. Even though this will be included as an error in Table 5, BOWI's output is a valid representation of business intuition in this case.

When no batch processing is included for a resource-activity combination in the event log, i.e. when all cases are expected to be included as a singleton in each of the BMs, the error proportion of BOWI is higher. The mean error proportion equals 54.08 % with a standard deviation of 16.20 % point and a median of 56.77 %. Studying the error proportion on the activity level for the *'no batch - seq'* situation shows that it is the highest for the start activity. This can, once again, be attributed to the less strict definition due to the absence of an arrival proxy. For the other activities, errors can be explained by the arrival of cases in, e.g., a simultaneous batch which is not handled immediately upon arrival. Even though it is recorded as an error, it presents a valid occurrence of sequential batch processing in a business context. Even when cases arrive separately, sequential batch processing can also be detected when long queues are formed. In this case, a subset of queueing cases fulfills the conditions of Definition 7. Despite the fact that Definition 7 aims to distinguish between regular queue handling and sequential batch processing, it should be noted that the definition aims to strike a balance between accuracy and clarity. Instead of enumerating and

excluding all possible exceptions, leading to an incomprehensible definition, a limited set of understandable conditions is specified.

For the sake of completeness, a one-sided one-sample Wilcoxon signed-rank test is performed for each class in Table 5 testing the null hypothesis that the median observed error proportion is zero against the alternative hypothesis that it is larger than zero. This test is used as the normality assumption underlying the t-test is not deemed appropriate in the current context. As anticipated, the null hypothesis is rejected for *'seq - seq'* ($V = 5995, p :< 2.2 \cdot 10^{-16}$) and *'no batch - seq'* ($V = 53301, p :< 2.2 \cdot 10^{-16}$), even when a Bonferroni correction [12] is applied with a family-wise significance level of 5 %. For the other combinations of event log input and BOWI's output, no test statistics can be calculated as all observations equal zero.

4.2. Real-life event log

To demonstrate that BOWI can generate insights in batching behaviour in a real world business context, the algorithm is applied to real-life event logs from two different contexts: a call center and a production company.

4.2.1. Event log of a call center

BOWI is applied to a real-life event log, based on data of a bank's call center made available by the Technion Service Enterprise Engineering Center[2]. Incoming calls are directed to a voice response unit (VRU), where automated voice information guides the caller. When the VRU does not enable callers to service themselves, they are redirected to a queue, after which they are connected to an agent. After converting the dataset to an event

[2]http://ie.technion.ac.il/Labs/Serveng

log format, 34 resource-activity combinations are included. More specifically, the log contains the *VRU - Handling by VRU* combination and the activity *Handling by agent*, which is executed by 33 distinct staff members. The results reported in this section are based on an analysis of 169065 calls registered in the first semester of 1999.

Within the analysis set, batching behaviour is detected for 31 resource-activity combinations. For *Handling by VRU*, which is always handled by resource *VRU*, both concurrent and simultaneous batching is detected, with respectively 26 % and 0.33 % of all calls being batched. The significant number of calls handled concurrently is due to the VRU's design to handle multiple calls concurrently on different lines. Simultaneous batching is present to a far lesser extent as it requires that, by coincidence, multiple calls arrive at exactly the same time and require the same processing time.

For 30 out of 33 agents performing *Handling by agent*, batching behaviour is detected. Concurrent batching is present, but its prevalence is low as, on average, only 1.85 % of the calls are included in a concurrent batch. Sequential batch processing is also discovered, but to a far lesser extent with an average of 0.03 % of the calls belonging to a sequential batch. When focusing on concurrent batching, Table 6 summarises some of BOWI's metrics for the five agents handling calls concurrently the most often.

Table 6 shows that, even for the agents for which concurrent batching is observed the most, the proportion of batched calls is rather limited as it ranges between 2.05 % and 2.53 %. The mean batch size varies between 2.16 and 2.23 calls. Hence, batching is not fundamentally integrated in the operations of a call center, which could be anticipated given its characteristics. Concurrent batching can take place when an agent already takes another call while the caller is looking for a particular document or the agent is

Table 6: BOWI metrics calculated for concurrent batching by five resources for activity *Handling by agent* in the call center event log

agent	**frequency**	**batch size**		**# batched**	**duration (mean)***		**time overlap**
		mean	sd	**cases (rel.)**	batch	no batch	
SHARON	152	2.23	0.42	2.49	4.27	2.28	0.46
KAZAV	121	2.17	0.39	2.53	4.71	3.21	0.47
MORIAH	114	2.18	0.38	2.64	4.21	3.14	0.50
TOVA	107	2.16	0.39	2.54	4.32	2.84	0.47
STEREN	87	2.18	0.39	2.05	6.13	3.04	0.52

** expressed in minutes*

awaiting input from the bank. This is supported by the fact that the mean duration tends to be longer for batched calls than for non-batched calls.

4.2.2. Event log of a production company

BOWI is also applied to a real-life event log of a production process, which is available at the 4TU Data Center[3]. It contains process execution data for 225 cases undergoing activities such as *flat grinding* and *packing*. In the log, 27 distinct activities and 31 unique resources are included.

Applying BOWI shows that batch processing is detected for 29 of the 57 resource-activity combinations in the event log. More specifically, simultaneous, concurrent and sequential batching is present for respectively, 9, 25 and 17 resource-activity combinations. This includes 14 resource-activity pairs for which both concurrent and sequential batches are present and 7 resource-activity pairs for which all batch types are detected.

Using the *number of cases included in a batch* metric, it is concluded that concurrent batch processing is the most prevalent. When considering all resource-activity combination where concurrent batch processing occurs, on average 23.50 % of all cases is batched. For simultaneous and sequen-

[3]http://data.4tu.nl/repository/uuid:68726926-5ac5-4fab-b873-ee76ea412399

tial batching, this is 15.55 % and 11.31 % respectively. Consequently, the remainder of this discussion focuses on concurrent batching.

When concurrent batching occurs, an important part of the cases is batched. This indicates that batching is fundamentally integrated in the organisation's process. Table 7 summarises some BOWI metric values for the five resource-activity combinations for which the highest number of concurrent batches is detected. For these resource-activity combinations, the proportion of cases being part of a concurrent batch ranges from 28 % to 77 %. The batch sizes are situated between 2.34 and 3.33, with standard deviations between 0.61 and 2.11. The influence of batch processing on activity duration outlined in literature does not hold as batched cases tend to take longer than non-batched cases. It might be the case that batching takes place for a particular type of product, which requires less intensive processing. Concerning the difference in waiting times between batched and non-batched cases, the results are mixed depending on the resource-activity combination. From the time overlap metric, it follows that there is a significant overlap between concurrently handled cases. This indicates that genuine concurrent batch processing is detected, and not sequential batch processing with inaccurate timestamp registration.

5. Related work

BOWI is based on a distinction between simultaneous, concurrent and sequential batching. While [6] and [13] distinguish between the parallel and sequential execution of activities, other references such as [14] and [15] only consider simultaneous batch processing. Consequently, this paper presents a more versatile perspective on batch processing.

Table 7: BOWI metrics calculated for concurrent batching for five resource-activity combinations from the production company event log

res.-act. comb.*	**freq.**	**batch size (mean)**	**# batched cases (rel.)**	**duration (mean)**		**waiting time (mean)**		**time overlap**
				batch	no batch	batch	no batch	
1	121	3.33	0.77	2.23	1.27	22.92	48.11	0.86
2	118	2.83	0.66	1.74	1.27	18.83	17.54	0.79
3	61	2.36	0.39	2.31	1.45	45.90	58.37	0.65
4	34	2.47	0.34	6.07	5.88	7.15	3.35	0.51
5	29	2.34	0.28	6.45	5.11	5.77	9.09	0.50

** 1: Qual. Check 1 - Final Insp. Q.C., 2: Qual. Check 1 - Turn. & Mil. Q.C , 3: Machine 1 - Lapping, 4: Machine 4 - Turn. & Mil., 5: Machine 6 - Turn. & Mil.*

Batch processing is studied in several domains, but mainly within the field of operations management, with a key focus on topics such as order batching [16], scheduling [17, 18] and operational excellence [19, 20]. Often, the trade-off between additional waiting times and reduced setup costs is mentioned [19, 20]. This is also explicitly recognised in business process management literature [14, 21, 22, 23].

Within the process modelling and execution domain, [13] specify the concept of batch activities and identify specification parameters such as the batch size. While [13] focus on a single batch activity, [23] extend these concepts to batch regions. The latter are a series of model constructs such as activities that handle cases in a batch. Recently, batch processing is studied for activities in different processes by means of object life cycles [24]. As [13], [23] and [24] primarily focus on the activity level, they do not explicitly take into account that the organisation of work for a particular activity can differ among resources. BOWI includes this perspective by considering the resource-activity level as the key level of analysis. This is consistent with [21] given that batching strategies can differ among resources. While

[13], [23] and [24] focus on process modelling and the specification of execution semantics, [14] focus on performance evaluation of batch activities. Solely considering the simultaneous batch processing case, cost functions are defined for both service and waiting costs and an analytical solution is proposed making use of queuing theory. In this way, the benefits of introducing simultaneous batch processing can be quantified and a recommended batch size can be calculated. However, the suggested approach focuses on a single activity which, moreover, must fulfill the conditions of a particular queuing model [14]. As follows from the above discussion, related work tends to focus on modelling batch processing at design time. However, [22] suggest an approach to dynamically adjust the batch activity configuration parameters depending on, e.g., the planned maintenance of a machine. This more flexible perspective on batching is also included in [24], where a set of cases that might be batched is solely suggested to the resource.

Besides [15] and [25] as notable exceptions, no research attention is devoted to batch processing within the process mining field. This is consistent with [3], where the retrieval of batch processing insights from event logs is marked as a research gap. [15] consider the problem of mining the process control-flow when the process contains activities where simultaneous batch processing occurs. For these activities, the authors assume that, for a particular batch, events are only logged for one of the cases in this batch. This is similar to [21], where batched cases are temporarily merged and decomposed afterwards. In contrast, BOWI assumes that events are recorded for all individual cases in a batch. When this is not the case, the work presented in [15] forms a valuable starting point for the simultaneous batch processing case. In [15], a method is developed that aims to add the missing events of batched cases, after which, e.g., existing control-flow discovery algorithms

can be applied. This complemented log can also be used to apply BOWI.

[25] proposes a method to identify batch processing in which all resource actions, i.e. executions of activities, are placed on a timeline and grouped in so called chunks. A new chunk is started when the elapsed time between the end of an action and the start of the following action exceeds one hour. When a period such as a working day is composed of multiple chunks, [25] states that batch processing occurs. This paper extends the work of [25] in several ways. Firstly, in contrast to [25], BOWI does not make abstraction from the difference between activities, reflecting the fact that some activities might be more eligible for batch processing. Secondly, the arbitrary delay of one hour between periods of activity is replaced by a formal definition of several types of batch processing. Finally, BOWI complements the work of [25] by distinguishing between batch processing and regular queue handling.

6. Limitations

Despite BOWI's ability to mine and describe batching behavior from an event log, some limitations need to be recognised. Firstly, the log should contain both start and complete events and resource information, which is often not the case in existing real-life event logs. Moreover, the level of detail at which timestamps and resources are recorded determines the granularity at which batching behaviour is identified. When, e.g., only resource classes are recorded, no distinction can be made between specific resources.

Secondly, BOWI does not explicitly consider the issue of noise in timestamp registration. Hence, it relies on accurate event registration for each case, which can require that a process is backed by a system which automatically logs resource action instead of relying on manual intervention to log

events. Nevertheless, some features of BOWI should be highlighted related to inaccurate timestamp registration. For sequential batching, a time tolerance that is allowed between consecutive instances in a sequential batch can be specified. When the start and complete timestamps of cases in a simultaneous batch are not identical, BOWI will label it as a concurrent batch. However, the value of the time overlap metric will show a high overlap, indicating that it might be an inaccurately recorded simultaneous batch.

Thirdly, the creation of an activity log requires mapping corresponding start and complete events. When a case passes a resource-activity combination multiple times, each start event is mapped to the first occurring unmapped complete event. When this mapping does not correspond to reality, it will influence batch detection as the activity log is its key input.

Finally, a case's arrival time at an activity is needed to distinguish sequential batching from regular queue handling. When this information is not included in the event log, it can be proxied by the completion time of the prior activity. However, this requires control-flow insights, i.e. the prior activity needs to be known, which is not trivial for complex processes. However, the absence of such a proxy does not impede BOWI from being applied, but renders the conditions to detect sequential batching less strict.

7. Conclusion

This paper focuses on the retrieval of batch processing insights from an event log. To this end, three types of batching are identified and formalised, i.e. simultaneous, concurrent and sequential batching. Using these definitions, the Batch Organisation of Work Identification algorithm (BOWI) is developed to identify batches and calculate metrics summarising batching

behaviour. The algorithm is evaluated on artificial event logs, showing that it can rediscover batches under most circumstances. Moreover, BOWI is applied to real-life event logs from a call center and production context.

Future extensions of BOWI can generate even more versatile batch processing insights from an event log. Firstly, the effect of noise on BOWI's performance can be studied in order to make the algorithm more resilient to noise. Secondly, BOWI's scope can be broadened by considering multiple consecutive activities instead of only a single activity, which is mentioned in [15] and [21] and is consistent with the batch regions notion in [23]. Finally, insights in batch logic by modelling the reasoning behind batching is an additional analysis dimension. Batching logic can be modelled by mining batch activation rules, which can merely depend on the number of queueing cases or can be contingent on, e.g., the time of day or case attributes.

References

[1] M. Dumas, M. La Rosa, J. Mendling, H. A. Reijers, Fundamentals of business process management, Springer, Heidelberg, 2013.

[2] D. McBride, The process of research in psychology, Sage, Thousand Oaks, 2016.

[3] N. Martin, B. Depaire, A. Caris, The use of process mining in business process simulation model construction: structuring the field, Business & Information Systems Engineering 58 (1) (2016) 73–87.

[4] W. van der Aalst, Business process management: a comprehensive survey, ISRN Software Engineering 2013 (2013) 1–37.

[5] N. Martin, M. Swennen, B. Depaire, M. Jans, A. Caris, K. Vanhoof, Batch processing: definition and event log identification, CEUR Workshop Proceedings 1527 (2015) 137–140.

[6] K. Wu, Taxonomy of batch queueing models in manufacturing systems, European Journal of Operational Research 237 (1) (2014) 129–135.

[7] P. Murthy, Production and operations management, New Age International Publishers, New Delhi, 2005.

[8] M. Telsang, Production management, S. Chand & Company Ltd, New Delhi, 2005.

[9] W. van der Aalst, Process mining: discovery, conformance and enhancement of business processes, Springer, Heidelberg, 2011.

[10] J. Cohen, A power primer, Psychological Bulletin 112 (1992) 155–159.

[11] J. Cohen, Statistical power analysis for the behavioral sciences, Lawrence Erlbaum Associates, New Jersey, 1988.

[12] S. Holm, A simple sequentially rejective multiple test procedure, Scandinavian Journal of Statistics 6 (2) (1979) 65–70.

[13] L. Pufahl, M. Weske, Batch activities in process modeling and execution, Lecture Notes in Computer Science 8274 (2013) 283–297.

[14] L. Pufahl, E. Bazhenova, M. Weske, Evaluating the performance of a batch activity in process models, Lecture Notes in Business Information Processing 202 (2014) 277–290.

[15] Y. Wen, Z. Chen, J. Liu, J. Chen, Mining batch processing workflow models from event logs, Concurrency and Computation: Practice and Experience 25 (13) (2013) 1928–1942.

[16] S. Henn, S. Koch, G. Wäscher, Order batching in order picking warehouses: a survey of solution approaches, Springer, London, 2012.

[17] P. Brucker, Scheduling algorithms, Springer, Heidelberg, 2007.

[18] M. L. Pinedo, Scheduling: theory, algorithms, and systems, Springer, New York, 2012.

[19] E. M. Goldratt, Theory of constraints, North River Press, Croton-on-Hudson, 1990.

[20] R. Arbulu, I. Tommelein, K. Walsh, J. Hershauer, Value stream analysis of a re-engineered construction supply chain, Building Research & Information 31 (2) (2003) 161–171.

[21] J. Liu, J. Hu, Dynamic batch processing in workflows: model and implementation, Future Generation Computer Systems 23 (3) (2007) 338–347.

[22] L. Pufahl, N. Herzberg, A. Meyer, M. Weske, Flexible batch configuration in business processes based on events, Lecture Notes in Computer Science 8831 (2014) 63–78.

[23] L. Pufahl, A. Meyer, M. Weske, Batch regions: process instance synchronization based on data, in: Proceedings of the IEEE International Enterprise Distributed Object Computing Conference, 2014, pp. 150–159.

[24] L. Pufahl, M. Weske, Batch processing across multiple business processes based on object life cycles, Lecture Notes in Business Information Processing 255 (2016) 195–208.

[25] J. J. Nakatumba, Resource-aware business process management: analysis and support, Ph.D. thesis, Eindhoven University of Technology (2013).