Detection of Batch Activities from Event Logs

Niels Martin[a,b,c,*], Luise Pufahl[d,e,*], Felix Mannhardt[f,g]

[a]*Research Foundation Flanders (FWO), Egmontstraat 5, 1000 Brussel, Belgium*
[b]*Hasselt University, Research group Business Informatics, Martelarenlaan 42, 3500 Hasselt, Belgium*
[c]*Vrije Universiteit Brussel, Data Analytics Laboratory, Pleinlaan 2, 1050 Brussel, Belgium*
[d]*HPI, University of Potsdam, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany*
[e]*SBE, Technische Universitaet Berlin, Einsteinufer 17, 10587 Berlin, Germany*
[f]*SINTEF Digital, Department of Technology Management, Postboks 4760 Torgarden, 7465 Trondheim, Norway*
[g]*NTNU, Department of Computer Science, 7491 Trondheim, Norway*

Abstract

Organizations carry out a variety of business processes in order to serve their clients. Usually supported by information technology and systems, process execution data is logged in an event log. Process mining uses this event log to discover the process' control-flow, its performance, information about the resources, etc. A common assumption is that the cases are executed independently of each other. However, batch work – the collective execution of cases for specific activities – is a common phenomenon in operational processes to save costs or time. Existing research has mainly focused on discovering individual batch tasks. However, beyond this narrow setting, batch processing may consist of the execution of several linked tasks. In this work, we present a novel algorithm which can also detect parallel, sequential and concurrent batching over several connected tasks, i.e., subprocesses. The proposed algorithm is evaluated on synthetic logs generated by a business process simulator, as well as on a real-world log obtained from a hospital's digital whiteboard system. The evaluation shows that batch processing at the subprocess level can be reliably detected.

[*]Corresponding authors

Email addresses: `niels.martin@uhasselt.be` (Niels Martin), `luise.pufahl@hpi.de` (Luise Pufahl), `felix.mannhardt@sintef.no` (Felix Mannhardt)

Keywords: Business Process, Batch Activity, Batch Processing, Discovery, Process Mining, Batch Mining

1. Introduction

In providing products or services to customers or clients, organizations carry out a variety of business processes. A business process consists of a set of activities which are executed in an organizational and technical environment to reach a certain business goal [1]. The enactment of business processes is usually supported by a so-called process-aware information system (PAIS) (e.g. an Enterprise Resource Planning System or a Hospital Information System), where process execution data is registered in an event log [2]. Process mining as a research discipline deploys this type of data to provide insights into the nature of a business process to, e.g., discover a process model.

A common assumption in process mining is that cases, also called process instances, are executed independently from each other. However, batch processing is commonly applied to reduce cost or save time by collectively handling several process instances in a group at specific activities in a business process [3]. Especially in operations research, batching of products or customers is well discussed [4, 5]: in an effort to reduce the set-up or execution cost of physical resources, specific types of products are scheduled in batches on a machine, or customers at a service desk are combined in batches before starting a service. Also, process participants, who get assigned tasks for execution and can organize their work on their own, tend to handle their work in batches [6]. For instance, usually a set of invoices is collected first before they are checked instead of approving each one individually.

Although event logs are a valuable information source to identify batch behavior, the detection of so-called *batch activities* in event logs was discussed by only few works, such as [6, 7, 8, 9]. It enables companies to identify existing batching behavior and identify opportunities to integrate batch processing in a more structured way. Batch processing can occur as *parallel* batching (also referred to as *simultaneous* batching) where several items are processed simultaneously (e.g., blood tubes being tested by a laboratory machine), or *sequential* batching where the cases are still processed individually but, by batching a set, setup time/cost can be reduced (e.g., processing a set of invoices one after the other). Martin et al. [6] additionally consider *concurrent* batch work, which can be observed when a resource starts working on a task

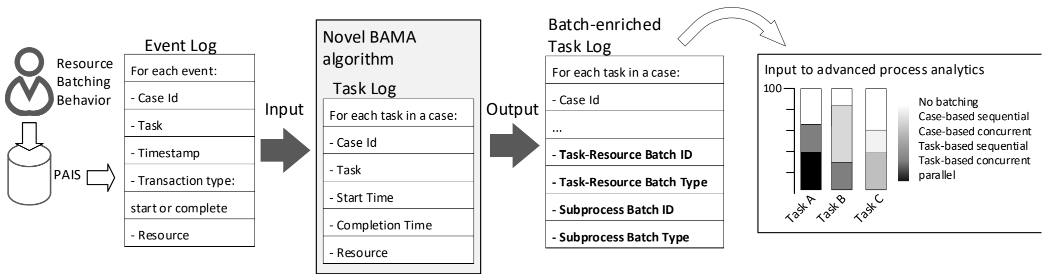


Figure 1: Input and result of the novel BAMA algorithm.

and, while executing it, also starts another task instance in parallel. So far, the focus of literature on batch mining in the process mining field was situated on discovering single batch tasks only, providing only a limited view on batch work. Pufahl and Weske [3] observed that batch processing operations can span not only over one task, but also over several connected ones.

In this paper, we present the novel Batch Activity Mining Algorithm (BAMA) which expands existing research by detecting different types of batch activities. It extends the algorithm proposed in Martin et al. [6], where batching behavior is identified at the level of individual tasks, to the more general notion of a batch activity. In BAMA, a batch activity can be either non-decomposable (i.e. a task) or can have internal behavior (i.e. a subprocess), and for both of them *parallel*, *sequential*, or *concurrent* behavior could be detected.

Thereby, we also consider that different types of batch behavior can be applied during a subprocess. In case of subprocesses, *sequential* batching can occur in two versions [10]:

- with a *task-based* orientation (i.e., cases are executed for each task in a subprocess in a sequential fashion) or,
- with a *case-based* orientation (i.e., first a case needs to finish all tasks of a subprocess before the next case of a batch can be started).

A similar distinction is made for *concurrent* batching. Human resources can apply these types of batch behavior for a set of cases to ease their work. Each staff member might apply his/her own strategy for specific tasks. The concrete batch execution of activities is often recorded in a PAIS, showing the potential of the proposed batch mining algorithm.

As shown in Figure 1, the novel BAMA takes as input – as other process mining techniques – an event log deduced from a PAIS. Each row represents an event expressing, for instance, the start or completion of a task. BAMA transforms the event log into a task log, where one row contains all timestamps for one task execution. The task log is used for batch mining purposes. The output of BAMA is a *batch-enriched task log*, a task log to which batching information is added. For each task executed for a specific case, i.e. for each task instance, it highlights whether it belongs to a batch by providing the batch ID and the type of batch behavior. The algorithm provides this at a task level (i.e., *task-resource*) and at a subprocess level. By its ability to detect both batch tasks and batch subprocesses, BAMA provides organizations with rich data-driven insights in batching behavior. This can be used for advanced analytics, e.g., the occurrences of different batching behavior for specific tasks can be analyzed. In this paper, the algorithm is formally described, based on which a prototypical implementation is developed. Besides an evaluation of the algorithm's effectiveness using synthetic data, this work also uses real-life data to illustrate how the algorithm's output can help practitioners to learn more about batching in their business process.

The remainder of this paper is structured as follows. Section 2 presents the related work. After providing the preliminaries on batch processing in Section 3, BAMA is introduced in Section 4. In Section 5, the algorithm's effectiveness is evaluated using artificial logs. Moreover, it is applied to a real-world event log obtained from a hospital's digital whiteboard system. The paper ends with a discussion in Section 6 and a conclusion in Section 7.

2. Related Work

Batch processing has been studied in various research fields. This section provides an overview of prior work related to (i) batch processing in operations management/research, (ii) batch activity modeling, and (iii) batch mining.

Batch processing in operations management/research. Batch processing is a highly relevant topic in operations management/research to save costs or time by processing or transporting identical jobs in batches, or by utilizing the same machine/tool set-up [11]. In the following, we do not aim to provide an exhaustive literature study, but highlight some of the research topics.

The order batching problem, which focuses on grouping orders in batches, is heavily studied in the warehousing context [12, 13, 14, 15]. To this end, optimization techniques such as integer programming [14, 15] or tabu search

[13] are used. Research on this topic is also conducted in other sectors such as the transportation [16, 17, 18] and the steel sector [19].

The scheduling of jobs in batches is also intensively studied as shown in review papers, such as [20, 21]. Typically, two types of batching are distinguished: *serial* batching (i.e. similar jobs are scheduled together to save setup costs, but they are still executed in sequence) and *parallel* batching (i.e. batches of jobs which can be executed at the same time).

Furthermore, operations research offers methods to study the queue of a batch service (i.e., a service which can handle a group of customers or jobs in parallel, such as a rollercoaster) to determine, for instance, the expected waiting time or the expected service time [5]. Thereby, different optimization policies are proposed. Consider, for instance, the *threshold rule*, which states that a batch is started when the length of the queue is equal or greater than a given threshold and the server is free [22]. Several studies (e.g., [23]) investigate how to determine the optimal threshold, and the resulting expected waiting time under varying assumptions.

Batch activity modeling. In the business process management field, several approaches were recently developed to capture and specify batch work in business process models [10, 24, 25]: Pufahl et al. [10, 26] suggest the *batch activity* as a new type of activity with a set of configuration parameters to set up batch processing for specific activities at design time. The configuration includes an activation rule, a maximum capacity, and the type of processing - *parallel* vs. *sequential*. The batch activity can be either a simple task or an activity with internal behavior (i.e. a subprocess). The performance evaluation of such batch activities is studied in [27]. Natschläger et al. [24] propose a so-called *compound activity* for which the optimization goal and the constraints need to be defined to enable batching. Pflug and Rinderle-Ma [25] focus on sequential batching and provide an algorithm to rearrange cases into similar batches which can be processed faster by resources as a way to organize work for activities with long queues.

Batch mining. To have more insights in unknown batching behavior in business processes, process mining techniques were developed [6, 7, 8, 9, 28]. These techniques use event logs and try to detect tasks for which resources perform their work in batches. Whereas Wen et al. [7] assume that process execution data is recorded at the level of a batch, Liu et al. [9], Nakatumba [8] and Martin et al. [6] use the common event log format in which events are recorded at the level of individual cases. Liu et al. [9] use an event log to mine association rules for a staff member to recommend work items for batching, which have been executed collectively in the past. Nakatumba [8] and Martin et al. [6] propose algorithms to retrieve batching behavior from an event log. Nakatumba [8] discovers a batch when a time gap of more than one hour is present between groups of activity instances. This work was extended by Martin et al. [6] by providing an algorithm to detect different types of batch work from a log, and providing performance indicators, such as the size of a batch, waiting time and duration of instances in a batch.

Martin et al. [6] not only differentiate between *parallel* and *sequential* batch processing, but also consider *concurrent* batching as it occurs in practice that resources work on different instances in a partially overlapping time-frame, which could be considered as unstructured batch behavior. To distinguish between sequential batching and regular queue handling, Martin et al. [6] assume that all cases need to be present in the queue before the resource starts processing a batch. When the arrival of a case at a task is recorded, as is the case in a Q-log in the queue mining field [29], this information can be immediately retrieved from the data.

Different from prior works, Weber et al. [30] provide a pre-processing step to an existing process discovery algorithm to identify subprocesses in which multiple instances are initiated in parallel, but executed independently. The latter holds, for instance, when several reviews are initiated and handled for one paper submission. Traditionally, these types of subprocesses are called multi-instance activities [31]. Multi-instance activities allow for running several instances of the same activity to handle several sub-items of one process instance. These instances might be executed independently of each other and not necessary by the same resource.

Besides batch mining and multi-instance initiation, other works, such as [32, 33], consider a broader spectrum of relations between instances, which were also called instance-spanning constraints [34]. Whereas Senderovich et al. [32] use inter-case dependencies to improve the quality of process predictions with a special focus on case priorities, Winter et al. [33] provide a technique to detect different kinds of instance-spanning constraints from a given event log to analyze a process regarding compliance issues. As the focus is on compliance analysis, different batching behaviors are not analyzed in depth, which is the goal of our work.

So far, the focus of batch mining algorithms was only on detecting single batch tasks [6, 7, 8, 9] or batching logic at the level of a single task [28].

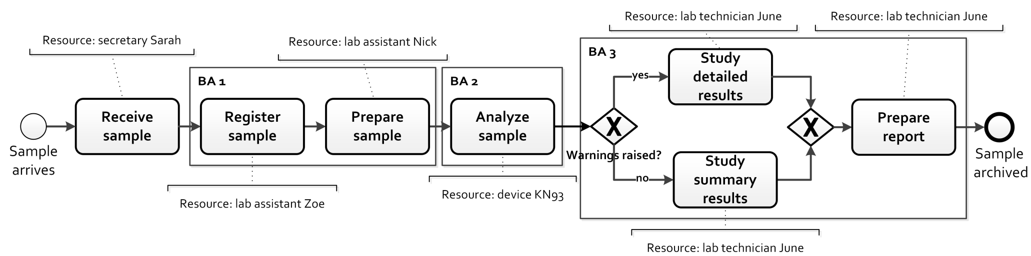


Figure 2: BPMN process diagram of running example 1 - a simplified laboratory process. Using the terminology that will be introduced in Section 3.2, the following batch processing types are included: BA 1 - sequential task-based batching, BA 2 - parallel batching, BA 3 - sequential case-based batching.

However, batching is often also done over several linked activities, i.e. in subprocesses, which can also have complex behavior, such as parallel activities. Given this research gap, this paper presents an algorithm which can detect batch behavior at both the task and subprocess level.

3. Preliminaries

The algorithm presented in this paper automatically detects batch activities from an event log. Before proceeding to the outline of the algorithm, this section outlines some preliminary concepts: Section 3.1 introduces the notion of a batch activity, Section 3.2 presents an overview of the batch processing types, and Section 3.3 provides an introduction to event logs.

3.1. Batch activity

Batch processing is an organization of work in which a resource bundles cases such that they can be processed as a group [3, 6]. A batch activity is a task or a subprocess in which batching behavior is present. We define a batch subprocess as a set of connected tasks for which cases are processed in batches and all cases stay in the same batch for all subprocess tasks.

To illustrate the notion of a batch activity, two running examples are introduced. The process models, annotated with the resource responsible to perform a task, are visualized as BPMN process diagrams in Figures 2 and 3.

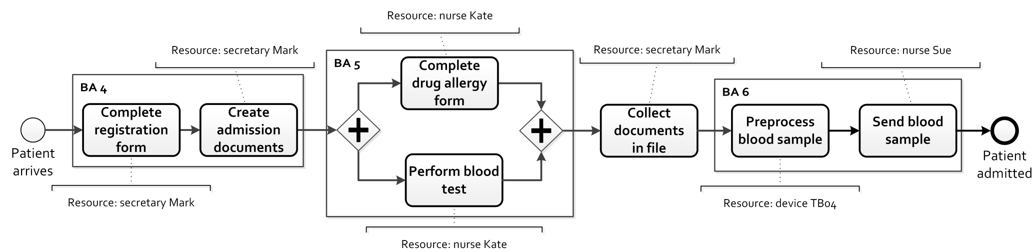


Figure 3: BPMN process diagram of running example 2 - a simplified patient admission process. Using the terminology that will be introduced in Section 3.2, the following batch processing types are included: BA 4 - concurrent task-based batching, BA 5 - concurrent case-based batching, BA 6 - parallel batching.

3.1.1. Running example 1: A laboratory process

The first running example relates to the simplified process that a blood sample follows at a hospital laboratory (Figure 2). When a blood sample is received by the secretary, it is registered and prepared for the analysis by a lab assistant. Afterwards, the actual analysis is conducted by a laboratory device. Depending on whether or not warnings are raised during the analysis, a lab technician studies either the detailed results or the summary results. After studying the results, a report is prepared.

Figure 2 shows that three batch activities are present: one consisting of a single task (BA 2) and two batch subprocesses (BA 1, BA 3). Besides differences in the number of tasks involved, the batch activities also vary in terms of the type of batching behavior. In the batch activity consisting of task *'Analyze sample'* (BA 2), multiple samples can be processed by the laboratory device simultaneously. In contrast, in BA 1, the lab assistant waits until a number of samples is waiting to be processed as this saves setup time. Once several samples are available, the lab assistant registers them one after the other, after which the same group of samples are prepared one at the time. In last batch activity, BA 3, the lab technician will also handle samples in groups. She studies the (detailed or summarized) results and prepares a report for a particular sample and then immediately proceeds to the next sample.

3.1.2. Running example 2: A patient admission process

The second running example focuses on a simplified patient admission process at a hospital unit (Figure 3). When a patient arrives, a registration form needs to be completed, after which the admission documents are created. Afterwards, a drug allergy form is completed and a blood test is performed, followed by the collection of all documents in the patient's file. When a number of blood samples have been collected, the samples are preprocessed using a specialized device and sent to the laboratory for further examination.

As shown in Figure 3, three batch subprocesses (BA 4, BA 5, BA 6) are present. BA 4 contains tasks *'Complete registration form'* and *'Create admission documents'*. While a particular patient is filling out the registration form, the secretary can already start registering another patient. Afterwards, while the admission files are being printed, the secretary can already compile the required documents for the next patient. The next batch subprocess, BA 5, consists of tasks *'Complete drug allergy form'* and *'Perform blood test'*. While the patient is filling out a drug allergy form, the nurse already starts performing the blood test for this patient. Only after several blood samples are available, BA 6 is executed. This involves preprocessing multiple blood samples simultaneously, after which all samples are sent to the laboratory at once for additional analyses.

From these running examples, it follows that several types of batching behavior can be distinguished, which will be introduced in the next subsection.

3.2. Batch processing types

BAMA will distinguish five types of batch processing. These differ according to two dimensions: (i) the time relationship between batched cases, and (ii) the task- or case-based orientation of a batch. With respect to the *time relationship* between the execution of tasks on batched cases, a distinction is made between parallel, sequential and concurrent batching. This is consistent with Martin et al. [6], where batching behavior is considered at the level of individual tasks. However, as our algorithm is more generic and also considers batch detection at the level of a subprocess, an additional distinction is made between *task-based batching and case-based batching* [10]. While work is organized around tasks in the former (i.e. the same task is executed for a group of cases before switching to another task), the latter centers around cases (i.e. several tasks are performed for a particular case before switching to another case). When both dimensions are taken into account, five types of batching behavior can be distinguished. Figure 4 illustrates these batch processing types considering three tasks (A, B, C) and three batched cases (c1, c2, c3).

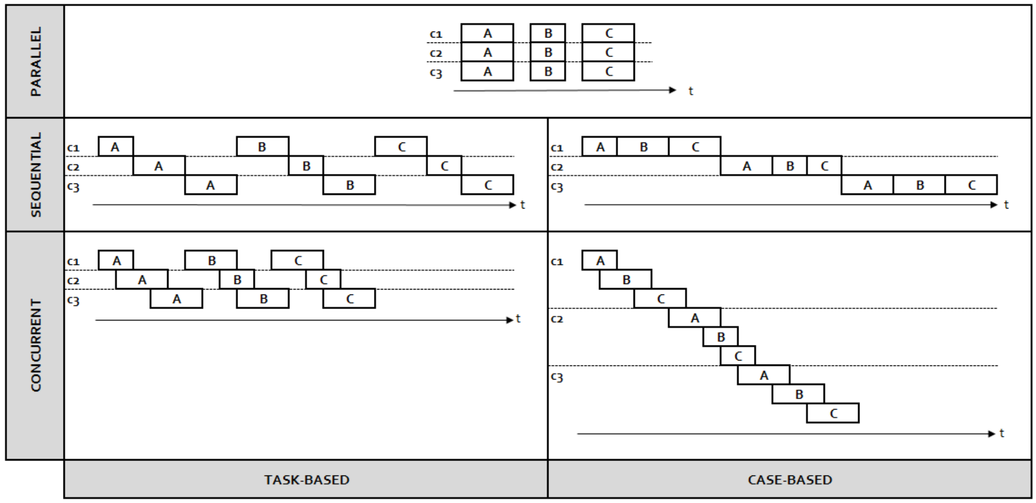


Figure 4: Batch processing types

- **Parallel batching.** In parallel batching, a set of cases is processed at exactly the same time for one or multiple consecutive tasks. For each task, all cases are processed by the same resource. In Figure 4, a resource first performs task A on cases 1, 2 and 3 simultaneously, and does the same for tasks B and C. The task *'Analyze sample'* in Figure 2 (BA 2) also processes several cases in parallel. In Figure 3, BA 6 is an example of two connected activities where parallel batching prevails.

- **Sequential task-based batching.** In sequential batching, a resource handles a group of cases (almost) immediately after each other to save set-up time/costs (e.g., the time required to get familiar with a type of task). In an effort to differentiate sequential batching from regular queue handling, a set of cases can only form a sequential batch when all of them are available for the resource before it starts processing the batch. In sequential task-based batching, a task will be performed on a set of cases sequentially, after which other tasks will sequentially be performed to that same set of cases. As with parallel batching, all cases are processed by the same resource for a particular task. Figure 4 shows that task A is performed on the three cases sequentially, which is immediately followed by task B and, finally, task C. Batch activity BA 1 in Figure 2 is an example of sequential task-based batching.

- **Sequential case-based batching.** In sequential case-based batching, a resource will perform a series of tasks sequentially for a particular case, and will (almost) immediately perform this same task sequence for other cases. In Figure 4, it is illustrated that tasks A, B and C are first performed for case 1 first, followed by case 2 and case 3. Batch activity BA 3 in Figure 2 exemplifies sequential case-based batching behavior.

- **Concurrent task-based batching.** When concurrent batching prevails, there is a partial overlap in time between the execution of tasks on a case. This implies that a resource can start executing a new task on a case before the current task is finished, or that a new case can be started while the current one has not fully been processed. In concurrent task-based batching, a task is performed for all batched cases, followed by the other tasks, with the characterizing partial time overlaps being present. For each task, all cases are processed by the same resource. Concurrent task-based batching is illustrated in Figure 4, where work is organized around tasks A, B and C. In Figure 3, BA 4 provides an example of concurrent task-based batching: while the patient is filling out the registration form, the secretary can already start the registration of another patient. Similarly, while the admission documents for one patient are being printed, the secretary can already start creating the admission documents for another patient.

- **Concurrent case-based batching.** When concurrent case-based batching prevails, the batch is organized around cases, implying that the tasks in the batch activity are performed on a particular case before proceeding to the next case. As shown in Figure 4, the partial overlap in time between task instances is present both within a case and between cases. Concurrent case-based batching is also illustrated by BA 5 in Figure 3 because a nurse can start performing a blood test for a patient while this patient is completing a drug allergy form.

Four remarks need to be made with respect to the batch processing types outlined above. Firstly, no task-based and case-based variant of parallel batching is specified. Parallel batching implies that a resource can perform a specific task on a set of cases simultaneously. From this, it follows that it is, by definition, task-oriented. When parallel batching would imply that one resource performs several tasks during exactly the same timeframe, these tasks would, in fact, constitute a single task.

Secondly, the requirements regarding resource involvement differ depending on the batch processing type under consideration. For parallel batching and sequential/concurrent task-based batching, all instances of a particular task have to be executed by the same resource. The requirements are more strict for sequential/concurrent case-based batching as all batched task instances need to be executed by the same resource. For all batch processing types, it holds the resource cannot be linked to instances which are not part of a batch while the resource is processing that batch.

Thirdly, for case-based batching, the time relation between task instances associated to a particular case is dominant to distinguish between sequential and concurrent case-based batching. Consider, for example that a concurrent relationship holds for task instances for a particular case, but the cases in the batch are handled sequentially. Under these circumstances, the batch will be considered as a concurrent case-based batch.

Finally, in batch subprocesses, the parallel, sequential or concurrent relationship between batched instances might differ across tasks (for task-based batching) or across cases (for case-based batching). For instance, a task of a task-based batch subprocess can be executed in parallel and another task in a sequential way. Similarly, the task instances included in a case-based batch subprocess might be executed sequentially for one case and concurrently for another case. In such situations, the detected batch subprocess will be labeled as a hybrid task-based or case-based batch subprocess. Hybrid batching is not included as an autonomous batch processing type as it merely consists of a combination of different types of batching behavior within a batch subprocess.

3.3. Event log

An event log is the data source that BAMA will use. It contains process execution information originating from a PAIS [2]. An event log is composed of a series of events reflecting, e.g., the start or completion of an activity. An excerpt of the event log from the laboratory process shown Figure 2 is represented in Table 1. The first line indicates that lab assistant Zoe started the task *'Register sample'* for sample 9845 on January, 14th at 11:22:33. She completed this task at 11:26:04, as shown in the second line.

The example provided in Table 1 outlines the minimal event log requirements to apply the algorithm presented in this paper. The event log should

Table 1: Illustration of event log structure (running example 1)

case id	timestamp	task	transaction type	resource
...	...	...	...	...
9845	14/01/2019 11:22:33	Register sample	start	Lab assistant Zoe
9845	14/01/2019 11:26:04	Register sample	complete	Lab assistant Zoe
9852	14/01/2019 11:26:04	Register sample	start	Lab assistant Zoe
9852	14/01/2019 11:30:21	Register sample	complete	Lab assistant Zoe
9845	14/01/2019 11:30:21	Prepare sample	start	Lab assistant Nick
9893	14/01/2019 11:36:17	Receive sample	start	Secretary Sarah
9845	14/01/2019 11:37:58	Prepare sample	complete	Lab assistant Nick
9852	14/01/2019 11:37:58	Prepare sample	start	Lab assistant Nick
9893	14/01/2019 11:38:12	Receive sample	complete	Secretary Sarah
9852	14/01/2019 11:46:11	Prepare sample	complete	Lab assistant Nick
...	...	...	...	...

be an ordered set of events related to a particular case and task. Moreover, the timestamp, resource and transaction type needs to be recorded for each event. Two transaction types are assumed to be registered: the start and the completion of a task, both transitions of the XES-lifecycle extension [35]. Each start event should have a corresponding complete event with the same resource being associated to both events.

Organizations usually have multiple resources being able to execute the same task. Hence, resource information is needed to identify which resource has worked on a group of of cases for this task. If resource information is absent and multiple resources perform the same task for different cases, this is parallel processing of cases and not batching behavior. The start and end time of a task are relevant for identifying the types of batch behavior. With only one timestamp for a task, parallel, sequential and concurrent behavior are not distinguishable.

The aforementioned event log requirements will be formalized below. Throughout the paper, we generally use upper case letters for sets and lower case letters for elements of sets. So, if C denotes the set of case identifiers, we use $c \in C$ for a specific case identifier. Table 2 summarizes the key mathematical notation which will be used in the paper.

Definition 1 (Event log requirements)**.** *Let C be a set of case identifiers. Let L be a set of task labels. Let R be a set of resource identifiers. An event log E contains a set of events e. Each event e is represented as tuple of attributes $e = (c, t, r, \tau, \varphi)$ with:*

- *$c \in C$ represents the case identifier,*

Table 2: Summary of mathematical notation

Symbol	Description
E/e	Set of events / event
C/c	Set of case identifiers / case
L/l	Set (element) of task labels / task
R/r	Set (element) of resource identifiers / resource
$\mathbb{R}^+$	Set of positive real numbers
τ	Timestamp attribute of an event
φ	Transaction type
$\#_a(e)$	Function returning the value of attribute a of event e
E_{start}	Set of events with transaction type *start*
$E_{complete}$	Set of events with transaction type *complete*
T	Set of task instance (Task log)
T_{ex}	Set of extended task instances (Extended task log)
$\tau_{arrival}$	Timestamp attribute of the task instance's arrival
τ_{start}	Timestamp attribute of the task instance's start
$\tau_{complete}$	Timestamp attribute of the task instance's completion
χ	Set of task labels
T_χ	Subset of an extended task log having task labels χ
ϕ	Set of case identifiers
T_ϕ	Subset of an extended task log having case identifiers ϕ
T_{be}	Set of batch-enriched task instances (Batch-enriched task log)

- *$l \in L$ is the task label,*
- *$r \in R$ represents the resource identifier,*
- *$\tau \in \mathbb{R}^+$ reflects the timestamp, and*
- *$\varphi \in \{start, complete\}$ refers to the transaction type.*

We use the notation $\#_a(e)$ to access the value for attribute a for event e.

Moreover, every start event should have an accompanying complete event. In formal terms, let $E_{start} = \{e \in E \mid \#_\varphi = start\}$ and $E_{complete} = \{e \in E \mid \#_\varphi = complete\}$ be a partition of event log E. Then, there exists a bijective function $\omega : E_{start} \rightarrow E_{complete}$, such that $\forall e \in E_{start} : \#_c(e) = \#_c(\omega(e)) \wedge \#_l(e) = \#_l(\omega(e)) \wedge \#_r(e) = \#_r(\omega(e)) \wedge \#_\tau(e) \leq \#_\tau(\omega(e))$.

4. Batch Activity Mining Algorithm

This section outlines the Batch Activity Mining Algorithm (BAMA), which automatically identifies batching behavior in an event log, both at the task and the subprocess level. Section 4.1 provides a general overview of the algorithm, after which its key steps are outlined in more detail in Sections 4.2- 4.6. The prototypical implementation is briefly discussed in

Section 4.7. Section 4.8 provides some pointers on how the output of BAMA can be used to gain a rich understanding in batching behavior.

4.1. General overview

BAMA aims to identify batching behavior at two distinct levels: (i) at the level of individual tasks and (ii) at the level of subprocesses. As shown in Figure 5, an event log is used as an input. This event log is converted to a task log by mapping start events to their corresponding completion events. The task log is used for batch identification purposes.

To achieve BAMA's goals, three phases can be distinguished in the algorithm. In the first phase, the task log is extended with batch formation insights at the task-resource level, i.e. batches consisting of task instances of a particular task executed by a particular resource. For instance: when several blood samples are analyzed by *'Device KN93'* simultaneously, a parallel batch is formed at the task-resource level for task-resource combination *'Analyze sample'* - *'Device KN93'*. For this phase of the algorithm, the algorithm presented in Martin et al. [6] is used.

While the first phase of the algorithm focuses on the identification of batching behavior at the level of individual tasks, the other two phases target the discovery of batch subprocesses. To this end, the extended task log is split into two subsets using the batch insights at the task-resource level: subset 1 containing candidates for a parallel or task-based sequential/concurrent batch subprocess, and subset 2 with candidates for a case-based sequential/concurrent batch subprocess. The former is used as input for batch subprocess discovery in the second phase of the algorithm and the latter in the third phase.

Merging the outcomes of the second and third phase generates a batch-enriched task log. Compared to the original task log, a batch-enriched task log contains four additional columns providing batching information: two columns highlighting whether the task instance is part of a batch at the task-resource level (one column carrying a batch identifier and another one containing the batch type), and two columns reflecting its membership of a batch subprocess (again, a batch identifier column and batch type column are added). These columns enable analysts to retrieve rich insights in batching behavior starting from real-life process execution data.

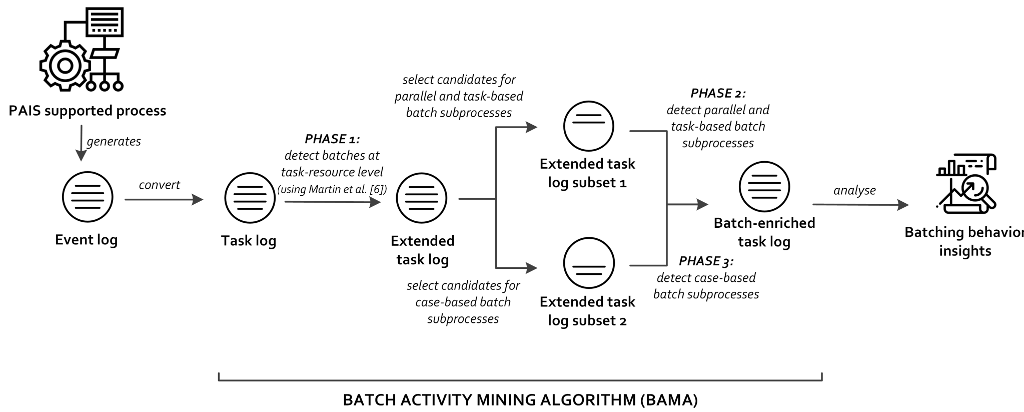


Figure 5: Overview of the Batch Activity Mining Algorithm (BAMA)

4.2. Convert to task log

The input of the algorithm is an event log, of which the minimal requirements were outlined in Section 3.3. As the detection of batching behavior requires studying the time relationship between task instances, the event log needs to be converted to a task log. In a task log, each row represents a task instance, i.e. the execution of a particular task by a particular resource in a particular case.

To convert the event log to a task log, each start event is mapped to its corresponding completion event, where the latter refers to the complete event which is linked to the same case, activity and resource. For instance: the first and second line in Table 1 will occur as a single line in the task log. When multiple start and complete events are recorded for a particular task-resource-case combination, the first occurring start event will iteratively be mapped to the first occurring unmapped completion event. For more complex mappings, techniques such as the one described by Baier et al. [36] can be used. The task log created from the event log excerpt in Table 1 is shown in Table 3.

Definition 2 (Task log)**.** *Let E represent an event log and let T be a task log. To convert E into T, a mapping function m_1 is applied. Function $m_1 : E \rightarrow T$ maps the events in E to task instances in task log T. This is achieved by mapping start events $e_s \in E_{start}$ to their corresponding complete events $\omega(e_s) \in E_{complete}$ to combine start and completion times. Consequently, T consists of a set of task instances t. Each task instance t is represented as*

Table 3: Illustration of task log structure (running example 1)

case id	**task**	**resource**	τ_{start}	$\tau_{complete}$
...	...	...	...	...
9845	Register sample	Lab assistant Zoe	14/01/2019 11:22:33	14/01/2019 11:26:04
9852	Register sample	Lab assistant Zoe	14/01/2019 11:26:04	14/01/2019 11:30:21
9845	Prepare sample	Lab assistant Nick	14/01/2019 11:30:21	14/01/2019 11:37:58
9893	Receive sample	Secretary Sarah	14/01/2019 11:36:17	14/01/2019 11:38:12
9852	Prepare sample	Lab assistant Nick	14/01/2019 11:37:58	14/01/2019 11:46:11
...	...	...	...	...

a tuple of attributes $t = (c, l, r, \tau_{start}, \tau_{complete})$*, where* τ_{start} *refers to the start time, and* $\tau_{complete}$ *is the completion time.*

4.3. Phase 1: detect batches at the task-resource level

The algorithm's first phase focuses on the detection of batching behavior at the task-resource level, i.e. on identifying groups of task instances which are processed as a batch at a particular task by a particular resource. This task-resource level is purposefully selected as it conveys valuable insights in the way in which a specific resource executes a task.

Besides the useful information it offers in itself, batching behavior at the task-resource level is also a prerequisite for the presence of some types of batch subprocesses. From the outline of the batch processing types in Section 3.2, it follows that parallel and task-based sequential/concurrent batch subprocesses require that batching behavior is detected at the task-resource level. To this end, the first phase of BAMA involves detecting parallel (par), sequential (seq) or concurrent (conc) batches at the task-resource level.

To detect batches at the task-resource level, the algorithm presented in Martin et al. [6] is used. This algorithm will group task instances for which a batching relationship holds by marking these instances with the same batch number. The parallel, sequential or concurrent character of the batch is also added.

As follows from the description of sequential batching in Section 3.2, the arrival time of a case at a task is used to distinguish between sequential batching and regular queue handling. Consequently, each task instance in the task log needs to be enriched with the arrival time (i.e. the time at which a task instance is enabled). The arrival time differs from the start time of a task instance when queues are formed because of limited resources to perform a particular task. In a Q-log, which is an event log containing queue-related

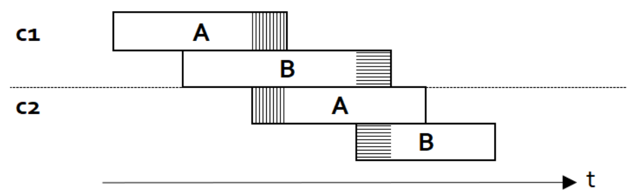


Figure 6: Illustration of concurrent batching at the task-resource level

events such as queue arrival events [29], arrival times are explicitly recorded in the event log. Hence, they can just be included when the event log is converted to a task log and no further efforts are required. When, instead, arrival times are unknown, they can be imputed using a suitable heuristic. For example, the task arrival time of a case can be approximated by the end of its preceding task. This requires insights in the process control-flow, which can be gathered from the event log using an existing control-flow discovery algorithm. As control-flow discovery is beyond the scope of this paper, the reader is referred to, e.g., van der Aalst [2], De Weerdt et al. [37], and Augusto et al. [38] for more background on this topic.

When batches are detected at the task-resource level, these task instances are candidates to be part of a parallel or task-based sequential/concurrent batch subprocess (cf. phase 2 of the algorithm). Conversely, task instances which are not batched at the task-resource level are candidates to be part of a case-based sequential/concurrent batch subprocess (cf. phase 3 of the algorithm). The latter follows from the observation that case-based batching does not require batching behavior at the task-resource level, as shown at the right side of Figure 4. However, detected concurrent and sequential batches at the task-resource level are also included as candidates for case-based batch subprocesses. To illustrate why this is the case, consider the illustrations of a concurrent case-based batch subprocess in Figure 6 and Figure 7 (where all instances are performed by the same resource). Due to the strong time overlap between the instances, the two instances of task A form a concurrent batch at the task-resource level in Figure 6 and a sequential batch in Figure 7. A similar remark holds for the two instances of task B. These simple illustrations show why concurrent and sequential batches at the task-resource level are also candidates to be part of a case-based batch subprocess.

From what said, it follows that the first phase of the algorithm involves extending the task log with three columns: the task arrival time, a task-

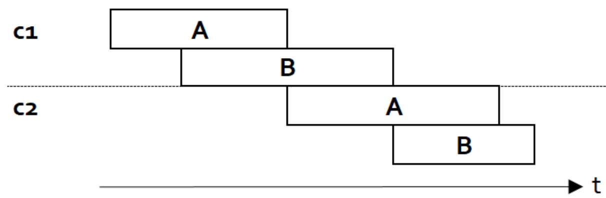


Figure 7: Illustration of sequential batching at the task-resource level

resource batch identifier and a task-resource batch type. When extending the task log in Table 3, the log in Table 4 is obtained. Table 4 shows that two sequential batches are detected at the task-resource level: batch 138 containing two instances of task *'Register sample'* and batch 374 consisting of two instances of the task *'Prepare sample'*.

Definition 3 (Extended task log)**.** *Let T represent a task log and let T_{ex} be an extended task log. To convert T into T_{ex}, two functions m_2 and m_3 are sequentially applied:*

- *$m_2 : T \rightarrow T'$ adds the arrival time $\tau_{arrival}$ for each task instance. In case the arrival times are recorded in the event log, m_2 can be merged with m_1 in Definition 2.*

- *$m_3 : T' \rightarrow T_{ex}$ assigns a task-resource batch identifier $b_{tr,id}$ and a task-resource batch type $b_{tr,t}$ to all task instances $t \in T'$ with $b_{tr,id} \in \mathbb{N}$ and $b_{tr,t} \in \{par, seq, conc\}$. The values of $b_{tr,id}$ and $b_{tr,t}$ are shared among the task instances t which are part of the same batch at the task-resource level* [1]*.*

After the application of m_2 and m_3, the extended task log T_{ex} consists of a set of batched task instances t_b. Each batched task instance is represented as a tuple of attributes $t_b = (c, l, r, \tau_{arrival}, \tau_{start}, \tau_{complete}, b_{tr,id}, b_{tr,t})$.

4.4. Phase 2: detect parallel and task-based sequential/concurrent batch subprocesses

Using the task instances which are batched at the task-resource level as an input (cf. subset 1 in Figure 5), the second phase involves detecting parallel and task-based sequential/concurrent batch subprocesses.

[1] The application of function m_3 involves the use of the method defined by Martin et al. [6] to identify batches at the task-resource level.

Table 4: Illustration of extended task log structure (running example 1)

case id	**task**	**resource**	$\tau_{arrival}$	τ_{start}	$\tau_{complete}$	**task-resource batch id**	**task-resource batch type**
...	...	...	...	...	...	...	...
9845	Register sample	Lab assistant Zoe	14/01/2019 10:17:38	14/01/2019 11:22:33	14/01/2019 11:26:04	138	seq
9852	Register sample	Lab assistant Zoe	14/01/2019 10:32:44	14/01/2019 11:26:04	14/01/2019 11:30:21	138	seq
9845	Prepare sample	Lab assistant Nick	14/01/2019 11:26:04	14/01/2019 11:30:21	14/01/2019 11:37:58	374	seq
9893	Receive sample	Secretary Sarah	14/01/2019 11:30:21	14/01/2019 11:36:17	14/01/2019 11:38:12	-	-
9852	Prepare sample	Lab assistant Nick	14/01/2019 11:14:17	14/01/2019 11:37:58	14/01/2019 11:46:11	374	seq
...	...	...	...	...	...	...	...

To identify these subprocesses, two conditions need to be verified. Firstly, it is checked whether there are different batches at the task-resource level having exactly the same composition in terms of cases. For instance, in Table 4, the detected batches for *'Register sample'* - *'Lab assistant Zoe'* and *'Prepare sample'* - *'Lab assistant Nick'* contain the same cases 9845 and 9852. Secondly, when such batches are present, it is verified whether, for each case, these tasks immediately follow each other (i.e. that no other tasks have been executed in between). When both conditions are fulfilled, a batch subprocess has been detected. All task instances included in the subprocess will be marked with the same batch subprocess identifier.

To determine whether the detected batch subprocess is a parallel or task-based sequential/concurrent batch subprocess, the batch types at the task-resource level are taken into consideration. When all tasks included in a batch subprocess are executed as a parallel batch, the batch subprocess type will be parallel. Similarly, when all included tasks are executed as a sequential or concurrent batch, the batch subprocess type will be sequential and concurrent, respectively. In case a combination of batch types is present at the task-resource level, e.g. one task is executed in parallel and another one sequential, the batch subprocess will be referred to as a hybrid batch subprocess.

With respect to sequential batching at the task-resource level, Martin et al. [6] allow for a time gap between the end of a particular task instance and the start of the next one. A non-zero time gap can accommodate, e.g., for some set-up time required between cases in practical applications [6]. In the detection of sequential task-based subprocesses, a similar time gap can be defined. The time gap expresses the maximal tolerable time period that can elapse between the end of the last task instance of a particular task in the subprocess and the start of the first instance at the next task. Note that the requirement that no other tasks can be executed in this time period still holds. An appropriate value for the time gap is context-specific and, hence, requires domain knowledge. The event log can support domain experts by providing insights in the time gaps prevailing in reality.

We now formally define the different batch subprocess types. First, requirements common for all task-based batch subprocess types are identified. Then, we define parallel batching and sequential/concurrent task-based batching. To clarify the definitions, an illustration for each task-based batching type is provided, based on the running examples introduced earlier. For each batching type, both its footprint in the batch-enriched task log and a visualization is included.

Definition 4 (Base task-based batch conditions)**.** *Let $\chi \subseteq L$ represent a set of task labels. The base batch conditions common to all task-based batch types that a set of task instances $T_\chi \subseteq T_{ex}$ needs to fulfill are:*

1. *all task instances refer to task labels in χ, i.e., $\forall_{t \in T_\chi} \#_l(t) \in \chi$;*
2. *the same number*[2] *and at least two instances for each task label are recorded, i.e.:*

$$\forall_{l_1,l_2 \in \chi}(\,|\{t \in T_\chi \mid \#_l(t) = l_1\}| = |\{t \in T_\chi \mid \#_l(t) = l_2\}| \geq 2);$$

3. *given $1 \leq k \leq n-1$ and the sequence of task instances $\langle t_1, \dots, t_k, \dots, t_n \rangle \in T_\chi^*$ with $\#_{\tau_{start}}(t_k) \leq \#_{\tau_{start}}(t_{k+1})$ and $t_k \neq t_{k+1}$ all task instances with*

[2] Note that we simplify the definition by restricting subprocesses to not repeat individual tasks. However, this limitation can be lifted by a suitable renaming of the task labels such that each repetition is uniquely labeled.

the same task label are started before proceeding to task instances with another task label:

$$\#_l(t_k) \neq \#_l(t_{k+1}) \implies \forall_{k+1 \leq j \leq n}(\#_l(t_j) \neq \#_l(t_k))$$

4. *the same set of cases is involved in the batch, i.e.:*

$$\forall_{l_1,l_2 \in \chi}(\{c \in C \mid \exists t \in T_\chi : \#_c(t) = c \wedge \#_l(t) = l_1\} \\ = \{c \in C \mid \exists t \in T_\chi : \#_c(t) = c \wedge \#_l(t) = l_2\});$$

5. *the same resource executes tasks with the same task label in the full batch, i.e.,*

$$\forall_{t_a,t_b \in T_\chi}(\#_l(t_a) = \#_l(t_b) \implies \#_r(t_a) = \#_r(t_b)).$$

Definition 5 (Parallel batch)**.** *Let $\chi \subseteq L$ represent a set of task labels. A parallel batch is composed of a set of task instances $T_\chi \subseteq T_{ex}$ fulfilling the base task-based batch conditions (cf. Definition 4) and for all task instances $t_a, t_b \in T_\chi$ the following additional conditions hold:*

1. *tasks with the same label are performed in parallel:*

$$\#_l(t_a) = \#_l(t_b) \iff \#_{\tau_{start}}(t_a) = \#_{\tau_{start}}(t_b) \\ \wedge \#_{\tau_{complete}}(t_a) = \#_{\tau_{complete}}(t_b);$$

2. *tasks with different labels are non-overlapping:*

$$\#_l(t_a) \neq \#_l(t_b) \iff \#_{\tau_{complete}}(t_a) < \#_{\tau_{start}}(t_b));$$

3. *no other task instance performed by the same resource, which is not included in the parallel batch, can be started or completed when processing the batched instances, i.e., given:*

$$T_r = \{t \in (T_{ex} \setminus T_\chi) \mid \exists_{t_a,t_b \in T_\chi}\big(\#_r(t_a) = \#_r(t) = \#_r(t_b) \\ \wedge (\#_{\tau_{start}}(t_a) \leq \#_{\tau_{start}}(t) \leq \#_{\tau_{complete}}(t_b) \\ \vee \#_{\tau_{start}}(t_a) \leq \#_{\tau_{complete}}(t) \leq \#_{\tau_{complete}}(t_b))\big)\}$$

we require $T_r = \emptyset$.

An illustration of a parallel batch subprocess within the context of running example 2 is provided in Table 5 and visualized in Figure 8. The parallel batch subprocess consists of cases 9072 and 9080. First, these cases are processed in a parallel batch at task *'Preprocess blood sample'*. Afterwards, task *'Send blood sample'* is executed in these same cases as a parallel batch. As *'Preprocess blood sample'* and *'Send blood sample'* immediately follow each other for both cases which indicates that a parallel batch subprocess is detected.

Definition 6 (Sequential task-based batch)**.** *Let $\chi \subseteq L$ represent a set of task labels. A sequential task-based batch is composed of a set of task instances $T_\chi \subseteq T_{ex}$ fulfilling the base task-based batch conditions (cf. Definition 4) and additionally the following conditions:*

1. *for all $t \in T_\chi$ the arrival time of the instance cannot be later than the start time of the first instance for that particular task in the batch, i.e., for all $t_x \in T_\chi : (\#_l(t_x) = \#_l(t)) \implies \#_{\tau_{arrival}}(t) \leq \#_{\tau_{start}}(t_x)$;*
2. *for $1 \leq k \leq n-1$ and the sequence[3] of task instances $\langle t_1, \ldots, t_k, \ldots, t_n \rangle \in T_\chi^*$ with $\#_{\tau_{start}}(t_k) < \#_{\tau_{start}}(t_{k+1})$ the following conditions cumulatively hold:*
   - *task instances referring to the same label are at most separated by a time gap γ, i.e.:*

$$\#_l(t_k) = \#_l(t_{k+1}) \implies (\#_{\tau_{start}}(t_{k+1}) - \#_{\tau_{complete}}(t_k)) \in [0, \gamma] \quad \textit{with } \gamma \geq 0,$$

   - *the time gap between the completion of the last instance for a particular task in χ and the start of the first instance of the next task in χ cannot exceed the tolerated time gap θ, i.e.:*

$$\#_l(t_k) \neq \#_l(t_{k+1}) \implies (\#_{\tau_{start}}(t_{k+1}) - \#_{\tau_{complete}}(t_k)) \in [0, \theta] \quad \textit{with } \theta \geq 0.$$

[3] We denote with X^* the set of all sequences over set X. Moreover, we assume in the remainder of this paper that it is possible to obtain a totally ordered sequence of events performed by a single resource. If this is not fulfilled, the event logs can be pre-processed based on a secondary order criteria e.g., the completion time.

Table 5: Illustration of a parallel batch subprocess (running example 2)

case id	**task**	**resource**	$\tau_{arrival}$	τ_{start}	$\tau_{complete}$	**task-resource batch id**	**task-resource batch type**	**batch subprocess id**	**batch subprocess type**
...	...	...	...	...	...	...	...	...	...
9072	Preprocess blood sample	Device TB04	10/01/2019 12:28:02	10/01/2019 13:03:17	10/01/2019 13:07:52	52	par	8	par
9080	Preprocess blood sample	Device TB04	10/01/2019 12:35:02	10/01/2019 13:03:17	10/01/2019 13:07:52	52	par	8	par
...	...	...	...	...	...	...	...	...	...
9072	Send blood sample	Nurse Sue	10/01/2019 11:07:52	10/01/2019 13:13:24	10/01/2019 13:17:04	59	par	8	par
9080	Send blood sample	Nurse Sue	10/01/2019 11:07:52	10/01/2019 13:13:24	10/01/2019 13:17:04	59	par	8	par
...	...	...	...	...	...	...	...	...	...

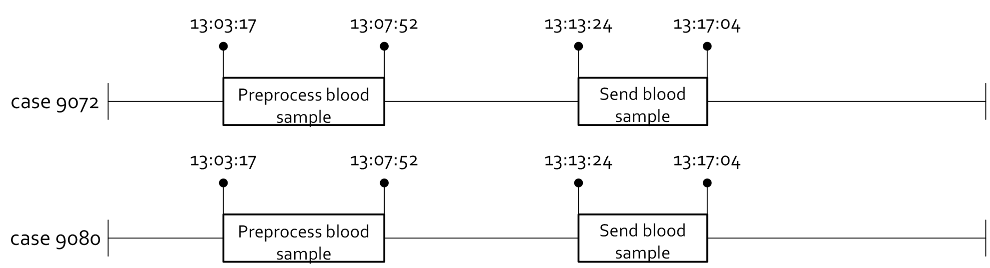


Figure 8: Visualization of a parallel batch subprocess (running example 2)

3. *no other task instance performed by the same resource, which is not included in the sequential task-based batch, can be started or completed when processing the batched instances, i.e., given:*

$$T_r = \{t \in (T_{ex} \setminus T_\chi) \mid \exists_{t_a, t_b \in T_\chi} \big(\#_r(t_a) = \#_r(t) = \#_r(t_b) \wedge (\#_{\tau_{start}}(t_a) \leq \#_{\tau_{start}}(t) \leq \#_{\tau_{complete}}(t_b) \vee \#_{\tau_{start}}(t_a) \leq \#_{\tau_{complete}}(t) \leq \#_{\tau_{complete}}(t_b))\big)\}$$

we require $T_r = \emptyset$.

As illustrated in Table 6 and visualized in Figure 9, a sequential task-based batch subprocess is detected for the task instances included in Table 4 (related to running example 1). More specifically, the batches with task-resource batch identifier 138 and 374 fulfill the conditions of a task-based sequential batch process as they both consist of the same cases 9845 and 9852, and the tasks immediately follow each other for both cases. Because both tasks are sequential batches at the task-resource level, the task-based subprocess is marked as a task-based sequential batch subprocess.

Definition 7 (Concurrent task-based batch)**.** *Let $\chi \subseteq L$ represent a set of task labels. A concurrent task-based batch is composed of a set of task instances $T_\chi \subseteq T_{ex}$ fulfilling the base task-based batch conditions (cf. Definition 4) and additionally the following conditions:*

1. *given $1 \leq k \leq n-1$ and the sequence of task instances $\langle t_1, \ldots, t_k, \ldots, t_n \rangle \in T_\chi^*$ with $\#_{\tau_{start}}(t_k) \leq \#_{\tau_{start}}(t_{k+1})$ and $t_k \neq t_{k+1}$ subsequent task instances are performed concurrently, i.e.:*

$$\begin{aligned}\#_{\tau_{start}}(t_k) \leq \#_{\tau_{start}}(t_{k+1}) < \#_{\tau_{complete}}(t_k)& \\ \wedge\big(\#_{\tau_{start}}(t_k) \neq \#_{\tau_{start}}(t_{k+1}) \vee& \\ \#_{\tau_{complete}}(t_k) \neq \#_{\tau_{complete}}(t_{k+1})\big)&\end{aligned}$$

2. *no other task instance performed by the same resource, which is not included in the concurrent task-based batch, can be started or completed when processing the batched instances, i.e., given:*

$$\begin{aligned}T_r = \{t \in (T_{ex} \setminus T_\chi) \mid \exists_{t_a, t_b \in T_\chi} \big(&\#_r(t_a) = \#_r(t) = \#_r(t_b) \\ &\wedge (\#_{\tau_{start}}(t_a) \leq \#_{\tau_{start}}(t) \leq \#_{\tau_{complete}}(t_b) \\ &\vee \#_{\tau_{start}}(t_a) \leq \#_{\tau_{complete}}(t) \leq \#_{\tau_{complete}}(t_b))\big)\}\end{aligned}$$

we require $T_r = \emptyset$.

An example of a concurrent task-based batch in the patient admission process (running example 2) is presented in Table 7 and Figure 10. At the task-resource level, the instances of both *'Complete registration form'* and *'Create admission documents'* partially overlap in time. Consequently, these instances form concurrent batches at the task-resource level (with task-resource batch identifier 36 for *'Complete registration form'* and 42 for *'Create*

Table 6: Illustration of a sequential task-based batch subprocess (running example 1)

case id	**task**	**resource**	$\tau_{arrival}$	τ_{start}	$\tau_{complete}$	**task-resource batch id**	**task-resource batch type**	**batch subprocess id**	**batch subprocess type**
...	...	...	...	...	...	...	...	...	...
9845	Register sample	Lab assistant Zoe	14/01/2019 10:17:38	14/01/2019 11:22:33	14/01/2019 11:26:04	138	seq	38	seq task-based
9852	Register sample	Lab assistant Zoe	14/01/2019 10:32:44	14/01/2019 11:26:04	14/01/2019 11:30:21	138	seq	38	seq task-based
...	...	...	...	...	...	...	...	...	...
9845	Prepare sample	Lab assistant Nick	14/01/2019 11:26:04	14/01/2019 11:30:23	14/01/2019 11:37:58	374	seq	38	seq task-based
9852	Prepare sample	Lab assistant Nick	14/01/2019 11:14:17	14/01/2019 11:37:58	14/01/2019 11:46:11	374	seq	38	seq task-based
...	...	...	...	...	...	...	...	...	...

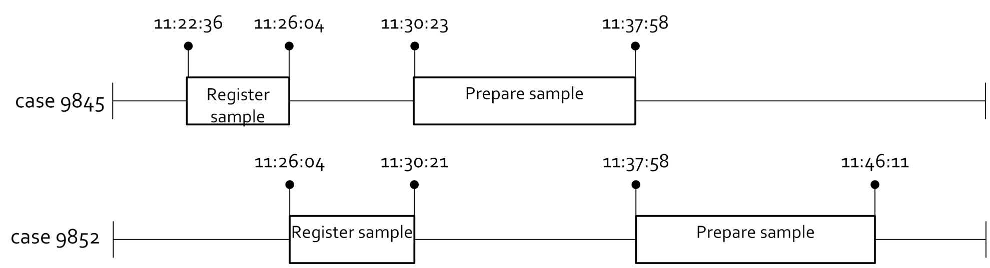


Figure 9: Visualization of a sequential task-based batch subprocess (running example 1)

admission documents'). As these tasks immediately follow each other for both cases, a concurrent task-based batch subprocess is present.

In summary, phase 2 of the algorithm can identify sets of task instances fulfilling the conditions of either parallel or task-based sequential/concurrent batch subprocesses. Two columns are added in this phase: a batch subprocess identifier and a batch subprocess type. In the output, task instances belonging to a batch subprocess will share the same batch subprocess identifier.

Table 7: Illustration of a concurrent task-based batch subprocess (running example 2)

case id	task	resource	$\tau_{arrival}$	τ_{start}	$\tau_{complete}$	task-resource batch id	task-resource batch type	batch subprocess id	batch subprocess type
...	...	...	...	...	...	...	...	...	...
9097	Complete registration form	Secretary Mark	10/01/2019 11:12:18	10/01/2019 11:12:18	10/01/2019 11:19:54	36	conc	4	conc task-based
9098	Complete registration form	Secretary Mark	10/01/2019 11:16:04	10/01/2019 11:16:04	10/01/2019 11:23:31	36	conc	4	conc task-based
9097	Create admission documents	Secretary Mark	10/01/2019 11:19:54	10/01/2019 11:23:31	10/01/2019 11:28:19	42	conc	4	conc task-based
9098	Create admission documents	Secretary Mark	10/01/2019 11:23:31	10/01/2019 11:26:48	10/01/2019 11:32:08	42	conc	4	conc task-based
...	...	...	...	...	...	...	...	...	...

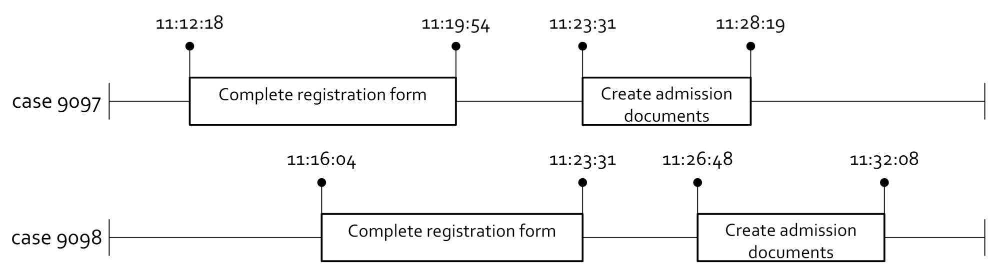


Figure 10: Visualization of a concurrent task-based batch subprocess (running example 2)

4.5. Phase 3: detect case-based sequential/concurrent batch subprocesses

The algorithm's third phase aims to detect case-based sequential or concurrent batch subprocesses. To this end, for the reasons outlined earlier, it uses the task instances which are not part of a task-resource batch or which are included in a concurrent or sequential task-resource batch (cf. subset 2 in Figure 5). Regarding the inclusion of instances in a concurrent/sequential batch at the task-resource level, it should be noted that instances included in a concurrent/sequential task-based batch subprocess are removed from subset 2 after phase 2. This avoids that task instances will be duplicated in the merged batch-enriched task log after phase 3.

To identify case-based batching behavior for a particular set of connected tasks (i.e. a task subsequence representing a potential subprocess), three steps are taken. Firstly, *within-case batching* is checked, which focuses on the presence of a batching relationship between task instances of one particular case. To illustrate this, consider Table 8, where the subsequence *'Study summary results'* - *'Prepare report'* is considered. As shown in Figure 4, sequential/concurrent case-based batching requires that a batching relationship is present between the execution of different tasks in a particular case. This holds for case 9969 in Table 8, where a sequential relationship holds between the execution of *'Study summary results'* and *'Prepare report'*. The same holds for case 9974.

Secondly, when within-case batching is detected, the involved task instances related to a particular case are replaced by a *single aggregated instance*. This is a preparatory step for the third step of case-based subprocess detection. The arrival and start timestamp of this aggregated instance correspond to the earliest arrival and start timestamp of the included instances. For the complete timestamp, the aggregated instance will take the latest complete timestamp of the included instances. For example: the aggregated task for case 9969 will have 13:45:17 as arrival time, 15:22:47 as start time and 15:57:54 as completion time. For case 9974, the arrival, start and completion time correspond to 13:58:17, 15:57:54 and 16:28:57, respectively.

Finally, the third and last step uses the aggregated instances to determine whether *between-case batching* is present. Between-case batching determines whether a batching relationship also holds between the aggregated instances created in the second step (representing batches at the level of specific cases). To this end, similar sequential/concurrent time relationships are detected as the ones used for batch detection at the task-resource level. When between-case batching is also detected, a case-based batch subprocess is identified. In Table 8, a sequential relationship holds for the aggregated instances of cases 9969 and 9974. Hence, a case-based sequential batch subprocess is found, which is highlighted by adding a shared batch subprocess identifier. The batch subprocess is also visualized in Figure 11.

The aforementioned three steps make it possible to determine whether case-based batching prevails for one particular task subsequence. However, a multitude of task subsequences can be present in the input file for phase

Table 8: Illustration of a sequential case-based batch subprocess (running example 1)

case id	task	resource	$\tau_{arrival}$	τ_{start}	$\tau_{complete}$	task-resource batch id	task-resource batch type	batch subprocess id	batch subprocess type
...	...	...	...	...	...	...	...	...	...
9969	Study summary results	Lab technician June	14/01/2019 13:45:17	14/01/2019 15:22:47	14/01/2019 15:41:08	-	-	61	seq case-based
9969	Prepare report	Lab technician June	14/01/2019 15:41:08	14/01/2019 15:41:08	14/01/2019 15:57:54	-	-	61	seq case-based
9974	Study summary results	Lab technician June	14/01/2019 13:58:17	14/01/2019 15:57:54	14/01/2019 16:11:12	-	-	61	seq case-based
9974	Prepare report	Lab technician June	14/01/2019 16:11:12	14/01/2019 16:11:12	14/01/2019 16:28:57	-	-	61	seq case-based
...	...	...	...	...	...	...	...	...	...

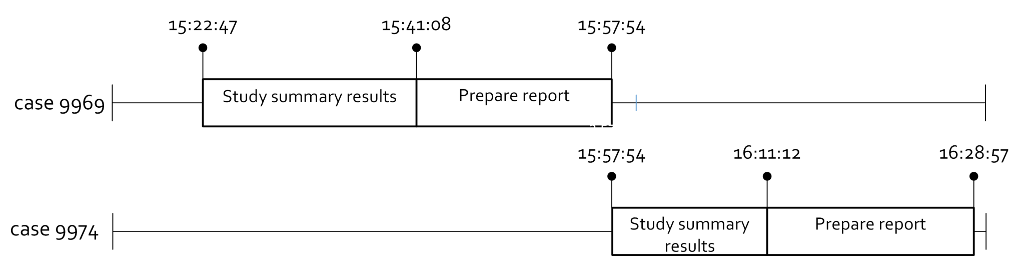


Figure 11: Visualization of a sequential case-based batch subprocess (running example 1)

3. Without loss of generality, two methods to identify candidate task subsequences are proposed, which are (1) enumeration and (2) sequence mining. For each of these candidates, case-based batch detection will be performed following the three steps outlined above.

When *enumeration* is used, all potential task subsequences prevailing in the input subset for BAMA's phase 3 are considered. This involves subsequences of all orders (i.e. number of tasks), going from two tasks to the length of a trace included in the input subset. To avoid that very rare subsequences also need to be checked for batching behavior, a threshold can be specified expressing the minimal occurrence frequency of a subsequence. Moreover, a filtering mechanism is integrated to avoid that higher-order subsequences are unnecessarily checked. Higher-order subsequences containing a particular lower-order subsequence will not be checked when within-case batching is absent for this lower-order subsequence. The filtering mechanism builds upon the idea that within-case batching is a prerequisite for the detection of a case-based batch subprocess. Suppose, for example, that no within-case batching is detected for a task subsequence A - B. This implies that, for none of the cases in the input log, a batching relation exists for the case's instances related to tasks A and B. Hence, a case-based batch containing instances of A and B will not exist. Consequently, higher-order subsequences containing A - B such as A - B - C should not be checked as the prerequisite for case-based batching is not fulfilled for A - B.

Besides enumeration, existing *sequence mining* methods can also be used to identify candidate case-based subprocesses. In the implementation, the SPADE algorithm introduced by Zaki [39] has been incorporated. SPADE looks for frequent tasks sequences taking into account a user-defined minimum support level. The implementation of BAMA hard-codes a SPADE parameter to ensure that only tasks that immediately follow each other are taken into consideration.

We now formally define the different batch subprocess types for case-based batching. First, requirements common for all case-based batch subprocess types are identified. Then, we define sequential, and concurrent case-based batching. Illustrations are provided to clarify the definitions, consisting of an example footprint in the batch-enriched task log and a visualization.

Definition 8 (Base case-based batch conditions)**.** *Let $\psi \subseteq C$ represent a set of case identifiers. The base batch conditions common to all case-based batch types that a set of task instances $T_\psi \subseteq T_{ex}$ needs to fulfill are:*

1. *all task instances refer to case identifiers in ψ, i.e., $\forall_{t \in T_\psi} \#_c(t) \in \psi$;*
2. *given $1 \leq k \leq n-1$ and the sequence of task instances $\langle t_1, \ldots, t_k, \ldots, t_n \rangle \in T_\psi^*$ with $\#_{\tau_{start}}(t_k) \leq \#_{\tau_{start}}(t_{k+1})$ and $t_k \neq t_{k+1}$ all task instances with the same case identifier are started before proceeding to task instances with another case identifier:*

$$\#_c(t_k) \neq \#_c(t_{k+1}) \implies \forall_{k+1 \leq j \leq n}(\#_c(t_j) \neq \#_c(t_k))$$

3. *the same number and at least two instances for each case identifier are recorded:*

$$\forall_{c_1,c_2 \in \psi}(|\{t \in T_\psi \mid \#_c(t) = c_1\}| = |\{t \in T_\psi \mid \#_c(t) = c_2\}| \geq 2);$$

4. *the same set of task labels is involved in the batch:*

$$\begin{aligned}\forall_{c_1,c_2 \in \psi}(\{l \in L \mid \exists t \in T_\psi : \#_l(t) = l \wedge \#_c(t) = c_1\}\\ = \{l \in L \mid \exists t \in T_\psi : \#_l(t) = l \wedge \#_c(t) = c_2\});\end{aligned}$$

5. *the same resource executed the full batch, i.e.,* $\forall_{t_a,t_b \in T_\psi}(\#_r(t_a) = \#_r(t_b))$.

Definition 9 (Sequential case-based batch)**.** *Let $\psi \subseteq C$ represent a set of case identifiers. A sequential case-based batch is composed of a set of task instances $T_\psi \subseteq T_{ex}$ fulfilling the base case-based batch conditions (cf. Definition 8) and additionally the following conditions:*

1. *for $1 \leq k \leq n-1$ and the sequence of task instances $\langle t_1, \dots, t_k, \dots, t_n \rangle \in T_\psi^*$ with $\#_{\tau_{start}}(t_k) < \#_{\tau_{start}}(t_{k+1})$ the following conditions cumulatively hold:*
   - *task instances referring to the same case identifier are at most separated by a time gap γ:*

     $$\begin{aligned}\#_c(t_k) = \#_c(t_{k+1}) \implies (\#_{\tau_{start}}(t_{k+1}) - \#_{\tau_{complete}}(t_k)) \in [0, \gamma]\\ \textit{with } \gamma \geq 0,\end{aligned}$$

   - *the time gap between the completion of the last instance for a particular case identifier in ψ and the start of the first instance of the next case identifier in ψ cannot exceed the tolerated time gap θ:*

     $$\begin{aligned}\#_c(t_k) \neq \#_c(t_{k+1}) \implies (\#_{\tau_{start}}(t_{k+1}) - \#_{\tau_{complete}}(t_k)) \in [0, \theta]\\ \textit{with } \theta \geq 0.\end{aligned}$$

2. *no other task instance performed by the same resource, which is not included in the sequential case-based batch, can be started or completed when processing the batched instances:*

   $$\begin{aligned}T_r = \{t \in (T_{ex} \setminus T_\psi) \mid \exists_{t_a,t_b \in T_\psi}\big(&\#_r(t_a) = \#_r(t) = \#_r(t_b)\\ &\wedge (\#_{\tau_{start}}(t_a) \leq \#_{\tau_{start}}(t) \leq \#_{\tau_{complete}}(t_b)\\ &\vee \#_{\tau_{start}}(t_a) \leq \#_{\tau_{complete}}(t) \leq \#_{\tau_{complete}}(t_b))\big)\}\end{aligned}$$

   we require $T_r = \emptyset$.

An illustration of a sequential case-based batch was already provided in Table 8 and Figure 11 within the context of running example 1. Tasks *'Study summary results'* and *'Prepare report'* are performed first for case 9969 in a sequential way, and afterwards for case 9974. Moreover, the relationship between the task instances of cases 9969 and 9974 also has a sequential character. Consequently, a sequential case-based batch is detected.

Definition 10 (Concurrent case-based batch)**.** *Let $\psi \subseteq C$ represent a set of case identifiers. A concurrent case-based batch is composed of a set of task instances $T_\psi \subseteq T_{ex}$ fulfilling the base case-based batch conditions (cf. Definition 8) and additionally the following conditions:*

1. *given $1 \leq k \leq n-1$ and the sequence of task instances $\langle t_1, \ldots, t_k, \ldots, t_n \rangle \in T_\psi^*$ with $\#_{\tau_{start}}(t_k) \leq \#_{\tau_{start}}(t_{k+1})$ and $t_k \neq t_{k+1}$ subsequent task instances are performed concurrently, but not perfectly in parallel:*

$$\begin{aligned} \#_{\tau_{start}}(t_k) \leq \#_{\tau_{start}}(t_{k+1}) < \#_{\tau_{complete}}(t_k) \\ \wedge \big(\#_{\tau_{start}}(t_k) \neq \#_{\tau_{start}}(t_{k+1}) \vee \\ \#_{\tau_{complete}}(t_k) \neq \#_{\tau_{complete}}(t_{k+1})\big) \end{aligned}$$

2. *no other task instance performed by the same resource, which is not included in the sequential case-based batch, can be started or completed when processing the batched instances;*

$$\begin{aligned} T_r = \{t \in (T_{ex} \setminus T_\psi) \mid \exists_{t_a, t_b \in T_\psi} \big(\#_r(t_a) = \#_r(t) = \#_r(t_b) \\ \wedge\ (\#_{\tau_{start}}(t_a) \leq \#_{\tau_{start}}(t) \leq \#_{\tau_{complete}}(t_b) \\ \vee\ \#_{\tau_{start}}(t_a) \leq \#_{\tau_{complete}}(t) \leq \#_{\tau_{complete}}(t_b))\big)\} \end{aligned}$$

we require $T_r = \emptyset$.

To illustrate this last type of batch activity, consider Table 9 and Figure 12, which are derived from running example 2. For case 9123, tasks *'Complete drug allergy form* and *'Perform blood test'* are performed concurrently. The same holds for case 9124. When aggregating both instances at the case level, a concurrent relationship even exists between the cases due to the time overlap between *'Perform blood test'* for case 9123 and *'Complete drug allergy form'* for case 9124. Hence, a concurrent case-based subprocess is detected.

Table 9: Illustration of a concurrent case-based batch subprocess (running example 2)

case id	**task**	**resource**	$\tau_{arrival}$	τ_{start}	$\tau_{complete}$	**task-resource batch id**	**task-resource batch type**	**batch subprocess id**	**batch subprocess type**
...	...	...	...	...	...	...	...	...	...
9123	Complete drug allergy form	Nurse Kate	10/01/2019 15:02:41	10/01/2019 15:21:18	10/01/2019 15:33:09	-	-	21	conc case-based
9123	Perform blood test	Nurse Kate	10/01/2019 15:02:41	10/01/2019 15:31:30	10/01/2019 15:37:14	-	-	21	conc case-based
...	...	...	...	...	...	...	...	...	...
9124	Complete drug allergy form	Nurse Kate	10/01/2019 15:26:01	10/01/2019 15:35:24	10/01/2019 15:44:55	-	-	21	conc case-based
9124	Perform blood test	Nurse Kate	10/01/2019 15:26:01	10/01/2019 15:41:32	10/01/2019 15:48:09	-	-	21	conc case-based
...	...	...	...	...	...	...	...	...	...

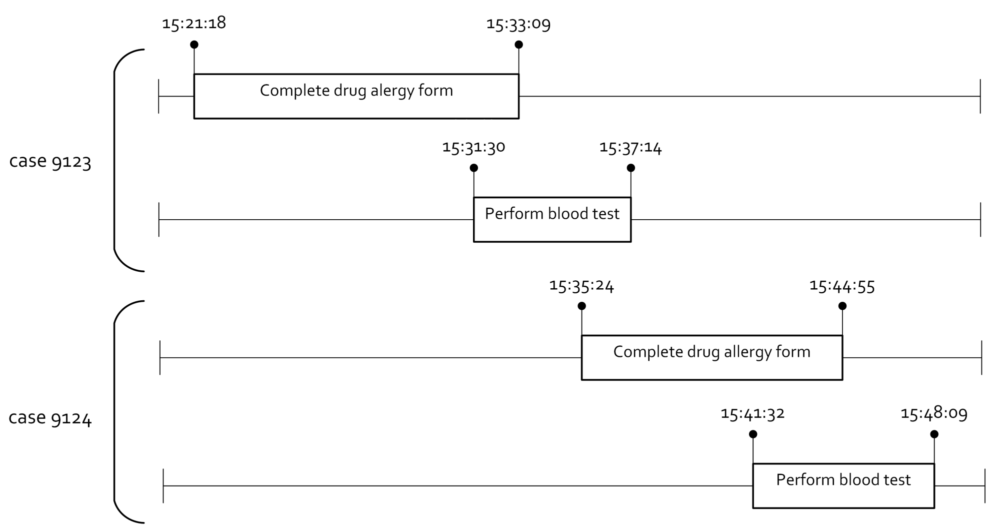


Figure 12: Visualization of a concurrent case-based batch subprocess (running example 2)

In summary, phase 3 of the algorithm will group task instances which fulfill the requirements of a case-based sequential/concurrent batch subprocess. This will be expressed in the output by assigning a shared batch subprocess identifier to these instances. As a consequence, phase 3 will not add new columns, but will complete the columns added in phase 2 (batch subprocess identifier and batch subprocess type) with information on detected case-based batch subprocesses.

4.6. Batch-enriched task log creation

The output of BAMA is a batch-enriched task log, having a structure as exemplified in Tables 5-9. It is obtained by merging the outputs of phases 2 and 3 of the algorithm. Compared to the task log, the batch-enriched task log will contain four additional columns conveying batching information: the task-resource batch identifier, the task-resource batch type, the batch subprocess identifier, and the batch subprocess type. Hence, when applicable, the algorithm will have grouped task instances in batches at the task-resource level and identified batch subprocesses.

Definition 11 (Batch-enriched task log)**.** *Let T_{ex} represent an extended task log and let T_{be} be a batch-enriched task log. To transform T_{ex} to T_{be}, two mapping functions m_4 and m_5 are sequentially applied:*

- *$m_4 : T_{ex} \rightarrow T'_{ex}$ assigns a batch subprocess identifier $b_{s,i}$ and a batch subprocess type $b_{s,t}$, with $b_{s,i} \in \mathbb{N}$ and $b_{s,t} \in \{$par, seq task-based, conc task-based$\}$. The values of $b_{s,i}$ and $b_{s,t}$ are shared among the task instances i which are part of the same parallel batch subprocess or sequential/concurrent task-based batch subprocess.*
- *$m_5 : T'_{ex} \rightarrow T_{be}$ assigns a batch subprocess identifier $b_{s,i}$ and a batch subprocess type $b_{s,t}$, with $b_{s,i} \in \mathbb{N}$ and $b_{s,t} \in \{$seq case-based, conc case-based$\}$. The values of $b_{s,i}$ and $b_{s,t}$ are shared among the task instances i which are part of the same sequential/concurrent case-based batch subprocess.*

After the application of m_4 and m_5, the batch-enriched task log T_{be} consists of a set of batched task instances $t'_b = (c, t, r, \tau_{arrival}, \tau_{start}, \tau_{complete}, b_{tr,i}, b_{tr,t}, b_{s,i}, b_{s,t})$.

4.7. Implementation

A prototypical implementation of BAMA has been developed in `R`[4] and is publicly available at `https://github.com/nielsmartin/bama` or `https://doi.org/10.5281/zenodo.3653952`. `R` is a programming language providing extensive functionalities for data manipulation and statistical analysis. The key packages that are used are `dplyr`[5] for data manipulations and summarisations, `lubridate`[6] to work with timestamps and `reshape` to convert the event log to a task log. When applying sequence mining during case-based batch detection, the implementation of the SPADE algorithm in the `arulesSequences`[7] package is used.

In order to use the implementation, an event log is required. This event log should take the form of a data frame, which is a rectangular data table format in R in which columns contain variables and rows represent observations [40]. When an event log is available in the XES-format[8], it can be converted to the correct input format by leveraging the R-package `xesreadR`[9], which is part of the `bupaR` suite supporting process analysis in R [41].

Once the event log has been imported, a helper function is available to convert the event log to a task log. Afterwards, batch identification can be initiated. An integrated function enables the detection of batching behavior at both the task-resource and subprocess level. This function must be parameterized by the user. The following parameters need to be specified to support the identification of batching behavior at the task-resource level:

- For sequential batch detection: a list containing the maximal tolerated time gaps between consecutive task instances. Each entry represents the time gap related to a particular task as, e.g., setup times might differ across tasks. A helper function is available to create this list.
- For sequential batch detection: a boolean indicating whether the event log contains the arrival time of a case at a task. When available, the arrival time is used to distinguish between sequential batching behavior and regular queue handling.

[4] `https://www.r-project.org`
[5] `https://CRAN.R-project.org/package=dplyr`
[6] `https://CRAN.R-project.org/package=lubridate`
[7] `https://CRAN.R-project.org/package=arulesSequences`
[8] XES is an XML-based standard for the exchange of event logs [2].
[9] `https://CRAN.R-project.org/package=xesreadR`

- A boolean indicating whether timestamps are expressed numerically, instead of in a date-time format.

- The format of the timestamps in the event log (e.g. *yyyy-mm-dd hh:mm:ss*).

The following parameters need to be specified for the detection of sequential/concurrent case-based batch subprocesses:

- An entry reflecting the way in which subsequences were generated, i.e. by means of enumeration or using sequence mining.

- A list containing the subsequences for which case-based sequential or concurrent batch subprocesses need to be detected. Helper functions are developed to create this list using enumeration or sequence mining using the SPADE algorithm.

- For sequential case-based batch detection: a maximal tolerated time gap between consecutive instances associated to a particular case (i.e. used to detect within-case batching).

- For sequential case-based batch detection: a maximal tolerated time gap between the (aggregated) within-case batches across consecutive cases (i.e. used to detect between-case batching).

4.8. Metrics and visualizations

As follows from Section 4.6, the outcome of BAMA is a batch-enriched task log, which is a task log containing four additional columns with batching information. As real-life task logs typically contain a large number of entries, a manual inspection of the batch-enriched task log is infeasible. Consequently, it is important that the batching information in the batch-enriched task log is presented in a concise and intuitive way in order to gain a rich understanding in batching behavior. While a detailed overview on this matter is beyond the scope of this paper, the remainder of this subsection provides some pointers regarding (i) the potential of batching behavior metrics, and (ii) potential visualizations based on a batch-enriched task log.

A first way to convey batching insights in a concise way is the development of *batching behavior metrics*. The authors have discussed metrics for batch activities in previous works [6, 27]. A relevant metric is, for instance, the number of occurred batch activities as a percentage of the total number of occurrences of the task(s). This allows insights into the actual usage of batching for certain tasks and subprocesses. Another metric can relate to the average or median batch size, providing insights in the number of cases which are typically included in a batch. Batching is usually applied to save time and costs, such that the average or median execution time and related costs of the batches can be compared to task executions which are not batched. When several resources can perform a particular batch activity, metrics can be calculated in general (i.e. disregarding the distinction between resources), or can be expressed at the level of individual resources. Positioning metrics at the resource level enables to analyze whether batching behavior is concentrated amongst a limited number of resources.

Besides numeric batching behavior metrics, the batch-enriched task log can also be used to *visualize batching behavior*. Graphs can take the metrics as a starting point and, e.g., show how the instances of a particular task are divided over the batch types. This can show the analyst how prevalent batching behavior is for a particular task. The evolution of particular metrics over time can also be visualized, which highlights whether batching behavior is related to particular times of day or days of the week. When considering a longer time horizon, it can also be observed whether batching behavior evolves over time. Next to graphs taking metrics as a starting point, the batching information in the batch-enriched task log can be projected on a Dotted Chart, in which the dots of batched task instances are colored, while non-batched instances are grey. Some examples of visualizations will be included in the evaluation of the proposed algorithm using real-life data (Section 5.2).

The batch-enriched task log can also be leveraged as an input for a visual analysis tool for batching behavior. Such a tool would enable an analyst to start from a high-level overview of batching behavior and interactively drill-down to study a specific batch or resource. Other features of such a tool could include the benchmark of resources, or the comparison between the characteristics of batched task instances and task instances which are not batched.

5. Evaluation

To evaluate BAMA, a twofold approach is followed: an evaluation using synthetic event logs and an evaluation using a real-life log. Both types of evaluation serve different goals. In Section 5.1, synthetic event logs are used to evaluate the algorithm's ability to rediscover known batches. Because it is known which task instances belonged to a batch in (simulated) reality, synthetic logs make it possible to demonstrate the effectiveness of BAMA to correctly detect batching behavior when it is present. Afterwards, in Section 5.2, the algorithm is applied to a real-life event log. To this end, historic data from a digital whiteboard system deployed in a surgical ward of a university hospital in Norway is used. Besides demonstrating BAMA's applicability within a real-life setting, the insights that it can generate in such a real-life context are also highlighted.

5.1. Experiments with synthetic event logs

5.1.1. Experimental design

The goal of the evaluation using synthetic data is to determine whether the algorithm can rediscover known batches solely using an event log. To this end, synthetic event logs are created by simulating a BPMN process diagram using the extendable and open-source BPMN process simulator *Scylla* [42]. Nine distinct models with varying degrees of complexity are included, as shown in Figure 13. We generated different variants of batch subprocesses (i) with a linear behavior (i.e., consisting of several tasks connected sequentially) including also a process model with two batch subprocesses (cf. #3 in Figure 13), (ii) with parallel behavior through an AND gateway, (iii) with a choice through a XOR gateway, and (iv) with a combination of AND and XOR gateways.

For each process diagram, a separate event log is generated for each batch processing type, i.e. parallel, sequential/concurrent task-based, and sequential/concurrent case-based batching. As a consequence, 45 distinct event logs are generated. The number of simulated process instances per event log is set to 1,000. All synthetic event logs are publicly available at `https://github.com/nielsmartin/bama` or at `https://doi.org/10.5281/zenodo.3653952`.

For each synthetic event log, BAMA is applied using a maximum allowed time gap for sequential batching of zero seconds. Changing this value would have no impact on the results as the synthetic data originates from a simulated environment in which each task or batch activity has dedicated resources. Regarding the method to generate subsequences as candidates for case-based batching, *enumeration* is selected.

It should be stressed that, before feeding the event log to the algorithm, all information reflecting how batches are formed according to the simula-

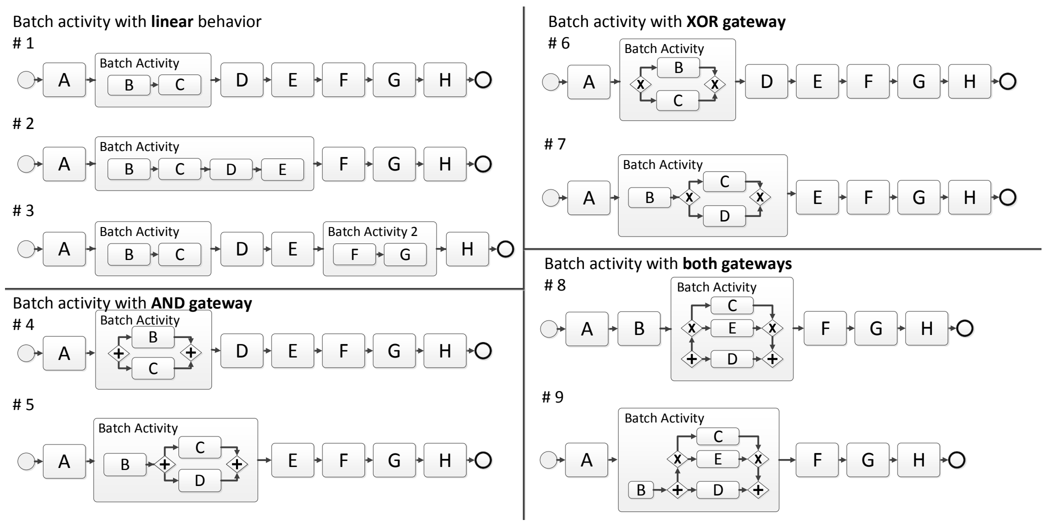


Figure 13: BPMN process diagrams used for the generation of synthetic event logs

tor's batching logic is removed. Using this event log, completely agnostic of the batches present in reality, the algorithm detects batches following the procedure outlined in Section 4.

To evaluate the algorithm's performance, the detected batches are compared to the batches created during the simulation. The simulated environment enables us to know which task instances are grouped together in a batch according to the simulator's batching logic. Hence, the algorithm should be able to identify these batches from the synthetic log, providing a basis to evaluate whether the algorithm can identify different types of batching behavior when we know that the input data contains such batching behavior. For performance evaluation purposes, a task instance is considered to be incorrectly assigned when it either (i) is included in a batch of the correct type, but in a composition with instances different from the composition created by the simulator's batching logic, or (ii) is included in a wrong type of batch. Note that the first criterion is rather rigorous as the composition of discovered batches has to be completely correct. For instance: when BAMA rediscovers a batch for all but one instance, the entire batch will be considered to be incorrectly assigned.

5.1.2. Results

The use of synthetic data enables a direct comparison of the task instances which are batched by the batching logic embedded in the simulator on the one hand, and task instances which are batched according to BAMA's output. To quantify the degree to which BAMA can correctly rediscover batches in synthetic data, the following three key output measures are calculated for each synthetic event log:

- **The percentage of correctly rediscovered batched instances.** This measure represents the percentage of task instances which are batched according to the simulator's batching logic which BAMA correctly rediscovers. In other words, this measure represents the percentage of task instances batched according to the simulator's batching logic, which are correctly marked as part of the correct batch by the algorithm.

- **The percentage of correctly rediscovered instances.** This measure represents the percentage of task instances for which BAMA detects the correct batching behavior. Differently from the previous output measure, this measure also takes into account task instances which are not batched according to the simulator's batching logic. As a consequence, this measure considers all task instances, while the previous measure focused on task instances which were batched according to the simulator's batching logic.

- **The percentage of instances which are not batched according to the simulator's batching logic, but which are reported as part of a sequential batch by BAMA.** This measure focuses on task instances which are not explicitly batched according to the simulator's batching logic. Despite the fact that these instances are not purposefully batched by the simulator, a sequential batching pattern might still be present because handling long queues can fulfill the requirements for sequential batching[10]. To quantify the extent to which this phenomenon is present, this measure highlights the percentage of instances which are not batched according to the simulator's batching logic, but for which BAMA indicates that sequential batching occurs.

[10]Suppose that a large number of cases is waiting to be processed by a particular resource at a particular task. In that case, a group of queuing cases might be present at the task before the resource starts to process the first case in this group. As a consequence, under these conditions, sequential batching can be detected.

Table 10: Summary statistics on the output measures (excluding diagram #6)

output measure	**mean**	**sd**	**median**	**min**	**max**
% of correctly rediscovered batched instances	100.00	0.00	100.00	100.00	100.00
% of correctly rediscovered instances	90.12	2.65	90.97	83.33	91.67
% of instances not batched according to the simulator's batching logic reported as part of sequential batch	16.19	6.84	13.49	11.21	33.33

Batch identification on all 45 synthetic event log is executed on a standard laptop computer. The average runtime was well below 10 seconds. The detailed results for each synthetic log are included in Table A.11 in Appendix A. From these results, it follows that the algorithm will correctly rediscover all batching behavior which is introduced according to the simulator's batching logic. The only exception is process diagram #6, where the batch subprocess is not rediscovered as the model consists of two tasks in an XOR-construct. As either activity B or C will be observed in a trace, the algorithm will not discover that both tasks are part of a batch subprocess. This observation will be revisited in the discussion (Section 6).

When disregarding diagram #6, which is a clear outlier in terms of performance, Table 10 provides summary statistics on the three output measures. The table shows that all task instances which the simulator batches according to its batching logic are correctly rediscovered by the algorithm (100% rediscovery). When instances which are not batched according to the simulator's batching logic are also taken into account, the algorithm classifies, on average, 90.12% of the instances correctly. Given the fact that all instances that the simulator batches according to its batching logic are correctly rediscovered, this last result indicates that BAMA reports some task instances which are not batched by the simulator as being part of a batch. This is supported by the last output measure in Table 10: on average 16.19% of the instances which are not batched according to the simulator's batching logic are marked as being part of a sequential batch by BAMA. This latter observation explains the entire deviation between the designed batching behavior in the simulator and our algorithm's output.

From what said, the following two observations follow: (i) all batched task instances are correctly rediscovered for all process diagrams besides diagram #6, and (ii) the identified batch behavior is not correctly discovered for all task instances because BAMA detects sequential batching behavior which is not introduced according to the simulator's batching logic. These observations will be discussed in Section 6.

As highlighted in the experimental design, *enumeration* was used as the method to generate subsequences which are candidate for case-based batching. For the sake of completeness, it should be noted that rerunning all experiments with *sequence mining* as a subsequence identification method generated identical results. Differences in terms of runtime were minimal and remained, on average, well below 10 seconds for each synthetic log using a standard laptop computer.

5.2. Discovering Batch Activities in Usage of a Hospital Ward

BAMA is also applied to a real-life event log obtained from a digital whiteboard system that is deployed in a surgical ward of an university hospital in Norway. The event log has been subject to a previous study in which higher level activities were recognized from the low-level events recording every change in the system [43]. The event log is a good candidate for evaluating BAMA as it is expected to contain some batching behavior in how information is entered for patients. Indeed, batching behavior was already presumed in Mannhardt et al. [43] albeit solely based on a preliminary visual analysis using a Dotted Chart [44]. Specifically, the hypothesis regarding batching was that nurses would often only use the whiteboard to update certain information around the times of a shift change.

The whiteboard system was introduced as a light-weight support system for coordination and collaboration among nurses and is meant to support the daily work of nurses in the hospital [46]. Figure 14 shows the main screen of the whiteboard in which each row corresponds to all the information entered for a single patient. Each patient is assigned to a responsible nurse and that nurse can update information for that patient in the whiteboard, e.g., the planned treatment, necessary reports, or the planned discharge date. Next to medical information, the whiteboard is also linked to a call signal system, which records when patients raise an alarm or nurses attend the patient in their room. Finally, the whiteboard system also records when the responsible nurse for a patient changes. Thus, resource information is available for all events.

5.2.1. Experimental design

We obtained an event log with 8,487 cases tracking the updates made on the whiteboard system for individual patients. In total, there are 298,636 events recorded. The events are recorded at fine granularity, i.e. every change

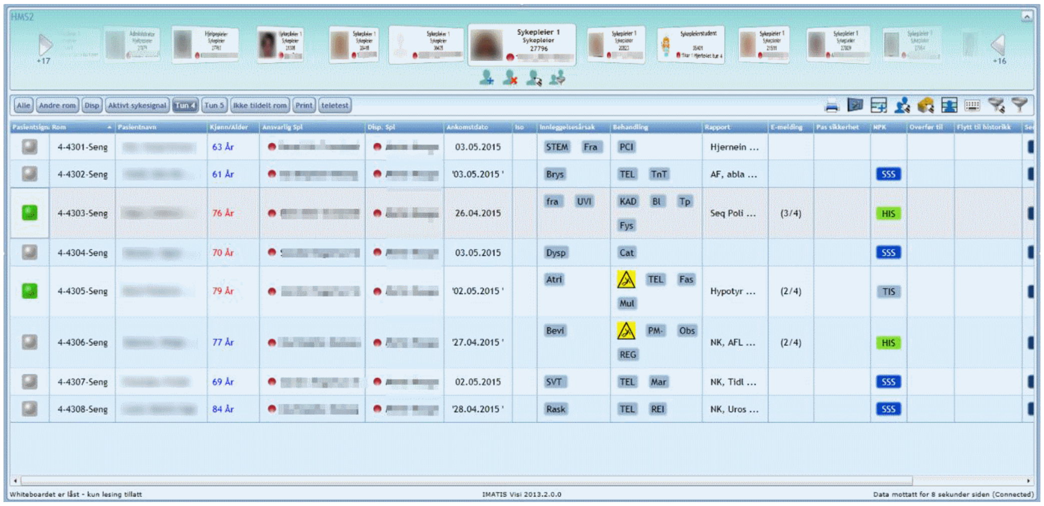


Figure 14: Whiteboard system used in the hospital as described in [45]. Each row contains information about a patient can be updated directly through a touch screen interface. The event log of the system contains events for all the updates made grouped by patient. Note that some entries are purposefully blurred for anonymization purposes.

of a cell in the whiteboard yields an event, and carries only a single timestamp as payload. We used the abstraction method and patterns described in Mannhardt et al. [43] to derive a high-level event log with 6,455 cases[11] and 70,936 instances of 14 distinct activities that have a defined *start* and *completion* timestamp. The activities refer to:

- pre-announcement, registration, and transfer of the patient;
- five different usage patterns of the nurse call signal system;
- updates of treatment and diagnosis information as well as generic reports on the patient;
- handover between nurses.

Further details on the semantics of all activities are provided in Mannhardt et al. [43].

[11] For some cases the abstraction method did result in empty traces as no activity was recognized. These cases were filtered out.

BAMA is applied to the prepared event log. We used SPADE to identify frequent subsequences as exhaustively computing all subsequences took about 10 minutes and returned more than 130,000 subsequences, which would result in a very long computation time when applying BAMA. By varying the minimum support level parameter between 0.005 and 0.4, we investigated the trade-off on the frequency of detected case-based batching and found 0.01 to be giving the best result within a computation time of 3 minutes. The tolerated time gap parameters of BAMA were set to 3 minutes based on the domain knowledge that some activities may incur additional work beyond what is captured in the timeframe between the *start* and *complete* time of an activity instance. Again, we investigated the impact of the parameter by varying it from 30 seconds to 12 minutes. As expected, more batch behavior is detected when increasing the parameter value. BAMA detects 15.7% more batching when using 12 minutes compared to 30 seconds. Finally, in absence of more precise information in the event log, we assumed the arrival time of tasks to be 5 minutes before their start.

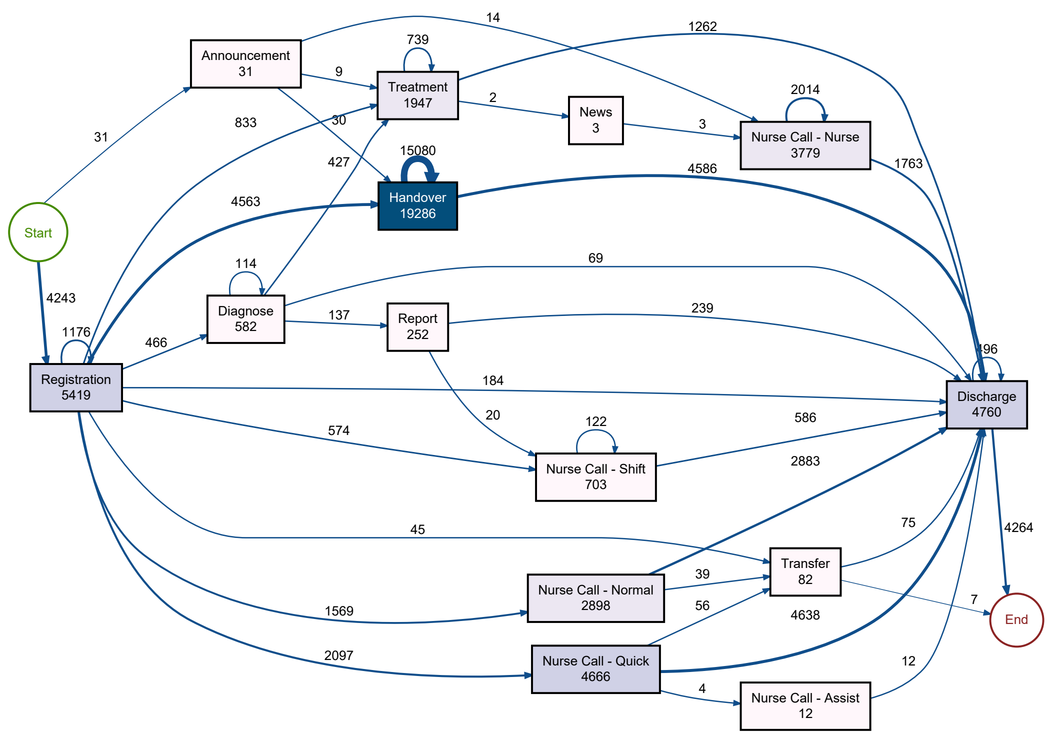


Figure 15: Process map discovered with Flexible Heuristics Miner (FHM) on the whiteboard event log showing the main activities supported by the whiteboard system.

Figure 15 gives an overview of the process behavior for all completed cases, i.e., those starting with either *'Registration'* or *'Announcement'* and ending with *'Discharge'* or *'Transfer*. The process map shows causal dependencies discovered with the Flexible Heuristics Miner (FHM) together with projected frequencies from the event log. As expected there is little structure since most of the activities can be performed in any order. Some of the activities can be repeated such as *'Handover'*, *'Nurse Call - Nurse'* and *'Nurse Call - Shift'*, as well as activities which update patient information such as *'Diagnose'* and *'Treatment*. The former three activities are initiated by the nurse, who visits the patient's room without a request from the patient. Finally, both *'Registration'* and *'Discharge'* are repeated, which can be attributed to technical issues as the source system duplicates these events.

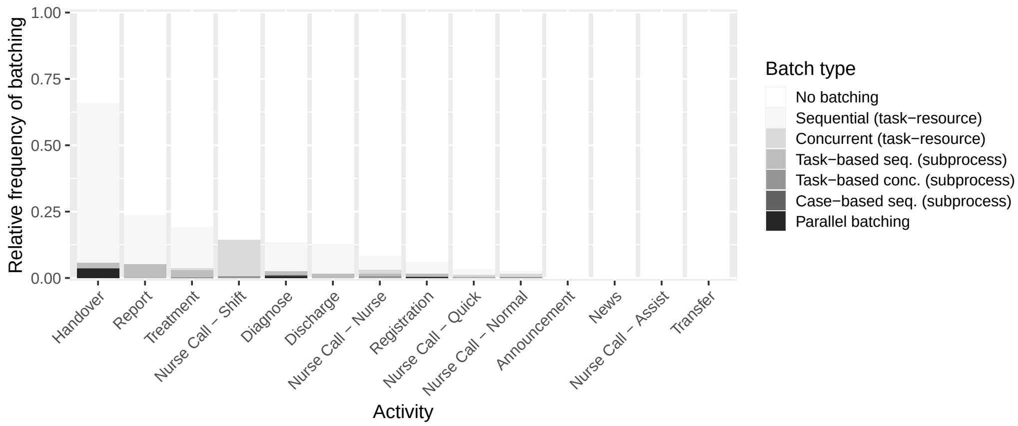
5.2.2. Results

The BAMA algorithm detected 30,530 task instances that were batched in the digital whiteboard system. The execution time was about 3 minutes on a standard laptop computer. Figure 16 shows an overview of the relative frequencies with which batching behavior is detected for the different activities. This overview includes both batched instances at the task-resource level, which could already be detected by the state-of-the-art technique described in Martin et al. [6], as well as batch subprocesses, which are the core contribution of this work.

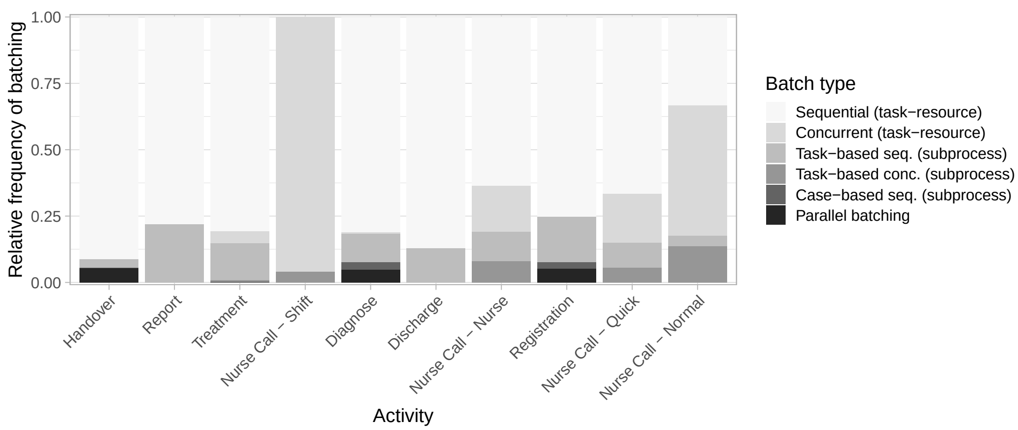
The most frequent task related to batching behavior is *'Handover'*. This is not surprising as handover of work normally takes place at the end of a shift and the change in responsibility for each patient is registered on the whiteboard. So, it is to be expected that this task is executed by nurses as a sequential batch. Beyond this obvious batching behavior, BAMA identified batching behavior in more than 20% of the task instances for the activities *'Report'*, *'Treatment'*, *'Diagnose'*, and *'Discharge'* as well as two types of interactions with the call signal system: *'Nurse Call - Shift'* and *'Nurse Call - Nurse'*. In the latter two tasks, nurses use the call signal to indicate their position in the ward.

These results partially confirm the batching hypothesis that was formulated in Myrstad [45]. However, in contrast to the basic analysis with a Dotted Chart visualization in Myrstad [45], the application of BAMA accurately quantifies the amount of batching behavior, which is less than expected.

Next, we looked at the batching behavior at the subprocess level that was identified by BAMA. Figure 17 shows how frequently a task appears to



(a)



(b)

Figure 16: Distribution of the batch types (cf. Figure 4) identified for task instances. In (a), instances for which no batching behavior is detected are included (*No batching*). These instances are excluded in (b).

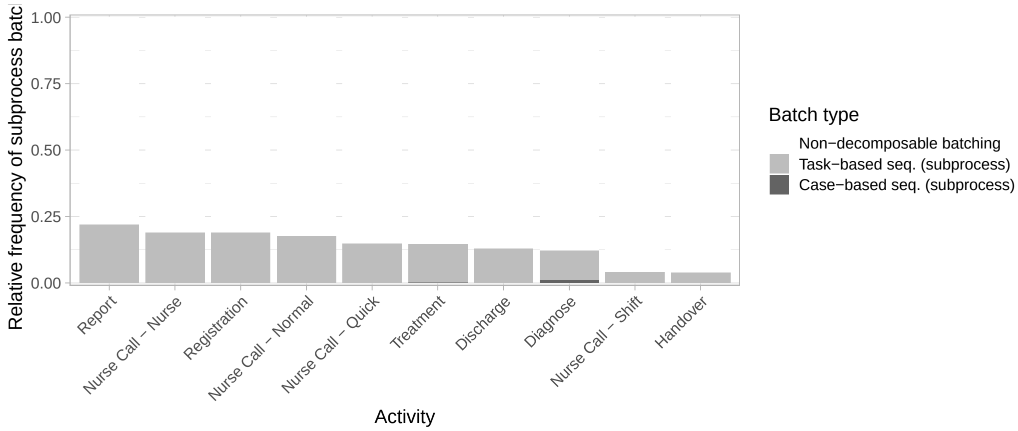


Figure 17: Distribution of type of batching at the subprocess level compared based on the overall batching.

be batched in a subprocess in relation to the overall batching behavior that was identified for that task, i.e., the number of activity instances in subprocess batching compared to the overall number of instances performed in batches. Overall, subprocess batching was observed infrequently in the event log. For example, in about 12% of the cases, the tasks *'Nurse Call - Normal'* and *'Report'* were identified as being part of sequential subprocess batching. Concurrent subprocess batching is almost entirely absent, which is to be expected since there is normally only one nurse responsible for the patients on a ward. We did not discover parallel batch subprocesses. In total, BAMA discovered 1,220 task instances that were executed in subprocess batching out of which 1,194 instances were part of task-based batch subprocesses and 26 were part of case-based batch subprocesses.

In Figure 18, we grouped the detected batch subprocesses by the tasks involved and indicate their ordering with the symbol '→' to investigate the batching behavior in more detail. Most of the subprocesses include the task *'Handover'* batched together with tasks *'Discharge'*, *'Treatment'*, *'Nurse Call - Nurse'*, *'Report'*, and *'Registration'*. Whereas batching of the *'Handover'* task is to be expected, entering information about the treatment for patients (*'Treatment'*) or entering information in the report field of the whiteboard (*'Report'*) does not necessarily need to be batched. The whiteboard is supposed to be used continuously during the shift to always have the latest information available. However, as already investigated in Myrstad [45], it is often only entered afterwards.

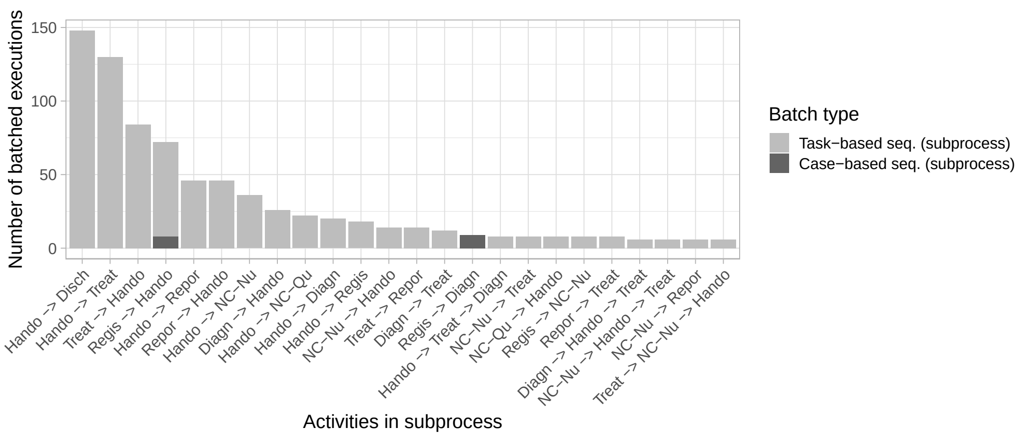


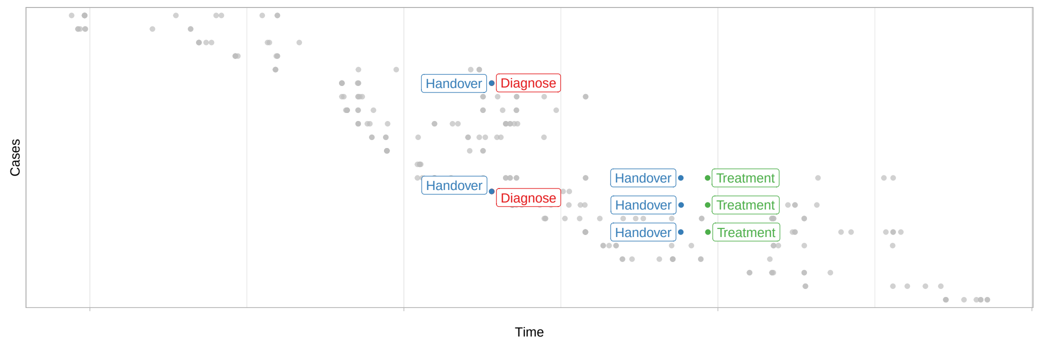
Figure 18: Frequency of individual subprocesses that are detected to be executed in batches and appear more than 10 times in the event log.

We further investigate some of the batching behavior using a Dotted Chart analysis for specific working days. Figure 19 shows three exemplary time periods on which batching behavior at the subprocess level is visible. Each row (y-axis) corresponds to a distinct case being handled and the x-axis shows calendar time with each vertical line indicating a 12 hour period. We highlighted the start timestamps of task executions that are part of sequential task-based batch subprocesses. For example, Figure 19a shows an example of how the handover of a shift and the update of the treatment information is batched together for three patients. In Figure 19b, there are two handover task executions registered followed by a discharge for two patients. Moreover, in another batch two sequential executions of using the call signal system (*'Nurse Call - Nurse'* and *'Nurse Call - Quick'*) are registered. Figure 19c shows again a batch consisting of instances of *'Handover'* and *'Discharge'*.

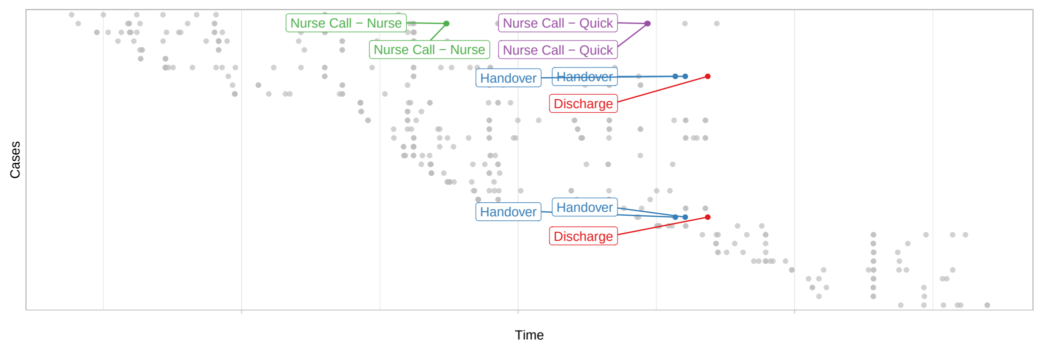
6. Discussion

6.1. Analysis of findings

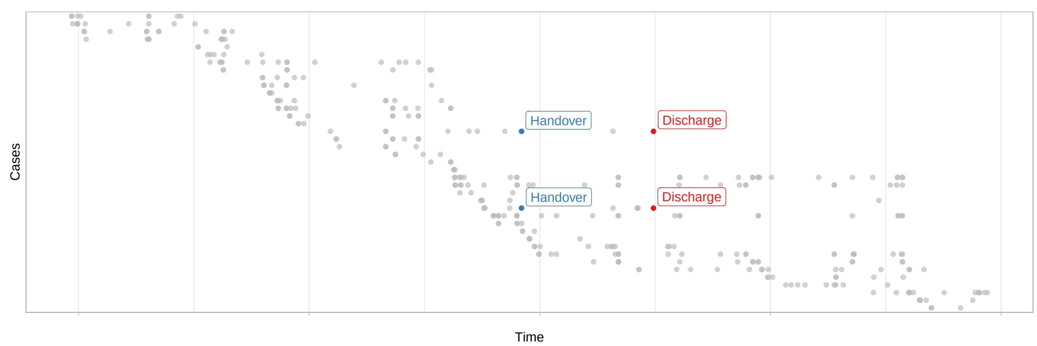
From the evaluation using synthetic data, it follows that the algorithm can correctly rediscover all batches for all but one of the considered BPMN process diagrams. This shows that BAMA can identify batch tasks and batch subprocesses when these are present. The BPMN diagram for which the algorithm does not retrieve the correct batch subprocess is model #6 (cf. Figure 13). The batch subprocess included in this model consists of two



(a)



(b)



(c)

Figure 19: Dotted chart of events recorded in the observation ward with those events highlighted that have been detected by BAMA as task-based subprocesses within the course of a single day. All batch subprocesses are sequential, i.e., the task instances did not occur in parallel, which is not visible due to the coarse time scale. The exact time is not revealed for privacy reasons.

tasks in an XOR-construct. Consequently, either task B or task C will be executed for a batch. Hence, the algorithm will identify batching behavior at the task-resource level for B or C instead of a batch subprocess consisting of B and C. To circumvent this limitation, BAMA's output can be projected on a process model generated using an existing control-flow discovery algorithm. This will highlight the relationship between these batch tasks and, hence, the fact that a batch subprocess is formed.

Another observation from the evaluation using synthetic data is that BAMA detects more batching behavior than the batches which were purposefully introduced by the simulator. More specifically, some task instances which are not batched according to the simulator's batching logic are marked by BAMA as part of a sequential batch (at the task-resource level). Even though this behavior is not introduced by the simulator's batching logic, it presents valid batching behavior as long queues which are sequentially processed can fulfill the criteria for sequential batching. In particular, sequential batching requires that a group of cases is queuing for the resource at a task before this resource starts processing the first case within this group. When queues are long, this condition can be fulfilled. Against this background, the detection of sequential batching behavior by BAMA does not constitute a performance issue of the algorithm.

The application of the algorithm on real-life data shows that BAMA identifies batching both at the task-resource level and at the subprocess level. Overall, the application of BAMA generated more profound insights into batching behavior than previous analyses of the same data with standard methods, such as a Dotted Chart analysis. In particular, BAMA can be used to exactly quantify the fraction of batch executions for certain parts of the process. Compared to the previous analysis in Mannhardt et al. [43] and Myrstad [45], batching of certain subprocesses (e.g., *'Report'* followed by *'Handover'*) was confirmed, but at a lower frequency than expected. The generated insights provides support to the hospital to investigate the operational use of the whiteboard system. For instance: the observation that particular reporting tasks are batched together with handover tasks indicates that the whiteboard is not always used continuously during the shift.

While the synthetic logs and the real-life log in Section 5 originate from a single process, BAMA can also identify batching behavior in a multi-process context. This is due to the fact that the algorithm focuses on how resources perform tasks. Consequently, when tasks of several processes are included in the event log, BAMA can also detect batch subprocesses containing tasks of different processes.

6.2. Limitations

Besides the contributions of this paper, its limitations also need to be recognized. Firstly, the algorithm will identify a batch subprocess solely consisting of a set of tasks in an XOR-construct as batching behavior at the task-resource level. As outlined above, this limitation can be circumvented by projecting the algorithm's output on the output of a control-flow discovery algorithm.

Secondly, the algorithm imposes some requirements on the event log used as an input. Foremost, the start and completion of a task instance need to be recorded, which might not be the case in particular real-life event logs. When events are recorded at a fine granularity, this issue can be tackled by applying abstraction methods, as discussed in the real-life data evaluation. Besides start and completion times, the arrival time of a case at an activity is used to distinguish sequential batching from regular queue handling. However, it should be noted that the absence of an arrival time or a suitable proxy does not impede the application of BAMA. When the arrival time is not available, the conditions for sequential batching are less stringent as the requirement that all cases need to be present before the processing of a batch starts cannot be enforced.

Finally, the algorithm centers around the identification of batching behavior, but does not focus on the operational effects of such behavior. However, the algorithm identifies a wide variety of batching behavior, both at the task-resource level and at the subprocess level. This provides a solid basis to investigate why particular cases are batched and whether this is desirable from a performance perspective.

7. Conclusion

This paper presents the Batch Activity Mining Algorithm (BAMA), which is a novel algorithm to automatically detect batching behavior from an event log. It extends prior research on this matter by enabling the discovery of both batch tasks and batch subprocesses. The evaluation, both on synthetic and real-life data, demonstrates the algorithm's ability to identify batches in an event log. BAMA's output provides organizations with quantitative insights in the occurrence of a wide variety of batching behaviors. Compared to a mere visual analysis, e.g. using Dotted Charts, the presented algorithm provides a more structured approach to identify batching behavior. Batching can have a positive impact (e.g. a reduction in the number of setups), as well as a negative impact (e.g. an increase in waiting times for some cases) on the performance of a process. The algorithm offers detailed insights into the recorded batch behavior and can be helpful to explain the observed process duration in more detail. As the algorithm focuses on the relation between tasks and resources, it could also detect batch behavior over multiple processes if tasks of the different processes are included in one event log.

Building upon the work presented in this paper, several interesting directions for future research can be distinguished. Firstly, BAMA can be extended with a set of aggregated metrics and visualization functions. Currently, the algorithm focuses on the identification of batch tasks and subprocesses. Taking this output as a starting point, future developments could create a framework to, e.g., interactively analyze batching behavior. Some pointers were already included in Section 4.8. Secondly, future work can link the identified batches to its operational effects (e.g. its impact on waiting times from a customer's perspective) to determine whether a particular type of batching behavior is desirable from the organizational perspective. Moreover, the organization could determine whether particular persons or teams are more inclined to exhibit batching behavior. Finally, it is worthwhile to investigate whether inductive batching insights can be embedded in a predictive process monitoring framework. This would imply that knowledge about batching behavior from the past would be used as one of the predictors to estimate, e.g., the remaining time required to finish a case.

Acknowledgement. We would like to thank Leon Bein (Master student at Hasso Plattner Institute) for extending the simulator Scylla and for supporting the generation of the syntactic event logs. We would also like to sincerely thank the reviewers for their constructive feedback during the review process.

Appendix A. Detailed output measures of the evaluation on synthetic data

1. Process diagram #1: simple var1 - one batch activity with two tasks
2. Process diagram #2: simple var2 - one batch activity with four tasks
3. Process diagram #3: simple var3 - two batch activities, each consisting of two tasks
4. Process diagram #4: AND var 1 - one batch activity with two tasks

5. Process diagram #5: AND var 2 - one batch activity with three tasks
6. Process diagram #6: XOR var 1 - one batch activity with two tasks in XOR construct
7. Process diagram #7: XOR var 2 - one batch activity with one 'fixed' task and two tasks in XOR construct
8. Process diagram #8: MIX var 1 - one batch activity with AND followed by XOR in one branch
9. Process diagram #9: MIX var 2 - one batch activity with one 'fixed' task, followed by an AND with an XOR in one branch

Table A.11: Detailed output measures of evaluation on synthetic data

log number	**process diagram**	**batch processing type***	**% of correctly rediscovered batched instances**	**% of correctly rediscovered instances**	**% of instances not batched according to the simulator's batching logic reported as seq. batch**
1	1	par.	100.00	91.52	11.31
2	1	s.t.b.	100.00	91.49	11.34
3	1	s.c.b.	100.00	91.59	11.21
4	1	c.t.b.	100.00	91.49	11.34
5	1	c.c.b.	100.00	91.59	11.21
6	2	par.	100.00	91.67	16.67
7	2	s.t.b.	100.00	91.67	16.67
8	2	s.c.b.	100.00	91.67	16.67
9	2	c.t.b.	100.00	91.67	16.67
10	2	c.c.b.	100.00	91.67	16.67
11	3	par.	100.00	83.33	33.33
12	3	s.t.b.	100.00	83.33	33.33
13	3	s.c.b.	100.00	83.33	33.33
14	3	c.t.b.	100.00	83.33	33.33
15	3	c.c.b.	100.00	83.33	33.33
16	4	par.	100.00	91.47	11.38
17	4	s.t.b.	100.00	91.59	11.21
18	4	s.c.b.	100.00	91.54	11.28
19	4	c.t.b.	100.00	91.59	11.21
20	4	c.c.b.	100.00	91.54	11.28
21	5	par.	100.00	91.59	13.45
22	5	s.t.b.	100.00	91.57	13.49
23	5	s.c.b.	100.00	91.57	13.49
24	5	c.t.b.	100.00	91.57	13.49
25	5	c.c.b.	100.00	91.57	13.49
26	6	par.	0.00	76.05	11.28
27	6	s.t.b.	0.00	76.02	11.31
28	6	s.c.b.	0.00	76.16	11.14
29	6	c.t.b.	0.00	76.02	11.31
30	6	c.c.b.	0.00	76.16	11.14
31	7	par.	100.00	90.25	13.65
32	7	s.t.b.	100.00	90.36	13.49
33	7	s.c.b.	100.00	90.39	13.45
34	7	c.t.b.	100.00	90.36	13.49
35	7	c.c.b.	100.00	90.39	13.45
36	8	par.	100.00	90.48	13.33
37	8	s.t.b.	100.00	90.48	13.33
38	8	s.c.b.	100.00	90.48	13.33
39	8	c.t.b.	100.00	90.48	13.33
40	8	c.c.b.	100.00	90.48	13.33
41	9	par.	100.00	90.48	16.67
42	9	s.t.b.	100.00	90.48	16.67
43	9	s.c.b.	100.00	90.48	16.67
44	9	c.t.b.	100.00	90.48	16.67
45	9	c.c.b.	100.00	90.48	16.67

** par.: parallel, s.t.b.: sequential task-based, s.c.b.: sequential case-based, c.t.b.: concurrent task-based, c.c.b.: concurrent case-based*

[1] M. Weske, Business process management: concepts, languages, architectures, Springer-Verlag Berlin Heidelberg, 2012.

[2] W. M. P. van der Aalst, Process mining: data science in action, Springer, Heidelberg, 2016.

[3] L. Pufahl, M. Weske, Requirements framework for batch processing in business processes, Lecture Notes in Business Information Processing 287 (2017) 85–100.

[4] C. N. Potts, M. Y. Kovalyov, Scheduling with batching: a review, European journal of operational research 120 (2000) 228–249.

[5] J. Medhi, Stochastic models in queueing theory, Academic Press, 2002.

[6] N. Martin, M. Swennen, B. Depaire, M. Jans, A. Caris, K. Vanhoof, Retrieving batch organisation of work insights from event logs, Decision Support Systems 100 (2017) 119–128.

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

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

[9] Y. Liu, L. Zhang, J. Wang, Mining workflow event log to facilitate parallel work item sharing among human resources, International Journal of Computer Integrated Manufacturing 24 (2011) 864–877.

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

[11] N. Slack, S. Chambers, R. Johnston, Operations and process management: principles and practice for strategic impact, Pearson Education, 2009.

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

[13] S. Henn, G. Wäscher, Tabu search heuristics for the order batching problem in manual order picking systems, European Journal of Operational Research 222 (2012) 484–494.

[14] S. Hong, A. L. Johnson, B. A. Peters, Batch picking in narrow-aisle order picking systems with consideration for picker blocking, European Journal of Operational Research 221 (2012) 557–570.

[15] S. Hong, Y. Kim, A route-selecting order batching model with the S-shape routes in a parallel-aisle order picking system, European Journal of Operational Research 257 (2017) 185–196.

[16] J. K. Higginson, J. H. Bookbinder, Policy recommendations for a shipment-consolidation program, Journal of Business Logistics 15 (1994).

[17] S. Cetinkaya, J. H. Bookbinder, Stochastic models for the dispatch of consolidated shipments, Transportation Research Part B: Methodological 37 (2003) 747–768.

[18] F. Mutlu, S. i. l. Cetinkaya, J. H. Bookbinder, An analytical model for computing the optimal time-and-quantity-based policy for consolidated shipments, IIE Transactions 42 (2010) 367–377.

[19] L. Tang, G. Wang, Decision support system for the batching problems of steelmaking and continuous-casting production, Omega 36 (2008) 976–991.

[20] C. N. Potts, M. Y. Kovalyov, Scheduling with batching: a review, European Journal of Operational Research 120 (2000) 228–249.

[21] M. Mathirajan, A. I. Sivakumar, A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor, The International Journal of Advanced Manufacturing Technology 29 (2006) 990–1001.

[22] M. F. Neuts, A general class of bulk queues with Poisson input, The Annals of Mathematical Statistics 38 (1967) 759–770.

[23] K. Sikdar, U. C. Gupta, Analytic and numerical aspects of batch service queues with single vacation, Computers & Operations Research 32 (2005) 943–966.

[24] C. Natschläger, A. Bögl, V. Geist, M. Biró, Optimizing resource utilization by combining activities across process instances, in: EuroSPI, Springer, pp. 155–167.

[25] J. Pflug, S. Rinderle-Ma, Application of dynamic instance queuing to activity sequences in cooperative business process scenarios, International Journal of Cooperative Information Systems (2016) 1650002.

[26] L. Pufahl, M. Weske, Batch activity: enhancing business process modeling and enactment with batch processing, Computing (2019) 1–25.

[27] 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.

[28] N. Martin, A. Solti, J. Mendling, B. Depaire, A. Caris, Mining batch activation rules from event logs, IEEE Transactions on Services Computing (2019).

[29] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, Queue mining: predicting delays in service processes, Lecture Notes in Computer Science 8484 (2014) 42–57.

[30] I. Weber, M. Farshchi, J. Mendling, J.-G. Schneider, Mining processes with multi-instantiation, in: Proceedings of the 30th Annual ACM Symposium on Applied Computing, pp. 1231–1237.

[31] OMG, Notation BPMN version 2.0, OMG Specification, Object Management Group (2011).

[32] A. Senderovich, C. Di Francescomarino, C. Ghidini, K. Jorbina, F. M. Maggi, Intra and inter-case features in predictive process monitoring: A tale of two dimensions, in: International Conference on Business Process Management, Springer, pp. 306–323.

[33] K. Winter, F. Stertz, S. Rinderle-Ma, Discovering instance and process spanning constraints from process execution logs, Information Systems 89 (2020) 101484.

[34] W. Fdhila, M. Gall, S. Rinderle-Ma, J. Mangler, C. Indiono, Classification and formalization of instance-spanning constraints in process-driven

applications, in: International Conference on Business Process Management, Springer, pp. 348–364.

[35] C. W. Günther, H. M. W. Verbeek, XES standard definition, Technical report, Eindhoven Unversity of Technology, Eindhoven, The Netherlands, 2014.

[36] T. Baier, J. Mendling, M. Weske, Bridging abstraction layers in process mining, Information Systems 46 (2014) 123–139.

[37] J. De Weerdt, M. De Backer, J. Vanthienen, B. Baesens, A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs, Information Systems 37 (2012) 654–676.

[38] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, F. M. Maggi, A. Marrella, M. Mecella, A. Soo, Automated discovery of process models from event logs: review and benchmark, IEEE Transactions on Knowledge and Data Engineering 31 (2018) 686–705.

[39] M. J. Zaki, Spade: an efficient algorithm for mining frequent sequences, Machine learning 42 (2001) 31–60.

[40] H. Wickham, G. Grolemund, R for data science: import, tidy, transform, visualize, and model data, O'Reilly, Sebastopol, 2017.

[41] G. Janssenswillen, B. Depaire, M. Swennen, M. Jans, K. Vanhoof, bupar: enabling reproducible business process analysis, Knowledge-Based Systems 163 (2019) 927–930.

[42] L. Pufahl, T. Y. Wong, M. Weske, Design of an extensible BPMN process simulator, Lecture Notes in Business Information Processing 308 (2017) 782–795.

[43] F. Mannhardt, M. de Leoni, H. A. Reijers, W. M. P. van der Aalst, P. J. Toussaint, From low-level events to activities - a pattern-based approach, Lecture Notes in Computer Science 9850 (2016) 125–141.

[44] M. Song, W. M. P. van der Aalst, Supporting process mining by showing events at a glance, in: Proceedings of the 17th Annual Workshop on Information Technologies and Systems, pp. 139–145.

[45] I. A. Myrstad, Prosessstøtte i sengetun, Master's thesis, NTNU, 2017.

[46] M. R. Halvorsen, H. O. Austad, A. D. Landmark, D. Ausen, I. Svagård, T. Tomasevic, T. Trondsen, Redesigning work with a lightweight approach to coordination technology, CIN: Computers, Informatics, Nursing 37 (2019) 124–132.