Information Systems 89 (2020) 101484

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

Discovering instance and process spanning constraints from process execution logs[☆]

Karolin Winter[a], Florian Stertz[a], Stefanie Rinderle-Ma[a,b,*]

[a] Research Group Workflow Systems and Technology, Faculty of Computer Science, University of Vienna, Austria
[b] Data Science@Uni Vienna, University of Vienna, Vienna, Austria

ARTICLE INFO

Article history:
Received 29 April 2019
Received in revised form 2 December 2019
Accepted 3 December 2019
Available online 9 December 2019

Keywords:
Digitalized compliance management
Constraint mining
Instance spanning constraints
Process mining

ABSTRACT

Instance spanning constraints (ISC) are the instrument to establish controls across multiple instances of one or several processes. A multitude of applications crave for ISC support. Consider, for example, the bundling and unbundling of cargo across several instances of a logistics process or dependencies between examinations in different medical treatment processes. Non-compliance with ISC can lead to severe consequences and penalties, e.g., dangerous effects due to undesired drug interactions. ISC might stem from regulatory documents, extracted by domain experts. Another source for ISC are process execution logs. Process execution logs store execution information for process instances, and hence, inherently, the effects of ISC. Discovering ISC from process execution logs can support ISC design and implementation (if the ISC was not known beforehand) and the validation of the ISC during its life time. This work contributes a categorization of ISC as well as four discovery algorithms for ISC candidates from process execution logs. The discovered ISC candidates are put into context of the associated processes and can be further validated with domain experts. The algorithms are prototypically implemented and evaluated based on artificial and real-world process execution logs. The results facilitate ISC design as well as validation and hence contribute to a digitalized ISC and compliance management.

1. Introduction

Business Process Compliance (BPC) means *"to formally and (semi-)automatically prove that business processes comply with relevant constraints such as regulations, laws, or guidelines"* [1] and poses a tremendous challenge for companies and organizations nowadays. One example is the ISO 27001 security standard[1] where *"for a company the average cost (respectively duration) to implement the ISO 27001 standard is estimated between $6500 and $26 000 (respectively between 6 and 12 months)"* [2]. Another example is IT-supported compliance in the medical domain [3] which helps to prevent *"errors in medicine [that] are not rare and may cause severe harm"* [4]. An example for a compliance constraint from the medical domain is: *If adoption of laboratory tests occurs in a trace, it is never followed by a follow-up outpatient consultation* (taken and translated from [1]). Associated treatment processes must comply to this constraint.

Ensuring BPC refers to both constraints and business processes. For the latter, typically, a process type (e.g., dermatology treatment) is described by a process model. At runtime the model can be instantiated, e.g., creating an instance per patient, and the instances are executed through, for example, a Process-Aware Information System. As stated in [5] *"many organizations maintain repositories that contain hundreds of business process models"*, naming e.g., IBM's insurance architecture (IAA). Another example is a large hospital where a patient might undergo more than one treatment process. Finally, [6] describes a case study from the higher education institution with 108 process types/models and 375 compliance constraints.

1.1. Problem statement

BPC follows a life cycle that includes several phases, among them *Constraint elicitation*, *Constraint formalization*, *Constraint verification*, and *Constraint validation & redesign* (adapted from [7]). A body of work exists for the formalization and verification of the compliance of business process models and process instances with compliance constraints during design time (e.g., [8]) and

☆ This work has been partly funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-072 and by the Austrian Research Promotion Agency (FFG) via the "Austrian Competence Center for Digital Production" (CDP) under the contract number 854187.

* Corresponding author at: Data Science@Uni Vienna, University of Vienna, Vienna, Austria.

E-mail addresses: karolin.winter@univie.ac.at (K. Winter), florian.stertz@univie.ac.at (F. Stertz), stefanie.rinderle-ma@univie.ac.at (S. Rinderle-Ma).

1 https://www.iso.org/isoiec-27001-information-security.html.

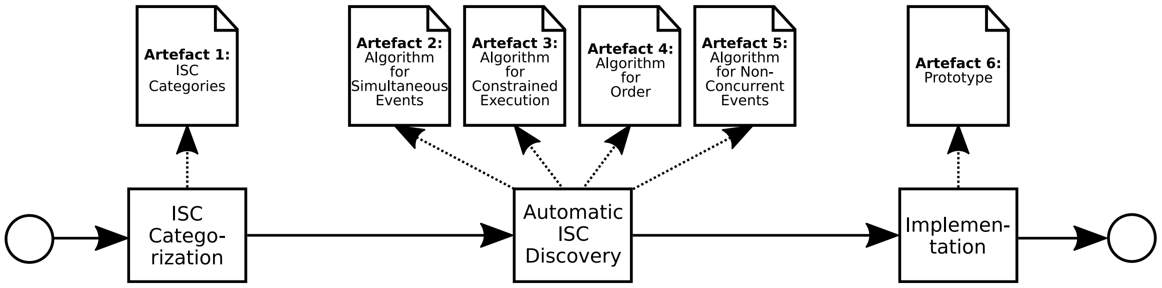

Artefact 1: ISC Categories
Artefact 2: Algorithm for Simultaneous Events
Artefact 3: Algorithm for Constrained Execution
Artefact 4: Algorithm for Order
Artefact 5: Algorithm for Non-Concurrent Events
Artefact 6: Prototype
ISC Catego-rization
Automatic ISC Discovery
Implemen-tation

Fig. 1. Method and created artifacts.

runtime (e.g., [1]). Despite this, among others, the following two challenges persist:

Ch1 Handling of constraints that span multiple instances of one or several process types (cf. [9–11]). Such constraints are called *Instance Spanning Constraints* (ISC) [9].
Ch2 Support for constraint elicitation from data sources such as text [12], often in the context of regulatory documents [13, 14] and process execution logs [15].

This work addresses both challenges *Ch1* and *Ch2* in interplay. Challenge Ch1 refers to supporting ISC that are present in various domains, including logistics, production, and health care [9]. ISC might span multiple instances of one or several process types (the latter abbreviated as $\text{ISC}_P \subseteq \text{ISC}$). An example for an ISC_P is: *"A patient is involved simultaneously in two different treatments, i.e., dermatology and diabetes. A list of drugs should not be taken during the dermatology treatment. The diabetes treatment should consider this list."* [16]. In detail: two process types exist, i.e., one for dermatology and one for diabetes treatment. For the dermatology treatment a list of drugs is contra-indicated. If a patient undergoes both treatments at the same time, it has to be ensured that for the diabetes treatment none of the contra-indicated drugs for the dermatology treatment is administered. A violation might lead to severe consequences regarding patient health. This constraint obviously spans both process types and hence is an ISC_P.

In this work, we refine Challenge *Ch2* to ISC and specifically to *ISC discovery and analysis* based on process execution logs in an ex post manner. Although ISC play a significant role in digitalized process compliance management, ISC discovery from process execution logs has not been considered comprehensively enough until now.

The focus is on providing an automatic discovery approach. This constitutes the next step towards a digitalized compliance management for companies that is capable of dealing with ISC. In particular, the approach alleviates the cumbersome discovery of ISC which is up to now often a manual and error-prone task. Doing so the approach has to go far beyond existing work suggesting semi-automatic ISC discovery [10].

Note that we envision to discover ISC *candidates* from process execution logs. These ISC candidates can be employed in two ways:

(i) for discovering new ISC on the process instances in case the ISC were not known beforehand and
(ii) checking existing ISC.

The latter is particularly interesting for flexible and volatile environments where concept drifts [17] occur in the process execution logs.

Reacting to Challenges *Ch1* and *Ch2*, this work tackles the following research questions.

RQ1 How to design an algorithm for discovering ISC candidates with minimal human involvement?
RQ2 Which characteristics and requirements must process execution logs meet to enable the discovery of ISC candidates?
RQ3 How can the ISC mining algorithm control varying quality of process execution logs?

RQ1 targets the provision of discovery algorithms that yield ISC candidates, i.e., suggestions of possible ISC that can be further discussed and validated with domain experts. Thus, *RQ1* aims at addressing the sweet spot between results machines can provide in terms of algorithms and minimal human involvement in terms of validation. *RQ2* focuses on identifying requirements that process execution logs need to fulfill to enable ISC candidate discovery. This connects also to *RQ3* in terms of finding parameter settings in the algorithms to adjust the quality of the outcome, mainly in terms of false positives and negatives, considering the quality of the underlying log.

1.2. Contribution

The work follows the design science research methodology [18]. Hence, at an abstract level, it follows the steps "artefact creation" and "artefact evaluation". Fig. 1 depicts the structure of the paper and the artifacts created in this paper. At first, we discuss related work in Section 2 and introduce fundamentals in Section 3. As a first contribution ISC are analyzed and categorized along their type and corresponding manifestation in the process execution log ($\mapsto$ Artifact 1: ISC categorization, cf. Section 3), e.g., ISC demanding an order between tasks in processes of different types reflected by observing the respective order in the process execution logs. Artifact 1 is utilized for designing the automatic discovery approach for ISC candidates.

The approach is developed in Section 4 and results in four algorithms (Artifacts 2 – 5) along the ISC categorization. All artifacts are prototypically implemented ($\mapsto$ Artifact 6). Implementation details are described in Section 5. The automatic discovery algorithms are also evaluated based on 1 artificial and 2 real-life logs, the latter from the manufacturing and the higher education domains (cf. Section 6). It can be shown that on the one hand ISC candidates for actual ISC are discovered for the different categories, but on the other hand, non-existing ISC are not discovered. An in-depth analysis of the trade-off of precision and recall is offered in order to support users in the application of the presented algorithms, specifically, in setting parameter values for the algorithms. With a suitable parameter setting, for example, the false positive rate can be kept low. Subsequent to the evaluation, we provide a discussion of the approach and its limitations in Section 7. The paper closes with a summary and outlook on future research directions in Section 8.

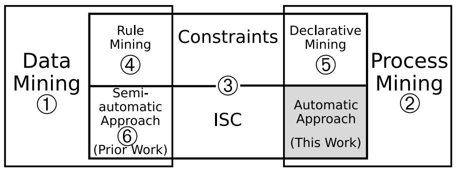


Fig. 2. Related fields and their intersections.

2. Related work

ISC discovery is rooted in the area of Business Process Compliance (BPC) [1,8,19]. In the BPC life cycle (cf. [7]) ISC discovery can be located in the phases of constraint elicitation and validation. ISC discovery can be regarded as constraint mining technique. Existing constraint mining techniques – from a method point of view – can be found at the intersection of data mining and process mining (cf. Fig. 2).

① *"[D]ata mining is originally defined as the analysis-step in the process of the knowledge discovery in data bases which has a more exploratory nature"*, [20] and is concerned with mining patterns from different kind of data [21].

② *Process mining* has unfolded as a set of techniques to discover, analyze, and enhance processes based on process execution logs [22]. Most of the existing algorithms, e.g., the Heuristics Miner [23] or Fuzzy Miner [24], are applied ex post, i.e., after process instance execution has completed. Recently, online process mining has gained interest in discovering process models based on event streams [25,26]. Conformance checking determines in how far a given process execution log conforms to a given process model, again either ex post [27] or during runtime [28]. The quality of discovered process models is measured based on fitness, simplicity, precision, and generalization [29]. Regarding the trade-off between the automatic provision of results by mining algorithms and the involvement of humans, the Process Mining Manifesto further states: *"Balancing fitness, simplicity, precision and generalization is challenging. This is the reason that most of the more powerful process discovery techniques provide various parameters. Improved algorithms need to be developed to better balance the four competing quality dimensions. Moreover, any parameters used should be understandable by end-users."* [29]. This work also involves a set of input parameters chosen by users. Moreover, the output, i.e., the ISC candidates, typically need to be evaluated by users such as domain experts.

③ *Compliance constraints* and *ISC* are the subject of investigation in this paper, i.e., the envisioned output of the provided algorithms. In process compliance management, constraints and ISC implement regulatory norms on top of the processes. [30] presents an abstract framework for normative requirements, e.g., permissions (something is allowed) and prohibitions (something is not allowed). The framework in [1] proposes modeling functionalities for constraints, i.e., they might refer to control flow of the processes, data, time, and resources. The goal is to check and achieve compliance of the processes with the constraints. One option is compliance by design as introduced specifically for artifact-centric business processes in [31]. In the latter work the business process model is used together with compliance constraints in order to generate a compliant process model in a declarative manner. The other options are checking compliance during design time (i.e., using the process model [8]) or during runtime (i.e., compliance monitoring using process execution logs [1]). The approaches for checking compliance during runtime can be also utilized to check compliance ex post, i.e., on process execution logs instead of event streams.

ISC form a subset of constraints by spanning multiple instances of one or several process models [32]. They also follow a lifecycle. At first, ISC are modeled and formalized using, for example, Event Calculus (EC) with a well-defined structure and semantics [9]. ISC can the be implemented, enacted and monitored. One ISC implementation option on top of a Rete rule engine is provided in [33]. ISC are and can be used in various applications including instance batching [34,35], instance queuing [36,37], and security [32]. [11] aim at predicting inter-case features in process monitoring which can be a first step towards prediction on ISC compliance.

④ Discovering constraints from sources such as text using text mining and Natural Language Processing has emerged as recent research topic (cf. e.g., [12–14]), but is outside the scope of this work. Some *data mining* techniques are concerned with the discovery of patterns in the form of rules, i.e., association rule mining [38], sequence mining [39], and episode mining [40]. [41] employs association rule mining for anomaly detection in process execution logs. Other approaches utilize classification to find rules on decision points in processes [15,42]. However, the focus of these approaches is not on ISC.

⑤ *Declarative process mining* approaches employ process mining techniques to discover rules from process execution logs, e.g., MobuconEC [43] and MobuconLTL [44], and from event streams [45]. As pointed out in [1], these approaches are not concerned with ISC yet. Here, the automatic approach presented in this paper steps in and presents process mining based techniques to discover ISC from process execution logs.

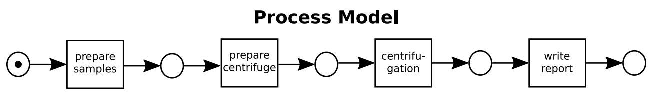
⑥ *Semi-automatic ISC discovery:* In prior work, we presented a semi-automatic approach for ISC discovery from process execution logs based on classification techniques in [10]. At first, the process execution logs are pre-processed as for discovering ISC it becomes necessary to merge traces of the instances based on "shared" (instance spanning) attributes such as resources, data and timestamps. The selection of these attributes is done by the user. Then decision trees are built based on the pre-processed process execution logs and ISC candidates displayed as decision rules are derived. Promising results were achieved based on artificial logs and a real-world process execution log from the higher education domain. For the latter two previously unknown ISC expressing best practices were discovered and acknowledged by experts. However, the approach presented in [10] – beyond being semi-automatic – does not cover ISC_P.

For detecting batch activation rules from process execution logs, [46] employ an approach similar to the semi-automatic approach provided in [10], i.e., based on classification. By contrast, this work presents algorithms for automatic discovery for the far wider set of ISC including e.g., order relations between process models.

3. ISC categorization

As said in the introduction, a *process type*, e.g., a lab process, is described by a *process model*. Fig. 3 depicts a lab process modeled using Petri Net notation that consists of 4 process activities ordered in a sequence. We opt for Petri Net notation as process mining results are also often presented as Petri Nets and because Petri Nets are suitable for demonstrating the semantics of ISC based on these models in the sequel. Another notation would be, for example, Business Process Modeling and Notation (BPMN) as the de facto standard.

At runtime the process model can be instantiated, e.g., per patient or lab sample. The resulting *process instances* are then executed. The execution information of the process instances can be either reflected by markings (see the token in Fig. 3) or logged in so called *process execution logs*. More precisely, for each activity execution one or several corresponding *events* together with their timestamp and possibly additional data are stored.



Process Execution Log

Case ID	Event ID	Event Attributes		
		Timestamp	Activity	Lifecycle Transition
1	7181	15-04-2019:08.30	prepare samples	start
	7182	15-04-2019:09.14	prepare samples	complete
	7183	15-04-2019:09.18	prepare centrifuge	start
	7184	15-04-2019:09.22	prepare centrifuge	complete
	7185	15-04-2019:13.56	centrifugation	start
	7186	15-04-2019:14.08	centrifugation	complete
	7187	15-04-2019:14.23	write report	start
	7188	15-04-2019:15.01	write report	complete
2	7281	15-04-2019:08.33	prepare samples	start
	⋮			

Fig. 3. Example of a process model (using petri net notation) and a corresponding process execution log.

Fig. 3 at the bottom shows an excerpt of an execution log for instances executed based on the process model at the top. The events are grouped by case ids that correspond and identify the associated process instances. Formally, a process execution log can be defined as follows:

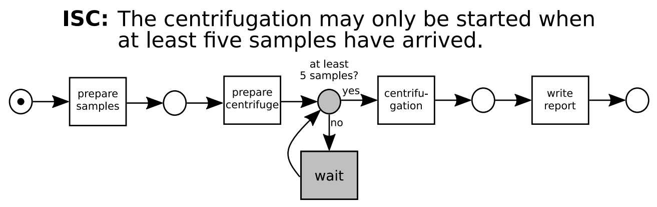
Definition 1 (*Process Execution Log, [22]*). *A* process execution log L *(log in short) for a* process type P *consists of several* cases, *i.e., process instances. Each instance corresponds to a* trace t_i *of unique ordered* events. *Each event can be equipped with* (event) attributes *which may contain information about timestamps, organizational resources, and costs.*

In Fig. 3, the displayed trace with case ID 1 consists of eight events, each having an event ID as well as the attributes *timestamp, activity (label)* and *lifecycle transition*. The latter refers to the fact that the execution of activities might follow a life cycle, differentiating, for example, the start and completion of activities, reflected by start and completion events in the log.

This work aims at discovering ISC based on logs. In the following structure and semantics of ISC are explained and illustrated. In a nutshell, an ISC refers to

- a *context*, i.e., one or multiple instances of one or multiple processes,
- a *connection*, i.e., an activity or process pattern in the underlying process model(s) the ISC refers to. This comes together with a *trigger position* that reflects whether the ISC is triggered before or after the associated activity/process pattern,
- a *condition* as a logical expression over the data, time, or resources and their values of the underlying process instances,
- a *behavior* describing the action part triggered by the ISC, i.e., a resource attribution, timing attribution, or component semantic exception.

Consider the ISC *"The centrifugation may only be started when at least five samples have arrived"*. (taken from [10]) based on the Petri Net depicted in Fig. 4. The context of the ISC is the lab process. The connection is activity `centrifugation` where the trigger position is before the activity start. The condition is that the number of instances before the start of `centrifugation` is ≥ 5. In case the condition is met, `centrifugation` starts (behavior).


ISC: The centrifugation may only be started when at least five samples have arrived.

Process Execution Log

Case ID	Event ID	Event Attributes		
		Timestamp	Activity	Lifecycle Transition
1	7181	15-04-2019:08.30	prepare samples	start
	⋮			
	7185	15-04-2019:13.56	centrifugation	start
	7186	15-04-2019:14.08	centrifugation	complete
	⋮			
2	7281	15-04-2019:08.33	prepare samples	start
	⋮			
	7285	15-04-2019:13.56	centrifugation	start
	7286	15-04-2019:14.03	centrifugation	complete
	⋮			
⋮				
5	7581	15-04-2019:09.13	prepare samples	start
	⋮			
	7585	15-04-2019:13.56	centrifugation	start
	7586	15-04-2019:14.15	centrifugation	complete
	⋮			
6	7681	15-04-2019:10.04	prepare samples	start
	⋮			
	7685	15-04-2019:16.19	centrifugation	start
	7686	15-04-2019:16.45	centrifugation	complete
	⋮			

Fig. 4. Explaining ISC semantics based on ISC decision points (cf. [10]).

The semantics of this ISC can be described as follows: within the underlying process model, right before the start of activity `centrifugation`, the ISC triggers an implicit action `wait` (until the 5 instances have arrived at the `centrifugation`). The resource `centrifuge` constitutes a shared resource where each time 5 instances are collected. In order to reflect this behavior, we introduce a decision point into the process model that invokes the `wait` activity until 5 instances have arrived, and then the `centrifugation` activity is invoked, i.e., the decision rule reflects the condition part of the ISC. The `wait` activity corresponds to a silent task that is not reflected in the log by any event.

The example shows that this ISC cannot be (exclusively) determined based on log information produced by process activities (activity `wait` is artificial and silent and not traceable in the log). In fact, event attributes *time, resource* or *data*, have to be additionally considered, i.e., the logs have to be prepared based on these attributes for "making the ISC visible". As indicated by Fig. 4, we assume the timestamps of the events thrown by starting activity `centrifugation` being equal for process instances 1 to 5. That, in turn, indicates that the execution of activity `centrifugation` is synchronized across a certain number of process instances.

Until now the structure and semantics of ISC were introduced and explained. Now we reason about how ISC manifest in logs. Based on the results, ISC discovery algorithms are designed and provided in the next section. For this, ISC characteristics with respect to underlying log information based on the data set of

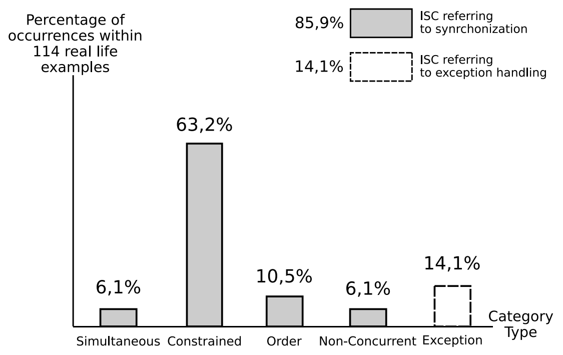


Fig. 5. Percentage values of ISC categories within the 114 real-life examples.

114 ISC examples from [16] are analyzed (cf. Fig. 5). The analysis shows that 85.9% of the ISC examples (98 overall) trace back to a synchronization at a certain point, regardless whether they span multiple instances or multiple processes. In this case, synchronization means that a sequence of events/instances/processes must be coordinated, i.e., synchronization can happen at different levels, i.e., single or multiple events and instances or entire processes can synchronize. For synchronization, mostly the action `wait` is triggered (cf. Fig. 4). For exception handling, which the remaining 14.1% of the ISC examples refer to, the action can encounter more complex actions like rollback or compensations [47]. Hence, ISC regarding exception handling will be treated in future work as they require different concepts for discovery, mainly due to the different behavior part of the ISC.

For the 98 ISC referring to synchronization, the following categories with respect to the effects on the underlying logs are extracted (together with number of occurrences):

I Simultaneous execution of activities: 7

II Constrained activity execution (based on time or data constraints and constraints regarding the absolute number of executions): 72

III Order of activity execution: 12

IV Non-concurrent execution of activities: 7

The categorization serves for automatically discovering ISC candidates ($\mapsto$ RQ1) and the analysis of ISC examples yields that for this, logs need to fulfill the following minimal requirements ($\mapsto$ RQ2):

- Events in the log must have
  - *concept names*, i.e., activity labels (mandatory for all categories, must be unique);
  - *timestamps* (mandatory for all categories).
- For Category IV *lifecycle transitions start* and *end* of an activity execution are mandatory.
- Whenever ISC span multiple process types, traces or events should have a *unique case identifier*, i.e., an attribute that enables the merging of traces stemming from different processes, e.g., a customer id.

If no unique identifier is given, the merging of traces could be done manually or by using the techniques described in [48] (cf. Section 4). These types of information are usually provided by common Process Aware Information Systems (PAIS) and represent standard extensions of the XES standard.[2] In order to meet RQ3 the ISC mining algorithms presented in the next section contain several parameters for adapting them based on the quality of the underlying log files.

4. Algorithms for ISC discovery

The automatic discovery of ISC candidates employs four algorithms, each of them reflecting one of the ISC Categories I – IV. (cf. Section 3, $\mapsto$ Artifact 2 – 5). At first the logs are prepared as described in Section 4.1. As illustrated in Fig. 6, in parallel to the ISC candidate discovery, a process model for each log is mined using a process discovery algorithm.[3] Per category, the result contains one or several process models enriched with the ISC candidates for the corresponding category.

Pre-processing and the algorithms are illustrated by the example depicted in Fig. 7.

4.1. Pre-processing of logs

If one process type is subject to ISC discovery the log associated to this process type can be taken as is. If more than one process type and hence more than one log is analyzed (i.e., to discover process-spanning ISC candidates), the different logs must be linked. The reason is that the discovery algorithms have to “know” which instances of the different process types “belong together”, i.e., are linked. Linking is mostly based on the fact that the instances refer to the same subject such as the same patient, customer, or product. In detail, linking logs for multiple process types is done using a linking attribute which is, in the optimum case, a unique case identifier. Examples comprise customer id, patient id, or an arbitrary id like `1b5` and `4z7` in the running example depicted in Fig. 7. Based on this attribute, associated traces, i.e., traces having the same attribute value, are combined. The resulting (merged) log file consequently consists of merged traces of different log files as illustrated in Fig. 8. For example, if one patient is involved in different processes, the patient’s id is used to retrieve the traces of this patient. During the merging of two or more traces into one single trace the events are rearranged based on their timestamps. If no unique case identifier is given, the merging can be based on the techniques presented in [48] where process execution logs are merged based on identifying shared data that is assumed to stem from the same process instance and stating merging rules for creating a new process execution log.

Since ISC of Categ. III and IV are ISC_P, i.e., ISC spanning multiple process types, the merging of logs becomes mandatory. For Categ. I and II ISC spanning only one process type are possible, consequently the input can be the original log.

4.2. Processing

For Algorithms 1–4, the (merged) log is used as input data structure. The output is a list of ISC candidates discovered from the (merged) log. The ISC candidates enrich the corresponding process model(s) which are mined in parallel. Each ISC candidate must be verified by the user afterwards, i.e., the algorithms deliver suggestions for ISC. In order to reduce the number of false positives caused by chance or, e.g., logging errors, each algorithm can be fine-tuned by at least one parameter. In the following, all of these parameters are introduced to give an overview. A short subsequent description as well as the algorithms show how the parameters are utilized. The evaluation (cf. Section 6) and the discussion (cf. Section 7) elaborate on the effects of the different parameter settings.

[2] http://www.xes-standard.org.

[3] In the evaluation the Heuristics Miner [23] is exemplary used.

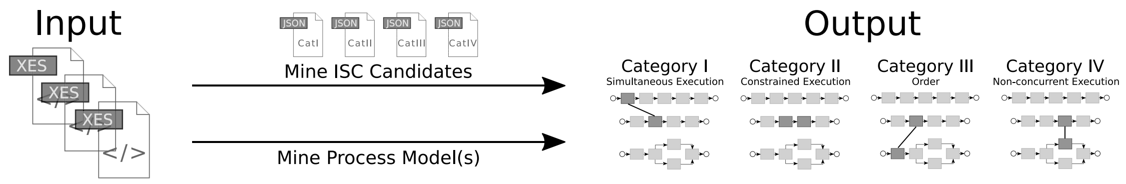
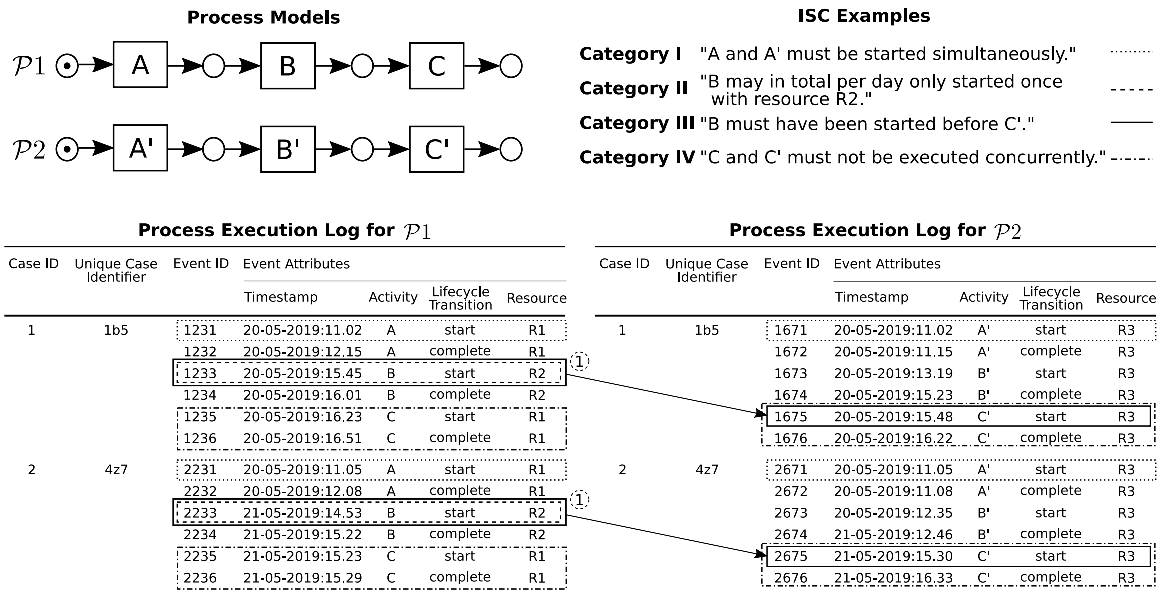


Fig. 6. Overview of ISC discovery approach.



ISC Examples

Category I "A and A' must be started simultaneously."

Category II "B may in total per day only started once with resource R2."

Category III "B must have been started before C'."

Category IV "C and C' must not be executed concurrently."

Process Execution Log for $\mathcal{P}1$

Case ID	Unique Case Identifier	Event ID	Event Attributes			
			Timestamp	Activity	Lifecycle Transition	Resource
1	1b5	1231	20-05-2019:11.02	A	start	R1
		1232	20-05-2019:12.15	A	complete	R1
		1233	20-05-2019:15.45	B	start	R2
		1234	20-05-2019:16.01	B	complete	R2
		1235	20-05-2019:16.23	C	start	R1
		1236	20-05-2019:16.51	C	complete	R1
2	4z7	2231	20-05-2019:11.05	A	start	R1
		2232	20-05-2019:12.08	A	complete	R1
		2233	21-05-2019:14.53	B	start	R2
		2234	21-05-2019:15.22	B	complete	R2
		2235	21-05-2019:15.23	C	start	R1
		2236	21-05-2019:15.29	C	complete	R1

Process Execution Log for $\mathcal{P}2$

Case ID	Unique Case Identifier	Event ID	Event Attributes			
			Timestamp	Activity	Lifecycle Transition	Resource
1	1b5	1671	20-05-2019:11.02	A'	start	R3
		1672	20-05-2019:11.15	A'	complete	R3
		1673	20-05-2019:13.19	B'	start	R3
		1674	20-05-2019:15.23	B'	complete	R3
		1675	20-05-2019:15.48	C'	start	R3
		1676	20-05-2019:16.22	C'	complete	R3
2	4z7	2671	20-05-2019:11.05	A'	start	R3
		2672	20-05-2019:11.08	A'	complete	R3
		2673	20-05-2019:12.35	B'	start	R3
		2674	21-05-2019:12.46	B'	complete	R3
		2675	21-05-2019:15.30	C'	start	R3
		2676	21-05-2019:16.33	C'	complete	R3

Fig. 7. Running example for illustration of algorithms.

Merged Process Execution Log

Case ID = Unique Case Identifier	Event ID	Event Attributes			
		Timestamp	Activity	Lifecycle Transition	Resource
1b5	1231	20-05-2019:11.02	A	start	R1
	1671	20-05-2019:11.02	A'	start	R3
	1672	20-05-2019:11.15	A'	complete	R3
	1232	20-05-2019:12.15	A	complete	R1
	1673	20-05-2019:13.19	B'	start	R3
	1674	20-05-2019:15.23	B'	complete	R3
	1233	20-05-2019:15.45	B	start	R2
	1675	20-05-2019:15.48	C'	start	R3
	1234	20-05-2019:16.01	B	complete	R2
	1676	20-05-2019:16.22	C'	complete	R3
	1235	20-05-2019:16.23	C	start	R1
	1236	20-05-2019:16.51	C	complete	R1
4z7	2231	20-05-2019:11.05	A	start	R1
	2671	20-05-2019:11.05	A'	start	R3
	2672	20-05-2019:11.08	A'	complete	R3
	2232	20-05-2019:12.08	A	complete	R1
	2673	20-05-2019:12.35	B'	start	R3
	2674	21-05-2019:12.46	B'	complete	R3
	2233	21-05-2019:14.53	B	start	R2
	2234	21-05-2019:15.22	B	complete	R2
	2235	21-05-2019:15.23	C	start	R1
	2236	21-05-2019:15.29	C	complete	R1
	2675	21-05-2019:15.30	C'	start	R3
	2676	21-05-2019:16.33	C'	complete	R3

Fig. 8. Preprocessing the logs for the running example.

- Algorithm 1: $\gamma_1 \in [0, 1]$ and $\varepsilon_1 \in \mathbb{R}_0^+$ in seconds
- Algorithm 2: $\gamma_2 \in [0, 1]$ and $\varepsilon_2 \in \mathbb{R}_0^+$ in seconds
- Algorithm 3: $\gamma_3 \in [0, 1]$ and $\kappa \in [0, 0.5)$
- Algorithm 4: $\varepsilon_4 \in \mathbb{R}_0^+$ in seconds.

Parameters $\gamma_i, i = 1, 2, 3$ represent the filtering of relative occurrences, i.e., if $\gamma_i = 1$ then in 100% of all cases the observation must have been made whereas for $\gamma_i = 0$ in 0% of all cases the observation must have been made. Parameters $\varepsilon_j, j = 1, 4$ are intended to compensate logging errors and are therefore given in seconds. ε_2 also has time unit seconds, but represents an activity execution time limit. The parameter κ is specific for Alg. 3 and represents the number of inverse orders that may occur.

In the pseudo-codes of the algorithms

- activity labels are abbreviated as *lb*,
- timestamps as *ts*,
- lifecycle transitions as *lc* and
- organizational resources as *rs*.

4.2.1. Category I

Algorithm 1 discovers ISC candidates of Category I, i.e., based on *simultaneous execution of activities*. This, in turn, is reflected in the (merged) log by events that occur at (almost) the same time for more than one instance.

The algorithm starts by iterating over all traces in the (merged) log. If `checkLC` evaluates true, i.e., if no or only one type of lifecycle transitions is given ($|e.log.lc| \leq 1$) or, if more than one type of lifecycle transitions is given, only start events are considered, events of traces are appended to the list `events` and their absolute occurrences are counted (cf. lines 1–9). Hereby, events must start simultaneously, but can finish at different times, which is reasonable when considering, e.g., the centrifuge example mentioned before (some samples might need more centrifugation time than others).

Algorithm 1: Simultaneous Activity Execution

```
   Input: (merged) log, $\varepsilon_1$, $\gamma_1$
   Result: simActivities (list of ISC candidates for Category I)
1  simActivities = dict(), absOccEvents = dict(), events = [ ]
2  for trace in (merged) log do
3      for event in trace do
4          if checkLC(event) then
5              events.append(event)
6              absOccEvents[event.lb] += 1
7          end
8      end
9  end
10 events.sort(key=ts)
11 filteredEventLabels = filter (events, absOccEvents, $\gamma_1$, $\varepsilon_1$)
12 for i in 0:len(events) do
13     ev1 = events[i]
14     if ev.lb not in filteredEventLabels then
15         continue
16     end
17     cmp = [ev1]
18     for j in i+1:len(events) do
19         ev2 = events[j]
20         if ev2.lb not in filteredEventLabels then
21             continue
22         end
23         if $|ev2.ts - ev1.last.ts| \leq \varepsilon_1$ then
24             if ev2.trace!=ev1.trace then
25                 cmp.append(ev2)
26             end
27         else
28             break
29         end
30     end
31     if len(cmp) > 1 then
32         simActivities[cmp.labels.uniq.join]. append(cmp)
33     end
34 end
```

Algorithm 1: Function `checkLC`

```
   Function checkLC(e):
       return ( $|e.log.lc| \leq 1$ or e.lc == "start");
```

In the running example, all start events, e.g., event with ID 1231, are appended to `events`. Afterwards, the events are sorted by timestamps (line 10) and filtered (cf. function `filter`). Hereby, the algorithm iterates over all events and counts the number of events with similar timestamps (lines 2–21). Similar means that timestamps must be within the given tolerance interval determined by the parameter ε_1. This constant should compensate time measurement errors produced by, e.g., a machine and should therefore be chosen accordingly, e.g., $\varepsilon_1 = 0.1$ s. Events within a range ± 0.1 second are then indicated as simultaneous. In the running example ε_1 could be set to 0 s, since the timestamps of the *start* events of *A* and *A′* are exactly the same. In the next step, function `filter` identifies false positives that happened by chance by returning event labels whose relative occurrences are above a threshold $\gamma_1 \in [0, 1]$ (lines 23–27). Choosing, e.g., $\gamma_1 = 0.9$ means that at least 90% of all observed events must have occurred simultaneously. Based on set `filteredEventLabels` only events with these labels are retained in the final ISC candidate list (`simActivities`, lines 29–43). For the running example `simActivities` $= \{A_{start}A'_{start} \rightarrow [[1231, 1671], [2231, 2671]]\}$, i.e., an ISC candidate is discovered stating that *A and A′ must be executed simultaneously.* Regarding ISC$_P$, it is likely that more than two different events occur simultaneously (imagine a third process in the running example with another event A''). This situation is covered by the algorithm.

Algorithm 1: Function `filter`

```
1  Function filter(events, absOccEvents, $\gamma_1$, $\varepsilon_1$):
2      simOccEvents = absOccEvents
3      found = False
4      for i in 0:len(events) do
5          ev1 = events[i]
6          for j in i+1:len(events) do
7              ev2 = events[j]
8              if $|ev1.ts - ev2.ts| \leq \varepsilon_1$ then
9                  if ev1.trace!=ev2.trace then
10                     found = True
11                     break
12                 else
13                     if found then
14                         found=False
15                     else
16                         simOccEvents[ev1.lb] −= 1
17                     end
18                 end
19             end
20         end
21     end
22     result = [ ]
23     for key in absOccEvents.keys() do
24         if $\frac{float(simOccEvents[key])}{absOccEvents[key]} \geq \gamma_1$ then
25             result.append(key)
26         end
27     end
28     return result;
```

4.2.2. Category II

Algorithm 2 covers the *constrained activity execution* and is the most complex algorithm of the four presented ones. The reason is that constraints can cause different effects in the log, e.g., delays w.r.t. execution time, due to various reasons. An example is a resource that may only be accessed five times a day. The result set of ISC candidates for this category consists of several subsets created by Functions `detectRegularities`, `detectExecutionConstraint`, `detectDataConstraint`. In order to cover different cases, three types of event pairs are computed (cf. line 2, function `createPairs`), i.e., (a) list *begin_end* between all begin and end events of traces in the (merged) log, (b) list *start_start* between all directly follows events, and (c) list *start_complete* for start and completion of one event if lifecycle transitions *start* and *complete* are given. In addition the time differences between the events are also stored.

For the running example
`pairs` $= \{begin_end \rightarrow \{(A, C) \rightarrow$x
$[[1231, 1235], 349m], (A, C') \rightarrow [[2231, 2675, 1768m]]\},$
$start\_start \rightarrow \{(A, A') \rightarrow$
$[[1231, 1671, 0m], [2231, 2671, 0m]], (A', B') \rightarrow$
$[[1671, 1673, 137m], [2671, 2673, 90m]], (B', B) \rightarrow$
$[[1673, 1233, 146m], [2673, 2233, 138m]], (B, C') \rightarrow$
$[[1233, 1675, 3m]], (C', C) \rightarrow [[1675, 1235, 35m]], (B, C) \rightarrow$
$[[2233, 2235, 30m]], (C, C') \rightarrow [[2235, 2675, 7m]]\}$
$start\_complete \rightarrow \{A \rightarrow$
$[[1231, 1232, 73m], [2231, 2232, 63m]], B \rightarrow$
$[[1233, 1234, 16m], [2233, 2234, 29m]], C \rightarrow$

Algorithm 2: Constrained Activity Execution

```
   Input: (merged) log, $\gamma_2$, $\varepsilon_2$
   Result: limEvents (list of ISC candidates for Category II)
1  limEvents = dict()
2  pairs = createPairs ((merged) log)
3  limEvents["regularities_begin_end"] =
   detectRegularities (pairs[0], $\gamma_2$, $\varepsilon_2$)
4  limEvents["regularities_start_start"] =
   detectRegularities (pairs[1], $\gamma_2$, $\varepsilon_2$)
5  limEvents["regularities_start_complete"] =
   detectRegularities (pairs[2], $\gamma_2$, $\varepsilon_2$)
6  limEvents["execution"] = detectExecutionConstraint
   (pairs[2], $\gamma_2$)
7  outliersStSt = detectOutliers (pairs[1])
8  outliersStCo = detectOutliers (pairs[2])
9  limEvents["data_start_start"] = detectDataConstraint
   (pairs[1], outliersStSt)
10 limEvents["data_start_complete"] =
   detectDataConstraint (pairs[2], outliersStCo)
```

[[1235, 1236, 28m], [2235, 2236, 6m]], $A' \rightarrow$
[[1671, 1672, 13m], [2671, 2672, 3m]], $B' \rightarrow$
[[1673, 1674, 124m], [2673, 2674, 1451m]], $C' \rightarrow$
[[1675, 1676, 34m], [2675, 2676, 63m]]}}

Algorithm 2: Function createPairs

```
1  Function createPairs ((merged) log):
2      begin_end = dict()
3      start_start = dict()
4      start_complete = dict()
5      for trname, trace in (merged) log do
6          begin_end[(trace[0].lb,
           trace[-1].lb)].append((trace[0], trace[-1],
           |trace[0].ts − trace[−1].ts|))
7      end
8      for trname, trace in (merged) log do
9          for i in 0:len(trace) do
10             $e_1$ = trace[i]
11             if $e_1$.lc !='start' then
12                 continue
13             end
14             for j in i+1:len(trace) do
15                 $e_2$ = trace[j]
16                 if $e_1$.lb == $e_2$.lb and $e_2$.lc=='complete' then
17                     start_complete[$e_1$.lb]
                       .append(($e_1$, $e_2$, |$e_1$.ts − $e_2$.ts|))
18                     break
19                 end
20             end
21             for j in i+1:len(trace) do
22                 $e_2$ = trace[j] if $e_2$.lc=='start' then
23                     start_start[($e_1$.lb, $e_2$.lb)]
                       .append(($e_1$, $e_2$, |$e_1$.ts − $e_2$.ts|))
24                     break
25                 end
26             end
27         end
28     end
29     return [begin_end, start_start, start_complete];
```

Function detectRegularities: This function tackles the effects of constraints that are mainly caused by Service Level Agreements (SLAs) such as *In 99% of all cases the duration of the process or a specific event may not exceed a certain time frame.*

Algorithm 2: Function detectRegularities

```
1  Function detectRegularities (pairs, $\gamma_2$, $\varepsilon_2$):
2      result = dict()
3      if !$\varepsilon_2$ and $\gamma_2$ then
4          for key, val in pairs do
5              $n$=⌊len(val) - $\gamma_2$∗ len(val)⌋
6              result[key]=reverse(sorted(val))[$n$]
7          end
8      else if $\varepsilon_2$ and !$\gamma_2$ then
9          for key, val in pairs do
10             count = 0
11             for event in val do
12                 if event[2] ≤ $\varepsilon_2$ then
13                     count += 1
14                 end
15             end
16             result[key] = $\frac{float(count)}{len(val)}$
17         end
18     else
19         for key, val in pairs do
20             count = 0
21             for event in val do
22                 if event[2] ≤ $\varepsilon_2$ then
23                     count += 1
24                 end
25             end
26             if $\frac{float(count)}{len(val)} \geq \gamma_2$ then
27                 result[key] = True
28             else
29                 result[key] = False
30             end
31         end
32     end
33     return result;
```

Two parameters control the results of this part of the algorithm. Parameter γ_2 corresponds to the percentage value present in the ISC, e.g., in the above mentioned SLA 99%. Parameter ε_2 refers to a time frame in seconds.

For checking ISC over a log, values for γ_2 and ε_2 need to be set (cf. lines 3–7). The result turns out as `true` if the ISC was not violated and `false` if the ISC was violated. For this, the occurrences of events fulfilling the given ε_2 is counted and divided by the absolute number of occurrences of this event in all instances. If this value is above the user defined threshold γ_2, the ISC is not violated. Since in this case the user already knows the ISC and just wants to check if it was violated, the analysis can be restricted to specific events.

For discovering ISC candidates, either ε_2 or the constant γ_2 must be provided. In particular, if ε_2 is given, γ_2 is computed (cf. lines 8–17) or γ_2 is given and the corresponding time frame ε_2 is determined (cf. lines 18–32). Constant γ_2 indicates how many cases need to be executed within a certain time frame (ε_2), e.g., if $\gamma_2 = 0.99$ then in 99% of all instances the execution of an activity must be carried out within a certain ε_2. For a given ε_2, the constant γ_2 is calculated as the fraction between the occurrences of events fulfilling ε_2 and the number of occurrences of this event in all instances. When threshold γ_2 is provided by the user the algorithm determines the ε_2. For each event e, $n = \lfloor \#occurrences\ of\ e * (1 - \gamma_2) \rfloor$ is computed and ε_2 corresponds to the nth maximum w.r.t all time frames for events of type e, e.g., if $\gamma_2 = 0.99$ and *#occurrences of e* $= 1000$ then $n = 10$, i.e., the 10-th maximum is taken as ε_2.

Function `detectExecutionConstraint`: The second function `detectExecutionConstraint` covers four different cases: (a) number of executed instances per day (`per_day`) (b) number of executed instances per day per resource (`per_day_res`) (c) number of executions per activity per day (`per_activity`) (d) number of executions per activity per resource per day (`per_activity_res`).

Algorithm 2: Function `detectExecutionConstraint`

```
Function detectExecutionConstraint ((merged) log, $\gamma_2$):
    result = dict()
    per_day = dict()
    per_day_res = dict()
    per_activity = dict()
    per_activity_res = dict()
    for trname, trace in (merged) log do
        per_day[trace[0].ts.day] += 1
        per_day_res[(trace[0].ts.day, trace[0].rs)] += 1
        for event in trace do
            if |event.lc|== 0 or event.lc == "start" then
                per_activity[(event.ts.day, event.lb)] += 1
                per_activity_res[(event.ts.day, event.lb,
                  event.rs)] += 1
            end
        end
    end
    result["day"] = filter (per_day, $\gamma_2$)
    result["day_resource"] = filter (per_day_res, $\gamma_2$)
    result["event"] = filter (per_activity, $\gamma_2$)
    result["event_resource"] = filter (per_activity_res, $\gamma_2$)
    return result;
Function filter (inter, $\gamma_2$):
    result = list()
    sum = dict()
    for val in inter.values() do
        sum[val] += 1
    end
    for key, val in sum do
        if $\frac{val}{len(inter)} \leq \gamma_2$ then
            result.append((key, val))
        end
    end
    return result;
```

For the running example:

- per_day = {"20-05-2019"→ 2}
- per_day_res = {["20-05-2019", $R1$] → 1, ["20-05-2019", $R3$] → 1]}
- per_activity = {["20-05-2019", A] → 2, ["20-05-2019", B] → 1, ["20-05-2019", C] → 1, ["20-05-2019", A'] → 2, ["20-05-2019", B'] → 2, ["20-05-2019", C'] → 1, ["21-05-2019", B] → 1, ["21-05-2019", C] → 1, ["21-05-2019", C'] → 1}
- per_activity_res = {["20-05-2019", A, $R1$] → 2, ["20-05-2019", B, $R2$] → 1["20-05-2019", C, $R1$] → C, ["20-05-2019", A', $R3$] → 2, ["20-05-2019", B', $R3$] → 2, ["20-05-2019", C', $R3$] → 1, ["21-05-2019", B, $R2$] → 1, ["21-05-2019", C, $R1$] → 1, ["21-05-2019", C', $R3$] → 1}.

This function can be extended and customized, e.g., by replacing "day" by a different time span or considering different combinations based on the given event attributes.[4] This part of the algorithm requires parameter γ_2 as input which describes the minimal occurrence of the cases where the ISC is fulfilled compared to all observed cases. In the running example $\mathcal{ISC}2$ represents an example for case (d). The initial results are filtered (cf. lines 17–20, Function `filter`) based on the parameter γ_2.

Function `detectDataConstraint`: Constraints can also cause delays of execution times which depend on data attributes. The method for discovering such delays is similar to the decision miner approach (cf. [15]), i.e., first of all the decision points are determined and afterwards decision rules are computed. In our case, instead of searching for decision points, we try to detect deviations of execution times. These can be understood as outliers with respect to time. Within Alg. 2, function `detectOutliers` performs this task. A large variety of outlier detection techniques exists and hence, this function can be adapted and customized depending on the given data set.

In a second step, the algorithm detects the reason for the deviation in terms of decision rules based on data elements (cf. function `detectDataConstraint`). The given event attributes of each pair of events can be used for classification and the target variable reflects in this case whether the pair is marked as an outlier or not, i.e., the result from `detectOutliers`. Again, there are a lot of classification techniques available and hence no pseudo-code is provided for this step. In Section 5, an exemplary implementation is described.

4.2.3. Category III

Algorithm 3 discovers ISC candidates that impose an *order of activities* between different processes. Such an order is reflected by pairs of start events (e_1, e_2) where e_1 and e_2 stem from different process execution logs, i.e., different processes. Only orders across different processes are of interest since the order in between one process would be an intra instance constraint and not an ISC. The algorithm creates an initial list of all cross process orders (`ordActivities`) and counts the occurrences of each pair as well as of each event label (`countEvs`, lines 1–22).

For the running example
`ordActivities` = {(A, B') → {"*pairs*" → [[1231, 1673], [2231, 2673]], "*count*" → 2},
(A', B) → {"*pairs*" → [[1671, 1233], [2671, 2233]], "*count*" → 2},
(B, C') → {"*pairs*" → [[1233, 1675], [2233, 2675]], "*count*" → 2},
(B', B) → {"*pairs*" → [[1673, 1233], [2673, 2233]], "*count*" → 2},
(C', C) → {"*pairs*" → [[2675, 2235]], "*count*" → 1},
(C, C') → {"*pairs*" → [[1235, 1675]], "*count*" → 1}}
`countEvs` = {$A \to 2, A' \to 2, B \to 2, B' \to 2, C \to 2, C' \to 2$}

Afterwards, `ordActivities` is filtered based on the assumption that, if e_1 should always be followed by e_2, the inverse order, i.e., e_2 before e_1 should not be discovered (line 23). Since logging errors are likely to occur in real-life settings, the user can specify a threshold $\kappa \in [0, 0.5)$ indicating how many inverse pairs may be observed, with $\kappa = 0$ meaning that no inverse pairs may be present in the log. Note that a value of 0.5 or higher does not make sense since this would imply that at least 50% of the observed cases may happen in the inverse order and it cannot be concluded that there has to be a constraint imposing a strict order. A second threshold $\gamma_3 \in [0, 1]$ describes the relative occurrence of the order between two events w.r.t. to the minimal occurrence of the regarded two events.

To illustrate this, consider an order e_2 after e_1, with e_1 and e_2 stemming from different processes. Assume that this order has occurred 10 times, i.e., $\#(e_1, e_2) = 10$. Let $\#e_1 = 20$ and $\#e_2 = 30$. Then $\frac{\#(e_1,e_2)}{min(\#e_1,\#e_2)} = 0.5$, i.e., only in 50% of all cases this order was

[4] Based on the analysis of the 114 real-life examples we chose "day" and "resource" for illustrating the algorithm as these event attributes appear in most of the cases.

Algorithm 3: Order of Events

```
Input: merged log, $\kappa$, $\gamma_3$
Result: ordActivities (list of ISC candidates for Category III)
ordActivities = dict()
countEvs = dict()
for tr in merged log do
    for i in len(tr) do
        $e_1$ = tr[i]
        if !checkLC($e_1$.lc) then
            continue
        end
        countEvs[$e_1$.lb] += 1
        j = 1
        while i+j < len(tr)) do
            $e_2$ = tr[i+j]
            if $e_1$.log == $e_2$.log or !checkLC($e_2$.lc) or
            $e_1$.ts == $e_2$.ts then
                j += 1
                continue
            end
            ordActivities[($e_1$.lb,$e_2$.lb)][“pairs”] .append([$e_1$,$e_2$])
            ordActivities[($e_1$.lb,$e_2$.lb)][“count”] +=1
            break
        end
    end
end
ordActivities = filter (ordActivities, countEvs, $\kappa$, $\gamma_3$)
```

observed. Therefore, it is not likely that there is an ISC e_2 *after* e_1 resulting in a false positive case. Setting, e.g., $\gamma_3 = 0.9$ filters out such false positives.

Algorithm 3: Function `filter`

```
Function filter(ordActivities, countEvs, $\kappa$, $\gamma_3$):
    result = dict()
    for tuple in ordActivities do
        if $\frac{ordActivities[tuple]['count']}{min(countEvs[tuple[0]],countEvs[tuple[1]])} \geq \gamma_3$ then
            count = ordActivities[tuple]['count']
            if reverse(tuple) not in ordActivities then
                result[tuple]= ordActivities[tuple]['pairs']
                continue
            end
            counttotal = ordActivities[reverse(tuple)]['count']
            +count
            if $\frac{float(count)}{counttotal} \leq \kappa$ then
                result[tuple]= ordActivities[tuple]['pairs']
            end
        end
    end
    return result;
```

In the running example setting κ to 0 filters out the false positives (C', C), (C, C'). Four ISC candidates remain, i.e., (A, B'), (A', B), (B', B) and (B, C'). ISC candidates (A, B') and (A', B) are discovered as A, A' are always executed simultaneously and the reverse orderings will consequently never be present. ISC candidate (B, C') represents the actual ISC that *B must be started before* C'. In this case γ_3 has no filtering effect since each of the four ISC candidates (A, B'), (A', B), (B', B) and (B, C') happens in 100% of all cases.

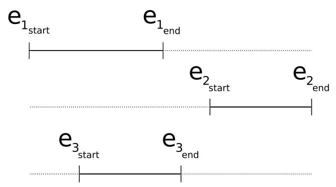


Fig. 9. Non-concurrent and concurrent activity execution and how this is reflected by the corresponding events (e_1, e_2 and e_1, e_3 respectively) in the log.

Note that the algorithm considers start events. Hence ISC such as *B must be finished before* C' are discovered based on the start events of B and C'. In order to verify that activity B must not only have started before C', but also have finished, the complete events would have to be compared in addition. This necessitates that start and complete events are present in the log. The algorithm can then be adapted such that the first component of tuples like (B, C') is the complete event of B and not the start event of B.

4.2.4. Category IV

ISC of this category impose a *non-concurrent execution of activities* and are discovered by Alg. 4. In this case, it is mandatory that activity lifecycle transitions *start* and *complete* are represented by start and complete events in the log. The reasoning behind this is, that two activities that may not be executed concurrently should not overlap in time, i.e., the start of the second event cannot take place before the first event has been completed. Fig. 9 illustrates this (cf. possible relations between time intervals, e.g., [49]): e_1 and e_2 do not overlap, i.e., are not executed concurrently while e_1 and e_3 overlap and can therefore be executed concurrently. A user can specify a fuzzy interval using the parameter ε_4 analogously to Alg. 1. Note that the difference between the timestamps of the corresponding start events may not be 0 since in this case the two events would be simultaneous, i.e., belong to Category I which is the opposite of non-concurrency.

The algorithm first of all determines all *start* and *complete* event pairs per trace that have the same labels (lines 1–11). For the running example

`pairs` $= \{$“$1b5$” $\rightarrow$ [[1231, 1232], [1671, 1672], [1673, 1674], [1233, 1234], [1675, 1676], [1235, 1236]],
“$4z7$” $\rightarrow$ [[2231, 2232], [2671, 2672], [2673, 2674], [2233, 2234], [2235, 2236], [2675, 2676]]$\}$.

Afterwards, an initial list of non-concurrent events is created. Therefore the event pairs are compared, i.e., we check whether they stem from different processes and whether the completion of the first event and the start of the second event are not overlapping with a tolerance of ε_4 (lines 12–26). For the running example with $\varepsilon_4 = 1$ second,

`nonConActivities` =
$\{(A, B') \rightarrow$ [[[1231, 1232], [1673, 1674]], [[2231, 2232], [2673, 2674]]],
$(A, C') \rightarrow$ [[[1231, 1232], [1675, 1676]], [[2231, 2232], [2675, 2676]]],
$(A', B) \rightarrow$ [[[1671, 1672], [1233, 1234]], [[2671, 2672], [2233, 2234]]],
$(A', C) \rightarrow$ [[[1671, 1672], [1235, 1236]], [[2671, 2672], [2235, 2236]]],
$(B', B) \rightarrow$ [[[1673, 1674], [1233, 1234]], [[2673, 2674], [2233, 2234]]],
$(B', C) \rightarrow$ [[[1673, 1674], [1235, 1236]], [[2673, 2674], [2235, 2236]]],
$(B, C') \rightarrow$ [[[1233, 1234], [1675, 1676]], [[2233, 2234], [2675, 2676]]],
$(C', C) \rightarrow$ [[[1675, 1676], [1235, 1236]]],
$(C, C') \rightarrow$ [[[2235, 2236], [2675, 2676]]]$\}$.

Sequentially ordered activities in the processes might lead to an initially high number of false positives, i.e., `nonConActivities` contains many false positives. In order to reduce the false positive rate, only events that reflect parallel execution of activities based on the (merged) log are taken into account. For this, the logic of the alpha miner is applied, i.e., two events e_1, e_2 are in a parallel branch if and only if e_1 is directly followed by e_2 and vice versa (cf. function `calculate_parallels`). Function

Algorithm 4: Non-concurrent Activity Execution

```
Input: merged log, $\varepsilon_4$
Result: nonConActivities (list of ISC candidates for Category IV)
pairs = dict()
for tr in merged log do
    for $e_1$ in tr do
        for $e_2$ in tr do
            if $e_1$.lb == $e_2$.lb and $e_1$.lc == "start" and $e_2$.lc == "complete" then
                pairs[tr.lb].append([$e_1$, $e_2$])
                break
            end
        end
    end
end
nonConActivities = dict()
for tr,events in pairs do
    for i in 0:len(events) do
        $e_1$ = events[i]
        for j in i+1:len(events) do
            $e_2$ = events[j]
            if $e_2$[0].ts < $e_1$[0].ts then
                swap($e_1$,$e_2$)
            end
            if $e_1$.log != $e_2$.log and $e_2$[0].ts − $e_1$[1].ts ≥ $\varepsilon_4$ and $e_2$[0].ts − $e_1$[0].ts != 0 then
                nonConActivities[($e_1$.lb,$e_2$.lb)] .append([$e_1$, $e_2$])
            end
        end
    end
end
nonConActivities = filter (calculate_parallels ((merged) log), nonConActivities)
Function filter(parallel, nonConActivities):
    result = dict()
    for key,value in nonConActivities do
        if key in parallel then
            result[sorted(key)].append(value)
        end
    end
    return result;
```

Algorithm 4: Function `calculate_parallels`

```
Function calculate_parallels((merged) log):
    labels = dict()
    for tr,events in (merged) log do
        for i in 0:len(events) do
            $e_1$ = events[i]
            if $e_1$.lc != "start" then
                continue
            end
            for j in i+1:len(events) do
                $e_2$ = events[j]
                if $e_2$.lc == "start" then
                    labels[($e_1$.lb, $e_2$.lb)]+= 1
                    break
                end
            end
        end
    end
    parallels = set()
    for lb in labels do
        if reverse(lb) in labels then
            parallels.add(lb)
        end
    end
    return parallels;
```

`filter` takes the initial list `nonConActivities` and the as parallel identified events as input in order to reduce the number of false positives in the final result (line 27).

In the running example, C and C' are identified as parallel and the final result after filtering is
`nonConActivities` =
$\{(C', C) \rightarrow [[[1675, 1676], [1235, 1236]]],$
$(C, C') \rightarrow [[[2235, 2236], [2675, 2676]]]\}$,

The resulting ISC candidate turns out as *C and C′ may not be executed concurrently*.

5. Implementation

This section targets Artifact 6, the prototype for the ISC discovery approach presented in Section 4.

The prototype is implemented in Python 3, following common guidelines for object oriented programming, in order to increase accessibility and maintainability of the code as well as performance. The prototype is provided as a web service available at http://isc-mining.wst.univie.ac.at.

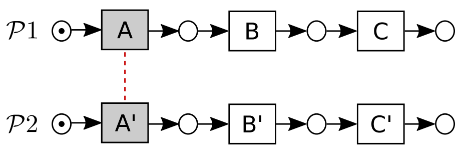


Fig. 10. Visualization of the result for category 1 for the running example.

In addition to the ISC candidate discovery one process model per log is mined using the Heuristics Miner [23]. For visualizing the process models and the results of the ISC discovery algorithms GraphViz[5] is used. Each ISC candidate is represented by a gray filled node which, in the process spanning setting is connected by a red dashed line to at least one other gray filled node. The ISC for Category I in the running example (cf. Fig. 7) is visualized in Fig. 10: Activities A and A' are colored gray and connected by a red line. This indicates the ISC candidate *A and A′ are executed simultaneously.* In the case of Alg. 2 the nodes are enriched with decision rules. The results of each ISC discovery algorithm are additionally available as JSON files, which can be easily read and further processed by technical experts.

All parameters γ_i, $i = 1, 2, 3$, $\varepsilon_j, j = 1, 2, 4$, κ, *minerabs* and *minerrel* (the latter two are for the Heuristics Miner) are adjustable by the user. For Alg. 2 the user can additionally provide a list of attributes that should be used for classification within function `detectDataConstraint`. If such a list is not given, all event attributes except for *concept name, lifecycle transition*, the *merging attribute*, i.e., the attribute that was used for creating the merged log, as well as *uuids* are used for the classification. The first two attributes are already used during the outlier detection and therefore omitted and the latter two are per definition unique, i.e., do not contribute to finding decision rules since every instance would be classified into a separate branch. Moreover, *timestamps* have also proven to be inappropriate for classification, cf. [10].

5 https://www.graphviz.org/.

For the outlier detection within function `detectOutliers`, the z-score, which is suitable for normally distributed data, is applied. For the classification task, the decision tree classifier of the Python package scikit-learn (cf. [50]) is used in combination with the Gini index since this is not as computationally intense as the entropy criterion while nearly the same classification performance can be achieved (cf., e.g., [51]). The maximum depth of the decision tree is chosen as the number of attributes that can be used for classification. The minimum number of samples required to split an internal node is set to 2. Since, only paths resulting in the class "outlier = true" are of interest, only these are displayed in the visualization.

From a theoretical point of view, the implicit assumption was made that event labels are unique within and across processes. From a technical point of view, a unique process id can be added to every event label in order to ensure uniqueness.

6. Evaluation

For evaluating the approach, artificial process execution logs as well as real-life process execution logs are used. The log files and evaluation results are available at http://bit.ly/2lztLv6.

For each of the presented algorithms,

$$precision = \frac{\#true\ positives}{\#(true\ positives + false\ positives)}$$

and

$$recall = \frac{\#true\ positives}{\#(true\ positives + false\ negatives)}$$

are computed with $precision, recall \in [0; 1]$. Precision corresponds to the fraction between ISC candidates that are correctly retrieved and the number of all retrieved ISC candidates. Recall is the fraction between ISC candidates that are correctly retrieved and the sum of correctly retrieved ISC candidates and the missing ones. True positive results correspond to discovered ISC candidates representing actual ISC. False positive results are discovered ISC candidates that do not represent actual ISC. False negative results correspond to actual ISC that are not represented by a discovered ISC candidate.

The evaluation is carried out on a virtual machine with 2 Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20 GHz and 16 GB RAM. As stated in Section 5, the resulting ISC candidates are visualized on top of process models as gray filled nodes in combination with either red dashed lines (in the case that the ISC is process spanning) or decision rules next to the nodes (for Alg. 2).

For applying the ISC candidate discovery algorithms presented in Section 4 different parameters have to be set. In this section, each algorithm is run for each data set with a parameter setting that leads to an "optimum" result with respect to precision and recall. Note that the parameter settings are not necessarily unique, i.e., other settings might lead to the same precision and recall. We discuss the parameter value choices based on certain characteristics of the logs (e.g., high regularity of the artificial log). Appendix presents tables where precision and recall are summarized for different parameter settings in the algorithms for each of the logs. Based on the findings of this section and the tables we draw conclusions on how to set parameters based on certain log characteristics for the different algorithms in Section 7.

6.1. *Artificial example*

The artificial logs are generated based on the three process models depicted in Fig. 11. $\mathcal{P}1$ describes the steps of designing, printing and dispatching flyers, $\mathcal{P}2$ the corresponding bill handling process for different orders and $\mathcal{P}3$ the creation and dispatch of photo posters. The synchronization of processes $\mathcal{P}1, \mathcal{P}2, \mathcal{P}3$ obeys the following ISC which are reflected in the corresponding logs:

$\mathcal{ISC}1$ Finished orders of one day are delivered to the post office simultaneously in the evening. ($\mapsto$ Category I)

$\mathcal{ISC}2a$ All print jobs must be completed within 10 min in at least 95% of all cases. ($\mapsto$ Category II, time constraint)

$\mathcal{ISC}2b$ Printer 1 may only print 10 times per day. ($\mapsto$ Category II, absolute number of executions)

$\mathcal{ISC}2c$ If the filling amount of the printing ink is below 5%, it has to be refilled before further jobs may be printed. ($\mapsto$ Category II, data constraint)

$\mathcal{ISC}3$ If a flyer or poster order is received $\mathcal{P}2$ is started. ($\mapsto$ Category III)

$\mathcal{ISC}4$ Flyers and posters as well as bills and posters cannot be printed concurrently on one printer since they require a different paper format. ($\mapsto$ Category IV)

$\mathcal{ISC}1$ to $\mathcal{ISC}4$ are formulated based on the catalog of real-world ISC in [16]. Moreover, we make sure that $\mathcal{ISC}1$ to $\mathcal{ISC}4$ cover all ISC categories determined in Section 3. In the case of Category II for each of the presented functions one ISC is provided.

The logs are generated using the techniques described in [52] resulting in 900 instances of $\mathcal{P}1$ and $\mathcal{P}3$ and 1800 instances of $\mathcal{P}2$. The generation starts with modeling processes $\mathcal{P}1$, $\mathcal{P}2$ and $\mathcal{P}3$ (cf. Fig. 11) using Signavio[6] and afterwards transforming them into executable code through the workflow engine CPEE [53][7] which also ensures the compliance with $\mathcal{ISC}1-4$. In detail, every time an activity is executed, a notification is sent from the process execution engine via HTTP to a separate logging service, which stores the information of the event directly in XES format. Per instance one XES log is produced, i.e., in order to receive the complete process execution log for one process, all corresponding instances are merged into a single process execution log by appending the traces iteratively. The information of an event is provided directly by the CPEE, i.e., the activity label, the resource executing the activity, for example which printer, and the duration of the activity, which is randomly sampled from a given interval.

Since the injected artificial ISC are known, the retrieved ISC candidates can be easily validated.

The parameters for the Heuristics Miner are set to *minerabs* = 100 and *minerrel* = 0.3, i.e., only paths between activities occurring with an absolute frequency of at least 100 times and a relative frequency of at least 0.3 are considered. In order to demonstrate the effects of the parameters, precision and recall are computed for various parameter settings and summarized in tables (cf. Appendix).

6.1.1. *Category I*

Fig. 12 displays the results for Alg. 1 with $\gamma_1 = 0.95$, $\varepsilon_1 = 0.1$ s, i.e., it depicts the process models annotated with ISC candidates representing activities that are executed simultaneously. The red dashed edges imply that the gray filled nodes `deliver flyer`, `deliver poster`, `deliver bill` are executed simultaneously. Hence, $\mathcal{ISC}1$ (*Finished orders of one day are delivered to the post office simultaneously in the evening.*) is discovered and no false positives are contained in the result with this parameter setting. Precision and recall consequently both equal 1. How the parameters γ_1 and ε_1 affect precision (p) and recall (r) is depicted in Table A.2. It can be deduced that the ISC is not discovered for all parameter settings since for $\gamma_1 = 1$ the recall is only $\frac{1}{3}$. Moreover, activities do not happen close after each other as ε_1 needs to be at least set to 100 s to affect the precision.

6 https://academic.signavio.com/.
7 http://cpee.org/.

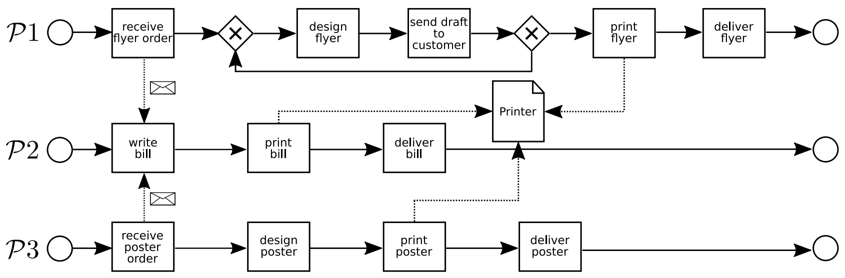


Fig. 11. Artificial example: Process models with ISC.

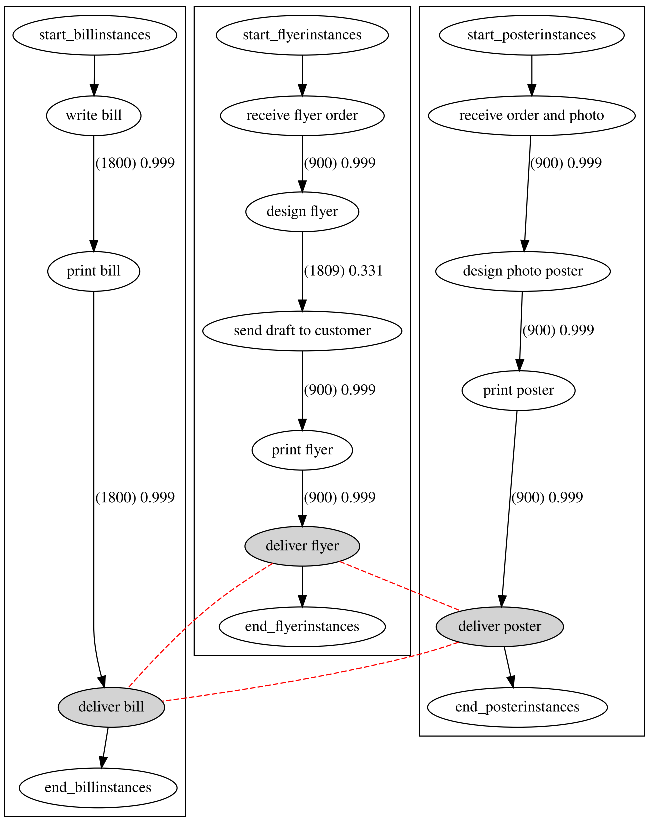


Fig. 12. Result for artificial example for Algorithm 1 with $\gamma_1 = 0.95, \varepsilon_1 = 0.1$ s.

6.1.2. Category II

In the following, we summarize the results for the different functions of Algorithm 2.

Function `detectRegularities`**:** For the artificial log files we know $\mathcal{ISC}2a$ ("All print jobs must be completed within 10 min in at least 95% of all cases. ") beforehand. Therefore, we can demonstrate how to use this function for checking ISC. Hence, no table containing precision and recall for different parameter settings is provided.

In order to check $\mathcal{ISC}2a$, we set $\gamma_2 = 0.95, \varepsilon_2 = 600$ s. With this setting, for begin and end events of instances, `detectRegularities` does not deliver any results, which is correct.

For directly follows activities, two regularities are discovered between `print bill` and `deliver bill` resp. `deliver bill` and `deliver poster`. This indicates that as soon as `print bill` starts, bills are delivered the same day and they are delivered in conjunction with posters which corresponds to the results of Alg. 1. Consequently, this ISC candidate is present due to $\mathcal{ISC}1$.

Between start and complete of same event types, all `print` events are discovered as ISC candidates with the given parameter setting. Consequently, $\mathcal{ISC}2a$ is fulfilled by the underlying log files. Moreover, regularities for `write bill`, `deliver bill`, `receive flyer order`, `deliver flyer`, `receive order and photo` and `deliver poster` are discovered. These are false positives and can be explained by, e.g., an automatic system that covers the incoming orders within a regular time frame of less than 600 s (`receive flyer order`, `receive order and photo`) and that `deliver` activities do not take longer than 600 s in 95% of all cases.

Function `detectExecutionConstraint`**:** In this case $\mathcal{ISC}2b$ (*Printer 1 may only print 10 times per day.*) should be discovered. Setting γ_2 to 0.99 or 0.95 does not deliver any results. By setting γ_2 to 0.8 the following four ISC candidates are discovered

1. *On 42 days 40 instances were started.*
2. *On 42 days 20 instances with* `receive flyer order` *were started.*
3. *On 42 days 40 instances with* `write bill` *were started.*
4. *On 42 days 20 instances with* `receive order and photo` *were started.*

These candidates are false positives and trace back to the fact that the underlying artificial logs obey additional ISC, in this case $\mathcal{ISC}3$ which states that whenever a flyer or poster order is received the billing process is started immediately.

With $\gamma_2 = 0.5$ besides these ISC candidates it is discovered that *On 41 days 10 instances with* `print flyer` *were started by* `printer1` . This ISC candidate corresponds to $\mathcal{ISC}2b$ and precision is in this case $\frac{1}{5}$ while recall equals 1.

Function `detectDataConstraint`**:** For this function no parameters are required.

First of all the set of start and complete pairs, i.e., list *start_complete* is viewed. Fig. 13 displays the corresponding process models enriched with the ISC candidates. The decision rules are displayed next to the gray nodes and can be summarized as *Whenever the filling amount of the printing ink is below 5.5, a delay between the start and end of an* `print` *activity occurs.* This reflects $\mathcal{ISC}2c$ and precision as well as recall both equal 1. Note that the discovered ISC candidates could only be intra instance and not instance spanning constraints as the connection between the `print` activities which access the common shared resource, is not explicitly visualized. This can only be resolved by inspecting the corresponding JSON files.

For directly follows activities, i.e., list *start_start* two ISC candidates are discovered:

Whenever the resource of `print bill` is `printer1` and

- `filling amount of print bill` $\leq$ 15.5 and `filling amount of print bill` $>$ 4 or
- `filling amount of print bill` $>$ 15.5 and `filling amount of print bill` $>$ 95.0

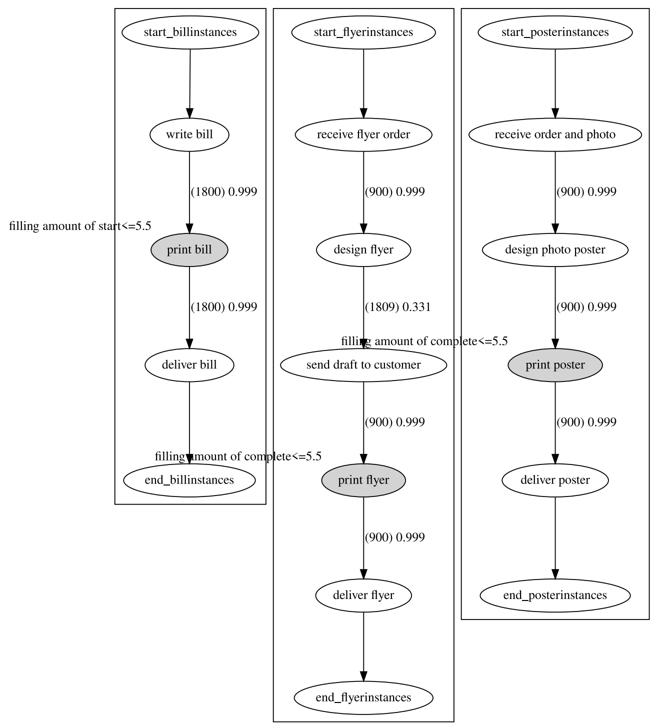


Fig. 13. Result for artificial example Algorithm 2 using `detectDataConstraint` for `start` and `complete` Pairs.

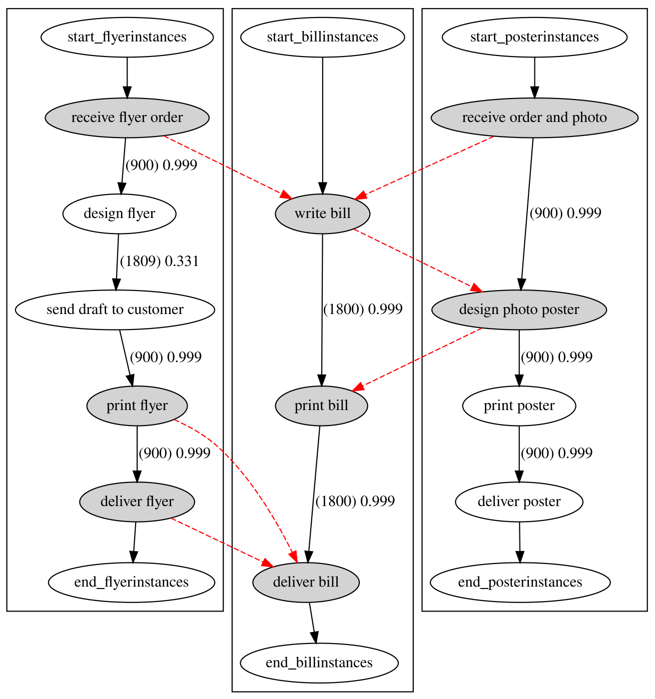


Fig. 14. Result for the artificial example for Algorithm 3 with $\gamma_3 = 0.99, \kappa = 0.05$.

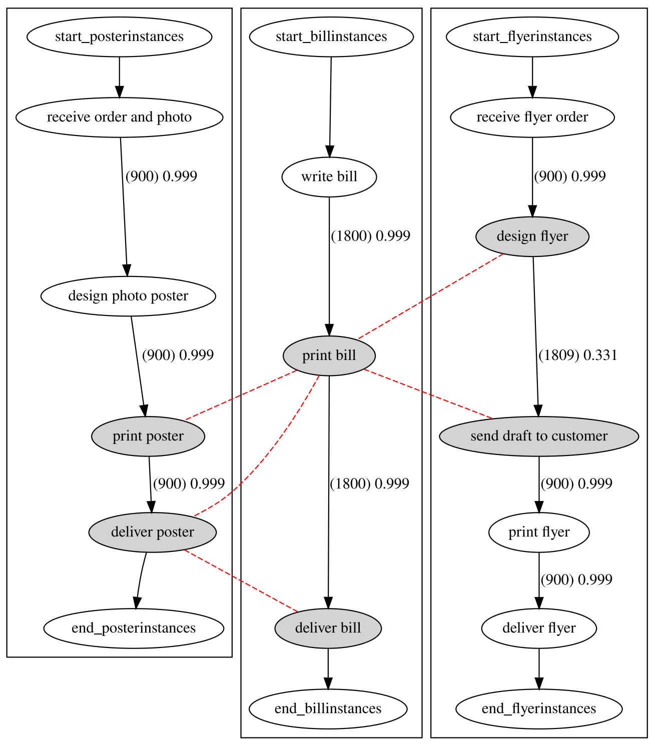


Fig. 15. Result for artificial example for Algorithm 4 with $\varepsilon_4 = 0$ s.

then there is a delay between `print bill` and `print poster`.

These are not ISC and inspecting the relevant traces and events reveals the reason why they are discovered. Each of the `print bill` activities is executed in the morning and afterwards all other `print bill` and `print flyer` activities are executed. This results in a delayed execution of all `print` events with a different format, thus resulting in an unusually long time difference between these events, which marks them as an outlier.

6.1.3. *Category III*

Table A.3 contains the precision and recall scores for various parameter settings. It can be deduced that for these process execution logs γ_3 is more pivotal than κ. The best precision and recall scores, i.e., $p = \frac{1}{3}, r = 1$, are achieved with $\gamma_3 = 0.99$ and choosing an arbitrary κ. The corresponding annotated process models are displayed in Fig. 14. Again ISC candidates are represented as gray filled nodes and red dashed lines across the process models. It can be seen that whenever an order arrives, activity `write bill` is issued, i.e., $\mathcal{ISC}3$ is clearly supported by the results. The order between `write bill` ($\mathcal{P}2$) and `design photo poster` ($\mathcal{P}3$) that does not refer to an ISC can be recognized as well as orders between `print flyer` and `deliver bill` resp. `deliver flyer` and `deliver bill` as well as `design photo poster` and `print bill`. These are false positives and present due to the regular nature of the process execution logs.

6.1.4. *Category IV*

In this case, the non-concurrent execution of activities is targeted. Table A.4 displays the recall and precision scores for different values for ε_4.

Fig. 15 displays the result of Alg. 4 with $\varepsilon_4 = 0$ s. Five ISC candidates are discovered, `design flyer` not concurrent with `print bill`, `send draft to customer` not concurrent with `print bill`, `deliver poster` not concurrent with `print bill` and `deliver poster` not concurrent with `deliver bill`. It can be deduced that bills and posters may not be printed concurrently, i.e., $\mathcal{ISC}4$ is partly discovered. The remaining four candidates are false positives whereas the last ISC candidate can be explained by the fact that posters take longer for printing than bills. Consequently poster orders are delivered one day after their corresponding bills.

6.1.5. Conclusion

For Category I and III a recall of 1 can be achieved, i.e., all injected ISC are represented by an ISC candidate. However, for Category IV we can only receive a recall of $\frac{1}{2}$. A precision of 1 is achieved for Alg. 1, whereas for Alg. 3 precision is in the optimum case $\frac{1}{3}$ and for Alg. 4 the precision score is $\frac{1}{5}$. Consequently, the discovery of false positive ISC candidates could not be entirely avoided. This might happen as the artificially generated processes might contain regularities and hence are not completely random. Moreover, ISC that are from a different category can have affects on the results for other categories. For Alg. 2 it was demonstrated how to check ISC if they are known beforehand and all ISC could be resolved.

6.2. Real-life example — manufacturing domain

The process execution logs for this real-life example stem from the manufacturing domain and consist of nine process models (cf. [54]). The processes describe the production and quality assurance of valve lifters. For evaluating the results a domain expert has been consulted. According to the expert, only ISC of Category III are present in the logs. Thus precision and recall are only calculated with several parameter settings for Alg. 3. For the other categories the recall cannot be determined as the denominator would evaluate to 0. However, we can determine whether the approach delivers false positives or not.

The parameters for the Heuristics Miner are set to $minerabs = 1$ and $minerrel = 0.3$.

6.2.1. Category I

As no ISC from this category are contained in the example, we only want to determine how the parameters affect the number of false positives. Starting with $\gamma_1 = 1$ and $\varepsilon_1 \in \{0, 0.1, 1\}$ s results in no false positives. Setting ε_1 to 10 s, results in a high number of false positives. These observations differ from the artificial example because the starts of the events occur way closer to each other than in the artificial example. The parameter ε_1 has indeed an effect on the false positive rate. This finding also holds for $\gamma_1 \in \{0.99, 0.95\}$. For $\gamma_1 \in \{0.8, 0.6, 0.4, 0.2\}$ one to two false positives are even present for $\varepsilon_1 = 1$ s.

6.2.2. Category II

Function `detectRegularities`**:** In contrast to the artificial example the ISC are not known beforehand and must be discovered from scratch. For this either γ_2 or ε_2 must be provided and each combination between begin and end events of instances, between start and start of directly follows events as well as start and complete of one event is marked as ISC candidate.

An application scenario demonstrating how to use γ_2 to determine the production quality is as follows. According to the domain expert, the duration of `Fetch3`, i.e., between start of `Fetch3` and complete of `Fetch3`, must last at least three minutes to ensure proper production of a valve lifter. This is not an ISC since this condition must hold per instance. However, in order to discover, e.g., quality issues like determining whether in 95% of all instances the time range (ε_2) for ensuring a proper production is met, γ_2 can be used. In this case it can be set to 0.05, in order to calculate the ε_2 for 95% of occurrences. This ε_2 can be compared to the demanded ε_2 of 3 min provided by the domain expert. Such a condition spans all instances and is therefore regarded as an ISC. The ε_2 determined by the approach with $\gamma_2 = 0.05$ for `Fetch3` is 238.605 s, i.e., in at least 95% of all cases the production quality is as desired.

For $\gamma_2 \in \{1, 0.8, 0.6\}$ **function** `detectExecutionConstraint` does not detect any execution constraints while with $\gamma_2 = 0.4$ one execution constraint (*On 2 days 4 instances were started.*) and with $\gamma_2 = 0.2$ 15 additional ones are discovered. According to the domain expert there should not be any execution constraints within the log files.

No results are retrieved for **function** `detectDataConstraint` which is correct according to the domain expert.

6.2.3. Category III

According to the domain expert, there should be a red dashed line connecting all process models such that each process is connected with another process. Fig. 16 contains the ISC candidates for Alg. 3 with $\gamma_3 = 1$ and $\kappa = 0$.

According to Table A.5 which contains the precision and recall scores for different parameter settings, the precision for this case is $\frac{3}{4}$ and recall $\frac{3}{7}$. The false positive result (`Check State6` before `Measure with MicroVu9`) emerges due to physical conditions within the production process since measuring can only be carried out if the production has finished and the check for the clamp opening has been done.

6.2.4. Category IV

With $\varepsilon_4 \in \{0, 0.1\}$ s, we discover that `Check State1` and `Machining8` must not be executed concurrently. Even though the two processes containing these activities are executed in parallel, the fact that `Check State1` and `Machining8` are never executed concurrently is random, according to the domain expert. This is therefore one false positive result. With $\varepsilon_4 = 1$ s or higher no false positive results are discovered.

6.2.5. Conclusion

For Category I and IV it is possible to avoid false positives with the optimal parameters while for Category III the precision evaluates in the optimal case to $\frac{3}{4}$. For Alg. 2 it could be demonstrated how to use the algorithm for, e.g., monitoring quality during the production.

6.3. Real-life example — higher education domain

The second real-life case stems from the higher education domain (cf. [55]), describes the activities of students within one semester course and consists of one process, i.e., only one log file. Therefore, Categories III and IV are not present as they only appear in process spanning settings. For evaluating the results a domain expert has been consulted and again precision and recall for Alg. 1 are calculated and displayed for various parameter settings.

The parameters for the Heuristics Miner are set to $minerabs = 12$ and $minerrel = 0.3$.

6.3.1. Category I

Table A.6 summarizes the precision and recall scores for Alg. 1. When setting the parameters to $\gamma_1 = 1$ and $\varepsilon_1 = 0$, 12 ISC candidates are discovered and Fig. 17 depicts the results. All gray activities like, e.g., the `kick-off meeting` or the `written exam` have happened for 100% of all students at exactly the same time. An example of an ISC candidate would therefore be that *The written exam took place for all students simultaneously.*. All ISC candidates are indeed ISC according to the expert and were logged by an automatic system. It can be seen that the `upload` activities do not happen simultaneously as each student should work and resp. upload on his or her own which can happen at different times.

With $\gamma_1 = 0.95$ an interesting ISC candidate emerges that is a mixture of an intra instance and instance spanning constraint. The algorithm discovers that `3. submission deadline` and `final project meeting` are executed simultaneously. In fact, these activities have the same timestamp for almost all instances.

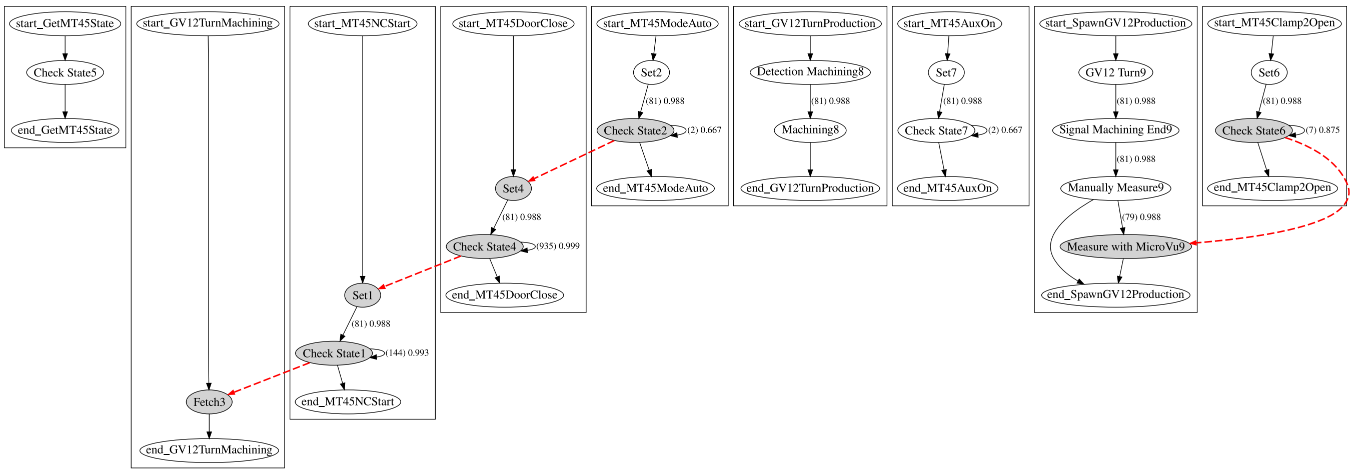


Fig. 16. Result for Algorithm 3 for the real-life example from the manufacturing domain with $\gamma_3 = 1,\ \kappa = 0$.

Table 1
Durations of algorithms based on the evaluation log files.

Log file	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4
Artificial example, 30 646 events, 11.2 MB	41.091 s	25.633 s	6.411 s	36.806 s
Manufacturing example, 2546 events, 11.58 MB	4.839 s	8.2 s	4.558 s	29.11 s
Higher education example, 8913 events, 2.8 MB	14.266 s	6.757 s	N/A	N/A

Table A.2
Precision and recall for the artificial example for Algorithm 1 with different parameter combinations.

	$\gamma_1 = 1$	$\gamma_1 = 0.95$	$\gamma_1 = 0.8$	$\gamma_1 = 0.6$	$\gamma_1 = 0.4$	$\gamma_1 = 0.2$
$\varepsilon_1 = 0$ s	p = 1 r = $\frac{1}{3}$	p = 1 r = 1	p = 1 r = 1	p = 1 r = 1	p = 1 r = 1	p = 1 r = 1
$\varepsilon_1 = 0.1$ s	p = 1 r = $\frac{1}{3}$	p = 1 r = 1	p = 1 r = 1	p = 1 r = 1	p = 1 r = 1	p = 1 r = 1
$\varepsilon_1 = 1$ s	p = 1 r = $\frac{1}{3}$	p = 1 r = 1	p = 1 r = 1	p = 1 r = 1	p = 1 r = 1	p = 1 r = 1
$\varepsilon_1 = 10$ s	p = 1 r = $\frac{1}{3}$	p = 1 r = 1	p = 1 r = 1	p = 1 r = 1	p = 1 r = 1	p = 1 r = 1
$\varepsilon_1 = 100$ s	p = 1 r = $\frac{1}{3}$	p = 1 r = 1	p = 1 r = 1	p = $\frac{3}{26}$ r = 1	p = $\frac{3}{45}$ r = 1	p = $\frac{3}{45}$ r = 1

Table A.3
Precision and recall for the artificial example for Algorithm 3 with different parameter combinations.

	$\gamma_3 = 1$	$\gamma_3 = 0.99$	$\gamma_3 = 0.95$	$\gamma_1 = 0.8$	$\gamma_3 = 0.6$	$\gamma_3 = 0.4$	$\gamma_3 = 0.2$
$\kappa = 0$	p = 1 r = 0	p = $\frac{1}{3}$ r = 1	p = $\frac{2}{7}$ r = 1	p = $\frac{1}{4}$ r = 1	p = $\frac{1}{4}$ r = 1	p = $\frac{1}{5}$ r = 1	p = $\frac{1}{6}$ r = 1
$\kappa = 0.05$	p = 1 r = 0	p = $\frac{1}{3}$ r = 1	p = $\frac{2}{7}$ r = 1	p = $\frac{1}{4}$ r = 1	p = $\frac{1}{4}$ r = 1	p = $\frac{1}{5}$ r = 1	p = $\frac{1}{6}$ r = 1
$\kappa = 0.1$	p = 1 r = 0	p = $\frac{1}{3}$ r = 1	p = $\frac{2}{7}$ r = 1	p = $\frac{1}{4}$ r = 1	p = $\frac{1}{4}$ r = 1	p = $\frac{1}{5}$ r = 1	p = $\frac{1}{6}$ r = 1
$\kappa = 0.2$	p = 1 r = 0	p = $\frac{1}{3}$ r = 1	p = $\frac{2}{7}$ r = 1	p = $\frac{1}{4}$ r = 1	p = $\frac{1}{4}$ r = 1	p = $\frac{1}{5}$ r = 1	p = $\frac{1}{6}$ r = 1
$\kappa = 0.3$	p = 1 r = 0	p = $\frac{1}{3}$ r = 1	p = $\frac{2}{7}$ r = 1	p = $\frac{1}{4}$ r = 1	p = $\frac{1}{4}$ r = 1	p = $\frac{1}{5}$ r = 1	p = $\frac{1}{6}$ r = 1
$\kappa = 0.4$	p = 1 r = 0	p = $\frac{1}{3}$ r = 1	p = $\frac{2}{7}$ r = 1	p = $\frac{1}{4}$ r = 1	p = $\frac{1}{4}$ r = 1	p = $\frac{1}{5}$ r = 1	p = $\frac{2}{13}$ r = 1

However, as most process mining algorithms do only consider the workflow and not the timestamps it is not possible to discover that there should be a parallel branch because from the workflow perspective `3. submission deadline` is always followed by `final project meeting`. Therefore, it cannot be deduced based on the model that there is an intra instance constraint that these two activities must be executed simultaneously. According to the expert this is not an ISC even though it was logged in that way by the system.

6.3.2. *Category II*

Function `detectRegularities`: In this case we test different values for $\gamma_2 \in \{1, 0.8, 0.6, 0.4, 0.2\}$ and determine the corresponding limit ε_2. For begin and end events of instances regularities between `Kick-off meeting` and `Final mark`, `Kick-off meeting` and `Final project meeting`, `Kick-off meeting` and `Run unit test phase 2` as well as `Kick-off meeting` and `Run unit test phase 3`. These are the discovered ISC candidates and it can be deduced that not every instance finishes with the

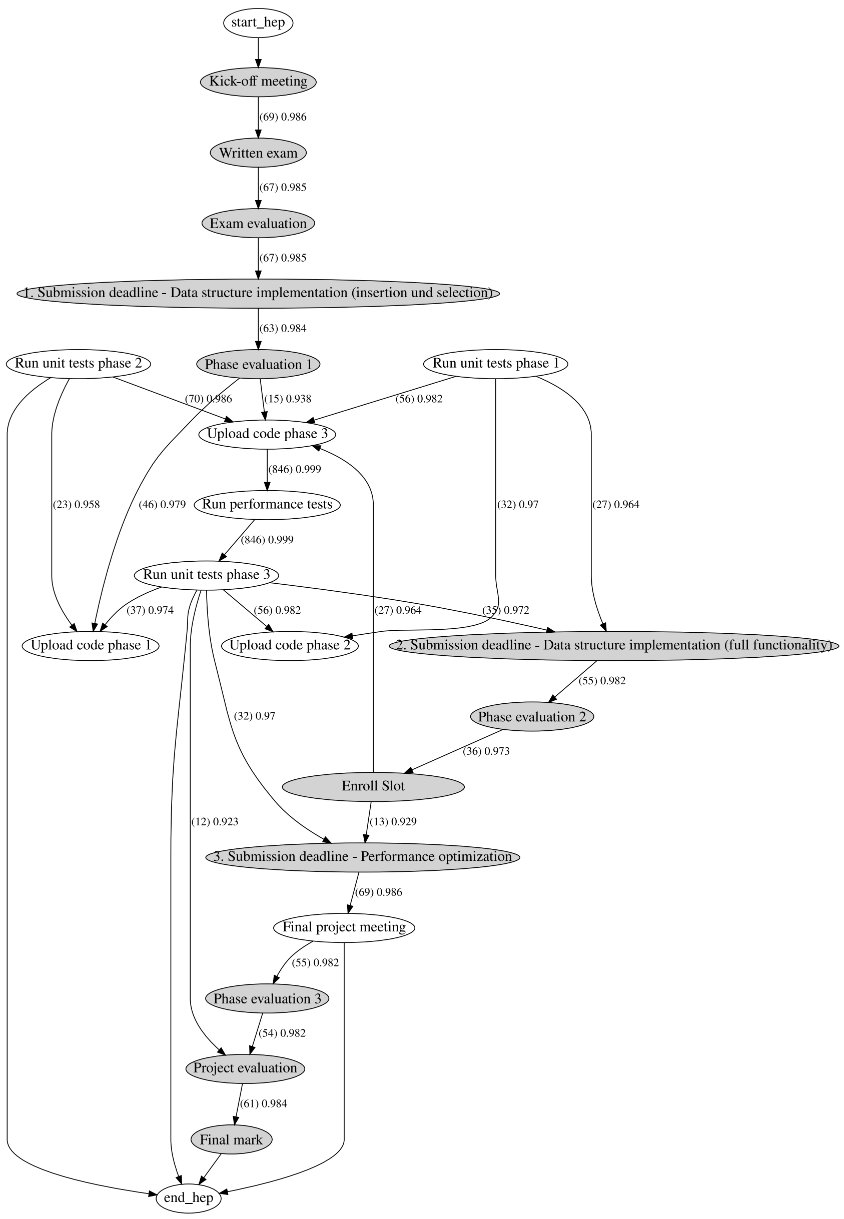


Fig. 17. Result for Algorithm 1 for the Real-life Example from the Higher Education Domain with $\gamma_1 = 1$ and $\varepsilon_1 = 0$.

same event. Each γ_2 delivers the same ε_2 for the first two ISC candidates. This finding corresponds to the duration of the course, whereas the first limit describes the duration until the grading and the second one the duration until the last activity that a student has to take, the final project meeting. These are constraints for one student but they span all instances as they apply to each student. The third ISC candidate also has the same limit for each γ_2 but like the last ISC candidate, which has different limits depending on γ_2, it is not an ISC according to the domain expert.

For directly follows events, again different values for γ_2 are tested and 31 out of 73 ISC candidates have the same limit for each γ_2. Again this can be explained by the timespans between deadlines which are the same for each student.

As there are no complete events in the log file, list start_complete is empty.

The second function, `detectExecutionConstraint` discovers one ISC candidate *On 1 day 69 instances were started* which is present due to the simultaneous execution of this event like discovered within Category I.

For **function** `detectDataConstraint` no results are retrieved which is correct according to the domain expert.

6.3.3. Conclusion

For this real-life example precision as well as recall equal 1 for Category I with suitable parameters. Moreover, compared to the previous examples, the precision score remains at a high level when loosening the parameters. For Category II

7. Discussion

Requirements on logs: Events must have unique *concept names* and *timestamps*. Traces or events should have a *unique case identifier*. For Alg. 4 *lifecycle transitions start* and *complete* are mandatory. Even though the requirements on the logs are kept to a minimum and are covered by the XES standard lifecycle transitions, they might not be fulfilled for every real-life case. Then preprocessing of the log files might become necessary, e.g., inserting missing timestamps or amending lifecycle transitions.

Table A.4
Precision and recall for the artificial example for Algorithm 4 with different parameter combinations.

	Precision	Recall
$\varepsilon_4 = 0$ s	$\frac{1}{5}$	$\frac{1}{2}$
$\varepsilon_4 = 0.1$ s	$\frac{1}{5}$	$\frac{1}{2}$
$\varepsilon_4 = 1$ s	$\frac{1}{5}$	$\frac{1}{2}$
$\varepsilon_4 = 5$ s	$\frac{1}{5}$	$\frac{1}{2}$
$\varepsilon_4 = 10$ s	$\frac{1}{5}$	$\frac{1}{2}$

Interplay between parameters and logs: As demonstrated by the evaluation, the parameters in Algorithms 1–4 allow the user to carry out the analysis in a flexible way and to adjust it to the underlying process execution log files. Parameters $\gamma_i, i = 1, 2, 3$ represent the relative occurrences of observations. If it is known that the underlying log file(s) are likely to contain errors, e.g., describe settings in which the logging is done after the work day like it is often the case in the medical domain, these parameters should be chosen smaller in order to detect ISC candidates. For Alg. 1, ε_1 controls small logging errors and affects the results whenever event executions are close together, i.e., if the underlying log contains short durations between events ε_1 should be chosen small.

Quality of Process Mining Results: As mentioned in Section 6.3, the visualization depends on to the process mining results and if the model is not correctly mined, ISC candidates might not be visualized correctly. This is, e.g., the case for the real-life example from the higher education domain in Section 6.3 where two events that should be in a parallel branch were not determined to be parallel. Such issues can be resolved by considering the JSON files.

Runtime: An algorithm should not only be precise and have a high recall in order to be useful. Furthermore, in order to be applicable in practice, it should be reasonably fast. Therefore, average durations of each algorithm are measured and summarized in Table 1.

Each algorithm was run ten times for $\gamma_1 = 0.95, \varepsilon_1 = 0, \gamma_2 = 0.95, \gamma_3 = 0.99, \kappa = 0, \varepsilon_4 = 0$ and the average duration over all these runs is taken. The computation includes the parsing of the XES file(s), the ISC algorithm computations, the discovery of the process model(s) as well as the drawing. It can be seen that the duration in general depends on the number of events. For Alg. 2, the speed is also affected by the file size, i.e., the amount of event attributes. We did not measure in detail how the parameters affect the durations but the lower a γ and the higher an ε the more observations need to be tested during the filtering which can increase the duration.

8. Conclusion and outlook

ISC play a crucial rule for companies from various domains. Violating ISC can cause severe consequences such as fines or quality problems. Consequently, checking as well as discovering ISC from process execution logs plays an essential part in a digitalized compliance management. Hence, in the introduction, we stated the following research questions:

RQ1 How to design an automatic algorithm for discovering ISC candidates?
In this paper, we have provided the first automatic ISC candidate discovery approach. It is based on a novel categorization of ISC and the evaluation shows that the approach is precise and has a high recall, i.e., in most of the cases all ISC could be discovered.

RQ2 Which characteristics and requirements must process execution logs meet to enable the discovery of ISC candidates?
Requirements on the logs are kept at a minimum and are usually provided by Process Aware Information Systems. Thus the approach is applicable to a wide range of logs.

RQ3 How can the ISC mining algorithm control varying quality of process execution logs?
For each of the four presented algorithms a user can set different parameters making the approach adjustable to the underlying log. We showed how the parameters can be used to deal with, for example, logging errors and to control varying durations between activity executions.

Concluding, we can state that the three research questions have been met. Some limitations exist that we aim to tackle in future work: regarding RQ1 and RQ3, for example, the number of false positives and regarding RQ2, for example, the provision of pre-processing methods dealing with, e.g., logs only containing start or complete events. Finally, future work will address the discovery of ISC candidates that trace back to exception handling.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Table A.5
Precision and recall for the manufacturing example for Algorithm 3 with different parameter combinations.

	$\gamma_3 = 1$	$\gamma_3 = 0.99$	$\gamma_3 = 0.95$	$\gamma_1 = 0.8$	$\gamma_3 = 0.6$	$\gamma_3 = 0.4$	$\gamma_3 = 0.2$
$\kappa = 0$	$p = \frac{3}{4}$ $r = \frac{3}{7}$	$p = \frac{5}{12}$ $r = \frac{5}{7}$	$p = \frac{5}{12}$ $r = \frac{5}{7}$	$p = \frac{7}{15}$ $r = 1$	$p = \frac{7}{15}$ $r = 1$	$p = \frac{7}{15}$ $r = 1$	$p = \frac{7}{16}$ $r = 1$
$\kappa = 0.05$	$p = \frac{3}{4}$ $r = \frac{3}{7}$	$p = \frac{5}{12}$ $r = \frac{5}{7}$	$p = \frac{5}{12}$ $r = \frac{5}{7}$	$p = \frac{7}{15}$ $r = 1$	$p = \frac{7}{15}$ $r = 1$	$p = \frac{7}{15}$ $r = 1$	$p = \frac{7}{16}$ $r = 1$
$\kappa = 0.1$	$p = \frac{3}{4}$ $r = \frac{3}{7}$	$p = \frac{5}{12}$ $r = \frac{5}{7}$	$p = \frac{5}{12}$ $r = \frac{5}{7}$	$p = \frac{7}{15}$ $r = 1$	$p = \frac{7}{15}$ $r = 1$	$p = \frac{7}{15}$ $r = 1$	$p = \frac{7}{16}$ $r = 1$
$\kappa = 0.2$	$p = \frac{3}{4}$ $r = \frac{3}{7}$	$p = \frac{5}{12}$ $r = \frac{5}{7}$	$p = \frac{5}{12}$ $r = \frac{5}{7}$	$p = \frac{7}{15}$ $r = 1$	$p = \frac{7}{15}$ $r = 1$	$p = \frac{7}{15}$ $r = 1$	$p = \frac{7}{16}$ $r = 1$
$\kappa = 0.3$	$p = \frac{3}{4}$ $r = \frac{3}{7}$	$p = \frac{5}{12}$ $r = \frac{5}{7}$	$p = \frac{5}{12}$ $r = \frac{5}{7}$	$p = \frac{7}{15}$ $r = 1$	$p = \frac{7}{15}$ $r = 1$	$p = \frac{7}{15}$ $r = 1$	$p = \frac{7}{16}$ $r = 1$
$\kappa = 0.4$	$p = \frac{3}{4}$ $r = \frac{3}{7}$	$p = \frac{5}{12}$ $r = \frac{5}{7}$	$p = \frac{5}{12}$ $r = \frac{5}{7}$	$p = \frac{7}{15}$ $r = 1$	$p = \frac{7}{15}$ $r = 1$	$p = \frac{7}{15}$ $r = 1$	$p = \frac{7}{16}$ $r = 1$

Table A.6
Precision and recall for the higher education example for Algorithm 1 with different parameter combinations.

	$\gamma_1 = 1$	$\gamma_1 = 0.95$	$\gamma_1 = 0.8$	$\gamma_1 = 0.6$	$\gamma_1 = 0.4$	$\gamma_1 = 0.2$
$\varepsilon_1 = 0$ s	p = 1 r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1
$\varepsilon_1 = 0.1$ s	p = 1 r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1
$\varepsilon_1 = 1$ s	p = 1 r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1
$\varepsilon_1 = 10$ s	p = 1 r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1
$\varepsilon_1 = 100$ s	p = 1 r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{12}{13}$ r = 1	p = $\frac{3}{4}$ r = 1

Appendix. Precision and recall tables

See Tables A.2–A.6.

References

[1] L.T. Ly, F.M. Maggi, M. Montali, S. Rinderle-Ma, W.M.P. van der Aalst, Compliance monitoring in business processes: Functionalities, application, and tool-support, Inf. Syst. 54 (2015) 209–234.

[2] K. Winter, S. Rinderle-Ma, W. Grossmann, I. Feinerer, Z. Ma, Characterizing regulatory documents and guidelines based on text mining, in: On the Move to Meaningful Internet Systems, 2017, pp. 3–20.

[3] R. Lenz, M. Reichert, IT support for healthcare processes - premises, challenges, perspectives, Data Knowl. Eng. 61 (1) (2007) 39–58.

[4] R. Blaser, M. Schnabel, C. Biber, M. Bäumlein, O. Heger, M. Beyer, E. Opitz, R. Lenz, K.A. Kuhn, Improving pathway compliance and clinician performance by using information technology, Int. J. Med. Inform. 76 (2–3) (2007) 151–156.

[5] R.M. Dijkman, B. Gfeller, J.M. Küster, H. Völzer, Identifying refactoring opportunities in process model repositories, Inf. Softw. Technol. 53 (9) (2011) 937–948.

[6] S. Rinderle-Ma, S. Kabicher-Fuchs, An indexing technique for compliance checking and maintenance in large process and rule repositories, Enterp. Model. Inf. Syst. Archit. 11 (2016) 2:1–2:24.

[7] T. Voglhofer, S. Rinderle-Ma, Collection and elicitation of business process compliance patterns with focus on data aspects, Bus. Inf. Syst. Eng. (2019).

[8] S.W. Sadiq, G. Governatori, K. Namiri, Modeling control objectives for business process compliance, in: Business Process Management, 2007, pp. 149–164.

[9] W. Fdhila, M. Gall, S. Rinderle-Ma, J. Mangler, C. Indiono, Classification and formalization of instance-spanning constraints in process-driven applications, in: Business Process Management, 2016, pp. 348–364.

[10] K. Winter, S. Rinderle-Ma, Discovering instance-spanning constraints from process execution logs based on classification techniques, in: Enterprise Distributed Object Computing, 2017, pp. 79–88.

[11] 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: Business Process Management, 2017, pp. 306–323.

[12] H. van der Aa, H. Leopold, H.A. Reijers, Checking process compliance against natural language specifications using behavioral spaces, Inf. Syst. 78 (2018) 83–95.

[13] K. Winter, S. Rinderle-Ma, Detecting constraints and their relations from regulatory documents using NLP techniques, in: On the Move to Meaningful Internet Systems, 2018, pp. 261–278.

[14] K. Winter, S. Rinderle-Ma, Untangling the GDPR Using ConRelMiner, Tech. rep., 2018, URL http://arxiv.org/abs/1811.03399.

[15] A. Rozinat, W.M.P. van der Aalst, Decision mining in prom, in: Business Process Management, 2006, pp. 420–425.

[16] S. Rinderle-Ma, M. Gall, W. Fdhila, J. Mangler, C. Indiono, Collecting Examples for Instance-Spanning Constraints, Tech. rep., 2016.

[17] R.P.J.C. Bose, W.M.P. van der Aalst, I. Zliobaite, M. Pechenizkiy, Handling concept drift in process mining, in: Advanced Information Systems Engineering, 2011, pp. 391–405.

[18] R.J. Wieringa, Design Science Methodology for Information Systems and Software Engineering, Springer, 2014.

[19] L.T. Ly, S. Rinderle-Ma, K. Göser, P. Dadam, On enabling integrated process compliance with semantic constraints in process management systems - requirements, challenges, solutions, Inf. Syst. Front. 14 (2) (2012) 195–219.

[20] W. Grossmann, S. Rinderle-Ma, Fundamentals of Business Intelligence, Data-Centric Systems and Applications, Springer, 2015.

[21] J. Han, J. Pei, M. Kamber, Data Mining: Concepts and Techniques, Elsevier, 2011.

[22] W.M.P. van der Aalst, Process Mining - Data Science in Action, Second Edition, Springer, 2016.

[23] A. Weijters, W.M. van Der Aalst, A. De Medeiros, Process mining with the heuristics miner-algorithm, Tech. Rep. WP 166, Technische Universiteit Eindhoven, 2006, pp. 1–34.

[24] C.W. Günther, W.M.P. van der Aalst, Fuzzy mining - adaptive process simplification based on multi-perspective metrics, in: Business Process Management, 2007, pp. 328–343.

[25] S.J. van Zelst, B.F. van Dongen, W.M.P. van der Aalst, Event stream-based process discovery using abstract representations, Knowl. Inf. Syst. 54 (2) (2018) 407–435.

[26] F. Stertz, S. Rinderle-Ma, Process histories - detecting and representing concept drifts based on event streams, in: On the Move to Meaningful Internet Systems. OTM 2018 Conferences - Confederated International Conferences, 2018, pp. 318–335.

[27] A. Rozinat, W.M.P. van der Aalst, Conformance checking of processes based on monitoring real behavior, Inf. Syst. 33 (1) (2008) 64–95.

[28] A. Burattin, S.J. van Zelst, A. Armas-Cervantes, B.F. van Dongen, J. Carmona, Online conformance checking using behavioural patterns, in: Business Process Management, 2018, pp. 250–267.

[29] W.M.P. van der Aalst et al., Process mining manifesto, in: Business Process Management Workshops, 2011, pp. 169–194.

[30] M. Hashmi, G. Governatori, M.T. Wynn, Normative requirements for regulatory compliance: An abstract formal framework, Inf. Syst. Front. 18 (3) (2016) 429–455.

[31] N. Lohmann, Compliance by design for artifact-centric business processes, Inf. Syst. 38 (4) (2013) 606–618.

[32] M. Leitner, J. Mangler, S. Rinderle-Ma, Definition and enactment of instance-spanning process constraints, in: Web Information Syst. Eng., 2012, pp. 652–658.

[33] C. Indiono, J. Mangler, W. Fdhila, S. Rinderle-Ma, Rule-based runtime monitoring of instance-spanning constraints in process-aware information systems, in: On the Move to Meaningful Internet Syst., 2016, pp. 381–399.

[34] L. Pufahl, A. Meyer, M. Weske, Batch regions: Process instance synchronization based on data, in: Enterprise Distrib. Object Comp., 2014, pp. 150–159.

[35] J. Mangler, S. Rinderle-Ma, Rule-based synchronization of process activities, in: Commerce and Enterprise Computing, 2011, pp. 121–128.

[36] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, Queue mining for delay prediction in multi-class service processes, Inf. Syst. 53 (2015) 278–295.

[37] J. Pflug, S. Rinderle-Ma, Application of dynamic instance queuing to activity sequences in cooperative business process scenarios, Int. J. Coop. Inf. Syst. 25 (1) (2016) 1–28.

[38] R. Agrawal, T. Imielinski, A.N. Swami, Mining association rules between sets of items in large databases, in: SIGMOD, 1993, pp. 207–216.

[39] R. Agrawal, R. Srikant, Mining sequential patterns, in: Data Engineering, 1995, pp. 3–14.

[40] H. Mannila, H. Toivonen, A.I. Verkamo, Discovery of frequent episodes in event sequences, Data Min. Knowl. Discov. 1 (3) (1997) 259–289.

[41] K. Böhmer, S. Rinderle-Ma, Association rules for anomaly detection and root cause analysis in process executions, in: Advanced Information Systems Engineering, 2018, pp. 3–18.

[42] F. Mannhardt, M. de Leoni, H.A. Reijers, W.M.P. van der Aalst, Decision mining revisited - discovering overlapping rules, in: Advanced Information Systems Engineering, 2016, pp. 377–392.

[43] M. Montali, F.M. Maggi, F. Chesani, P. Mello, W.M.P. van der Aalst, Monitoring business constraints with the event calculus, ACM TIST 5 (1) (2013) 17:1–17:30.

[44] F.M. Maggi, M. Montali, M. Westergaard, W.M.P. van der Aalst, Monitoring business constraints with linear temporal logic: An approach based on colored automata, in: Business Process Management, 2011, pp. 132–147.

[45] A. Burattin, M. Cimitile, F.M. Maggi, A. Sperduti, Online discovery of declarative process models from event streams, IEEE Trans. Serv. Comput. 8 (6) (2015) 833–846.

[46] N. Martin, A. Solti, J. Mendling, B. Depaire, A. Caris, Mining batch activation rules from event logs, IEEE Trans. Serv. Comput. (2019).

[47] N. Russell, W.M.P. van der Aalst, A.H.M. ter Hofstede, Workflow exception patterns, in: Advanced Information Systems Engineering, 2006, pp. 288–302.

[48] J. Claes, G. Poels, Merging event logs for process mining: A rule based merging method and rule suggestion algorithm, Expert Syst. Appl. 41 (16) (2014) 7291–7306.

[49] J.F. Allen, Maintaining knowledge about temporal intervals, Commun. ACM 26 (11) (1983) 832–843.

[50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[51] L.E. Raileanu, K. Stoffel, Theoretical comparison between the gini index and information gain criteria, Ann. Math. Artif. Intell. 41 (1) (2004) 77–93.

[52] F. Stertz, S. Rinderle-Ma, T. Hildebrandt, J. Mangler, Testing processes with service invocation: Advanced logging in cpee, in: Service-Oriented Computing, 2017, pp. 189–193.

[53] J. Mangler, S. Rinderle-Ma, Cpee-cloud process execution engine, in: BPM Demos, in: ceur, vol. 1295, 2014.

[54] F. Pauker, J. Mangler, S. Rinderle-Ma, C. Pollak, Centurio.work - modular secure manufacturing orchestration, in: Industrial Track at BPM 2018, 2018, pp. 164–171.

[55] L.T. Ly, C. Indiono, J. Mangler, S. Rinderle-Ma, Data transformation and semantic log purging for process mining, in: Advanced Information Systems Engineering, 2012, pp. 238–253.