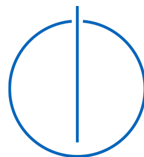
DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Discovering and Visualizing Ordering Instance Spanning Constraints from Event Streams

Julian Bouchard



DEPARTMENT OF INFORMATICS

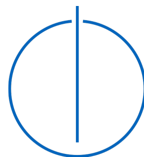
TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Discovering and Visualizing Ordering Instance Spanning Constraints from Event Streams

Entdeckung und Visualisierung von ordnungsbezogenen instanzübergreifenden Beschränkungen aus Event Streams

Author:	Julian Bouchard
Supervisor:	Prof. Dr. Stefanie Rinderle-Ma
Advisor:	Dr. Karolin Winter
Submission Date:	15.09.2022



I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.09.2022 Julian Bouchard

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Research Questions . . . 2
1.3 Related Work . . . 3
1.3.1 Extent of Current Research . . . 3
1.3.2 Existing Tools and Algorithms . . . 3
1.4 Challenges and Contributions . . . 4

2 Online Ordering ISC Approach **5**
2.1 Pre-processing . . . 7
2.2 Necessary Data Structures . . . 9
2.3 Processing . . . 9
2.4 Challenges . . . 12

3 Visualization Method **16**

4 Implementation **23**
4.1 Overview . . . 24
4.2 Planned Features . . . 27

5 Evaluation **28**
5.1 Experiment Setup . . . 28
5.2 Experiment Results and Assessment . . . 30

6 Conclusion **34**

Bibliography **36**

1 Introduction

Process mining is the means by which process models are extracted from event logs [AWM04]. These event logs are often in the form of *.XES* files generated directly from companies. Certain mining algorithms, such as the Alpha Miner [AWM04] or Heuristics Miner [WDD06], are then employed to discover models that reflect the real world data. These are in turn displayed as a Petri net or in the Business Process Model Notation (BPMN) [Kun+10].

1.1 Motivation

From this, the relatively young field of online process mining has sprouted and garnered some scientific attention as of late. The first scientific papers appeared around 2014 [BSA14] and in recent years more and more papers are being dedicated to this topic. The main advantage of online process mining is that the data can be used as it's being produced. This brings with it multiple positive advantages.

For one, it obviously eliminates the need to wait for larger batches of data to produce models. Not only that, but the models that are produced adapt in real time as more data flows in and thus provide live feedback to the end-user. Secondly, in many instances there is no system in place or time to store data. Take Amazon, for example, that processes 1.1 million requests per second [AMZN]. This was back in 2013. Since then companies have only amassed more data. In cases like these there simply is no other choice but to process the data immediately. Alternatively, online process mining can deal with situations where an event log might be too large to store or process in its entirety. It is therefore useful to possess an algorithm capable of dealing with virtually unlimited data that is fed in gradually.

For these reasons, online process mining has a large number of real world applications — from optimizing business processes, to Big Data processing [ERF16], to fields with low error tolerance such as the medical domain —– and is therefore in need of expansion.

Instance Spanning Constraints (ISC), as the name entails, are observed constraints between multiple instances within or across various process types [WSR20]. They are often overlooked yet play an important role with potentially life saving consequences.

Take, for example, a case where a patient is being treated at a hospital by two different departments (D_A and D_B) for two unrelated illnesses. From the hospital's point of view these are two different instances of two unrelated processes. However, if D_A executes task T_A "give out medication M_A" in order to treat the patient and the patient then undergoes a surgery in D_B, with the task T_B "prepare for surgery", for which M_A is not allowed to be in the patient's system, then going through with the surgery can have dire consequences. Hence, an ISC must exist stating that T_B can never occur after T_A. This kind of ordering mistake must be prevented in real time, before fatal harm can ensue, therefore highlighting the need to discover ISC from event streams.

ISC have been grouped into four categories: I) "Simultaneous execution of activities", II) "Constrained activity execution", III) "Order of activity execution" and IV) "Non-concurrent execution of activities" [WSR20]. In the above example, the order of activity constraint is prominent and this work will limit itself to this particular constraint type.

This thesis aims to bridge the gap between offline instance spanning constraint discovery and the ever widening online setting, in addition to overhauling the existing visualization methods of online process models.

1.2 Research Questions

Over the course of my work I will address the following research questions:

RQ_1: How to design an online algorithm for discovering ordering ISC?
RQ_2: How to visualize the results of an online algorithm for discovering ordering ISC?

The existing algorithm to derive ordering based ISC [WSR20] will serve as a base to answer RQ_1. However, this method must be adequately adapted to use multiple event streams as its input. This in and of itself is a challenge. The streams must first be merged for two reasons. Reason number one: Each stream represents an individual process type. Therefore, in order to even apply the ordering ISC discovery, the pre-processing phase as described in Section 4.1 of [WSR20] must take place. Analogous to merging event logs, I will be merging streams. Reason number two: All current online process model discovery algorithms assume a single stream input. Merging is therefore advantageous, seeing as I implement my process discovery algorithm using existing methods developed by [ZDA17].

For RQ_2 I will propose an entirely new method which will incorporate statistical information from the streams to deliver the user a more intuitive overview. Here I will need to consider how and when to dynamically update the visual models.

1.3 Related Work

1.3.1 Extent of Current Research

The first papers dealing with an event stream instead of an event log were, as mentioned, already conceived in 2014 [BSA14]. This early research focused almost entirely on the basic principles of process discovery from streams. This initial approach relied on the Heuristics Miner as a base and was further developed and abstracted as time went on [ZDA17].

With this groundwork laid, within the past 3 years emerging papers have branched out that explore various extensions. ProM [ProM] implementations were created with more algorithms than just the Heuristics Miner. The issue of unordered streams was explored [AWS20]. Others researched to what extent activities could be parallelized to increase efficiency [Tav+19]. Some focused on filtering anomalies from the event streams [KF21]. Lastly, many papers have been dedicated to the topic of concept drift [SR18], [CG12], [Mag+13].

However, certain areas are still in need of further attention. For example, how to increase memory efficiency while not sacrificing accuracy is a question that remains open. Furthermore, the subject of Instance Spanning Constraints is deserving of attention, which will make up a large part of my work. ISC discovery on non-online process mining has been detailed already [IFR18], [WR17], [WSR20], however no approach exists in which ISC are identified in an online setting—i.e. from event streams.

1.3.2 Existing Tools and Algorithms

Some public tools and algorithmic implementations already exist in the online process mining space. The most prominent are the ProM [ProM] packages. These are implemented in Java and published through the ProM Package Manager. The biggest upside is the number of algorithms to choose from. Yet, data importation can be tricky, the process requires multiple steps and more importantly there is no option to derive Instance Spanning Constraints.

Multiple other online process mining implementations exist, each with slight variations and different focal points – often incorporating concept drift [Has+15], [ERF16], [ZCH19], [Nav+20]. However, no tool or implementation exists where numerous streams are taken into account and ISC are discovered.

On the flip side, there are tools for ISC discovery, mainly [ISC]. However, this tool is for an offline setting — i.e. it requires event logs and is incapable of dealing with event streams.

1.4 Challenges and Contributions

Discovering ISC and process models from event streams is not without hurdles. Seeing as the incoming data stream could be "unlimited" but our model must fit in storage, a trade-off has to be made between memory usage and accuracy [Cer+20].

Moreover, the real world is disorderly and processes can change over time. Detecting and dealing with this so called concept drift [SR18] can be quite arduous, yet enables models to more accurately reflect the real world changes. That being said, this thesis will not focus on dealing with concept drift.

Another issue arises when data arrives in a different order than their timestamps. This is often referred to as unorderedness and is dealt with here [AWS20]. This can cause many difficulties, especially when examining order related ISC. However, for the purposes of this work an ordered stream is assumed.

Similarly, most current techniques cannot detect and properly deal with anomalous data in the incoming stream. Anomalous data can occur for many reasons: an event was not logged, an event was logged twice, data was lost or corrupted during transmission etc. This can have a massive negative effect on the resulting model as it usually assumes all data to be valid and complete. This too was solved here [KF21] and an anomaly free stream will be assumed from here on out.

My primary contribution to this field of work will be an algorithm for discovering ISC of constraint type "Order of activity execution" directly from an event stream. My secondary contribution is a method of visualizing mined process models as well as the aforementioned ISC in the context of an online environment. These two initially theoretical components will then be practically instantiated within a web-based tool. This tool will allow the end-user to upload *.XES* files which will be used to simulate an event stream. The resulting process model—initially mined with the alpha algorithm—and ISC will then be visually displayed to the user.

2 Online Ordering ISC Approach

Figure 2.1 illustrates the procedure of the online ISC discovery approach from start to end compared to the existing offline approach (Figure 2.2).

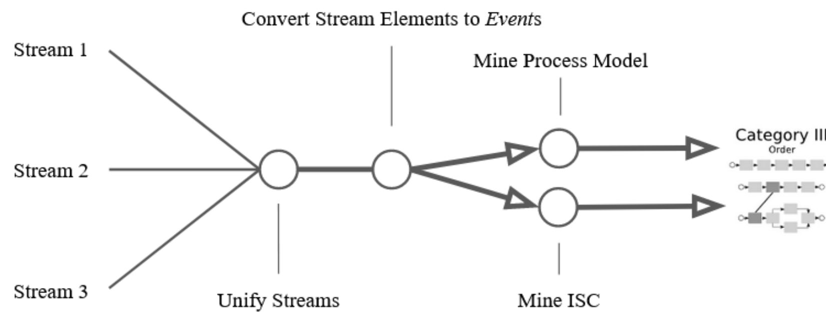


Figure 2.1: Overview of the novel online approach.

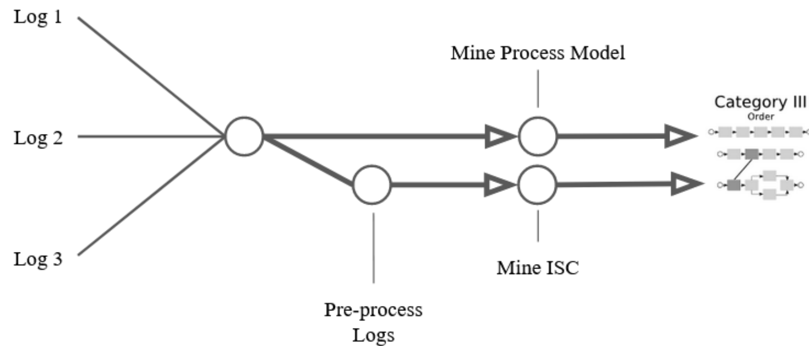


Figure 2.2: Overview of the existing offline approach.

The following small example will serve to illustrate each step from raw input to final output. The example consists of two instances which were individually logged. The goal will be to find ordering constraints between the two instances. Their individual process execution logs and process models can be viewed in Table 2.1 and Figure 2.3 respectively.

Process Execution Log L_1

Activity Label	Event ID	Unique Case ID	Lifecycle	Timestamp
Prepare A	1	1a	start	11-08-2022:11.53
Execute A	2	1a	start	11-08-2022:11.54
Prepare A	3	2b	start	11-08-2022:11.54
Prepare A	4	3c	start	11-08-2022:12.02
Execute A	5	2b	start	11-08-2022:11.58
Conclude A	6	2b	start	11-08-2022:12.03
Conclude A	7	1a	start	11-08-2022:12.09
Execute A	8	3c	start	11-08-2022:12.10
Conclude A	9	3c	start	11-08-2022:12.16

Process Execution Log L_2

Activity Label	Event ID	Unique Case ID	Lifecycle	Timestamp
Prepare B	11	2b	start	11-08-2022:11.55
Prepare B	12	1a	start	11-08-2022:11.56
Examine B	13	2b	start	11-08-2022:11.58
Upload Result of B	14	2b	start	11-08-2022:11.59
Examine B	15	1a	start	11-08-2022:12.03
Prepare B	16	3c	start	11-08-2022:12.04
Upload Result of B	17	1a	start	11-08-2022:12.06
Examine B	18	3c	start	11-08-2022:12.10
Upload Result of B	19	3c	start	11-08-2022:12.15

Table 2.1: Unprocessed execution logs based on two *.XES* files.

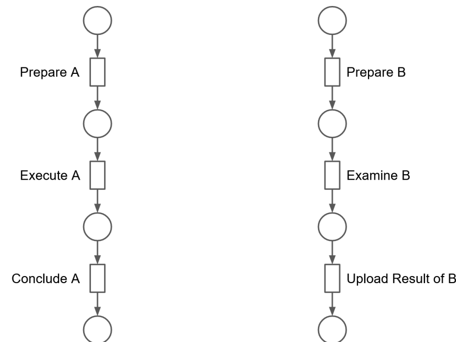


Figure 2.3: Process models for L_1 (left) and L_2 (right) modeled as Petri nets.

2.1 Pre-processing

There are three possible scenarios of how the raw input data might present itself:

(1) Multiple files

Akin to an offline approach, multiple complete event logs may also be used as an input method. These can be employed to simulate a stream, allowing for better replayability and testing. This input type is purely for testing and research purposes. For offline ISC discovery on log files please see [WSR20].

(2) Multiple streams

Here each individual stream corresponds to a different instance. We assume each stream emits their own ordered events independently and immediately — i.e. they do not buffer or delay the emission of events.

(3) Singular stream

This is the easiest and preferred method to work with. One stream emits events from all instances in the order that they occurred.

It is important that, no matter the method, all events contain the following attributes:

- Activity label | *concept:name*
- Timestamp | *time:timestamp*
- Unique Case Identifier

These three attributes are of utmost necessity considering that without any one of them the algorithm ceases to function. The lifecycle attribute (*lifecycle:transition*) is technically optional and all lifecycle checks can then be left out of the algorithm. Lastly, when a single merged stream is present, it is paramount that each event also contains an instance or log identifier to know from which instance said event originated from.

The algorithm assumes a single stream as its direct input. Therefore in both scenarios (1) and (2) extra steps are required to bring them into the correct form. The singular stream is utilized as the base input because it allows for easy compatibility with existing online process mining algorithms which rely on a singular stream.

Multiple files $\rightarrow$ Singular stream:

Given n process execution logs. Per log file, all events within the file are given the additional log identifier attribute. This can either be $i \in \{1, \dots, n\}$, the filename or any other alphanumeric sequence unique to this file. Then all events from all files are accumulated in an ordered data structure, such as a list. Lastly, this data structure must be sorted by timestamp, earliest event first. The resulting data can now be streamed.

Merged Process Excution Logs of L₁ and L₂

Activity Label	Event ID	Unique Case ID	Log ID	Lifecycle	Timestamp
Prepare A	1	1a	1	start	11-08-2022:11.53
Execute A	2	1a	1	start	11-08-2022:11.54
Prepare A	3	2b	1	start	11-08-2022:11.54
Prepare B	11	2b	2	start	11-08-2022:11.55
Prepare B	12	1a	2	start	11-08-2022:11.56
Execute A	5	2b	1	start	11-08-2022:11.58
Examine B	13	2b	2	start	11-08-2022:11.58
Upload Result of B	14	2b	2	start	11-08-2022:11.59
Prepare A	4	3c	1	start	11-08-2022:12.02
Examine B	15	1a	2	start	11-08-2022:12.03
Conclude A	6	2b	1	start	11-08-2022:12.03
Prepare B	16	3c	2	start	11-08-2022:12.04
Upload Result of B	17	1a	2	start	11-08-2022:12.06
Conclude A	7	1a	1	start	11-08-2022:12.09
Examine B	18	3c	2	start	11-08-2022:12.10
Execute A	8	3c	1	start	11-08-2022:12.10
Upload Result of B	19	3c	2	start	11-08-2022:12.15
Conclude A	9	3c	1	start	11-08-2022:12.16

Table 2.2: Execution log as a result of merging L_1 and L_2.

When this approach is applied to our running example the merged log, as can be seen in Table 2.2, is constructed.

Multiple streams → Singular stream:

Given n streams that emit events. Monitor all streams. Once stream i emits an event, the log identifier attribute with value i is added and the event is passed on as the output of the merged stream. Additional precautions might be necessary in the situation where two or more streams emit elements at the same time.

In the final step of preparing the input, all stream emissions — which were thus far still xml elements — are mapped to an *Event* type. An *Event*, within the bounds of this paper, is defined as a 5-Tuple: (*uid*: Unique Case Identifier, *lb*: Activity Label, *ts*: Timestamp, *tc*: Lifecycle Value, *log*: Log Identifier). A singular stream of *Event*s will serve as the final input for our Algorithm. In our running example all entries would also undergo this conversion. For example, the entry with Event ID 3 turns into (2b, Prepare A, 11-08-2022:11.54, start, 1).

In addition to the stream, the algorithm also requires $\gamma_3 \in [0, 1]$ as well as $\kappa \in [0, 0.5)$. These two parameters are taken directly from the offline algorithm [WSR20] and used exclusively in the filter function (Algorithm 3: Function *filter* in [WSR20]).

2.2 Necessary Data Structures

The ISC algorithm operates on three internal data structures: *countEvs*, *ordActivities* and *d*. The former two are taken from the offline version of the algorithm [WSR20] and serve the same purpose here. All three data structures are hash maps.

ordActivities maps Tuples of activity labels (lb_1, lb_2) to an object which contains a "pairs" and "count" attribute, where "pairs" is a list of Tuples (or Lists) and "count" an Integer. It records the ordered pairs and how often they have occurred. For example: {(*Prepare A, Prepare B*) $\rightarrow$ {"pairs" $\rightarrow$ [*(3, 11), (1, 12), (4, 16)*], "count" $\rightarrow$ 3}}. Side note: All Tuple elements within "pairs" should be *Event*s, but for brevity and readability their Event ID was used. This structure is needed to note all potential ISC. Their count is important for calculations in the filter function.

countEvs maps activity labels to Integers. It is responsible for keeping count of how often a certain activity has been seen thus far. This information is necessary later on to determine whether a particular pair in *ordActivities* is kept or filtered out.

d maps unique case identifiers to Sets of *Event*s. It keeps track of previously seen, unpaired e_1 candidates for each case. This data structure is integral to discovering *ordActivities* pairs.

2.3 Processing

The original offline algorithm relies on a different type of pre-processed event log. Here all events are merged into traces based on their *unique case identifier*. Subsequently all traces are sorted by timestamp. At the end, each trace contains all sorted events across all instances with the same *uid*. During the offline algorithm, each event (e_1) in a given trace is compared to every following event (e_2) in the same trace [WSR20]. This works perfectly in an environment where we have knowledge of all succeeding events. However, in an online environment we cannot possibly know what events are yet to come, we can only ever look backwards at the prior emissions. In other words, we always operate from the perspective of e_2 and we look through the past observed events with the same *unique case identifier* to find our matching e_1.

Per each incoming event, the algorithm checks whether this event's particular *uid* has already been witnessed. If this is the case, the algorithm inspects all past events with the same *uid*. These can be found in $d[uid]$. Next, similar checks are performed per possible (e_1, e_2) pair as in the offline algorithm in line 13 [WSR20]. This entails checking whether the two events originated from separate logs. This ensures that this constraint is indeed instance spanning. Additionally, it is necessary to verify that

their timestamps are different. If the above-mentioned checks are passed, the pair is added to ordActivities and e_1 is removed from the $d[uid]$ set. This removal is important as it reflects the functionality of the *break* statement in the offline algorithm in line 19 [WSR20], inhibiting the creation of superfluous (e_1, e_i) pairs. Lastly, e_2 is added to $d[uid]$ allowing it to be discovered as a possible e_1 for the next emitted event with the same case identifier. In effect, e_2 replaces every e_1 it paired with during the iteration.

The data structures are updated indefinitely. Consequently we cannot simply apply the filter function once the main algorithm has completed its run, as this may never happen. Therefore, the filtering function must be employed based on the intended use case. Whenever output is required, be that every n events or upon a button press, the filter function can be run with *ordActivities*, *countEvs*, γ_3 and κ to obtain the final *ordActivities* data.

Running the algorithm on the above example yields the following *ordActivities* once all entries have been processed:

Before filtering:

ordActivities = {
(Prepare A, Prepare B) → {"pairs" → [(3, 11), (1, 12), (4, 16)], "count" → 3},
(Execute A, Prepare B) → {"pairs" → [(2, 12)], "count" → 1},
(Prepare B, Execute A) → {"pairs" → [(11, 5), (16, 8)], "count" → 2},
(Execute A, Upload Result of B) → {"pairs" → [(5, 14), (8, 19)], "count" → 2},
(Examine B, Conclude A) → {"pairs" → [(13, 6), (15, 7), (19, 9)], "count" → 3},
(Upload Result of B, Conclude A) → {"pairs" → [(14, 6), (17, 7), (19, 9)], "count" → 3},
(Prepare B, Conclude A) → {"pairs" → [(12, 7)], "count" → 1}
}

After filtering with $\gamma_3 = 1$ and $\kappa = 0$:

ordActivities = {
(Prepare A, Prepare B) → {"pairs" → [(3, 11), (1, 12), (4, 16)], "count" → 3},
(Examine B, Conclude A) → {"pairs" → [(13, 6), (15, 7), (19, 9)], "count" → 3},
(Upload Result of B, Conclude A) → {"pairs" → [(14, 6), (17, 7), (19, 9)], "count" → 3}
}

The visual exemplification of which can be found in Figure 2.4.

Algorithm 1: Category 3 ISC Discovery from Event Stream

```
   Input: ordered stream, κ, γ3
   Result: ordActivities (list of ISC candidates for Category 3)
1  ordActivities = dict()
2  countEvs = dict()
3  d = dict()
4  while true do
5      e2 ← observe(ordered stream)
6      if e2.lc != "start" then
7          continue
8      end
9      countEvs[e2.lb] += 1
10     if e2.uid in d then
11         for e1 in d[e2.uid] do
12             if e1.log != e2.log and e1.ts != e2.ts then
13                 ordActivities[(e1.lb,e2.lb)]["pairs"].append([e1,e2])
14                 ordActivities[(e1.lb,e2.lb)]["count"] +=1
15                 d[e2.uid].remove(e1)
16             end
17         end
18         d[e2.uid].add(e2)
19     else
20         d[e2.uid] ← {e2}
21     end
22     send filter(ordActivities, countEvs, κ, γ3) tofrontend
23 end
```

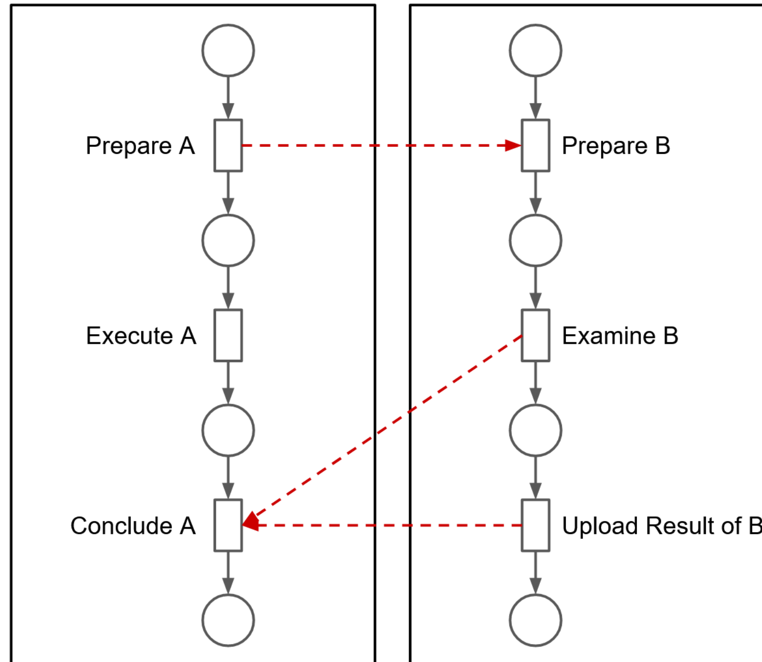


Figure 2.4: ISC discovered by applying the algorithm with $\kappa = 0$ and $\gamma_3 = 1$ to the running example.

2.4 Challenges

The biggest challenge, as mentioned in Section 1.4, is that there are potentially infinitely many incoming events yet we only have finite physical storage. This problem has been explored within the context of online process mining algorithms. Many approaches with varying advantages and drawbacks have been explored [BSA14], [Nav+20], [Bur+15], [Cer+20].

These techniques can be transferred to the ISC algorithm to ensure that the employed data structures remain upward bound. Currently, *countEvs*, *ordActivities* and d could all take on an interminable amount of activity labels or unique case identifiers. Moreover, *ordActivities* and d both contain sub-structures that also grow in $\mathcal{O}(n)$. Therefore, it is crucial to limit the growth of these structures even at the cost of accuracy.

The final implementation (Chapter 4) incorporates Lossy Counting with Budget (LCB). I chose this technique in particular for its decent practical performance according to [MM02]. In order to realize LCB, the algorithm must be altered in the following ways. First, a new input representing the available budget is required. This budget $\mathcal{B}$ is shared by all three data structures. Second, all data structure entries are given

an additional attribute "delta" and the entries in *d* are expanded to now include their "count". Finally, a check and clean is performed every time an element is about to be inserted into any data structure.

Beyond the introduction of Lossy Counting with Budget, *ordActivities* can also be simplified by removing the "pairs" attribute of an entry entirely. The List of Tuples of activity pairs are not accessed in any further parts of this particular implementation nor is it needed during the filtering operation. Accordingly, the reduced *ordActivities* now maps a Tuple of activity labels to an Integer representing the count of said Tuple. However, given alternative scenarios it might be useful or even necessary to keep the "pairs" attribute.

The pseudo code for the memory bound algorithm using Lossy Counting with Budget can be found below (Algorithm 2). The largest change from Algorithm 1 is the introduction of lcb_clean() (Algorithm 3) before every possible new insertion into any data structure. This sub-function first performs the LCB check — whether the sum of all data structures is equal to $\mathcal{B}$. In this case the budget was reached, the current bucket is incremented and at very least one entry stemming from any data structure is removed based on its sum of frequency ("count") and "delta" value. Once complete, there is now space within the allotted budget for the new entry to be inserted.

Please keep in mind that other techniques could possibly fit certain use cases better and should be considered when implementing this algorithm.

Chapter 2 Summary and Conclusion.
In this chapter, the approach was outlined to address the research question RQ_1: *How to design an online algorithm for discovering ordering ISC.* All pre-processing steps were covered which detailed how the raw input in the form of process execution logs or streams is converted into an input that is utilized by the core algorithm. Next, the algorithm's data structures were defined and the algorithm's procedure demonstrated using the running example. Finally, the algorithm was expanded upon in order to address the original algorithm's shortcomings.

Algorithm 2: Algorithm 1 with Lossy Counting with Budget

Input: ordered stream, κ, γ_3, $\mathcal{B}$

Result: ordActivities (list of ISC candidates for Category 3)

```
1  ordActivities = dict()
2  countEvs = dict()
3  d = dict()
4  b_curr = 0
5  while true do
6      e2 ← observe(ordered stream)
7      if e2.lc != "start" then
8          continue
9      end
10     if e2.lb in countEvs then
11         countEvs[e2.lb]["count"] += 1
12     else
13         lcb_clean()
14         countEvs[e2.lb] = {"count": 1, "delta": b_curr}
15     end
16     if e2.uid in d then
17         for e1 in d[e2.uid] do
18             if e1.log != e2.log and e1.ts != e2.ts then
19                 if (e1.lb,e2.lb) in ordActivities then
20                     ordActivities[(e1.lb,e2.lb)]["count"] +=1
21                 else
22                     lcb_clean()
23                     ordActivities[(e1.lb,e2.lb)] = {"count": 1, "delta": b_curr}
24                 end
25                 d[e2.uid].remove(e1)
26             end
27         end
28         d[e2.uid].add(e2)
29     else
30         lcb_clean()
31         d[e2.uid] = {"set": {e2}, "count": 1, "delta": b_curr}
32     end
33     send filter(ordActivities, countEvs, κ, γ3) tofrontend
34 end
```

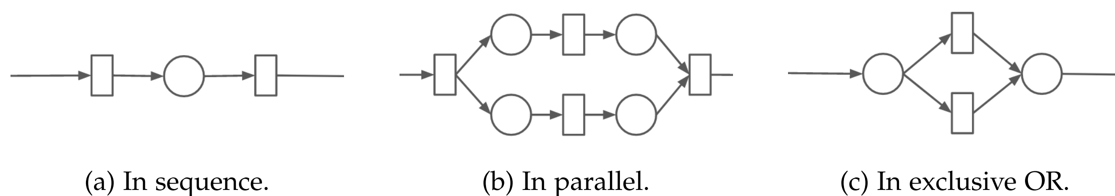
Algorithm 3: Sub-function for the LCB Check and Clean Phase

```
1  Function lcb_clean():
2      if |countEvs| + |ordActivities| + |d| = B then
3          b_curr += 1
4          while |countEvs| + |ordActivities| + |d| = B do
5              for entry in countEvs, ordActivities and d do
6                  if entry["count"] + entry["delta"] ≤ b_curr then
7                      remove entry
8                  end
9              end
10             if no entry was removed then
11                 b_curr = smallest ("count" + "delta") across all data structures
12             end
13         end
14     end
15 return
```

3 Visualization Method

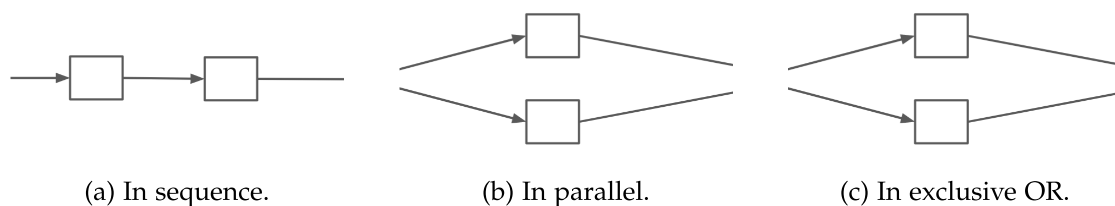
Owing to the fact that we use either an Alpha or Heuristics Miner to derive the process model, their output — a Petri net [RR98] — will serve as the base for our visualization method. In the case of the Heuristics Miner, it is also possible to apply the following visualization technique to a dependency graph [WDD06]. The downside of only using a dependency graph is that there is no visual indication of possible AND or XOR transitions, meaning certain relational information is not present.

A Petri net is a graphical construct consisting of three main components: places, transitions and links. From a process model perspective, transitions represent the activities of an event stream; they are visually represented by rectangles. The places, depicted as circles, and links, as arrows, then allow us to model certain relationships between two or more activities. For example, whether they occur in sequence (Fig. 3.1a), in parallel (Fig. 3.1b) or in a mutually exclusive manner (Fig. 3.1c). The relations, referred to earlier, are not recognizable from a dependency graph alone, as illustrated by the lack of differentiation in Figure 3.2b and 3.2c.



(a) In sequence. (b) In parallel. (c) In exclusive OR.

Figure 3.1: Types of relationships within a Petri net.



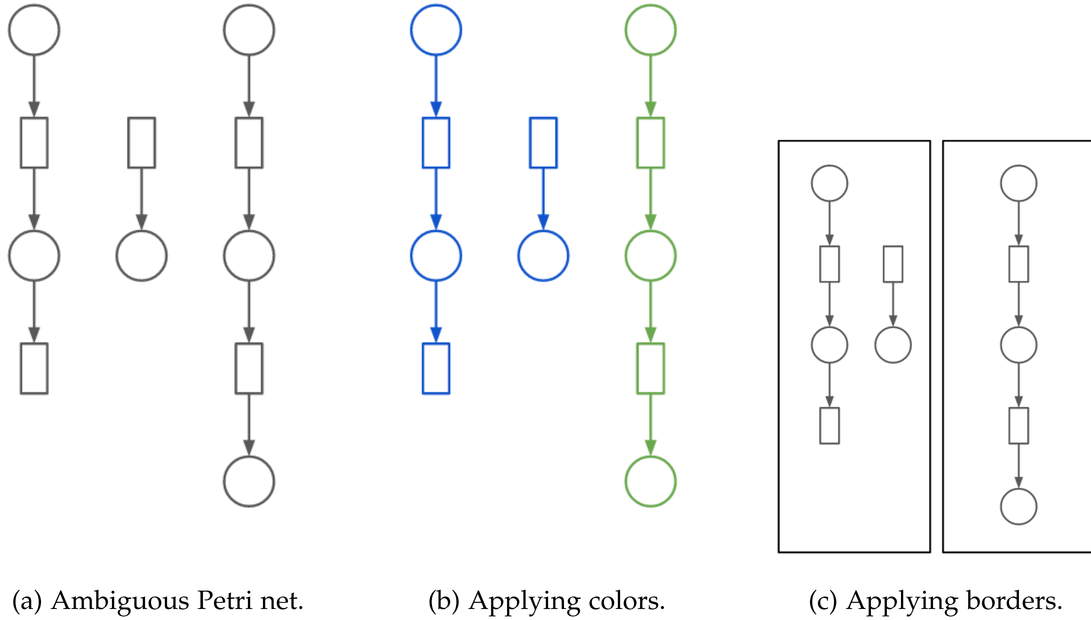
(a) In sequence. (b) In parallel. (c) In exclusive OR.

Figure 3.2: Relationships from Fig. 3.1 as dependency graphs.

The visualization method can be separated into three categories: general, intra-process links and inter-process links (a.k.a ISC links). The general categories contain all visual contributions that apply to every component in entire graph equally, be that a place, transition or link. The second category covers visualization specifics that only affect links within a process model; in other words only those links which connect components that originated from the same log. In contrast, the last category encompasses the visual standard that only pertains to ISC links which bridge separate processes.

1. General.
Because we are working on multiple instances it is crucial to properly and clearly differentiate between the process models created by the different logs / streams. This is important, considering scenarios where a process model belonging to one log contains more than one component [Wik22]. This is especially likely to occur when utilizing an Alpha Miner or a Heuristics Miner without the *all-activities-connected heuristic*. This heuristic is an additional option for the Heuristics Miner which insures that every activity is connected to at least one other activity. Thus no free standing activities can come to be [WDD06]. Otherwise, choosing too high heuristic thresholds and not having the *all-activities-connected heuristic* enabled may result in disconnects in the graph. Such a resulting graph can be observed on the left side of Figure 3.3. Here it is challenging to immediately differentiate between the two process models. To combat this ambiguity we introduce either colors or borders or both. In the former case, all places, transitions and links belonging to $Stream_i$ are assigned a unique color c_i. These components are subsequently outlined or filled with c_i, as shown in Figure 3.3b. Alternatively, one can add borders around each process model to make it visually clear that all graphical components within those borders resulted from the same log id Figure 3.3c.

Due to the nature of the online setting, process models resulting from streams are far from static. This constant change can sometimes make it very hard to keep track of which components were recently added or subtracted. Therefore, in order to make the tracking of changes more intuitive, all new components are given *fade-in* animations and all components that were removed from one update to the next receive *fade-out* animations. Here *fade-in* and *fade-out* refer to the gradual increase or decrease of the component's opacity.



(a) Ambiguous Petri net. (b) Applying colors. (c) Applying borders.

Figure 3.3: Eliminating ambiguity with colors or borders.

2. Intra-Process Links.
In a Petri net example (Fig. 3.4) created by the log L_e, any two activities a_i and a_j ($i \in \mathbb{N}, j = i + 1$) are connected in the following way: l_{a_i,p_i} links a_i to p_i and then l_{p_i,a_j} links p_i to a_j. With regards to these links we use two dimensions to better symbolize the relation between a_i and a_j: i) the certainty — i.e. how certain we are that a_i and a_j are in a causal relation — and ii) the time — i.e. we need to visually reflect whether we have observed the link just recently or already some time ago.

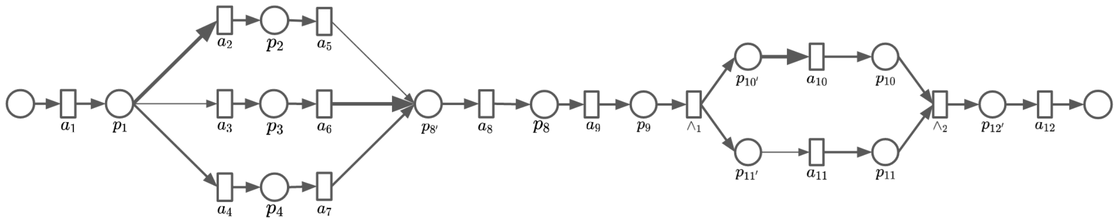


Figure 3.4: Example of a Petri net with all three relation types.

i) The first dimension is the width of each link. The width of the links that connect a_i to a_j represents how sure we are that a_j does indeed follow a_i. In calculating these widths we must differentiate between links similar to l_{a_1,p_1} or l_{p_1,a_2} (Fig. 3.5a) and the special case of AND-splits or joins (Fig. 3.6). Generally speaking, all widths are computed by multiplying the maximum allowed width w_{max} with a scalar h. In the most basic case, as depicted in Figure 3.1a, this h is simply the heuristics dependency value $a_i \Rightarrow_{L_e} a_j$ which depicts "how certain we are that there is truly a dependency relation between two events" [WDD06].

However, for l_{a_1,p_1} we run into the issue that a_1 can possess numerous subsequent activities $a_2, \ldots, a_n$ which are all connected from p_1 with links of the form l_{p_1,a_k} with $k \in [2, \ldots, n]$. If we were to set the h of l_{a_1,p_1} to any single $a_1 \Rightarrow_{L_e} a_k$ value or a static value, like 1, we lose insight into the average relation between a_1 and all a_k. Thus, we cannot calculate a singular heuristic value $a_1 \Rightarrow_{L_e} a_k$ to represent the certainty of all postceding events. Instead, we assign the average of all these dependency heuristics as h to calculate the final width of l_{a_1,p_1}. As a result, the width of l_{a_1,p_1} signifies the average certainty that a subsequent activity of p_1 depends on a_1 (Fig. 3.5a). This allows a user to quickly gain an intuition about the certainty of an activity and all of its subsequent activities even when the graphical representation of these ensuing activities becomes more convoluted.

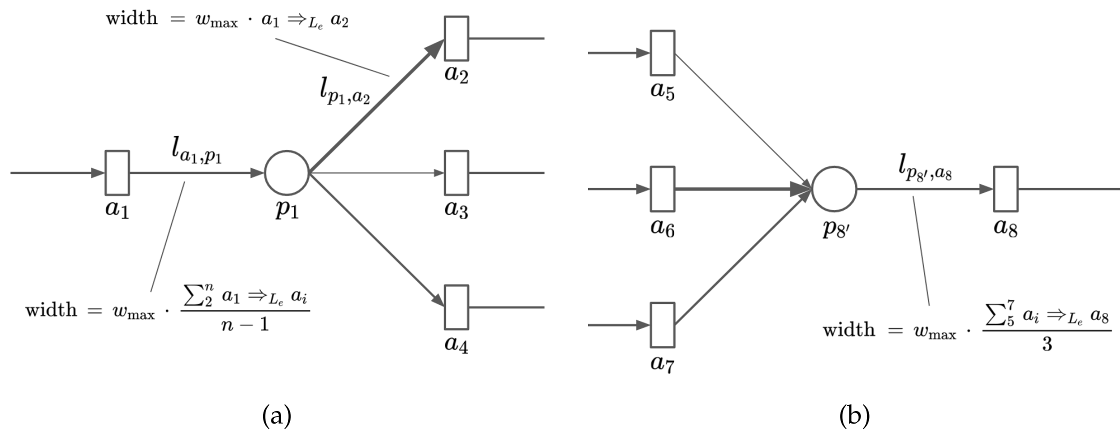


Figure 3.5: Example showcasing how widths are assigned.

Naturally, the inverse problem, as portrayed in Figure 3.5b with numerous antecedent activities, also exists. This obstacle is overcome in an equivalent manner to l_{a_1,p_1} by averaging the heuristic dependency values of all directly preceding activities that connect to the same place. In the specific case of l_{p_1,a_2} there is only one antedate activity. Consequently, h purely equals the heuristic dependency value $a_1 \Rightarrow_{L_e} a_2$.

The introduction of AND-splits and AND-joins is accompanied by the addition of new links. The h of these links is directly equal to the AND-dependency value. For details on the AND-dependency value see [WDD06]. In the example demonstrated in Figure 3.6 this would mean that the width of all three links $l_{p_9,\wedge_1}$, $l_{\wedge_1,p_{10'}}$ and $l_{\wedge_1,p_{11'}}$ is given by $w_{max} * (a_9 \Rightarrow_{L_e} a_{10} \wedge a_{11})$. In the grand scheme of the graph, this measure visualizes how certain we are that a_{10} and a_{11} are in an AND-relation, allowing the viewer to quickly gain a good understanding of the relationship between a_{10} and a_{11}.

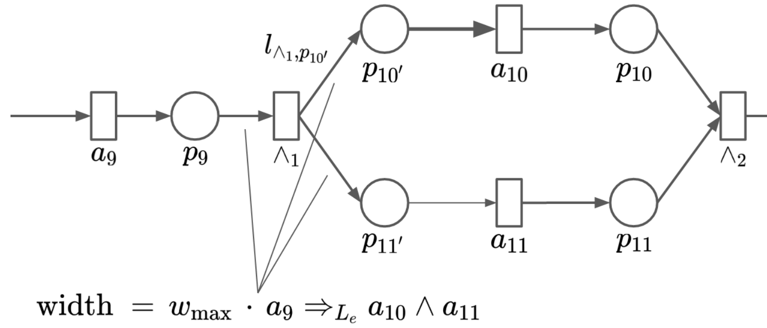


Figure 3.6: Calculating widths of links in an AND-Split.

ii) The second dimension is the opacity of each link. The transparency of any link visualizes how recently the directly follows pair which created this link was observed.

This is achieved by introducing a rate of decay r_d and rate of growth r_g. Additionally, every directly follows pair is assigned a new attribute intensity with a value between 0.1 and 1. This attribute directly corresponds to the opacity of the link; 1 equates to the link being fully opaque and 0.1 means the link is almost completely transparent. We choose 0.1 as our minimal threshold so that even at lowest opacity the links can still be viewed in the graph. This minimum opacity can also be decreased or raised based on the use-case. Alternatively, one could outline all links with a solid border. This would allow for the minimum opacity of the links to sink to 0 without losing the visual indication of their existence.

We determine each pair's intensity during runtime. Anytime a directly follows pair is formed during the online process mining algorithm, all pairs' intensity is reduced by r_d and the observed pair's intensity is increased by r_g. Here the biggest challenge is to choose an optimum r_d and r_g —– given they depend on the total number of activities and the emission rate of the stream. A too large r_d or a too small r_g will result in most links being practically transparent while a too small r_d and a too large r_g will cause

most links to be opaque. Either extreme will lead to information loss as the average difference in transparency becomes imperceptible. This would defeat the purpose of this visual indicator; an observer could no longer gain an intuitive understanding of which pairs are more recent versus which have not been detected for a while.

Similarly to the first dimension, a_1 can be involved in multiple directly follows pairs. For our example in Figure 3.5a we have the following pairs: (a_1, a_2), (a_1, a_3), (a_1, a_4). Therefore, analogous to the width, the intensity of l_{a_1,p_1} is given as the average of all three pairs' intensities.

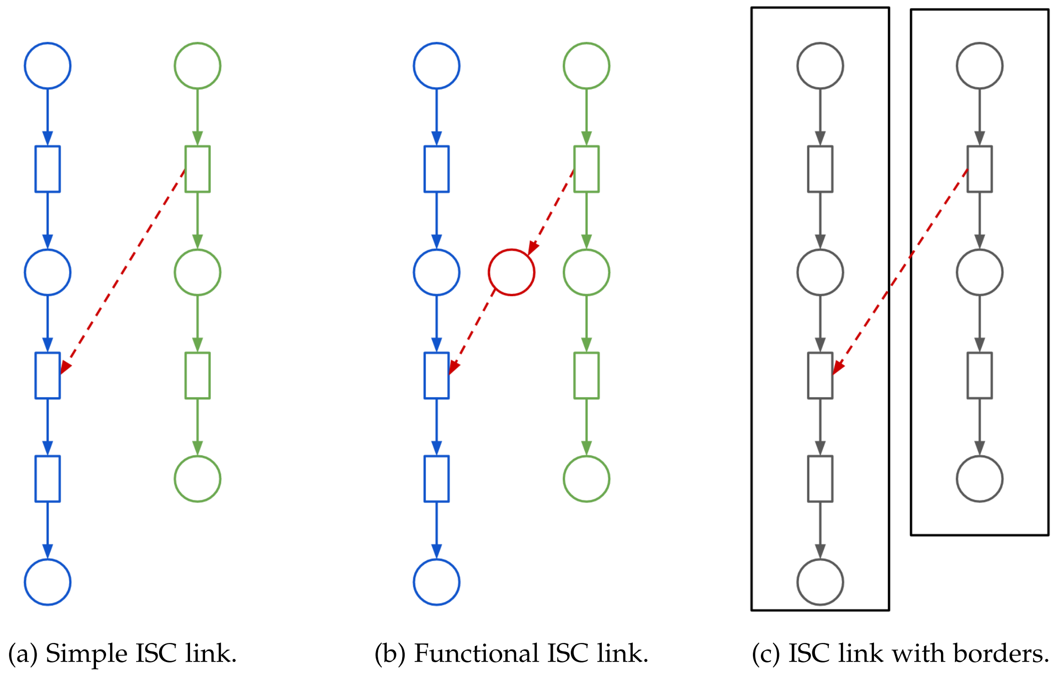
Lastly, the opacity of all links directly leading into or out of AND-splits and AND-joins is always 1, as they are byproducts of these additional transitions.

3. Inter-Process Links.
The links that represent the Instance Spanning Constraints stand out from the normal process model links in two ways. Visually, ISC links, unlike normal links, are presented with a dashed line instead of a solid line. Moreover, they can be given a red color to help them stand out further from all other graphical components. Structurally, these links only ever connect two transitions denoting that the source activity must be executed before the destination activity — like in Figure 3.7a. Linking two transitions, technically speaking, threatens the integrity of the Petri net as this kind of link is not permitted per its syntactic definition. Hence, if a structurally sound Petri net is desired then a supplementary place needs to be introduced as exemplified in Figure 3.7b. Nevertheless, the transition-to-transition links can be kept as a shorthand if visual clarity is prioritized to the functionality of the Petri net.

Contrary to the intra-process links, the ISC links do not currently possess a display of certainty and therefore do not vary in size. However, in future work, the value being compared to γ_3 (see line 4 of Algorithm 3: Function *filter* in [WSR20]) could be employed to represent the certainty.

Further, new ISC links are also given a *fade-in* animation and no longer existing ISC links are faded out. This, just like the fading in and out of all other components, helps the viewer to better follow and keep track of the ever changing model.

Chapter 3 Summary and Conclusion.
This chapter outlined the approach to address the research question RQ_2: *How to visualize the results of an online algorithm for discovering ordering ISC?* Techniques to visually display important information in an intuitive manner were outlined for all graphical components of a resulting Petri net. The main methods discussed pertain to the width and opacity of links as well as their color scheme. Through this we can better express the relationship between activities. Furthermore, edge cases were covered and illustrated with various figures. These visualizing techniques will help the user to easily follow the process models evolution.



(a) Simple ISC link.

(b) Functional ISC link.

(c) ISC link with borders.

Figure 3.7: Three different ISC link implementations.

4 Implementation

All of the previously mentioned algorithms and methods were implemented within a free and open source web-tool called Tool for Online Instance-Spanning-Constraints Discovery (TOID). This proof-of-concept implementation can currently be found at:

https://lehre.bpm.in.tum.de/ports/8030

It is still in its early stages and is intended for scientific use only.

I wrote TOID using *Python* in the backend. This python server is hosted using *Flask* and provides a RESTful API for the frontend client. I wrote the frontend in *TypeScript* using *React* and *MaterialUI*. The graphical output itself is greatly aided by *Cytoscape.js*. Once the start button has been pressed, a post request containing the files and settings is sent to the backend. In turn, the files are parsed, processed and their events are fed into the main algorithm. Finally, the resulting stream of JSON encoded outputs is continuously provided to the frontend over an *EventSource* connection.

While building TOID certain design and implementation decisions had to be made. First, the application accepts *.XES files* — which are internally converted to a singular event stream — instead of multiple event streams as its input. This was done in order to facilitate testing and increase controllability. Because the stream is created internally, we can easily manipulate the emission rate, even during runtime. This allows us to artificially slow down or speed up the stream depending on our current observational needs. Moreover, *.XES* is the standard for process execution logs. Therefore, the myriad of files which already exist can be reused without additional modification in an online setting. The downside is, of course, that real streams cannot be used with the current implementation. At least not yet.

Second, colors were chosen to differentiate the process models instead of borders. Partially due to library restrictions, but mainly for the reason that the colors serve as log identifiers in other parts of TOID. For example, the colors used in the timeline (see feature 14. Timeline) correspond to the colors in the main graph.

Last, simple ISC links were implemented rather than functional ones (Fig. 3.7) for the sake of reducing unnecessary clutter in the graph. Furthermore, TOID is not designed to run or verify Petri nets, simply to provide information about Instance Spanning Constraints in a manner that is visually easy to digest.

4.1 Overview

Below is a screenshot of the complete product in action along with a detailed legend which is categorized into three parts: Input, Options and Output.

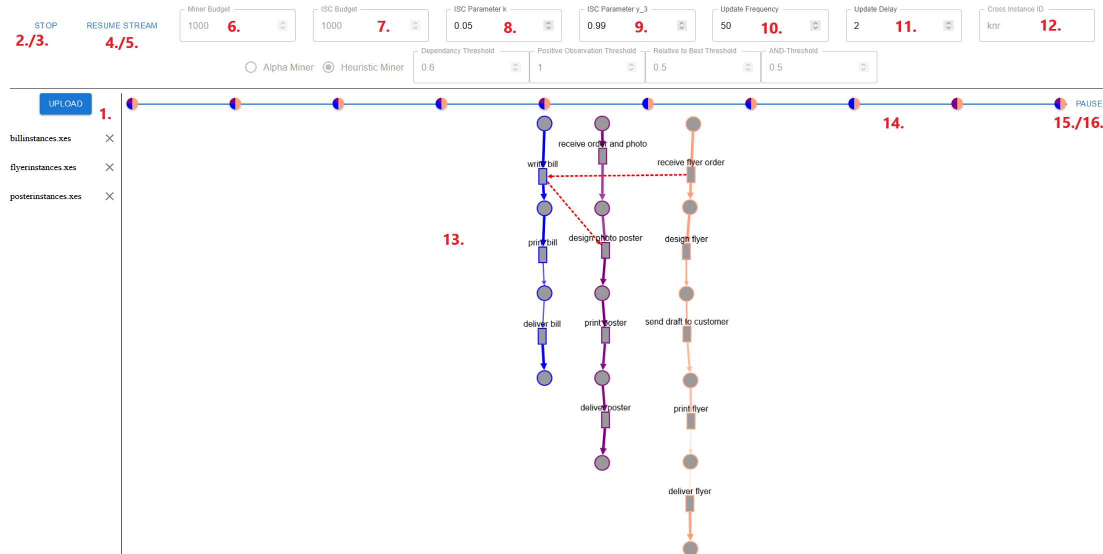


Figure 4.1: A screenshot of TOID during runtime.

Input.
1. Upload: The upload button allows the user to submit up to 10 *.XES* files. These files must possess a valid XES structure [XES] and every therein contained event must own all necessary attributes as described in Section 2.1. Uploaded files are displayed in a list view below the upload button and can be removed by clicking the X next to the filename.

2. Start: Once the start button is pressed, all selected files and option settings are transmitted to the backend. Here, the files — as detailed in Section 2.1 — are internally converted to simulate an event stream, the algorithms are applied and the output is sent back.

3. Stop: When TOID is running, the start button turns into a stop button. This can be clicked to completely halt all backend activity.

4. Pause Stream: The pause stream button enables the user to temporarily suspend the internal event stream. No new event can be emitted during this time. This functionality

serves to facilitate testing. It allows the user to take as much time as needed to fully explore the Petri net in detail, which would otherwise not be possible given a real stream.

5. Resume Stream: When the stream is paused, the unpause button replaces the pause button allowing the user to resume the internal event stream.

Options.
6. Miner Budget: This option permits the user to set the maximum budget to be used for the Lossy Counting with Budget in the online process mining algorithm. [Nav+20]

7. ISC Budget: Similar to 6., the user can set the maximum budget for the LCB in the ISC discovery algorithm, see Section 2.4 and Algorithm 2.

8. ISC Parameter k: This feature allows the user to set the algorithm parameter κ. The parameter accepts a number between [0, 0.5) to be used in the filtering function of Algorithm 1 and 2.

9. ISC Parameter y_3: This field allows the user to set the algorithm parameter γ_3. The parameter accepts a number between [0, 1] to be used in the filtering function of Algorithm 1 and 2.

10. Update Frequency: The update frequency determines how often the backend performs a filter and subsequently sends the output to the frontend. An update frequency of 1 means an update is sent after every single event. A value of 50 means an update is sent after every 50 events.

11. Update Delay: This field controls how long is waited in seconds between sending updates. An update frequency of 50 together with an update delay of 2 means that every 50 events the update is sent and 2 seconds are waited before sending the update of the next 50.

Both the update frequency (10.) and the update delay (11.) fields provide the user with further freedom and versatility while testing. If so desired, one could set the frequency to 1 and the delay to 10 which results in an update being shown after every incoming event but with a long enough pause for the user to fully acknowledge and understand the changes.

12. Cross Instance ID: Because the attribute that represents the unique case identifier might be named differently depending on what tool the log was created with, this option exists to allow the user to specify the attribute name whose value should be used as the *uid*.

Output.
13. Graph: Most of TOIDs space is designated to the canvas which displays the graph. The resulting graphs are configured to grow top to bottom. All transitions and places within the graphs are movable. Additionally, a link can be hovered over to view their exact width and opacity percentages.

14. Timeline: The timeline component above the graph keeps snapshots of the last 10 ISC states. These snapshots are visually indicated in the form of multicolored circles. The color ratio expresses which logs are involved and to which extent. For example, the coloring of the leftmost snapshot in Figure 4.1 denotes that the log corresponding to the color orange was involved in half of the ISC link changes at that time. While blue and purple had an equal 25% percent participation in link changes. Hovering over the snapshots also displays their timestamp and how many ISC links were added or removed. Each of these snapshots can be clicked to display the graph as it was at that point in time. Further, the *fade-in* and *fade-out* animations are replayed so that a user might gain a better understanding of the updates that were made at the time. While viewing such a snapshot the stream does not pause, only the graphical view does. In order to return to viewing the latest graph, the live button (16.) can be pressed.

15. Pause: The pause button next to the timeline can be used to freeze the current frontend view. Unlike the pause stream button (4.) this option does not halt the internal stream or stop the backend from sending any updates. The client will continuously receive updates in the background. This is once more a feature to aid testing by allowing the user to closely examine a specific frame without halting the progress of the algorithm.

16. Live: When in a paused view, the live button replaces the pause button (15.) A user may press it to return to the live view of the stream. Hovering over either the pause or live button displays the timestamp of the most recently received update.

Furthermore, TOID offers the choice of using the Alpha Miner or the Heuristics Miner. In the latter case, additional option boxes appear which allow all thresholds for the Heuristics Miner to be set individually. Moreover, some options are changeable during runtime, such as both ISC parameters as well as the update frequency and update delay. This provides greater flexibility after the start button was pressed. Otherwise the whole stream would have to be restarted any time one would want to test out different ISC parameters or change the streams emission rate. At the same time, certain input options become greyed out and inaccessible once the stream has been started. This is because they either make no sense to change during runtime, such as the Cross Instance ID field which is only ever used to create the stream, or because a post-start change would cause problems within the algorithm, like changing the Miner Budget or ISC Budget.

4.2 Planned Features

Due to time constraints, certain features were left out for the time being. In the future, TOID would benefit from allowing the user to specify multiple event streams as the primary input source. One possibility of realizing this is through the CPEE [CPEE], allowing the users to create or publish streams which can then be specified in TOID. This input option mirrors reality far more closely and enables anyone to experiment on real world stream data sets. This would be a direct alternative to the current file upload system, but would not replace this current feature.

Additionally, the current implementation only allows for one backend process that can be viewed from every browser. Eventually, multiple users should be able to run separate experiments simultaneously. In affiliation with this, either an account or stream ID system would be useful to allow multiple devices to observe the same specified stream.

Chapter 4 Summary and Conclusion.
In summary, the proof-of-concept implementation TOID was introduced for research and testing purposes in the field of online ISC discovery. It combines all algorithms and techniques from the previous two chapters into one web-based tool. In the beginning, important design decision were elaborated on, in addition to the application workflow. Following this, all components of TOID and their functionality were described in detail all. Lastly, future features which would drastically increase the applications utility were laid out. Successfully running sample scenarios through TOID proved that it is a viable instrument for online ISC discovery.

5 Evaluation

For the sake of direct comparison, the evaluation was performed on the same two examples as used in [WSR20].

The first is an artificially generated set of three logs: *flyerinstances.xes*, *billinstances.xes* and *posterinstances.xes*. Flyer- and posterinstances are comprised of 900 instances while billinstances encompasses 1800 instances. Each log contains between 3 and 5 activities which are all connected in a linear fashion and without loops of any kind. The cross instance ID for these logs is “knr” and the intended true ISC are “receive flyer order” $\rightarrow$ “write bill” and “receive order and photo” $\rightarrow$ “write bill” (Fig. 5.1a).

The second is taken from the manufacturing industry and is a real-life example. It is composed of 9 files that detail “the production and quality assurance of valve lifters” [WSR20]. This example also only consists of linear process models, however there are some loops of length 1. Its cross instance ID is “machine_merge_identifier” and there should be ISC connecting all processes to one another according to [WSR20].

Both sets of logs, along with the evaluation data of the offline algorithm can be found here http://bit.ly/2lztLv6 under data sets.

5.1 Experiment Setup

The process models were mined using the Heuristics Miner with a dependency threshold of 0.45, a positive observations threshold of 1 and a relative to best threshold of 0.4. These are the recommended default values as described by [WDD06]. Additionally, the all-activities-connected heuristic was used. These thresholds differ slightly from those used by [WSR20], but because of the all-activities-connected heuristic the resulting process models remain the same.

In order to compare the impact of budget on the result of the algorithm all tests were executed with three different ISC budgets. All measurements were conducted with the budget levels of 100, 1000 and 10000 which represent a major, minor and no constraint on the algorithm respectively. This is interesting to observe in order to find out how drastically, if at all, the results differ between budget levels. The hope is that this insight provides a better understanding of how to allocate the appropriate amount of

budget. During all measurements the miner budget, however, remained constant and non-constrictive as to maintain consistency while testing.

For the purpose of evaluating the implemented algorithm, the following three metrics were measured with regards to ISC discovery: *precision* $\in [0, 1]$, *recall* $\in [0, 1]$ and *computation time* in seconds. Exactly these three measurements were chosen in order to directly compare the newly proposed online algorithm to its offline counterpart (Algorithm 3 in [WSR20]).

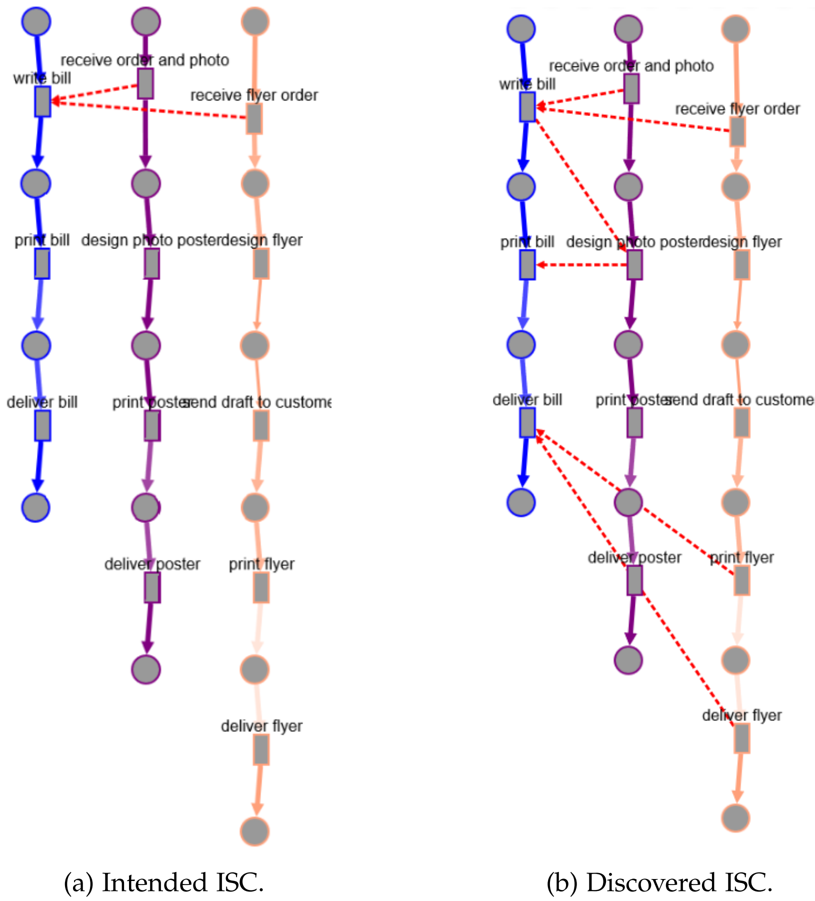
Precision in this case describes how many of the discovered ISC links are actually meant to be there. It is calculated by dividing the number of true positives by the addition of true and false positives. For example, if our graph shows 4 ISC links but we know that only two of these links are correct then the precision in this case would be $\frac{2}{4} = 0.5$.

$$Precision = \frac{\#TP}{\#TP + \#FP}$$

Recall, on the other hand, informs us about how many ISC are present compared to how many ISC should be present. It is calculated by dividing the number of true positives by the addition of true positives and false negatives. For instance, if the algorithms discovered 1 ISC link but we know that there should be 4 in total then the recall would be $\frac{1}{4} = 0.25$.

$$Recall = \frac{\#TP}{\#TP + \#FN}$$

Lastly, the *computation time* was measured by calling python's *time.time()* method before and after creating the stream and letting the algorithm run. This results in two time measurements. *Preprocessing* is the time it took to parse and merge the logs as described in Section 2.1. *Runtime* is the time it took for the algorithm to complete, i.e. consume every element of the stream and perform one final *filter()*. An implementation of the offline algorithm was also timed for comparison purposes. All time measurements were not directly taken from TOID itself, but rather a separate, bare-bones test bench with the intention of reducing network overhead and runtime variation. Moreover, this separate test bench allowed for both algorithms to be implemented side by side in the exact same environments, making it ideal for a direct comparison. All measurements were carried out on an AMD Ryzen 7 3700X @ 3.6 GHz with 16GB of RAM @ 2133 MHz and using a κ of 0 and a γ_3 of 1. The parameters κ and γ_3 were once again selected to best compare with [WSR20].



(a) Intended ISC.

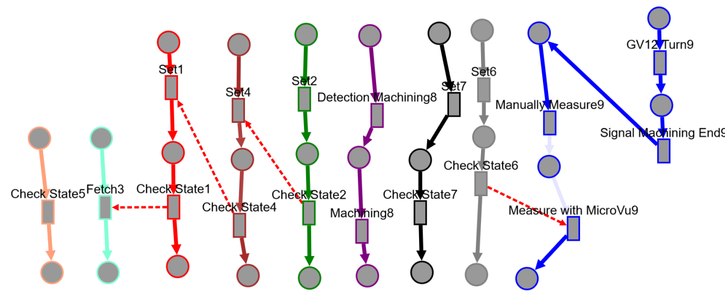
(b) Discovered ISC.

Figure 5.1: Truly intended ISC compared to actually discovered ISC using $\kappa = 0.05$ and $\gamma_3 = 0.99$.

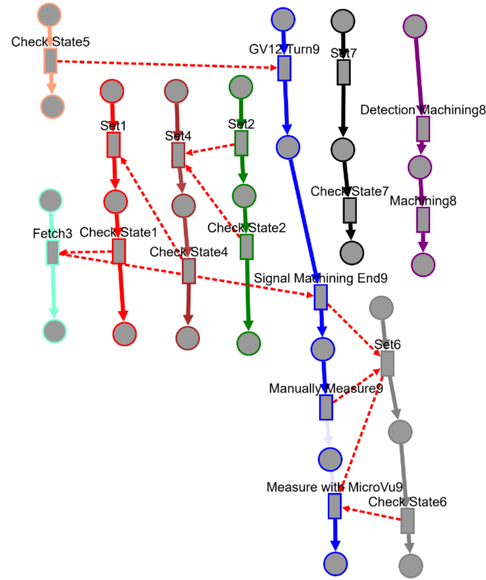
5.2 Experiment Results and Assessment

The resulting tables with precision and recall, based on various combinations of κ and γ_3, can be viewed below. The main focus is Table 5.1 where the artificial example was examined in unison with a high enough budget as to not constrain the ISC algorithm. This allows us to directly compare the results to that of the offline algorithm — specifically Table A.3 in [WSR20]. While doing so we find that all columns, except for the first, are the exact same. In other words, the online algorithm achieves the same precision and recall and delivers the same results as its offline predecessor. The occasional discrepancy is explained by a slight difference in implementation. In the paper detailing the offline algorithm [WSR20], the measurements were made based

on an implementation where in line 4 of the filter function the equal sign of the greater-equals was left out. Therefore, $\geq$ became $>$ and certain ISC were filtered out. This difference is especially profound in the manufacturing example. Here the online implementation with the greater-equals outputs significantly more ISC, as illustrated by Figure 5.2. In the artificial example, 6 ISC have a γ_3 value of 1. These are subsequently filtered out in [WSR20] which explains the precision of 1 and recall of 0 in their first column of Table A.3. By removing this equal sign in the filter function of the online implementation, all results consequently completely overlap with those of the offline implementation.



(a) ISC results using $>$ as implemented in [WSR20].



(b) ISC results using $\geq$ as implemented in TOID.

Figure 5.2: Discovered ISC from the manufacturing example using either $>$ or $\geq$ with $\kappa = 0$ and $\gamma_3 = 1$.

Comparing Budgets.
Comparing the three tables 5.1, 5.2 and 5.3 we can observe the impact of changing the ISC budget. At the lowest budget level the least amount of ISC are discovered, because less frequently observed ones are deleted from the data structure. In this particular example, this leads to a precision score of at least double compared to its higher budget correspondents. At the same time however, its recall scores for a γ_3 of 1, 0.99 and 0.95 are 0 because there are no ISC being outputted. Therefore, just because the precision scores happen to be higher it does not mean that a very low budget is optimal. The medium budget level manages to discover almost all ISC links and only differs from the offline algorithm in the first and second column. As expected, the highest budget level acts as if there were no budget and coincides with the offline algorithm.

κ \ γ₃	1	0.99	0.95	0.8	0.6	0.4	0.2
0	p = 1/3 r = 1	p = 1/3 r = 1	p = 2/7 r = 1	p = 1/4 r = 1	p = 1/4 r = 1	p = 1/5 r = 1	p = 1/6 r = 1
0.05	p = 1/3 r = 1	p = 1/3 r = 1	p = 2/7 r = 1	p = 1/4 r = 1	p = 1/4 r = 1	p = 1/5 r = 1	p = 1/6 r = 1
0.1	p = 1/3 r = 1	p = 1/3 r = 1	p = 2/7 r = 1	p = 1/4 r = 1	p = 1/4 r = 1	p = 1/5 r = 1	p = 1/6 r = 1
0.2	p = 1/3 r = 1	p = 1/3 r = 1	p = 2/7 r = 1	p = 1/4 r = 1	p = 1/4 r = 1	p = 1/5 r = 1	p = 1/6 r = 1
0.3	p = 1/3 r = 1	p = 1/3 r = 1	p = 2/7 r = 1	p = 1/4 r = 1	p = 1/4 r = 1	p = 1/5 r = 1	p = 1/6 r = 1
0.4	p = 1/3 r = 1	p = 1/3 r = 1	p = 2/7 r = 1	p = 1/4 r = 1	p = 1/4 r = 1	p = 1/5 r = 1	p = 2/13 r = 1

Table 5.1: Precision and recall for various κ and γ_3 with a budget of 10000.

κ \ γ₃	1	0.99	0.95	0.8	0.6	0.4	0.2
0	p = 0 r = 0	p = 2/5 r = 1	p = 2/7 r = 1	p = 1/4 r = 1	p = 1/4 r = 1	p = 1/5 r = 1	p = 1/6 r = 1
0.4	p = 0 r = 0	p = 2/5 r = 1	p = 2/7 r = 1	p = 1/4 r = 1	p = 1/4 r = 1	p = 1/5 r = 1	p = 2/13 r = 1

Table 5.2: Precision and recall for various κ and γ_3 with a budget of 1000.

κ \ γ₃	1	0.99	0.95	0.8	0.6	0.4	0.2
0	p = 1 r = 0	p = 1 r = 0	p = 1 r = 0	p = 1 r = 1	p = 2/3 r = 1	p = 2/3 r = 1	p = 1/3 r = 1
0.4	p = 1 r = 0	p = 1 r = 0	p = 1 r = 0	p = 1 r = 1	p = 2/3 r = 1	p = 2/3 r = 1	p = 1/3 r = 1

Table 5.3: Precision and recall for various κ and γ_3 with a budget of 100.

Comparing Runtime.
When it comes to the runtime of the online algorithm, it performs very similarly to its offline counterpart (see Table 5.4). For the smaller manufacturing example, which contains 2546 events in total, the difference in measured time for the algorithms is almost negligible — 0.0003 seconds. Yet, this does change in the larger artificial example of 30 646 events. Here, the online algorithm seems to be moderately slower, resulting in a 0.0462 second difference in completion time. Nevertheless, the computation times of the online algorithm are still within an acceptable range and with some optimisation it is perfectly viable for practical use. The biggest slowdown in either case is parsing of the files and preparing the simulated stream, as illustrated by the two *Preprocessing* columns. Using an actual stream as the input would naturally eliminate this overhead.

	Offline Algorithm		Online Algorithm	
	Preprocessing	Runtime	Preprocessing	Runtime
Artificial Example	7.6559 s	0.0270 s	7.7678 s	0.0732 s
Manufacturing Example	6.1103 s	0.0143 s	6.0795 s	0.0146 s

Table 5.4: Comparison of algorithm duration using $\kappa = 0$ and $\gamma_3 = 1$.

Additional Observations.
One last observation I would like to point out is that during the runtime of the algorithm, the amount of discovered ISC often fluctuates drastically. They are not linearly discovered and once discovered, they rarely remain in place for the rest of the stream. Newly added ISC seem to disappear and reappear frequently. Therefore, the snapshot of found ISC at time A might significantly differ to that of time B which both might substantially differ from the final result. I hypothesize that when an ISC between activities a_1 and a_2 is first found its ordActivities[(a_1, a_2)][“count”], countEvs[a_1] and countEvs[a_2] values are all quite low. This means that even when a small change is made (say countEvs[a_1] is increased by 1), the relative impact this change has is quite high. This changing power is then significantly reduced later on in the stream when all counts are higher. This might explain the initially sensitive nature of newly discovered ISC and the constant fluctuations in ISC count throughout time.

Chapter 5 Summary and Conclusion.
In conclusion, this evaluation has demonstrated that the newly designed online version of the existing category 3 (order of activity execution) ISC discovery algorithm delivers the same ISC results in a comparably timed fashion. However, a slight distinction in implementation was identified which led to an initial divergence in output. Nonetheless, when this implementation difference was accommodated for all ISC, precision and recall results lined up perfectly.

6 Conclusion

Two accompanying research questions were posed in the introduction of this thesis. These have been addressed throughout this work and are summarized up below:

RQ_1: How to design an online algorithm for discovering ordering ISC?

We have seen that the online version of the ordering ISC algorithms faces additional challenges that its offline correspondent does not, such as limited storage as well as the inability to look into the future. Both of these challenges were overcome and it was shown, through the design of an algorithm and ultimately through its implementation, that such an algorithm is feasible. Moreover, its viability was demonstrated as this online algorithm produces the same results as its counterpart given a comparable implementation — i.e. both algorithms either use $>$ or $\geq$ in line 4 the filter function of [WSR20]. Prior to this thesis the online setting was limited to a handful of process mining algorithms; now the doors have been opened to explore Instance Spanning Constraints in a stream based environment.

RQ_2: How to visualize the results of an online algorithm for discovering ordering ISC?

The chapter dedicated to visualization (Chapter 3) directly answers RQ_2 and provides methods to visually highlight important information about the process model and the discovered ISC in a way that is instantaneously absorbable by an observer. Furthermore, it affords possible points of flexibility, such as choosing between using colors or borders, to better reflect potentially diverse use cases. Despite little research interest to date in visualizing online process mining, this field provides real end-user benefits.

Subsequently, online process mining, online discovery of ISC category 3 and the aforementioned visualization techniques were all combined into a singular web based proof-of-concept implementation as explained in Chapter 4.

Real Life Application and Impact.
This thesis serves as the first step towards creating online counterparts for all ISC categories. Not only are algorithms that function in an online setting altogether useful in practical and commercial scenarios, as in real life the input data is usually present in the form of a stream, but specifically ISC play a crucial role in preventing possibly fatal mistakes across a plethora of professional industries.

Furthermore, the tool will be improved to include all features as presented in Section 4.2 and finally presented to the process mining community over a platform similar to the ICPM Demo Track [ICPM]. Hopefully by doing so it will increase the visibility and access to this topic for other researchers. The intent is to encourage the expansion of this fairly new field of research.

Future Work.
The next big step is developing online algorithms for the other three ISC categories: *Simultaneous execution of activities*, *Constrained activity execution* and *Non-concurrent execution of activities*. *Constrained activity execution* would hold the highest priority as it seems to be the most common type found in real life [WSR20]. Another aspect to explore here would be how to visually differentiate between the various categories so that a user might easily tell apart the different constraint links.

In addition, an interesting phenomenon to inquire into would be recurring concept drift in the context of ISC. When it comes to online process mining concept drift has been examined in various papers [SR18], [Cer+20], [CG12], [Mag+13]. However, it would be compelling to explore how concept drift presents itself in ISC. Moreover, is it possible to detect whether this drift occurs repeatedly, for example certain constraints that reappear every winter and then disappear in the spring.

On a similar note, the constant fluctuation, as described at the end of the evaluation (Chapter 5), could also be explored further. So far it is uncertain whether there is a pattern, whether these fluctuation only occur for ISC of category 3 or whether they can be smoothed out.

Lastly, real life data streams are often far from perfect. They are subject to packet loss, duplicate events, missing attributes, ordering anomalies etc., all of which the presented algorithm is incapable of dealing with. And while this thesis did not delve into dealing with imperfections, if practical and commercial products are to be developed for this field then an algorithm that is capable of dealing with these kinds of anomalies must be developed in the future. Certain techniques for dealing with anomalous data do exist [AWS20], [KF21], [Tav+19], but once again only for online process mining and not yet for online ISC discovery.

Bibliography

[AMZN] *Amazon S3 – Two Trillion Objects, 1.1 Million Requests / Second | AWS News Blog*. https://aws.amazon.com/blogs/aws/amazon-s3-two-trillion-objects-11-million-requests-second/. Accessed: 2022-09-05.

[AWM04] W. M. P. van der Aalst, T. Weijters, and L. Maruster. "Workflow Mining: Discovering Process Models from Event Logs." In: *IEEE Trans. Knowl. Data Eng.* 16.9 (2004), pp. 1128–1142. DOI: 10.1109/TKDE.2004.47.

[AWS20] A. Awad, M. Weidlich, and S. Sakr. "Process Mining over Unordered Event Streams." In: *2nd International Conference on Process Mining, ICPM 2020, Padua, Italy, October 4-9, 2020*. Ed. by B. F. van Dongen, M. Montali, and M. T. Wynn. IEEE, 2020, pp. 81–88. DOI: 10.1109/ICPM49681.2020.00022.

[BSA14] A. Burattin, A. Sperduti, and W. M. P. van der Aalst. "Control-flow discovery from event streams." In: *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2014, Beijing, China, July 6-11, 2014*. IEEE, 2014, pp. 2420–2427. DOI: 10.1109/CEC.2014.6900341.

[Bur+15] A. Burattin, M. Cimitile, F. M. Maggi, and A. Sperduti. "Online Discovery of Declarative Process Models from Event Streams." In: *IEEE Transactions on Services Computing* 8.6 (2015), pp. 833–846. DOI: 10.1109/TSC.2015.2459703.

[Cer+20] P. Ceravolo, G. Marques Tavares, S. B. Junior, and E. Damiani. "Evaluation Goals for Online Process Mining: a Concept Drift Perspective." In: *IEEE Transactions on Services Computing* (2020), pp. 1–1. DOI: 10.1109/TSC.2020.3004532.

[CG12] J. Carmona and R. Gavaldà. "Online Techniques for Dealing with Concept Drift in Process Mining." In: *Advances in Intelligent Data Analysis XI - 11th International Symposium, IDA 2012, Helsinki, Finland, October 25-27, 2012. Proceedings*. Ed. by J. Hollmén, F. Klawonn, and A. Tucker. Vol. 7619. Lecture Notes in Computer Science. Springer, 2012, pp. 90–102. DOI: 10.1007/978-3-642-34156-4_10.

[CPEE] *CPEE - The Engine*. https://cpee.org/. Accessed: 2022-09-05.

[ERF16] J. Evermann, J. Rehse, and P. Fettke. “Process Discovery from Event Stream Data in the Cloud - A Scalable, Distributed Implementation of the Flexible Heuristics Miner on the Amazon Kinesis Cloud Infrastructure.” In: *2016 IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2016, Luxembourg, December 12-15, 2016*. IEEE Computer Society, 2016, pp. 645–652. DOI: 10.1109/CloudCom.2016.0111.

[Has+15] M. Hassani, S. Siccha, F. Richter, and T. Seidl. “Efficient Process Discovery From Event Streams Using Sequential Pattern Mining.” In: *IEEE Symposium Series on Computational Intelligence, SSCI 2015, Cape Town, South Africa, December 7-10, 2015*. IEEE, 2015, pp. 1366–1373. DOI: 10.1109/SSCI.2015.195.

[ICPM] *Demo Track – Process Mining Conference 2022*. https://icpmconference.org/2022/call-for-demo-papers/. Accessed: 2022-09-02.

[IFR18] C. Indiono, W. Fdhila, and S. Rinderle-Ma. “Evolution of Instance-Spanning Constraints in Process Aware Information Systems.” In: *On the Move to Meaningful Internet Systems. OTM 2018 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2018, Valletta, Malta, October 22-26, 2018, Proceedings, Part I*. Ed. by H. Panetto, C. Debruyne, H. A. Proper, C. A. Ardagna, D. Roman, and R. Meersman. Vol. 11229. Lecture Notes in Computer Science. Springer, 2018, pp. 298–317. DOI: 10.1007/978-3-030-02610-3_17.

[ISC] *ISC Mining Algorithm*. http://isc-mining.wst.univie.ac.at/. Accessed: 2022-04-05.

[KF21] P. Krajsic and B. Franczyk. “Variational Autoencoder for Anomaly Detection in Event Data in Online Process Mining.” In: *Proceedings of the 23rd International Conference on Enterprise Information Systems, ICEIS 2021, Online Streaming, April 26-28, 2021, Volume 1*. Ed. by J. Filipe, M. Smialek, A. Brodsky, and S. Hammoudi. SCITEPRESS, 2021, pp. 567–574. DOI: 10.5220/0010375905670574.

[Kun+10] S. Kunz, T. Fickinger, J. Prescher, and K. Spengler. “Managing Complex Event Processes with Business Process Modeling Notation.” In: *Business Process Modeling Notation - Second International Workshop, BPMN 2010, Potsdam, Germany, October 13-14, 2010. Proceedings*. Ed. by J. Mendling, M. Weidlich, and M. Weske. Vol. 67. Lecture Notes in Business Information Processing. Springer, 2010, pp. 78–90. DOI: 10.1007/978-3-642-16298-5_8.

[Mag+13] F. M. Maggi, A. Burattin, M. Cimitile, and A. Sperduti. “Online Process Discovery to Detect Concept Drifts in LTL-Based Declarative Process Models.”

In: *On the Move to Meaningful Internet Systems: OTM 2013 Conferences - Confederated International Conferences: CoopIS, DOA-Trusted Cloud, and ODBASE 2013, Graz, Austria, September 9-13, 2013. Proceedings*. Ed. by R. Meersman, H. Panetto, T. S. Dillon, J. Eder, Z. Bellahsene, N. Ritter, P. D. Leenheer, and D. Dou. Vol. 8185. Lecture Notes in Computer Science. Springer, 2013, pp. 94–111. DOI: 10.1007/978-3-642-41030-7_7.

[MM02] G. S. Manku and R. Motwani. “Approximate Frequency Counts over Data Streams.” In: *Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002, Hong Kong, August 20-23, 2002*. Morgan Kaufmann, 2002, pp. 346–357. DOI: 10.1016/B978-155860869-6/50038-X.

[Nav+20] N. Navarin, M. Cambiaso, A. Burattin, F. M. Maggi, L. Oneto, and A. Sperduti. “Towards Online Discovery of Data-Aware Declarative Process Models from Event Streams.” In: *2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020*. IEEE, 2020, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9207500.

[ProM] *start | ProM Tools*. https://www.promtools.org/doku.php. Accessed: 2022-04-05.

[RR98] W. Reisig and G. Rozenberg, eds. *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*. Vol. 1491. Lecture Notes in Computer Science. Springer, 1998. ISBN: 3-540-65306-6. DOI: 10.1007/3-540-65306-6.

[SR18] F. Stertz and 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: CoopIS, C&TC, and ODBASE 2018, Valletta, Malta, October 22-26, 2018, Proceedings, Part I*. Ed. by H. Panetto, C. Debruyne, H. A. Proper, C. A. Ardagna, D. Roman, and R. Meersman. Vol. 11229. Lecture Notes in Computer Science. Springer, 2018, pp. 318–335. DOI: 10.1007/978-3-030-02610-3_18.

[Tav+19] G. M. Tavares, P. Ceravolo, V. G. T. da Costa, E. Damiani, and S. B. Junior. “Overlapping Analytic Stages in Online Process Mining.” In: *2019 IEEE International Conference on Services Computing, SCC 2019, Milan, Italy, July 8-13, 2019*. Ed. by E. Bertino, C. K. Chang, P. Chen, E. Damiani, M. Goul, and K. Oyama. IEEE, 2019, pp. 167–175. DOI: 10.1109/SCC.2019.00037.

[WDD06] A. Weijters, W. M. van Der Aalst, and A. A. De Medeiros. “Process mining with the heuristics miner-algorithm.” In: *Technische Universiteit Eindhoven, Tech. Rep. WP* 166.July 2017 (2006), pp. 1–34.

[Wik22] Wikipedia contributors. *Component (graph theory) — Wikipedia, The Free Encyclopedia*. [Online; accessed 3-September-2022]. 2022.

[WR17] K. Winter and S. Rinderle-Ma. "Discovering Instance-Spanning Constraints from Process Execution Logs Based on Classification Techniques." In: *21st IEEE International Enterprise Distributed Object Computing Conference, EDOC 2017, Quebec City, QC, Canada, October 10-13, 2017*. Ed. by S. Hallé, R. Villemaire, and R. Lagerström. IEEE Computer Society, 2017, pp. 79–88. DOI: 10.1109/EDOC.2017.20.

[WSR20] K. Winter, F. Stertz, and S. Rinderle-Ma. "Discovering instance and process spanning constraints from process execution logs." In: *Inf. Syst.* 89 (2020), p. 101484. DOI: 10.1016/j.is.2019.101484.

[XES] *IEEE 1849-2016 XES Standard*. https://xes-standard.org/. Accessed: 2022-08-11.

[ZCH19] R. Zaman, A. Cuzzocrea, and M. Hassani. "An Innovative Online Process Mining Framework for Supporting Incremental GDPR Compliance of Business Processes." In: *2019 IEEE International Conference on Big Data (IEEE BigData), Los Angeles, CA, USA, December 9-12, 2019*. Ed. by C. Baru, J. Huan, L. Khan, X. Hu, R. Ak, Y. Tian, R. S. Barga, C. Zaniolo, K. Lee, and Y. F. Ye. IEEE, 2019, pp. 2982–2991. DOI: 10.1109/BigData47090.2019.9005705.

[ZDA17] S. J. van Zelst, B. F. van Dongen, and W. M. P. van der Aalst. "Event Stream-Based Process Discovery using Abstract Representations." In: *CoRR* abs/1704.08101 (2017). arXiv: 1704.08101.