Object-Centric Streaming-Based Process Discovery

Lukas Liss, Nina Löseke, and Wil M. P. van der Aalst

RWTH Aachen University, Aachen, Germany
{liss,wvdaalst}@pads.rwth-aachen.de,nina.loeseke@rwth-aachen.de

Abstract. Traditional process mining analyzes case-centric processes where cases evolve in isolation. However, in real-world processes, most events involve multiple objects. Object-centric process mining supports processes where objects are intertwined, but results in larger datasets that quickly face scalability challenges. While online process mining with limited memory can address this, existing approaches are restricted to traditional process mining. We propose a framework for stream-based object-centric process discovery, supporting object-centric directly-follows graphs, object-centric Petri nets, and temporal object type models. The publicly accessible implementation is evaluated using open-access object-centric event logs. The runtime evaluation shows that the stream item processing time grows linearly with the buffer size, but is feasible for high data velocity. The structural comparison to offline models shows the significant importance of cache management strategies that match the process characteristics for the quality of the resulting process models.

Key words: Object-Centric Event Streams, Online Process Discovery, Object-Centric Petri Nets, Object-Centric DFG, TOTeM model

1 Introduction

Process mining generates insights into processes by analyzing event data from information systems [16]. Traditional techniques focus on single case identifiers, limiting their ability to capture real-world organizational processes involving multiple objects that are involved in shared events [9]. Object-centric process mining has gained popularity by enabling the analysis of processes involving multiple objects with shared events [17]. This paradigm shift allows analyzing complex organization-wide processes where objects of different types participate in various activities. However, storing event data for case-centric processes already poses scalability challenges [20]. For object-centric processes designed to capture comprehensive system behavior, this challenge intensifies. Object-centric event data, including events with objects, attributes, and object-to-object relations, results in significantly larger data volumes, making single-file storage, as assumed by current offline approaches, often infeasible. To address this scalability challenge, online analysis of object-centric processes is needed. We propose a streaming framework for object-centric process discovery that operates

Table 1: Excerpt of the running example object-centric event log, showing the event-to-object relations (left) and the object-to-object relations (right).

Time	Event ID	Activity	Object Type				
			Customer	Order	Item	Parcel	Shipping label
1.1.25	e_1	Place order	c_1	o_1	i_1, i_2		
2.1.25	e_2	Pick item		o_1	i_2		
3.1.25	e_3	Pick item		o_1	i_1		
4.1.25	e_4	Package		o_1		p_1	
5.1.25	e_5	Label					l_1
...							

Time	Source	Target	Qualifier
6.1.25	p_1	l_1	"labeled"
...			

on object-centric event streams. Our approach adapts existing models to work with limited memory footprints while maintaining essential characteristics for meaningful process insights. To visualize the problem and explain our framework, we introduce a running example of an order management process where orders are placed, items picked, orders packed in parcels, and labels created. Labels connect to parcels via object-to-object relations. The resulting events and object-to-object relations in Table 1 follow the OCEL 2.0 standard [4]. All process characteristics of the OCEL 2.0 standard are supported, which includes the features described in the OCED standard [10].

It is easy to see that these tables will quickly gain a size that is difficult to store in one file and even harder to analyze at once, particularly with many orders arriving at a fast pace in real-world scenarios. To obtain insights into processes where data arrives at such high velocity, new streaming-based approaches are needed. Here, we focus on a common first analysis step: process discovery.

This paper makes five contributions: (1) we define object-centric event data streams; (2) we define intermediate buffers that capture relevant process behavior with fixed memory footprints; (3) we propose cache replacement strategies and object (type) priority policies for buffer management; (4) we adapt the discovery algorithms of three prominent object-centric process models, Object-Centric Petri Nets (OCPN), Object-Centric Directly-Follows Graphs (OC-DFG), and Temporal Object Type Models (TOTeM), to operate on the proposed streaming representations as a publicly accessible implementation[1]; and (5) we evaluate our stream-based approach using publicly accessible object-centric event logs.

2 Related Work

The presented work belongs to the field of process discovery [1] within process mining [16]. The goal in process discovery is to discover a process model that describes the real process behavior captured in event data, for which different formats exist, such as OCEL 2.0 [4] or event knowledge graphs [8]. To describe the behavior of real-world processes, which tend to contain multiple objects and

[1] `https://github.com/loeseke/object-centric-streaming-discovery`

consist of interacting sub-processes, especially object-centric process models have been developed in recent years. The TOTeM model focuses on temporal relations between object types [14]. Directly-follows graphs and their object-centric variant describe the precedence of activities [3]. Additionally, the identifier-sound object-centric process trees [18] and object-centric causal nets, along with their respective discovery approaches, have been recently proposed. A commonly used model is the object-centric Petri net [17] for which extensions like object-centric Petri nets with identifiers [11] have been introduced.

Existing object-centric discovery algorithms assume that all the data can be analyzed at once in an offline setting. For traditional process mining, extensive data volume and high velocity have already been addressed by multiple researchers. Van Zelst et al. propose the Stream-Based Abstract Representation (S-BAR) architecture for discovering process models from case-centric event streams using buffered abstract representations [20]. Also, online conformance checking approaches have been proposed [19]. Burattin et al. demonstrate that buffering fixed-size event windows is inefficient, proposing instead to condense information into specialized streaming data structures that abstract essential model components [6], similar to the S-BAR approach. Our framework uses the same concept. Extending upon the streaming Heuristics Miner [6], Hassani et al. develop the StrProM approach, which achieves better processing times at similar space requirements [12]. Other approaches support certain fitness guarantees [13] or work with unordered event streams [2]. Various cache replacement policies have been adapted for streaming process mining, including traditional recency and frequency-based approaches that determine which buffered information to retain under memory constraints [15].

However, existing streaming process discovery research focuses on traditional, case-centric process mining. So far, the issue of extensive data volume and velocity has not yet been addressed for object-centric process discovery. To address this research gap, we propose an object-centric streaming-based process discovery approach that can discover three object-centric process models.

3 Preliminaries

An object-centric event log describes events from the universe of events $\mathbb{U}_{ev}$ with activity labels from $\mathbb{U}_{act}$ and timestamps from $\mathbb{U}_{time}$. An event can have attributes from $\mathbb{U}_{attr}$ and can be connected to objects from $\mathbb{U}_{obj}$. Objects belong to an object type from $\mathbb{U}_{otype}$. Given non-overlapping universes, Definition 1 describes an object-centric event log matching the OCEL 2.0 format [4].

Definition 1 (Object-Centric Event Log [4]). *An Object-Centric Event Log (OCEL) is a tuple* $L = (E, O, EA, OA, E2O, O2O, \pi_{time}, \pi_{act}, \pi_{objtype}, \pi_{eatype}, \pi_{oatype}, \pi_{eaval}, \pi_{oaval})$ *where*

- $E \subseteq \mathbb{U}_{ev}$ *is a set of events;* $O \subseteq \mathbb{U}_{obj}$ *is a set of objects;*
- $EA \subseteq \mathbb{U}_{attr}$ *is a set of event attributes;* $OA \subseteq \mathbb{U}_{attr}$ *is a set of object attributes;*

- $E2O \subseteq E \times \mathbb{U}_{qual} \times O$ *is a set of qualified event-to-object relations*
- $O2O \subseteq O \times \mathbb{U}_{qual} \times O$ *is a set of qualified object-to-object relations*
- $\pi_{time} : E \to \mathbb{U}_{time}$ *maps each event to a timestamp*
- $\pi_{act} : E \to \mathbb{U}_{act}$ *maps each event to an activity*
- $\pi_{objtype} : O \to \mathbb{U}_{otype}$ *maps each object to an object type*
- $\pi_{eatype} : EA \to \mathbb{U}_{act}$ *maps each event attribute to an activity*
- $\pi_{oatype} : OA \to \mathbb{U}_{otype}$ *maps each object attribute to an object type*
- $\pi_{eaval} : (E \times EA) \not\to \mathbb{U}_{val}$ *maps an event attribute to a value*
- $\pi_{oaval} : (O \times OA \times \mathbb{U}_{time}) \not\to \mathbb{U}_{val}$ *maps an object attribute to a value*

In an OC-DFG, the graph's nodes represent activities, and its arcs are each associated with an object type. An arc is present if and only if the target activity can directly follow the source activity for an object of the given object type. Additionally, nodes and arcs can be annotated with frequencies or durations. For concrete definitions, we refer to [3].

An OCPN is a bipartite graph of transitions that represent activities and places that are associated with object types. These places restrict the allowed behavior of the Petri net for the given object type. Places can contain tokens that represent objects if they match the associated object type of the place. Firing a transition (representing that an event happens) consumes tokens for all involved objects from input places and creates tokens in all output places. Arcs can be marked as variable arcs, which allows them to consume or produce multiple tokens at once. For concrete definitions, we refer to [17].

The TOTeM model is a graph with object types as nodes and arcs between object types if and only if objects of these types are connected, which means they share an event or have an object-to-object relation [14]. It describes three characteristics of connected object types of a process. First, the log cardinality describes how many objects of one type are connected to how many objects of another type. So in Figure 1, a log cardinality of $1..*$ from order to item means that one order is connected to at least 1 (but possibly multiple) items. Whereas the log cardinality is 1 from item to order because each item belongs only to one order. Second, the event cardinality describes how many objects of each object type are involved in events of the source object type. An event cardinality of 0..1 from order to item would mean that there are events for the order objects that involve no or exactly one item, but there are no events for orders with multiple items. Third, the temporal relation describes how the lifespans of connected objects are related. There are five types of temporal relations: during (■), during inverse (□), precedes (▶), precedes inverse (▷), and parallel (‖). A during (■) relation from type item to type order describes that the lifespans of items are

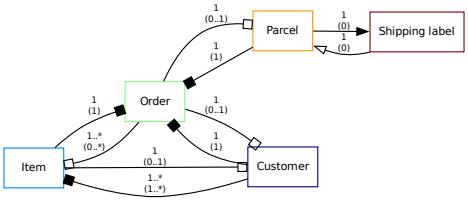

Fig. 1: TOTeM mined with our framework for the running example. Annotated arcs with log cardinalities and event cardinalities in brackets below.

fully contained within the lifespan of connected orders. So there are events with the order before the first event for related items, and there are events with the order after the last event of the item. A precedes ($\blacktriangleright$) relation from parcel to label describes that parcels exist, i.e., occur in events, before the label, and the label appears in events at a later point in time than the parcel's final event. The inverse relations describe exactly the inverse behavior. The parallel ($\|$) relation is the default temporal relation when there is no clear lifespan relation of objects of the connected object types. For precise definitions, we refer to [14].

4 Object-Centric Event Streams

To discover object-centric process models from event streams, we first need to define object-centric event streams. Object-centric event streams are potentially infinite streams of object-centric stream items. Each stream item can represent events, O2O updates, and object-attribute changes. This allows covering all characteristics of OCED, all of OCEL 2.0, and most of the features captured in event knowledge graphs. We define object-centric event streams in Definition 2.

Definition 2 (Object-Centric Event Stream). *An object-centric event stream $S \in (\mathbb{U}_{time} \times ES \times O2OS \times OAUS)^*$ is a possibly infinite sequence of timestamped stream items with:*

- *the set of events $ES = \mathcal{P}(\mathbb{U}_{act} \times \mathcal{P}(\mathbb{U}_{attr} \times \mathbb{U}_{val}) \times \mathcal{P}(\mathbb{U}_{obj} \times \mathbb{U}_{otypes} \times \mathbb{U}_{qual}))$,*
- *the set of O2O updates $O2OS = \mathcal{P}(\mathbb{U}_{obj} \times \mathbb{U}_{otype} \times \mathbb{U}_{qual} \times \mathbb{U}_{obj} \times \mathbb{U}_{otype})$, and*
- *the set of possible object-attribute updates $OAUS = \mathcal{P}(\mathbb{U}_{obj} \times \mathbb{U}_{attr} \times \mathbb{U}_{val})$.*

Let $S(i)$ denote the stream item at position $i \in \mathbb{N}$ in S. The stream is partially ordered based on time, such that for $i, j \in \mathbb{N}$, $S(i) = (t_i, ES_i, O2OS_i, OAUS_i)$, and $S(j) = (t_i, ES_i, O2OS_i, OAUS_i)$ it holds that $i \leq j \Rightarrow t_i \leq t_j$.

This formalization of object-centric event streams enables multiple events, O2O updates, or object-attribute updates to occur concurrently within one stream item. In contrast to van Zelst et al.'s S-BAR architecture [20], our streaming framework considers the concrete timestamps of stream items, which serve as input for model annotation and cache-replacement policies. Currently, our method does handle lateness or missordering but assumes a correct order based on the timestamps.

The behavior stored in object-centric event logs can also be represented in object-centric event streams. Our publicly accessible implementation supports the transformation of OCEL 2.0 event logs into object-centric event streams. The first steam item for the running example event log from Table 1 would be: (“1.1.25”, {(“Place order”, $\emptyset$, {(c_1, “Customer”, “”), (o_1, “Order”, “”), (i_1, “Item”, “”), (i_2, “Item”, “”)})}, $\emptyset$, $\emptyset$)

We use the empty string as the E2O qualifier since the running example does not qualify E2O relations. Since no O2O updates or object-attribute updates occur at this time, the respective sets in the stream-item tuple are empty.

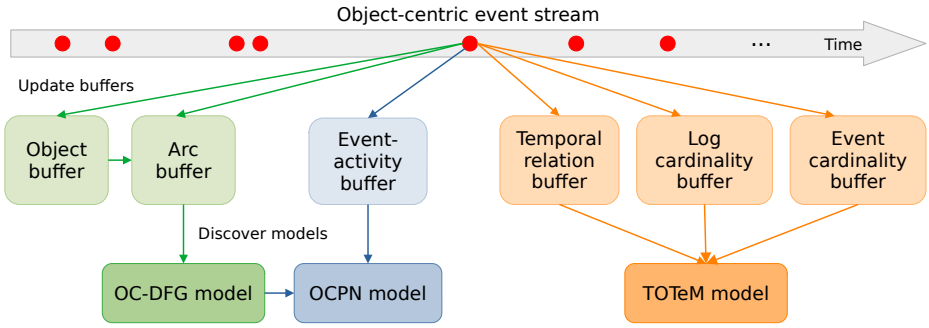

Fig. 2: Overview of proposed streaming framework for the online discovery of object-centric process models.

5 Streaming Framework for Object-Centric Discovery

Our proposed framework to discover object-centric process models from object-centric event streams uses six model buffers that serve as an intermediate representation to capture relevant process characteristics to enable the discovery of OC-DFGs, OCPNs, and TOTeMs. The concept of intermediate buffers with a fixed memory size is similar to case-based streaming frameworks like S-BAR [20]. The size of the buffers can be selected by process owners and is fixed. With each new object-centric event stream item arriving, the model buffers are updated, or an entry is added to the buffers. If the buffer is already full, the cache replacement strategy decides which buffer element to replace. The cache replacement strategies are unaware of the object-type-specific characteristics, which can lead to an unbalanced distribution of object types in the buffers. To compensate for this, process owners can optionally select object-centric priority policies. These can influence which buffer element is replaced based on object (type) characteristics observed in the log. The optional buffer synchronization ensures that information about objects or object types, which are deleted completely from one buffer, is also deleted from all other buffers. Figure 2 shows an overview of the framework.

5.1 Model buffers

The model buffers are designed to capture the relevant information to construct the three object-centric process models. Note that some buffers, like the object, arc, and event-activity buffer, store information separated by object type. This means that each buffer element only contains information related to one object type. These buffers can be split up into object-type-specific buffers if all object types are known initially. Since this assumption may not hold, the framework and our implementation use buffers with mixed object types by default. But the framework and our implementation also support individual type buffers. The benefit of type-specific buffers is that one can adjust the desired replacement strategies in more detail to the characteristics of the object type.

Table 2: Mixed OC-DFG node and arc buffers of size 5 are updated for event e_1 from Table 1. New buffer items are highlighted. (Place Order noted as PlOr).

Object Buffer				
	Object	Object type	Activity	Last seen
0	c_1	Customer	PlOr	1.1.25
1	o_1	Order	PlOr	1.1.25
2	i_1	Item	PlOr	1.1.25
3	i_2	Item	PlOr	1.1.25
4				

Arc Buffer					
	Arc	Object	Object type	Target-activity frequency	Activity duration
0	(*null*, PlOr)	c_1	Customer	0.25	*null*
1	(*null*, PlOr)	o_1	Order	0.25	*null*
2	(*null*, PlOr)	i_1	Item	0.25	*null*
3	(*null*, PlOr)	i_2	Item	0.25	*null*
4					

OC-DFG Buffers To discover an OC-DFG, one needs to know which activities can directly follow another activity for a certain object type. Similar to the case-based setting [20], we use two buffers to compute and store this information. With every new object-centric stream item that contains an event, we first update the object buffer and then the arc buffer. The object buffer stores for each object its object type, the last activity that was seen, and the last time it was seen. If an object of an incoming event is already in the buffer, the buffered information is updated; otherwise, a new buffer entry is added.

The arc buffer describes which directly-follows relations were observed for which object (and object type). It also tracks the frequency of the target activity and the duration from the previous event. The arc buffer is updated for each object of arriving events based on the information in the object buffer. If an arriving object is not present in the object buffer, we assume it is the first occurrence of that object and create an initial arc $(null, [activity\ of\ the\ event])$ for the activity. Otherwise, we add an arc from the buffered last-seen activity of the object to the activity of the current event to the buffer and compute the duration based on the last-seen timestamp and the current one. The target activity frequency is computed by $\frac{1}{number\ of\ obj\ in\ event}$ to avoid the convergence issue common to case-centric approaches. Since one new line is added per object, this allows us to compute the total target activity frequency by simply summing up the values in the frequency column per arc. Table 2 shows the resulting two buffers after processing the first event stream item.

The OC-DFG can then be computed by simply grouping the arcs per object type in the arc buffer, summing up the values in the frequency column, and averaging the durations. Thereby, we obtain an OC-DFG annotated with node frequencies and arc durations and frequencies. The arc frequencies correspond to the number of occurrences of each arc in the arc buffer and are thus constrained by the chosen buffer size.

OCPN Buffers The OCPN discovery approach, proposed by Van der Aalst and Berti [17], first discovers case-centric Petri nets per object type and then merges them. For the combination step, it is important to make sure that activities that can involve multiple objects of a certain object type are assigned variable arcs for that type. Given the OC-DFG, which can be computed using the object and arc buffer, we can also compute Petri nets for

each object type. For each object type, we create a case-centric directly-follows graph from the OC-DFG by removing all nodes and arcs that do not relate to the object type. Then we use a variant of the Inductive Miner that discovers a Petri net from directly-follows graphs, which is also implemented in PM4Py [5]. The only remaining information needed to create an OCPN is which activities can consume multiple objects of the same type to assign variable arcs. The event-activity buffer provides this information. For each arriving event, it stores the activity, and for each involved object type, it stores whether multiple objects of that type were involved or not.

Table 3: Event-activity buffer after updates for e_1 and e_2 from Table 1.

Event-Activity Buffer			
	Activity	Object type	Single obj
0	PlOr	Customer	True
1	PlOr	Order	True
2	PlOr	Item	False
3	PiIt	Order	True
4	PiIt	Item	True

TOTeM Buffers The TOTeM miner [14] expects three types of information: (1) The first and last event of each object, to compute the objects lifespan and then the temporal relations, (2) the tuples of connected objects and their object types to compute the log cardinality, and (3) applicable cardinalities between involved object types for each event.

The temporal-relation buffer stores timestamps for objects' first-seen and last-seen events. If objects of incoming events are not yet in the buffer, a new entry with the timestamp as the value for the *First* and *Last* columns is added; otherwise, only the *Last* column is updated.

The log-cardinality buffer is updated for both events and O2O updates. For each pair of objects in the O2O or event, a new element is added to record that these two objects are connected. This allows us to identify for each object type if their objects are connected to 0, 1, or multiple objects of another type.

Table 4: Temporal realtion buffer after event e_1 from Table 1.

Temporal-Relation Buffer				
	Object	Object type	First	Last
0	c_1	Customer	1.1.25	1.1.25
1	o_1	Order	1.1.25	1.1.25
2	i_1	Item	1.1.25	1.1.25
3	i_2	Item	1.1.25	1.1.25
4				

The event-cardinality buffer saves for each event and each directed pair of involved object types all allowed event cardinalities that match this event. So if multiple objects of the target type are contained, only $1..*$ and $0..*$ are stored and not 0 or 1. To compute the event cardinality for a directed arc in the TOTeM model, we can then select the most precise cardinality for a pair of connected types based on a minimum support threshold for the buffered cardinalities. Thereby, one can compute TOTeM models from these three buffers. Figure 1 shows the resulting TOTeM model for the running example.

5.2 Cache Replacement Strategies

When an incoming stream item results in new entries for a buffer that has already reached its maximum capacity, one of the existing entries must be re-

Table 5: Log and event-cardinality buffers after event e_1 from Table 1.

Log-Cardinality Buffer		
	Undirected object-type pair	Undirected object pair
0	(Customer, Order)	(c_1, o_1)
1	(Customer, Item)	(c_1, i_1)
2	(Customer, Item)	(c_1, i_2)
3	(Item, Order)	(i_1, o_1)
4	(Item, Order)	(i_2, o_1)

Event-Cardinality Buffer			
	Directed object-type pair	Event ID	Event cardinalities
0	(Customer, Order)	0	{1, 0..1, 1..∗, 0..∗}
1	(Order, Customer)	0	{1, 0..1, 1..∗, 0..∗}
2	(Customer, Item)	0	{1..∗, 0..∗}
3	(Item, Customer)	0	{1, 0..1, 1..∗, 0..∗}
4	(Order, Item)	0	{1..∗, 0..∗}

placed. There are well-studied cache replacement strategies, also used by other streaming frameworks [20], that we offer in our framework as well. What follows is a list of supported strategies and which items they remove. Note that the cache replacement strategies do not consider any behavioral differences between objects or object types, but only the elements' position, or update/arrival time.

- **FIFO**: the first item that was inserted
- **RR**: the item is chosen randomly
- **LFU**: the item used the least amount of times
- **LFU-DA**: the item used the least amount of times and simultaneously residing in the buffer the longest
- **LRU**: the item that was used least recently

5.3 Priority Policies

Object-centric priority policies consider process characteristics of objects or object types, allowing users to influence replacement likelihood based on these characteristics. Users can prioritize replacing the one with the highest or lowest value for a given characteristic. If activated, rankings are normalized relative to other objects/types and weighted with the cache replacement strategy rankings. The following characteristics are supported in our implementation:

- **Stride per object**: Average time between events involving a given object
- **Stride per object type**: Average time between an object type's events
- **Lifespan per object**: Total time between an object's first and last event
- **Lifespan per object type**: Average lifespan of objects per type
- **Objects per event per type**: Average number of objects in events of an object type
- **Objects per type**: Count of unique objects per type
- **Events per type**: Total observed events with E2O relations per object type
- **Custom**: Static, user-specified ranking of object types

5.4 Buffer Synchronization

After updating model buffers with incoming stream items, the streaming framework performs optional coupled removal to synchronize all model buffers by reducing buffer items to those belonging to objects or object types that appear in all buffers. This cleanup step ensures optimal memory utilization and maintains consistency across all streaming data structures.

6 Evaluation

This evaluation investigates how well and fast our proposed streaming framework can discover object-centric models, compared to the one that would have been mined without memory limitations in an offline setting. We used our publicly accessible Python implementation of our framework and two publicly accessible event logs, namely the container logistics log (CL) and the Age of Empires log (AoE). The log stats are as follows for CL (AoE): 14,013 (361,935) objects, 7 (25) object types, 35,413 (34,860) events, 14 (829) activities, 15,926 (33,407) O2O updates, and 13,052 (0) attribute updates. The framework comes with a *CacheMonitor* and *RuntimeMonitor* class that records, for example, performance, buffer utilization, and object type distribution within the buffers. Using these classes, we performed a variety of evaluations and visualized them, which can be seen on GitHub. Here, we focus on the structural similarity analysis and a runtime analysis.

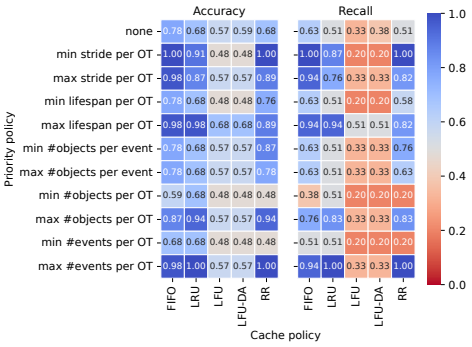

Fig. 3: Cache replacement and priority policies' impact on structural similarity of online and offline TOTeM on CL.

For the structural analysis, we compared online models for different buffer sizes with the process model mined offline on the full log as the ground truth, thereby investigating the impact of the buffer size on the structural similarity. We computed recall, accuracy, and precision for the online model. This showed that the overall structural similarity improves with bigger buffer sizes until a plateau is reached. The precision was constantly 1. By increasing buffer size, good accuracy and recall values can be reached, as shown in Figure 4. We evaluated the impact of different combinations of cache policies and priority policies

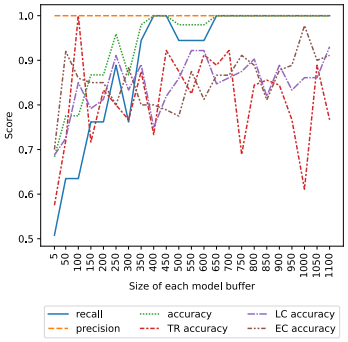

Fig. 4: TOTeM model similarity on CL dataset using FIFO.

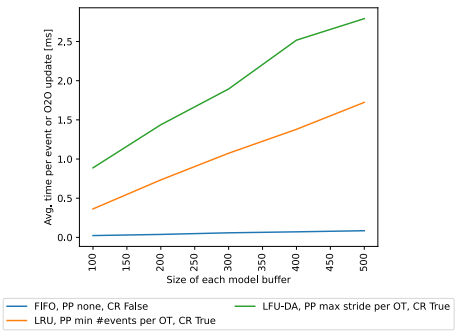

Fig. 5: Runtime on CL log, priority policy (PP), buffer synchronization (CR).

on the structural similarity. Figure 3 shows the severe impact for a buffer size of 125. Further investigations are needed to identify which settings are most suited for streams with certain characteristics. The runtime evaluation shows that the framework's implementation is able to handle incoming stream items reasonably fast (in about 2.5ms) even with object-centric priority policies and buffer synchronization enabled, as shown for the TOTeM runtime in Figure 5.

7 Discussion and Conclusion

The proposed streaming framework successfully discovers object-centric process models from event streams using fixed-size buffers and provides flexibility through multiple cache strategies and object-centric priority policies.

However, the framework and its evaluation also come with limitations. The used Inductive Miner variant on directly-follows graphs does not give the same guarantees as the original Inductive Miner, limiting the theoretical guarantees of this proposed framework. The evaluation is currently limited to a structural comparison with offline models. With future advances in object-centric research, a semantic/language-based comparison would be more effective in evaluating the similarity of models. Additionally, other use cases for streaming-based object-centric discovery should also be evaluated, following streaming-based evaluation goals [7]. For example, streaming-based discovery has been shown to be well-suited for processes with concept drift, since they tend to prioritize recent information due to the limited buffer size. This motivates publicly accessible object-centric event logs with concept drift. Our evaluation reveals that strategy selection significantly impacts model quality, with performance varying considerably based on underlying event data characteristics. Investigating the relation between object-centric process characteristics, like a high degree of interaction or an uneven object distribution between object types, and the outcome for different replacement strategies could be a valuable guide for practitioners in the future. These limitations should be addressed in future research.

In conclusion, this work establishes the first streaming framework for object-centric process discovery, bridging the gap between traditional streaming process mining and object-centric process mining. The substantial impact of strategy selection on model quality emphasizes the need for guided configuration approaches and theoretical guarantees in future extensions.

Acknowledgment upon was funded by the German Federal Ministry of Research, Technology and Space under grant number 01IS25011. The responsibility for the content of this publication lies with the authors.

References

1. Adriano Augusto, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, Andrea Marrella, Massimo Mecella, and Allar Soo. Automated discovery of process models from event logs: Review and benchmark. *IEEE Trans. Knowl. Data Eng.*, 31(4):686–705, 2019.

2. Ahmed Awad, Matthias Weidlich, and Sherif Sakr. Process Mining over Unordered Event Streams. In *ICPM*, pages 81–88. IEEE, 2020.
3. Alessandro Berti and Wil M. P. van der Aalst. OC-PM: analyzing object-centric event logs and process models. *Int. J. Softw. Tools Technol. Transf.*, 25(1):1–17, 2023.
4. Alessandro Berti and et. al. OCEL (object-centric event log) 2.0 specification. *CoRR*, abs/2403.01975, 2024.
5. Alessandro Berti, Sebastiaan J. van Zelst, and Daniel Schuster. Pm4py: A process mining library for python. *Softw. Impacts*, 17:100556, 2023.
6. Andrea Burattin. Streaming Process Mining. In *Process Mining Handbook*, volume 448 of *LNBIP*, pages 349–372. Springer, 2022.
7. Paolo Ceravolo, Gabriel Marques Tavares, Sylvio Barbon Junior, and Ernesto Damiani. Evaluation goals for online process mining: A concept drift perspective. *IEEE Trans. Serv. Comput.*, 15(4):2473–2489, 2022.
8. Stefan Esser and Dirk Fahland. Multi-dimensional event data in graph databases. *CoRR*, abs/2005.14552, 2020.
9. Dirk Fahland. Process mining over multiple behavioral dimensions with event knowledge graphs. In *Process Mining Handbook*, volume 448 of *LNBIP*, pages 274–319. Springer, 2022.
10. Dirk Fahland and et. al. Towards a simple and extensible standard for object-centric event data (oced)–core model, design space, and lessons learned. *arXiv preprint arXiv:2410.14495*, 2024.
11. Alessandro Gianola, Marco Montali, and Sarah Winkler. Object-centric processes with structured data and exact synchronization - formal modelling and conformance checking. In *CAiSE*, volume 15702 of *LNCS*, pages 185–202, 2025.
12. Marwan Hassani, Sergio Siccha, Florian Richter, and Thomas Seidl. Efficient Process Discovery From Event Streams Using Sequential Pattern Mining. In *SSCI*, pages 1366–1373. IEEE, 2015.
13. Volodymyr Leno, Abel Armas-Cervantes, Marlon Dumas, Marcello La Rosa, and Fabrizio Maria Maggi. Discovering process maps from event streams. In *ICSSP*, pages 86–95. ACM, 2018.
14. Lukas Liss, Jan Niklas Adams, and Wil M. P. van der Aalst. Totem: Temporal object type model for object-centric process mining. In *BPM Forum*, volume 526 of *LNBIP*, pages 107–123. Springer, 2024.
15. Gurmeet Singh Manku and Rajeev Motwani. Approximate Frequency Counts over Data Streams. In *VLDB*, pages 346–357. Morgan Kaufmann, 2002.
16. Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.
17. Wil M. P. van der Aalst and Alessandro Berti. Discovering object-centric petri nets. *Fundam. Informaticae*, 175(1-4):1–40, 2020.
18. Jan Niklas van Detten, Pol Schumacher, and Sander J. J. Leemans. Discovering compact, live and identifier-sound object-centric process models. In *ICPM*, pages 113–120. IEEE, 2024.
19. Sebastiaan J. van Zelst, Alfredo Bolt, Marwan Hassani, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Online conformance checking: relating event streams to process models using prefix-alignments. *Int. J. Data Sci. Anal.*, 8(3):269–284, 2019.
20. Sebastiaan J. van Zelst, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Event stream-based process discovery using abstract representations. *Knowl. Inf. Syst.*, 54(2):407–435, 2018.