ABSTRACT Object-centric process mining is a new process mining paradigm with more realistic assumptions about underlying data by considering several case notions, e.g., an order handling process can be analyzed based on order, item, package, and route case notions. Including many case notions can result in a very complex model. To cope with such complexity, this paper introduces a new approach to cluster similar case notions based on Markov Directly-Follow Multigraph, which is an extended version of the well-known Directly-Follow Graph supported by many industrial and academic process mining tools. This graph is used to calculate a similarity matrix for discovering clusters of similar case notions based on a threshold. A threshold tuning algorithm is also defined to identify sets of different clusters that can be discovered based on different levels of similarity. Thus, the cluster discovery will not rely merely on analysts’ assumptions. The approach is implemented and released as a part of a python library, called processmining, and it is evaluated through a Purchase-to-Pay (P2P) object-centric event log file. The discovered clusters are evaluated by discovering Directly Follow-Multigraph by flattening the log based on the clusters. The similarity between identified clusters is also evaluated by calculating the similarity between the behavior of the process models discovered for each case notion using inductive miner based on footprints conformance checking.

INDEX TERMS Object-centric process mining, object-centric directly-follow multigraph, markov clustering.

I. INTRODUCTION Process mining is a research area supporting data-based process analysis. The main input for this analysis is data, mostly in the form of log files, recording events that occurred during process enactment. The output is a model describing some analysis aspects that can help analysts to improve the business process. These models can be in the form of a process model, diagnostic information, etc.

Different event log formats are defined over time to facilitate applying process mining techniques in practice. eXtensible Event Stream (XES) [1] is an IEEE Standard defined in 2014 to standardize the input log formats, supported by many process mining software. This standard assumes the existence of only one case notion in practice. An example of a case notion is an order in an order-handling process.

Recent studies challenge applying process mining based on only one case notion [5], [21], [23]. For example, a simple order handling process can have many potential case notions like order, item, package, and route, which enable analyzing the business process from different perspectives. Indeed, it is more realistic to consider an event to be related to several case notions, as several business entities might get affected by performing an activity in a business process.

Analysts usually flat these logs to apply process mining techniques, which are developed under the assumption of dealing with one case notion. Such flattening raises problems known as convergence and divergence [21].
An example of a convergence problem is repeating an event related to the occurrence of a batch job that handles many items - when flattening the log based on the item notion. It might enable discovering the batch activity in the discovered process model, but it can cause the problem of counting the wrong occurrence of the batch job activity.

An example of a divergence problem is losing the order between checking the availability of an item and picking it up when flattening the log based on the order notion. It can cause undesirable and incorrect loops because the activities’ order will be lost by removing the item notion which is needed to correlate events related to an item.

Object-Centric Event Log (OCEL) [9] is the standard for relating one event to multiple objects representing different case notions. Object-Centric Process Mining is a new process-mining paradigm that supports several case notions when analyzing log files that can address both convergence and divergence problems. These logs are considered to be closer to information systems’ data in reality [23].

Directly-Follows Multigraph (DFM) [21] is a graph showing the relationship between business activities by incorporating several case notions. Relations in DFM show how the control in the process can move from one activity to another based on a case notion. It can be considered as an equivalent graph like the well-known Directly-Follows Graph (DFG) but incorporates several case notions. This graph is also defined as Object-Centric Directly-Follows multiGraph (OC-DFG) [7].

FIGURE 1 shows an example of a DFM discovered from a toy example log file containing 39 events related to four case notions, i.e., item, order, package, and route, where their corresponding flows are colored in red, dark-red, green, and dark-green, respectively. The model is discovered using PM4Py [6], which is a python library that supports process mining.

As it can be seen in FIGURE 1, a DFM can easily become complex due to the existence of several case notions, for each of which the process might have different underlying behavior. This result is discovering spaghetti models, which are hard to analyse [20]. On the one hand, we can come up with a spaghetti model that is too complex to use by incorporating all object types when discovering a process model. On the other hand, we can face convergence and divergence problems by flattening logs based on objects with shared events. Flattening the log based on similar object types can make a balance between these two tradeoffs.

Currently, there is a gap in theory and tool support to enable the clustering of similar object types for a given Object-Centric Event Log. As similarity is a relative subject, these clusters can differ based on the expected similarity level. Thus, it is also important to discover possible levels based on which different clusters can be identified. Thus, this paper aims to address this gap by answering these research questions:

- How can a set of similar object-type clusters be discovered from an object-centric event log based on a given similarity level?
- How can a possible range of similarity levels to discover clusters be identified from an object-centric event log?

To answer the first research question, this paper introduces a new approach to cluster similar case notions by defining Markov Directly-Follow Multigraph. This graph is used to calculate a similarity matrix that enables the clustering of the case notions based on a threshold. Markov clustering is selected as it is widely applied in practice for different purposes, e.g., identifying protein-protein interaction networks [14], traffic state clustering [16], document clustering [11], and comparing similarities between different process variants [13]. To answer the second research question, a threshold tuning algorithm is defined to identify sets of different clusters that can be discovered based on all possible thresholds.

The approach is implemented as a part of an open-source python library, called processmining, which is available to be installed via the Python Package Index (PyPI). The approach is evaluated through a Purchase-to-Pay (P2P) object-centric
event log file. Some discovered clusters are evaluated by discovering Directly Follow-Multigraph by flattening the log based on the clusters. The similarity between identified clusters is also evaluated by calculating the similarity between the behavior of the process models discovered for each case notion using inductive miner based on footprints conformance checking.

The rest of the paper is organized as follows. Section II gives a short background. Section III formalizes the approach. Section IV elaborates on the implementation. Section V reports the evaluation results. Section VI concludes the paper and introduces future research.

II. BACKGROUND
Object-Centric process mining (OCPM) is a new yet fast-paced growing area thanks to identified gaps experienced in practice. These gaps are discovered by using many commercial and open-source tools, developed under the assumption of having only one case notion in the log file. Most of these tools focus on the generation of Directly-Follows Graphs (DFGs) as a means to visualize the control flow - which is used a lot by practitioners due to their simplicity [22].

Although DFGs can be misleading due to lack of support for concurrency [22], they can be helpful as an intermediate model to discover more advanced models as done by, e.g., Split Miner [3], Heuristics Miner [25] and Fodina [24]. DFGs are also used in variant analysis where different models of a business process representing different variations can be compared to each other [13], [17], [18].

In Object-Centric process mining, Directly Follow Multigraph (DFM) is defined to discover process models from OCEL [21]. Object-centric Petri nets (OC-Petri nets) is another discovery technique that can generate process models from OCEL [23]. From the tools support perspective, PM4Py [6] is a python library that supports discovering DFM and object-centric Petri nets. PM4Py-MDL is a python library that extends the functionality of PM4Py to support performance and conformance analysis through the token-based replay [23]. In addition, a stand-alone object-centric process cube tool is developed to support cube operations, i.e., slice and dice [10].

Recently, a few online tools are also developed to support Object-Centric Process Mining. For example, OC-PM is developed to enable the discovery of both OC-DFG and OC-Petri nets with different annotations, such as the frequency of activities and paths among them [7]. This tool also supports discovering process models by filtering event logs. It also provides functionalities to apply some conformance checking, such as the number of related objects and the objects’ lifecycle duration. We also can see a rising interest in supporting OCPM by commercial tools, e.g., MEHRWERK Process Mining (MPM) [15], which indicates how relevant this problem is in practice.

The tool support for OCPM is expanding not only in analysis but also in the pre-analysis phase, where data shall be Extracted, Transformed, and Loaded for conducting process mining. For example, a tool is developed to extract OCEL from ERP systems, i.e., SAP ERP System [5], which enables extracting OCELS from well-known processes in SAP ERP, e.g., Purchase-to-Pay (P2P) and Order-to-Cash (O2C). Indeed, sample P2P and O2C logs in OCEL format are available through http://ocel-standard.org [9], which empowers researchers to develop further artifacts and evaluate them based on these data.

The rise of big data introduces some challenges in applying process mining in practice, like scalability or discovering process models from logs that do not fit the memory of a computer [12], which is also the case for OCPM. Graph databases provide good capabilities to overcome these challenges [4], [12]. Several studies show how databases like Neo4j and MongoDB can be used to store and analyze both traditional and object-centric log files [4], [8], [12].

The application of OCPM techniques also requires adaptations in four competing quality dimensions of process mining, i.e., fitness, precision, simplicity, and generalization [2]. Adams J.N. and van der Aalst W.M.P. define how the precision and fitness of object-centric Petri nets can be calculated by replaying the model with respect to an OCEL [2]. Calculating these measures based on other techniques like alignment is still open for research, which is also the case for simplicity and generalization measures.

In summary, OCPM is a new paradigm that needs further research to be applied in practice. The current algorithms that enable discovering object-centric process models generate very complex process models. One way to deal with this complexity would be the separation of case notions into clusters based on their similarities. Such separation can also help future process discovery algorithms to consider object-type similarities when discovering process models from OCEL. The next section explains how such separation can be performed using Directly Follow Multigraphs.

III. APPROACH
This section defines the approach to identifying different clusters of similar case notions. To explain the definitions, a part of FIGURE 1 will be used as a running example, shown in FIGURE 2.

For simplicity, acronyms are used instead of the activities’ names, which are shown in parenthesis in the figure. For...
example, we will use \(po\) instead of \(place\_order\), \(o\) instead of \(order\), and so on.

**Definition 1 (Directly-Follows Multigraph (DFM))**: A Directly-Follows Multigraph (DFM) is a tuple \(G = (OT, T, R, f)\), where:
- \(OT\) is the set of object types,
- \(T\) represents the set of tasks
- \(R = (T \times OT \times T)\) is the set of relations connecting two tasks based on an object type. We call the first task the source and the second one the target, representing the task from/to which the relation starts/ends, respectively.
- \(f \in R \rightarrow \mathbb{N}\) is a function that assigns a natural number, representing the frequency, to each relation.

Considering \(\Theta \subseteq OT\) as a subset of object types, two operators on the graph’s tasks can be defined as follow:
- \(\bullet\) represents the operator that retrieves the set of tasks from which there are relations to task \(t\) for an object types within \(\Theta\), i.e.:
  \[\bullet\Theta = \{t' \in T | \exists t \in \Theta (t', \theta, t) \in R\}\].
- \(\circ\) represents the operator that retrieves the set of tasks to which there are relations from task \(t\) for an object types within \(\Theta\), i.e.:
  \[\circ\Theta = \{t' \in T | \exists \theta \in \Theta (t, \theta, t') \in R\}\].

**Example 1**: We can define the Directly-Follows Multigraph (DFM) for our running example in FIGURE 2 as \(G = (OT, T, R, f)\), where:
- \(OT = \{o, i, p\}\) is the set of object types.
- \(T = \{po, ca, pi, sp, sr\}\) is the set of tasks.
- \(R = \{(po, o, ca), (po, i, ca), (ca, o, ca), (ca, i, ca), (ca, o, pi), (ca, i, pi), (pi, o, ca), (sp, p, sr)\}\) is the set of relations. \(po\) is the source and \(ca\) is the target of \((po, ca)\) relation.
- \(f((po, o, ca)) = 3, f((po, i, ca)) = 6, f((ca, o, ca)) = 3, f((ca, i, ca)) = 3, f((ca, o, pi)) = 6, f((ca, i, pi)) = 6, f((pi, o, ca)) = 3, f((sp, p, sr)) = 2\) assigns frequencies to relations.

Examples of the operations based on the running example are given below:
- \(\bullet\Theta = \{po\}\) retrieves a set of tasks from which there are outgoing object type \(item\) (i). Note that we can have different result if we change the object type, i.e., \(\circ\Theta = \{po, ca, pi\}\) which retrieves the set of tasks from which there is a relation to check availability (ca) for object type order (o).
- \(\bullet ca = \{ca, pi\}\) and \(\circ ca = \{ca, pi\}\) retrieves a set of tasks to which there is a relation from check availability (ca) using item (i) object type and from place order (po) using order (o) object type, respectively.

To find similarities between the control flow for different case notions, we convert the Directly Follow Multigraph to Markov Directly Follow Multigraph, defined below. Also, we define a similarity measure that calculates how similar the control flow of the process model is with respect to two given object types.

**Definition 2 [Markov Directly-Follows Multigraph (Markov DFM)]:** Let \(G = (OT, T, R, f)\) be a DFM. \(M = (G, p, sim)\) is a Markov DFM, where \(p \in R \rightarrow [0-1] \subseteq \mathbb{Q}\) is a function that assigns a positive rational number between zero and one, representing the probability, to a relation. \(sim \in OT \times OT \rightarrow [0-1] \subseteq \mathbb{Q}\) is a function that assigns a positive rational number between zero and one, representing the similarity, to an object types pair, where:

\[
p(t, \theta, t') \leftarrow \frac{f((t, \theta, t'))}{\sum_{\forall t' \in \Theta} f((t, \theta, t'))}
\]

\[
sim(\theta_1, \theta_2) \leftarrow \frac{\sum_{\forall t, t' \in T} (p(t, \theta_1, t') \cdot p(t, \theta_2, t'))}{\sum_{\forall t, t_1, t_2 \in T} (p(t, \theta_1, t_1)^2 + p(t, \theta_2, t_2)^2)}
\]

Note that the type of inputs for each function is defined by the domain of the function, i.e., \((t, \theta, t') \in R\) and \((\theta_1, \theta_2) \in OT \times OT\).

We can define the Markov Directly-Follows Multigraph (DFM) for our running example as \(M = (G, OT, T, R, f, p, sim)\). Let’s calculate \(p\) using an example.

**Example 2**: - \(p((ca, o, pi)) = f((ca, o, pi)) / (\sum_{\forall t \in ca} f((ca, o, t))) = 6 / (\sum_{\forall t \in ca} f((ca, o, t))) = 6 / (3 + 6) = 6/9 = 2/3\), which is the probability of occurrence of check availability given place order is occoured for object type order in this model.

We can illustrate our graph based on this definition visually through FIGURE 3, where the frequencies and probabilities of relations are shown by \(p\) and \(f\), respectively. Note that probabilities can be represented by a matrix per object type, where rows and columns indicate the source and target tasks of a relation, as shown in TABLE 1. This table also makes it easier to explain the similarity calculation using \(sim\) function.

**Example 3**: As an example, let us to calculate \(sim(i, o)\), where the probabilities of relations for item and order object types can be represented by \(P_i\) and \(P_o\) matrices as also shown.
TABLE 1. The probability of each relation is represented through a matrix per object type, where rows and columns represent the source and target task, respectively.

(a) Probability of relations for Item

|   | ca | pi | po | sp | sr |
|---|----|----|----|----|----|
| ca | 1/3 | 2/3 | 0  | 0  | 0  |
| pi | 0  | 0  | 0  | 0  | 0  |
| po | 1  | 0  | 0  | 0  | 0  |
| sp | 0  | 0  | 0  | 0  | 0  |
| sr | 0  | 0  | 0  | 0  | 0  |

(b) Probability of relations for Order

|   | ca | pi | po | sp | sr |
|---|----|----|----|----|----|
| ca | 1/3 | 2/3 | 0  | 0  | 0  |
| pi | 1  | 0  | 0  | 0  | 0  |
| po | 0  | 0  | 0  | 0  | 0  |
| sp | 0  | 0  | 0  | 0  | 0  |
| sr | 0  | 0  | 0  | 0  | 0  |

(c) Probability of relations for Package

|   | ca | pi | po | sp | sr |
|---|----|----|----|----|----|
| ca | 0  | 0  | 0  | 0  | 0  |
| pi | 0  | 0  | 0  | 0  | 0  |
| po | 0  | 0  | 0  | 0  | 0  |
| sp | 0  | 0  | 0  | 0  | 0  |
| sr | 0  | 0  | 0  | 0  | 0  |

TABLE 2. Calculated Similarity Matrix that shows the similarity of the process for object type pairs.

|   | o  | i  | p  |
|---|----|----|----|
| o | 1.0| 0.76| 0.0 |
| i | 0.76| 1.0| 0.0 |
| p | 0.0| 0.0| 1.0 |

The similarity function calculates the similarity accordingly. First, it calculates the denominator by summing up every element of $(P_i^2 + P_o^2)/2$, which is equivalent to $\sum_{t_1,t_2 \in T} \left( (p(t_1, i, t_2)^2 + p(t_1, o, t_2)^2)/2 \right)$, which is equal to $37/18$. Second, the similarity will be calculated by $P_i \cdot P_o$ divided by the calculated denominator, which will be equal to $252/333 = 0.76$.

It is straightforward to calculate the similarity of Package with Item and also with Order in our running example. As the numerator will always be zero, the similarity will be zero. The similarity result of the process for each object type pair for this example can be shown as a matrix, represented in TABLE 2. We call this matrix the similarity matrix.

Algorithm 1: discoverClusters

Data: $((OT, T, R, f), p, sim)$, threshold

Result: clusters

begin

clusters $\leftarrow \{\}$;

foreach $\theta_1, \theta_2 \in OT$ do

if $sim(\theta_1, \theta_2) \geq$ threshold then

$X \leftarrow \bigcup_{C \in \text{clusters}} \{C | \{\theta_1\} \subseteq C \lor \{\theta_2\} \subseteq C\}$;

clusters $\leftarrow$ clusters $\cup \{C \cup \{\theta_1, \theta_2\} \}$;

clusters $\leftarrow$ clusters $\setminus X$;

end

return clusters;

end

TABLE 3 shows the result of calling this algorithm for the running example using different thresholds in addition to the filtered similarity matrix.

TABLE 3. Filtered similarity matrix and Identified clusters for the running example by setting different thresholds.

(a) 1 cluster when threshold=0, i.e., $(\{i, o, p\})$

|   | o  | i  | p  |
|---|----|----|----|
| o | 1.0| 0.76| 0.0 |
| i | 0.76| 1.0| 0.8 |
| p | 0.0| 0.0| 1.0 |

(b) 2 clusters when threshold=0.01, i.e., $(\{i, o\}, \{p\})$

|   | o  | i  | p  |
|---|----|----|----|
| o | 1.0| 1.0| 0.0 |
| i | 1.0| 1.0| 1.0 |
| p | 1.0| 1.0| 1.0 |

(c) 3 clusters when threshold=0.77, i.e., $(\{i, o\}, \{p\})$

|   | o  | i  | p  |
|---|----|----|----|
| o | 1.0| 1.0| 0.0 |
| i | 1.0| 1.0| 1.0 |
| p | 1.0| 1.0| 1.0 |

TABLE 3(a) represents the result and the filtered similarity matrix when we call the algorithm by setting the threshold to zero. In this case, we will receive only one cluster that includes all object types. It is because the value for $sim(\theta_1, \theta_2)$ is always greater or equal to zero, so all object types will be added to the returned cluster, so the result for our running example will be $(\{i, o, p\})$.

TABLE 3(b) represents the result and the filtered similarity matrix when we call the algorithm by setting the threshold to one percent. As seen in the filtered matrix, the relation between $p$ and other object types will be filtered as it is lower than the threshold. This result in the separation of these object types from others, so we will receive two clusters, i.e., $(\{i, o\}, \{p\})$. 

TABLE 3(c) represents the result and the filtered similarity matrix when we call the algorithm by setting the threshold to 0.77. As seen in the filtered matrix, the relation between $p$ and other object types will be filtered as it is lower than the threshold. This result in the separation of these object types from others, so we will receive three clusters, i.e., $(\{i, o\}, \{p\})$. 

It is straightforward to calculate the similarity of Package with Item and also with Order in our running example. As the numerator will always be zero, the similarity will be zero. The similarity result of the process for each object type pair for this example can be shown as a matrix, represented in TABLE 2. We call this matrix the similarity matrix.

Algorithm 1 defines how clusters of similar object types can be discovered from a Markov DFM given a threshold - answering the first research question. It defines an empty set for clusters of object types. Then, for each pair of object types, if the similarity between them is greater or equal to the given threshold, it i) retrieves a union of all sets of clusters that contains one of the object types and ii) excludes the identified sets from the clusters and add all object types within these clusters in addition to two compared object types as a new cluster in the clusters set. It finally returns the identified set of clusters.

TABLE 3 shows the result of calling this algorithm for the running example using different thresholds in addition to the filtered similarity matrix.
TABLE 3(c) represents the result and the filtered similarity matrix when we call the algorithm by setting the threshold to 77 percent. This result is \{\{i\}, \{o\}, \{p\}\}. It can be said that if an object type in the filtered matrix has similarities to others, they will be in the same cluster.

Algorithm 2: tuneClusters

**Data:** \((M, \text{threshold}, \text{res})\) such that \(M\) is a

**Result:** \(\text{res}\) such that

```
begin
if \(\text{res} = \emptyset\) then
    \text{res} ← \{(0, \text{discoverClusters}(M, 0))\};
    \text{res} ← \text{res} \cup \{(1, \text{discoverClusters}(M, 1))\};
    \text{return tuneClusters}(M, 0.5, \text{res});
else
    if \((\text{threshold}, _) \in \text{res}\) then
        \text{return \text{res};}
    else
        \(CT ← \text{discoverClusters}(M, \text{threshold});\)
        \text{res} ← \text{res} \cup \{(\text{threshold}, \text{CT})\};
        u ← \min(\{i \mid \forall (i, _) \in \text{res} \ i > \text{threshold}\});
        l ← \max(\{i \mid \forall (i, _) \in \text{res} \ i < \text{threshold}\});
        if \((|\{C \mid V(t, C) \in \text{res} \ t = u\}| ≠ |CT|)\) then
            \(t ← \text{round}(\text{threshold} + u)/2, 2);\)
            \text{res} ← \text{res} \cup \{(t, \text{discoverClusters}(M, t))\};
        if \((|\{C \mid V(t, C) \in \text{res} \ t = l\}| ≠ |CT|)\) then
            \(t ← \text{round}(\text{threshold} + l)/2, 2);\)
            \text{res} ← \text{res} \cup \{(t, \text{discoverClusters}(M, t))\};
        \text{return \text{res};}
```

In practice, it is difficult to change the threshold to find all possible clusters manually, so Algorithm 2 tunes the threshold to identify all possibilities - answering the second research question. This algorithm gets the \(M\) (as a DFM), \text{threshold}, and \text{res} - which is the result set representing the result of the previous tuning attempt. When calling this algorithm, the \text{res} is an empty set as no tuning has happened. The algorithm performs recursively.

If it is the first time the algorithm is called (if \(\text{res} = \emptyset\)), it discovers clusters for thresholds 0 and 1 and adds the result to the \text{res}. Then, it calls itself to tune the cluster discovery based on a threshold in between (i.e., 0.5). The algorithm will perform recursively, so when the tuning is finished, the result will be returned. If it is not the first time that Algorithm 2 is called, it checks if the set of clusters for a given threshold is already discovered (if \((\text{threshold}, _) \in \text{res}\)). If yes, it returns the result; otherwise, it discovers a set of similar clusters for a given threshold \((CT)\) and adds them to the result \((\text{res})\). Then, it retrieves the lower- \((l)\) and upper- \((u)\) bounds of threshold in \text{res}.

Considering all thresholds in \text{res} that have a lower number of clusters than identified clusters \((CT)\), the lower bound is the threshold among them with the highest numbers of clusters. For example, if we have only three identified clusters in \text{res} like TABLE 3, the number of identified clusters for thresholds 0 and 0.01 is less than threshold 0.77. In this case, the lower bound for threshold 0.77 will be 0.01 as it has fewer clusters than threshold 0.77 but more than threshold 0.

Considering all thresholds in \text{res} that have a higher number of clusters than identified clusters \((CT)\), the upper bound is the threshold among them with the lowest numbers of clusters. For example, if we have only three identified sets of clusters in \text{res} like TABLE 3, the number of identified clusters for thresholds 0.01 and 0.77 is more than threshold 0. In this case, the upper bound for threshold 0 will be 0.01 as it has more clusters than threshold 0 but less than threshold 0.77.

If the number of clusters in the current threshold is not equal to the upper bound \((u)\), it recursively discovers a cluster for a threshold. To avoid running this algorithm infinitely, we calculated the value in between by rounding the value by having two digits after the decimal points. It also does the same for the lower bound \((l)\). This algorithm tune the threshold parameter through a half-interval search.

**IV. IMPLEMENTATION**

The approach is implemented and is available as a part of a python library called processing. The source is available in Github, and the library is available in PyPI - which enables users to install and use it easily by running the pip command, if python and PM4Py are installed. The library aims to provide more functionalities to perform process mining using python and other libraries like PM4Py. The codes to repeat the running example and evaluation can be found at Github.

FIGURE 4 shows the result of cluster tuning for DFM in FIGURE 1, where it discovered four sets of clusters. In this

---

1. https://github.com/jalaliamin/processmining
2. pip install processmining
3. https://github.com/jalaliamin/ResearchCode/tree/main/ot-clustering-markov-dfm-ocpm
A. Jalali: Object Type Clustering Using Markov Directly-Follow Multigraph in OCPM

FIGURE 5. The similarity matrices for identified clusters in FIGURE 4.

FIGURE 6. Discovered DFMs based on two identified clusters by a similarity threshold of 0.16. The figure is made intentionally small just to show supporting the separation of similar object types.

V. EVALUATION

This section evaluates the presented approach using the given implementation on a Purchase-to-Pay (P2P) object-centric event log file. For the evaluation, SAP ERP IDES instance - P2P log file is used which is provided by http://ocel-standard.org [9]. This log file records the events in FIGURE 1 will look in general when flattening the log based on similar object types. Such flattening still enables the study of the connection among similar object types, yet focusing on the related ones. The code for reproducing this experiment can be found in the Github.4

4https://github.com/jalaliamin/ResearchCode/blob/main/ot-clustering-markov-dfm-ocpm/running-example.ipynb

VOLUME 10, 2022

126575
A. Jalali: Object Type Clustering Using Markov Directly-Follow Multigraph in OCPM

for the Purchase-to-Pay process, containing 24,854 events and 9 object types.

Purchase-to-Pay is the standard end-to-end procurement process that can be configured in Enterprise Resource Planning (ERP) systems like SAP. This process enables organizations to control and automate their P2P activities, including making orders, inspecting incoming goods, payments, etc. The procurement process is widely used in different sorts of organizations. Thus, the evaluation of our approach based on such a log file makes it easier to be applied and employed in other companies and log files.

These steps are followed to evaluate the approach. First, sets of thresholds are identified by applying this technique, which demonstrates how a possible range of similarity levels can be discovered from an object-centric event log (the second research question). Second, different clusters of similar object types are discovered based on identified similarity levels (the first research question). Then, some of the identified clusters are evaluated by flattening the log based on clustered object types that show the application of our approach based on real data. Third, the log is flattened based on each object type, and a process model is discovered using the inductive miner for each flattened log. For each pair of object types, their corresponding discovered models using inductive miner are compared using the footprint analysis technique. Finally, the result of the footprint analysis is compared with identified clusters. The comparison validates the correctness of identified clusters in a given log file.

The footprint analysis supports measuring the similarity between a log file and a process model, two log files, and two process models. These comparisons are known as log-to-model, log-to-log, and model-to-model, respectively [19]. The analysis is based on calculating the footprint matrix that captures the choice, causality, and parallel relations among tasks in a process model or a log. Thus, it measures the similarity beyond considering merely the directly follow relations. Thus, it can be a good choice to validate if the similarity between object types is discovered correctly [19].

A. CLUSTER DISCOVERY

This section presents the result of the first and second steps in evaluating the proposed approach. FIGURE 7 (a) shows the result of threshold parameter tuning, where four different thresholds have been identified to discover different sets of clusters.

As can be seen from this figure, the first threshold (i.e., zero) will classify all object types into one cluster, and the last one will classify each object type in one cluster. Thus, we only present the two similarity matrices for the two remaining sets of clusters in FIGURE 7(b) and (c).

As can be seen in FIGURE 7 (a), setting the threshold to 0.01 will result in 7 clusters. The similarity matrix in FIGURE 7(b) shows that all object types except EBELN, EBELN_EBELP, and MATNR are classified into their own clusters, meaning that they do not share any similar behavior. This finding can be validated by flattening the log based on these object types and discovering one process model, shown in FIGURE 7 (a). The process is made intentionally...
small to show that there are unconnected tasks in addition to some disconnected control flow for different object types. The result confirms that these objects do not share similar behavior. Indeed, except for BANFN and BELNR object types, we do not see any occurrence of two consequent events for other object types. The control flow for BANFN and BELNR object types also do not share any task, so they are distinct.
Increasing the threshold to 0.31 will discover a cluster with two object types, i.e., EBELN, EBELN_EBELP. The process model, which is discovered by flattening the log based on these two object types, shows very similar behavior among them (see FIGURE 8(b)).

**B. FOOTPRINT ANALYSIS FOR FLATTENED LOGS**

This section presents the result of the remaining steps in evaluating the proposed approach. FIGURE 9 shows the result of the conformance checking, where rows and columns represent object types, and cells represent the conformance of discovered process models using inductive miner by flattening the log - based on each object type.

As can be seen, the highest difference in footprints belongs to MATNR, which has been identified as one cluster when setting the threshold to 0.31. Taking this object type apart, the footprint difference for EBELN and EBELN_EBELP has the highest difference with other object types having minimum difference from each other. This result aligns with the identification of the cluster that contains these two object types when setting the threshold between 0.31 and 0.94. The code for this experiment is available in Github.5

**VI. CONCLUSION**

This paper introduced a new approach to cluster similar case notions by defining Markov Directly-Follow Multigraph. The graph is used to define an algorithm for discovering clusters of similar case notions based on a threshold. The paper also defined a threshold-tuning algorithm to identify sets of different clusters that can be discovered based on different levels of similarity. Thus, cluster discovery does not merely rely on analysts’ assumptions. The approach is implemented and released as a part of a python library, called processmining, and it is evaluated through a Purchase-to-Pay (P2P) object-centric event log file. Some discovered clusters are evaluated by discovering Directly Follow-Multigraph by flattening the log based on the clusters. The similarity between identified clusters is also evaluated by calculating the similarity between the behavior of the process models discovered for each case notion using inductive miner based on footprints conformance checking.

This approach can be used to define an object-centric process discovery algorithm that takes the similarity of object types into account when discovering process models from object-centric event logs, which will be a future direction of this work.

This approach identifies clusters by creating a stochastic model incorporating the probability of the occurrence of the next activity. Thus, it does not incorporate control-flow patterns such as parallel or exclusive splits and joins when clustering object types. As future research, it would be interesting to investigate how clustering can be performed based on other process modeling languages that support these patterns rather than Directly-Follow Multigraph. The evaluation is applied on a Purchase-to-Pay (P2P) object-centric event log file, which is a workflow-based business process. It would also be interesting to apply this approach to log files obtained from knowledge-intensive processes.

**REFERENCES**

[1] G. Acampora, A. Vitiello, B. Di Stefano, W. Van Der Aalst, C. Günther, and E. Verbeek, “IEEE 1849™: The XES standard,” IEEE Comput. Intell. Mag., vol. 12, no. 2, pp. 4–8, May 2017.

5https://github.com/jalaliamin/ResearchCode/blob/main/ot-clustering-markov-dfm-ocpm/p2p.ipynb
[2] J. N. Adams and W. M. P. Van Der Aalst, “Precision and fitness in object-centric process mining,” in Proc. 3rd Int. Conf. Process Mining (ICPM), Oct. 2021, pp. 128–135.

[3] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, and A. Polychyanyy, “Split miner: Automated discovery of accurate and simple business process models from event logs,” Knowl. Inf. Syst., vol. 59, no. 2, pp. 251–284, 2019.

[4] A. Berti, A. F. Ghaifarokhi, G. Park, and W. M. P. Van Der Aalst, “A scalable database for the storage of object-centric event logs,” in Proc. ICPM Doctoral Consortium Demo Track, 2021, pp. 19–20.

[5] A. Berti, G. Park, M. Rafei, and W. M. P. Van Der Aalst, “An event data extraction approach from SAP ERP for process mining,” in Proc. Int. Conf. Process Mining. Cham, Switzerland: Springer, 2021, pp. 255–267.

[6] A. Berti, S. J. van Zelst, and W. M. P. Van Der Aalst, “Process mining for Python (PM4Py): Bridging the gap between process- and data science,” in Proc. ICPM Demo Track, 2019, pp. 1–4.

[7] A. Berti and W. M. P. Van Der Aalst, “OC-PM: Analyzing object-centric event logs and process models,” Int. J. Softw. Tools Technol. Transf., early access, pp. 1–17, Sep. 2022.

[8] S. Esser and D. Fahland, “Multi-dimensional event data in graph databases,” J. Data Semantics, vol. 10, nos. 1–2, pp. 109–141, Jun. 2021.

[9] A. F. Ghaifarokhi, G. Park, A. Berti, and W. M. P. Van Der Aalst, “OCEL: A standard for object-centric event logs,” in Proc. Eur. Conf. Adv. Databases Inf. Syst., Cham, Switzerland: Springer, 2021, pp. 169–175.

[10] A. F. Ghaifarokhi and W. M. P. Van Der Aalst, “A Python tool for object-centric process mining comparison,” in Proc. ICPM Doctoral Consortium Demo Track, 2021, pp. 31–32.

[11] R. Huang, W. Xu, Y. Qin, and Y. Chen, “Hierarchical Dirichlet multinomial allocation model for multi-source document clustering,” IEEE Access, vol. 8, pp. 109917–109927, 2020.

[12] A. Jalali, “Graph-based process mining,” in Proc. Int. Conf. Process Mining (Lecture Notes in Business Information Processing). Cham, Switzerland: Springer, 2020, pp. 273–285.

[13] A. Jalali, P. Johannesson, E. Perjons, Y. Askfors, A. R. Kalladj, T. Shemeikka, and A. Vég, “Dfgcompare: A library to support process variant analysis through Markov models,” BMC Med. Informat. Decis. Making, vol. 21, no. 1, pp. 1–13, Dec. 2021.

[14] X. Lei, F. Wang, F.-X. Wu, A. Zhang, and W. Pedrycz, “Protein complex identification through Markov clustering with firefly algorithm on dynamic protein–protein interaction networks,” Inf. Sci., vol. 329, pp. 303–316, Feb. 2016.

[15] J. Meyer, J. Reimold, and C. Wehmshchulte, “Associative intelligence for object-centric process mining with MPM,” in Proc. ICPM Doctoral Consortium Demo Track, 2021, pp. 39–40.

[16] B. Priambodo, A. Ahmad, and R. A. Kadir, “Predicting traffic flow propagation based on congestion at neighbouring roads using hidden Markov model,” IEEE Access, vol. 9, pp. 85933–85946, 2021.