Super Variants

Jan Niklas Adams[1][0000−0001−8954−4925], Emilie Hastrup-Kiil[1], Gyunam Park[1][0000−0001−9394−6513], and Wil M.P. van der Aalst[1,2][0000−0002−0955−6940]

[1] Process and Data Science, RWTH Aachen University
{niklas.adams,gnpark,wvdaalst}@pads.rwth-aachen.de
[2] Fraunhofer Institute for Applied Information Technology (FIT)

Abstract. Process mining offers methods to analyze the actual control-flow behavior of a process. The two main available methods are process discovery and variant analysis. While process discovery aggregates all variants into one model, the models often suffer from high complexity and lack detailed granularity. Conversely, variants are simple and offer fine granularity but their sheer number makes them difficult to comprehend. To bridge this gap, we introduce the concept of super variants. These represent a middle ground between the complexity of discovered process models and the simplicity of variants, offering an aggregation of closely related variants. We propose a super-variant mining framework based on object-centric variants, evaluate its scalability, and demonstrate its utility through a practical use case. This new approach promises to enhance control-flow analysis by striking a new balance between complexity and aggregation.

Keywords: Process Mining · Variant Analysis · Process Variants · Object-Centric Process Mining

1 Introduction

Process mining allows process owners to analyze event logs extracted from the information systems supporting their business processes [18]. Control-flow analysis is a main task in process mining aiming to uncover the precedence constraints between different activities of a process. Control-flow analysis primarily focuses on two key tasks: process discovery and variant analysis [2].

Variants are sequences of activities observed in process executions and represented as a frequency-sorted list of activity sequences [12]. The resulting variant list is a simple, disaggregated representation of the control flow. Process discovery aims to provide a comprehensive overview that aggregates all (frequent) variants into one process model [9]. The resulting model is a complex, aggregated representation of the control flow. While the aggregation of process models is desirable, they are often overly complex or underfitting [1] and while the simplicity of variants is desirable, their number exceeds the capacity of human comprehension [35]. Considering alternative combinations of simplicity and aggregation, the following problem is apparent: Simple, aggregated models are utopic, and

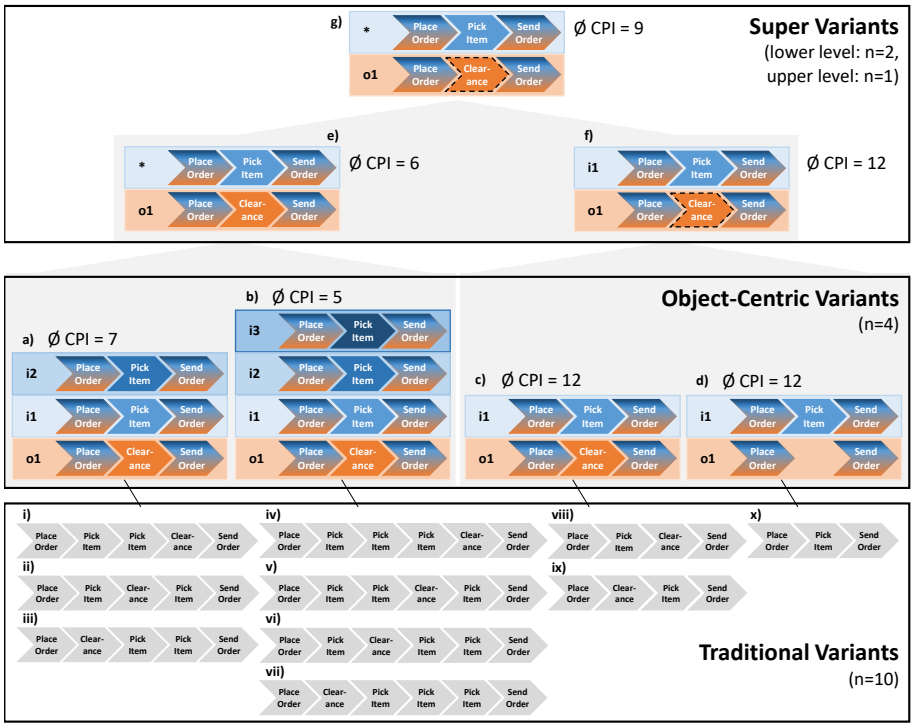


Fig. 1: Super variants to hierarchically aggregate variants into a comprehensive notation. The example depicts an order-to-cash process with one order (o1) and items (i1, i2, i3). We show the traditional variants labeled from i) through x). Object-centric variants a)-d) aggregate traditional variants and are the starting point for super variants. The first variant (a) contains two items while the second variant (b) contains three. These are merged into one super variant (e) with an arbitrary number of items, indicated with the *-symbol. This super variant is further merged with another super variant (f) with an optional clearance activity, leading to a final super variant (g) aggregation. We depict the cost per item (CPI) next to each super variant.

complex, disaggregated representations are impractical. A balance between complexity and simplicity needs to be achieved, increasing the aggregation of variants while preserving part of their simplicity.

A middle ground between aggregation and complexity is necessary for the visualization and bottom-up construction of behavioral clusters in business processes. Next to a more comprehensible way of understanding the overall control-flow composition of the process, such a representation also allows businesses to drill down performance issues to conduct root-cause analysis for process redesign and consolidation. Currently, process owners can find behavioral clusters using trace clustering [45]. While some trace clustering techniques do refine clusters in a top-down approach [33], they do not offer a bottom-up approach that connects clusters to individual variants. A bottom-up approach allows process owners to

decompose and visualize clusters in a transparent analysis. On the other hand, techniques that incorporate variants and visualization, i.e., variant analysis [34], do not aggregate individual variants into behavioral clusters.

Object-centric variants have recently been introduced as a generalization of traditional variants [8]. These are composed of multiple traditional variants, where each traditional variant is an instantiation of a certain subprocess, i.e., an object. This composition of traditional variants allows for shared activities, such that the resulting object-centric variant forms a graph, i.e., a partial order of activities. Therefore, one object-centric variant aggregates several traditional variants as it is able to encode concurrency between objects. An example of this is depicted in the lower two boxes of Figure 1. An object-centric variant consists of lanes that each describe the traditional variant of an object of a specific type indicated by the color. Activities with multiple colors are shared between objects. Non-shared activities do not underlie direct ordering constraints, i.e., they are concurrent. All realizations of the concurrency between objects are the traditional variants an object-centric variant aggregates. As an object-centric variant already functions as an aggregation of traditional variants, this capacity leads to object-centric variants as a natural starting point for the contributions of this paper.

In this paper, we introduce *super variants*. Super variants are aggregations of multiple traditional variants and, thus, situated in between variants and process models. An event log can be represented as a hierarchical representation of super variants as depicted in Figure 1. We depict the traditional variants at the lowest level. These can be aggregated using object-centric variants. The next layer of the hierarchy shows how two object-centric variants are aggregated into one super variant. The super variant of Figure 1 e) contains an arbitrary number of item objects inferred from the varying number of items in a) and b). The super variant in f) depicts the optionality of the clearance activity for the order, inferred from the clearance activity missing in one of the two input variants. The final super variant in g) depicts an arbitrary number of items and an optional clearance activity. The number of variants reduces from ten traditional variants to four object-centric variants to two super variants and one final super variant. From a business perspective, this super variant hierarchy allows us to drill down performance issues. The example also depicts the average cost per item next to each super variant. Going to the second highest hierarchy level, we observe a significant difference between average item costs. Since there are two differences, item multiplicity and clearance optionality, these could both be the source of that difference. Investigating the individual object-centric variants, we can see that the optionality is not the contributing factor but the number of items reduces individual unit cost. Based on this insight, business actions could be taken to increase the number of items per order. Such an analysis would not be possible using top-down trace clustering since clusters are not traced to individual variants.

The main contribution of this paper is introducing super variants to process mining. This contribution boils down to the following individual contributions:

C1 We introduce the concept of super variants.
C2 We propose a super variant mining framework.
C3 We present a super variant visualization.
C4 We evaluate our implementation of the super-variant mining framework.
C5 We provide a super variant use case as a proof of concept.

The remainder of this paper is structured as follows. We introduce preliminary concepts in Section 2 and define super variants in Section 3 along with a super variant mining framework in Section 4. Section 5 provides details on our implementation of the framework. We depict a proof of concept in Section 6 and evaluate the scalability in Section 7. We compare our contributions to the related work in Section 8 and conclude this paper in Section 9.

2 Preliminaries

We introduce some concepts used throughout the paper. The superset of a set X is denoted with $\mathcal{P}(X)$ and contains all subsets. $\mathcal{P}^{i+}(X)$ is the power set of all subsets of size $i \in \mathbb{N}$ or larger. The universe of events is denoted with $\mathcal{E}$, the universe of activities with $\mathcal{A}$, and the universe of timestamps with $\mathcal{T}$. Events are related to objects, which are denoted with $\mathcal{O}$. Each object is of a certain type of the universe of object types $\mathcal{OT}$. The type mapping is given by $\pi_{type} : \mathcal{O} \to \mathcal{OT}$.

The starting point for our concepts is an Object-Centric Event Log (OCEL). We define an OCEL as follows:

Definition 1 (Event Log). *An object-centric event log is a tuple* $L = (E, O, OT, \pi_{obj}, \pi_{act}, \pi_{time})$ *with*

- *events* $E \subseteq \mathcal{E}$*, objects* $O \subseteq \mathcal{O}$ *of types* $OT = \{\pi_{type}(o) \mid o \in O\}$*,*
- *a mapping of events to object* $\pi_{obj} : E \to \mathcal{P}(O)$*,*
- *an activity mapping for events* $\pi_{act} : E \to \mathcal{A}$*, and*
- *timestamps for events* $\pi_{time} : E \to \mathcal{T}$*.*

$\prec_L$ *is the set of precedence constraints in the event log such that* $\prec_L = \{(e, e') \in E \times E \mid e \neq e' \wedge \exists_{o \in O}\ o \in \pi_{obj}(e) \wedge o \in \pi_{obj}(e') \wedge \pi_{time}(e) \leq \pi_{time}(e') \wedge \neg\exists_{e'' \in E}\ o \in \pi_{obj}(e'') \wedge \pi_{time}(e) \leq \pi_{time}(e'') \leq \pi_{time}(e')\}$.

An OCEL consists of events that have an activity and a timestamp. Furthermore, each event is associated with a set of objects of potentially different types. Please note that an OCEL where each event is only associated with one object and all objects are of the same type corresponds to a traditional event log.

The subject of traditional process analysis are typically end-to-end executions of the process called cases. This concept was generalized to the domain of object-centric process mining with the introduction of process executions [8].

Definition 2 (Process Executions). *Let* $L = (E, O, OT, \pi_{obj}, \pi_{act}, \pi_{time})$ *be an OCEL. A process execution is a set of events and precedence constraints* $p = (E', K)$ *with* $E' \subseteq E$ *and* $K \subseteq (E' \times E') \cap \prec_L$*.*

A process execution is a directed graph of events. The edges describe the precedence constraints between events given by objects. Therefore, it is a generalization of a traditional case, which is a sequence of events. A sequence is a special case of a graph.

Process executions need to be extracted from the event log. In general, an extraction technique builds a set of process executions from extracted events and their corresponding precedence constraints.

Definition 3 (Process Execution Extraction). *Let* $L = (E, O, OT, \pi_{obj}, \pi_{act}, \pi_{time})$ *be an OCEL. Given any extraction technique* $ext(L) \subseteq \mathcal{P}(E)$ *we construct process executions from the extracted sets of events* $\pi_{ext}(L, ext) = \{(E', K) \in \mathcal{P}(E) \times \mathcal{P}(E \times E) \mid E' \in ext(L) \wedge K = (E' \times E') \cap \prec_L\}$.

There are different strategies for the extraction of process executions. Connected-component extraction and leading-type extraction are defined in [8]. Connected-components extraction is free of convergence, divergence, and deficiency problems [20], as proven in [6]. Other techniques have to be employed in cases of a fully connected event log. However, our concepts work independently of the chosen extraction technique.

Each process execution describes a specific control-flow behavior. The class of control-flow behavior is called the *variant*. As a process execution is a graph, an object-centric variant is a graph of activities associated with objects of different types.

Definition 4 (Object-Centric Variant). *An object-centric variant is a directed graph* $v = (A, P, \pi_{nact}, \pi_{nobj})$ *of nodes* A*, edges* $P \subseteq A \times A$*, node activities* $\pi_{nact} : A \to \mathcal{A}$*, and node objects* $\pi_{nobj} : A \to \mathcal{P}(\mathcal{O})$*.* $\pi_{vo}(v) = \{\pi_{nobj}(a) \mid a \in A\}$ *are the variant objects.*

Whether two process executions exhibit the same object-centric variant is a graph isomorphism problem. An approach to calculating object-centric variants is described in [8]. We assume the variants to be given in the beginning, therefore, we do not describe the algorithm to determine the object-centric variants of an event log here. An example of object-centric variants is depicted in Figure 1 (a)-(d). The visualization is explained in subsection 5.2.

3 Super Variants

Super variants aim to fill the gap in control-flow representation between variants and process models to enable hierarchical variant analysis. We introduce the concepts expressible by variants and object-centric process models to find a middle ground for super variants.

Traditional variants can express a sequential ordering of activities. Object-centric variants are a generalization of traditional variants [8]. An object-centric variant consists of a traditional variant per object where some activities can be shared between objects, i.e., it resembles a graph or partial order of activities.

As such, it can express sequentiality within an object and concurrency between objects.

Discoverable process models can cover a wider range of behavior. We will focus on Object-Centric Petri Nets (OCPNs) as it is one of the most established modeling techniques for which a discovery algorithm exists [3]. OCPNs are composed of Petri nets for each object type, therefore, they can express choice and concurrency within objects. OCPNs add two main constructs on top of Petri nets that allow further behavior: Typed places and variable arcs. Transitions can have input places from multiple types, acting as a synchronization point between objects. The non-shared transitions model concurrency between objects. The variable arcs allow for certain object types to participate in the transition with an arbitrary cardinality, modeling object multiplicity.

A super variant aims to represent two (or more) variants. It should incorporate additional concepts into variants that are discoverable when comparing two variants. OCPNs are able to capture three concepts that object-centric variants do not capture: concurrency within an object, choice, and object multiplicity. Only two of them can be detected when comparing two variants: choice and object multiplicity. Concurrency within an object needs large data amounts, it is equivalent to the problem of traditional process discovery [3]. Therefore, we define super variants as object-centric variants with added choice and object multiplicity.

Definition 5 (Super Variant). *A super variant is a tuple* $sv = (A, P, \pi_{nact}, \pi_{obj}, O_{sup})$ *consisting of nodes* A*, edges* $P \subseteq A \times A$*, node activities and choices* $\pi_{nact} : A \to \mathcal{A} \cup \mathcal{C}$*, node objects* $\pi_{nobj} : A \to \mathcal{P}(\mathcal{O})$*, and super objects* $O_{sup} \subseteq \pi_{vo}(sv)$ *where* $\pi_{vo}(sv) = \{\pi_{nobj}(a) \mid a \in A\}$*.* $\mathcal{C} = \mathcal{P}^{2+}(\mathcal{A}^*)$ *denotes the universe of choices, i.e., sets of at least two activity sequences including the empty sequence for optionality.*

We introduce the concept of object multiplicity through so-called *super objects*, i.e., objects that are a placeholder for an arbitrary number of actually instantiated objects. Choices are implemented by mapping a node to a set of activity sequences instead of a single activity. Every sequence is one option of the choice. If the empty sequence is part of this set, there is the option to skip. We define super variant equivalence as graph isomorphism under equivalence of activity and choice labels when mapping objects and super objects.

Definition 6 (Super Variant Equivalence). *Let* $sv_1 = (A_1, P_1, \pi_{nact,1}, \pi_{obj,1}, O_{sup,1})$ *and* $sv_2 = (A_2, P_2, \pi_{nact,2}, \pi_{obj,2}, O_{sup,2})$ *be super variants. These are equivalent iff there exists a mapping* $varmap = (\pi_{objmap}, \pi_{nmap})$ *consisting of a bipartite object mapping* $\pi_{objmap} : \pi_{vo}(sv_1) \to \pi_{vo}(sv_2)$*, and a bipartite node mapping* $\pi_{nmap} : A_1 \to A_2$ *such that*

- $(a, a') \in P_1 \Leftrightarrow (\pi_{nmap}(a), \pi_{nmap}(a')) \in P_2$ *(graph isomorphism),*
- $\forall_{a \in A_1}\ \pi_{nact,1}(a) = \pi_{nact,2}(\pi_{nmap}(a))$ *(activity and choice equivalence),*
- $\forall_{a \in A_1}\ o \in \pi_{nobj,1}(a) \Leftrightarrow \pi_{objmap}(o) \in \pi_{nobj,2}(\pi_{nmap}(a))$ *(equivalent objects),*
- $o \in O_{sup,1} \Leftrightarrow \pi_{objmap}(o) \in O_{sup,2}$ *(equivalent super objects).*

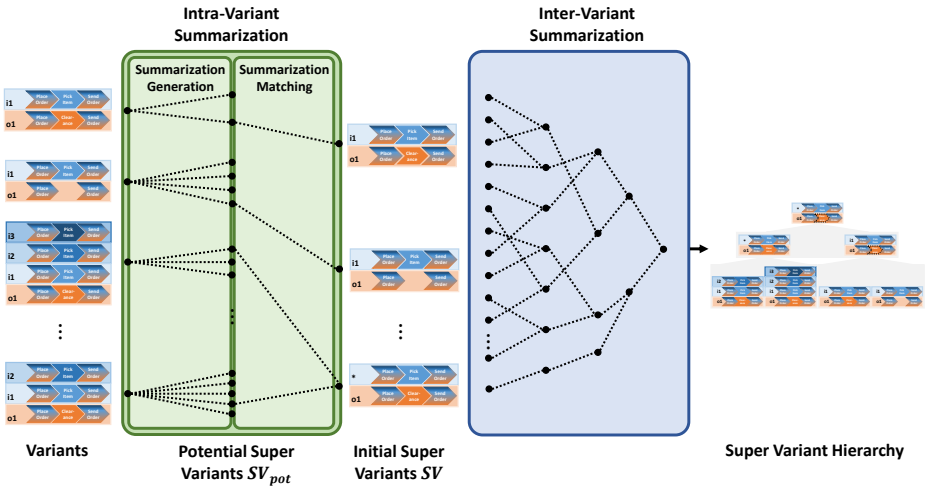


Fig. 2: Overview of our super variants mining framework. In the first step, potential super variants SV_{pot} for each variant are calculated and a minimal set of super variants SV to cover all initial variants is selected. In the second step, super variants are recursively matched and summarized to create the different hierarchy levels of the final super variant hierarchy.

This problem formulation is graph isomorphism under equality of node labels and object associations. We denote super-variant equivalence by $sv_1 \sim sv_2$.

Please note the close resemblance to variant equivalence. With the only differences being a subset of super objects and some activities being replaced by choices, one can reduce the problem of super variant equivalence to variant equivalence by creating new object types for super objects of each type and by sorting choices lexicographically and transforming them into an activity label. This enables the employment of labeled graph isomorphism techniques to determine super variant equivalence, as described in [8] for variant equivalence, which leads to efficient computation.

4 Super Variant Mining Framework

Our proposed super variants mining framework (cf. Figure 2) is split into two main steps: intra-variant summarization and inter-variant summarization. In the intra-variant summarization, we compute potential initial super variants for each variant and choose the minimal set of super variants to cover all variants. In the inter-variant summarization, we recursively merge the most similar super variants into a combined super variant to generate a super variant hierarchy.

4.1 Intra-Variant Summarization

Intra-variant summarization consists of two main steps: generating potential super variants and selecting an initial set of super variants. The first step has a

central function that can generate potential super variants given a variant, i.e., super variants that generalize the behavior seen in a variant.

Definition 7 (Intra-Variant Summarization). *Let $v = (A, P, \pi_{nact}, \pi_{nobj})$ be an object-centric variant. An intra-variant summarization maps the object-centric variant to a set $SV = \{sv_1, \ldots, sv_n\}$ of super variants that constitute a generalization of the variant's control flow $intra(v) = \{sv_1, \ldots, sv_n\}$.*

The potential super variants generated in this stage can cover object multiplicity that can already be observed by looking at one variant or choices that happen for different objects of the same type within a variant. An example of a possible intra-variant summarization can be seen in Figure 1. Since some objects within the first (a) and second (b) variant show exactly the same dependencies and control flow, they can be merged into a super object.

With multiple potential super variants per variant, we need to select a set of initial super variants. A super variant can be a potential super variant for two different variants. We choose an initial set of super variants with the goal of minimizing the size of the set of initial super variants.

Definition 8 (Intra-Variant Summarization Matching). *Let $v_1, \ldots, v_m$ be object-centric variants with summarizations $intra(v_i) = \{sv_{i,1}, \ldots, sv_{i,n}\}$ for $1 \leq i \leq m$, providing a set of potential super variants $SV_{pot} = \bigcup_{1 \leq i \leq m} intra(v_i)$. The variant summarization matching is the smallest set of summarizations that contains a summarization of each variant:*

$$\min_{l \in \mathbb{N}} \{SV = \{sv_1, \ldots, sv_l\} \subseteq SV_{pot} \mid \forall_{1 \leq i \leq m} \exists_{1 \leq j \leq n} \exists_{1 \leq k \leq l} \; sv_{i,j} \sim sv_k\}$$

This can be reduced to a hitting-set problem [25].

We end up with a problem that can be reduced to a hitting set problem, i.e., finding a subset of given size that covers an element of each set. Please note, this is an NP-complete problem [25].

4.2 Inter-Variant Summarization

We generate a super variants hierarchy based on the initial super variant set. This happens in the inter-variant summarization step. We recursively merge two super variants into a new one that covers the control-flow behavior of both.

The central step of inter-variant summarization is the merging of two super variants into a new super variant covering the behavior of both. This step is called the inter-variant summarization itself. We define an abstract function that fulfills these requirements.

Definition 9 (Inter-Variant Summarization). *Let $sv_1 = (A_1, P_1, \pi_{nact,1}, \pi_{obj,1}, O_{sup,1})$ and $sv_2 = (A_2, P_2, \pi_{nact,2}, \pi_{obj,2}, O_{sup,1})$ be two super variants. An inter-variant summarization constructs a super variant sv_3=$(A_3, P_3, \pi_{nact,3}, \pi_{obj,3}, O_{sup,3})$ that constitutes a generalization of the control flow of both variants $inter(sv_1, sv_2) = sv_3$.*

An inter-variant summarization function enables us to merge two super variants into a new one. However, it does not tell us which super variants to merge. We would like to merge the most similar super variants to minimize the amount of newly-added operators. To this end, we define a distance function that measures the similarity of two super variants.

Definition 10 (Super Variant Distance). *Let* $sv_1 = (A_1, P_1, \pi_{nact,1}, \pi_{obj,1}, O_{sup,1})$ *and* $sv_2 = (A_2, P_2, \pi_{nact,2}, \pi_{obj,2}, O_{sup,2})$ *be super variants. A function* $\delta_{dist}(sv_1, sv_2) \in \mathbb{N}$ *is a super variant distance function such that* $\delta_{dist}(sv_1, sv_2) = 0$ *iff* $sv_1 \sim sv_2$*, i.e., equivalent super variants have a distance of 0.*

The closest variants need to be matched such that they can be summarized together. We define a matching problem for that. Additionally, we allow for some variants to be not matched, given a penalty to allow for an uneven number of super variants and outlier behavior.

Definition 11 (Inter-Variant Matching). *Let* $SV = \{sv_1, \ldots, sv_n\}$ *be a set of super variants. Given a non-assignment penalty* $\beta \in \mathbb{R}^+$ *the matching between super variants is a minimization problem:*

$$
\begin{aligned}
&\underset{M \subseteq (SV \times SV)}{\arg\min} && \sum_{(sv_i, sv_j) \in M} \delta_{dist}(sv_i, sv_j) + \beta C \\
&\textit{subject to} && |\{(sv_i, sv_j) \in M \mid sv_i \nsim sv_j \wedge (sv_i \sim sv' \vee sv_j \sim sv')\}| \leq 1 \quad \forall sv' \in SV, \\
& && D = \bigcup_{(sv_i, sv_j) \in M} \{sv_i, sv_j\} \quad , \\
& && C = |SV \setminus D| \quad .
\end{aligned}
$$

Our framework consists of five main steps: intra-variant summarization, summarization matching, inter-variant summarization, variant distance function, and matching. While both matchings are defined as optimization problems, the summarizations and the distance function provide design choices. In the following section, we briefly describe the choices for our initial framework implementation.

5 Implementation

We implemented our framework based on the OCPA library [7]. The code and all following experiments are publicly available[‡]. We provide a description of the intra-variant summarization, inter-variant summarization, and distance function we use. We refer to the repository for the detailed computations. Furthermore, we provide a description of our visualization technique.

5.1 Computation

In the intra-variant summarization (cf. Definition 7), we merge objects into super objects and introduce possible choices observed between objects of the same type.

[‡] `https://github.com/EmilieHK/Object-Centric-Super-Variants`

To begin with, we find all objects with the same dependencies as other objects. Only such objects can be merged into a super object or used to infer choices, as we would otherwise introduce different object dependencies. Note that object dependency is a transitive relationship. In the retrieved sets of objects with the same dependencies, we check for equivalence of paths to form super objects. Furthermore, we check for the existence of choices to merge two objects into a super object under the introduction of a choice.

We define the distance function (cf. Definition 10) between super variants using Levenshtein distance [27]. We go through different object mappings between two super variants that preserve object dependencies. We find the object mapping that provides the minimal aggregated Levenshtein distance for all objects. This minimal Levenshtein distance is the distance between two super variants.

In the inter-variant summarization (cf. Definition 9), we find a super variant that covers the behavior of both input super variants. To do this, we find a mapping between the objects of both super variants with respect to the dependencies between objects and behavioral closeness per object, i.e., the same mapping found through the calculation of the distance. Using this mapping, we can identify additional (optional) objects as well as choices that are happening between two mapped objects.

5.2 Visualization

We use a visualization close to the initially proposed visualization for object-centric variants [8]. First, we explain the basic concept behind the visualization of object-centric variants, and second, we introduce the additional visualization elements that are used for super variants. We refer to the example in Figure 1.

An object-centric variant is visualized using three main components: object colors, activity chevrons and object lanes. Every object type has a different base color and every object of that type has a different shade of that base color. An activity chevron symbolizes the occurrence of an activity. It can be colored in multiple colors if multiple objects are involved in this activity. An object lane contains all activity chevrons associated with an object ordered by time, i.e., it is a traditional variant for one specific object. If an activity is shared, it is placed in all involved object lanes on the same horizontal position. Only shared activities impose an ordering between activities of different objects. Activities of different objects before and after a shared activity are concurrent.

We incorporate the additional concepts of choice and object multiplicity in the following way. For object multiplicity, i.e., super objects, we use a notation of a star instead of an object identifier, indicating the arbitrary number of times an object can be instantiated. We visualize choice using a vertical split of the choice within a chevron. If one option of the choice is empty, i.e., the choice expresses optionality, we draw the chevron of the activity using a dotted border.

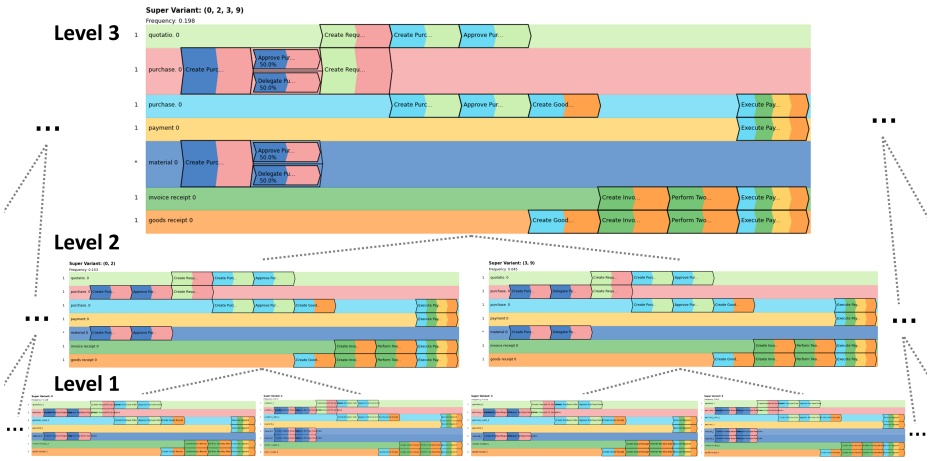


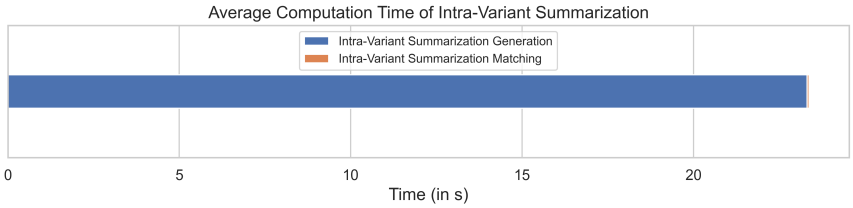
Fig. 3: Part of the super variant hierarchy mined from the procure-to-pay event log. From levels 1 to 2, the variants are summarized with super objects for material objects. From levels 2 to 3, the variants are summarized with a choice between approving or delegating a purchase order. The resulting super variant covers 20% of the control flow.

6 Proof of Concept

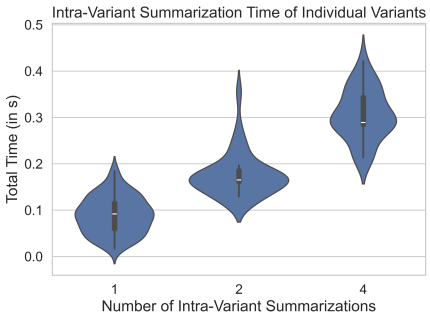
We apply our proposed super variants mining framework to an event log to mine a super variant hierarchy. We use a procure-to-pay event log [37]. The event log contains 516 process executions that have 314 different object-centric variants. There are 5128 objects of 7 object types. We depict a sub-hierarchy of the generated super variant hierarchy to ensure readability of the results. The remaining super variants can be found in the repository. The corresponding sub-hierarchy is depicted in Figure 3. The hierarchy shows how four object-centric variants are summarized into one super variant over three hierarchy levels.

The underlying process is a procure-to-pay process. A purchase request associated with multiple materials is created. This purchase request triggers the creation of a quotation object which itself will create a purchase order. After the order arrives, a good receipt will be created for the purchase order. An invoice is created and the order is paid.

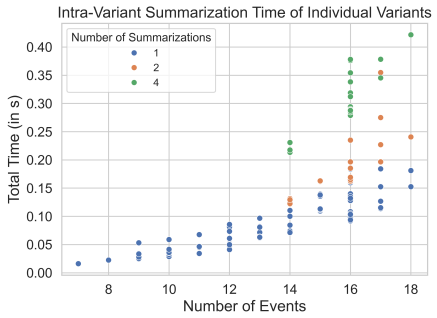
The first level of the hierarchy contains four variants. These variants are pairwise matched and summarized. From the first to the second level, the intra-variant summarization algorithm introduces super objects for material since the variants show different numbers of material objects. From the second to the third level, the two super variants are summarized under introduction of the choice operator. The inter-variant summarization algorithm uncovers the choice between approving or delegating the purchase order as the difference between the super variants. This choice is propagated to the connected material super object. We end up with one super variant with a material super object and a choice between delegating and approving the purchase order that covers 20%



(a) Decomposed computation time.



(b) Summarization generation time based on the number of summarizations.



(c) Summarization generation time based on the number of events.

Fig. 4: Intra-variant summarization computation time decomposed into individual parts and analyzed on the variant level. We analyze the relationship to the number of generated intra-variant summarizations and events.

of the control-flow behavior of the whole event log while it only adds minimal additional elements to the initial variants, namely an object multiplicity operator (*) and a choice between two activities. This shows super variants' usefulness in summarizing several variants into one understandable representation.

7 Scalability Evaluation

Since the super-variant mining framework is composed of computationally demanding problems we evaluate the scalability of our approach to ensure its real-life feasibility. We drill down the scalability of intra- and inter-variant summarization into the different compounding factors. We use the procure-to-pay event log and different numbers of variants as input to evaluate the scalability. We generate five super variant hierarchies of five levels with thirty-two randomly selected variants and average the computation times.

7.1 Intra-Variant Summarization

Figure 4a depicts the decomposed average computation time of the intra-variant summarization. We observe that the largest part of the computation is used

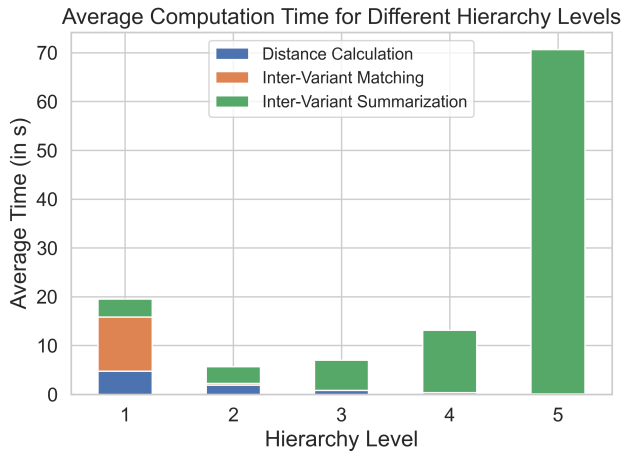


Fig. 5: Decomposition of the inter-variant summarization running times into the three steps.

for generating the potential super variants, i.e., the possible summarizations. Since the summarization exhibits a larger share of the computation time, we investigate factors that influence the computation time of the summarization for individual variants. The results are depicted in Figure 4b and Figure 4c. There are two main findings: the more potential super variants we compute and the more events (nodes) a variant has, the longer the computation time. Since the number of events is a non-changeable parameter, one could limit the number of generated summarizations to limit the computation time.

7.2 Inter-Variant Summarization

Figure 5 depicts the decomposed average computation time of inter-variant computation depending on the hierarchy level. Hierarchy level 1 is the lowest hierarchy level, i.e., the level using the intra-variant summarization as input. We observe that a larger number of super variants leads to higher computation times in the distance calculation and inter-variant matching. These computation times decrease with lower numbers of variants in the corresponding hierarchy level. In the last hierarchy level, the inter-variant summarization time dominates all other times. For super variant hierarchies with many more variants, this can pose a scalability problem. This inter-variant summarization is the first of the introduced problems that needs scalability solutions to ensure applicability to large event logs.

8 Related Work

In this section, we discuss process discovery, variant analysis, and related approaches to finding super variant hierarchies. First, we discuss object-centric

event data, the starting point for our work. Second, we introduce relevant work on variant analysis and object-centric process discovery. Third, we introduce related approaches to super variants.

Real-life processes often consist of objects with complex co-dependencies [19]. This is a generalization of traditional process mining: Instead of one object per end-to-end execution of the process there are now multiple objects involved. Forcing the event data of such *object-centric processes* into a traditional event log format leads to quality issues [36]. Therefore, we base our work on object-centric event data.

A variant is a sequence of activities that is a viable process execution according to the event log. Variant analysis is the exploration of all individual control-flow variants to derive insights into the process [34]. It is a key technique in most practical process assessment and auditing frameworks [24,15], analyzing variant characteristics [44] in a mostly visual way [19,41,11]. Several approaches have been proposed to incorporate higher-level concepts into variants, such as parallelism of non-atomic events [4,40], concurrency between objects in object-centric variants [8] or other structures control-flow [39]. However, no approach has been introduced to mine variants that are summarizations of multiple variants.

Process discovery has the opposite approach of variant analysis: All variants are summarized into one single model. A plethora of object-centric process discovery approaches have been proposed: Object-centric Petri nets [3], proclets [32,31], cardinality and declarative constraints [28], and discovery with object-type clustering [23]. While comprehensive, they lack the granularity and simplicity of variants as they can grow overly complex [1].

The idea of super variants has two main components: Finding similar patterns between variants and constructing a hierarchical control-flow from them. There are two main categories of techniques that relate to these ideas: Local process models [43] and hierarchical process models [21]. These two have opposing approaches: Local process models explicitly discover frequent patterns across variants while hierarchical process models mine models that abstract frequent patterns away to show them in a lower level of the hierarchy.

Local process model mining offers techniques to discover local behavior in variants that contain concurrency [26,43], exclusive choices [14,30] as well as loops [43,22]. An iterative construction of local models based on joining existing patterns yields a set of defining structures present in the process [38,5]. The main differentiation to our work is the focus on end-to-end behavior: While we preserve end-to-end behavior of a process execution, local process model mining focuses on only local behavior.

Hierarchical process discovery aims to discover a process model that is composed of multiple different submodels arranged in a hierarchy [21,33] by utilizing pattern mining [10,29]. The models are mined through a bottom-up construction of a lattice structure [17,16]. There are two main differentiations to our work: First, the variants are still summarized into one single process model. Second, the complexity of the model is still the same, sub-behavior is only abstracted away

in the hierarchy. Therefore, no approach equivalent to super variants has been proposed, constituting a middle ground between variants and process models in terms of summarization and complexity.

Trace clustering methods follow the objective of grouping behaviorally close traces together [45,42,13]. Few clustering approaches use a top-down strategy to discovery hierarchies by checking the conformance of clusters to discovered process models [21,33]. While such an approach allows a decomposed exploration of the control flow, the connection to specific variants is lost and the clustering is not inherently tied to specific control-flow differences that are considered in a visualization.

9 Conclusion

We introduced the concept of super variants in this paper. Super variants aggregate closely related variants into relatively simple models, less complex than process models but more comprehensive than variants. To discover super variants, we proposed a super-variant mining framework consisting of two main steps and five main problems: intra-variant summarizations, intra-variant summarization matching, inter-variant summarization, super variant distance, and inter-variant matching. With each of these problems being computationally demanding, we evaluated the scalability of the super-variant mining framework to investigate real-life applicability and uncover potential points of future improvement. We proposed a super variant visualization and applied the framework to a procure-to-pay event log to mine a super variant hierarchy. The results show promising levels of control-flow aggregation for little added complexity, providing a proof of concept. While super variants are defined on object-centric event data, the concept is equally applicable to case-centric event data as case centricity is a special case of object centricity.

Two main limitations can be addressed in future work: technical limitations and more elaborate evaluations of added value. Technical limitations include the inter-variant summarization scalability and vagueness of super variants. Inter-variant summarization has shown to be the part of our framework with the highest computation time. A scalable practical application necessitates an efficient inter-variant summarization. Highly optimized code or heuristics could help to reduce the computation time. With respect to object multiplicity, super variants are similar to object-centric Petri nets. If there is multiplicity, the number of objects is arbitrary and it is not defined whether objects are batched or not. A more restrictive formalization of super variants can address this. However, the generalization of super variants would suffer from this. This paper has provided the general concept and a brief proof of concept where we could aggregate a large part of the control-flow information in one super variant. However, we would like to assess the added benefit to the end user in future research, proofing the real-life value of super variants.

References

1. van der Aalst, W.M.P.: Process discovery: Capturing the invisible. IEEE Comput. Intell. Mag. **5**(1), 28–41 (2010)
2. van der Aalst, W.M.P.: Process mining: A 360 degree overview. In: Process Mining Handbook, pp. 3–34. Springer (2022)
3. van der Aalst, W.M.P., Berti, A.: Discovering object-centric Petri nets. Fundam. Informaticae **175**(1-4), 1–40 (2020)
4. van der Aalst, W.M.P., Santos, L.: May i take your order? In: BPM Workshops. pp. 99–110. Springer (2022)
5. Acheli, M., Grigori, D., Weidlich, M.: Efficient discovery of compact maximal behavioral patterns from event logs. In: CAiSE. pp. 579–594. Springer (2019)
6. Adams, J.N., van der Aalst, W.M.P.: Addressing convergence, divergence, and deficiency issues. In: BPM Workshops. Springer (2023)
7. Adams, J.N., Park, G., van der Aalst, W.M.P.: ocpa: A python library for object-centric process analysis. Softw. Impacts **14**, 100438 (2022)
8. Adams, J.N., Schuster, D., Schmitz, S., Schuh, G., van der Aalst, W.M.P.: Defining cases and variants for object-centric event data. In: ICPM. pp. 128–135. IEEE (2022)
9. Augusto, A., Conforti, R., Dumas, M., Rosa, M.L.: Split miner: Discovering accurate and simple business process models from event logs. In: ICDM. pp. 1–10. IEEE (2017)
10. Begicheva, A.K., Lomazova, I.A., Nesterov, R.A.: Discovering hierarchical process models: an approach based on events clustering (2023)
11. Berti, A., Li, C.Y., Schuster, D., van Zelst, S.J.: The process mining toolkit (PMTK): Enabling advanced process mining in an integrated fashion. ICPM Demos (2021)
12. Bodesinsky, P., Alsallakh, B., Gschwandtner, T., Miksch, S.: Exploration and assessment of event data. In: EuroVA@EuroVis. pp. 67–71. EA (2015)
13. Bose, R., van der Aalst, W.M.: Context aware trace clustering: Towards improving process mining results. In: SDM. pp. 401–412 (04 2009)
14. Chen, W., Lu, J., Keech, M.: Discovering exclusive patterns in frequent sequences. Int. J. Data Min. Model. Manag. **2**(3), 252–267 (2010)
15. Chiu, T., Jans, M.: Process Mining of Event Logs: A Case Study Evaluating Internal Control Effectiveness. Accounting Horizons **33**(3), 141–156 (2019)
16. Diamantini, C., Genga, L., Potena, D.: Behavioral process mining for unstructured processes. Journal of Intelligent Information Systems **47**(1), 5–32 (Aug 2016)
17. Diamantini, C., Potena, D., Storti, E.: Mining usage patterns from a repository of scientific workflows. In: SAC. p. 152–157. ACM (2012)
18. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, Second Edition. Springer (2018)
19. Fahland, D.: Extracting and pre-processing event logs (2022). https://doi.org/10.48550/arXiv.2211.04338
20. Fahland, D.: Process mining over multiple behavioral dimensions with event knowledge graphs. In: Process Mining Handbook, pp. 274–319. Springer (2022)
21. Greco, G., Guzzo, A., Pontieri, L., Saccà, D.: Discovering expressive process models by clustering log traces. IEEE Trans. Knowl. Data Eng. **18**(8), 1010–1027 (2006)
22. Jagadeesh Chandra Bose, R.P., van der Aalst, W.M.P.: Abstractions in process mining: A taxonomy of patterns. In: BPM. pp. 159–175. Springer (2009)

23. Jalali, A.: Object type clustering using Markov directly-follow multigraph in object-centric process mining. IEEE Access **10**, 126569–126579 (2022)
24. Jans, M., Eulerich, M.: Process mining for financial auditing. In: Process Mining Handbook, pp. 445–467. Springer (2022)
25. Karp, R.M.: Reducibility among combinatorial problems. In: 50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art, pp. 219–241. Springer (2010)
26. Leemans, M., van der Aalst, W.M.P.: Discovery of frequent episodes in event logs. In: Data-Driven Process Discovery and Analysis. pp. 1–31. Springer, Cham (2015)
27. Levenshtein, V.I., et al.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet physics doklady **10**(8), 707–710 (1966)
28. Li, G., de Carvalho, R.M., van der Aalst, W.M.P.: Automatic discovery of object-centric behavioral constraint models. In: BIS. pp. 43–58. Springer (2017)
29. Liu, C.: Hierarchical business process discovery: Identifying sub-processes using lifecycle information. In: ICWS. pp. 423–427 (2020)
30. Lu, J.: Exclusive sequential patterns and their graphical representation. In: IADIS. pp. 71–78 (7 2011)
31. Lu, X.: Artifact-centric log extraction and process discovery. Master's thesis, Eindhoven University of Technology (2013)
32. Lu, X., Nagelkerke, M., Wiel, D., Fahland, D.: Discovering interacting artifacts from erp systems. IEEE Transactions on Services Computing **8**, 1–1 (11 2015)
33. Medeiros, A., Guzzo, A., Greco, G., van der Aalst, W.M., Weijters, A., Dongen, B., Saccà, D.: Process mining based on clustering: A quest for precision. In: BPM Workshops. pp. 17–29. Springer (2007)
34. Milani, F., Lashkevich, K., Maggi, F.M., Di Francescomarino, C.: Process mining: A guide for practitioners. In: RCIS. pp. 265–282. Springer (2022)
35. Miller, G.A.: The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychological review **63**(2), 81 (1956)
36. Nooijen, E.H.J., van Dongen, B.F., Fahland, D.: Automatic discovery of data-centric and artifact-centric processes. In: BPM Workshops. pp. 316–327. Springer (2013)
37. Park, G., Tacke genannt Unterberg, L.: Procure-To-Payment (P2P) Object-centric Event Log in OCEL 2.0 Standard (2023). https://doi.org/10.5281/zenodo.8412920
38. Peeva, V., Mannel, L.L., van der Aalst, W.M.P.: From place nets to local process models. In: PETRI NETS. pp. 346–368. Springer (2022)
39. Rubensson, C., Mendling, J., Weidlich, M.: Variants of variants: Context-based variant analysis for process mining. In: CAiSE. pp. 387–402. Springer (2024)
40. Schuster, D., Schade, L., van Zelst, S.J., van der Aalst, W.M.P.: Visualizing trace variants from partially ordered event data. In: ICPM Workshops. pp. 34–46. Springer (2022)
41. Schuster, D., van Zelst, S.J., van der Aalst, W.M.P.: Cortado—an interactive tool for data-driven process discovery and modeling. In: PETRI NETS. pp. 465–475. Springer (2021)
42. Song, M., Günther, C.W., van der Aalst, W.M.P.: Trace clustering in process mining. In: BPM Workshops. pp. 109–120. Springer (2009)
43. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.: Mining local process models. Journal of Innovation in Digital Ecosystems **3**(2), 183–196 (2016)
44. Taymouri, F., Rosa, M.L., Dumas, M., Maggi, F.M.: Business process variant analysis: Survey and classification. Knowledge-Based Systems **211**, 106557 (2021)
45. Zandkarimi, F., Rehse, J.R., Soudmand, P., Hoehle, H.: A generic framework for trace clustering in process mining. In: ICPM. pp. 177–184 (2020)