Analyzing Actor Behavior in Process Executions

Eva L. Klijn, Felix Mannhardt, Dirk Fahland
Eindhoven University of Technology, Mathematics and Computer Science, Eindhoven, the Netherlands
{e.l.klijn, f.mannhardt, d.fahland}@tue.nl

***Abstract*—Process executions are driven by actors and machines jointly performing work on incoming cases. Actors typically divide and structure their work into *tasks* – multiple consecutive actions performed together – before handing a case to the next actor. *Routines research* investigates how and why actors divide and structure work in a process, how it changes over time, and its impact on process executions. So far, process event log data has not been used to investigate these phenomena. We conducted an exploratory case study to identify process mining concepts and data structures that help answering the above questions. We found that modeling case and actor traces together in an event knowledge graph allows to identify instances of task executions; clustering task instances reveals tasks. Extending the event knowledge graph by aggregation wrt. tasks reveals, both, local process models of intra-task behavior, and global process models of inter-task behavior in a case and between actors. We show on the BPIC'17 dataset that querying the extended graph reveals new insights into (changes in) actor behavior, work division, and significant impacts on performance and outcomes.**

***Index Terms*—knowledge graph, actor, routines**

I. Introduction

Processes are executed by human actors and automated resources performing work on the cases of the process. For example, multiple employees of a bank jointly check a credit application, create (one or more) loan offers, contact the client for additional information, to finally decline or prepare a contract. The work on each case itself is structured into actions ordered by the process' control-flow [1].

In contrast, human actors typically structure their work differently by performing multiple actions on the same case before handing the case to the next actor, e.g., creating and sending two loan offers to the same client; such a larger unit of work is called *task* [2], [3] in routines research. Actors exhibit behavior on their own within a case and across cases, and by handing work to other actors. Routines research investigates, among others, (Q1) how actors jointly structure and divide work in a process into (recurring) tasks, i.e., larger units of work [4], (Q2) which factors determine differences in how actors performs task [2], (Q3) whether this changes over time [4], and (Q4) whether differences in tasks or actor behavior impacts process outcomes or performance [2].

Event data in principle holds all relevant information about actor behavior, e.g., sequences of events performed by actors in various process executions. However, process mining has not been used to comprehensively investigate *how actors structure and organize their work and how this impacts process executions* with the aim of answering (Q1)-(Q4). We conducted an exploratory case study to test the hypothesis that *event data, as commonly used in process mining, does indeed hold the required information to investigate (Q1)-(Q4) quantitatively and qualitatively*. We chose the BPIC'17 dataset [5] as it records in each event the individual actor involved.

We argue that investigating actor-process interactions from event data requires to analyze two behavioral dimensions together: the classical traces of all process cases and the traces of all actors working across all cases. The default data model of sequential event logs used in process mining is unable to do so [6]. In previous work, recalled in Sect. III, we showed how to translate a log into an *event knowledge graph* [6] modeling traces of cases and actors in the same data structure; any sub-graph where an actor follows multiple events in a case corresponds to an execution of some task [3]. 98% of the BPIC'17 events are shown to be part of a larger task; however, the actual *contents* of these tasks has not been studied.

To investigate our hypothesis, we explored the event knowledge graph of BPIC'17 over traces of cases and actors and iteratively identified several aggregation operations on the graphs to identify tasks and analyze actor behavior over tasks. Each aggregation operation extends the graph with new information that can be used by subsequent aggregation operations, shown in Sect. IV: (1) Clustering the sub-graphs of task executions by the contained action names identifies homogeneous tasks with multiple, similar variants. (2) Aggregating all sub-graphs of a cluster results in a Local Process Model (LPM) [7] summarizing the behavior within each task. (3) Abstracting a cluster into a task node allows to aggregate the underlying traces (over cases and actors) into a global process model over tasks. (4) Filtering allows to create more specific models describing process and actor behavior wrt. tasks.

Applying our techniques on BPIC'17 we found 20 distinct tasks, each comprising 2-43 variants: (Q1) We found several tasks that overlap in their actions splitting up cases into very different units of work. (Q2) Different tasks are performed by different sets of actors; the choice between tasks with overlapping actions depends for some on process characteristics and for others on actor preferences. (Q3) We found actors change preference in which task to perform for the same objective in the process; this change in preference occurred at the same moment in time but in a non-uniform way. (Q4) We observed that choosing different tasks has an impact on the execution of some parts of the process and on process performance. Details are given in Sect. V and we conclude in Sect. VI.

II. Related Work

We compare to related work that is also concerned with analyzing actor behavior from event logs.

Information on actors in processes has been previously used in process mining, e.g., work assignment rules [8], resource productivity [9], and resource availability [10]. None of them consider work organized into larger tasks or collaboration between actors. Kumar and Lui [11] do consider the collaboration between actors and detect actor hand-off patterns; however, the contents of work between hand-offs is disregarded. Recently, Yang et al. [12] discover more comprehensive organizational models that include grouping of resources and their relation to execution contexts. However, an execution context consists of single activities disregarding work may be aggregated to the task level. Hulzen et al. [13] define the closely related concept of resource profiles and cluster several activity instance to activity instance archetypes related to actors, which can be seen as representing a task. Delcoucq et al. [14] analyze control-flow and resource perspective by clustering over an actor-activity matrix. Both works [13] and [14] ignore the behavior within a task and between actors. In an extension [15] LPMs of actor behavior are discovered. Only frequent, gapped behavior of an actor over the entire trace is considered instead of consecutive actions forming a task. Resulting models are not related to the case making it impossible to study their context and impact on case execution.

Methods on batch processing detection use the resource perspective in event logs to identify executions of a certain activity by a certain resource across cases [16], possibly interrupted [17], that are likely performed in a batch. This was extended to more complex behavior forming sub processes in [16]. Batches may form a larger unit of work performed by a single actor. Such batch work may coincide with tasks detected by our method; however, the tasks we aim to analyze are not necessarily batched. Pika et al. [17] investigates BPIC'17 and detects some batching related to the single activity *O_Accepted*. Manual investigation described in [17] revealed that some actors are processing this activity in batches. These batches are interrupted by three other activities, which according to [17] indicates a sub process. Indeed, we also find the same activities to be involved in a task. Note that our method directly identified this task without using the timing of events. Thus, the analysis of actor behavior at task level and batch processing are complementary concerns.

Event abstraction methods aim to identify and aggregate high-level activities from lower-level events. The aggregation performed is not necessarily related to actor behavior and, indeed, only very few approaches use data on actors. A notable exception are Senderovich et al. [18] who use information on the (spatial) interaction between resources in a hospital to derive activities from location sensor data. Leoni and Dündar [19] use waiting time between events as heuristic to group consecutive low-level events into "batch sessions" and cluster them using the most frequently executed activity in a cluster as label. In a single actor setting this may identify tasks through temporally close activities. Compared to event abstraction we aim to identify tasks in cases worked on by multiple actors and use actors as features in subsequent analysis without relying on time as heuristic. We preserve the structure of the cluster as an LPM which allows us to meaningfully compare clusters describing different ways of dividing the same work over time.

Concept drift studies gradual or sudden changes in processes that can be observed over time. We investigate concept drift in actor behavior of the BPIC'17 event log. Few approaches for concept drift identification and comparison consider the resource dimension. None of them looks at the actor-case interplay and at concept drift at the task level. The only multi-perspective comparison method, including the resource perspective, was proposed by Nguyen et al [20]. It allows to identify significant differences between event log subsets using a graph representations of a certain process perspectives. In contrast to our method the aggregation of events and comparison is done for each case separately and no analysis of actor traces is provided. Adams et al. [21] propose a detection method for concept drift that considers resources by aggregating the information on them (e.g., workload) into a numeric representation. A concept drift in BPIC'17 with increased workload for resources is found. Whereas we do not consider workload, none of the more complex changes in actor behavior can be found by [21] due to the encoding into a flat numeric representation.

Finally, task mining is also related. Task mining aims to investigate actor routines by leveraging desktop interaction logs. Several approaches [22], [23] aim to discover task executions by segmenting an event log of desktop interactions such that repetitive patterns or pre-identified routines are found similar to our previous work [24]. However, tasks are limited to a single actor ignoring collaboration and do not investigate changes and process context.

III. Preliminaries

We recall how to build a multi-dimensional representation using event knowledge graphs, how to aggregate specific multi-dimensional dynamics into task instances, and how to aggregate events into event classes.

A. Event Data Representation in Event Graphs

A process-aware system can record an action execution as an *event* in an event log. Each event records at least the *action* that occurred, the *time* of occurrence, and at least one *entity identifier* indicating on which entity the action occurred. An event can also record entity identifiers for multiple types of entities, e.g., *case identifier* and *actor identifier* (or *resource*). An event can also record additional *attributes* describing the event or case further. Tab. I shows an example event log containing 10 events occurring on the same day.

Process mining [1] analyzes event data by grouping events wrt. a chosen *case identifier* attribute. Ordering all events of a case by time yields the *trace* as a sequence of events. Grouping the events in Tab. I by *Case* yields the traces $\langle e1, e2, e3, e4, e5\rangle$ and $\langle e6, e7, e8, e9, e10\rangle$. We can also group the events in Tab. I by *Resource*, yielding the traces $\langle e1, e2, e6, e7, e8\rangle$ (resource $a1$), $\langle e3, e4, e9, e10\rangle$ ($a5$), and $\langle e5\rangle$ ($a29$). An event log can only model traces wrt. one dimension (case *or* actor). We use

TABLE I: Example of an event table.

Event	Action	Time	Case	Resource	Item Category
e1	A	12:02	c3	a1	Electronics
e2	B	12:04	c3	a1	Electronics
e3	E	14:38	c3	a5	Electronics
e4	F	14:41	c3	a5	Electronics
e5	C	16:21	c3	a29	Electronics
e6	A	12:08	c4	a1	Clothing
e7	B	12:09	c4	a1	Clothing
e8	D	12:15	c4	a1	Clothing
e9	E	14:54	c4	a5	Clothing
e10	F	14:59	c4	a5	Clothing

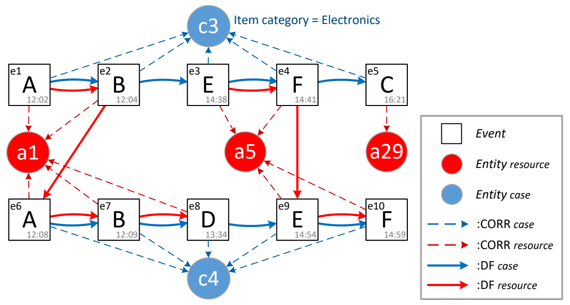


Fig. 1: Event graph with the events and entities from Tab. I.

event knowledge graphs [6] to model traces in both dimensions together (case *and* actor).

An event knowledge graph is a specific type of labeled property graph (LPG) that is used in Graph Databases [25] for modeling various entities as nodes and various relationships as edges between them. In an LPG G, each node o and each relationship R with edge $\overrightarrow{R} = (o, o')$ from o to o' has a label ℓ, denoted $o \in \ell$ or $R \in \ell$; $x.a = v$ denotes that property a of node/relationship x has value v. An event knowledge graph is an LPG G that can be obtained from an event table, e.g., Fig. 1 shows the graph for events in Tab. I; each event and each entity (i.e., each case id or resource id) is represented by a node with label Event or Entity, respectively. Each $e \in Event$ defines $e.action$ and $e.time$; each $n \in Entity$ defines $n.type$ (e.g., case or resource). The graph has relationship labels:

- CORR (correlation): $R \in CORR$, $\overrightarrow{R} = (e, n)$ iff event $e \in Event$ is correlated to entity $n \in Entity$; we write $(e, n) \in CORR$ as short-hand.
- DF (directly-follows): $R \in DF$, $\overrightarrow{R} = (e, e')$ iff events e, e' are correlated to the same entity n $(e, n), (e, n') \in CORR$, $e.time < e'.time$ and there is no other event $(e'', n) \in CORR$ with $e.time < e''.time < e'.time$; we write $(e, e')^{n.type} \in DF$ as short-hand, i.e., $(e, e')^c$ for entity type case and $(e, e')^r$ for actor.

See [6] for formal details and how to create G from events.

The example in Fig. 1 shows the event graph obtained from Tab. I: each square (white) node is an *Event* node; each circle is an *Entity* node of the corresponding type (blue for *Case*, red for *Resource*). *CORR* relationships are shown as dashed edges, e.g., $e1, e2, e3, e4, e5$ are correlated to case $c3$ and $e3, e4, e9, e10$ are correlated to resource $a5$. *DF*-relationships are shown as solid edges. The *DF*-relationships between the events correlated to the same entity form a *DF-path* for that entity; the graph in Fig. 1 defines 2 DF-paths for case entities, e.g., $\sigma_{c3} = \langle (e1, e2)^c, (e2, e3)^c, (e3, e4)^c, (e4, e5)^c \rangle$ and 3 DF-paths for resource entities, e.g., $\sigma_{a5} = \langle (e3, e4)^r, (e4, e9)^r, (e9, e10)^r \rangle$.

B. Detecting Task Instances

In [3] we have shown that a connected subgraph where a case and a resource DF-path synchronize for several subsequent events forms a "unit of work" or *task execution*. While a variety of such task subgraphs can be characterized [3], we here focus on the most simple ones: a subgraph of events $\{e_1, ..., e_k\}$ and adjacent DF-edges that contains (1) exactly one (part of a) case DF-path $\langle \ldots (e_1, e_2)^c, \ldots, (e_{k-1}, e_k)^c \ldots \rangle$ for a case c and (2) exactly one (part of an) actor DF-path $\langle \ldots (e_1, e_2)^r, \ldots, (e_{k-1}, e_k)^r \ldots \rangle$ for an actor r, i.e., both paths synchronize over the same subsequent events; see [3] for details. In Fig. 1, subgraphs of events that meet these criteria are $\{e_1, e_2\}$, $\{e_3, e_4\}$, $\{e_5\}$, $\{e_6, e_7, e_8\}$ and $\{e_9, e_{10}\}$. Each such subgraph ti describes one *task instance*.

C. Aggregation of Events

An event knowledge graph G can be extended. We specifically use (1) aggregation of sets of events in G into a new kind of node, and (2) lifting DF-edges from events to the aggregated nodes. We use two types of aggregation.

1) Aggregation of Events into Task Instances: We aggregate each task instance subgraph ti into a new "high-level" event node $h_{ti} \in TaskInstance$ and add relationship $(e, h_{ti}) \in CONTAINS$ for each $e \in ti$ and properties $h_{ti}.time_{start} = e_1.time$ and $h_{ti}.time_{end} = e_k.time$.

Fig. 2 shows the task instances derived from Fig. 1 represented as grey rectangles, and all corresponding relationships. For example, ti_1 over e_1, e_2 and ti_4 over e_6, e_7, e_8 derived from Fig. 1 results in nodes $h1$ and $h4$ in Fig. 2, respectively. The *task variant* v described by a task instance ti over $e_1, ..., e_k$ is its sequence of action names along the DF-path in ti, set as property $h_{ti}.variant = e_1.action, ..., e_k.action$. For example, $h_{ti}.variant = A, B$. DF-edges lift from *Event* nodes to *TaskInstance* nodes. If $(h_{ti}, e), (h'_{ti}, e') \in CONTAINS$ and $(e, e')^n \in DF$, then add $(h_{ti}, h'_{ti}) \in DF$. Fig. 2 shows the lifted DF-edges, e.g., $(h1, h4)^r$.

Because subgraphs describing these task instances are formed by the DF-path of the case and the DF-path of the actor together, a task instance is inherently a piece of a case *and* an actor. Actor DF-paths over task instances show how actors perform work/execute tasks on different cases [3].

2) Aggregating Events into Event Classes: We also aggregate all events sharing the same property called *event class*, e.g., all events with $e.Action = A$. Picking a property name X as event classifier $class(e) = e.X$ defines the event classes $C_{class} = \{class(e) \mid e \in Event\}$, e.g., all values for "Action" in the data. For each value $v \in C_{class}$ create a new event class node cl with $cl.type = X$ and

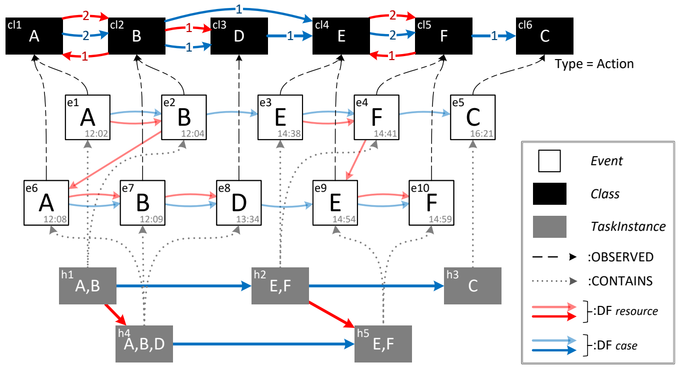


Fig. 2: Event graph containing the h_{ti} nodes (top) and cl nodes (bottom) and lifted DF relations built from Fig. 1 events.

$cl.id = v$, and for each $e \in Event$ with $class(e) = v$ add relationship $(e, cl) \in OBSERVED$, e.g., in Fig. 2 events e_1 and e_6 are observations of action A. We lift DF as before: if $(e, e')^{type} \in DF$ and $(e, cl), (e', cl') \in OBSERVED$, then add $(cl, cl')^{type} \in DF$. We set *count* properties to record how many events/DF-edges were aggregated.

Fig. 2 shows the event classes of the events in Fig. 1 defined by the type *Action* represented as black (rectangular) nodes, and all corresponding relationships. For example, $cl1$ aggregates events $e1$ and $e6$ and $(cl1, cl2)^r$ aggregates DF-relationships $(e1, e2)^r$ and $(e6, e7)^r$.

IV. Method

We assume data to be given in an event graph G including the aggregation of events into task instances. The BPIC'17 graph had 171 200 task instances of 1 208 task variants. Here, we describe methods we identified as effective to summarize tasks and actor behavior in such a graph. We discuss how to obtain task *clusters* (Sect. IV-A), how to discover an intra-task description of a task as an LPM (Sect. IV-B), and how to discover inter-task models of process and actor behavior by abstraction (Sect. IV-D) for various subsets (Sect. IV-C).

A. Clustering Task Variants

Our first problem to solve was to summarize the task instances to study the contents of the "units of work". We chose to cluster *TaskInstance* nodes by their $h_{ti}.variant$ feature (the sequence of actions performed). Our goal was to find a way to obtain homogeneous clusters in the sense that a human analyst could judge whether all instances in a cluster are variants of the same general task (by their judgment). Next we report on the feature engineering and an iterative clustering method to obtain homogeneous clusters.

Let $A = \{a_1, \ldots, a_n\} = \{e.action \mid e \in Event\}$ be all actions and $V = \{h_{ti}.variant \mid h_{ti} \in TaskInstance\}$ all task variants in G; note that $V \subseteq A^*$. We encode each $v \in V$ as feature vector in aggregate encoding: $enc(v) = \langle x_1, \ldots, x_n \rangle$ with x_i being how often a_i occurs in v.

For example, from h_{ti} nodes in Fig. 2 we retrieve $v_1 = \langle A, B \rangle$, $v_2 = \langle A, B, D \rangle$, $v_3 = \langle E, F \rangle$ and $v_4 = \langle C \rangle$. From v_2 we retrieve feature vector $enc(v_2) = \langle 1_{(A)}, 1_{(B)}, 0_{(C)}, 1_{(D)}, 0_{(E)}, 0_{(F)} \rangle$. Based on domain knowledge insights, more weight can be assigned to certain actions.

We cluster the set of task variants into N clusters through hierarchical agglomerative clustering [26]. We use ward as linkage criteria, i.e., the distance between clusters is the sum of squared distances within all clusters, and Eucledian distance as distance metric [26]. We first determine the optimal number of clusters using the silhouette coefficient sc [26], which is calculated from the mean intra-cluster distance and the mean nearest-cluster distance for each data point, between $sc = -1$ and $sc = +1$. To find a homogeneous clustering, we first compute clusterings for $N = 2, \ldots, max_N$ to identify N with maximal sc_N, where max_N depends on. An analyst can visually inspect each cluster for homogeneity (e.g., using intra-task DFGs, see IV-B) and explore clusterings for $N-1$ or $N+1$ until a desired qualitative homogeneity is reached.

The result of the clustering step is a set of N task clusters $tc_i \subseteq V, i = 1, \ldots, N$ identified by a cluster number $1 \leq i \leq N$; we assign each *TaskInstance* node its cluster number, i.e., $h_{ti}.cluster = i$ iff $h_{ti}.variant \in tc_i$. Clustering the task instances in Fig. 2 could result in $tc_1 = \{\langle A, B \rangle, \langle A, B, D \rangle\}, tc_2 = \{\langle E, F \rangle\}$ and $tc_3 = \{\langle C \rangle\}$. Each task instance is a sequence of events; thus all task instances in the same cluster i together form a "local log" of a task.

B. Deriving Intra-Task DFGs

Clustering groups similar task variants into a "local log" of a task, but does not summarize the variants to understand variations in task behavior (or to assess cluster homogeneity). A LPM [7] of the local task log would achieve this. Aiming for simple operations, we chose to build a basic directly-follows graph as a model of behavior in a task cluster, as follows.

We adapt the event-to-class aggregation (see Sect. III-C2) to be local to a cluster. As the same action may occur in different clusters, we aggregate events per cluster with the event classifier $class(e) = (e.action, cluster(e))$ with $cluster(e) = i$ iff $(e, h_{ti}) \in CONTAINS, h_{ti}.cluster = i$. Further, we aggregate $(e, e')^n \in DF$ to $(cl, cl')^n \in DF$ only if $cl = (a, i), cl' = (b, i)$ belong to the same cluster i.

Fig. 3 (bottom) shows how events $e1, e2, e6, e7, e8$ are aggregated to the intra-task DFG of cluster 1, thereby describing the local behavior of a task in one model. In Sect. IV-D we discuss how to query the complete DFG including start/end nodes from the aggregation result. Analysts can use task DFGs to understand task contents (e.g., overlap with others) and homogeneity of clusters.

C. Summarizing Behavior over Abstracted Tasks

The LPMs of tasks obtained in Sect. IV-B are not disjoint as the same actions may occur in multiple tasks. Thus, the collection of LPMs of tasks is not a classical hierarchical abstraction of process control-flow. To understand whether a task cluster is a valid abstraction also regarding the control-flow, we had to study the context of all task instances in a cluster. Our basic idea was to abstract each cluster (each LPM)

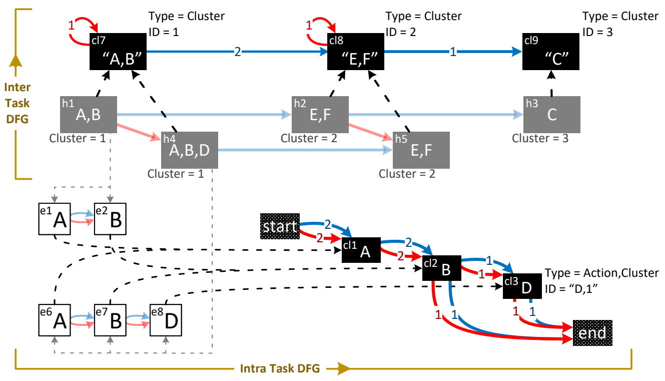


Fig. 3: Task instances aggregated into task classes for deriving inter-task DFGs (top). Subset of lower-level events aggregated into event classes for deriving intra-task DFGs (bottom).

of a task into a single "task node" and derive a model of case and actor behavior over these task nodes.

We adopted event-to-class aggregation (Sect. III-C2) and aggregate the *TaskInstance* nodes into task instance classes using $class(h_{ti}) = h_{ti}.cluster$ as classifier. Applying this operation on our running example (see Fig. 3 top) creates 3 class nodes $cl7, cl8, cl9$ of type "cluster" with ID "1", "2", and "3" respectively, adds OBSERVED-edges from each h_{ti} node to their *Class* node and aggregates all DF-edges from *TaskInstance* to *Class* nodes (recording frequency of aggregated DF-edges as property). For instance, $cl8$ aggregates task instance $h2$ ad $h5$. Note that this operation preserves for which entity-type (case or resource) a DF-relationship holds: e.g., $(cl8, cl8)^r$ aggregates $(h2, h5)^r$ and $(cl7, cl8)^c$ aggregates $(h1, h2)^c$, $(h4, h5)^c$. We also assign each *Class* node cl a *name* being the set of all actions occurring in all h_{ti} nodes of the cluster, e.g., $cl7.name =$ "A, B". The analyst can modify this name. This results in a "global DFG" on the level of tasks.

Note that this operation also aggregates the resource DF-relationships with regard to tasks. We show in Sect. V that this gives new insights into actor behavior.

D. Deriving Inter-Task DFGs

The aggregation of Sect. IV-C results in a complex graph over two behavioral dimensions that is difficult to visualize and possibly not specific to answer an analysis question. To obtain more specific DFGs, we identified the following parameters: (1) **node aggregation** by using more specific classifiers for *TaskInstance* nodes, (2) **filtering** by using different criteria to decide which *TaskInstance* nodes to keep and (3) **edge aggregation** by selecting which DF-edges to aggregate. Each parameter is defined by the properties of the entire event knowledge graph, including the underlying events.

(1) We can refine the aggregation of *TaskInstance* nodes to *Class* nodes using a classifier over multiple properties. For example, the following classifier distinguishes tasks per actor: $class_{T\times R}(h) = (h.cluster, resource(h))$ with $resource(h) = a$ iff $(e, h) \in CONTAINS, e.resource = a$. The DF-relationships are then aggregated per actor, allowing to compare different actors wrt. their behavior over tasks.

(2) To obtain a DFG for specific parts of the data, we define a subset $TI' \subseteq TaskInstance$ that satisfy some property wrt. the underlying data and only aggregate the nodes $h_{ti} \in TI'$. For instance: (1) only h_{ti} nodes correlated to an entity based on a specific property, e.g., in Fig. 2, related to resource entities where $n.ID = a5$, i.e., $h2$ and $h5$, or case entities where $n.item\_category =$ Electronics, i.e., $h1$, $h2$ and $h3$; or (2) based on temporal properties, e.g., only h_{ti} nodes in cases that end before 15:00, i.e., $(h_{ti}, e) \in CONTAINS, (e, n) \in CORR, n.type = case$ and all events $(e', n) \in CORR$ have $e'.time < 15:00$.

(3) We can limit the DF-relationships to aggregate to a subset $DF' \subseteq DF$ determined by structural or temporal properties in the same way as in (2). Note that if DF' is chosen independent of TI' there may be no aggregated DF-edges between *Class* nodes.

After choosing parameters, we set the *frequency* property of the aggregated Class and DF-edges based on TI' and DF'. For visualizing the DFG, we query the *Class* nodes (of a particular type) and all DF-relationships between them. We identify how often a *Class* node cl is a start node of the DFG (for entity type n) by querying the number of $h_{ti} \in TI'$ nodes with $(h_{ti}, cl) \in OBSERVED$ and no incoming DF-edge $(h'_{ti}, h_{ti})^n \in DF'$. For example, in Fig. 3, $cl7$ is start node once for r and twice for c. Correspondingly for end nodes. We visualize this as edges of the artificial start/end nodes.

We applied the above concepts to derive DFGs over task variants occurring at least 10 times for case DF-edges only (Fig. 4) and for various subsets (Fig. 8). Further, we constructed a composition of multiple actor DFGs inter-connected by cases as follows: (1) classifier $class_{T\times R}(h)$ defined above, i.e., create cluster nodes per actor, (2) TI' contains only *TaskInstance* nodes related to one of three specific resources, (3) include a resource DF-edge $(h, h')^r \in DF'$ only if $h.time_{end}$ and $h'.time_{start}$ occur on the same day, and include any case DF-edge in DF'. Fig. 9 shows an example of such time filtering; the resulting DFG summarizes for each actor the behavior executed over a day (no DF-edges to a task on the next day) and the aggregated case DF-edges show how often an actor handed a case from one task to another actor with another task. The complete results are described in Sect. V.

V. Results

To test our hypothesis whether Q1-Q4 can be answered from process event data, we applied the operations we identified in Sect. IV on the event knowledge graph [6] of the BPIC'17 data [5] in our case study. All the steps in Sect. IV could be implemented in Cypher queries invoked via parameterized Python scripts[1] on the graph database Neo4j. Naive queries took 1.5h to build the graph and about 1m per DFG on an Intel i7 CPU @ 2.2GHz machine with 32GB RAM.

[1]Available at: https://zenodo.org/record/6727896#.YrYcjXZBwuU

TABLE II: Clusters (tasks of Fig. 4) and actions (name+lifecycle*) common to its task variants.

C0	A_Accept, O_Create, O_Sent, W_Compl appl+E, W_Call offers+S, A_Complete (nr. of variants: 43)
C1	A_Create, A_Concept, W_Compl appl+S (3)
C2	A_Create, A_Submit, W_Handle Lds+S (3)
C3	*infrequent (count $<$ 30) (2)*
C4	O_Accept, A_Pending, W_Call inc+E/W_Validate+E (27)
C5	A_Create, A_Concept, W_Compl appl+S, A_Accept, O_Create, W_Call offers+S, A_Complete (21)
C6	*leftovers (12)*
C7	A_Denied, O_Refused, W_Call inc+E/W_Validate+E/ W_Call off+E (15)
C8	W_Call inc+E/W_Call offers+E, W_Validate+S, A_Validating, W_Validate+E, W_Call inc+S, A_Incomplete (25)
C9	O_Create (18)
C10	A_Create, W_Compl appl+S, A_Concept, A_Accept, O_Create, O_Sent, W_Compl appl+S, W_Call offers+S, A_Complete, W_Call offers+E, W_Validate+S, A_Validating, W_Validate+E, W_Call inc+S, A_Incomplete (3)
C11	W_Validate+E, W_Call inc+S, A_Incomplete (7)
C12	A_Cancel, O_Cancel, W_Call off+E/W_Call inc+E/ W_Validate+E (25)
C13	W_Handle Lds+E, W_Compl appl+S, A_Concept (7)
C14	W_Call off+E/W_Call inc+E, W_Validate+S, A_Validating (10)
C15	W_Call inc+E, W_Validate+S, A_Validating, O_Accept, A_Pending, W_Validate+E (3)
C16	A_Create, W_Compl appl+S, A_Concept, A_Accept (4)
C17	W_Handle Lds+E, W_Compl appl+S, A_Concept, A_Accept, O_Create, O_Sent, W_Compl appl+E, W_Call offers+S, A_Complete (2)
C18	A_Accept, O_Create (4)
C19	(O_Create,) O_Sent, W_Complete appl+E, W_Call off+S, A_Complete (11)

* S (abbreviation for schedule, start) and E (withdraw, abort, complete).

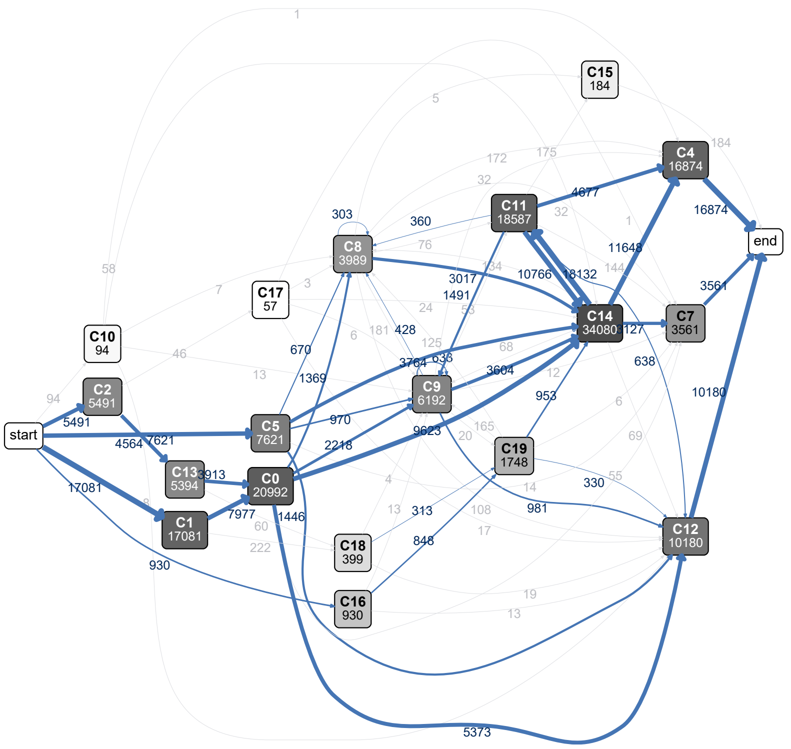


Fig. 4: Inter-task DFG with edge thickness denoting frequency.

The BPIC'17 dataset is a large, real-life event log describing the process of clients applying for a loan at a bank (application documents) and receiving back loan offers (offer documents). We iteratively explored and summarized the data using our implementation with different parameters to identify relevant subsets for answering Q1-Q4.

To identify tasks by clustering (Sect. IV-A) we first removed all *TaskInstance* nodes describing variants occurring < 10 times (1%) and of length $= 1$ (6%). For clustering, we ended up giving A_ actions (application documents) a weight of 2 in feature encoding; we found $N = 19$ clusters to yield the best $sc = 0.66$, but adjusted to $N = 20$ ($sc = 0.63$) after visual inspection of intra-task DFGs of clusters (Sect. IV-B) to further split variants describing creation of offer documents and leftover variants. Tab. II shows the name and life-cycle transition of actions occurring in each variant of a cluster.

To get an overview of the complete process, we first derived an inter-task DFG (Sect. IV-D) of all task clusters except C3 (very infrequent) and C6 (leftovers) from the case perspective (Sect. IV-C) as shown in Fig. 4. Note that frequencies reported are all specific to the (filtered) subset used in the analysis. This inter-task DFG, reveals that the process is quite structured from a task perspective. The variants within a cluster are largely performed at the same spot in the process (Q1), confirming that the conceptualization of task variant clusters as higher-level tasks is meaningful.

Table II shows how actors divide work along a process execution (Q1). We observe tasks that are completely disjoint C4, C7, and C12 corresponding to three distinct ways of ending the process, but also tasks with overlapping sets of actions, e.g., *there are five ways of starting the process* (C1, C2, C5, C10 and C16) and a task (C8) that is the *composite of two other tasks* (C11, C14). Task C14 is by far the most frequently executed task (34 080 times as visible in Fig. 4).

To investigate the reason for 5 alternative tasks to start a case (Q2), we analyzed which case and actor attributes correlated with the choice C1, C2, C5, C10, C16. We only found correlations for C2 (by automated resource) and for C10 (correlated to specific case attributes), and show details in App A [24]. This suggests that the choice for starting with C1, C5 or C16 is actor-driven. The choice of the first task impacts performance (Q4); we find that cases starting with C1, C2, C5, C10 or C16 have (statistically significant) different case durations, with average durations 22.4, 24.2, 19.7, 13.6 and 18.9 days, respectively (details in App A [24]).

To investigate how work is structured within a task (Q1) we created the intra-task DFG (Sect. IV-B) in Fig. 5 showing the local behavior of C14; variants V5 and V6 are analyzed further later. We observe frequent and infrequent task variants, as well as a variability in alternative actions, e.g., choice between W_Call. off.-A and W_Call. inc.-A. In other clusters, we additionally observe a variability in the order of actions (same set of actions but ordered differently), e.g., the switching around of $\langle$A_Conc+Comp$\rangle$ and $\langle$W_Compl app+Sched,W_Compl app+Start$\rangle$ in C1, C5 and C16.

Next, we analyzed for changes in the organization of work.

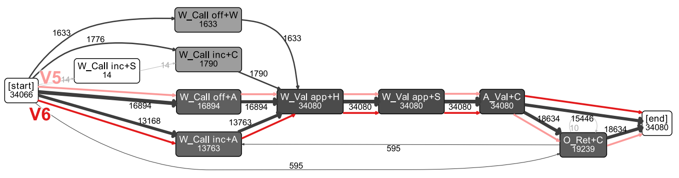


Fig. 5: Intra-task DFG of C14 with variants V5 and V6.

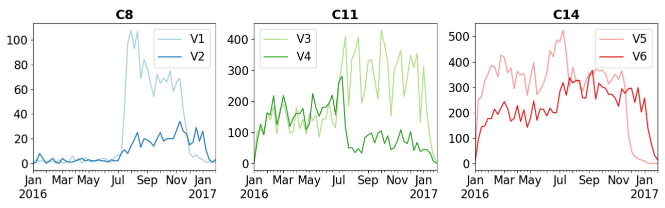


Fig. 6: Frequency of variants in clusters show concept drift.

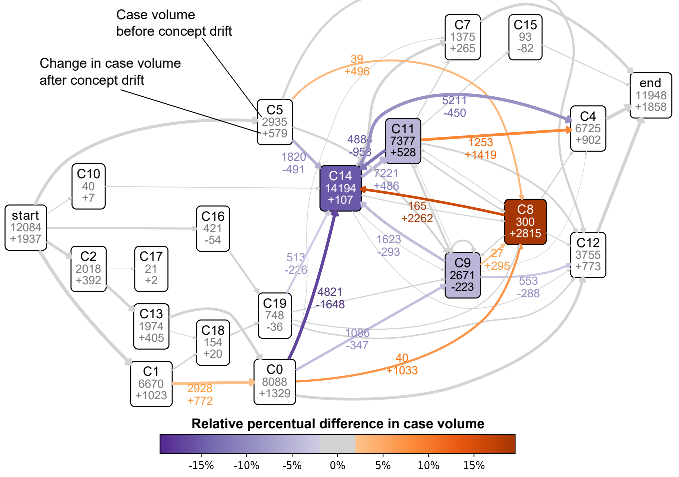


Fig. 8: Inter-task DFG showing (relative percentual) change in case volume before and after concept drift.

We focused on clusters C8, C11 and C14 as C8 combines actions of C11 and C14. We first analyzed the frequency of individual task variants in C8, C11, C14 over time, revealing concept drift. In Fig. 6, we observe the introduction of new task variants (C8, in July), a changeover (C11, in July), and drops (variant V1 in Nov., V4 in July, V5 in Nov.). We observe similar changes in other clusters as well as spikes related to the drops in November (details in App B [24]). The shifts we observe within clusters are globally connected, i.e., certain ways of doing a task are dropped or replaced by variants in other tasks (Q3). Such shifts on a global level suggest a process-driven change (Q2).

We analyzed the frequency of performing V3 and V4 per actor and find that different actors adopt to these changes differently. Fig. 7 shows actors adapting to the concept drift in variants V3 and V4 in a non-uniform way. In Fig. 7, we observe actors adapting with concept drift (users U29 & U90), not at all (U100) and against concept drift (U68). Despite the global change over the entire process, actors have their own preferences (Q2). We see similar adoption behavior for variants in some other tasks (details in App B [24]).

We computed an inter-task DFG of all cases ending before July and one for all cases starting after July (excluding July as transition period); App B [24] shows both DFGs, Fig. 8 shows their difference in case volume before and after concept drift highlighted on the nodes and edges on the inter-task DFG. In

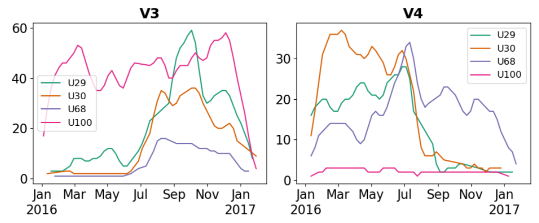


Fig. 7: Four actors adopting to concept drift in variants of C11.

most tasks in Fig. 8, we observe a workload increase that is in line with the overall increase of cases (+1937) at the start, except for C14 and C11: the absolute load of C14 is nearly the same (decrease 15.3%), the load of C11 increases less (decrease 4.5%) than overall. Interestingly, C8 (combines C11 and C14) increases load with an relative increase of 19.5%. We observe a shift in flow over tasks: inflow in C8 increased by 7.3% (from C0) and 3.5% (from C5) while incoming edges in C14 decreased by -17% (from C0) and -6% (from C5). More cases are routes from C8 to C14 (16%). While rework (forth-and-back between C11 and C14) reduced by -12% because more cases go directly from C11 to C4 (+9%). This suggests that the growth in cases at the start (+1937) is absorbed by the combined task C8, while C11 and C14 stay nearly constant. The increased workload is now routed through a different path.

We also investigated whether the process changes had an effect on case duration and outcome (Q4). We found that separate tasks C11 and C14 previously performed by different actors are now combined into a single task C8 done by a single actor. We compared two new task variants V1 and V2 in C8 that are combinations of already existing variants in C11 (V4) and in C14 (V5, V6). We compared the case duration for the new variants and found that the change led to 5% (V1) and 13% (V2) faster processing and overall 11% more declined applications. More details are provided in App C [24].

We, finally, investigated whether the concept drift changes the behavior of an actor (Q3). Fig. 9 shows the resource-specific DFGs and handovers of work between them before (top) and after concept drift (bottom), (Sect. IV-D). In Fig. 9, the count in the start node represents the user's working days in the data. U113 now works more on C8 and less on C14 (in line with earlier results). Surprisingly, U29 executes C8 less, suggesting that this has become a main task of other users (e.g., U113). Also, the work is more structured after concept drift: the number of different handovers between U29

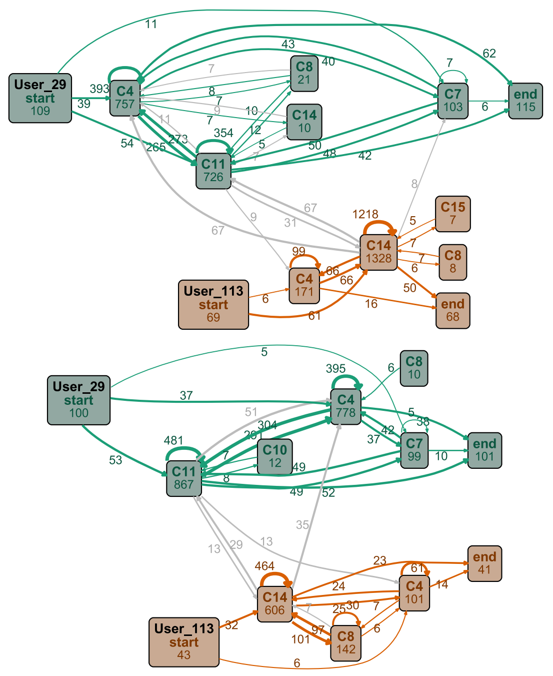


Fig. 9: Inter-task DFGs showing behavior of users 29 (green) and 113 (orange) interconnected by handovers (grey) before (top) and after concept drift (bottom).

and U113 decreases from 5 to 4. The handovers from (U29, C11) to (U114, C14) decrease stronger than the load in (U29, C11). Also, for U29 the amount of self-handovers increases (from 34 to 51) but the type of handovers decreased (from 4 to 1), suggesting a more structured way of working.

VI. Conclusion

Through our exploratory case study we showed that graph-based process mining can help answer fundamental questions in routines research. We modeled actor and case behavior jointly and showed how to extend the used event knowledge graph through clustering, aggregation, and querying operations to obtain local models of a task and the global behavior across tasks to answer specific analysis questions – considering all underlying case and actor properties and behavior.

By analyzing the BPIC'17 event data in this way, we were able to answer questions Q1-Q4: we found how work is structured and divided among actors (Q1) is different within and across different parts of the process, (Q2) is affected by process instance properties and participating actors, (Q3) is subject to concept drift, and (Q4) impacts performance. Doing so gives new insights on BPIC'17. We believe that such analysis of event data can give new insights into actor behavior in the context of routines [2], [4], organizational models [12], batching [16], concept drift analysis [21] and more.

A necessary next step is to overcome the limitation of this exploratory case study: validation of the techniques on further cases including feedback from domain experts and to gain requirements for automating steps, specifically clustering and identifying task clusters and actors of interest for analysis.

References

[1] W. M. P. van der Aalst, *Process Mining: Data Science in Action*. Springer, 2016.

[2] B. Pentland, M. Feldman, M. Becker, and P. Liu, "Dynamics of organizational routines: A generative model," *J. of Mngmt. Studies*, vol. 49, pp. 1484–1508, 12 2012.

[3] E. L. Klijn, F. Mannhardt, and D. Fahland, "Classifying and detecting task executions and routines in processes using event graphs," in *BPM Forum*, vol. 427 of *LNBIP*, pp. 212–229, Springer, 2021.

[4] K. Goh and B. Pentland, "From actions to paths to patterning: Toward a dynamic theory of patterning in routines," *Acad. Manage. J.*, vol. 62, pp. 1901–1929, 12 2019.

[5] B. F. van Dongen, "BPI Challenge 2017." Dataset, 2017. https://doi.org/10.4121/12705737.v2.

[6] S. Esser and D. Fahland, "Multi-dimensional event data in graph databases," *J. Data Semant.*, vol. 10, p. 109–141, 2021.

[7] M. Brunings, D. Fahland, and B. van Dongen, "Defining meaningful local process models," in *TOPNOC XVI*, vol. 13220 of *LNCS*, pp. 24–48, Springer, 2022.

[8] C. Cabanillas, L. Ackermann, S. Schönig, C. Sturm, and J. Mendling, "The ralph miner for automated discovery and verification of resource-aware process models," *Softw. Syst. Model.*, vol. 19, no. 6, pp. 1415–1441, 2020.

[9] A. Pika, M. Leyer, M. T. Wynn, C. J. Fidge, A. H. M. ter Hofstede, and W. M. P. van der Aalst, "Mining resource profiles from event logs," *ACM Trans. Manag. Inf. Syst.*, vol. 8, no. 1, pp. 1–30, 2017.

[10] N. Martin, B. Depaire, A. Caris, and D. Schepers, "Retrieving the resource availability calendars of a process from an event log," *Inf. Syst.*, vol. 88, 2020.

[11] A. Kumar and S. Liu, "Analyzing a helpdesk process through the lens of actor handoff patterns," in *BPM (Forum)*, vol. 392 of *LNBIP*, pp. 313–329, Springer, 2020.

[12] J. Yang, C. Ouyang, W. M. P. van der Aalst, A. H. M. ter Hofstede, and Y. Yu, "*OrdinoR*: A framework for discovering, evaluating, and analyzing organizational models using event logs," *Decis. Support Syst.*, vol. 158, p. 113771, 2022.

[13] G. van Hulzen, N. Martin, and B. Depaire, "Looking beyond activity labels: Mining context-aware resource profiles using activity instance archetypes," in *BPM (Forum)*, vol. 427 of *LNBIP*, pp. 230–245, 2021.

[14] L. Delcoucq, T. Dupiereux-Fettweis, F. Lecron, and P. Fortemps, "Resource and activity clustering based on a hierarchical cell formation algorithm," *Applied Intelligence*, 2022.

[15] L. Delcoucq, F. Lecron, P. Fortemps, and W. M. P. van der Aalst, "Resource-centric process mining: clustering using local process models," in *SAC 2020*, pp. 45–52, ACM, 2020.

[16] N. Martin, L. Pufahl, and F. Mannhardt, "Detection of batch activities from event logs," *Inf. Syst.*, vol. 95, p. 101642, 2021.

[17] A. Pika, C. Ouyang, and A. H. M. ter Hofstede, "Configurable batch-processing discovery from event logs," *ACM Trans. Manage. Inf. Syst.*, vol. 13, no. 3, 2022.

[18] A. Senderovich, A. Rogge-Solti, A. Gal, J. Mendling, and A. Mandelbaum, "The ROAD from sensor data to process instances via interaction mining," in *CAiSE*, vol. 9694 of *LNCS*, pp. 257–273, Springer, 2016.

[19] M. de Leoni and S. Dündar, "Event-log abstraction using batch session identification and clustering," in *SAC 2020*, pp. 36–44, ACM, 2020.

[20] H. Nguyen, M. Dumas, M. L. Rosa, and A. H. M. ter Hofstede, "Multi-perspective comparison of business process variants based on event logs," in *ER*, vol. 11157 of *LNCS*, pp. 449–459, Springer, 2018.

[21] J. N. Adams, S. J. van Zelst, L. Quack, K. Hausmann, W. M. P. van der Aalst, and T. Rose, "A framework for explainable concept drift detection in process mining," in *BPM*, vol. 12875 of *LNCS*, pp. 400–416, 2021.

[22] S. Agostinelli, F. Leotta, and A. Marrella, "Interactive segmentation of user interface logs," in *ICSOC*, vol. 13121 of *LNCS*, pp. 65–80, 2021.

[23] V. Leno, A. Augusto, M. Dumas, M. L. Rosa, F. M. Maggi, and A. Polyvyanyy, "Discovering data transfer routines from user interaction logs," *Inf. Syst.*, vol. 107, p. 101916, 2022.

[24] *Analyzing Actor Behavior in Process Executions*, Zenodo, https://doi.org/10.5281/zenodo.6719505, July 2021.

[25] A. Bonifati, G. H. L. Fletcher, H. Voigt, and N. Yakovets, *Querying Graphs*. Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2018.

[26] C. C. Aggarwal, *Data Mining - The Textbook*. Springer, 2015.

Appendix

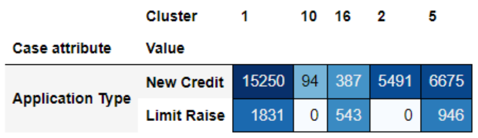
A. *Analysis of Alternative Tasks to Start a Case.*

This subsection contains additional analysis results related to the analysis of the choice of alternate tasks to start a case in Sect. V. Figs. 10 to 12 compare case attribute values for three case attributes across cases that start the process in a different task. Fig. 13 compares actor distribution across the different tasks that start the process.

In Figs. 10 and 12 we observe that C2 is correlated to application type "New Credit", requested amounts > 0 and the system resource (resource $= 1$). C10 is correlated to application type "New Credit", loan goal "Unknown" and requested amount $= 0$. C10 is also executed by a different set of resources than the other case starts C1, C5, C16 are. In Fig. III we observe that the choice for a specific case start affects performance; case durations across the different case starts are significantly differently distributed, except C5 vs C16 with p=0.052.

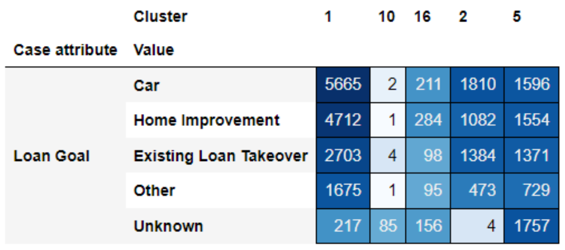
TABLE III: Average case duration for disjoint subsets of traces in BPIC'17 that all start the process in a different task cluster C1, C2, C5, C10 or C16.

Cluster	Average case duration (days)	P-value			
		C2	C5	C10	C16
C1	22.4	1.28e-18	8.75e-51	4.09e-11	4.09e-11
C2	24.2		6.19e-82	1.54e-14	2.16e-16
C5	19.7			4.03e-6	0.052
C10	13.6				3.48e-5
C16	18.9				



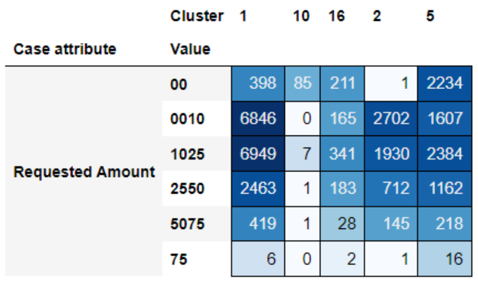
Case attribute	Cluster / Value	1	10	16	2	5
Application Type	New Credit	15250	94	387	5491	6675
	Limit Raise	1831	0	543	0	946

Fig. 10: Heatmap comparing the case attribute Application Type across traces that all start the process in a different task cluster C1, C2, C5, C10 or C16.



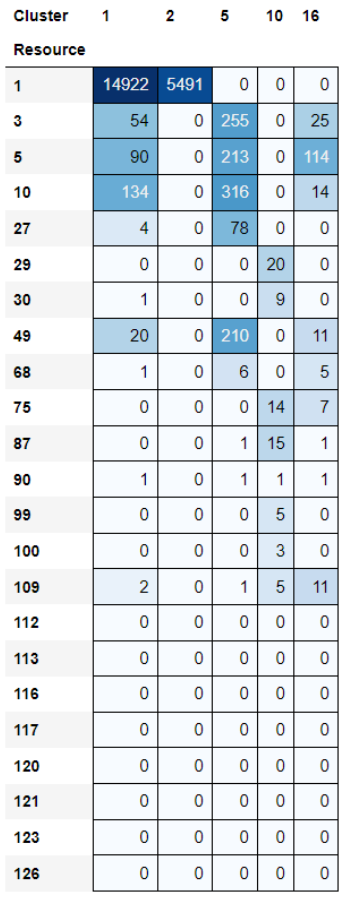
Case attribute	Cluster / Value	1	10	16	2	5
Loan Goal	Car	5665	2	211	1810	1596
	Home Improvement	4712	1	284	1082	1554
	Existing Loan Takeover	2703	4	98	1384	1371
	Other	1675	1	95	473	729
	Unknown	217	85	156	4	1757

Fig. 11: Heatmap comparing the case attribute Loan Goal across traces that all start the process in a different task cluster C1, C2, C5, C10 or C16.



Case attribute	Cluster / Value	1	10	16	2	5
Requested Amount	00	398	85	211	1	2234
	0010	6846	0	165	2702	1607
	1025	6949	7	341	1930	2384
	2550	2463	1	183	712	1162
	5075	419	1	28	145	218
	75	6	0	2	1	16

Fig. 12: Heatmap comparing the case attribute Requested Amount across traces that all start the process in a different task cluster C1, C2, C5, C10 or C16.



Resource \ Cluster	1	2	5	10	16
1	14922	5491	0	0	0
3	54	0	255	0	25
5	90	0	213	0	114
10	134	0	316	0	14
27	4	0	78	0	0
29	0	0	0	20	0
30	1	0	0	9	0
49	20	0	210	0	11
68	1	0	6	0	5
75	0	0	0	14	7
87	0	0	1	15	1
90	1	0	1	1	1
99	0	0	0	5	0
100	0	0	0	3	0
109	2	0	1	5	11
112	0	0	0	0	0
113	0	0	0	0	0
116	0	0	0	0	0
117	0	0	0	0	0
120	0	0	0	0	0
121	0	0	0	0	0
123	0	0	0	0	0
126	0	0	0	0	0

Fig. 13: Heatmap comparing the distribution of (20 most active) resources executing the different process starts C1, C2, C5, C10 or C16.

B. *Concept Drift Analysis.*

This subsection contains additional analysis results related to the concept drift analysis in Sect. V. Fig. 14 shows the trends of 6 additional task variants showing similar changes as observed in Fig. 6 (Sect. V): V7 and V8 show a similar changeover as V3 and V4, V9 and V10 show a different changeover in November, and V11 and V12 show peaks that fall together with the drops of V1 and V5. In Fig. 15 we see trends of four actors showing adaptation rates similar to their adaptation rates shown in Fig. 7 (Sect. V).

Figs. 16 and 17 show the inter-task DFGS (of cases before and after concept drift, respectively) from which the DFG in Fig. 8 (Sect. V is computed.

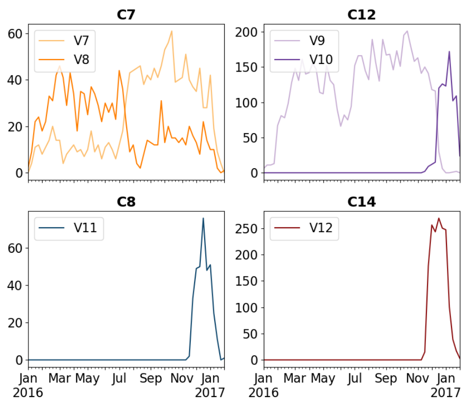


Fig. 14: Trends of 6 additional task variants in clusters 7, 12, 8 and 14 in BPIC'17 showing concept drift in July and November.

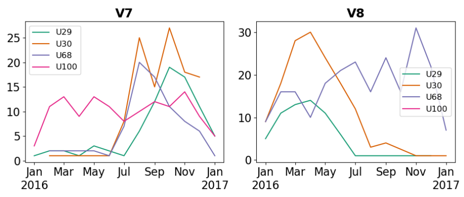


Fig. 15: Trends of four actors adopting to concept drift (July) in task variants 7 and 8 (C7) in different ways in BPIC'17.

C. *Performance analysis of new task adoption.*

This subsection contains additional results related to the performance analysis of a change in the way tasks are executed after concept drift (see Sect. V). Tab. IV shows average case durations and distributions of case outcome for subsets of cases executed in the old way, and subsets of cases executed according to the change.

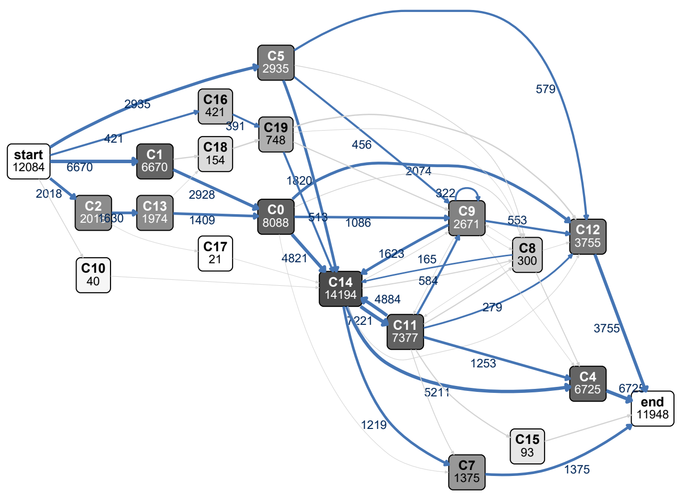


Fig. 16: Inter-task DFG of all cases ending before concept drift in BPIC'17.

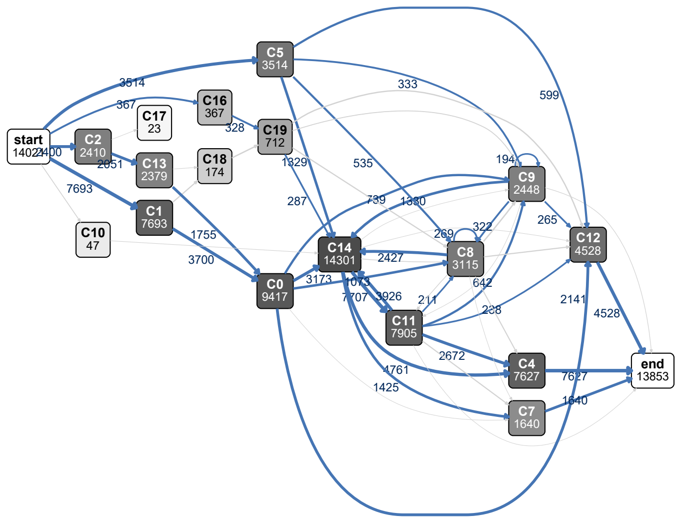


Fig. 17: Inter-task DFG of all cases starting after concept drift in BPIC'17.

TABLE IV: Average case duration and distribution of case outcome for subsets of BPIC'17 traces describing a larger piece of work that is split in different combinations of task variants, and for all traces in BPIC'17.

Subset with work division	Average case duration (days)	P-value	Accepted	Denied	Cancelled
$S1_{old}$: V5, V4, V6 and not V1	23.4	0.03	87.8 %	8.7 %	3.5 %
$S1_{new}$: V1, V6	22.3		76.6 %	19.2 %	4.2 %
$S2_{old}$: V6, V4, V6 and not V2	28.9	0.0004	92.7 %	5.8 %	1.6 %
$S2_{new}$: V2, V6	25.2		82.2 %	16.1 %	1.7 %
All cases	21.9	-	55.1 %	11.6 %	33.3 %