# No Time to Dice: Learning Execution Contexts from Event Logs for Resource-Oriented Process Mining

Jing Yang[1 [0000−0001−9218−6954]], Chun Ouyang[1 [0000−0001−7098−5480]], Arthur H.M. ter Hofstede[1 [0000−0002−2730−0201]], and Wil M.P. van der Aalst[2,1 [0000−0002−0955−6940]]

[1] Queensland University of Technology, Brisbane, Australia
[2] RWTH Aachen University, Aachen, Germany

**Abstract.** Process mining enables extracting insights into human resources working in business processes and supports employee management and process improvement. Often, resources from the same organizational group exhibit similar characteristics in process execution, e.g., executing the same set of process activities or participating in the same types of cases. This is a natural consequence of division of labor in organizations. These characteristics can be organized along various process dimensions, e.g., case, activity, and time, which ideally are all considered in the application of resource-oriented process mining, especially analytics of resource groups and their behavior. In this paper, we use the concept of execution context to classify cases, activities, and times to enable a precise characterization of resource groups. We propose an approach to automatically learning execution contexts from process execution data recorded in event logs, incorporating domain knowledge and discriminative information embedded in data. Evaluation using real-life event log data demonstrates the usefulness of our approach.

**Keywords:** execution context · resource group · event log · process mining · workforce analytics

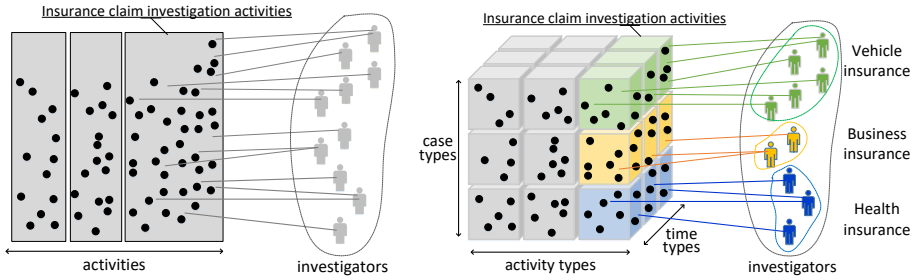
## 1 Introduction

The success of an organization depends on how well its workforce is mobilized and managed [4]. Modern organizations deploy their employees in business processes [5] to deliver on products and services. In order to streamline business processes and to improve employee satisfaction of work [5], process managers need to be able to analyze and understand employee behavior and performance in process execution [8].

Process mining can discover employee-related insights from event log data [14] to support workforce analytics in the context of business processes. Often, employees (human resources) from the same organizational group (such as role, team, department, etc.) share similar characteristics in process execution [12], e.g., resources of the same role are in charge of a subset of process activities.

These characteristics may transcend the clustering of activities and manifest in other dimensions including case and time, e.g., resource group taking part in a specific type of cases, working on time shifts, or being in different locations. The connection between resource groupings and groups' characteristics in process execution is a natural consequence of the *specialization* of work, i.e., division of labor in an organization [4]. Some existing work on resource-oriented process mining considers such characteristics [12,9,8] but treats different dimensions separately. Only few [15,17] has considered the characterization of resource groups across various process dimensions holistically.

*Execution context* [17] is a concept proposed to enable a precise characterization of resource groups, considering various process dimensions. A set of execution contexts is defined by classifying and combining *case types*, *activity types*, and *time types*. Given an event log, applying execution contexts creates a multidimensional view on event data, where subsets of the log can be linked to the signature behavior of resource groups for analyses [16,17]. Fig. 1 illustrates the idea. In an insurance company, there are investigators specialized in different types of insurance claims, such as vehicles, business, and health insurance. While these investigators may perform the same field investigation activities in a claim handling process, they are likely to participate in only the type of cases where their expertise fits. If we use merely the activity dimension to characterize resources (Fig. 1a), we will not be able to discover or analyze investigators with specialty that manifests on other process dimensions. But, when we apply execution contexts to view events from multiple dimensions (Fig. 1b), more precise characterization and more dedicated analyses on investigators can be performed.




(a) link group(s) to activities only (b) link group(s) to execution contexts

Fig. 1: Subsets of events (dots) in a log can be linked to groups of resources as a natural consequence of the specialization of work

Our previous work [17] has shown how to manually define execution contexts using prior knowledge about an event log. However, it assumes the availability of relevant domain knowledge. In this paper, we propose an approach that

supports *learning* execution contexts from an event log, by exploiting the discriminative information of events embedded in the data rather than relying on domain knowledge. Our approach is built on a customized decision learning algorithm and is capable of deriving categorization rules that can be used to dice event log data to obtain execution contexts. We demonstrate the usefulness of our approach through experiments conducted on real-life event logs.

Our research contributes a solution to the problem of learning execution contexts, thus enhances resource-oriented process mining techniques that focus on analyzing human resources [8] and their groups [12,16,17]. Our work also contributes a method to derive process cube views in multidimensional process mining research [13,2]. Process mining techniques can then be applied to selected sublogs to analyze process variants concerned with certain groups of resources.

## 2 Related Work

Process mining can be applied on event logs to extract insights about human resources participating in process execution. For many resource-oriented process mining topics, an essential step is to identify the behavioral characteristics specific to resources or groups of resources in process execution. Then, analytics can be conducted on resources for different purposes, e.g., mining resource profiles [8] and mining organizational models [12,1,17]. To achieve that characterization, it is common to consider many process dimensions, e.g., activity [12,8,1], case [12,8], time [8], and location [9].

However, for organizational model mining, much of the literature has not yet employed a holistic view on those various dimensions. Instead, existing work exploits each dimension separately by modeling how resources perform different activities (e.g., [1]), how they hand over between activities (e.g., [3]), or how they participate in the same cases (e.g., [12]). This poses a challenge of mining complex resource groupings where resource characteristics are concerned with multiple dimensions, e.g., employees with the same business role but working different shifts. To address the challenge, we need an approach that jointly exploits multidimensional event log information.

Some recent research on organizational model mining [15,17] has contributed to addressing that literature gap. Van Hulzen et al. [15] propose the notion of "activity instance archetype" to capture contextual factors impacting how activity instances were executed. Activity instance archetypes can be discovered by applying model-based clustering on events enriched with selected attributes. Then, resources are characterized by their execution of activity instance archetypes, and resource groupings concerned with contextual factors can be discovered. In our previous work [17], we propose the notion of "execution context". Execution contexts are built upon categorizing cases, process activities, and time periods. Different from activity instance archetypes [15] (which focus on the homogeneity of activity instances), execution contexts aim at finding a structured, multidimensional way to organize event log data.

Execution contexts can be derived from an event log based on domain knowledge about existing categorization. Resources are characterized by their participation in execution contexts. Then, resource groupings can be discovered and analyzed using event logs, and their conformance can be checked with respect to the logs. However, it remains a problem how such execution contexts can be derived from event log data without relying on available domain knowledge about a process and the event data.

In this paper, we extend our previous work [17] and explore how to automatically learn execution contexts from a given event log. Outside resource-oriented process mining, our work is relevant to the research on multidimensional process mining. A multidimensional data model named event cube was proposed [10] to allow exploiting and integrating different aspects of business process and providing various levels of abstraction on process data to improve business analysis. The notion of process cube [13,2] was later proposed, which provides a more comprehensive view to organize both process models and their data using different dimensions. Process cubes support OLAP (Online Analytical Processing)-like operations dedicated to process mining, therefore enable decomposing large event logs into smaller sublogs to enhance process mining performance and scalability. Our idea of execution contexts resembles that of process cubes regarding the consideration of multidimensionality. As such, our approach to learning execution contexts can be seen as a way to constructing process cube views dedicated to resource-oriented process mining.

## 3 Preliminaries

A business process consists of tasks conducted in an organization to achieve a business goal [5]. An instance of executing a process is a case [14]. An *event log* (Def. 1) is a set of timestamped events recording how (human) resources performed those tasks (i.e., process activities) within different cases.

**Definition 1 (Event Log).** *Let $\mathcal{E}$ be the universe of event identifiers, $\mathcal{U}_{Att}$ be the universe of possible attribute names, and $\mathcal{U}_{Val}$ be the universe of possible attribute values. $EL = (E, Att, \pi)$ with $E \subseteq \mathcal{E}$, $E \neq \varnothing$, $Att \subseteq \mathcal{U}_{Att}$, and $\pi \colon E \to (Att \not\to \mathcal{U}_{Val})$ is an event log. Event $e \in E$ has attributes $dom(\pi(e))$. For an attribute $x \in dom(\pi(e))$, $\pi_x(e) = \pi(e)(x)$ is the attribute value of $x$ for event $e$.*

Events carry multiple data attributes, i.e., *event attributes* (Def. 2), which can be either categorical or numeric, depending on features of the process and the recording information systems. In this paper, we consider that any event log records at least four standard event attributes: case identifier (*case*), activity label (*act*), timestamp (*time*), and resource identifier (*resource*). Each case identifier corresponds to a unique process execution instance. Specifically, an event attribute is a case attribute if events belonged to the same case share an identical value on that attribute. Case identifier is a case attribute.

**Definition 2 (Event Attributes).** *Let $\mathcal{C} \subseteq \mathcal{U}_{Val}$, $\mathcal{A} \subseteq \mathcal{U}_{Val}$, $\mathcal{T} \subseteq \mathcal{U}_{Val}$ and $\mathcal{R} \subseteq \mathcal{U}_{Val}$ denote the universe of case identifiers, the universe of activity names,*

*the universe of timestamps, and the universe of resource identifiers, respectively. Any event log* $EL = (E, Att, \pi)$ *has three special attributes from the set* $D = \{case, act, time\}$*, referred to as the core event attributes, and a special attribute res, i.e.,* $D \cup \{res\} \subseteq Att$*, such that for any* $e \in E$*:*

- $D \subseteq dom(\pi(e))$,
- $\pi_{case}(e) \in \mathcal{C}$ *is the case to which e belongs,*
- $\pi_{act}(e) \in \mathcal{A}$ *is the activity e refers to,*
- $\pi_{time}(e) \in \mathcal{T}$ *is the time at which e occurred, and*
- $\pi_{res}(e) \in \mathcal{R}$ *is the resource that executed e if* $res \in dom(\pi(e))$.

*Given a resource* $r \in \mathcal{R}$*, let* $[E]_r = \{\, e \in E \mid res \in dom(\pi(e)) \wedge \pi_{res}(e) = r \,\}$ *denote the set of events in the log execution by that resource.* $[E]_{\mathcal{R}} = \bigcup_{r \in \mathcal{R}} [E]_r$ *is the set of all events in the log that have resource information.*

*Types* describe the categorization of events. We consider *case types*, *activity types*, and *time types* (Def. 3) related to the three core dimensions of process execution. Case types describe the categories of cases, for example, insurance claims can be classified by the type of insurance (e.g., health insurance vs. car insurance). Similarly, activity types categorize activity labels into groups of relevant activities (e.g., claim investigation vs. customer support), and time types categorize timestamps into periods (e.g., weekdays vs. weekends).

**Definition 3 (Case Types, Activity Types, and Time Types).** *Let* $\mathcal{CT}$*,* $\mathcal{AT}$*, and* $\mathcal{TT}$ *denote the sets of names of case types, activity types, and time types, respectively. The functions* $\varphi_{case} \colon \mathcal{CT} \to \mathcal{P}(\mathcal{C})$*,* $\varphi_{act} \colon \mathcal{AT} \to \mathcal{P}(\mathcal{A})$*, and* $\varphi_{time} \colon \mathcal{TT} \to \mathcal{P}(\mathcal{T})$ *define partitions over* $\mathcal{C}$*,* $\mathcal{A}$*, and* $\mathcal{T}$*, respectively.*

Given an event log $EL = (E, Att, \pi)$, a set of types for one of the process execution dimensions (i.e., case, activity, and time) can be derived based on a set of event attributes, if those attributes are a set of *type-defining attributes* for that dimension (Def. 4). This notion supports deriving types via partitioning the values of some selected event attributes that are related to the categorization of cases, activities, or times. For example, in an insurance claim process event log, we may use case attributes "customer type" and "insurance type" as the type-defining attributes for case types — cases can be categorized into disjoint groups such as ("gold customer", "health insurance") vs. ("silver customer", "car insurance", "boat insurance").

**Definition 4 (Type-Defining Attributes).** *Let* $d \in D$ *be a core event attribute. Given an event log* $EL = (E, Att, \pi)$*, for any* $e \in E$*, let* $X \subseteq dom(\pi(e))$ *be some event attributes recorded in the log,* $\pi(e) \restriction_X$ *the restriction of* $\pi(e)$ *on* $X$*,* $V = \{\, \pi(e) \restriction_X \mid e \in E \,\}$ *the mappings of the attributes in* $X$ *recorded in EL.*

*$X$ is a set of type-defining attributes for $d$ in EL, if there exists a partition $P$ of $V$, such that for all $p, q \in P$,*

$$p \neq q \Rightarrow \{\, \pi_d(e) \mid e \in E \wedge \pi(e) \restriction_X \in p \,\} \cap \{\, \pi_d(e) \mid e \in E \wedge \pi(e) \restriction_X \in q \,\} = \varnothing,$$

*i.e., the partition $P$ corresponds to a partition of the set of distinct values of $d$ recorded in EL.*

## 4 Problem Modeling

In this section, we introduce how we model the problem of learning execution contexts from an event log. We first present the idea of *categorization rules* for defining the classification of case types, activity types, and time types (Sect. 4.1). Then, we discuss how to measure the *quality* of execution contexts with regard to an event log (Sect. 4.2). Finally, we formulate the execution context learning problem based on the notion of categorization rules and the quality measures.

### 4.1 Categorization Rules

A set of execution contexts specifies a way of partitioning events by defining case types, activity types, and time types. Hence, to learn execution contexts from an event log requires learning those types, i.e., the classification of cases, activities, and times. To this end, we propose to use *categorization rules* to represent types and execution contexts.

A categorization rule is a conjunctive boolean formula (Def. 5) consisting of one or more clauses. Each clause can evaluate an event by its value of some event attribute. For instance, $\sigma = \text{customer\_type} \in \{\mathsf{gold}\} \wedge \text{amount} \in [10,000, \infty)$ is a categorization rule evaluating a categorical (case) attribute "customer type" and a numeric attribute "amount". Given a set of events, evaluating this rule filters events that record gold customers and amount greater than 10,000.

**Definition 5 (Categorization Rule).** *Given an event log $EL = (E, Att, \pi)$, let $d \in D$ be a core event attribute, let $X \subseteq Att$ be a set of type-defining attributes for $d$. $\sigma = \bigwedge_{x \in X} x \in U_x$ is a categorization rule, where $e \in E$ and $U_x \in \mathcal{P}(\mathcal{U}_{Val})$ is a set of attribute values for $x \in X$. For any $e \in E$, $\sigma$ can be evaluated as follows: $[\![\sigma]\!](e) = true$ if and only if $\pi_x(e) \in U_x$ for all $x \in X$.*

- $[E]_\sigma = \{\, e \in E \mid [\![\sigma]\!](e) \,\}$ *is the set of events in the log satisfying the categorization rule $\sigma$.*
- *We introduce a default rule $\sigma_{\text{true}}$ such that $[\![\sigma_{\text{true}}]\!](e) = true$ for all $e \in E$. It follows that $[E]_{\sigma_{\text{true}}} = E$.*
- *Any two categorization rules $\sigma_1$ and $\sigma_2$ are equivalent, i.e., $\sigma_1 \cong \sigma_2$, if and only if $[E]_{\sigma_1} = [E]_{\sigma_2}$ for any $E \subseteq \mathcal{E}$. Otherwise, we write $\sigma_1 \ncong \sigma_2$.*

A set of categorization rules can be used to define a set of types on an event log (Def. 6). Consider the example of defining case types. Assume an event log of the insurance claim process has "customer type" as a case attribute. Then the set of rules $\Sigma_1 = \{\text{customer\_type} \in \{\mathsf{gold}\}, \text{customer\_type} \in \{\mathsf{silver}\}, \text{customer\_type} \in \{\mathsf{bronze}\}\}$ can define three case types for this event log, as long as a customer can only be either gold, silver, or bronze. But, for example, another set with two rules, $\Sigma_2 = \{\text{customer\_type} \in \{\mathsf{gold}\}, \text{customer\_type} \in \{\mathsf{silver}, \mathsf{bronze}\}\}$, would also define case types.

**Definition 6 (Define Types by Categorization Rules).** *Given an event log $EL = (E, Att, \pi)$, let $d \in D$ be a core event attribute. $\Sigma$ is a set of categorization rules that define a set of types on $d$, if and only if:*

1. *for any* $\sigma_1, \sigma_2 \in \Sigma$, $\{ \pi_d(e) \mid e \in [E]_{\sigma_1} \} \cap \{ \pi_d(e) \mid e \in [E]_{\sigma_2} \} = \varnothing$; *and*
2. $\bigcup_{\sigma \in \Sigma} \{ \pi_d(e) \mid e \in [E]_\sigma \} = \bigcup_{e \in E} \{\pi_d(e)\}$,

*i.e., the subsets of events satisfying categorization rules in* $\Sigma$ *induce a partition of all values of* $d$ *recorded in EL.*

Execution contexts can be defined by three sets of categorization rules that define case types, activity types, and time types, respectively (Def. 7). Given an event log, a set of execution contexts enables (i) projecting the events as data points onto a three-dimensional data space and (ii) partitioning them into sub-logs that can be selected and linked with resources for analyses [16,17].

**Definition 7 (Execution Context).** *Given an event log* $EL = (E, Att, \pi)$*, let* $\Sigma_{case}$*,* $\Sigma_{act}$*, and* $\Sigma_{time}$ *be three sets of categorization rules that define case types, activity types, and time types, respectively.* $CO = \Sigma_{case} \times \Sigma_{act} \times \Sigma_{time}$ *is a set of execution contexts defined by the three sets of categorization rules.*

*CO specifies a way of partitioning EL. Given an execution context* $co = (\sigma_c, \sigma_a, \sigma_t) \in CO$*,* $[E]_{co} = [E]_{\sigma_c} \cap [E]_{\sigma_a} \cap [E]_{\sigma_t}$ *is the set of events in the log having that execution context.*

### 4.2 Quality Measures for Execution Contexts

Given an event log, any categorization rules — as long as they fulfill the requirement (Def. 6) — can be proposed for defining types, resulting in many candidate sets of execution contexts. In this section, we discuss how to measure the quality of execution contexts learned from event logs.

Execution contexts can be applied to characterize resource behavior that concern certain process execution features determined by the specialization of work, a.k.a, division of labor [4]. On the one hand, when specialization is low in a process, resources tend to be interchangeable when performing in process execution, and events they originated are mostly similar. On the other hand, when specialization is high, resources are limited to undertaking specific kinds of tasks, as exhibited by the differences among their originated events. This idea motivates us to consider the following criteria for a set of *good* execution contexts: (i) events originated by the same resource should be partitioned into few execution contexts; and (ii) events in the same execution contexts should be originated by few resources.

Fig. 2 illustrates the idea. When resources are considered generalized due to low specialization of work, a small number of execution contexts should be sufficient. When (a group of) resources are highly specialized, it is desired to have dedicated execution contexts for each of them to capture their specific characteristics. We define two quality measures, namely *dispersal* and *impurity*.

*Dispersal.* Dispersal measures the extent to which events originated by the same resource disperse across different execution contexts (Eqn. 1), and yields values in $[0, 1]$. High quality execution contexts have low dispersal, i.e., characterizing

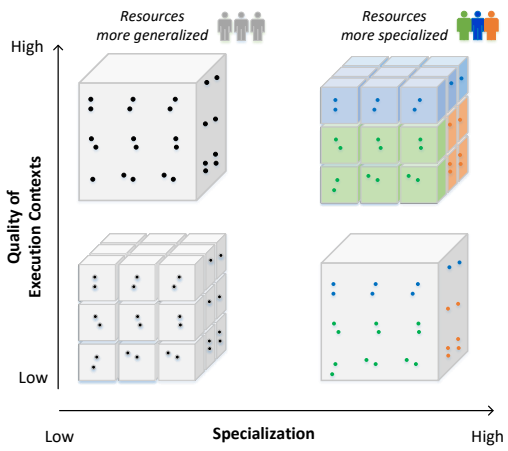



Fig. 2: Execution context quality: It is desired to use few dedicated execution contexts (cells) to characterize resource behavior recorded in events

the behavior of each individual resource with few execution contexts. Given an event log $EL$ and a set of execution contexts $CO$,

$$Dis(EL, CO) = \sum_{r \in \mathcal{R}} \left( \frac{|[E]_r|}{|[E]_{\mathcal{R}}|} \times \frac{\sum_{e_1, e_2 \in [E]_r} d_{CO}(e_1, e_2)}{\binom{|[E]_r|}{2}} \right), \tag{1}$$

is the dispersal of $CO$ with regard to $EL$. In the context of the given execution contexts $CO$, any event $e \in E$ corresponds to a unique execution context $co^e = (ct^e, at^e, tt^e) \in CO$, for which $e \in [E]_{co^e}$. Then, for any two events $e_1, e_2 \in E$, we define the distance between them using their corresponding execution contexts $co^{e_1} = (ct^{e_1}, at^{e_1}, tt^{e_1})$ and $co^{e_2} = (ct^{e_2}, at^{e_2}, tt^{e_2})$, that is,

$$d_{CO}(e_1, e_2) = \frac{[ct^{e_1} \not\cong ct^{e_2}] + [at^{e_1} \not\cong at^{e_2}] + [tt^{e_1} \not\cong tt^{e_2}]}{ndim}, \tag{2}$$

where $[\varphi]$ is the Iverson bracket that returns 1 if a boolean formula $\varphi$ holds and 0 otherwise, and $ndim \in \{1, 2, 3\}$ is the number of process dimensions considered in a set of execution contexts. By default, we let $ndim = 3$. However, it is possible that there are not any types defined on a dimension. For example, the case dimension can be omitted if, for any $(ct, at, tt) \in CO$, $ct = \sigma_{\text{true}}$, and thus we have $ndim = 2$. Specifically, if there is only one execution context for all events in a log, then $Dis(EL, CO) = 0$.

*Impurity.* Impurity measures the extent to which the same execution context contains events originated by different resources. A set of execution contexts is good when most of them contain only events originated by few resources, i.e.,

characterizing the behavior specific to each individual resource. This is built upon the existing measure of entropy in data mining. Given an event log $EL$ and a set of execution contexts $CO$,

$$Imp(EL, CO) = \frac{1}{\sum_{r \in \mathcal{R}} p_r \log_2 p_r} \sum_{co \in CO} \left( \frac{|[E]_{\mathcal{R}} \cap [E]_{co}|}{|[E]_{\mathcal{R}}|} \times \sum_{r \in \mathcal{R}} p_{r,co} \log_2 p_{r,co} \right), \tag{3}$$

is the impurity of $CO$ with regard to $EL$, where

$$p_r = \frac{|[E]_r|}{|[E]_{\mathcal{R}}|},\ p_{r,co} = \frac{|[E]_r \cap [E]_{co}|}{|[E]_{\mathcal{R}} \cap [E]_{co}|} \tag{4}$$

are the relative frequency of events originated by a resource $r$ in terms of the entire log and an execution context $co$, respectively. Impurity yields a value in $[0, 1]$. If there is only one execution context for all events in a log, then $Imp(EL, CO) = 1$.

**Problem Statement** Learning execution contexts is to derive from an event log three sets of categorization rules that define case types, activity types, and time types, respectively, such that the resulting execution contexts have low dispersal and low impurity with respect to the input log.

## 5 Problem Solution

We propose an approach based on decision trees to solve the problem of deriving categorization rules (see Fig. 3). Below, we elaborate on the approach.

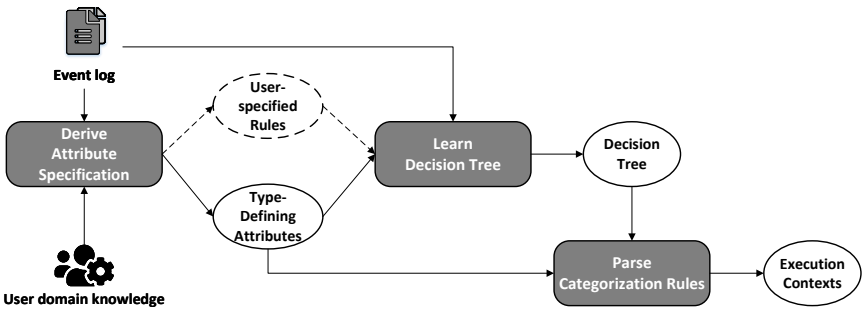



Fig. 3: Approach to deriving categorization rules from an event log to learn execution contexts

### 5.1 Derive Attribute Specification

Inputs to the approach include an event log and domain knowledge from the user. First, an *attribute specification* (Def. 8) is derived to capture user domain knowledge about the events attributes in the log. An attribute specification comprises (1) $X^{EL}_{case}$, $X^{EL}_{act}$, $X^{EL}_{time}$, which are three sets of type-defining attributes (see Def. 4) regarding case types, activity types, and time types; and optionally, (2) $\Lambda$, which is a set of user-specified categorization rules capturing any existing categorization of values for those event attributes. If user-specified categorization rules are unavailable for any event attribute $x$, we let $\Lambda(x) = \varnothing$.

**Definition 8 (Attribute Specification).** *Let $EL = (E, Att, \pi)$ be an event log and $\Sigma$ be the set of all possible categorization rules defined on $Att$. $S = (X^{EL}_{case}, X^{EL}_{act}, X^{EL}_{time}, \Lambda)$ is an attribute specification on $EL$ for learning categorization rules. $X^{EL}_{case} \subseteq Att$, $X^{EL}_{act} \subseteq Att$, and $X^{EL}_{time} \subseteq Att$ are three disjoint, non-empty sets of type-defining attributes. $\Lambda : Att \nrightarrow \mathcal{P}(\Sigma)$ defines a set of categorization rules for the event attributes in $X^{EL}_{case} \bigcup X^{EL}_{act} \bigcup X^{EL}_{time}$.*

An attribute specification informs how attributes values should be handled in the following step of decision tree learning, and subsequently how the categorization rules extracted from a decision tree specify case types, activity types, and time types.

### 5.2 Learn Decision Tree

We apply a decision tree induction framework to extract categorization rules from an event log. In decision tree learning, a dataset of multivariate data tuples is iteratively partitioned into smaller subsets by deriving splitting rules on data attributes. The output is represented in a tree structure, where tree nodes hold the subsets of the input data and branches record the disjunctive splitting rules used to obtain the subsets. Decision tree learning is a common solution for classification tasks. For that purpose, splitting rules are often derived following a greedy heuristic that minimizes the information needed to classify data tuples.

We decide to apply decision tree learning since it resembles how we expect to develop execution contexts defined by categorization rules — deriving rules to partition a dataset based on data attributes. In addition, the tree representation provides an intuitive way to understand how execution contexts are derived incrementally. However, compared to the conventional decision tree learning problem, learning rules for execution contexts imposes two challenges: (i) we require splitting rules extracted from a decision tree to be categorization rules which can be used for defining types (see Def. 6); and (ii) the goal of learning is to derive execution contexts instead of building a predictive model for classification or regression.

To address the first issue regarding categorization rules, we choose to construct *Oblivious Decision Trees* (ODT) [7]. An ODT is different from a conventional decision tree in such a way that an ODT's nodes at the same level are constructed by splitting rules based on the *same* data attribute. For any two leaf

nodes on an ODT, if we project their data subsets onto a split attribute, then the two projected sets are either disjoint or identical. This feature ensures that a learned ODT can be used to produce categorization rules for defining types.

To address the second issue regarding the learning goal, we use harmonic mean to combine dispersal and impurity and apply a greedy heuristic — whenever there exist several sets of categorization rules as candidates, choose the set that leads to the lowest harmonic mean.

Algorithm 1 describes the customized decision tree learning algorithm. It begins with an empty root node which holds all events in a given log (Line 1). At each iteration, the best split will be found, i.e., finding the best type-defining attribute and its corresponding categorization rules to be applied (Line 3). This selection (`FindBestSplit`) is based on calculating the harmonic mean of dispersal and impurity. We elaborate on this step later. If that best attribute for splitting can be found, the decision tree is expanded by applying the categorization rules to every leaf node and growing a subtree there. This ensures that the tree is grown as an ODT (Line 5). The decision tree keeps growing either until the next best split cannot be found (Line 6–7) or until the height of the decision tree exceeds a preset maximum value (Line 2). After the iterative tree growth stops, we traverse every level of the current tree and select a subtree to be returned for the subsequent step of parsing categorization rules (Line 10).

Below, we explain the two key operations in the procedure, `FindBestSplit` (Line 3) and `SelectSubTree` (Line 10).

`FindBestSplit` is a sub-procedure for selecting a type-defining attribute and its corresponding categorization rules, i.e., the best split. First, for all type-defining attribute given in the attribute specification ($x \in X^{EL}_{case} \cup X^{EL}_{act} \cup X^{EL}_{time}$), we generate the corresponding categorization rules as candidate splits. When $\Lambda(x) \neq \varnothing$, i.e., there exist user-specified categorization rules for $x$, we use those rules. Otherwise, we consider $x$ a generic data attribute and apply methods in conventional decision tree learning [6] to infer possible ways to split on $x$: if $x$ is numeric, apply a histogram-based algorithm; if $x$ is categorical, compute the possible two-subset partitions over its values and sample from all partitions. Note that the attributes are used with replacement, i.e., $x$ can be split more than once in the iterative procedure.

**Algorithm 1:** The customized decision tree learning algorithm

```
   input : EL = (E, Att, π), an event log; S, an attribute specification;
           H, a constant specifying the maximum height of the tree
   output: a decision tree
 1 root ← CreateTreeNode(E)
 2 for h ← 1 to H do
 3 |   attr, rules ← FindBestSplit(root, EL, S)
 4 |   if attr ≠ ∅ then
 5 |   |   ExpandTreeOnEveryLeafNode(root, attr, rules)
 6 |   else
 7 |   |   break
 8 |   end
 9 end
10 return SelectSubTree(root)
```

After the candidate splits are generated for every type-defining attribute, we need to evaluate and select one of them to expand the tree. To do this, test how the current tree would be expanded if a candidate split applied. Then, parse the full set of categorization rules from the "test tree" to determine the execution contexts (see Sect. 5.3). This way, we can obtain $CO_x$ for every candidate split. Finally, the best split can be decided by choosing the candidate whose $CO_x$ would lead to the lowest harmonic mean of dispersal and impurity.

`SelectSubTree` is a sub-procedure for deciding the subtree to be returned. Here, a subtree refers to a subtree sharing the same root node as the entire decision tree, i.e., any intermediate result obtained during the iterative tree growth process or the complete decision tree. To make the selection, we first apply the elbow method to identify turning points where dispersal and impurity changed significantly. Then, a single subtree can be decided by selecting from the identified turning points.

The values of dispersal and impurity are expected to show opposite trends as a decision tree grows. Initially, all events are placed together (held by the root node), and hence dispersal is 0 while impurity is 1. As the decision tree grows, the number of leaf nodes increases (so is the number of their corresponding execution contexts), which leads to the increase in dispersal and decrease in impurity.

### 5.3 Parse Categorization Rules

The parsing of categorization rules is to transform the rules recorded on a decision tree into execution contexts that we need (Def. 7). This transformation happens both when we need to evaluate intermediate results (`FindBestSplit`) and also when we need to obtain the final execution contexts after the decision tree learning stops.

To parse categorization rules, we first follow the conventional way of rule extraction from a decision tree. That is, for each path from the root to a leaf node, a decision rule is formed by conjoining all the rules recorded along the path. Then, for every decision rule obtained, we use the attribute specification as a reference to determine which part of the decision rule is related to case types, activity types, or time types, respectively. Formally, every such decision rule $\sigma$ can be written as a conjunction $(\sigma_c \wedge \sigma_a \wedge \sigma_t)$, where any of $\sigma_c$, $\sigma_a$, $\sigma_t$ can be a default rule ($\sigma_{\text{true}}$) if no type-defining attributes are included for any of the core event attributes.

As such, we will be able to transform a decision rule related a leaf node of a decision tree into an execution context $co = (\sigma_c, \sigma_a, \sigma_t)$. A set of execution context $CO$ is obtained by parsing the categorization rules for all leaf nodes.

## 6 Experiments

We implemented the approach and evaluated it through an experiment. The aim is two-fold: (i) to test the feasibility of using our approach to learn execution contexts, and (ii) to demonstrate how the learned execution contexts can be

applied for resource-oriented analyses. We share our implementation and the experiment details in an open repository online[3].

### 6.1 Experiment Setup

We evaluated our approach on a publicly available, real-life event log dataset, BPIC-15[4], which consists of five event logs that record how five Dutch municipalities performed in a building permit application handling process. We merged them into a single log for the experiment.

Some event attributes were used as type-defining attributes for case types and activity types: *phase* is an event attribute indicating the phase of the process where activities belong to; and *case_parts* is a case attribute indicating the type of project related to the building permit applications. For time types, we derived two type-defining attributes from the original timestamps and appended them to the event logs, i.e., *weekday* and *am/pm* (AM time vs. PM time).

Table 1 describes the experiment dataset and the attribute specification used for learning execution contexts. Specifically, for attribute *case_parts*, we defined the following user-specified categorization rules based on the original description of the data — two rules partition the values of *case_parts* into two subsets, depending on whether a value contains the string 'Bouw' (indicating the case is related to construction) or not.

Table 1: The event log dataset and the attribute specification used for learning execution contexts

| Log statistics | | | | Attribute specification | | |
|---|---|---|---|---|---|---|
| #events | #cases | #activities | #resources | $Att_{case}$ | $Att_{act}$ | $Att_{time}$ |
| 193453 | 5599 | 154 | 71 | {*case_parts*} | {*phase*} | {*weekday*,*am/pm*} |

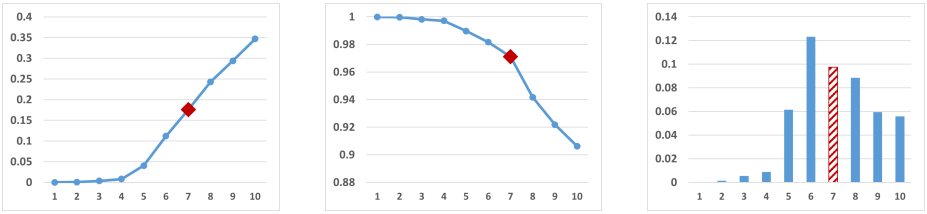
### 6.2 Learning Execution Contexts

We applied our approach with the maximum tree height $H$ set to 10 and selected the subtree corresponded to the point where the harmonic mean changed most significantly. Fig. 4 illustrates the values of dispersal and impurity per iteration when learning a decision tree and the of changes of their harmonic mean. From Fig. 4a and Fig. 4b, we can observe a clear upward trend of dispersal values and a downward trend of impurity values. This confirms our discussion on the feature of decision tree learning as mentioned before (see Sect. 5.2). There are obvious changes to dispersal and impurity from iteration 5 onwards, which align with the changes of their harmonic mean illustrated in Fig. 4c. We selected iteration 7

[3] Implementation and experiment details: `https://royjy.me/to/learn-co`

[4] BPIC-15 dataset: `https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1`

as the "elbow" point, where the increase of harmonic mean starts to slow down. We used its corresponding subtree to parse categorization rules and obtained a set of 12 execution contexts.




(a) dispersal (b) impurity (c) $\Delta$harmonic mean

Fig. 4: Dispersal, impurity, and changes of their harmonic mean per iteration

Table 2 shows the learned execution contexts. They are sorted by the number of events they contain. We can see that case type remains the default rule ($\sigma_{\text{true}}$), which means that attribute *case_parts* was not selected to derive categorization rules. A possible reason is that resources are similar in terms of handling applications concerned with different types of projects. For the activity types, we find the cluster of phases '0', '1', '4', '5' as a single type. These phases are related to the frequently executed activities in the process, where most resources participated in. Finally, the derived time types are only based on partitioning *weekdays*, which show a clear pattern that aligns with common working hours. The other type defining attribute *am/pm* was not used, implying similarities of resource workload in the morning vs. afternoon. These findings about types are consistent with our visual analyses on the behavior of the five municipalities, reported in our previous work [16].

Table 2: The 12 execution contexts learned from log BPIC-15

| id | Case type | Activity type | Time type | #events | #res. |
|---|---|---|---|---|---|
| 1 | $\sigma_{\text{true}}$ | (*phase* $\in$ {'0','1','4','5'}) | (*weekday* $\in$ {'Mon','Tue','Wed','Thu','Fri'}) | 150666 | 70 |
| 2 | | (*phase* $\in$ {'3'}) | (*weekday* $\in$ {'Mon','Tue','Wed','Thu','Fri'}) | 17836 | 53 |
| 3 | | (*phase* $\in$ {'2'}) | (*weekday* $\in$ {'Mon','Tue','Wed','Thu','Fri'}) | 16451 | 57 |
| 4 | | (*phase* $\in$ {'8'}) | (*weekday* $\in$ {'Mon','Tue','Wed','Thu','Fri'}) | 6871 | 47 |
| 5 | | (*phase* $\in$ {'0','1','4','5'}) | (*weekday* $\in$ {'Sat'}) | 778 | 35 |
| 6 | | (*phase* $\in$ {'7'}) | (*weekday* $\in$ {'Mon','Tue','Wed','Thu','Fri'}) | 585 | 32 |
| 7 | | (*phase* $\in$ {'0','1','4','5'}) | (*weekday* $\in$ {'Sun'}) | 149 | 22 |
| 8 | | (*phase* $\in$ {'3'}) | (*weekday* $\in$ {'Sat'}) | 49 | 6 |
| 9 | | (*phase* $\in$ {'2'}) | (*weekday* $\in$ {'Sat'}) | 41 | 6 |
| 10 | | (*phase* $\in$ {'6'}) | (*weekday* $\in$ {'Mon','Tue','Wed','Thu','Fri'}) | 15 | 5 |
| 11 | | (*phase* $\in$ {'8'}) | (*weekday* $\in$ {'Sat'}) | 10 | 1 |
| 12 | | (*phase* $\in$ {'7'}) | (*weekday* $\in$ {'Sat'}) | 2 | 2 |
| | | | Total | 193453 | 71 |

#res.: number of unique resources performed in an execution context

### 6.3 Applying Execution Contexts

This section reports how the learned execution contexts can be applied to (i) the analysis of resource profiles [8], i.e., describing the behavior of resources in process execution, and (ii) the discovery of organizational models [12], i.e., finding groups of resources having similar characteristics.

**Resource Profile Analysis** For an illustration purpose, we selected the 31 resources who performed in the last 6 execution contexts (i.e., id 7–12) and calculated their *activity frequency* [8] with regard to those execution contexts.

Fig. 5 shows a heatmap that visualizes the results. Darker colors indicate larger values. We can see that resources exhibit clear differences. Most of them only worked in execution context 7, which is the overtime work ('Sun') on the main phases ('0', '1', '4', '5'). Another distinct pattern is concerned with resources working only in execution contexts 8 and 9. They showed balanced activity frequency regarding the two activity types (phase '3' and '2'). Execution context 11 is specific to a single resource '560519'. This is an interesting observation compared to execution context 4, where the same set of activities performed on weekdays were covered by much more resources (47 of them).

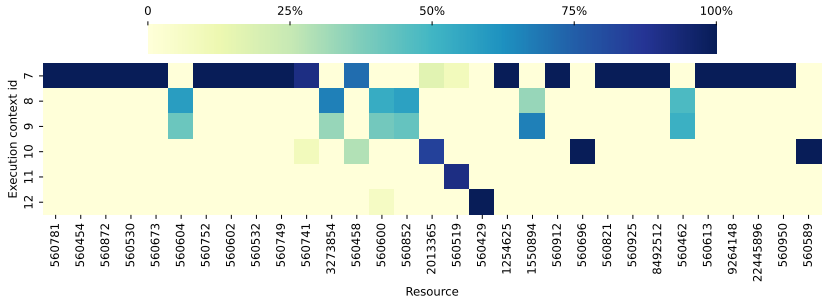



Fig. 5: Visualizing resources' activity frequency given execution contexts 7–12

**Organizational Model Discovery** The original AHC method [12] derives an "originator (resource) by activity" matrix, feeds it to agglomerative hierarchical clustering, and gets resource clusters as the output. In our demonstration, we use a "resource by execution context" matrix instead, i.e., count how frequently resource conducted events having specific execution contexts. We keep identical settings on all other steps and apply the silhouette score [11] to evaluate the quality of the outputs. We varied the desired cluster number between 2 and 70.

Fig. 6 shows the quality of discovery results, comparing between the use of activity labels vs. execution contexts. The X-axis corresponds to the desired number of resource clusters specified as a parameter, and the Y-axis corresponds to the quality (silhouette score) of the discovery result. We can observe that using execution contexts, compared to the original method, led to better output quality in most situations. This indicates that using the learned execution contexts contributed to uncover resource groups having more distinct characteristics.

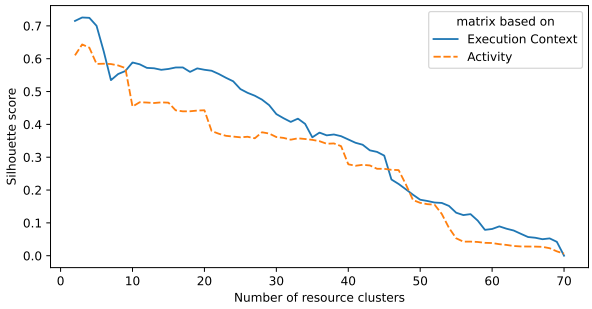



Fig. 6: Quality of discovery results measured by their silhouette scores, per selected number of clusters

**Summary** The above applications of our approach show that it can be applied before conducting resource-oriented process mining. The use of attribute specifications allows directly encoding user knowledge about the input event log, making the approach configurable with regard to event logs with different features. Learned execution contexts provide a multidimensional view on the input log, where sublogs may be related to behavior of specific resources. This can be a starting point to decompose large and complex event logs, so that resource-oriented process mining can be applied to analyze dedicated sublogs.

## 7 Conclusion

In this paper, we discussed the problem of learning execution contexts, which is concerned with characterization of resource groupings in process execution, considering three core process dimensions. We proposed an approach to automatically learn execution contexts using a dedicated decision-tree-based algorithm and tested it on real-life event log data.

Our current work has certain limitations. For one, learned execution contexts are local optimal due to the greedy heuristic and that tree induction is sequential-forward. A possible future direction is to explore searching methods with other heuristics, e.g., simulated annealing, to produce better near-optimal solutions. Second, the experiment used event log data from one process to demonstrate the usefulness of the approach. Future work can look into a more comprehensive evaluation using event logs recording different processes and containing more attributes. Furthermore, the proposed approach can be compared with methods that support the discovery of multidimensional resource characteristics related to resource groupings.

## Acknowledgments

The reported research is part of a PhD project supported by an Australian Government Research Training Program (RTP) Scholarship.

## References


1. Appice, A.: Towards mining the organizational structure of a dynamic event scenario. J. Intell. Inf. Syst. **50**(1), 165–193 (2018)
2. Bolt, A., van der Aalst, W.M.P.: Multidimensional process mining using process cubes. In: Enterprise, Business-Process and Information Systems Modeling. pp. 102–116. Springer International Publishing (2015)
3. Burattin, A., Sperduti, A., Veluscek, M.: Business models enhancement through discovery of roles. In: IEEE Symposium on Computational Intelligence and Data Mining (CIDM). pp. 103–110 (2013)
4. Daft, R.L., Murphy, J., Willmott, H.: Organization theory and design. South-Western Cengage Learning, USA (2010)
5. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer Publishing Company, Incorporated, 2nd edn. (2018)
6. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: LightGBM: a highly efficient gradient boosting decision tree. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. pp. 3149–3157. NIPS'17, Curran Associates Inc., Red Hook, NY, USA (Dec 2017)
7. Kohavi, R., Li, C.H.: Oblivious decision trees graphs and top down pruning. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence. pp. 1071–1077. Morgan Kaufmann Publishers Inc. (1995)
8. Pika, A., Leyer, M., Wynn, M.T., Fidge, C.J., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Mining resource profiles from event logs. ACM Trans. Manage. Inf. Syst. **8**(1), 1:1–1:30 (2017)
9. Reijers, H.A., Song, M., Jeong, B.: Analysis of a collaborative workflow process with distributed actors. Inf. Syst. Front. **11**(3), 307–322 (2009)
10. Ribeiro, J.T.S., Weijters, A.J.M.M.: Event cube: Another perspective on business processes. In: On the Move to Meaningful Internet Systems: OTM 2011. pp. 274–283. Springer (2011)
11. Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. J. Comput. Appl. Math. **20**, 53–65 (1987)
12. Song, M., van der Aalst, W.M.P.: Towards comprehensive support for organizational mining. Decis. Support Syst. **46**(1), 300 – 317 (2008)
13. Van der Aalst, W.M.P.: Process cubes: Slicing, dicing, rolling up and drilling down event data for process mining. In: Asia Pacific Business Process Management. pp. 1–22. Springer, Cham (2013)
14. Van der Aalst, W.M.P.: Process Mining: Data Science in Action (2nd Ed.). Springer, Berlin, Heidelberg (2016)
15. Van Hulzen, G., Martin, N., Depaire, B.: Looking beyond activity labels: Mining Context-Aware resource profiles using activity instance archetypes. In: Business Process Management Forum. pp. 230–245. Springer (2021)
16. Yang, J., Ouyang, C., ter Hofstede, A.H.M., van der Aalst, W.M.P., Leyer, M.: Seeing the forest for the trees: Group-oriented workforce analytics. In: International Conference on Business Process Management. pp. 345–362. Springer (2021)
17. Yang, J., Ouyang, C., van der Aalst, W.M.P., ter Hofstede, A.H.M., Yu, Y.: OrdinoR: A framework for discovering, evaluating, and analyzing organizational models using event logs. Decision Support Systems **158**, 113771 (2022)