Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 96 (2016) 63 – 72
Procedia Computer Science

20th International Conference on Knowledge Based and Intelligent Information and Engineering Systems

Log-based Evaluation of Label Splits for Process Models

Niek Tax[a,b,*], Natalia Sidorova[a], Reinder Haakma[b], Wil M. P. van der Aalst[a]

[a] *Eindhoven University of Technology, P.O. Box 513, Eindhoven, The Netherlands*
[b] *Philips Research, Prof. Holstlaan 4, 5665 AA Eindhoven, The Netherlands*

Abstract

Process mining techniques aim to extract insights in processes from event logs. One of the challenges in process mining is identifying interesting and meaningful event labels that contribute to a better understanding of the process. Our application area is mining data from smart homes for elderly, where the ultimate goal is to signal deviations from usual behavior and provide timely recommendations in order to extend the period of independent living. Extracting individual process models showing user behavior is an important instrument in achieving this goal. However, the interpretation of sensor data at an appropriate abstraction level is not straightforward. For example, a motion sensor in a bedroom can be triggered by tossing and turning in bed or by getting up. We try to derive the actual activity depending on the context (time, previous events, etc.). In this paper we introduce the notion of label refinements, which links more abstract event descriptions with their more refined counterparts. We present a statistical evaluation method to determine the usefulness of a label refinement for a given event log from a process perspective. Based on data from smart homes, we show how our statistical evaluation method for label refinements can be used in practice. Our method was able to select two label refinements out of a set of candidate label refinements that both had a positive effect on model precision.

Keywords: Label refinement; Process Mining; Sensor Networks

1. Introduction

Process mining is a fast growing discipline that brings together knowledge and techniques from computational intelligence, data mining, process modeling and process analysis[15]. The process mining task is the automatic or semi-automatic analysis of events that are logged during process execution, where event records contain information on what was done, by whom, for whom, where, when, etc. Events are grouped into cases (process instances), e.g. per patient for a hospital log, or per insurance claim for an insurance company. An important task within process mining is *process discovery*, which focuses on extracting interpretable models of processes from event logs. One of the attributes of the events is usually used as its label. These event labels are then used as transition/activity labels in the process models created by process discovery algorithms.

[*] Corresponding author. Tel.: +31-63-408-5760;
E-mail address: n.tax@tue.nl

doi:10.1016/j.procs.2016.08.096

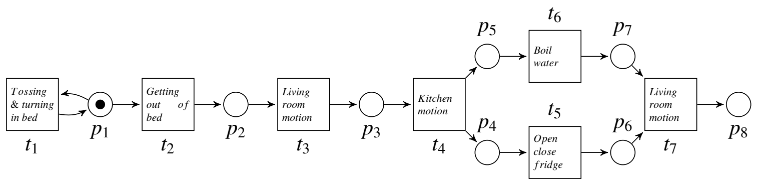


Fig. 1: A Petri net derived from the event log in Table 1

Table 1: The corresponding smart home sensor event log with refined labels

Id	Timestamp	Address	Sensor	Heart rate	Activity
1	03/11/2015 02:45	Mountain Rd. 7	Bedroom motion	74	Tossing & turning
2	03/11/2015 03:23	Mountain Rd. 7	Bedroom motion	72	Tossing & turning
3	03/11/2015 04:59	Mountain Rd. 7	Bedroom motion	71	Tossing & turning
4	03/11/2015 06:04	Mountain Rd. 7	Bedroom motion	73	Tossing & turning
5	03/11/2015 08:45	Mountain Rd. 7	Bedroom motion	85	Getting up
6	03/11/2015 09:10	Mountain Rd. 7	Living room motion	79	Living room motion
…	03/11/2015 …	Mountain Rd. 7	…	…	…
7	03/12/2015 01:01	Mountain Rd. 7	Bedroom motion	73	Tossing & turning
8	03/12/2015 03:13	Mountain Rd. 7	Bedroom motion	75	Tossing & turning
9	03/12/2015 07:24	Mountain Rd. 7	Bedroom motion	74	Tossing & turning
10	03/12/2015 08:34	Mountain Rd. 7	Bedroom motion	79	Getting up
11	03/12/2015 09:12	Mountain Rd. 7	Living room motion	76	Living room motion
…	03/12/2015 …	Mountain Rd. 7	…	…	…
12	03/13/2015 00:45	Mountain Rd. 7	Bedroom motion	75	Tossing & turning
13	03/13/2015 02:29	Mountain Rd. 7	Bedroom motion	75	Tossing & turning
14	03/13/2015 05:19	Mountain Rd. 7	Bedroom motion	74	Tossing & turning
15	03/13/2015 05:34	Mountain Rd. 7	Bedroom motion	79	Tossing & turning
16	03/13/2015 05:39	Mountain Rd. 7	Bedroom motion	77	Tossing & turning
17	03/13/2015 08:37	Mountain Rd. 7	Bedroom motion	79	Getting up
18	03/13/2015 08:52	Mountain Rd. 7	Living room motion	78	Living room motion
…	03/13/2015 …	Mountain Rd. 7	…	…	…
19	03/14/2015 03:41	Mountain Rd. 7	Bedroom motion	75	Tossing & turning
20	03/14/2015 05:00	Mountain Rd. 7	Bedroom motion	74	Tossing & turning
21	03/14/2015 08:52	Mountain Rd. 7	Bedroom motion	75	Getting up
22	03/14/2015 09:30	Mountain Rd. 7	Living room motion	74	Living room motion
…	03/14/2015 …	Mountain Rd. 7	…	…	…
23	03/15/2015 02:11	Mountain Rd. 7	Bedroom motion	77	Tossing & turning
24	03/15/2015 02:34	Mountain Rd. 7	Bedroom motion	76	Tossing & turning
25	03/15/2015 08:35	Mountain Rd. 7	Bedroom motion	79	Getting up
26	03/15/2015 08:57	Mountain Rd. 7	Living room motion	77	Living room motion
…	03/15/2015 …	Mountain Rd. 7	…	…	…

Process mining takes its roots in the field of business process management, where the definition of labels for events is considered to be rather straightforward. In recent years, the application domain of process mining has broadened. A wide variety of event types can be used as input and analysis may be challenging. One of the most challenging application areas is *LifeLogging*, which focuses on acquisition and analysis of personal daily life data. LifeLogs amongst others combine data collected through mobile phones, wearable devices, and/or smart home sensors. The emergence of LifeLogging tools and the resulting increase in availability of activity data enable a process-centric analysis of human behavior[14]. The aim of process mining analysis on LifeLogging data is to find frequent activity patterns and represent them in a human interpretable process model. Such a process model could then also be used to detect deviations from one's regular behavior. Process mining in the human behavior application domain closely relates to the field of activity recognition, which aims to detect human activities from sensors and finding patterns between human activities[2]. Process mining, however, aims to produce interpretable models that can provide insights by visually inspecting them. In contrast, most activity recognition techniques produce non-interpretable models.

Imagine an elderly person of whom we want to discover a process model describing his/her daily behavior. Events are generated by sensors, either periodically (e.g. by a temperature sensor or heart rate monitor), or triggered by some activity (e.g. motion). Table 1 shows an example log obtained by fusing data from such sensors. The dots indicate that only a fraction of the logged events are shown. Assigning meaningful labels to these events is not straightforward. A *Bedroom motion* event can be caused by different human activities, e.g. by *Tossing & turning* or by *Getting up*. In some cases it is necessary to distinguish between *Tossing & turning* and *Getting up*, for example when we aim to generate a timely reminder to take medication that needs to be taken before breakfast. Based on contextual information (e.g. a specific increase in heart rate, a time stamp, etc.), the distinction between the two types of activities might be identified, and each event with label *Bedroom motion* can be refined into either *Tossing & turning* or *Getting up*. The last column in Table 1 shows the desired event labels. Figure 1 shows a process model that can be deduced from such a log using existing process discovery techniques, like the ones from[17,21].

Many relabelings of *Bedroom motion* events are possible. Expert knowledge, data mining or machine learning techniques can be used to generate ideas for potential labeling functions. The goal of this labeling function is to give "similar" events the same label. However, similarity is a relative notion, *so the initially chosen labeling function can be too abstract or too fine-grained to generate an informative process model.* Once a process discovery algorithm has been applied and a process model is obtained, one can assess whether the labeling function used on the original event log allowed the process discovery algorithm to discover an informative process model. However, it is computationally costly to apply process mining algorithms to multiple event logs generated from a single original event log using different event labeling functions with varying levels of abstraction. Therefore, we provide a statistical approach to evaluate label refinement usefulness in the context of process discovery that is based on significance testing of differences in event ordering relations.

The Fodina[20] and the α^{*}[7] process discovery algorithms assume that there is one column in the event log that indicates the activity and refine this label based on a threshold of differentness on the event labels occurring directly before and after. In this paper we assume that the information what activity is performed is spread over multiple columns. We choose one column as primary activity column and refine the activity labels based on the other columns and temporal information. We validate whether a refinement makes sense from a process perspective by taking into account all temporal event information in the event log, using statistical testing and information gain. Evaluating splits based on information gain is a well-known approach in the area of decision tree learning[11], where ground truth labels are available in contrast to the label refinement setting. Label refinements draw similarities with automatic learning of ontologies[8] in the sense that both are concerned with inferring multiple levels of semantic interpretations from data. Ontology quality evaluation techniques[1] can be used to evaluate (automatically inferred) ontologies, however these techniques are not process-centric, i.e., they do not take into account ordering relations between elements of the ontology in execution sequences.

Section 2 gives formal definitions of label refinements, process models, and related concepts. In Section 3, we discuss when a label refinement is useful from a process mining perspective. A statistical method to evaluate the usefulness of a label refinement is described in Section 4. In Section 5 we discuss the results of the proposed method on a real life smart home data set. We draw conclusions in Section 6.

2. Label Refinements & Process Models

In this section we introduce the notions related to event logs and relabeling functions for traces and then define the notions of refinements and abstractions. We also introduce the Petri net process model notation.

We use the usual sequence definition, and denote a sequence by listing its elements, e.g. we write $\langle a_1, a_2, \ldots, a_n\rangle$ for a (finite) sequence $s : \{1, \ldots, n\} \rightarrow A$ of elements from some alphabet A, where $s(i) = a_i$ for any $i \in \{1, \ldots, n\}$. The length of a sequence $s : \{1, \ldots, n\} \rightarrow A$ is $|s| = n$; $s_1 s_2$ denotes the concatenation of sequences s_1 and s_2. A *language* $\mathfrak{L}$ over an alphabet A is a set of sequences over A. $\mathfrak{L}^p$ is the prefix closure of a language $\mathfrak{L}$ (with $\mathfrak{L} \subseteq \mathfrak{L}^p$).

An event is the most elementary element of an event log. Let $\mathcal{I}$ be a set of event identifiers, $\mathcal{T}$ be a set of timestamps, and $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ be an attribute domain consisting of n attributes (e.g. resource, activity name, cost, etc.), each of a certain type. An event is a tuple $e = (i, t, a_1, \ldots, a_n)$, with $i \in \mathcal{I}$, $t \in \mathcal{T}$, and $(a_1, \ldots, a_n) \in \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$. The *event label* of an event is the attribute set $(a_1 \ldots, a_n)$; e_i, e_t and e_a respectively denote the identifier, the timestamp and label of event e. $\mathcal{E} = \mathcal{I} \times \mathcal{T} \times \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ is a universe of events over $\mathcal{A}_1, \ldots, \mathcal{A}_n$. The lines of Table 1, where we do not consider the *activity* column for now, are events from an event universe over the event attributes *sensor*, *address*, and *heart rate*.

Events are often considered in the context of other events. We call $E \subseteq \mathcal{E}$ an *event set*, if E does not contain any events with the same event identifier. The events in Table 1 together form an event set. A *trace* σ is a finite sequence formed by the events from an event set $E \subseteq \mathcal{E}$ that respects the time ordering of events, i.e. for all $k, m \in \mathbb{N}$, $1 \leq k < m \leq |E|$, we have: $\sigma(k)_t \leq \sigma(m)_t$. We define the *universe of traces* over event universe $\mathcal{E}$, denoted $\Sigma(\mathcal{E})$, as the set of all possible traces over $\mathcal{E}$. We omit $\mathcal{E}$ in $\Sigma(\mathcal{E})$ and use the shorter notation Σ when the event universe is clear from the context.

Often it is useful to partition an event set into smaller sets in which events belong together according to some criterion. We might for example be interested in discovering the typical behavior of households over the course of a day. In order to do so, we can e.g. group together events with the same *address* and the same day-part of the *timestamp*, as indicated by the horizontal lines in Table 1. For each of these event sets, we can construct a trace; time stamps define the ordering of events within the trace. For events of a trace having the same time stamps, an arbitrary ordering can be chosen within a trace.

An *event partitioning function* is a function $ep : \mathcal{E} \rightarrow T_{id}$ that defines the partitioning of an arbitrary set of events $E \subseteq \mathcal{E}$ from a given event universe $\mathcal{E}$ into event sets $E_1, \ldots, E_j, \ldots$ where each E_j is the maximal subset of E such that for any $e_1, e_2 \in E_j$, $ep(e_1) = ep(e_2)$; the value of ep shared by all the elements of E_j defines the value of the *trace attribute* T_{id}. Note that complex, multidimensional trace attributes are also possible, i.e. a combination of the name of the person performing the event activity and the date of the event, so that every trace contains activities of one person during one day. The event sets obtained by applying an event partitioning can be transformed into traces (respecting the time ordering of events).

An event log L is a finite set of traces $L \subseteq \Sigma(\mathcal{E})$. $A_L \subseteq \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ denotes the *alphabet of event labels* that occur in log L. The traces of a log are often transformed before doing further analysis: very detailed but not necessarily informative event descriptions are transformed into some *informative* and *repeatable* labels. For the labels of the log in

Table 1, the heart rate values can be abstracted to *low, normal*, and *high* or the label can be redefined to a subset of the event attributes. Next to that, if the event partitioning function maps each event from Table 1 to its address and the day-part of the timestamp, these attributes (indicated in gray) become the trace attribute and can safely be removed from individual events. The new label is then defined as a combination of the sensor and abstracted heart rate values.

After this relabeling step, some traces of the log can become identically labeled (the event id's would still be different). The information about the number of occurrences of a sequence of labels in an event log is highly relevant for process mining, since it allows differentiating between the main stream behavior of a process (frequently occurring behavioral patterns) and exceptional behavior.

Let $\Sigma(\mathcal{E})$ and $\Sigma'(\mathcal{E}')$ be two universes of traces defined over event universes $\mathcal{E}, \mathcal{E}'$. A function $l : \Sigma \rightarrow \Sigma'$ is a *trace relabeling function* if for all traces $\sigma, \gamma \in \Sigma$ such that if σ is a prefix of γ, $l(\sigma)$ is a prefix of or equal to $l(\gamma)$. We lift l to event logs: for $L \subseteq \Sigma$, the relabeling $l(L)$ is defined as $\{l(\sigma) | \sigma \in L\}$.

Often, relabeling functions are defined using a more narrow approach: first defining an event relabeling function and then lifting that function to traces. In the context of business processes, event relabeling functions are mostly mere projections of events on the values of a single attribute, such as *activity name*. We consider a more general definition to allow for history-dependent interpretation of events, which is necessary in the context of LifeLogging. Prefix preservation requirement is necessary to allow for logging, compliance checking and other forms of analysis performed at run time.

Let Σ, Σ_1, and Σ_2 be trace universes over $\mathcal{E}, \mathcal{E}_1, \mathcal{E}_2$ respectively with $\mathcal{E}, \mathcal{E}_1, \mathcal{E}_2$ being pairwise different. Let $l_1 : \Sigma \rightarrow \Sigma_1$ and $l_2 : \Sigma \rightarrow \Sigma_2$ be trace relabeling functions. Relabeling function l_1 is a *refinement* of relabeling function l_2, denoted by $l_1 \preceq l_2$, iff $\forall_{\sigma_1,\sigma_2 \in \Sigma} : l_1(\sigma_1) = l_1(\sigma_2) \implies l_2(\sigma_1) = l_2(\sigma_2)$; l_2 is then called an *abstraction* of l_1. We call a refinement l_1 of l_2 a *strict* refinement, denoted by $l_1 \prec l_2$, when $\exists_{\sigma_1,\sigma_2 \in \Sigma} : l_1(\sigma_1) \neq l_1(\sigma_2) \wedge l_2(\sigma_1) = l_2(\sigma_2)$. We call refinement l_1 of l_2 an *equal length* refinement, denoted by $l_1 \preceq^{=} l_2$,when $\forall \sigma \in \Sigma : |l_1(\sigma)| = |l_2(\sigma)|$.

Let Σ, Σ_1 be trace universes over $\mathcal{E}, \mathcal{E}_1$ respectively, $l : \Sigma \rightarrow \Sigma_1$ a trace relabeling function, and $\mathfrak{L}_1$ be a language $\mathfrak{L}_1 \subseteq \Sigma_1$ over $\mathcal{E}_1$. *Trace concretization* $l^{-1} : \Sigma_1 \rightarrow 2^{\Sigma}$ is a function defined as $l^{-1}(\sigma_1) = \{\sigma \in \Sigma | l(\sigma) = \sigma_1\}$, for each $\sigma_1 \in \Sigma_1$. *Language concretization* of $\mathfrak{L}_1$ is language $l^{-1}(\mathfrak{L}_1) = \cup_{\sigma_1 \in \mathfrak{L}_1} l^{-1}(\sigma')$.

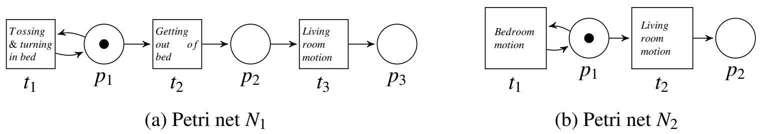
The goal of process discovery is to discover a process model that represents the behavior seen in an event log. A frequently used process modeling notation in the process mining field is the Petri net[12]. Petri nets are directed bipartite graphs consisting of transitions and places, connected by arcs. Transitions represent activities, while places represent the enabling conditions of transitions. Labels are assigned to transitions to indicate the type of activity that they model. A special label τ is used to represent invisible transitions, which are only used for routing purposes and not recorded in the execution log.

A *labeled Petri net* $N = \langle P, T, F, A_M, \ell \rangle$ is a tuple where P is a finite set of places, T is a finite set of transitions such that $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the flow relation, A_M is an alphabet of labels representing activities, with $\tau \notin A_M$ being a label representing invisible events, and $\ell : T \rightarrow A_M \cup \{\tau\}$ is a labeling function that assigns a label to each transition. For a node $n \in (P \cup T)$ we use $\bullet n$ and $n \bullet$ to denote the set of input and output nodes of n, defined as $\bullet n = \{n' | (n', n) \in F\}$ and $n \bullet = \{n | (n, n') \in F\}$. An example of a Petri net can be seen in Figure 1, where circles represent places and squares represent transitions.

A state of a Petri net is defined by its *marking* $M \in \mathbb{N}^P$ being a multiset of places. A marking is graphically denoted by putting $M(p)$ tokens on each place $p \in P$. A pair (N, M) is called a marked Petri net. State changes occur through transition firings. A transition t is enabled (can fire) in a given marking M if each input place $p \in {}^\bullet t$ contains at least one token. Once a transition fires, one token is removed from each input place of t and one token is added to each output place of t, leading to a new marking M' defined as $M' = M - \bullet t + t \bullet$. A firing of a transition t leading from marking M to marking M' is denoted as $M \xrightarrow{\ell(t)} M'$. $M_1 \xrightarrow{\ell(\sigma)} M_2$ indicates that M_2 can be reached from M_1 through a firing sequence $\sigma' \in {A_M}^*$. Many process modeling notations have formal executional semantics and define a *language of accepting traces* $\mathfrak{L}$. For Petri net N_2 in Figure 2, $\mathfrak{L}(N_2) = \{\langle Bedroom\ motion, Livingroom\ Motion \rangle, \langle Bedroom\ motion, Bedroom\ motion, Livingroom\ Motion \rangle, \langle Bedroom\ motion, \ldots, Bedroom\ motion, Livingroom\ Motion \rangle\}$.

3. On the Quality of Label Refinements for Process Mining

Process discovery algorithms discover a process model based on an event log, where event labels are obtained by applying an event relabeling function to an original log. The main quality metrics discovered process models are fitness,



(a) Petri net N_1 (b) Petri net N_2

Fig. 2: Petri nets discovered from two event logs obtained from the same event set with different relabeling functions.

precision, generalization and simplicity[15]. *Fitness* represents the share of the behavior seen in the log that is allowed by the process model. *Precision* aims at narrowing the set of traces that belong to the language of the discovered process model, but was not observed in the event log. *Generalization* aims at preventing overfitting, and *simplicity* measures the "understandability" and "well-structuredness" of models.

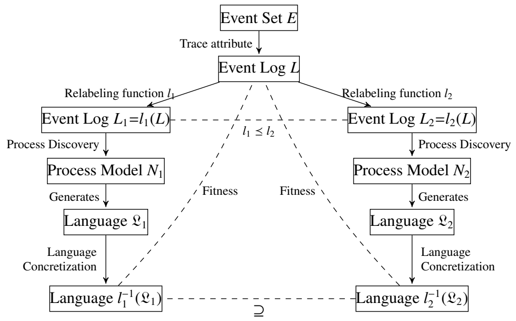


Fig. 3: Comparing two event relabeling functions

Intuitively, an event relabeling function is better than another one if it improves the quality of the discovered model along these quality dimensions. However, the quality metrics are currently defined in such a way that only results of discovery algorithms applied to the very same log can be compared, while two different relabeling functions produce logs with different event labels. The Petri net N_1 in Figure 2 has perfect precision and fitness for the event log with labels as shown in the refined label column of Table 1. At the same time, Petri net N_2 has perfect fitness and precision for the event log with labels as in the sensor column of Table 1. However, Petri net N_1 is useful for the purpose of sending a reminder message to take medicines after getting up, while Petri net N_2 is not. This suggests that Petri net N_1 is more precise than N_2, but only with respect to the original log. Thus we have to make the comparison in the context of the original log. Suppose we have a set of events E, which is part of some universe of events $\mathcal{E}$. We choose a case identifier and build an event log L from E. Then we choose relabeling functions l_1 and l_2 with $l_1 \prec l_2$ and obtain $L_1 = l_1(L)$ and $L_2 = l_2(L)$ (see Figure 3). Applying process discovery to L_1 and L_2 results in two process models, which respectively accept languages $\mathfrak{L}_1$ and $\mathfrak{L}_2$. These languages cannot be compared directly, since they contain traces consisting of different event labels. Precision metrics look at "redundant" traces in the mined models with respect to the log used as input for the discovery algorithm (see e.g.[10,13]). Using the inverse functions l_1^{-1}, l_2^{-1}, every trace of $\mathfrak{L}_1$ and $\mathfrak{L}_2$ can be mapped to a set of traces built from the events from $\mathcal{E}$. Taking the union of the sets obtained with l_1^{-1}, l_2^{-1} over the traces of the languages, we obtain comparable languages and can conclude whether the relabeling function results in a model that is more precise with respect to the original log.

Fitness and simplicity of the models depend mostly on the performance of the process discovery algorithm, and not on the choice of the relabeling function. Precision defined in terms of events of the original universe $\mathcal{E}$ of events is however highly dependent on the appropriateness of the relabeling function: choosing a more refined relabeling function can increase the precision by eliminating the behavior that would be allowed in the model discovered with a more abstract relabeling function. Generalization can potentially suffer as the result of a higher precision.

3.1. Label Refinement Quality

The comparison of the languages generated by models is not feasible due to its complexity; for many classes of process models, including Petri nets, the problem of language inclusion is just not decidable. Therefore, we need a different, practical approach to deciding on the usefulness of a relabeling function refinement. We start with discussing the usefulness by comparing the discovered models.

Consider event log L, relabeling functions l_1, l_2, l_3 such that $l_2 \prec l_1 \wedge l_3 \prec l_1$, and event logs $L_1 = l_1(L), L_2 = l_2(L), L_3 = l_3(L)$. Let the N_1, N_2, N_3 in Figure 4 be the Petri nets obtained by applying process discovery to L_1, L_2, L_3 respectively. The square inside the transition between places p_3 and p_4 indicates that it is a subprocess.

We can see that refinement l_2 does not lead to a meaningful interpretation of b as b_1 and b_2, since the behavior of the model is not related to the choice between b_1 and b_2: transitions labeled with b_1 and b_2 have the same input and output places. Refinement l_2 does not provide new insight and unnecessarily harms the understandability of the Petri net by cre-

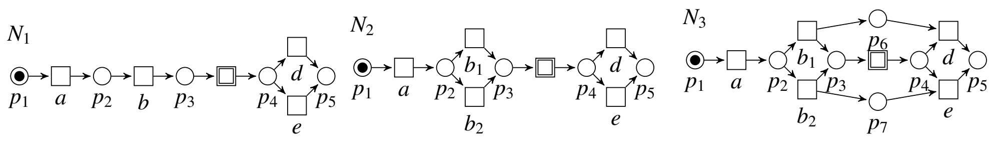


Fig. 4: N_2 is a non-useful refinement and N_3 is a useful refinement of N_1.

Table 2: Log-based ordering relations and their use by process discovery algorithms

Ordering relation	Miners using the relation
Direct successor	a miner[17], a^{++} miner[22], Multi-phase miner[18], Heuristics miner[21]
Length-two loop	a^{++} miner[22], Multi-phase miner[18], Heuristics miner[21]
Direct/indirect successor	a^{++} miner[22], Heuristics miner[21]

Table 3: A Log statistic in contingency table form

	a_1	a_2	a
+	$\#^+_{L_2,s}(a_1,b)$	$\#^+_{L_2,s}(a_2,b)$	$\#^+_{L_1,s}(a,b)$
−	$\#^-_{L_2,s}(a_1,b)$	$\#^-_{L_2,s}(a_2,b)$	$\#^-_{L_1,s}(a,b)$

ating more transitions then needed. On the other hand, l_3 results in gain of precision, as $\mathfrak{L}(N_3)$, does not contain $\langle a, b_1, e\rangle$ and $\langle a, b_2, d\rangle$, while N_1 does not distinguish between b_1 and b_2, which suggests that both types of traces are possible.

4. Evaluation Method for Label Refinements for Process Models

In the previous section we showed that we can compare the usefulness of a label refinement by inspecting the Petri net obtained with process discovery. A naive way to evaluate label refinement would be to apply process discovery to all possible label refinements. The number of possible label refinements to consider can however be large and process discovery is a computationally expensive task. Therefore, this naive approach quickly becomes computationally infeasible. We now present a way to estimate the usefulness of a label refinement based on statistics and log relations.

Algorithm 1 shows the steps of the label refinements evaluation method. The evaluation method consists of an entropy-based component that measures whether a label refinement makes the log statistics more unbalanced, and a statistical test that tests whether there is a label statistic that tests whether the label refinement makes a statistically significant difference to at least one of the log statistics. In the following two sections we described the entropy-based measure and the statistical testing respectively.

4.1. Log Statistics

Event ordering patterns are crucial to most process discovery algorithms. Table 2 provides an overview of well-known log-based ordering relations described in process discovery literature[17,18,22,21] and provides examples. Let L be an event log. Let $b, c \in A_L$. Formal definitions of these log-based ordering statistics are as follows:

- $\#^+_{L,>}(b,c)$ is the number of occurrences of b in the traces of L that are directly followed by c, i.e. in some $\sigma \in L, i \in \{1,\dots,|\sigma|\}$ we have $[\sigma(i)]_a = b$ and $[\sigma(i+1)]_a = c$ (*direct successor*), $\#^-_{L,>}(b,c)$ is the number of occurrences of b which are not directly followed by c;
- $\#^+_{L,>>}(b,c)$ and $\#^-_{L,>>}(b,c)$ is the number of occurrences of b that are, respectively, are not, followed by c: for a trace $\sigma \in L$ and $i \in \{1,\dots,|\sigma|\}$, and $[\sigma(i)]_a = [\sigma(i+2)]_a = b$ and $[\sigma(i+1)]_a = c$ and $b \neq c$ (length-two loops);
- $\#^+_{L,>>>}(b,c)$ and $\#^-_{L,>>>}(b,c)$ is the number of occurrences of b that are, respectively are not, eventually followed by c: for a trace $\sigma \in L, i, j \in \mathbb{N}$ with $i < j$, $[\sigma(i)]_a = b$ and $[\sigma(j)]_a = c$ (direct or indirect successor).

In the general sense, let $\#^+_{L,s}(b,c)$ and $\#^-_{L,s}(b,c)$ be the count of the number of b's that do, respectively do not, satisfy relation s in log L with respect to c.

Let L be an event log. Let l_1 and l_2 be two relabeling functions that are to be compared, such that $l_2 \prec^= l_1$. Let $L_1 = l_1(L)$ and $L_2 = l_2(L)$. Let l_1 and l_2 have the property $\{a_1, a_2 \in A_{L_2})|\exists_{\sigma_1,\sigma_2 \in L} : l_1(\sigma_1) = \lambda a \wedge l_1(\sigma_2) = \lambda' a \wedge l_2(\sigma_1) = \zeta a_1 \wedge l_2(\sigma_2) = \zeta' a_2\} \neq \emptyset$, that is, l_2 refines activity a into distinct activities a_1 and a_2. The difference in control flow between a_1 and a_2 can be expressed as the dissimilarity in log-based ordering statistics between event label a_1 and $b \in A_{L_2} \setminus \{a_1, a_2\}$ on the one hand, and a_2 and b on the other hand. Each log-based ordering statistics of a_1 and a_2 with regard to any other activity b can be formulated in the form of a contingency table, as shown in Table 3.

Table 4: Contingency tables for comparing the behavior of the two refined labels

Directly follows

	Tossing & turning	Getting up	Bedroom motion
$\overset{+}{\rightarrow}$	0	0	0
$\overset{-}{\rightarrow}$	16	5	21

Directly precedes

	Tossing & turning	Getting up	Bedroom motion
$\overset{+}{\rightarrow}$	0	5	5
$\overset{-}{\rightarrow}$	16	0	16

Eventually follows

	Tossing & turning	Getting up	Bedroom motion
$\overset{+}{\rightarrow}$	0	0	0
$\overset{-}{\rightarrow}$	16	5	21

Eventually precedes

	Tossing & turning	Getting up	Bedroom motion
$\overset{+}{\rightarrow}$	16	5	21
$\overset{-}{\rightarrow}$	0	0	0

4.2. Information Gain

The binary entropy function, $H_b(p) = -p\log_2 p - (1-p)\log_2(1-p)$, where $0\log_2 0 = 0$, is a measure of uncertainty. Applied on a log statistic, the binary entropy function represents a degree of nondeterminism. Nondeterministic, unbalanced, log statistics are a helpful to process discovery algorithms that operate of log statistics, as it provides low uncertainty to the mining algorithm. Low entropy in the log statistics indicate high predictability of the process, making it easier for process discovery algorithms to return a sensible process model.

Consider the contingency tables in Table 4, based on log statistics obtained from Table 1 between the events labeled *Tossing & turning* and *Getting up* and the events labeled *Living room motion*. On the right hand side of the table, separated by the bar, are the log statistics of the before-split label in the before-split log. All five events with label *Getting up* directly precede an event with label *Living room motion*, while all sixteen events with label *Tossing & turning* are not directly preceded by *Living room motion*. Furthermore, all events with refined labels do not directly or eventually follow an event with label *Living room motion*, and all events with refined labels do eventually precede an event with label *Living room motion*.

Log statistics with a high degree of non-determinism, like the directly precedes statistic of the bedroom motion events before the split, might confuse a mining algorithm as there is no clear structure here: the *Bedroom motion* event might directly precede *Livingroom motion*, but most of the time it does not. After the split we see a completely deterministic directly precedes statistic, where *Tossing & turning* never and *Getting up* always directly precedes *Livingroom motion*. This increased determinism is reflected by the entropy of the directly precedes statistic before and after the split. Before the split we have $-\frac{5}{5+16}\log_2\frac{5}{5+16} - \frac{16}{5+16}\log_2\frac{16}{5+16} = 0.7919$ bit of entropy in the directly precedes statistic, compared to $-\frac{0}{0+16}\log_2\frac{0}{0+16} - \frac{16}{0+16}\log_2\frac{16}{0+16} = 0$ bit of entropy for *Tossing & turning* and $-\frac{0}{0+5}\log_2\frac{0}{0+5} - \frac{5}{0+5}\log_2\frac{5}{0+5} = 0$ bit of entropy for *Getting up*. The conditional entropy of the log statistic after the split is the weighted average of the entropy of the labels created in the split, which is $-\frac{16}{21}0 \times \frac{5}{21}0 = 0$. The information gain of this label split with regard to the directly precedes *Livingroom motion* statistic is equal to the total entropy of the log statistic prior to the split, minus the conditional entropy after the split, this $0.7919 - 0 = 0.7919$. Relative information gain[6] is a metric that provides insight in the ratio of bits of entropy reduced by a refinement, and can be calculated by dividing the information gain by the before-split entropy. The relative information gain of the directly precedes *Livingroom motion* statistic is $\frac{0.7919}{0.7919} = 1$. Figure 2 shows the effect of this label refinement on the resulting Petri net obtained by process discovery.

So far we have calculated the Relative information gain for a single log statistic. A label refinement however can have impact on multiple log statistics at once. We need a measure that integrates the information gain values of all log statistics to express the quality of a label refinement with respect to the determinism of the log statistics. We therefore sum over the entropy of all log statistics before the label split to obtain the total before-split entropy. We sum over the conditional entropies of all log statistics after the label split to obtain the total after-split entropy. Information Gain and Relative information gain are calculated as before. We let $relative\_information\_gain(L_1, L_2)$ be the function that returns the Relative information gain based on the pre-split log L_1 and post-split log L_2, where the set of refined label pairs in L_2 from which the log statistics are used corresponds to $\{a_1, a_2 \in A_{L_2} | \exists_{\sigma_1,\sigma_2 \in L} : l_1(\sigma_1) = \lambda a \wedge l_1(\sigma_2) = \lambda' a \wedge$, with the a the corresponding label in L_1.

4.3. Statistical Testing

Relative information gain can be high by chance for a refinement when the generated refined labels are infrequent. Statistical testing of log statistic differences in addition to calculating relative information gain enables us to distinguish

Algorithm 1 Algorithm of the label refinement statistical evaluation method

```
Input:  Event log $L$, Relabeling functions $l_1$ and $l_2$ such that $l_2 <^= l_1$,
Output:  the Relative information gain of $l_1$ w.r.t $l_2$,
Parameters:  Set of log-based ordering statistics $S$,
Significance level $\alpha$.

    all_significant_different = true; $L_1$=$l_1(L)$; $L_2$=$l_2(L)$;
    split_set = $\{a_1, a_2 \in A(L_2) | \exists_{\sigma_1,\sigma_2 \in L} : l_1(\sigma_1) = \lambda a \wedge l_1(\sigma_2) = \lambda' a \wedge\ l_2(\sigma_1) = \zeta a_1 \wedge l_2(\sigma_2) = \zeta' a_2\}$;
    For each $a_1, a_2 \in$ split_set:
        $pair\_significant\_different$ = false;
        For each $\{b \in A(L_2) \setminus \{a_1, a_2\}\}$:
            For each $s \in S$ :
                p = $fisher\_test(\#^{+}_{L_2,s}(a_1,b), \#^{-}_{L_2,s}(a_1,b), \#^{+}_{L_2,s}(a_2,b), \#^{-}_{L_2,s}(a_2,b))$;
                If($p < \alpha$) pair_significant_different = true;
        If(!$pair\_significant\_different$)
            $all\_significant\_different$ = false;
    If($all\_significant\_different$)
        return $relative\_information\_gain(L_1, L_2)$;
    Else return 0.0;
```

between information gain obtained by chance and actual information gain. Fisher's exact test[5] is a statistical significance test for the analysis of contingency tables. When applied to the table above, it calculates a p-value for the null hypothesis that a_1 and a_2 events are equally likely to hold log relation s with regard to label b. Fisher's exact test assumes individual observations to be independent and row and column totals to be fixed. Independence of individual observations might be affected by the grouping of events in traces. In this paper we consider individual observations independence to be working assumption. The test was designed for experiments where both the row and column totals where conditioned. In our setting, the column totals are conditioned by the relabeling function, as the number of events of each label depends on the relabeling. The row totals however, are not conditioned and are an observation. Fisher's exact test is not strictly speaking exact when one or both of the row or column totals are unconditioned, but will instead be slightly conservative[9], meaning that the probability of the p-value being less than or equal to the significance level when the null hypothesis is true is less than the significance level. Fisher's exact test is computationally expensive for large numbers of observations. For large sample sizes, either the χ^2 test of independence or the G-test of independence can be used, which are both found to be inaccurate for small sample sizes. A popular guideline is to not use the χ^2 test of independence or the G-test for samples sizes less than one thousand[9]. The computational complexity of the evaluation procedure is $O(|S| \times |A(L)| \times |split\_set|)$. Many process discovery algorithms are exponential in the number of labels[16]. Based on this we can conclude that statistical evaluation of label refinements is computationally less expensive than checking label refinement usefulness through process discovery.

4.4. Correcting for Multiple Testing

The computational complexity indicates the number of hypothesis tests performed. When a large set of potential label refinements is evaluated, the evaluation method described is susceptible to the repeated testing problem. The larger the set of hypotheses tested, the higher the probability of incorrectly rejecting the null hypothesis in at least one of the hypothesis tests. Applying a Bonferroni correction[3,4] to the hypothesis tests performed in the statistical evaluation method of label refinements keeps the familywise error rate constant.

4.5. Example Case

Consider the event log in Table 1 and imagine a scenario where a home care worker knows from experience that the elderly always sets his alarm clock at 8:30 AM. Based on such expert knowledge we are able to define a label refinement such that all bedroom movements after 8:30 AM are considered as *Getting up* events, while all other bedroom movements are considered to be *Tossing & turning* events. The rightmost column shows the refined labels obtained through this expert relabeling function. To evaluate the usefulness of this label refinement from a process model point of view, we apply the statistical evaluation method described in Section 4. As parameters we set the significance level threshold to the frequently used value of 0.01.

Table 5 shows the outcome of the statistical tests performed as part of the label refinement usefulness evaluation. Four hypothesis tests have been performed, after Bonferroni correction each hypothesis test is tested at significance level $\frac{0.01}{4} = 0.0025$. The direct following statistic of *Tossing & turning* and *Getting up* with *Living room motion* is statistically significantly given this significance level. The label refinement constructed with expert knowledge is found to be a useful label refinement through statistical evaluation.

Table 5: Results of the statistical tests for the evaluation of label refinement usefulness

Log statistic	P-value
Directly follows	1
Directly precedes	4.91×10^{-5}
Eventually follows	1
Eventually precedes	1

5. Real life evaluation

We apply our label refinement evaluation method to a set of candidate label refinements on the Van Kasteren smart home environment data set[19] in order to illustrate the effects of label splits in the context of process mining of real life processes. The van Kasteren data set consists of 1285 events divided over fourteen different sensors. Events are segmented in days from midnight to midnight, to define cases in the event log. The candidate set of label refinements consists of splitting each of the fourteen event types *t* into two event types based on the their time in the day, such that *t* events where the time since the start of the day is smaller than the median for *t* are separated from *t* events where it is equal to or larger than the median. Figure 5 shows the *dependency graph* obtained with the Heuristics Miner[21]. A *dependency graph* depicts causal relations between activities that meet a certainty threshold. A *dependency graph* can be directly converted into a Petri net[21], however, for the sake of readability we included the *dependency graphs* instead of the Petri nets. The precision[10] of the Petri net corresponding to Figure 5 is 0.56 on a scale from 0 to 1.

Out of the fourteen candidate label refinements, two label refinements are selected by our approach. The first label refinement found is the split of *Hall-bathroom door* into *Hall-bathroom door_1* and *Hall-bathroom door_2*, with a timestamp below, respectively above or equal to the median time in the day of *Hall-bathroom door* events. The resulting labels of this refinement are statistically significantly different in terms of their eventually follows relation with *Front door* (p-value: 3.06×10^{-26}) and their eventually follows relation with *Plates cupboard* (p-value: 3.66×10^{-23}) and Microwave 1.85×10^{24}. The relative information gain on the whole event log caused by this label refinement is 3.47%. Figure 6 shows a Heuristics Net mined with the Heuristics Miner[21] on the van Kasteren log with the refined *Hall-bathroom door* label. The model discovered on the log with this label refinement (Figure 6) has a precision of 0.69, up from 0.53 without the refinement. The increased precision shows that the label refinement helps restricting the share of behavior allowed

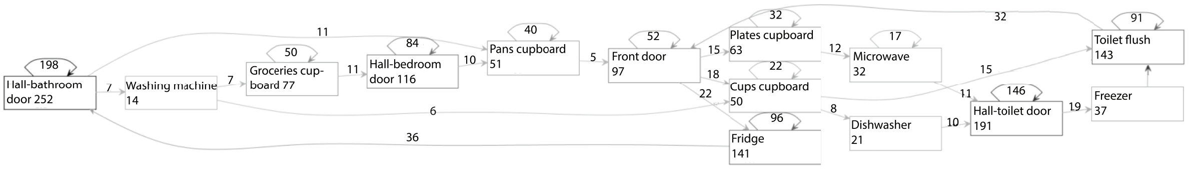


Fig. 5: Heuristics net showing original van Kasteren data set

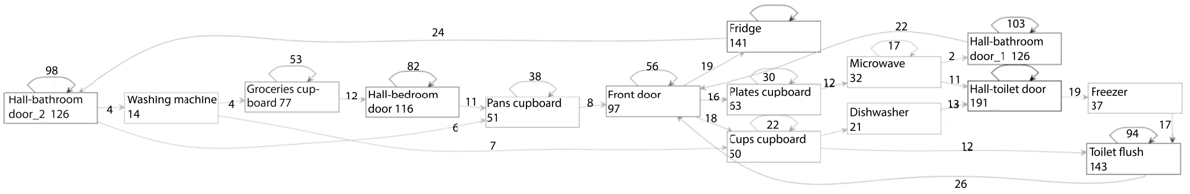


Fig. 6: Heuristics net showing the label refinement on *hall-bathroom door* on the van Kasteren data set

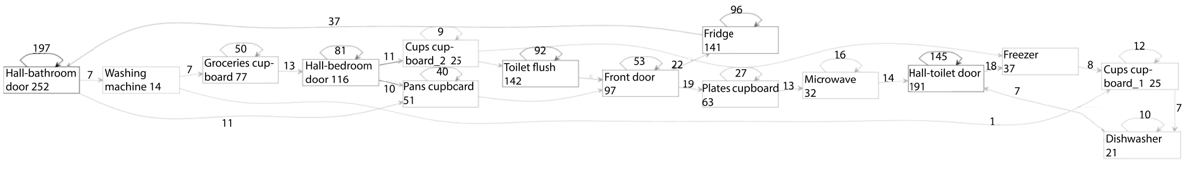


Fig. 7: Heuristics net showing the label refinement on *cups cupboard* on the van Kasteren data set

by the model that is not covered by the event log. The second label refinement found is the split of *Cups cupboard* into *Cups cupboard_1* and *Cups cupboard_2*. The resulting labels of this refinement are statistically significantly different in terms of their eventually precedes relation with *Groceries cupboard* (p-value: 2.53×10^{-34}) and their eventually follows relation with *Fridge* (p-value: 2.2×10^{-22}). The relative information gain on the whole event log caused by this label refinement is 0.53%. Figure 7 shows a Heuristics Net mined with the Heuristics Miner on the van Kasteren log with the refined *Cups cupboard* label, of which the precision is 0.61, up to 0.53 without the refinement. The label refinement with higher information gain also results in a higher improvement in terms of precision, which is in agreement with the intuition that more deterministic log statistics help the miner in mining structured, non-flower-like, models.

6. Conclusion & Future Work

We have provided a theoretical and conceptual notion of when label refinements and abstractions are useful from a process discovery point of view. Based on this notion of usefulness, we have shown a framework based on statistics and information theory to evaluate the usefulness of a label refinement or abstraction. In addition, we have shown the applicability of this statistical framework through a real life smart home case, where our method selected two label refinements out of a larger candidate set that increased the precision of the resulting process model. Methods for automatic inference of useful label refinements from event attributes are still to be explored. Such methods may generate a set of candidate label refinements, after which the statistical evaluation method described in this paper can be used to select the most promising label refinement from a set of candidate label refinements.

References

1. J. Brank, M. Grobelnik, and D. Mladenic. A survey of ontology evaluation techniques. In *Proceedings of the conference on data mining and data warehouses*, pages 166–170, 2005.
2. L. Chen, J. Hoey, C. Nugent, D. Cook, and Z. Yu. Sensor-based activity recognition. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 42(6):790–808, 2012.
3. O. J. Dunn. Estimation of the medians for dependent variables. *The Annals of Mathematical Statistics*, 30(1):192–197, 1959.
4. O. J. Dunn. Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293):52–64, 1961.
5. R. A. Fisher. *Statistical methods for research workers*. Number 5. Genesis Publishing Pvt Ltd, 1934.
6. S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.
7. J. Li, D. Liu, and B. Yang. Process mining: Extending α-algorithm to mine duplicate tasks in process logs. In *Advances in Web and Network Technologies, and Information Management*, volume 4537 of *LNCS*, pages 396–407. Springer Berlin Heidelberg, 2007.
8. A. Maedche. *Ontology learning for the semantic web*, volume 665. Springer Science & Business Media, 2012.
9. J. H. McDonald. *Handbook of biological statistics*, volume 2. Sparky House Publishing Baltimore, MD, 2009.
10. J. Muñoz Gama and J. Carmona. A fresh look at precision in process conformance. In *Business Process Management*, volume 6336 of *LNCS*, pages 211–226. Springer Berlin Heidelberg, 2010.
11. J. R. Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.
12. W. Reisig and G. Rozenberg. *Lectures on Petri nets I: basic models: advances in Petri nets*, volume 1491. Springer Science & Business Media, 1998.
13. A. Rozinat and W. M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.
14. T. Sztyler, J. Völker, J. Carmona, O. Meier, and H. Stuckenschmidt. Discovery of personal processes from labeled sensor data–an application of process mining to personalized health care. In *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data, ATAED*, pages 22–23, 2015.
15. W. M. P. van der Aalst. *Process mining: discovery, conformance and enhancement of business processes*. Springer Science & Business Media, 2011.
16. W. M. P. van der Aalst. Distributed process discovery and conformance checking. In *Fundamental Approaches to Software Engineering*, LNCS, pages 1–25. Springer, 2012.
17. W. M. P. van der Aalst, A. J. M. M. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
18. B. F. van Dongen and W. M. P. van der Aalst. Multi-phase process mining: Building instance graphs. In *Conceptual Modeling–ER 2004*, volume 3288 of *LNCS*, pages 362–376. Springer, 2004.
19. T. van Kasteren, A. Noulas, G. Englebienne, and B. Kröse. Accurate activity recognition in a home setting. In *Proceedings of the 10th International Conference on Ubiquitous Computing*, pages 1–9. ACM, 2008.
20. S. K. Vanden Broucke. *Advanced in Process Mining: Artificial Negative Events and Other Techniques*. PhD thesis, KU Leuven, 2014.
21. A. J. M. M. Weijters and J. T. S. Ribeiro. Flexible heuristics miner (fhm). In *Proceedings of the 2011 IEEE Symposium on Computational Intelligence and Data Mining*, pages 310–317. IEEE, 2011.
22. L. Wen, J. Wang, and J. Sun. Detecting implicit dependencies between tasks from event logs. *Frontiers of WWW Research and Development-APWeb 2006*, pages 591–603, 2006.