Interest-Driven Discovery of Local Process Models

Niek Tax[1], Benjamin Dalmas[2], Natalia Sidorova[1], Wil M.P. van der Aalst[1], and Sylvie Norre[2]

[1] Eindhoven University of Technology, Department of Mathematics and Computer Science, P.O. Box 513, 5600MB Eindhoven, The Netherlands
{n.tax,n.sidorova,w.m.p.v.d.aalst}@tue.nl
[2] Clermont-Auvergne University, LIMOS CNRS UMR 6158, Aubière, France
{benjamin.dalmas,sylvie.norre}@isima.fr

Abstract. Local Process Models (LPM) describe structured fragments of process behavior occurring in the context of less structured business processes. Traditional LPM discovery aims to generate a collection of process models that describe highly frequent behavior, but these models do not always provide useful answers for questions posed by process analysts aiming at business process improvement. We propose a framework for *goal-driven LPM discovery*, based on utility functions and constraints. We describe four scopes on which these utility functions and constrains can be defined, and show that utility functions and constraints on different scopes can be combined to form composite utility functions/constraints. Finally, we demonstrate the applicability of our approach by presenting several actionable business insights discovered with LPM discovery on two real life data sets.

Keywords: Process Discovery, Pattern Mining, Goal-driven Data Mining

1 Introduction

Process Mining [1] has emerged as a new discipline aiming at the improvement of business processes through the analysis of event data recorded by information systems. Such event logs capture the different steps (events) that are recorded for each instance of the process (case), and record for each of those steps what was done, by whom, for whom, where, when, etc. Process discovery, one of the main tasks in the process mining field, is concerned with the discovery of an interpretable model from this event log such that this model accurately describes the process. The process models obtained give insight in what is happening in the process, and can be used as a starting point for different types of further analysis, e.g. bottleneck analysis [14], and checking compliance with rules and regulations [18]. Many algorithms have been proposed for process discovery, e.g., [3,13,11,12,5] (see Section 2).

One type of process discovery is Local Process Model (LPM) discovery [22,21], which is concerned with the discovery of a ranking of process models, where each individual LPM describes only a subset of the process activities. Each LPM

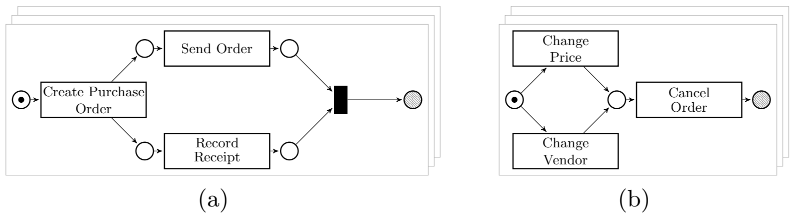


Fig. 1. *(a)* A frequent Local Process Model with low utility, and *(b)* a non-frequent Local Process Model with high utility.

describes one frequent pattern in a process model notation (e.g. Petri net [15], BPMN [16], or UML activity diagram [7]). This gives an LPM the full expressive power of the respective process model notation and allows it to represent more complex non-binary relations that cannot be expressed in declarative process models. LPMs aim to describe frequent local pieces of behavior, therefore, LPMs can be seen as a special form of *frequent pattern mining* [6] where each pattern is a process model. However, LPMs are not limited to subsequences [20] or episodes [10].

A recent trend in the frequent pattern mining field is to incorporate *utility* into the pattern selection framework, such that not just the most frequent patterns are discovered, but instead patterns are discovered that address typical business concerns, such as the patterns that represent high financial costs. Shen et al. [19] were the first to introduce utility-based itemset mining. Since then, utility-based pattern mining has spread to different types of pattern mining, including sequential pattern mining [23]. Utility-based pattern mining techniques assume that the value of each data point with respect to a certain business question is known and then discover the optimal patterns in terms of the value that they represent.

Imagine a Purchase-to-Pay process and as a process analyst we are interested in where in the process the employees of the company spend most of their time. Figure 1 shows two LPMs that could be discovered from such an event log. Figure 1a is a process fragment that describes the creation of a purchase order, which is followed by both the sending of the order and the recording of the receipt in an arbitrary order. This process fragment is frequent, as they are required steps for each order. Figure 1b describes a process fragment where the order is canceled after the price or the vendor of the order is changed. Even though the process fragment of Figure 1b is likely to be infrequent, it will take considerable resources of the department as canceling an order is an undesired action and considerable time will be spend trying to prevent it. Existing support-based LPM discovery [22] would not be able to discover Figure 1b because of its low frequency, motivating the need for utility-based LPM discovery.

In this paper we propose a framework to discover LPMs based on their utility in the context of a particular business question. Furthermore, we give an extensive overview of utility functions and their relevance in a BPM context. The techniques described in this paper have been implemented in the *ProM* process mining framework [4] as part of the *LocalProcessModelDiscovery*[1] package.

[1] `https://svn.win.tue.nl/trac/prom/browser/Packages/LocalProcessModelDiscovery`

Algorithm	Formal/Informal	Global/Local
Declare Miner[13]	Formal	Global
Language-based regions [3]	Formal	Global
Inductive Miner [11]	Formal	Global
LPM Discovery [22]	Formal	Local
Fuzzy Miner [5]	Informal	Global
Episode Miner [10]	Informal	Local

Table 1. A classification of process discovery methods.

This paper is organized as follows. Section 2 describes related work. Section 3 introduces the basic concepts used in this paper. Section 4 introduces utility functions and constraints in the context of LPMs. In Section 5 we demonstrate utility-based LPM discovery on two real-life event logs and show that we can obtain actionable insights. We conclude the paper in Section 6.

2 Related Work

In this section we discuss two areas of related work. First we discuss existing work in process discovery and position Local Process Model (LPM) discovery in the process discovery landscape. Secondly, we discuss related work from the pattern mining field.

2.1 Process Discovery

Process discovery techniques can be classified in several dimensions. Some process discovery techniques discover *formal process models*, where the behavior allowed by the model is formally defined, while others discover *informal process models* with unclear semantics. Orthogonally, process discovery algorithms can be classified in *end-to-end (global)* techniques that produce models that describe the logged process executions fully from start to end, and *pattern-based (local)* techniques that produce models that describe the behavior of the log only partially. Table 1 provides a classification of some existing process discovery techniques, and shows that LPM discovery is the only technique available which provides models that are both formal and local.

2.2 Interest-driven Pattern Mining

LPMs are able to express a richer set of relations (e.g. loops, XOR-constructs, concurrency) than sequential pattern mining techniques, which are limited to the sequential constructs. The mining of patterns driven by the interest of an analyst is known in the pattern mining field as high-utility pattern mining. Several high-utility sequential pattern mining algorithms have been proposed [23,9,24]. Sequential pattern mining techniques take as input a *sequence database*, a concept which is closely related to *event logs* in process mining. Sequence databases consist of sequences of tasks (called *items*). Some pattern mining techniques additionally assume the timestamp of each item to be logged. Sequence databases are not as rich as events logs found in process mining, where many data attributes are logged both for events and cases.

A second approach to interest-driven pattern mining pattern is constraint-based sequential pattern mining. In contrast to high-utility pattern mining, which gives preference to more useful patterns, constraint-based pattern mining completely removes non-useful ones. Pei et al. [17] provides a categorization of pattern constraints, consisting of four types of constraints on the ordering of items in the pattern and two types of constraints on the timestamps of the items in the log that are instances of a pattern. This richer notion of an event log allows us to define utility and constraints in a more flexible way (e.g. using event or case properties) compared to high-utility pattern mining techniques which define utility as mapping from an item type to utility and constraint-based pattern mining which limit the utility definition to ordering information and the time domain.

3 Basic Definitions

In this section we introduce notations related to event logs, Petri nets, and Local Process Models (LPMs), which are used in later sections of this paper.

3.1 Events, Traces, and Event Logs

For a given set A, A^* denotes the set of all sequences over A and $\sigma = \langle a_1, a_2, \dots, a_n \rangle$ is a sequence of length n, where $\sigma(i) = a_i$. $\langle\rangle$ is the empty sequence and $\sigma_1\sigma_2$ is the concatenation of sequences σ_1 and σ_2. $\sigma\upharpoonright_A$ is the projection of σ on A, e.g. $\langle a, b, c, a, b, c\rangle\upharpoonright_{\{a,c\}}=\langle a, c, a, c\rangle$.

Let $\mathcal{E}$ be the event universe, i.e., the set of all possible event identifiers. We assume that events are characterized by various properties, e.g., an event has a timestamp, corresponds to an activity, is performed by a particular resource, etc. We do not impose a specific set of properties, however, we assume that two of these properties are the activity and timestamp of an event, i.e., there is a function $\pi_{activity} : \mathcal{E} \rightarrow \mathcal{A}$ that assigns to each event an activity from a finite set of process activities $\mathcal{A}$, and a function $\pi_{time} : \mathcal{E} \rightarrow \mathcal{T}$ that maps each event to the time domain $\mathcal{T}$. In general, we write $\pi_p(e)$ to obtain the value of any property p of event e.

An *event log* is a set of events, each linked to one trace and globally unique, i.e., the same event cannot occur twice in a log. A trace in a log represents the execution of one case. A *trace* is a finite non-empty sequence of events $\sigma{\in}\mathcal{E}^*$ such that each event appears only once and time is non-decreasing, i.e., for $1{\leq}i{<}j{\leq}|\sigma| : \sigma(i){\neq}\sigma(j)$ and $\pi_{time}(\sigma(i)){\leq}\pi_{time}(\sigma(j))$. $\mathcal{C}$ is the set of all possible traces. An *event log* is a set of traces $L{\subseteq}\mathcal{C}$ such that each event appears at most once in the entire log.

Given a trace and a property, we often need to compute a sequence consisting of the value of this property for each event in the trace. To this end, we lift the function π_p that maps an event to the value of its property p, in such a way that we can apply it to sequences of events (traces).

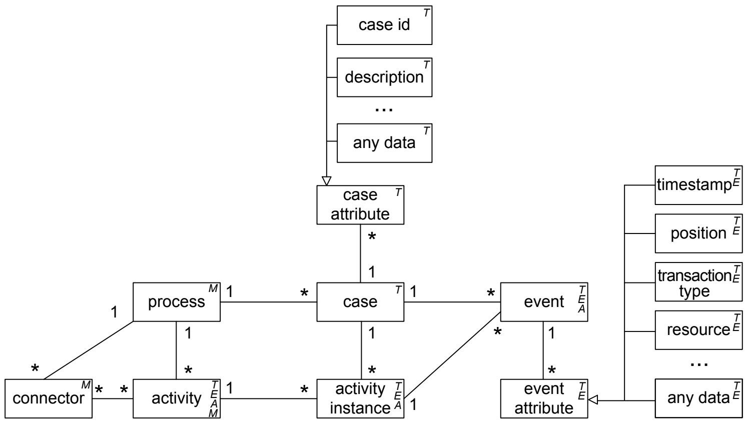


Fig. 2. Basic logging concepts conceptualized in a class diagram.

A partial function $f \in A \nrightarrow Y$ with domain $dom(f)$ can be lifted to sequences over A using the following recursive definition: (1) $f(\langle\rangle) = \langle\rangle$; (2) for any $\sigma \in A^*$ and $x \in A$:

$$f(\sigma \cdot \langle x \rangle) = \begin{cases} f(\sigma) & \text{if } x \notin dom(f), \\ f(\sigma) \cdot \langle f(x) \rangle & \text{if } x \in dom(f). \end{cases}$$

$\pi_{activity}(\sigma)$ transforms a trace σ to a sequence of its activities. For example, for trace $\sigma{=}\langle e_1, e_2 \rangle$ with $\pi_{activity}(e_1){=}a$ and $\pi_{activity}(e_2){=}b$: $\pi_{activity}(\sigma){=}\langle a, b \rangle$.

Traces themselves can also have properties, e.g., a case represented by trace $\sigma{\in}L$ can be associated with a branch of the company where the process was executed. We write $\phi_p(\sigma)$ to obtain the value of any property p of a case represented by trace σ.

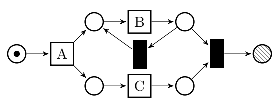
The class diagram in Figure 2 is obtained from [1] and it conceptualizes basic logging concepts in process mining. For now, ignore the annotations on the right side of the classes (*T,E,A,M*). A process model consists of a set of process activities and a *connector* connects elements of the process models to those activities.

3.2 Petri nets

A process model notation that is frequently used in the process mining area is the Petri net.

Definition 1 (Labeled Petri net). *A labeled Petri net $N = \langle P, T, F, \Sigma_M, \ell \rangle$ 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, Σ_M is a finite set of labels representing activities, and $\ell : T \nrightarrow \Sigma_M$ is a labeling function that assigns a label to transitions. Unlabeled transitions, i.e., $t{\in}T$ with $t{\notin}dom(l)$, are referred to as τ-transitions, or invisible transitions.*

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 $n\bullet = \{n' | (n, n'){\in}F\}$.



(a)

event id	activity	time	cost
1	A	26-3-2017 13:00	€ 100
2	B	26-3-2017 13:25	€ 500
3	X	26-3-2017 13:27	€ 60
4	B	26-3-2017 13:30	€ 400
5	C	26-3-2017 13:35	€ 100
6	C	26-3-2017 13:42	€ 500
7	A	26-3-2017 15:26	€ 300
8	B	26-3-2017 15:27	€ 50
9	C	26-3-2017 15:47	€ 100
10	B	26-3-2017 16:10	€ 250
11	B	26-3-2017 16:52	€ 300
12	X	26-3-2017 16:59	€ 10

(b)

$$\sigma = \langle A,B,X,B,C,C,A,B,C,B,B,X\rangle$$

$$\sigma\restriction_{\{A,B,C\}} = \langle \underbrace{}_{\lambda_1}\underbrace{A,B,B,C}_{\gamma_1},\underbrace{C}_{\lambda_2},\underbrace{A,B,C,B,B}_{\gamma_2}\underbrace{}_{\lambda_3}\rangle$$

$$\Gamma_{\sigma,LPM} = \langle A,B,B,C,A,B,C,B,B\rangle$$

(c)

Fig. 3. *(a)* Example accepting Petri net APN_1. *(b)* Trace σ of an event log L. *(c)* Segmentation of σ on APN_1.

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 $p \in P$. 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 $M \xrightarrow{t} M'$. $M_1 \xrightarrow{\sigma} M_2$ indicates that M_2 can be reached from M_1 through a firing sequence $\sigma \in T^*$.

Often it is useful to consider a Petri net in combination with an initial marking and a set of possible final markings. This allows us to define the language accepted by the Petri net and to check whether some behavior is part of the behavior of the Petri net (can be replayed on it).

Definition 2 (Accepting Petri net). *An* accepting Petri net *is a triple $APN{=}(N, M_0, MF)$, where N is a labeled Petri net, $M_0 \in \mathbb{N}^P$ is its initial marking, and $MF \subseteq \mathbb{N}^P$ is its set of possible final markings, such that $\forall_{M_1,M_2 \in MF} M_1 \not\subseteq M_2$. A sequence $\sigma \in T^*$ is called a* trace *of an accepting Petri net APN if $M_0 \xrightarrow{\sigma} M_f$ for some final marking $M_f \in MF$. The* language $\mathfrak{L}(APN)$ *of APN is the set of all label sequences belonging to its traces, i.e., if σ is a trace, then $l(\sigma) \in \mathfrak{L}(APN)$.*

Figure 3a shows an example of an accepting Petri net. Circles represent places and rectangles represent transitions. Invisible transitions (τ) are depicted as black rectangles. Places that belong to the initial marking contain a token and places belonging to a final marking contain a bottom right label f_i with i a final marking identifier, or are simply marked as ▨ in case of a single final marking. The language of this accepting Petri net, with $\Sigma_M = \{A, B, C\}$ is $\{\langle A, B, C\rangle, \langle A, C, B\rangle, \langle A, B, B, C\rangle, \langle A, B, C, B\rangle, \langle A, B, B, B, C\rangle, \ldots\}$. We refer the interested reader to [15] for a more thorough introduction of Petri nets.

3.3 Local Process Models

LPMs [22] are process models that describe the behavior seen in the event log only partially, focusing on frequently observed behavior. Typically, LPMs describe the behavior of only up to 5 activities. LPMs can be represented in any process modeling notation, such as BPMN [16], UML [7], or EPC [8]. Here we use Petri nets to represents LPMs. A technique to generate a ranked collection of LPMs through iterative expansion of candidate process models is proposed in [22]. The search space of process models is fixed, depending on the event log. We define $\mathit{LPMS}(L)$ as the set of possible LPMs that can be constructed for given event log L. We refer the reader to [22] for a detailed description of search space $\mathit{LPMS}(L)$.

To evaluate a given LPM on a given event log L, its traces $\sigma{\in}L$ are first projected on the set of activities Σ_M in the LPM, i.e., $\sigma'{=}\sigma{\upharpoonright}_{\Sigma_M}$. The projected trace σ' is then segmented into γ-segments that fit the behavior of the LPM and λ-segments that do not fit the behavior of the LPM, i.e., $\sigma'{=}\lambda_1\gamma_1\lambda_2\gamma_2\cdots\lambda_n\gamma_n\lambda_{n+1}$ such that $\gamma_i{\in}\mathfrak{L}(LPM)$ and $\lambda_i{\notin}\mathfrak{L}(LPM)$. We define $\Gamma_{\sigma,LPM}$ to be a function that projects trace σ on the LPM activities and obtains its subsequences that fit the LPM, i.e., $\Gamma_{\sigma,LPM} = \gamma_1\gamma_2\ldots\gamma_n$.

Let our LPM under evaluation be the Petri net of Figure 3a and let Figure 3b be our trace σ, with $\pi_{activity}(\sigma) = \langle A, B, X, B, C, C, A, B, C, B, B, X\rangle$. Projection on the activities of the LPM gives $\pi_{activity}(\sigma){\upharpoonright}_{\{A,B,C\}} = \langle A, B, B, C, C, A, B, C, B, B\rangle$. Figure 3c shows the segmentation the projected traces on the LPM, leading to $\pi_{activity}(\Gamma_{\sigma,LPM}) = \langle A, B, B, C, A, B, C, B, B\rangle$. The segmentation starts with an empty non-fitting segment λ_1, followed by fitting segment $\gamma_1{=}\langle A, B, B, C\rangle$, which completes one run through the model from initial to final marking. The second event C in σ cannot be replayed on LPM, since it only allows for one C and γ_1 already contains a C. This results in a non-fitting segment $\lambda_2{=}\langle C\rangle$. $\gamma_2{=}\langle A, B, C, B, B\rangle$ again represents a run through the model from initial to final marking, and $\lambda_3{=}\langle D\rangle$ does not fit the LPM. We lift segmentation function Γ to event logs, $\Gamma_{L,LPM}{=}\{\Gamma_{\sigma,LPM}|\sigma{\in}L\}$. An alignment-based [2] implementation of such a segmentation function Γ is proposed in [22].

We define $\mathit{activities}(a, L){=}|\{e{\in}\sigma|\sigma{\in}L{\wedge}\pi_{activity}(e){=}a\}|$ as a function that maps each activity in event log L to its occurrence frequency. $\mathit{activities}(L)$ represents the multiset with the frequencies of all activities that occur in L. $\mathit{events}(L) = \{e{\in}\sigma|\sigma{\in}L\}$ is the set of events of L. Note that functions *activities* and *events* can also be applied to $\Gamma_{L,LPM}$, which is itself an event log.

4 Local Process Model Constraints and Utility Functions

The discovery of Local Process Models (LPMs) can be steered towards the business needs of the process analyst by using a combination of *constraints* and *utility functions*. Constraints can for example be used to find fragments of process behavior that lead to a loan application getting declined, or to find fragments that only describe loan applications above €15 K and never those below. Utility functions can be used to discover LPMs that give insight in which fragments of process behavior are associated with high financial costs or long time delays.

Constraints are requirements that the LPM has to satisfy, therefore, we define constraints as functions that result in 1 when the requirement holds and is 0 when it does not hold. In a general sense they are defined as a function $c : X \rightarrow \{0, 1\}$ where X is the *scope* on which the function operates. We distinguish four different scopes on which X can be defined: *trace-level* (T), *event-level* (E), *activity-level* (A), and *model-level* (M). The class diagram in Figure 2 contains annotations which indicate which classes are included in each of the scopes.

Utility functions indicate to what degree an LPM is expected to be interesting and helpful to answer the business question of the process analyst at hand, and are defined as functions $f : X \rightarrow \mathbb{R}$. Like constraints, utility functions can be defined on the four scopes indicated in the class diagram of Figure 2.

Multiple utility functions and constraints can be combined to form one composite function that describes the total utility of an LPM given log L. Given constraints $c_1, c_2, \dots, c_n$ and utility functions $f_1, f_2, \dots, f_k$, the composite utility u is defined as:
$u(L, LPM) = \prod_{i=1}^{n} c_i(L, LPM) \cdot \sum_{j=1}^{k} f_j(L, LPM)$.
An LPM needs to satisfy all constraints c_i in order to have a utility larger than zero, i.e., $\exists c_i(L, LPM){=}0 \implies u(L, LPM){=}0$. Where the original LPM discovery method [22] selects and ranks LPMs based on support, utility allows for goal-oriented selection and ranking, i.e., LPMs are ranked based on their utility.

Utility functions and constrains of the trace, activity, and event-level scopes are all defined on a combination of an LPM and the event log. They define the utility of an LPM based on a function of log L and its fragments that fit LPM, given by $\mathit{\Gamma}_{L,LPM}$. The trace, event, and activity-level scopes differ in the perspectives on L and $\mathit{\Gamma}_{L,LPM}$ on which they operate. The model-level scope is defined solely on the LPM itself, and ignores the log argument L. Which concrete utility/constraint functions should be used depends on the business question of the process analyst. Figure 4 shows the arguments of utility and constraint functions on the four scopes. In the sections that follow we discuss definitions and properties of constraints and utility functions on each of these scopes and discuss several of their use cases.

4.1 Trace-level Constraints and Utility Functions

Trace-level utility functions are the most general class of utility functions that are defined on the event log. Trace-level utility functions calculate the utility of an LPM by aggregating the utility over the trace-fragments that fit the LPM behavior and allow the utility of fitting trace fragment to depend on the events in the trace fragment, their event properties, and properties of the case itself. Trace level-utility can for example be used to discover LPMs that describe the events that explain a high share of the total financial cost associated to a case, or a high share of the total running time of a case.

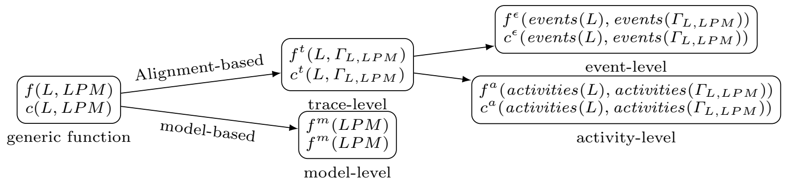

$f(L, LPM)$
$c(L, LPM)$
generic function
Alignment-based
model-based
$f^t(L, \mathit{\Gamma}_{L,LPM})$
$c^t(L, \mathit{\Gamma}_{L,LPM})$
trace-level
$f^m(LPM)$
$f^m(LPM)$
model-level
$f^e(events(L), events(\mathit{\Gamma}_{L,LPM}))$
$c^e(events(L), events(\mathit{\Gamma}_{L,LPM}))$
event-level
$f^a(activities(L), activities(\mathit{\Gamma}_{L,LPM}))$
$c^a(activities(L), activities(\mathit{\Gamma}_{L,LPM}))$
activity-level

Fig. 4. A tree of function specifications on different scopes.

A trace-level utility function is a function $f^t(L, \mathit{\Gamma}_{L,LPM})$ that indicates the utility of the fitting trace fragments in $\mathit{\Gamma}_{L,LPM}$. Consider the LPM of Figure 3a, the example log consisting of one trace σ in Figure 3b, and the segmentation of σ on the LPM of Figure 3c. Assume that σ has a case property *total_cost* which indicates the total cost of the trace. An example of a trace-level utility function is $f_1^t(L, \mathit{\Gamma}_{L,LPM}) = \sum_{\sigma' \in \mathit{\Gamma}_{L,LPM}} \sum_{e \in \sigma'} \frac{\pi_{cost}(e)}{\phi_{total\_cost}(\sigma')}$ which discovers LPMs that explain a large share of the total trace costs. Another example is a function $f_2^t(L, \mathit{\Gamma}_{L,LPM}) = \sum_{\sigma' \in \mathit{\Gamma}_{L,LPM}} \frac{1}{\pi_{time}(\sigma'(|\sigma'|)) - \pi_{time}(\sigma'(1))}$, which results in LPMs where the behavior described typically occurs in short time intervals. Trace-level constraints put requirements on the fitting trace segments $\mathit{\Gamma}_{L,LPM}$. All trace-level utility functions can be transformed into trace-level constraints by adding thresholds for a minimal or maximal value of the function.

4.2 Event-level Constraints and Utility Functions

Event-level utility functions and constraints can be used when the business question of the process stakeholder concerns certain event properties, but does not concern the trace-context of those events. Such utility functions can e.g. be used to discover LPMs that describe process behavior fragments with high financial cost. Constraints on this level can e.g. be used to limit LPMs that solely describe events that are executed by certain resources, or, in certain time periods. The trace-level scope is more expressive than the event-level scope, allowing the analyst to formulate more complex utility functions and constraints, but at the same time, it makes it harder to formulate such functions compared to the more restricted event-level scope.

An event-level utility function is a function $f^\epsilon(\mathit{events}(L), \mathit{events}(\mathit{\Gamma}_{L,LPM}))$ that indicates the utility of the fitting events in $\mathit{\Gamma}_{L,LPM}$. For the LPM, trace, and segmentation of Figure 3, an example event-level utility function is $f_1^\epsilon(\mathit{events}(L), \mathit{events}(\mathit{\Gamma}_{L,LPM})) = \sum_{e \in \mathit{events}(\mathit{\Gamma}_{L,LPM})} \pi_{cost}(e)$ which discovers LPMs with high costs. Note that L itself is also in the domain, allowing us to formulate a utility function that optimizes the share of utility explained per activity $f_2^\epsilon(\mathit{events}(L), \mathit{events}(\mathit{\Gamma}_{L,LPM})) = \sum_{a \in \Sigma_L} \frac{\sum_{e \in \mathit{events}(\mathit{\Gamma}_{L,LPM})} \pi_{cost}(e) \times (\pi_{activity}(e) = a)}{\sum_{e \in \mathit{events}(L)} \pi_{cost}(e) \times (\pi_{activity}(e) = a)}$.

An event level constraint is a function $c^\epsilon(\mathit{events}(L), \mathit{events}(\mathit{\Gamma}_{L,LPM}))$ and puts constraints on the events in $\mathit{events}(\mathit{\Gamma}_{L,LPM})$. An example is $c_1^\epsilon(\mathit{events}(L), \mathit{events}(\mathit{\Gamma}_{L,LPM})) = f_1^\epsilon(\mathit{events}(L), \mathit{events}(\mathit{\Gamma}_{L,LPM})) \geq 500$, results in LPMs with a total value of at least €500, or $c_2^\epsilon(\mathit{events}(L), \mathit{events}(\mathit{\Gamma}_{L,LPM})) = \forall_{e \in \mathit{\Gamma}_{L,LPM}} \pi_{cost}(e) \geq 100$, which results in LPMs that never represent events with a value of less than €100. Note that c_2^ϵ does not hold for our example trace and LPM, where the event with id 8 fits the LPM but only has value €50. Therefore, this LPM will not be found by the LPM discovery technique when we use constraint c_2^ϵ.

4.3 Activity-level Constraints and Utility Functions

Activity-level utility functions and constraints define the utility of an LPM based on the frequency of occurrence of each activity in log L and in $\mathit{\Gamma}_{L,LPM}$. Activity-level utility functions can for example be used by the process analyst to specify that he is more interested in some activities of high impact (e.g., lawsuits, security breaches, etc.) than in others, resulting in LPMs that describe the frequent behavior before and after such events. With activity-level constraints the process analyst can set a hard constraint on activity occurrences. The stakeholder/analyst can specify a function $f_1^a(\mathit{activities}(L), \mathit{activities}(\Gamma_{L,LPM}))$, which indicates how interested he is in each activity. Note that a utility function with equal importance assigned to all activities, i.e., $f^a(\mathit{activities}(L), \mathit{activities}(\Gamma_{L,LPM})) = |\mathit{activities}(\Gamma_{L,LPM})|$, results in support-based LPM discovery as described in [22].

Such utility functions can for example be used to get insight in the relations with other activities of some particular high-impact activities that the process analyst is interested in. Note that the occurrence of such activities in the log can be infrequent, in which case traditional LPM discovery without the use of utility functions and constraints is unlikely to return LPMs that concern those activities.

Adding a transition representing a zero-utility activity to an LPM can never increase its total utility, therefore, activity-level utility functions that assign zero value to some activities speed up discovery by limiting the LPM search space $LPMS(L)$, because LPMs with zero-utility activities do not have to be evaluated.

4.4 Model-level Utility Constraints and Utility Functions

Model-level utility functions and constrains can be used when a process analyst has preference or requirements for specific structural properties of the LPM. They have the form $f^m(LPM)$ and $c^m(LPM)$, i.e., they are independent of the log and dependent only on the LPM itself. When a process analyst for example wants to analyze the behavior that leads to the execution of a certain activity a which he is interested in, he can use a model-level constraint that enforces that all elements of $\mathcal{L}(LPM)$ end with a.

Generally we are interested in models that somehow represent the event log. Therefore, model-level utility functions and constraints are often not very useful on their own, but they become useful when combining them with utility functions and constraints on the event log, i.e., on the activity-level, event-level, or trace-level.

$\Gamma_{L,LPM}$ does not need to be calculated to determine whether a model-level constraint is satisfied, because model-level constraints are defined solely on the model. Therefore, model-level constraints can also be used to speed up LPM discovery by limiting the search space of models $LPMS(L)$ to its subspace for which the model-level constraints hold.

4.5 Composite Utility Functions

Utility functions and constraints on the different levels can be combined into one single utility function. In the beginning of this section we defined the utility of a LPM for a given event log in the following way:
$u(L, LPM) = \prod_{i=1}^{n} c_i(L, LPM) \cdot \sum_{j=1}^{k} f_j(L, LPM)$
The individual constraints c_i and the individual utility functions f_i can be defined on any of the levels discussed in the previous sections to form one composite, possibly multi-level, utility function. The total utility u is defined as an unweighted

sum over the individual utility functions f_i. Note that this still allows the process analyst to give priority to one utility function over another, as weights can be included as a part of the utility function itself by multiplying the utility function with a constant.

The presented framework of utility functions and constraints generalizes the method described in [22] and all LPM quality metrics presented there are instantiations of utility functions. An example is the *support* metric, which is an activity-level utility function. The minimum threshold for support as proposed in [22] is an example of an activity-level constraint. Another quality metric introduced in [22] is *determinism*, which is inversely proportional to the average number of enabled transitions in LPM during replay of the aligned event log. *Determinism* is an example of a composite utility function, consisting of a model-level and a trace-level component.

Note that the trace-level and event-level utility functions are not limited to continuous-valued properties. Many event logs contain ordinal event properties, such as *risk*, or, *impact*, which can take values such as *low*, *medium*, or *high*. Event-level utility functions can be applied to such event properties by specifying a mapping from the possible ordinal values to continuous values.

5 Case Studies

In this section we describe two case studies on real life event logs. The first log originates from an IT service desk of a large Dutch financial institution. The second log originates from the traffic fine handling process by the Italian police.

5.1 IT Service Desk

The IT service desk event log[2] is an event log that was made publicly available as part of the *Business Process Intelligence Challenge 2014*. The data set contains *incident* events which represent disruptions of IT-services within a large financial institution. Each incident event is associated with one or more *interactions*, which represent the calls and e-mails to the service desk agents that are related to this incident. When an incident occurs, it is assigned to an operator, who either solves the issue, or reassigns it to a colleague having more knowledge. For each incident event several properties are recorded, amongst others:

Service Component WBS This is a number that identifies the service component involved in the incident.

Configuration Item This contains the type (i.e., laptop, server, software application, etc.) of the service component that the incident concerns. Each *service component WBS* belongs to one *configuration item*.

Impact The impact of the service disruption to the customers as assessed by the operator. This property takes integer values from 1 to 5.

Closure code A code which classifies the cause of the service disruption, e.g., user error, software error, hardware error.

[2] http://dx.doi.org/10.4121/uuid:c3e5d162-0cfd-4bb0-bd82-af5268819c35

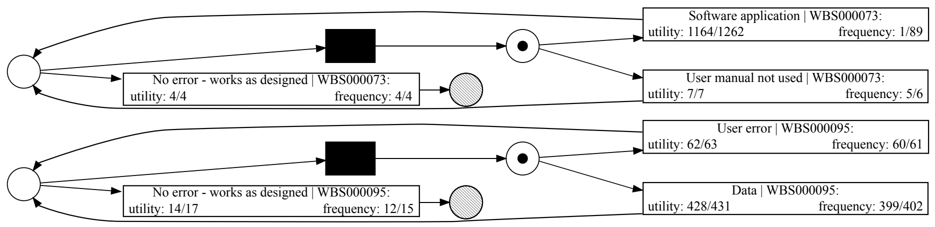


Fig. 5. Two LPMs discovered from the BPI'14 event log when using event property *number of interactions* as utility function.

CausedBy Incidents of a service component have another service component as root cause of the service disruption. This field contains the *service component WBS* number of the root cause service component.

Number of interactions The number of calls to the IT service desk that are related to this incident.

Number of reassignments The number of times that this incident was reassigned from one IT service desk operator to another.

We group together events by their *service component WBS* number, creating one case per service component consisting of incidents that these service components are involved in. We set the activity label of each incident event to a combination of the *closure code* and the *causedBy* attribute separated by the pipe character (|). The resulting event log contains 313 traces, 25,262 events and 944 activities.

Assume now that we are a process analyst concerned with the business question: "which process fragments are related to high numbers of e-mails and phone calls to the IT service desk?". To answer this question we formulate utility function $f^{\epsilon}(events(L), events(\Gamma_{L,LPM})) = \sum_{e \in events(\Gamma_{L,LPM})} \pi_{number\_of\_interactions}(e)$. Figure 5 shows the two LPMs with the highest utility that we discovered from the IT service desk event log using this utility function.

The total utility of the top LPM of Figure 5 is 1175, indicating that the behavior of this LPM explains 1175 calls and e-mails to the IT service desk. The LPM shows that in total 1262 calls and e-mails to the IT service desk were associated with incidents with closure code *software application* that were caused by *WBS000073*, 1164 of which were associated with such an incident that fits the LPM. Only one single incident with this closure code and causedBy number out of the total 89 in the log fit the LPM behavior, however, this single incident had high impact as it caused 1164 of the 1262 interactions. Finally, the LPM ends with an incident with closure code *No error - works as designed*. This indicates that this software application contains a feature that is working properly according to the IT service desk ("works as designed"), while bank employees perceive it as a problem, resulting in a lot of traffic to the IT service desk. Because only one single *software application* | *WBS000073* incident fits the behavior of this LPM, one could say that it describes an anomaly rather than a pattern. Note that the discovery of such anomaly-type LPMs is a result of the way we defined our utility function, which allows for highly skewed distribution of utility over events.

The second LPM shows a similar pattern, but it concerns incidents caused by server based software application *WBS000095.* This LPM has a total utility of 504, meaning that it describes in total 504 calls and e-mails to the IT service desk. The model shows that 60 out of 61 *user errors* are eventually followed by an incident with closure code *No error - works as designed.* These 60 incidents together generate 62 interactions with the IT service desk. Note that 399 data incidents related to this software application, causing 428 interactions with the service desk, also resulted in an incident with this closure code.

In the two LPMs we see that two service components (*WBS000073* and *WBS000095*) are the main cause of call and e-mails to the IT service desk. Furthermore, for both service components, the LPMs end in *No error - works as designed* events, which could indicate that such problems could be prevented.

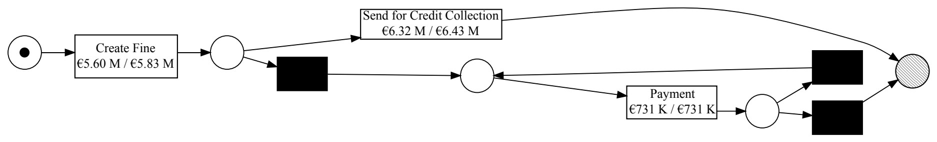
5.2 Traffic Fines

The road traffic fine management event log[3] is an event log where each case refers to a traffic fine. Each case starts with a *create fine* event, which has a property *amount* that specifies the amount of the fine. *Payment* events have a *paymentAmount* property, which indicates how much has been paid. Payment of the total fine amount can be spread out over multiple payments. Some fines are paid directly to the police officer when the fine is given, and some are sent by mail, in which case there is a *send fine* event. *Send fine* events have a property *expense*, which contains an additional administrative cost which adds to the total amount that has to be paid. When a fine is not paid in time, an *add penalty* event occurs which has an *amount* property that updates the fine amount set in the *create fine* event. If a fine is still not paid after the added penalty it is *send for credit collection.* Furthermore, a fine can be appealed at the prefecture and, at a later stage, can be appealed in court. In total the traffic fines event log contains 150,370 traces, 561,470 events, and 11 activities.

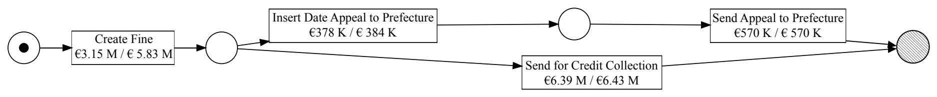
Assume we are a process analyst concerned with the business question: "which process fragments describe fines where the remaining amount to be paid is high?". Figure 6 shows the top three LPMs that we discovered from the traffic fine log using a trace-level utility function that defines utility as the remaining amount that is still to be paid at the time of the event. This utility function has a trace-level scope, as it is calculated from the latest seen *amount* in the case (either from a *create fine* or *add penalty* event) plus the *expense* fee if there is a *send fine* event minus the *paymentAmount* values of all *payment* events seen in the trace so far.

The first LPM in Figure 6 shows that in total €5.83 million has been created in fines, out of which €5.60 million was either send to credit collection or payments have been received for them. The remaining €200 thousand correspond to recent fines that have not yet been paid but are not yet due to be sent for credit collection. The total amount of payments received after a *create fine* event is €731 thousand, which is surprisingly low comparable to the total of €5.83 million. Noteworthy is that fines representing a total value of €6.43 million are

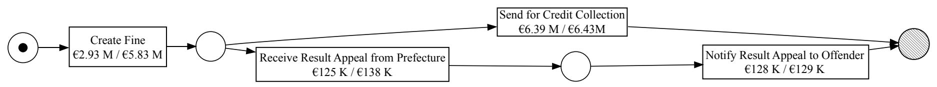
[3] http://dx.doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5



(1) total utility= €12.6 million


Create Fine
€3.15 M / € 5.83 M
Insert Date Appeal to Prefecture
€378 K / € 384 K
Send Appeal to Prefecture
€570 K / € 570 K
Send for Credit Collection
€6.39 M / €6.43 M

(2) total utility= €10.5 million


Create Fine
€2.93 M / €5.83 M
Send for Credit Collection
€6.39 M / €6.43M
Receive Result Appeal from Prefecture
€125 K / €138 K
Notify Result Appeal to Offender
€128 K / €129 K

(3) total utility= €9.6 million

Fig. 6. The top three LPMs in terms of total utility as discovered from the traffic fine log, using *remaining amount to pay* as utility.

send to credit collection, which is even more than the fines representing a value of €5.83 million that were created in the first place. That the total value of fines at credit collection is higher than the total value of fines that were handed out is because of added penalties and, to a lesser degree, added expenses. Finally, fines representing a value of €6.32 million that were sent to credit collection out of the total €6.43 million value of fines that were sent to credit collection fit the control flow pattern of the LPM, i.e., they occur after a *create fine* and are in an XOR-construct with *payment* events. Since we know that all traces start with a *create fine* event, the only possible explanation left is that fines representing a value of €6.32 million that were sent to credit collection have not received a single payment before being send to credit collection, while for the remaining fines representing a value of €110 thousand that were sent to credit collection at least one partial payment of the fine was already received.

The second LPM in Figure 6 shows that fines representing a value of €3.15 out of the total value of fines of €5.83 million was either sent to credit collection or appealed at the prefecture. The appeal procedure starts with an *insert data appeal to prefecture* event which is later followed by a *send appeal to prefecture* event. The LPM shows that for the appealed fines, the total value is €378 thousand at the time of *insert date appeal to the prefecture*, but added penalties for late paying raise the total appealed amount to €570 thousand at the time that the appeal is actually send to prefecture. This means appealed fines on average multiply 1.5x in value during the appeal procedure as a result of penalties being added for the payment term being overdue. Finally, the numbers in the *send for credit collection* transition show that only a very small portion of the total value of fines that were sent to credit collection were followed by an appeal procedure.

The third LPM in Figure 6 shows a pattern similar to the second LPM, with the difference that it now describes two later steps in the appeal to prefecture procedure. At the *receive result appeal from prefecture* step there is a total value of € 138 thousand. This is considerably lower than the fines representing values of € 378 thousand and € 570 thousand respectively that we found for the earlier steps of the appeal procedure. This shows that there are appealed fines representing a value of € 570 thousand - € 138 thousand = € 432 thousand which did not receive the appeal results. These appeals were either withdrawn before the verdicts on these appeals were made, or these appeals are still waiting for a verdict. Furthermore, fines representing a value of € 125 thousand out of the total value of € 138 thousand of fines at *receive result appeal from prefecture* fit the control flow of the pattern, indicating that fines representing a value of €13 thousand which received a result for appeal were either not (yet) notified, or they have been *sent for credit collection* prior to the appeal.

6 Conclusions & Future Work

This paper presents a framework of utility functions and constraints for Local Process Models (LPMs) that allows for combinations of utility functions and constraints on different scopes: on the activity, event, trace, and model level. We formalize utility functions on each of the levels and provide examples of how they can be used. Finally, we show on real-life event logs that the utility functions and constraints can be used to discover insightful LPMs that cannot be obtained using existing support-based LPM discovery.

When a model e.g. allows for one execution of activity a while multiple executions of a are observed in the log, different alignments are possible depending on the choice of a for the synchronous move and the ones for log moves. However, these events do not necessarily have equal utility. Thus, the utility of the LPM depends on the alignment returned by the alignment algorithm. As future work, we plan to make alignments utility-aware, so that the optimal alignment leads to the highest LPM utility.

References

1. van der Aalst, W.M.P.: Process mining: data science in action. Springer-Verlag Berlin Heidelberg (2016)
2. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Conformance checking using cost-based fitness analysis. In: Proceedings of the 15th IEEE Enterprise Distributed Object Computing Conference. pp. 55–64. IEEE (2011)
3. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: International Conference on Business Process Management. pp. 375–383. Springer (2007)
4. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The ProM framework: A new era in process mining tool support. In: International Conference on Application and Theory of Petri Nets. pp. 444–454. Springer Berlin Heidelberg (2005)
5. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In: International Conference on Business Process Management. pp. 328–343. Springer (2007)

6. Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: current status and future directions. Data Mining and Knowledge Discovery 15(1), 55–86 (2007)
7. International Organization for Standardization: ISO/IEC 19505-1:2012 - Information technology - Object Management Group Unified Modeling Language (OMG UML) - Part 1: Infrastructure (2012)
8. Keller, G., Scheer, A.W., Nüttgens, M.: Semantische Prozeßmodellierung auf der Grundlage" Ereignisgesteuerter Prozeßketten". Inst. für Wirtschaftsinformatik (1992)
9. Lan, G.C., Hong, T.P., Tseng, V.S., Wang, S.L.: Applying the maximum utility measure in high utility sequential pattern mining. Expert Systems with Applications 41(11), 5071–5081 (2014)
10. Leemans, M., van der Aalst, W.M.P.: Discovery of frequent episodes in event logs. In: International Symposium on Data-Driven Process Discovery and Analysis. pp. 1–31. Springer (2014)
11. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: International Conference on Business Process Management. pp. 66–78. Springer (2013)
12. Liesaputra, V., Yongchareon, S., Chaisiri, S.: Efficient process model discovery using maximal pattern mining. In: International Conference on Business Process Management. pp. 441–456. Springer (2015)
13. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-guided discovery of declarative process models. In: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining. pp. 192–199. IEEE (2011)
14. Măruşter, L., van Beest, N.R.T.P.: Redesigning business processes: a methodology based on simulation and process mining techniques. Knowledge and Information Systems 21(3), 267 (2009)
15. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
16. Object Management Group: Notation (BPMN) version 2.0. OMG Specification (2011)
17. Pei, J., Han, J., Wang, W.: Constraint-based sequential pattern mining: the pattern-growth methods. Journal of Intelligent Information Systems 28(2), 133–160 (2007)
18. Ramezani, E., Fahland, D., van der Aalst, W.M.P.: Where did I misbehave? diagnostic information in compliance checking. In: International conference on Business Process Management. pp. 262–278. Springer (2012)
19. Shen, Y.D., Zhang, Z., Yang, Q.: Objective-oriented utility-based association mining. In: Proceedings of the IEEE International Conference on Data Mining. pp. 426–433. IEEE (2003)
20. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: International Conference on Extending Database Technology. pp. 1–17. Springer (1996)
21. Tax, N., Sidorova, N., van der Aalst, W.M.P., Haakma, R.: Heuristic approaches for generating local process models through log projections. In: 2016 IEEE Symposium on Computational Intelligence and Data Mining. pp. 1–8. IEEE (2016)
22. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.P.: Mining local process models. Journal of Innovation in Digital Ecosystems 3(2), 183–196 (2016)
23. Yin, J., Zheng, Z., Cao, L.: USpan: an efficient algorithm for mining high utility sequential patterns. In: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 660–668. ACM (2012)
24. Yin, J., Zheng, Z., Cao, L., Song, Y., Wei, W.: Efficiently mining top-k high utility sequential patterns. In: Proceedings of the IEEE 13th International Conference on Data Mining. pp. 1259–1264. IEEE (2013)