Mining Context-Dependent and Interactive Business Process Maps using Execution Patterns

Jiafei Li[1,2], R.P. Jagadeesh Chandra Bose[2,3], and Wil M.P. van der Aalst[2]

[1] College of Computer Science and Technology, Jilin University, China 130012,
jiafei@jlu.edu.cn,l.j.f.Li@tue.nl,
[2] Eindhoven University of Technology, The Netherlands
j.c.b.rantham.prabhakara@tue.nl,w.m.p.v.d.aalst@tue.nl
[3] Philips Healthcare, Veenpluis 5-6, Best, The Netherlands

Abstract. Process mining techniques attempt to extract non-trivial knowledge and interesting insights from event logs. Process models can be seen as the "maps" describing the operational processes of organizations. Unfortunately, traditional process discovery algorithms have problems dealing with less-structured processes. Furthermore, existing discovery algorithms do not consider the analyst's context of analysis. As a result, the current models (i.e., "maps") are difficult to comprehend or even misleading. To address this problem, we propose a two-phase approach based on common execution patterns. First, the user selects relevant and context-dependent patterns. These patterns are used to obtain an event log at a higher abstraction level. Subsequently, the transformed log is used to create a hierarchical process map. The approach has been implemented in the context of ProM. Using a real-life log of a housing agency we demonstrate that we can use this approach to create maps that (i) *depict desired traits*, (ii) *eliminate irrelevant details*, (iii) *reduce complexity*, and (iv) *improve comprehensibility*.

1 Introduction

Process mining aims at extracting process-related information from event logs. Process mining techniques can deliver valuable, factual insights into how processes are being executed in real life. The majority of research in process mining so far has focussed on process discovery (both from a control-flow and organizational perspective). Process models can be seen as the "maps" describing the operational processes of organizations. Unfortunately, *accurate and interactive business process maps* are missing. Either there are no good maps or maps (if available) are static and/or outdated [1].

Process mining techniques can be used to generate process maps [2, 3, 4]. We have applied our process mining tool ProM in more than 100 organizations and our experiences show that processes tend to be less structured than expected. Traditional process discovery algorithms have problems dealing with such unstructured processes and generate spaghetti-like process models that are hard to comprehend. The granularity at which the events are logged is typically different from the desired granularity. Analysts and end users prefer a higher level of abstraction without being confronted with lower level events stored in raw event logs.

Analogous to cartography, process mining techniques should allow for various context-dependent views on the process maps. For example, the perspective of analysis may be different depending on someone's role and expertise e.g., a manager may be interested in a high level view, while a specialist may be interested in a detailed analysis of some process fragment. Process discovery techniques should facilitate the extraction of process maps eliciting the respective desired traits and hiding the irrelevant ones for various users. Furthermore, these techniques should uncover comprehensible models by providing a hierarchical view with a facility to seamlessly zoom in or zoom out the process maps. There is an imperative need for *techniques that automatically generate understandable and context-dependent business process maps* [1].

In this paper, we propose a *two-phase approach to mine interactive and context-dependent business process maps based on common execution patterns*. The *first phase* comprises the pre-processing of a log with desired traits and at a desired level of granularity. This paper will show one means to realize this by uncovering common execution patterns in the log, selecting context-dependent patterns, and defining abstractions over these patterns. Pattern selection and the mapping with abstractions can be interactively performed by the user. Event logs are then pre-processed (transformed) with these abstractions. In the *second phase*, the transformed log is used for process discovery. Any discovery algorithm with an ability to zoom-in/out the sub-processes defined by the abstractions can be used. This paper presents an adapted version of the Fuzzy Miner [3] and shows that it can provide such hierarchical view of process maps. The two-phase approach presented in this paper has been implemented in ProM 6.0[1]. Figure 1 highlights the difference between the traditional approach to do process discovery and the two-phase approach. Note that the process model (map) mined using the two-phase approach is simpler and that this approach enables the abstraction of activities based on functionality and provides a seamless zooming into the sub-processes captured in the abstractions.

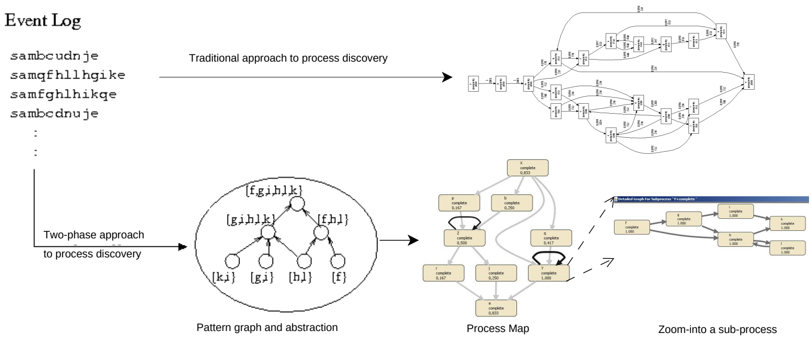


Fig. 1: Traditional approach vs. two-phase approach

[1] ProM 6.0 is not officially released yet, but nightly builds, including the reported functionality are available from `www.processmining.org`

The remainder of the paper is organized as follows. Our two-phase approach to mining process maps is introduced in Section 2. Section 3 presentes pattern definitions and pattern metrics while Section 4 proposes one approach to form abstractions based on patterns. In Section 5, we detail our two-step approach and describe an adaptation of Fuzzy Miner to discover process maps. Section 6 presents a case study of a real-life log from a rental agency. Related work is discussed in Section 7. Section 8 concludes the paper.

2 Two-phase Approach to Mine Process Maps

We use the following notations in this paper. Let Σ denote the set of activities. $|\Sigma|$ is the number of activities. Σ^+ is the set of all non-empty finite sequences of activities from Σ. We denote traces by bold face lower case letters $\mathbf{t}_1$, $\mathbf{t}_2$ etc. A trace $\mathbf{t}$ is an element of Σ^+. $\mathbf{t}$(i) denotes the i^{th} activity in the trace. For $i < j$, $\mathbf{t}(i,j)$ denotes the subsequence from the i^{th} position to the j^{th} position in the trace $\mathbf{t}$. An event log $\mathcal{L}$ corresponds to a bag (i.e., a multiset) of traces.

Phase-1: Preprocessing Log In this phase, the log is simplified based on the desired traits of the context of analysis. A mapping $\mathcal{M} \subseteq 2^{\Sigma} \times \mathcal{A}$ is defined between the *original alphabet* of the event log Σ, and an *abstract alphabet* $\mathcal{A}$. An example mapping is $\mathcal{M} = \{(\{\mathtt{a},\mathtt{b}\},\mathtt{x}),(\{\mathtt{b},\mathtt{c},\mathtt{d}\},\mathtt{y}),(\{\mathtt{e}\},\mathtt{z}),(\{\mathtt{d}\},\mathtt{z})\}$. This mapping is analogous to the grouping and tagging of streets as a town/city in cartography and to the selection of a desired perspective of viewing maps (restaurant maps vs. fuel station maps). The analyst can define this mapping based on domain knowledge or can be assisted by uncovering common execution patterns and relationships between them in the log. These common execution patterns typically capture a sub-process/functionality. Analysts would like to capture such subprocess behavior in its totality as an *abstract activity* in a mined process model with a facility to zoom in/out the subprocess if needed. The mapping is defined over the sets of activities manifested as patterns. We present techniques that assist in automatically uncovering such patterns and relationships between activities in Section 3.

$\mathcal{D} = \bigcup_{(A,\mathtt{a})\,\in\mathcal{M}} A$ denotes the set of activities in Σ for which a mapping is defined. The original event log $\mathcal{L}$, is transformed into an *abstract log* $\mathcal{L}'$. Each trace $\mathbf{t} \in \mathcal{L}$ is transformed into a corresponding trace $\mathbf{t}' \in \mathcal{L}'$. In each trace $\mathbf{t}$, the manifestation of each pattern captured by $(A,\mathtt{a}) \in \mathcal{M}$ is replaced with its *abstract activity*, $\mathtt{a}$, in the transformed trace. The activities in $\Sigma \setminus \mathcal{D}$ being not involved in the definition of mapping indicate activities that are insignificant from the context of analysis and are filtered from $\mathbf{t}$ during this transformation. In Section 5, we describe the transformation of log in detail.

Phase-2: Mining Maps The second phase is to mine a process model on the abstract log. The mapping defined in Phase-1 induces a hierarchy over the abstract activities. Upon zooming into an abstract activity, a process model depicting the subprocess captured by this abstract activity is shown. The patterns replaced by the abstract activity are used to create this sub-process model. We adapted Fuzzy Miner for this phase and the details are presented in Section 5. Note that this is a generic approach that can be iterated over any number of times with the event log for iteration $i+1$ being the output event log of iteration i.

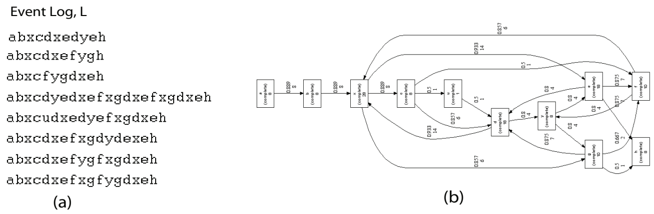


Fig. 2: (a) An example log (b) process models discovered using Heuristic miner

We use a running example log depicted in Figure 2(a) to illustrate the approach. This log contains 8 process instances with 11 event classes. Figure 2(b) depicts the model mined using traditional process discovery techniques. It is imperative to find that both these models are not easy to understand. In the following sections, we will present our two-phase approach in more detail using this example.

3 Pattern Definitions and Pattern Metrics

In this section, we adapt the pattern definitions proposed in [5] and focus on defining metrics over these patterns. We consider only the *maximal repeat* patterns for the discussion in this paper. However, other patterns such as *tandem arrays* capturing the manifestation of loop constructs proposed in [5] can also be used. These patterns are later used to define the mapping $\mathcal{M}$ between activities and abstractions.

3.1 Pattern definitions

Definition 1 (Maximal Repeat). *A maximal pair in a sequence, $\boldsymbol{s}$, is a pair of identical sub-sequences α and β such that the symbol to the immediate left/right of α is different from the symbol to the immediate left/right of β. In other words, extending α and β on either side would destroy the equality of the two strings. A maximal pair is denoted by the triple (i, j, α) where i and j correspond to the starting positions of α and β in $\boldsymbol{s}$ with $i \neq j$. A maximal repeat in a sequence, $\boldsymbol{s}$, is defined as a subsequence α that occurs in a maximal pair in $\boldsymbol{s}$.*

Maximal repeats capture execution patterns (sequence of activities) common within a trace and/or across a set of traces in an event log. Such patterns might be evidence of common functionality (often abstracted as a sub-process). In order to find these commonalities across multiple traces in the entire event log, we first construct a single sequence, say, $\mathbf{s}$, which is obtained by the concatenation of traces in the event log with a distinct delimiter between the traces. Maximal repeats are then discovered over this concatenated sequence $\mathbf{s}$. Maximal repeats can be efficiently discovered in linear time using suffix trees for strings [6]. Let $\mathcal{P}_{\mathcal{L}}$ be the set of all patterns in log $\mathcal{L}$. In this paper, $\mathcal{P}_{\mathcal{L}}$ includes all maximal repeats and $\{\mathtt{a} \mid \mathtt{a} \in \Sigma\}$. A *base pattern* is a pattern that does not contain any other pattern within it. The pattern `abxc` is a base pattern while `abxcdxe` is not because the latter pattern contains the pattern `x` within it. Let $\mathcal{P}^b_{\mathcal{L}}$ be the set of all base patterns in $\mathcal{P}_{\mathcal{L}}$.

Consider the trace $\mathbf{t}_6$ = `abxcdxefxgdyedxeh` in the log of Figure 2(a). The maximal pairs in $\mathbf{t}_6$ are $(3, 6, \mathtt{x})$, $(5, 11, \mathtt{d})$, $(7, 13, \mathtt{e})$ and $(5, 14, \mathtt{dxe})$. There are a total of 39 maximal repeats in the example log (e.g., `abxc`, `abxcd`, `abxcdxe`, `dxe`, `fxg`, `dye`, `fyg`, `dyedxe`, `dxedye`, `h`). $\mathcal{P}^b_{\mathcal{L}}$ = {`a`, `b`, `c`, `u`, `x`, `d`, `g`, `f`, `h`, `e`, `y`, `gf`, `gd`, `ef`, `ed`, `eh`, `fxg`, `dye`, `fyg`, `dxe`, `fxgd`, `abxc`, `gdxe`, `dxeh`, `dxef`, `efxg`, `efxgd`, `abxcd`, `gdxeh`,`fygdxeh`}.

Definition 2 (Pattern Alphabet). *The pattern alphabet $\Gamma(p)$, of a pattern, $p \in \mathcal{P}_{\mathcal{L}}$, is the set of activities that appear in p.*

Definition 3 (Equivalence Class of Pattern Alphabet). *The equivalence class of a pattern alphabet PA, is defined as $[PA] = \{p \mid p$ is a pattern and $\Gamma(p) = PA\}$*

For example, for the patterns `fxg`, `dyedxe` and `dxedye`, the pattern alphabets correspond to {`f`,`x`,`g`}, {`d`,`x`,`y`,`e`}, and {`d`,`x`,`y`,`e`} respectively. The equivalence class of the pattern alphabet {`d`,`x`,`y`,`e`} is {`dyedxe`,`dxedye`}. Equivalence classes of pattern alphabets capture variations of patterns e.g., due to parallelism.

3.2 Pattern Metrics

Pattern metrics such as the frequency of occurrence, significance etc. need to be estimated. A careful consideration needs to be done when estimating the frequency of a pattern. It is due to the fact that certain regions in a trace can contribute to more than one pattern (in the case of overlapping patterns) and might result in misleading frequency counts. For example, consider the trace $\mathbf{t}$ = `abxcdxedfxgdxeh` and the pattern alphabet equivalence classes [{`a`,`b`,`x`,`c`}] = {`abxc`}, [{`a`,`b`,`x`,`c`,`d`}] = {`abxcd`}, and [{`d`,`x`,`e`}] = {`dxe`,`dxed`}. Now, *what should be the pattern (alphabet) counts?*. If we consider each of the patterns separately, the pattern, frequency-count pairs are $(\mathtt{abxc}, 1), (\mathtt{abxcd}, 1), (\mathtt{dxe}, 2)$ and $(\mathtt{dxed}, 1)$. If we define the pattern alphabet count to be the sum of counts of the patterns captured in its equivalence class, then the pattern alphabet, frequency-count pairs are $(\{\mathtt{a}, \mathtt{b}, \mathtt{x}, \mathtt{c}\}, 1)$, $(\{\mathtt{a}, \mathtt{b}, \mathtt{x}, \mathtt{c}, \mathtt{d}\}, 1)$, and $(\{\mathtt{d}, \mathtt{x}, \mathtt{e}\}, 3)$. It is imperative to see that certain regions in the trace are contributing to more than one pattern (alphabet). The activities in the subsequence $\mathbf{t}(5,7)$ contributed to two patterns viz., `dxe` and `dxed`. Similarly, the activities in the subsequence $\mathbf{t}(1,4)$ contributed to two patterns viz., `abxc` and `abxcd`.

We identify three distinct methods of dealing with overlaps and counting pattern occurrences. The above method of computing pattern (alphabet) counts is referred to as *Overlapping Alphabet Count (OAC)*. The significance computed using overlapping alphabet counts may be misleading. A more accurate method of computing pattern frequencies is to consider non-overlapping pattern counts. We distinguish two variations here: (i) considering non-overlap counts for each alphabet separately (local) and (ii) considering non-overlap counts across all alphabets (global) in the event log. These two metrics are referred to as *Non-Overlapping Alphabet Count (NOAC)* and *Non-Overlapping Global Alphabet Count (NOGAC)* respectively. Conflicts arise when more than one pattern can potentially contribute to the count at a region in a trace. One can assign preference to say shorter (longer) patterns to resolve such conflicts. The *NOAC* (with preference to shorter patterns) for the above example is $(\{\mathtt{a}, \mathtt{b}, \mathtt{x}, \mathtt{c}\}, 1)$, $(\{\mathtt{a}, \mathtt{b}, \mathtt{x}, \mathtt{c}, \mathtt{d}\}, 1)$, and $(\{\mathtt{d}, \mathtt{x}, \mathtt{e}\}, 2)$. Note that the conflict at position 5 in $\mathbf{t}$ for pattern alphabet $\{\mathtt{d},\mathtt{x},\mathtt{e}\}$ is resolved in favor of the pattern `dxe` thereby making $\mathbf{t}(5,7)$ contribute to only one pattern. The *NOGAC* across all alphabets (with preference to longer patterns) is $(\{\mathtt{a}, \mathtt{b}, \mathtt{x}, \mathtt{c}\}, 0)$, $(\{\mathtt{a}, \mathtt{b}, \mathtt{x}, \mathtt{c}, \mathtt{d}\}, 1)$, and $(\{\mathtt{d}, \mathtt{x}, \mathtt{e}\}, 1)$. A position/subsequence in a trace can contribute to more than one pattern alphabet when considering *NOAC* for each alphabet separately (e.g., index 1 in $\mathbf{t}$) while in *NOGAC*, a position contributes to at most one pattern alphabet.

In order to assess the significance of a pattern alphabet PA, we define a metric *Conservedness* $(CON_{PA}) = \frac{NOAC}{\mu} * (1 - \frac{\sigma}{\mu}) * 100\%$ where μ and σ are the mean and standard deviation of the frequencies of activities in PA. *Conservedness* measures the degree to which the individual activities involved in the pattern alphabet manifest as the patterns defined by the alphabet. For example, consider the non-overlap alphabet count of three pattern alphabets $(\{\mathtt{d}, \mathtt{x}, \mathtt{e}\}, 100)$, $(\{\mathtt{d}, \mathtt{x}, \mathtt{e}, \mathtt{f}\}, 60)$, and $(\{\mathtt{d}, \mathtt{x}, \mathtt{e}, \mathtt{h}\}, 40)$. Let the frequency of activities be $(\mathtt{d}, 100)$, $(\mathtt{x}, 100)$, $(\mathtt{e}, 100)$, $(\mathtt{f}, 60)$, and $(\mathtt{h}, 40)$. Conservedness value of the pattern alphabets $\{\mathtt{d},\mathtt{x},\mathtt{e}\}$, $\{\mathtt{d},\mathtt{x},\mathtt{e},\mathtt{f}\}$, and $\{\mathtt{d},\mathtt{x},\mathtt{e},\mathtt{h}\}$ is 100%, 51% and 30% respectively. The formal definitions of the above pattern metrics are presented in [7].

4 Abstractions based on Patterns

4.1 Pattern Graph

Relationships exist between patterns (alphabets). For example, consider the patterns `dxefxg`, `dxe`, and `fxg`. It could be the case that `dxe` and `fxg` are subfunctionalities used also in a larger context `dxefxg`. One can try to define a partial order capturing the relationships on the pattern alphabets. For example, subsumption can be used as the cover relation. A pattern alphabet PA_i is defined to cover another pattern alphabet PA_j if $PA_j \subset PA_i$ and there is no PA_k such that $PA_j \subset PA_k \subset PA_i$. A *pattern graph* $G = (V, E)$, is a Hasse diagram defined over the partial order on the pattern alphabets, where $V = \{PA_1, PA_2, \ldots, PA_n\}$ represents the set of pattern alphabets and E denotes the set of edges (PA_i, PA_j) defined by the cover relation. One can choose either $\mathcal{P}_\mathcal{L}$ or $\mathcal{P}^b_\mathcal{L}$ to define V. Figure 3(a) depicts a pattern graph on some of the pattern alphabets identified for the example log. We considered pattern alphabets defined by $\mathcal{P}^b_\mathcal{L}$ with a conservedness value above 17% to generate this graph.

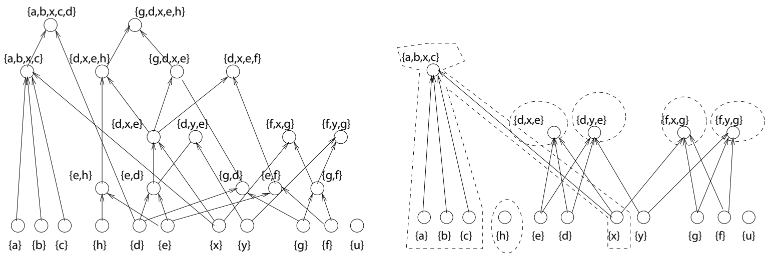
4.2 Pattern Selection

Nodes in a pattern graph form the basis for abstraction. An analyst can select the pattern nodes based on domain knowledge or by using the pattern metrics defined in Section 3. We provide two types of node selection modes for abstraction.

Single Node Mode: All manifestations of patterns under the equivalence class of this node's pattern alphabet are represented by the same abstract activity in the transformed log.

Sub-graph Mode: All manifestations of patterns under the equivalence classes of the pattern alphabets defined by the *induced subgraph* at the selected node are substituted by the abstract activity of the selected node during transformation.

It could be the case that a pattern graph contains a large number of nodes. We recommend to first filter the nodes in the pattern graph before considering them for abstractions. All the metrics defined in Section 3.2 can be used to prune the graph. For example, consider the pattern alphabets {a,b,x,c} and {a,b,x,c,d} in Figure 3(a). The *NOGAC* of {a,b,x,c,d} with preference to shorter patterns (ignoring individual activity patterns) is zero. Similarly, the *NOGAC* of {d,x,e, f}, {d,x,e,h}, {g,d,x,e,h}, {g,d,x,e}, {e,h}, {e,d}, {e,f}, {g,d} and {g,f} are all zero. This indicates that manifestations of all patterns under the equivalence class of these pattern alphabets in the log are overlapping with some other pattern. For example, the equivalence class of the pattern alphabet {e,d} is {ed}. There are two manifestations of the pattern `ed` in $\mathcal{L}$ (in traces `abxcdxedyeh` and `abxcudxedyefxgdxeh`). However, both of these manifestations overlap with `dxe` and `dye` in the example log; thus making the *NOGAC* of {e,d} as 0.



(a) (b)

Fig. 3: (a) Pattern graph (b) pattern graph with abstractions for the example log

We recommend to consider nodes capturing longer patterns with a high conservedness value and significant *NOAC* and *NOGAC* to be used under sub-graph mode for abstractions. However, for two pattern alphabet nodes PA_i and PA_j such that $(PA_i, PA_j) \in E$ (i.e., $PA_i \subset PA_j$), if $CON_{PA_i} > CON_{PA_j}$ then, we recommend to consider PA_i under sub-graph mode instead of PA_j though PA_i captures shorter patterns. For example, consider the pattern alphabets {d,x,e}, {d,x,e,f}, {d,x,e,h} and {g,d,x,e}. The conservedness value for these alphabets are $52\%, 22\%, 21\%, 22\%$ respectively. It could be seen that the pattern `dxe` (defined by the alphabet {d,x,e}) occurs in different contexts in $\mathcal{L}$. The different contexts are captured by the other three alphabets and are reflected with the relatively low conservedness values for these three alphabets. We recommend to consider {d,x,e} as a node for abstraction instead of the other three. Coincidentally in this example the *NOGAC* (with preference to shorter patterns) for the three larger alphabets is also zero.

If nodes in the sub-graph of a pattern node PA_i are covered by one or more nodes PA_j that are not in the sub-graph of PA_i then we recommend to consider PA_i under single-node mode for abstraction (assuming PA_i is selected). For example, the node {d,x,e} is recommended to be considered under single-node mode because the nodes {d} and {e} in the sub-graph of {d,x,e} is also covered by another node {d,y,e}. Note that these are just recommendations and an analyst can make exceptions if it makes sense according to the context of analysis. Using these guidelines we use the abstractions as defined in Figure 3(b). Here {a,b,x,c} is used in the sub-graph mode while {d,x,e}, {d,y,e}, {f,x,g}, {f,y,g} and {h} are chosen under single-node mode. Certain nodes not pertaining to the context of analysis can also be filtered out (e.g., {u}). Let us define the mapping $\mathcal{M}$ as {({a,b,x,c},A1), ({a},A1), ({b},A1), ({c},A1), ({x},A1), ({d,x,e},A2), ({f,x,g},A3), ({d,y,e},A4), ({f,y,g},A5), ({h},A6)} on the abstractions chosen for the example log.

5 Process Discovery based on Patterns

5.1 Transformation of Log

Algorithm 1 presents the details of transforming the log based on the patterns. The *basic idea* is to first replace the continuous and intermittent manifestation of each pattern alphabet chosen for abstraction with its abstract activity and make the corresponding low level manifestations part of the sub-log corresponding to the abstract activity. The sub-log of an abstract activity can be used to zoom in the detailed behavior. The intermittent manifestation here refers to the situation where the execution of the subsequence corresponding to a pattern is interrupted by other activities. For instance, let dye be a pattern, the manifestation of dye in the trace abxcdxefxgdydexeh is intermittent because dye is interrupted by d.

Steps 9-12 in the algorithm deal with the intermittent manifestation of a pattern and substitutes it with the abstract activity. Algorithm 1 will transform the trace abxcdxefxgdydexeh in our example log to A1A2A3A4A2A6. In this way, one can cope with situations where a common functionality is interrupted by other activities in concurrency.

5.2 Adapting Fuzzy Miner to Discover Maps

ProM's *Fuzzy Miner* [3] is inspired by cartography to provide business process maps. However, the existing miner has some limitations. It (i) cannot customize maps from a defined context (city maps vs. highway maps) (ii) introduces the risk of aggregating unrelated activities together in a cluster (a street in Eindhoven is clustered along with streets in Amsterdam) and (iii) provides two level hierarchy instead of a multi-level hierarchical view of the process map.

Let $\mathcal{M}$ be the mapping chosen by the user. $\mathcal{A} = \cup_{(PA,a)\in\mathcal{M}}\{a\}$ defines the set of defined abstractions. $\mathcal{SP} = \cup_{(PA,a)\in\mathcal{M}}[PA]$ denotes the set of all patterns for which abstractions are defined. Let $f : \mathcal{SP} \rightarrow \mathcal{A}$ be the function defining the abstraction for each pattern. Let $l : \mathcal{A} \rightarrow \mathcal{SL}$ be the function defining the sub-log for each abstraction. Let $\mathcal{L}'$ be the transformed log of $\mathcal{L}$. Initialize $\mathcal{L}' = \{\}$ and $l(a) = \{\}$ for all $a \in \mathcal{A}$
for all t $\in \mathcal{L}$ do
  Let t$'$ be an empty trace. Set $j = 1$.
  while $j \leq |\mathtt{t}|$ do
    Let LD_s be the list of patterns in $\mathcal{SP}$ starting with t(j) ordered in descending order of their length
    for every pattern $\alpha \in LD_s$ do
      if there exists a continuous manifestation of a pattern α at index j in t then
        $l(f(\alpha)) = l(f(\alpha)) \uplus \{\mathtt{t}(j, j + |\alpha|)\}$;Append $f(\alpha)$ to t$'$;Set $j = j + |\alpha| - 1$;exit for
      else if there exists an intermittent manifestation of α at index j in t then
        Re-adjust the intermittent manifestation in t.
        $l(f(\alpha)) = l(f(\alpha)) \uplus \{\alpha\}$; Append $f(\alpha)$ to t'; Set $j = j + |\alpha| - 1$; exit for
      end if
    end for
    Set $j = j + 1$
  end while
  $\mathcal{L}' = \mathcal{L}' \uplus \{\mathtt{t}'\}$
end for

Algorithm 1: Single-phase pattern-based log transformation

We adapted Fuzzy Miner to support the discovery of process maps. The pattern selection techniques presented in Section 4 facilitate customization from an user's context and getting meaningful abstract activities. By using the sub-log of each abstract activity, we implemented the functionality of zooming in/out the abstract activity and showing the detailed sub-process captured by it. Furthermore, by combining with the existing functions in the Fuzzy Miner of zooming in/out the cluster nodes, a three-level view of the process map is provided.

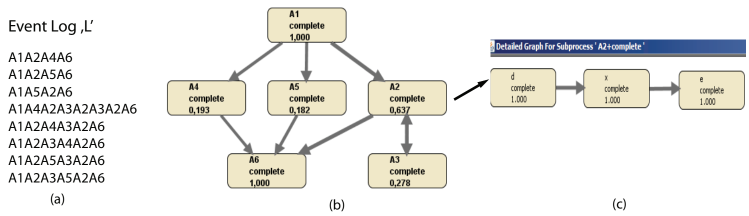


Fig. 4: (a) Transformed Log (b) process map mined from the transformed log using adapted Fuzzy miner (c) sub-process maps when zooming in on A2 and A3

To illustrate this two-phase approach, let us transform the log as described in Algorithm 1 using the mapping $\mathcal{M}$. The transformed log is shown in Figure 4(a). Figure 4(b) depicts the process map mined by the adapted Fuzzy miner, while Figure 4(c) shows the sub-process maps when zooming in the abstract activities A2 and A3. It is evident from Figure 2 and Figure 4 that our two-phase approach helps in presenting more accurate and more readable models.

6 Case Study and Discussion

We applied the techniques proposed in this paper on a real life log of a rental agency where the cases corresponded to cancellation of a current rental agreement and subsequent registration of a new rental agreement. This log was provided by a large Dutch agency that rents houses and apartments and contains 210 cases, 6100 events and 74 event classes. Figure 5(a) depicts the process model mined using heuristic miner. This process model included two types of cancelation as highlighted by two rectangles in Figure 5(a). The unselected region corresponds to common functionality used by both of them. The resulting model is difficult to comprehend.

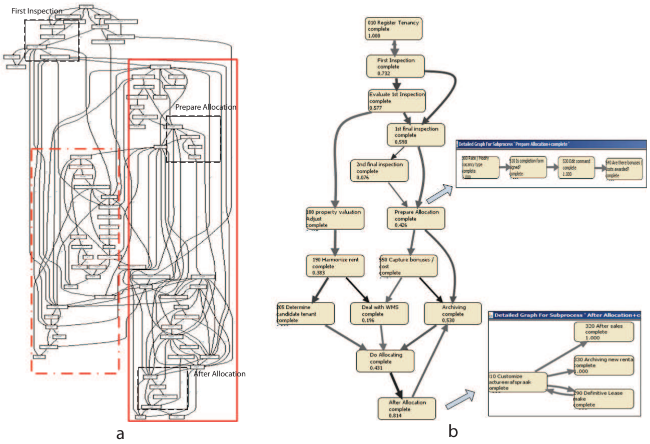


Fig. 5: (a) Heuristic net mined on the whole log (b) process map and zoomed-in sub-processes mined from transformed log based on interactive and context-dependent abstractions

Table 1: Three of the pattern alphabets chosen for abstraction

No.	Abstraction Name	Pattern Alphabet	$NOAC$	CON(%)
1	First Inspection	[050 Plans appointment 1st Inspection,060 Edit confirmation letter / Tenancy form, 070 Is 1st inspection performed?,100 Ready report 1st Insp. / Make-Calculation For]	80	66
2	Prepare Allocation	[500 Rate / Modify vacancy type,540 Are there bonuses / costs awarded?, 510 Is completion form signed?530 Edit command]	145	93
3	1st Final Inspection	[120 Plans final inspections,400 Is final inspection performed?, 440 Are there new or repaired defects?]	53	61

In this study, we assume that the analyst wants to focus on the type of cancellation process defined by the solid rectangle and we identify the patterns from the analyst's point of view. The primary steps involve the registration of a request, multiple inspections of the rented house, determining (future) tenants, (re-)allocation and archiving of the case. We first identified the common execution patterns in this log and chose 17 abstract activities (some involve pattern alphabets and some involve individual activities) concerned with the above primary steps. We used these seventeen abstractions to do the first phase of log transformation. Then, to make the result process map more comprehensible, we performed a second iteration of pattern identification. This visualized in a process map consisting of 14 abstract activities as shown in Figure 5(b). Three of these activities are given in Table 1 which shows the three pattern alphabets used in defining abstractions. Pattern alphabets capturing a functionality from a domain point of view are chosen as candidate nodes (under sub-graph mode) for abstractions. A meaningful name is defined for every candidate abstraction. Those pattern alphabets with a significant *NOAC* as well as a high *CON* value have priority to be selected for abstractions as can be seen in Table 1. Figure 5(b) also presents the sub-process when zooming in the abstract activities of `Prepare Allocation` and `After Allocation`. Each sub-process subsumes the manifestation of patterns captured in the sub-log defined by the abstraction.

Comparing with the cancellation process mainly defined by the solid rectangle in Figure 5(a), it is apparent that the process map discovered by our two-step approach is more comprehensible and captures the main steps of this specific type of rental cancellation process. This resulting process map not only facilitates the analyst to get an overview of the whole process, but also makes it easy to seamlessly zoom-in each abstract activity to observe the detailed subprocess. This shows that using our two-step approach indeed leads to better understandable process maps without sacrificing precision.

7 Related Work

Several approaches based on trace clustering [8, 9, 10] have been proposed in literature. Trace clustering enables the partitioning of the event log based on coherency of cases. Process models mined from each of the resulting clusters are expected to be simpler than that of the one mined from the entire event log. Greco *et al.* [10] augmented trace clustering with an approach to mine hierarchies of process models that collectively represent the process at different levels of granularity and abstraction. This approach tries to analyze the mined process models (post-processing) for identifying activities that can be abstracted. However, for large complex logs, the mined process models (even after clustering) can be quite spaghetti-like. In contrast, the approach proposed in this paper analyzes the raw traces and defines abstraction (pre-processing) and has the ability to zoom-in hierarchically into the abstract entities. Furthermore, the user has flexibility and control when selecting the abstractions/activities of interest based on his/her context of analysis.

Taking cartography as a metaphor, Günther and Aalst [3] have proposed the fuzzy mining approach to implement process simplification. Less significant activities/edges are either removed or clustered together in the model. However, this approach poses a danger of clustering activities/edges having no domain significance. Polyvyanyy *et al.* [11] have proposed a slider approach for enabling flexible control over various process model abstraction criteria. Approaches such as [11, 3] look at abstraction from the point of retaining highly significant information and discarding less significant ones in the process model where the notion of significance is defined over the (relative-)frequency of occurrence of an entity and not based on the context. In contrast, the approach proposed in this paper looks at abstraction from a functionality/subprocess point of view which performs filtering of activities based on the context of analysis. Our approach can be used as a preprocessing step for the logs and can be seamlessly integrated with other approaches for abstraction [10, 3] as well as with classical approaches for process discovery such as the heuristic approach in [4].

8 Conclusions and Future Work

This paper presented a two-phase approach to mining business process maps that comprises the pre-processing of a log based on desired traits and at a desired level of granularity as a first step and discovering the maps with seamless zoom-in facility as the second step. We discussed one means of realizing this two-phase approach by exploiting the common execution patterns in the event log. Metrics assessing the significance of these patterns and ways of selecting these patterns for abstractions were presented. Our initial results on a few real-life logs show encouraging results. Concurrency in process models adds complexity to the discovery of patterns. As future work, we focus on more real-life applications and improving the robustness of the approach in the context of concurrency.

Acknowledgments This work is supported in part by the NNSF (No.60873149) and the 863 project (No.2006AA10Z245) of China, and in part by EIT, NWO-EW, and STW. R.P.J.C. Bose and W.M.P. van der Aalst are grateful to Philips Healthcare for funding the research in Process Mining.

References

1. van der Aalst, W.M.P.: Challenges in Business Process Mining. Applied Stochastic Models in Business and Industry (to appear)
2. van der Aalst, W.M.P.: Using process mining to generate accurate and interactive business process maps. In: BIS (Workshops). Volume 37 of LNBIP. (2009) 1–14
3. Günther, C.W., van der Aalst, W.M.P.: Fuzzy Mining - Adaptive Process Simplification Based on Multi-perspective Metrics. In: Business Process Management (BPM). Volume 4714 of LNCS. (2007) 328–343
4. Weijters, A., van der Aalst, W.M.P.: Rediscovering workflow models from event-based data using little thumb. Integrated Computer-Aided Engineering **10**(2) (2003) 151–162
5. Bose, R.P.J.C., van der Aalst, W.M.P.: Abstractions in Process Mining: A Taxonomy of Patterns. In: Business Process Management (BPM). Volume 5701 of LNCS. (2009) 159–175
6. Gusfield, D.: Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press (1997)
7. Li, J., Bose, R.J.C., van der Aalst, W.M.: Mining Context-Dependent and Interactive Business Process Maps using Execution Patterns. Technical report, University of Technology, Eindhoven (2010) `http://www.win.tue.nl/~jcbose/MiningBusinessProcessMaps.pdf`.
8. Bose, R.P.J.C., van der Aalst, W.M.P.: Context Aware Trace Clustering: Towards Improving Process Mining Results. In: Proceedings of the SIAM International Conference on Data Mining (SDM). (2009) 401–412
9. Bose, R.P.J.C., van der Aalst, W.M.P.: Trace Clustering Based on Conserved Patterns: Towards Achieving Better Process Models. In: Business Process Management Workshops. Volume 43 of LNBIP. (2009) 170–181
10. Greco, G., Guzzo, A., Pontieri, L.: Mining Taxonomies of Process Models. Data Knowl. Eng. **67**(1) (2008) 74–102
11. Polyvyanyy, A., Smirnov, S., Weske, M.: Process Model Abstraction: A Slider Approach. In: Enterprise Distributed Object Computing. (2008) 325–331