Mining Modular Structure of Processes using Process Line Diagrams

Karnika Shivhare¹, Rushikesh K. Joshi¹

¹Indian Institute of Technology Bombay, Mumbai, India 400076

Abstract
Process mining algorithms are unable to retain the modularity details of the processes, which are inherent as part of implementations of process, but get missed out during mining. We propose to mine processes in form of Process Line Diagrams (PLDs) in order to retain modularity as part of process mining itself.

Keywords
Business Processes, Process Line Diagrams (PLD), Process Mining, Petri Nets, Modular Process Structures, PLD Mining

1. Introduction
Process Line Diagram (PLD) [1] is a visual modeling approach that bridges gap between high-level visual modeling languages such as BPMN [2], which are non-compact [3] [4], complex [5] [6], notational heavy [4], redundant [5] [6] and ambiguous [7] [8] [9], and low-level mathematical formalisms such as Petri Nets (PNs) [10] that lack communications and interaction capabilities required for processes [1]. It provides capabilities to capture modularity details of processes.

Modularity of processes refers to inherent structuring of processes into logical modules or components. Existing process mining algorithms focus on extracting the executional structures of processes from their trace logs. These algorithms utilize different modeling approaches for representation of extracted processes. For example, BPMN miners [11], [12] mine BPMN [2], [13], Alpha miner and family [14] [15] [16] [17], and Inductive Miners and variations [18] [19] mine PNs [10], Heuristics miner [20] mines Heuristics Nets [20], CPN Miner [21] mines Colored Petri Nets [22], L’ Process Miner [23] mines processes as finite automata, and DFG Miners [24] [25] mine Directly Follows Graphs. However, these modeling algorithms do not extract modularity of processes. Interdependencies, interactions, and boundaries between process modules get missed during the mining process, which may result in incomplete or inaccurate process models in this direction. This paper addresses this critical gap by proposing to preserve process modularity during mining itself.

Our idea centers on the extraction of modular structures from process trace sets, aiming to retain implementation-friendly modularity into the mining process. We utilize Process Line Diagrams (PLDs) as a means of capturing and visualizing the modular structure of processes extracted from trace logs.

2. An Exemplar Illustration
We consider an exemplar Petri Net (PN) shown in Figure 2 to demonstrate step-wise construction of corresponding PLD. Considering the trace set T = {a.c.t.x.z, a.c.t.y.z, a.c.t.z.x, a.c.t.z.y, b.c.t.x.z, b.c.t.y.z, b.c.t.z.x, b.c.t.z.y, c.a.t.x.z, c.a.t.z.x, c.a.t.z.y, c.b.t.x.z, c.b.t.y.z, c.b.t.z.x, c.b.t.z.y}, we illustrate mining of process using Figure 1, to obtain PLD shown in Figure 3. Process mining algorithms, such as alpha mining [14] are based on footprint matrices and several types of relations [26]. Our approach generates XOR split, XOR merge, AND fork, and AND join relationships as demonstrated respectively using (1) through (4) for our exemplar Petri Net of Figure 2. The approach differs from the other process mining algorithms in extraction and construction, because it directly extracts Process Line Diagrams for processes, rather than building PNs.

\[
xor\_split: \{t^* = \{x^*, y^*\}, 'start' = \{'a', 'b'\}\} \quad (1)
\]
\[
xor\_merge: \{t^* = \{'a', 'b'\}\} \quad (2)
\]
\[
and\_fork: \{t^* = \{'z'\}, \{'x', 'y'\}, 'start' = \{'c'\}, \{'a', 'b'\}\} \quad (3)
\]
\[
and\_join: \{t^* = \{'c'\}, \{'a', 'b'\}\} \quad (4)
\]

Absence of a transition in records of any of the relationships ((1) through (4)) can be assumed as an empty set for that transition. It represents the state of transition being not involved in that relationship. For example, there are no XOR splits from transitions a, b, c, x, y and z in this example, and thus they do not hold their records in xor_split shown in (1).

Construction of a process line diagram for the process begins with construction of a role from unnamed marking point, start (hidden). Presence of and_fork(start) in (3), constructs multicast event as shown in Figure 1(a), subsequently followed by two corresponding event catches into two new roles as shown in Figure 1(b), because and_fork(start) forks transition start into [c] and [a, b]. We name the multicast event and its corresponding event catches as a combination of two forks i.e. here abc. One of these event catches, representing the forking from start into [c], is immediately followed by construction of transition c due to absence of XOR merge and AND join at transition c. It is shown in Figure 1(c). Transition c is involved in and_join(t) = \{'c', \{'a, b'\}\} representing and_join at transition t. Ergo, a throw event t is constructed next to transition c as shown in Figure 1(d). An event catch t to correspond to this event throw t is constructed in a new role as shown in Figure 1(e), because merge(s)/join(s) exist at transition t and a role for transition t is not created yet. This newly created role is responsible for connecting all joins and merges incoming into transition t before its actual execution. It holds synchronisation conditions imposed.
Figure 1: Walk through construction of PLD shown in Figure 3 for exemplar Petri Net shown in Figure 2: (a) AND forking through Multicast event, (b) Corresponding event catches for Multicast event, (c) Constructing transition $c$ in absence of AND join and XOR merge at $c$, (d) event throw $t$ after transition $c$ for AND join at transition $t$, (e) corresponding event catch $t$, (f) a selection guard for XOR split, (g) constructing transition $a$ in absence of AND join and XOR merge at $a$, (h) an event throw $t$ corresponding to AND join and XOR merge at transition $t$, (i) Constructing an event catch $t$ in series of previous event catch $t$, (j) Constructing the the other branch of the selection guard, (k) Constructing transition $t$, (l) Constructing multicast event throw for AND fork from transition $t$.

Figure 2: An exemplar PN for construction of corresponding PLD.

Figure 3: Process Line Diagram (PLD) for Figure 2 exemplar PN.

for actual execution of transition $t$ to occur. The mining process subsequently continues for the event catch $abc$ for the second fork, which originates from start and forks into $\{a, b\}$. Existence of multiple (here, two) transitions in this fork represents XOR split into transitions that are present in $\text{and\_split}(\text{start}) = \{a, b\}$. Ergo, the event catch $(abc)$ of second fork is followed by a selection guard as shown in Figure 1(f). The guard is split into two XOR split paths, which represent paths corresponding to multiplicity (transitions in the fork) i.e. $a$ and $b$ in this fork. One of these xor split paths, corresponding to transition $a$, is followed by immediate construction of transition $a$ due to absence of XOR merge and AND join at transition $a$. It is illustrated in Figure 1(g).

It can be observed using (2) and (4) that transition $a$ is involved in $\text{xor\_merge}(t) = \{a, b\}$ and $\text{and\_join}(t) = \{c\}, \{a, b\}$ i.e. XOR merge and AND join taking place at transition $t$. Consequently, an event throw $t$ is constructed next to transition $a$ as shown in Figure 1(h). Notably, despite involvement of transition $a$ in both XOR merge and AND join, event throw $t$ is constructed only once. A corresponding event catch $t$ is constructed to continue the role that holds all joins and merges for transition $t$, which occur before its actual execution, as presented in Figure 1(i). Likewise, Figure 1(j) illustrates construction of the other XOR split path, corresponding to transition $b$. Transition $b$ is constructed, and followed by construction of a subsequent event throw due to (2) and (4). It can be observed that a new event catch is not constructed, because an XOR merge requires a single event catch and it has already been created during construction of previous XOR split path. As shown in Figure 1(k), the transition $t$ is constructed for its actual execution, after all the merges and joins synchronizing its execution through event catches on its role are constructed.
The mining approach iterates to continue the flow of process construction. The AND fork, originating from transition $t$, and $\text{fork}(t) = \{z\}, \{x, y\}$, is constructed using multicast event $xyz$ as shown in Figure 1(l), and its corresponding event catches $xyz$ in two new roles to represent forking into $\{z\}$ and $\{x, y\}$ are then constructed as shown in Figure 3. This construction for AND fork relationship from transition $t$ into $z, y$ and $x$ is similar to that from start into $c$, $a$ and $b$. Thus, the remainder of process mining continues in similar manner to obtain PLD shown in Figure 3, with event catches $xyz$ followed by construction of transition $z$ and construction of selection guard depicting XOR split into transitions $y$ and $x$, which are subsequently constructed respectively on splits from the selection guard. These roles and split paths end in absence of further relations from transition $z$, $y$ and $x$ respectively.

### 3. Conclusion and Future Work

In contrast to traditional PN and allied extractor mining algorithms, which generate complex, non-modular outputs, the paper presented an idea that can retain inherent modularity inside a process during process mining. The paper explores extraction of processes in the form of Process Line Diagrams (PLDs), as compared to other process mining algorithms that mine processes as BPMN, Petri Nets, heuristic nets, process trees, Directly follows graphs, finite automatons, etc., which do not attempt to preserve modularity details. The paper utilizes the implementation-friendly modular modeling approach of PLDs for representation of extracted processes. It is illustrated through an exemplar Petri Net converted into PLD. An automated evaluation system for process line diagrams for testing volumes of logs is the next step in direction of PLD modeling and mining. PLD mining algorithms can be developed to utilize the entire toolset of process line diagrams in mined processes, and incorporate patterns given in [27]. Also, PLD mining algorithms can be extended to incorporate data related features of processes.

### References