Freezing Sub-Models During Incremental Process Discovery*

Daniel Schuster[1,2], Sebastiaan J. van Zelst[1,2], and Wil M. P. van der Aalst[1,2]

[1] Fraunhofer Institute for Applied Information Technology FIT, Germany
{daniel.schuster,sebastiaan.van.zelst,wil.van.der.aalst}@fit.fraunhofer.de
[2] RWTH Aachen University, Aachen, Germany

Abstract. Process discovery aims to learn a process model from observed process behavior. From a user's perspective, most discovery algorithms work like a *black box*. Besides parameter tuning, there is no interaction between the user and the algorithm. Interactive process discovery allows the user to exploit domain knowledge and to guide the discovery process. Previously, an incremental discovery approach has been introduced where a model, considered to be "under construction", gets incrementally extended by user-selected process behavior. This paper introduces a novel approach that additionally allows the user to *freeze* model parts within the model under construction. Frozen sub-models are not altered by the incremental approach when new behavior is added to the model. The user can thus steer the discovery algorithm. Our experiments show that freezing sub-models can lead to higher quality models.

Keywords: Process mining · Process discovery · Hybrid intelligence.

1 Introduction

Executing business processes generates valuable data in the information systems of organizations. *Process mining* comprises techniques to analyze these *event data* and aims to extract insights into the executed processes to improve them [1]. This paper focuses on *process discovery*, a key discipline within process mining.

Conventional process discovery algorithms use observed process behavior, i.e., event data, as input and return a process model that describes the process, as recorded by the event data. Since event data often have quality issues, process discovery is challenging. Apart from modifying the input (event data), the algorithm's settings, or the output (discovered model), there is no user interaction.

To overcome this limitation, the field of *interactive process discovery* has emerged. The central idea is to exploit the domain knowledge of process participants within process discovery in addition to the standard input of event data. Several techniques have been proposed. However, most existing approaches only attempt to use additional inputs besides the event data. Thus, a user still has only limited options to interact with the algorithm during the actual discovery phase, and the algorithm remains a black box from a user's perspective.

* An extended version is available online: https://arxiv.org/abs/2108.00215

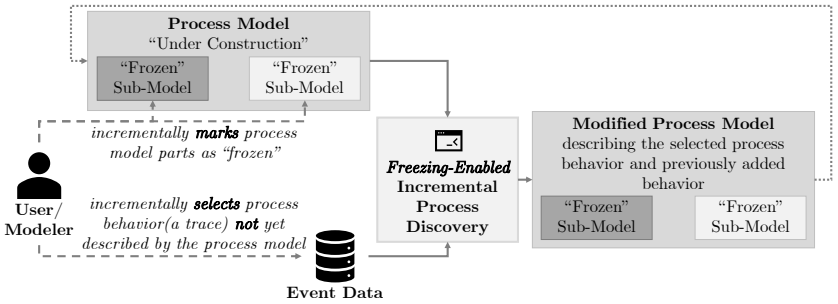


Fig. 1: Overview of the proposed freezing option extending *incremental* process discovery. A user incrementally selects traces from the event log and optionally freezes sub-models that should not get altered in the model "under construction"

In [11], we have introduced an incremental process discovery algorithm, allowing a user to incrementally add process behavior to a model under construction. This allows the user to control the algorithm by interactively deciding which process behavior to add next. In this context, we propose in this paper a novel way to interact with a discovery algorithm as a user. During the discovery phase, we allow a user to freeze sub-models of the model under construction. By marking sub-models as frozen, the incremental discovery approach does not alter these frozen model parts during the incremental discovery. Fig. 1 summarizes the proposed approach that can be applied with any incremental discovery algorithm.

There are many use cases where freezing sub models during incremental process discovery is beneficial. For instance, it enables a user to combine *de jure* and *de facto* models [1]. De jure models describe how a process should be executed (normative), and de facto models describe how a process was executed (descriptive). A user might freeze a model part because, from the user's perspective, the sub-model to be frozen is already normative. Therefore, a user wants to protect this sub-model from being further altered while incrementally adding new behavior to the model under construction. Thus, the proposed freezing approach allows combining process discovery with modeling. Our conducted experiments show that freezing sub-models can lead to higher quality models.

This paper is structured as follows. Section 2 presents related work while Section 3 presents preliminaries. Section 4 presents the proposed freezing approach. Section 5 presents an evaluation, and Section 6 concludes this paper.

2 Related Work

For an overview of process mining and conventional process discovery, we refer to [1]. Hereinafter, we mainly focus on interactive process discovery.

In [7], the authors propose to incorporate precedence constraints over the activities within process discovery. In [4], an approach is presented where an already existing process model is post-processed s.t. user-defined constraints are

Table 1: Example of an event log from an e-commerce process

Case-ID	Activity	Timestamp	···
151	place order (p)	10/03/21 12:00	···
153	cancel order (c)	10/03/21 12:24	···
152	place order (p)	11/03/21 09:11	···
151	payment received (r)	11/03/21 10:00	···
···	···	···	···

fulfilled. In [10], an approach is presented where domain knowledge in form of an initial process model is given. Compared to our extended incremental process discovery, all approaches remain a black-box from a user's perspective since they work in a fully automated fashion. In [5], an interactive Petri net modeling approach is proposed in which the user is supported by an algorithm.

Related work can also be found in the area of process model repair [6]. However, the setting of model repair, which attempts to make the repaired model as similar as possible to the original, differs from incremental process discovery. In [2] an interactive and incremental repair approach is proposed.

3 Preliminaries

We denote the power set of a set X by $\mathcal{P}(X)$. We denote the universe of multi-sets over a set X by $\mathcal{B}(X)$ and the set of all sequences over X as X^*, e.g., $\langle a,b,b\rangle \in \{a,b,c\}^*$. Given two sequences σ and σ', we denote their concatenation by $\sigma \cdot \sigma'$, e.g., $\langle a\rangle \cdot \langle b,c\rangle = \langle a,b,c\rangle$. We extend the $\cdot$ operator to sets of sequences, i.e., let $S_1, S_2 \subseteq X^*$ then $S_1 \cdot S_2 = \{\sigma_1 \cdot \sigma_2 \mid \sigma_1 \in S_1 \wedge \sigma_2 \in S_2\}$. For sequences σ, σ', the set of all interleaved sequences is denoted by $\sigma \diamond \sigma'$, e.g., $\langle a,b\rangle \diamond \langle c\rangle = \{\langle a,b,c\rangle, \langle a,c,b\rangle, \langle c,a,b\rangle\}$. We extend the $\diamond$ operator to sets of sequences. Let $S_1, S_2 \subseteq X^*$, $S_1 \diamond S_2$ denotes the set of interleaved sequences, i.e., $S_1 \diamond S_2 = \bigcup_{\sigma_1 \in S_1, \sigma_2 \in S_2} \sigma_1 \diamond \sigma_2$.

For $\sigma \in X^*$ and $X' \subseteq X$, we define the projection function $\sigma_{\downarrow_{X'}} : X^* \to (X')^*$ with: $\langle\rangle_{\downarrow_{X'}} = \langle\rangle$, $\big(\langle x\rangle \cdot \sigma\big)_{\downarrow_{X'}} = \langle x\rangle \cdot \sigma_{\downarrow_{X'}}$ if $x \in X'$ and $(\langle x\rangle \cdot \sigma)_{\downarrow_{X'}} = \sigma_{\downarrow_{X'}}$ otherwise.

Let $t = (x_1, \ldots, x_n) \in X_1 \times \ldots \times X_n$ be an n-tuple over n sets. We define projection functions that extract a specific element of t, i.e., $\pi_1(t) = x_1, \ldots, \pi_n(t) = x_n$, e.g., $\pi_2((a,b,c)) = b$.

3.1 Event Data and Process Models

The data that are generated during the execution of (business) processes are called *event data* [1]. Table 1 shows an example of an event log. Each row represents an event. Events with the same case-id belong to the same process execution often referred to as a *case*. The sequence of executed activities for a case is referred to as a *trace*, e.g., the partial trace for case 151 is: $\langle p, r, \ldots\rangle$.

Process models allow us to specify the control flow of a process. In this paper, we use process trees [1], e.g., see Fig. 2. Leaves represent activities and τ represents an unobservable activity, needed for certain control flow patterns. Inner nodes represent operators that specify the control flow among their subtrees. Four operators exist: *sequence* ($\rightarrow$), *excl. choice* ($\times$), *parallel* ($\wedge$), and *loop* ($\circlearrowleft$).

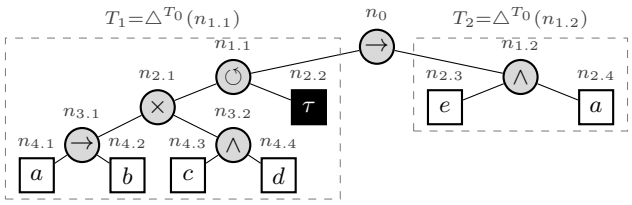


Fig. 2: Process tree $T_0{=}\big(\{n_o,\ldots,n_{4.4}\},\big\{(n_0,n_{1.1}),\ldots,(n_{3.2},n_{4.4})\big\},\lambda,n_0\big)$ with $\lambda(n_0){=}\rightarrow,\ldots,\lambda(n_{4.4}){=}d$

Definition 1 (Process Tree Syntax). *Let $\mathcal{A}$ be the universe of activities with $\tau{\notin}\mathcal{A}$. Let $\bigoplus{=}\{\rightarrow,\times,\wedge,\circlearrowleft\}$ be the set of process tree operators. We define a process tree $T{=}(V,E,\lambda,r)$ consisting of a totally ordered set of nodes V, a set of edges $E{\subseteq}V{\times}V$, a labeling function $\lambda{:}V{\rightarrow}\mathcal{A}{\cup}\{\tau\}{\cup}\bigoplus$, and a root node $r{\in}V$.*

- $\big(\{n\},\{\},\lambda,n\big)$ *with* $\lambda(n){\in}\mathcal{A}{\cup}\{\tau\}$ *is a process tree*
- *given $k{>}1$ trees $T_1{=}(V_1,E_1,\lambda_1,r_1),\ldots,T_k{=}(V_k,E_k,\lambda_k,r_k)$ with $r{\notin}V_1{\cup}\ldots{\cup}V_k$ and $\forall i,j{\in}\{1,\ldots,k\}(i{\neq}j\Rightarrow V_i{\cap}V_j{=}\emptyset)$ then $T{=}(V,E,\lambda,r)$ is a tree s.t.:*
  - $V{=}V_1{\cup}\ldots{\cup}V_k{\cup}\{r\}$
  - $E{=}E_1{\cup}\ldots{\cup}E_k{\cup}\big\{(r,r_1),\ldots,(r,r_k)\big\}$
  - $\lambda(x){=}\lambda_j(x)$ *for all* $j{\in}\{1,\ldots,k\},x{\in}V_j$
  - $\lambda(r){\in}\bigoplus$ *and* $\lambda(r){=}\circlearrowleft\Rightarrow k{=}2$

We denote the universe of process trees by $\mathcal{T}$.

Note that every operator (inner node) has at least two children except for the loop operator which always has exactly two children. Next to the graphical representation, any process tree can be textually represented because of its totally ordered node set, e.g., $T_0{\widehat{=}}{\rightarrow}\big(\circlearrowleft\big(\times\big(\rightarrow(a,b),\wedge(c,d)\big),\tau\big),\wedge(e,a)\big)$.

Given two process trees $T_1,T_2{\in}\mathcal{T}$, we write $T_1{\sqsubseteq}T_2$ if T_1 is a *subtree* of T_2. For instance, $T_1{\sqsubseteq}T_0$ and $T_1{\not\sqsubseteq}T_2$ in Fig. 2. The child function $c^T{:}V{\rightarrow}V^*$ returns a sequence of child nodes according to the order of V, i.e., $c^T(v){=}\langle v_1,\ldots,v_j\rangle$ s.t. $(v,v_1),\ldots,(v,v_j){\in}E$. For instance, $c_0^T(n_{1.1}){=}\langle n_{2.1},n_{2.2}\rangle$. For $T{=}(V,E,\lambda,r){\in}\mathcal{T}$ and $v{\in}V$, $\triangle^T(v)$ returns the with root node v. For example, $\triangle^{T_0}(n_{1.1}){=}T_1$.

For $T{=}(V,E,\lambda,r)$ and nodes $n_1,n_2{\in}V$, we define the *lowest common ancestor (LCA)* as $LCA(n_1,n_2){=}n{\in}V$ such that for $\triangle^T(n){=}(V_n,E_n,\lambda_n,r_n)$ $n_1,n_2{\in}V_n$ and the distance (number of edges) between n and r is maximal. For example, $LCA(n_{4.4},n_{2.2}){=}n_{1.1}$ and $LCA(n_{4.4},n_{2.3}){=}n_0$ (Fig. 2).

Next, we define running sequences and the language of process trees.

Definition 2 (Process Tree Running Sequences). *For the universe of activities $\mathcal{A}$ (with $\tau,open,close{\notin}\mathcal{A}$), $T{=}(V,E,\lambda,r){\in}\mathcal{T}$, we recursively define its running sequences $\mathcal{RS}(T){\subseteq}\big(V{\times}(\mathcal{A}{\cup}\{\tau\}{\cup}\{open,close\})\big)^*$.*

- *if $\lambda(r){\in}\mathcal{A}{\cup}\{\tau\}$ (T is a leaf node): $\mathcal{RS}(T){=}\big\{\big\langle(r,\lambda(r))\big\rangle\big\}$*
- *if $\lambda(r){=}\rightarrow$ with child nodes $c^T(r){=}\langle v_1,\ldots,v_k\rangle$ for $k{\geq}1$:*
  $\mathcal{RS}(T){=}\big\{\big\langle(r,open)\big\rangle\big\}{\cdot}\mathcal{RS}(\triangle^T(v_1)){\cdot}\ldots{\cdot}\mathcal{RS}(\triangle^T(v_k)){\cdot}\big\{\big\langle(r,close)\big\rangle\big\}$

$\gg$	$\gg$	$\gg$	a	b	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	c	$\gg$	f	$\gg$	$\gg$	$\gg$
$(n_{1.1},$ *open*$)$	$(n_{2.1},$ *open*$)$	$(n_{3.1},$ *open*$)$	$(n_{4.1},$ $a)$	$(n_{4.2},$ $b)$	$(n_{3.1},$ *close*$)$	$(n_{2.1},$ *close*$)$	$(n_{2.2},$ $\tau)$	$(n_{2.1},$ *open*$)$	$(n_{3.2},$ *open*$)$	$(n_{4.3},$ $c)$	$(n_{4.4},$ $d)$	$\gg$	$(n_{3.2},$ *close*$)$	$(n_{2.1},$ *close*$)$	$(n_{1.1},$ *close*$)$

Fig. 3: Optimal alignment $\gamma{=}\big\langle\big(\gg,(n_{1.1},\mathit{open})\big),\ldots,\big(\gg,(n_{1.1},\mathit{close})\big)\big\rangle$ for the trace $\langle a,b,c,f\rangle$ and the process tree T_1 (Fig. 2)

- *if* $\lambda(r){=}\times$ *with child nodes* $c^T(r){=}\langle v_1,\ldots,v_k\rangle$ *for* $k{\geq}1$*:*
  $\mathcal{RS}(T){=}\big\{\big\langle(r,\mathit{open})\big\rangle\big\}\cdot\big\{\mathcal{RS}(\triangle^T(v_1))\cup\ldots\cup\mathcal{RS}(\triangle^T(v_k))\big\}\cdot\big\{\big\langle(r,\mathit{close})\big\rangle\big\}$
- *if* $\lambda(r){=}\wedge$ *with child nodes* $c^T(r){=}\langle v_1,\ldots,v_k\rangle$ *for* $k{\geq}1$*:*
  $\mathcal{RS}(T){=}\big\{\big\langle(r,\mathit{open})\big\rangle\big\}\cdot\big\{\mathcal{RS}(\triangle^T(v_1))\diamond\ldots\diamond\mathcal{RS}(\triangle^T(v_k))\big\}\cdot\big\{\big\langle(r,\mathit{close})\big\rangle\big\}$
- *if* $\lambda(r){=}\circlearrowleft$ *with child nodes* $c^T(r){=}\langle v_1,v_2\rangle$*:*
  $\mathcal{RS}(T){=}\big\{\big\langle(r,\mathit{open})\big\rangle\cdot\sigma_1\cdot\sigma_1'\cdot\sigma_2\cdot\sigma_2'\cdot\ldots\cdot\sigma_m\cdot\big\langle(r,\mathit{close})\big\rangle\ \mid\ m{\geq}1\wedge\forall 1{\leq}i{\leq}m\big(\sigma_i{\in}\mathcal{RS}(\triangle^T(v_1))\big)\wedge\forall 1{\leq}i{\leq}m{-}1\big(\sigma_i'{\in}\mathcal{RS}(\triangle^T(v_2))\big)\big\}$

Definition 3 (Process Tree Language). *For given* $T{\in}\mathcal{T}$*, we define its language by* $\mathcal{L}(T){:=}\big\{\big(\pi_2^*(\sigma)\big)_{\downarrow_{\mathcal{A}}}\mid\sigma{\in}\mathcal{RS}(T)\big\}{\subseteq}\mathcal{A}^*$.

For example, consider the running sequences of T_2 (Fig. 2), i.e., $\mathcal{RS}(T_2){=}\big\{\big\langle(n_{1.2},\mathit{open}),(n_{2.3},e),(n_{2.4},a)),(n_{1.2},\mathit{close})\big\rangle,\big\langle(n_{1.2},\mathit{open}),(n_{2.4},a),(n_{2.3},e),(n_{1.2},\mathit{close})\big\rangle\big\}$. Hence, this subtree describes the language $\mathcal{L}(T_2){=}\big\{\langle e,a\rangle,\langle a,e\rangle\big\}$.

3.2 Alignments

Alignments quantify deviations between observed process behavior (event data) and modeled behavior (process models) [3]. Fig. 3 shows an alignment for the trace $\langle a,b,c,f\rangle$ and T_1 (Fig. 2). Ignoring the skip-symbol $\gg$, the first row of an alignment always corresponds to the trace and the second row to a running sequence of the tree. In general, we distinguish four alignment move types.

1. **synchronous moves** (shown light-gray in Fig. 3) indicate *no* deviation
2. **log moves** (shown black in Fig. 3) indicate a *deviation*, i.e., the observed activity in the trace is not executable in the model (at this point)
3. **visible model moves** (shown dark-gray in Fig. 3) indicate a *deviation*, i.e., an activity not observed in the trace must be executed w.r.t. the model
4. **invisible** $(\tau,\mathit{open},\mathit{close})$ **model moves** (shown white in Fig. 3) indicate *no* deviation, i.e., opening or closing of a subtree or an executed τ leaf node

Since multiple alignments exist for a given tree and trace, we are interested in an *optimal alignment*, i.e., the number of log and visible model moves is minimal.

4 Freezing-Enabled Incremental Process Discovery

In Section 4.1, we formally define the problem of freezing sub-models during incremental discovery. Then, we introduce the proposed approach in Section 4.2.

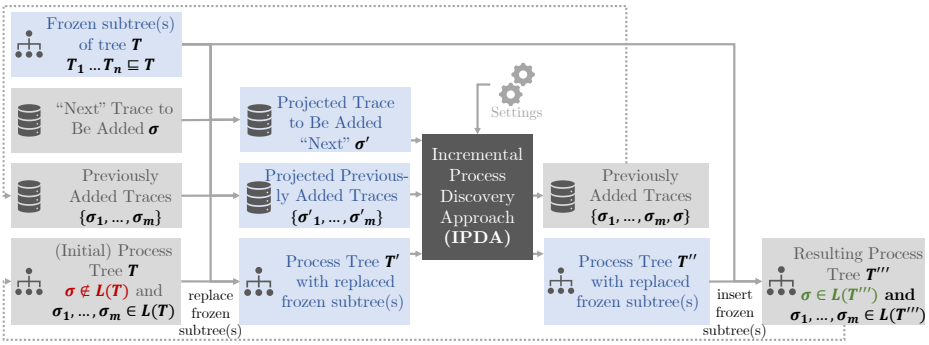


Fig. 4: Overview of the proposed freezing-enabled IPDA approach

4.1 Problem Definition

Reconsider Fig. 1 showing the overall framework of our proposal. A user incrementally selects subtrees from a process tree "under construction" and a trace σ from an event log. Both, the tree with frozen subtree(s) and the trace, are the input for an *freezing-enabled* incremental process discovery algorithm, which returns a modified tree that contains the frozen subtree(s) and accepts the selected trace. Next, we define an Incremental Process Discovery Algorithm (IPDA).

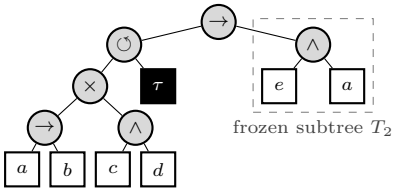
Definition 4 (IPDA). *$\alpha\colon \mathcal{T}\times\mathcal{A}^*\times\mathcal{P}(\mathcal{A}^*)\nrightarrow\mathcal{T}$ is an IPDA if for arbitrary $T\in\mathcal{T}$, $\sigma\in\mathcal{A}^*$, and previously added traces $\mathbf{P}\in\mathcal{P}(\mathcal{A}^*)$ with $\mathbf{P}\subseteq\mathcal{L}(T)$ it holds that $\{\sigma\}\cup\mathbf{P}\subseteq\mathcal{L}\big(\alpha(T,\sigma,\mathbf{P})\big)$. If $\mathbf{P}\not\subseteq\mathcal{L}(T)$, α is undefined.*

Starting from an (initial) tree T, a user incrementally selects a trace σ not yet described by T. The algorithm alters the process tree T into T' that accepts σ and the previously selected/added traces. T' is then used as input for the next incremental execution. For a specific example of an IPDA, we refer to our previous work [11]. Next, we formally define a freezing-enabled IPDA.

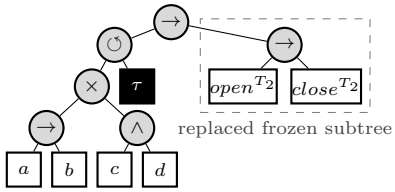
Definition 5 (Freezing-Enabled IPDA). *$\alpha_f\colon \mathcal{T}\times\mathcal{A}^*\times\mathcal{P}(\mathcal{A}^*)\times\mathcal{P}(\mathcal{T})\nrightarrow\mathcal{T}$ is a freezing-enabled IPDA if for arbitrary $T\in\mathcal{T}$, $\sigma\in\mathcal{A}^*$, previously added traces $\mathbf{P}\in\mathcal{P}(\mathcal{A}^*)$ with $\mathbf{P}\subseteq\mathcal{L}(T)$, and $n\geq 0$ frozen subtrees $\mathbf{T}=\{T_1,\ldots,T_n\}\in\mathcal{P}(\mathcal{T})$ s.t. $\forall i,j\in\{1,\ldots,n\}(T_i\sqsubseteq T\wedge i\neq j\Rightarrow T_i\not\sqsubseteq T_j)$ it holds that $\{\sigma\}\cup\mathbf{P}\subseteq\mathcal{L}\big(\alpha_f(T,\sigma,\mathbf{P},\mathbf{T})\big)$ and $\forall T'\in\mathbf{T}\big(T'\sqsubseteq\alpha_f(T,\sigma,\mathbf{P},\mathbf{T})\big)$.*
If $\mathbf{P}\not\subseteq\mathcal{L}(T)$ or $\exists i,j\in\{1,\ldots,n\}(T_i\not\sqsubseteq T\vee i\neq j\Rightarrow T_i\sqsubseteq T_j)$, α_f is undefined.

4.2 Approach

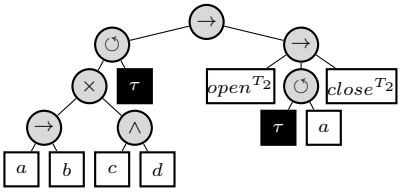
This section presents the proposed freezing approach, i.e., a freezing-enabled IPDA, that is based on an arbitrary, non-freezing-enabled IPDA. The central idea is to modify the input and output artefacts of an non-freezing-enabled IPDA. Thus, the proposed freezing approach is compatible with any IPDA. Fig. 4 provides an overview of the proposed approach. The remainder of this section is structured along the input/output modifications shown in Fig. 4.



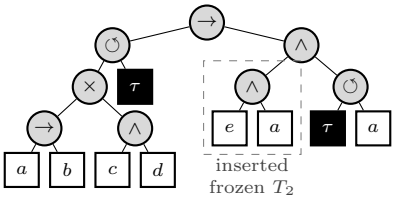
(a) Initial tree T (same as shown in Fig. 2) with frozen subtree T_2



(b) Tree T' with replaced frozen subtree



(c) Tree T'' after applying an IPDA



(d) Tree T''' containing frozen subtree T_2

Fig. 5: Running example of the freezing approach. Previously added traces: $\{\sigma_1{=}\langle d, c, a, b, a, e\rangle, \sigma_2{=}\langle a, b, e, a\rangle\}$. Trace to be added next: $\sigma{=}\langle c, d, a, e, a, a, e\rangle$

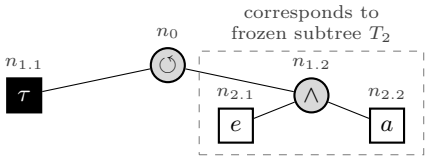
Replacing Frozen Subtrees As shown in Fig. 4, we assume an (initial) tree T with frozen subtrees $T_1, \ldots, T_n \sqsubseteq T$ and return a modified tree T'. For example, Fig. 5a shows the tree T (same as in Fig. 2) with the frozen subtree T_2. To replace T_2, we choose two unique labels which are neither in the event log nor in the tree, e.g., $open^{T_2}$ and $close^{T_2}$. Next, we replace T_2 by $\rightarrow(open^{T_2}, close^{T_2})$ and get T' (Fig. 5b). Semantically, $open^{T_2}$ represents the opening and $close^{T_2}$ the closing of T_2. In general, we iteratively replace each frozen subtree.

Projecting Previously Added Traces The set of previously added traces $\{\sigma_1, \ldots, \sigma_m\}$ (Fig. 4), which fits the tree T, does not fit T' because of the replaced frozen subtree(s). Thus, we have to modify the traces accordingly.

We replay each trace $\{\sigma_1, \ldots, \sigma_m\}$ on T and mark when a frozen subtree is *opened* and *closed*. Next, we insert in these traces the corresponding replacement label whenever a frozen subtree was opened/closed and remove all activities in between that are replayed in a frozen subtree. Other activities remain unchanged. For example, reconsider T (Fig. 5) and its frozen subtree T_2 that was replaced by $\rightarrow(open^{T_2}, close^{T_2})$. Assume the traces $\{\sigma_1{=}\langle d, c, a, b, a, e\rangle, \sigma_2{=}\langle a, b, e, a\rangle\}$. Below, we show the running sequence of σ_1 on T and the projected trace σ_1'.

extract of the running sequence for σ_1 on $T{=}T_0$ (see Fig. 2):
$\langle \ldots (n_{4.4}, d), (n_{4.3}, c) \ldots (n_{4.1}, a), (n_{4.2}, b) \ldots (n_{1.2}, open), (n_{2.4}, a), (n_{2.3}, e), (n_{1.2}, close) \ldots \rangle$
projected trace σ_1' based on above running sequence:
$\langle d, \quad c, \quad a, \quad b, \quad open^{T_2}, \quad close^{T_2} \rangle$

We transform $\sigma_1{=}\langle d, c, a, b, a, e\rangle$ into $\sigma_1'{=}\langle d, c, a, b, open^{T_2}, close^{T_2}\rangle$ (and σ_2 into $\sigma_2'{=}\langle a, b, open^{T_2}, close^{T_2}\rangle$). Note that $\sigma_1', \sigma_2' {\in} \mathcal{L}(T')$ since $\sigma_1, \sigma_2 {\in} \mathcal{L}(T)$.



(a) Abstraction tree A used to detect full executions of frozen tree T_2 (Fig. 2)

move index	1	2	3	4	5	6	7	8	9	10	11	12	13	14
trace	c	d	$\gg$	$\gg$	a	e	$\gg$	$\gg$	$\gg$	a	a	e	$\gg$	$\gg$
model	$\gg$	$\gg$	$(n_0,$ *open*$)$	$(n_{1.2},$ *open*$)$	$(n_{2.2},$ $a)$	$(n_{2.1},$ $e)$	$(n_{1.2},$ *close*$)$	$(n_{1.1},$ $\tau)$	$(n_{1.2},$ *open*$)$	$(n_{2.2},$ $a)$	$\gg$	$(n_{2.1},$ $e)$	$(n_{1.2},$ *close*$)$	$(n_0,$ *close*$)$

(b) Optimal alignment of $\sigma{=}\langle c, d, a, e, a, a, e\rangle$ and abstraction tree A

Fig. 6: Detecting full executions of T_2 (Fig. 2) in $\sigma{=}\langle c, d, a, e, a, a, e\rangle$

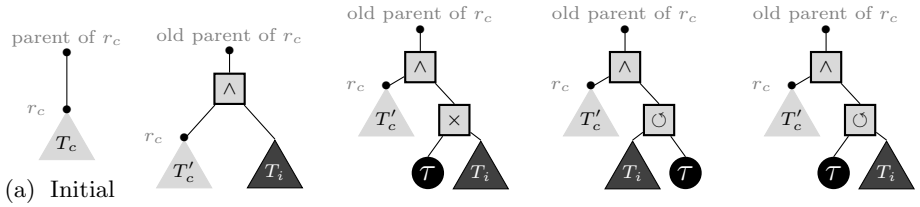
Projecting Trace to Be Added Next The idea is to detect full executions of the frozen subtree(s) within the trace to be added next and to replace these full executions by the corresponding replacement labels.

Reconsider Fig. 5 and the trace to be added next $\sigma{=}\langle c, d, a, e, a, a, e\rangle$. To detect full executions of the frozen subtree $T_2 \widehat{=} \wedge(e, a)$ independent from the entire tree T, we align σ with the abstraction tree $A \widehat{=} \circlearrowleft(\tau, \wedge(e, a))$, cf. Fig. 6a. The alignment (cf. Fig. 6b) shows that T_2 is twice fully executed, i.e, 4-7 and 9-13 move. Thus, we project σ onto $\sigma'{=}\langle c, d, \mathit{open}^{T_2}, \mathit{close}^{T_2}, \mathit{open}^{T_2}, a, \mathit{close}^{T_2}\rangle$.

Reinserting Frozen Subtrees This section describes how the frozen subtree(s) are reinserted into T'', returned by the IPDA (Fig. 4). Note that T'' can contain the replacement label for opening and closing of a frozen subtree multiple times because the IPDA may add leaf nodes having the same label. Thus, we have to find appropriate position(s) in T'' to insert the frozen subtree(s) back.

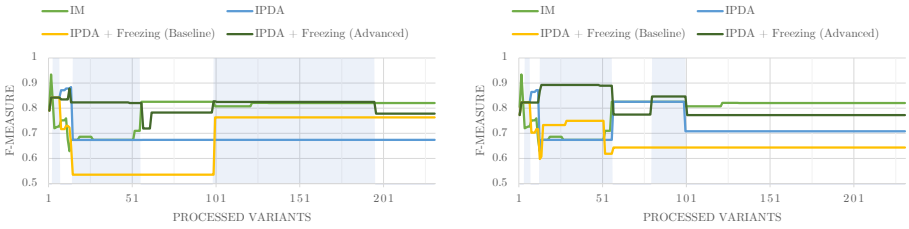
For example, reconsider Fig. 5c. We receive $T'' \widehat{=} \rightarrow\big(\wedge\big(\times\big(\rightarrow(a, b), \wedge(c, d)\big), \tau\big), \rightarrow(\mathit{open}^{T_2}, \circlearrowleft(\tau, a), \mathit{close}^{T_2})\big)$. We observe that between opening (open^{T_2}) and closing (close^{T_2}) of T_2, a loop on a was inserted. First, we calculate the LCA of open^{T_2} and close^{T_2}, i.e., the subtree $\rightarrow(\mathit{open}^{T_2}, \circlearrowleft(\tau, a), \mathit{close}^{T_2})$. Next, we do a semantic analysis of this subtree to determine how often open^{T_2} and close^{T_2} can be replayed. This analysis is needed because the IPDA changes the tree and open^{T_2} or close^{T_2} could be now skipped or executed multiple times. In T'', open^{T_2} and close^{T_2} must be executed exactly once. Hence, we apply the case $\{1\}$ visualized in Fig. 7b where T_i represents the frozen subtree and T'_c the LCA subtree after removing nodes labelled with open^{T_2} and close^{T_2}. We obtain T''' (Fig. 5d) that contains the frozen subtree T_2 and accepts the traces $\{\sigma, \sigma_1, \sigma_2\}$.

In general (cf. Fig. 4), we iteratively insert the frozen subtrees $\{T_1, \ldots, T_n\}$ back. For $T_i{\in}\{T_1, \ldots, T_n\}$, we calculate the LCA from all nodes in T'' that are labeled with the replacement label of T_i. Next, we do a semantic analysis of the LCA to determine how often T_i has to be executed. This semantic analysis results in one of the four cases shown in Fig. 7, which specify how the frozen subtree needs to be inserted back into T''.



(a) Initial situation (b) Case $\{1\}$ (c) Case $\{0, 1\}$ (d) Case $\{1, \infty\}$ (e) Case $\{0, \infty\}$

Fig. 7: Four cases showing how to insert a frozen subtree T_i back



(a) F-measure for experiment `rtfm_7` [3] (b) F-measure for experiment `rtfm_4` [3]

Fig. 8: F-measure for a real-life event log [9] using two different initial process models, each with a different frozen subtree. We refer to the proposed approach in this paper as *IPDA + Freezing (Advanced)*. Highlighted segments indicate that the proposed approach outperforms the other evaluated algorithms

5 Evaluation

This section presents an experimental evaluation. We compare four different discovery approaches: the Inductive Miner (a conventional discovery algorithm) [8], an IPDA [11], a baseline freezing approach (described in the extended version of this paper) using the IPDA in [11], and the proposed freezing approach (Section 4.2) using the IPDA in [11]. All four approaches guarantee replay fitness, i.e., traces given to the algorithm are accepted by the resulting tree. We use a publicly available event log [9]. We use the same initial model for all IPDA approaches per run. Further, we do not change the frozen subtree during incremental discovery. More detailed data of the experiments are available online[3].

Fig. 8 shows the F-measure of the incremental discovered trees based on the entire event log. We observe that the proposed advanced freezing approach clearly dominates the baseline freezing approach in both runs. Further, we observe that the advanced freezing approach outperforms the other approaches in the highlighted areas (Fig. 8a). Note that in reality, *incorporating all observed process behavior is often not desired* because the event data contains noise, incomplete behavior and other types of quality issues. For instance, after integrating the first 17 most frequent trace-variants of the RTFM log, the process

[3] https://github.com/fit-daniel-schuster/Freezing-Sub-Models-During-Incr-PD

model covers already 99% of the observed process behavior/traces. Comparing IPDA with the proposed advanced freezing approach (Fig. 8), we observe that the advanced freezing approach clearly dominates IPDA in most segments. In general, the results indicate that freezing subtrees during incremental process discovery can lead to higher quality models since we observe that the advanced freezing approach dominates the other algorithms in many segments.

6 Conclusion

This paper introduced a novel option to interact with a process discovery algorithm. By being able to freeze parts of a process model during incremental process discovery, the user is able to steer the algorithm. Moreover, the proposed approach combines conventional process discovery with data-driven process modeling. In the future, we plan to explore strategies that recommend appropriate freezing candidates to the user. Further, we plan to integrate the proposed approach into the incremental process discovery tool *Cortado* [12].

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action. Springer (2016)
2. Armas Cervantes, A., van Beest, N.R.T.P., La Rosa, M., Dumas, M., García-Bañuelos, L.: Interactive and incremental business process model repair. In: On the Move to Meaningful Internet Systems. Springer (2017)
3. Carmona, J., van Dongen, B.F., Solti, A., Weidlich, M.: Conformance Checking - Relating Processes and Models. Springer (2018)
4. Dixit, P.M., Buijs, J.C.A.M., van der Aalst, W.M.P., Hompes, B.F.A., Buurman, J.: Using domain knowledge to enhance process mining results. In: SIMPDA. LNBIP, vol. 244. Springer (2015)
5. Dixit, P.M., Verbeek, H.M.W., Buijs, J.C.A.M., van der Aalst, W.M.P.: Interactive data-driven process model construction. In: Conceptual Modeling, Proceedings. LNCS, vol. 11157. Springer (2018)
6. Fahland, D., van der Aalst, W.M.P.: Repairing process models to reflect reality. In: Business Process Management. Springer (2012)
7. Greco, G., Guzzo, A., Lupia, F., Pontieri, L.: Process discovery under precedence constraints. ACM Trans. Knowl. Discov. Data **9**(4) (2015)
8. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - A constructive approach. In: Application and Theory of Petri Nets and Concurrency. LNCS, vol. 7927. Springer (2013)
9. de Leoni, M., Mannhardt, F.: Road traffic fine management process (2015), https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5
10. Rembert, A.J., Omokpo, A., Mazzoleni, P., Goodwin, R.T.: Process discovery using prior knowledge. In: Service-Oriented Computing. Springer (2013)
11. Schuster, D., van Zelst, S.J., van der Aalst, W.M.P.: Incremental discovery of hierarchical process models. In: Research Challenges in Information Science. LNBIP, vol. 385. Springer (2020)
12. Schuster, D., van Zelst, S.J., van der Aalst, W.M.P.: Cortado—an interactive tool for data-driven process discovery and modeling. In: Application and Theory of Petri Nets and Concurrency. LNCS, vol. 12734. Springer (2021)