# Alignment Approximation for Process Trees

Daniel Schuster[1], Sebastiaan 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}@fit.fraunhofer.de
[2] RWTH Aachen University, Germany
wvdaalst@pads.rwth-aachen.de

**Abstract.** Comparing observed behavior (event data generated during process executions) with modeled behavior (process models), is an essential step in process mining analyses. Alignments are the de-facto standard technique for calculating conformance checking statistics. However, the calculation of alignments is computationally complex since a shortest path problem must be solved on a state space which grows non-linearly with the size of the model and the observed behavior, leading to the well-known *state space explosion problem*. In this paper, we present a novel framework to approximate alignments on process trees by exploiting their hierarchical structure. Process trees are an important process model formalism used by state-of-the-art process mining techniques such as the inductive mining approaches. Our approach exploits structural properties of a given process tree and splits the alignment computation problem into smaller sub-problems. Finally, sub-results are composed to obtain an alignment. Our experiments show that our approach provides a good balance between accuracy and computation time.

**Keywords:** Process mining · Conformance checking · Approximation.

## 1 Introduction

*Conformance checking* is a key research area within process mining [1]. The comparison of observed process behavior with reference process models is of crucial importance in process mining use cases. Nowadays, *alignments* [2] are the de-facto standard technique to compute conformance checking statistics. However, the computation of alignments is complex since a shortest path problem must be solved on a non-linear state space composed of the reference model and the observed process behavior. This is known as the *state space explosion problem* [3]. Hence, various approximation techniques have been introduced. Most techniques focus on decomposing Petri nets or reducing the number of alignments to be calculated when several need to be calculated for the same process model [4–8].

In this paper, we focus on a specific class of process models, namely process trees (also called *block-structured* process models), which are an important process model formalism that represent a subclass of sound *Workflow nets* [9]. For instance, various state-of-the-art process discovery algorithms return process

trees [9–11]. In this paper, we introduce an alignment approximation approach for process trees that consists of two main phases. First, our approach splits the problem of alignments into smaller sub-problems along the tree hierarchy. Thereby, we exploit the hierarchical structure of process trees and their semantics. Moreover, the definition of sub-problems is based on a *gray-box view* on the corresponding subtrees since we use a simplified/abstract view on the subtrees to recursively define the sub-problems along the tree hierarchy. Such sub-problems can then be solved individually and in parallel. Secondly, we recursively compose an alignment from the sub-results for the given process tree and observed process behavior. Our experiments show that our approach provides a good balance between accuracy and computation effort.

The remainder is structured as follows. In Section 2, we present related work. In Section 3, we present preliminaries. In Section 4, we present the formal framework of our approach. In Section 5, we introduce our alignment approximation approach. In Section 6, we present an evaluation. Section 7 concludes the paper.

## 2 Related Work

In this section, we present related work regarding alignment computation and approximation. For a general overview of conformance checking, we refer to [3].

Alignments have been introduced in [2]. In [12] it was shown that the computation is reducible to a shortest path problem and the solution of the problem using the A* algorithm is presented. In [13], the authors present an improved heuristic that is used in the shortest path search. In [14], an alignment approximation approach based on approximating the shortest path is presented.

A generic approach to decompose Petri nets into multiple sub-nets is introduced in [15]. Further, the application of such decomposition to alignment computation is presented. In contrast to our approach, the technique does not return an alignment. Instead, only partial alignments are calculated, which are used, for example, to approximate an overall fitness value. In [4], an approach to calculate alignments based on Petri net decomposition [15] is presented that additionally guarantees optimal fitness values and optionally returns an alignment. Comparing both decomposition techniques with our approach, we do not calculate sub-nets because we simply use the given hierarchical structure of a process tree. Moreover, our approach always returns a valid alignment.

In [5], an approach is presented that approximates alignments for an event log by reducing the number of alignments being calculated based on event log sampling. Another technique based on event log sampling is presented in [8] where the authors explicitly approximate conformance results, e.g., fitness, rather than alignments. In contrast to our proposed approach, alignments are not returned. In [6] the authors present an approximation approach that explicitly focuses on approximating multiple optimal alignments. Finally, in [7], the authors present a technique to reduce a given process model and an event log s.t. the original behavior of both is preserved as much as possible. In contrast, the proposed approach in this paper does not modify the given process model and event log.

Table 1: Example of an event log from an order process

| Event-id | Case-id | Activity name | Timestamp | ··· |
|---|---|---|---|---|
| ··· | ··· | ··· | ··· | ··· |
| 200 | 13 | create order (c) | 2020-01-02 15:29 | ··· |
| 201 | 27 | receive payment (r) | 2020-01-02 15:44 | ··· |
| 202 | 43 | dispatch order (d) | 2020-01-02 16:29 | ··· |
| 203 | 13 | pack order (p) | 2020-01-02 19:12 | ··· |
| ··· | ··· | ··· | ··· | ··· |

## 3 Preliminaries

We denote the power set of a given set $X$ by $\mathcal{P}(X)$. A multi-set over a set $X$ allows multiple appearances of the same element. We denote the universe of multi-sets for a set $X$ by $\mathcal{B}(X)$ and the set of all sequences over $X$ as $X^*$, e.g., $\langle a,b,b\rangle \in \{a,b,c\}^*$. For a given sequence $\sigma$, we denote its length by $|\sigma|$. We denote the empty sequence by $\langle\rangle$. We denote the set of all possible permutations for given $\sigma \in X^*$ by $\mathbb{P}(\sigma) \subseteq X^*$. Given two sequences $\sigma$ and $\sigma'$, we denote the concatenation of these two sequences by $\sigma \cdot \sigma'$. 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 traces $\sigma, \sigma'$, 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 recursively 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 $(\langle x\rangle \cdot \sigma)_{\downarrow_{X'}} = \sigma_{\downarrow_{X'}}$ else.

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., $\pi_2((a,b,c)) = b$. Analogously, given a sequence of length $m$ with $n$-tuples $\sigma = \langle (x_1^1, \ldots, x_n^1), \ldots, (x_1^m, \ldots, x_n^m)\rangle$, we define $\pi_1^*(\sigma) = \langle x_1^1, \ldots, x_1^m\rangle, \ldots, \pi_n^*(\sigma) = \langle x_n^1, \ldots, x_n^m\rangle$. For instance, $\pi_2^*\big(\langle (a,b),(a,c),(b,a)\rangle\big) = \langle b,c,a\rangle$.

### 3.1 Event Logs

Process executions leave *event data* in information systems. An *event* describes the execution of an activity for a particular *case*/process instance. Consider Table 1 for an example of an *event log* where each event contains the executed activity, a timestamp, a case-id and potentially further attributes. Since, in this paper, we are only interested in the sequence of activities executed, we define an event log as a multi-set of sequences. Such sequence is also referred to as a *trace*.

**Definition 1 (Event log).** *Let $\mathcal{A}$ be the universe of activities. $L \in \mathcal{B}(\mathcal{A}^*)$ is an event log.*

### 3.2 Process Trees

Next, we define the syntax and semantics of process trees.

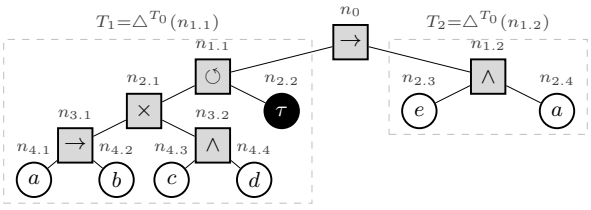



Fig. 1: Process tree $T_0{=}\big(\{n_o,\ldots,n_{4.4}\},\big\{(n_0,n_{1.1}),\ldots,(n_{3.2},n_{4.4})\big\},\lambda,n_0\big)$ with $\lambda(n_0){=}\rightarrow,\ldots,\lambda(n_{4.4}){=}d$

**Definition 2 (Process Tree Syntax).** *Let $\mathcal{A}$ be the universe of activities and $\tau{\notin}\mathcal{A}$. 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 $E$, 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* $k{>}1$ *process trees* $T_1{=}(V_1,E_1,\lambda_1,r_1),\ldots,T_k{=}(V_k,E_k,\lambda_k,r_k)$, $T{=}(V,E,\lambda,r)$ *is a process tree s.t.:*
  - $V{=}V_1{\cup}\ldots{\cup}V_k{\cup}\{r\}$ *(assume* $r{\notin}V_1{\cup}\ldots{\cup}V_k$*)*
  - $E{=}E_1{\cup}\ldots{\cup}E_k{\cup}\big\{(r,r_1),\ldots,(r,r_k)\big\}$
  - $\lambda(x){=}\lambda_j(x)\ \forall j{\in}\{1,\ldots,k\}\forall x{\in}V_j,\lambda(r){\in}\{\rightarrow,\wedge,\times\}$
- *given two process trees* $T_1{=}(V_1,E_1,\lambda_1,r_1)$ *and* $T_2{=}(V_2,E_2,\lambda_2,r_2)$, $T{=}(V,E,\lambda,r)$ *is a process tree s.t.:*
  - $V{=}V_1{\cup}V_2{\cup}\{r\}$ *(assume* $r{\notin}V_1{\cup}V_2$*)*
  - $E{=}E_1{\cup}E_2{\cup}\big\{(r,r_1),(r,r_2)\big\}$
  - $\lambda(x){=}\lambda_1(x)$ *if* $x{\in}V_1,\lambda(x){=}\lambda_2(x)$ *if* $x{\in}V_2,\lambda(r){=}\circlearrowleft$

In Figure 1, we depict an example process tree $T_0$ that can alternatively be represented textually due to the totally ordered node set, i.e., $T_0{\widehat{=}}{\rightarrow}({\circlearrowleft}({\times}({\rightarrow}(a,b),{\wedge}(c,d)),\tau),{\wedge}(e,a))$. We denote the universe of process trees by $\mathcal{T}$. The degree $d$ indicates the number of edges connected to a node. We distinguish between incoming $d^+$ and outgoing edges $d^-$, e.g., $d^+(n_{2.1}){=}1$ and $d^-(n_{2.1}){=}2$. For a tree $T{=}(V,E,\lambda,r)$, we denote its *leaf nodes* by $T^L{=}\{v{\in}V|d^-(v){=}0\}$. 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, $c^T(n_{1.1}){=}\langle n_{2.1},n_{2.2}\rangle$. For $T{=}(V,E,\lambda,r)$ and a node $v{\in}V$, $\triangle^T(v)$ returns the corresponding tree $T'$ s.t. $v$ is the root node, i.e., $T'{=}(V',E',\lambda',v)$. Consider $T_0$, $\triangle^{T_0}(n_{1.1}){=}T_1$ as highlighted in Figure 1. For process tree $T{\in}\mathcal{T}$, we denote its height by $h(T){\in}\mathbb{N}$.

**Definition 3 (Process Tree Semantics).** *For given $T{=}(V,E,\lambda,r){\in}\mathcal{T}$, we define its language $\mathcal{L}(T){\subseteq}\mathcal{A}^*$.*

- *if* $\lambda(r){=}a{\in}\mathcal{A}$, $\mathcal{L}(T){=}\{\langle a\rangle\}$
- *if* $\lambda(r){=}\tau$, $\mathcal{L}(T){=}\{\langle\rangle\}$
- *if* $\lambda(r){\in}\{\rightarrow,\times,\wedge\}$ *with* $c^T(r){=}\langle v_1,\ldots,v_k\rangle$
  - *with* $\lambda(r){=}\rightarrow$, $\mathcal{L}(T){=}\mathcal{L}(\triangle^T(v_1))\cdot\ldots\cdot\mathcal{L}(\triangle^T(v_k))$
  - *with* $\lambda(r){=}\wedge$, $\mathcal{L}(T){=}\mathcal{L}(\triangle^T(v_1))\diamond\ldots\diamond\mathcal{L}(\triangle^T(v_k))$

| *trace part* | $a$ | $b$ | $\gg$ | $\gg$ | $c$ | $f$ | $\gg$ | $\gg$ |
|---|---|---|---|---|---|---|---|---|
| *model part* | $n_{4.1}$ $\lambda(n_{4.1})=a$ | $n_{4.2}$ $\lambda(n_{4.2})=b$ | $n_{2.2}$ $\lambda(n_{2.2})=\tau$ | $n_{4.4}$ $\lambda(n_{4.4})=d$ | $n_{4.3}$ $\lambda(n_{4.3})=c$ | $\gg$ | $n_{2.4}$ $\lambda(n_{2.4})=a$ | $n_{2.3}$ $\lambda(n_{2.3})=e$ |

Fig. 2: Optimal alignment $\gamma=\big\langle(a,n_{4.1}),\ldots,(\gg,n_{2.3})\big\rangle$ for $\langle a,b,c,f\rangle$ and $T_0$

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

In this paper, we assume binary process trees as input for our approach, i.e, every node has two or none child nodes, e.g., $T_0$. Note that every process tree can be easily converted into a language equivalent binary process tree [9].

### 3.3 Alignments

*Alignments* [12] map observed behavior onto modeled behavior specified by process models. Figure 2 visualizes an alignment for the trace $\langle a,b,c,f\rangle$ and $T_0$ (Figure 1). The first row corresponds to the given trace ignoring the skip symbol $\gg$. The second row (ignoring $\gg$) corresponds to a sequence of leaf nodes s.t. the corresponding sequence of labels (ignoring $\tau$) is in the language of the process tree, i.e., $\langle a,b,d,c,a,e\rangle\in\mathcal{L}(T_0)$. Each column represents an alignment move. The first two are *synchronous moves* since the activity and the leaf node label are equal. The third and fourth are *model moves* because $\gg$ is in the log part. Moreover, the third is an *invisible* model move since the leaf node label is $\tau$ and the fourth is a *visible* model move since the label represents an activity. Visible model moves indicate that an activity should have taken place w.r.t. the model. The sixth is a log move since the trace part contains $\gg$. Log moves indicate observed behavior that should not occur w.r.t. the model. Note that we alternatively write $\gamma\widehat{=}\big\langle(a,a),\ldots,(\gg,e)\big\rangle$ using their labels instead of leaf nodes.

**Definition 4 (Alignment).** *Let $\mathcal{A}$ be the universe of activities, $\sigma\in\mathcal{A}^*$ be a trace and $T=(V,E,\lambda,r)\in\mathcal{T}$ be a process tree with leaf nodes $T^L$. Note that $\gg,\tau\notin\mathcal{A}$. A sequence $\gamma\in\big((\mathcal{A}\cup\{\gg\})\times(T^L\cup\{\gg\})\big)^*$ with length $n=|\gamma|$ is an alignment iff:*

1. $\sigma=\pi_1^*(\gamma)_{\downarrow_{\mathcal{A}}}$
2. $\Big\langle\lambda\Big(\pi_2\big(\gamma(1)\big)\Big),\ldots,\lambda\Big(\pi_2\big(\gamma(n)\big)\Big)\Big\rangle_{\downarrow_{\mathcal{A}}}\in\mathcal{L}(T)$
3. $(\gg,\gg)\notin\gamma$ *and* $(a,v)\notin\gamma\ \forall a\in\mathcal{A}\ \forall v\in T^L\big(a\neq\lambda(v)\big)$

For a given process tree and a trace, many alignments exist. Thus, costs are assigned to alignment moves. In this paper, we assume the *standard cost function*. Synchronous and invisible model moves are assigned cost 0, other moves are assigned cost 1. An alignment with minimal costs is called *optimal*. For a process tree $T$ and a trace $\sigma$, we denote the set of all possible alignments by $\Gamma(\sigma,T)$. In this paper, we assume a function $\alpha$ that returns for given $T\in\mathcal{T}$ and $\sigma\in\mathcal{A}^*$ an optimal alignment, i.e., $\alpha(\sigma,T)\in\Gamma(\sigma,T)$. Since process trees can be easily converted into Petri nets [1] and the computation of alignments for a Petri net was shown to be reducible to a shortest path problem [12], such function exists.

## 4 Formal Framework

In this section, we present a general framework that serves as the basis for the proposed approach. The core idea is to recursively divide the problem of alignment calculation into multiple sub-problems along the tree hierarchy. Subsequently, we recursively compose partial sub-results to an alignment.

Given a trace and tree, we recursively split the trace into sub-traces and assign these to subtrees along the tree hierarchy. During splitting/assigning, we regard the semantics of the current root node's operator. We recursively split until we can no longer split, e.g., we hit a leaf node. Once we stop splitting, we calculate optimal alignments for the defined sub-traces on the assigned subtrees, i.e., we obtain sub-alignments. Next, we recursively compose the sub-alignments to a single alignment for the parent subtree. Thereby, we consider the semantics of the current root process tree operator. Finally, we obtain a *valid*, but not necessarily optimal, alignment for the initial given tree and trace since we regard the semantics of the process tree during splitting/assigning and composing.

Formally, we can express the splitting/assigning as a function. Given a trace $\sigma \in \mathcal{A}^*$ and $T=(V, E, \lambda, r) \in \mathcal{T}$ with subtrees $T_1$ and $T_2$, $\psi$ splits the trace $\sigma$ into $k$ sub-traces $\sigma_1, \dots, \sigma_k$ and assigns each sub-trace to either $T_1$ or $T_2$.

$$\psi(\sigma, T) \in \Big\{ \big\langle (\sigma_1, T_{i_1}), \dots, (\sigma_k, T_{i_k}) \big\rangle \mid i_1, \dots, i_k \in \{1,2\} \wedge \sigma_1 \cdot \ldots \cdot \sigma_k \in \mathbb{P}(\sigma) \Big\} \quad (1)$$

We call a splitting/assignment *valid* if the following additional conditions are satisfied depending on the process tree operator:

- if $\lambda(r)=\times$: $k=1$
- if $\lambda(r)=\rightarrow$: $k=2 \wedge \sigma_1 \cdot \sigma_2 = \sigma$
- if $\lambda(r)=\wedge$: $k=2$
- if $\lambda(r)=\circlearrowleft$: $k \in \{1,3,5,\dots\} \wedge \sigma_1 \cdot \ldots \cdot \sigma_k = \sigma \wedge i_1 = 1 \wedge \forall j \in \{1, \dots, k-1\} \big( (i_j = 1 \Rightarrow i_{j+1} = 2) \wedge (i_j = 2 \Rightarrow i_{j+1} = 1) \big)$

Secondly, the calculated sub-alignments are recursively composed to an alignment for the respective parent tree. Assume a tree $T \in \mathcal{T}$ with sub-trees $T_1$ and $T_2$, a trace $\sigma \in \mathcal{A}^*$, a valid splitting/assignment $\psi(\sigma, T)$ , and a sequence of $k$ sub-alignments $\langle \gamma_1, \dots, \gamma_k \rangle$ s.t. $\gamma_j \in \Gamma(\sigma_j, T_{i_j})$ with $(\sigma_j, T_{i_j}) = \psi(\sigma, T)(j) \forall j \in \{1, \dots, k\}$. The function $\omega$ composes an alignment for $T$ and $\sigma$ from the given sub-alignments.

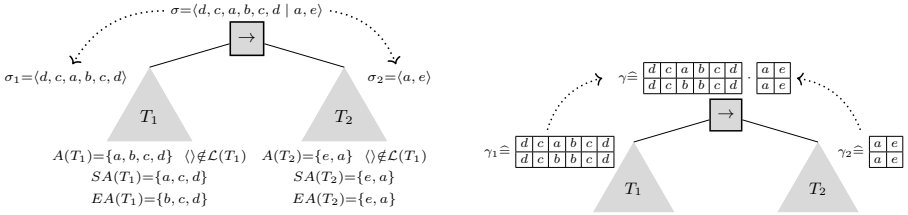
$$\omega(\sigma, T, \langle \gamma_1, \dots, \gamma_k \rangle) \in \{ \gamma \mid \gamma \in \Gamma(\sigma, T) \wedge \gamma_1 \cdot \ldots \cdot \gamma_k \in \mathbb{P}(\gamma) \} \quad (2)$$

By utilizing the definition of process tree semantics, it is easy to show that, given a valid splitting/assignment, such alignment $\gamma$ returned by $\omega$ always exists.

The overall, recursive approach is sketched in Algorithm 1. For a given tree $T$ and trace $\sigma$, we create a valid splitting/assignment (line 4). Next, we recursively call the algorithm on the determined sub-traces and subtrees (line 6). If given thresholds for trace length ($TL$) or tree height ($TH$) are reached, we stop splitting and return an optimal alignment (line 2). Hence, for the sub-traces created, we eventually obtain optimal sub-alignments, which we recursively compose to an alignment for the parent tree (line 7). Finally, we obtain a valid, but not necessarily optimal, alignment for $T$ and $\sigma$.

**Algorithm 1:** Approximate alignment

**input:** $T{=}(V, E, \lambda, r){\in}\mathcal{T}, \sigma{\in}\mathcal{A}^*, TL{\geq}1, TH{\geq}1$
**begin**
1 **if** $|\sigma|{\leq}TL \vee h(T){\leq}TH$ **then**
2 **return** $\alpha(\sigma, T)$; // optimal alignment
3 **else**
4 $\psi(\sigma, T){=}\langle(\sigma_1, T_{i_1}), \ldots, (\sigma_k, T_{i_k})\rangle$; // valid splitting
5 **for** $(\sigma_j, T_{i_j}){\in}\langle(\sigma_1, T_{i_1}), \ldots, (\sigma_k, T_{i_k})\rangle$ **do**
6 $\gamma_j \leftarrow$approx. alignment for $\sigma_j$ and $T_{i_j}$; // recursion
7 $\gamma \leftarrow \omega(\sigma, T, \langle\gamma_1, \ldots, \gamma_k\rangle)$; // composing
8 **return** $\gamma$;




(a) Trace splitting and assignment (b) Alignment composition

Fig. 3: Overview of the two main actions of the approximation approach

## 5 Alignment Approximation Approach

Here, we describe our proposed approach, which is based on the formal framework introduced. First, we present an overview. Subsequently, we present specific strategies for splitting/assigning and composing for each process tree operator.

### 5.1 Overview

For splitting a trace and assigning sub-traces to subtrees many options exist. Moreover, it is inefficient to try out all possible options. Hence, we use a *heuristic* that guides the splitting/assigning. For each subtree, we calculate four characteristics: the activity labels $A$, if the empty trace is in the subtree's language, possible start-activities $SA$ and end-activities $EA$ of traces in the subtree's language. Thus, each subtree is a *gray-box* since only limited information is available.

Consider the trace to be aligned $\sigma{=}\langle d, c, a, b, c, d, a, e\rangle$ and the two subtrees of $T_0$ with corresponding characteristics depicted in Figure 3a. Since $T_0$'s root node is a sequence operator, we need to split $\sigma$ once to obtain two subtraces according to the semantics. Thus, we have 9 potential splittings positions: $\langle|_1\ d\ |_2\ c\ |_3\ a\ |_4\ b\ |_5\ c\ |_6\ d\ |_7\ a\ |_8\ e\ |_9\rangle$. If we split at position 1, we assign $\sigma_1{=}\langle\rangle$ to the first subtree $T_1$ and the remaining trace $\sigma_2{=}\sigma$ to $T_2$. Certainly, this is not a good decision since we know that $\langle\rangle{\notin}\mathcal{L}(T_1)$, the first activity of $\sigma_2$ is not a start activity of $T_2$ and the activities $b, c, d$ occurring in $\sigma_2$ are not in $T_2$.

Assume we split at position 7 (Figure 3a). Then we assign $\sigma_1{=}\langle d, c, a, b, c, d\rangle$ to $T_1$. All activities in $\sigma_1$ are contained in $T_1$, $\sigma_1$ starts with $d{\in}SA(T_1)$ and ends with $d{\in}EA(T_1)$. Further, we obtain $\sigma_2{=}\langle a, e\rangle$ whose activities can be replayed in $T_2$, and start- and end-activities match, too. Hence, according to the gray-box-view, splitting at position 7 is a good choice. Next, assume we receive two alignments $\gamma_1$ for $T_1, \sigma_1$ and $\gamma_2$ for $T_2, \sigma_2$ (Figure 3b). Since $T_1$ is executed before $T_2$, we concatenate the sub-alignments $\gamma{=}\gamma_1{\cdot}\gamma_2$ and obtain an alignment for $T_0$.

### 5.2 Calculation of Process Tree Characteristics

In this section, we formally define the computation of the four tree characteristics for a given process tree $T{=}(V, E, \lambda, r)$. We define the activity set $A$ as a function, i.e., $A{:}\mathcal{T}{\to}\mathcal{P}(\mathcal{A})$, with $A(T){=}\{\lambda(n) \mid n{\in}T^L, \lambda(n){\neq}\tau\}$. We recursively define the possible start- and end-activities as a function, i.e., $SA{:}\mathcal{T}{\to}\mathcal{P}(\mathcal{A})$ and $EA{:}\mathcal{T}{\to}\mathcal{P}(\mathcal{A})$. If $T$ is not a leaf node, we refer to its two subtrees as $T_1$ and $T_2$.

$$SA(T){=}\begin{cases}\{\lambda(r)\} & \text{if } \lambda(r){\in}\mathcal{A}\\ \emptyset & \text{if } \lambda(r){=}\tau\\ SA(T_1) & \text{if } \lambda(r){=}{\to}\wedge\langle\rangle{\notin}\mathcal{L}(T_1)\\ SA(T_1){\cup}SA(T_2) & \text{if } \lambda(r){=}{\to}\wedge\langle\rangle{\in}\mathcal{L}(T_1)\\ SA(T_1){\cup}SA(T_2) & \text{if } \lambda(r){\in}\{\wedge,\times\}\\ SA(T_1) & \text{if } \lambda(r){=}{\circlearrowleft}\wedge\langle\rangle{\notin}\mathcal{L}(T_1)\\ SA(T_1){\cup}SA(T_2) & \text{if } \lambda(r){=}{\circlearrowleft}\wedge\langle\rangle{\in}\mathcal{L}(T_1)\end{cases}\quad EA(T){=}\begin{cases}\{\lambda(n)\} & \text{if } \lambda(r){\in}\mathcal{A}\\ \emptyset & \text{if } \lambda(r){=}\tau\\ EA(T_2) & \text{if } \lambda(r){=}{\to}\wedge\langle\rangle{\notin}\mathcal{L}(T_2)\\ EA(T_1){\cup}EA(T_2) & \text{if } \lambda(r){=}{\to}\wedge\langle\rangle{\in}\mathcal{L}(T_2)\\ EA(T_1){\cup}EA(T_2) & \text{if } \lambda(r){\in}\{\wedge,\times\}\\ EA(T_1) & \text{if } \lambda(r){=}{\circlearrowleft}\wedge\langle\rangle{\notin}\mathcal{L}(T_1)\\ EA(T_1){\cup}EA(T_2) & \text{if } \lambda(r){=}{\circlearrowleft}\wedge\langle\rangle{\in}\mathcal{L}(T_1)\end{cases}$$

The calculation whether the empty trace is accepted can also be done recursively.

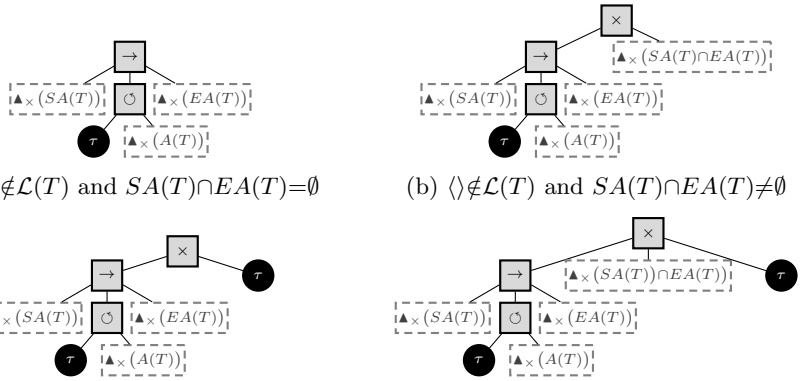
- $\lambda(r){=}\tau \Rightarrow \langle\rangle{\in}\mathcal{L}(T)$ and $\lambda(r){\in}\mathcal{A} \Rightarrow \langle\rangle{\notin}\mathcal{L}(T)$
- $\lambda(r){\in}\{\to,\wedge\} \Rightarrow \langle\rangle{\in}\mathcal{L}(T_1) \wedge \langle\rangle{\in}\mathcal{L}(T_2) \Leftrightarrow \langle\rangle{\in}\mathcal{L}(T)$
- $\lambda(r){\in}\times \Rightarrow \langle\rangle{\in}\mathcal{L}(T_1) \vee \langle\rangle{\in}\mathcal{L}(T_2) \Leftrightarrow \langle\rangle{\in}\mathcal{L}(T)$
- $\lambda(r){=}\circlearrowleft \Rightarrow \langle\rangle{\in}\mathcal{L}(T_1) \Leftrightarrow \langle\rangle{\in}\mathcal{L}(T)$

### 5.3 Interpretation of Process Tree Characteristics

The decision where to split a trace and the assignment of sub-traces to subtrees is based on the four characteristics per subtree and the process tree operator. Thus, each subtree is a gray-box for the approximation approach since only limited information is available. Subsequently, we explain how we interpret the subtree's characteristics and how we utilize them in the splitting/assigning decision.

Consider Figure 4 showing how the approximation approach assumes a given subtree $T$ behaves based on its four characteristics, i.e., $A(T), SA(T), EA(T)$, $\langle\rangle{\in}\mathcal{L}(T)$. The most liberal *interpretation* $\mathcal{I}(T)$ of a subtree $T$ can be considered as a heuristic that guides the splitting/assigning. The interpretation $\mathcal{I}(T)$ depends on two conditions, i.e., if $\langle\rangle{\in}\mathcal{L}(T)$ and whether there is an activity that is both, a start- and end-activity, i.e., $SA(T){\cap}EA(T){\neq}\emptyset$. Note that $\mathcal{L}(T){\subseteq}\mathcal{L}(\mathcal{I}(T))$ holds. Thus, the interpretation is an approximated view on the actual subtree.

In the next sections, we present for each tree operator a splitting/assigning and composing strategy based on the presented subtree interpretation. All strategies return a splitting per recursive call that minimizes the overall edit distance between the sub-traces and the closest trace in the language of the interpretation




(a) $\langle\rangle \notin \mathcal{L}(T)$ and $SA(T) \cap EA(T) = \emptyset$ (b) $\langle\rangle \notin \mathcal{L}(T)$ and $SA(T) \cap EA(T) \neq \emptyset$

(c) $\langle\rangle \in \mathcal{L}(T)$ and $SA(T) \cap EA(T) = \emptyset$ (d) $\langle\rangle \in \mathcal{L}(T)$ and $SA(T) \cap EA(T) \neq \emptyset$

Fig. 4: Most liberal interpretation $\mathcal{I}(T)$ of the four characteristics of a process tree $T \in \mathcal{T}$. For a set $X = \{x_1, \ldots, x_n\}$, $\blacktriangle_\times(X)$ represents the tree $\times(x_1, \ldots, x_n)$

of the assigned subtrees. For $\sigma_1, \sigma_2 \in \mathcal{A}^*$, let $\updownarrow(\sigma_1, \sigma_2) \in \mathbb{N} \cup \{0\}$ be the Levenshtein distance [16]. For given $\sigma \in \mathcal{A}^*$ and $T \in \mathcal{T}$, we calculate a valid splitting $\psi(\sigma, T) = \big\langle (\sigma_1, T_{i_1}), \ldots, (\sigma_j, T_{i_k}) \big\rangle$ w.r.t. Eq. (1) s.t. the sum depicted below is minimal.

$$\sum_{j \in \{1,\ldots,k\}} \big( \min_{\sigma' \in \mathcal{I}(T_{i_j})} \updownarrow(\sigma_j, \sigma') \big) \tag{3}$$

In the upcoming sections, we assume a given trace $\sigma = \langle a_1, \ldots, a_n \rangle$ and a process tree $T = (V, E, \lambda, r)$ with subtrees referred to as $T_1$ and $T_2$.

### 5.4 Approximating on Choice Operator

The choice operator is the most simple one since we just need to assign $\sigma$ to one of the subtrees according to the semantics, i.e., assigning $\sigma$ either to $T_1$ or $T_2$. We compute the edit distance of $\sigma$ to the closest trace in $\mathcal{I}(T_1)$ and in $\mathcal{I}(T_2)$ and assign $\sigma$ to the subtree with smallest edit distance according to Eq. (3).

Composing an alignment for the choice operator is trivial. Assume we eventually get an alignment $\gamma$ for the chosen subtree, we just return $\gamma$ for $T$.

### 5.5 Approximating on Sequence Operator

When splitting on a sequence operator, we must assign a sub-trace to each subtree according to the semantics. Hence, we calculate two sub-traces: $\langle (\sigma_1, T_1), (\sigma_2, T_2) \rangle$ s.t. $\sigma_1 \cdot \sigma_2 = \sigma$ according to Eq. (3). The optimal splitting/assigning can be defined as an optimization problem, i.e., Integer Linear Programming (ILP).

In general, for a trace with length $n$, $n+1$ possible splitting-positions exist: $\langle |_1\ a_1\ |_2\ a_2\ |_3\ \ldots\ |_n\ a_n\ |_{n+1} \rangle$. Assume we split at position 1, this results in $\big\langle (\langle\rangle, T_1), (\sigma, T_2) \big\rangle$, i.e., we assign $\langle\rangle$ to $T_1$ and the original trace $\sigma$ to $T_2$.

Composing the alignment from sub-alignments is straightforward. In general, we eventually obtain two alignments, i.e, $\langle \gamma_1, \gamma_2 \rangle$, for $T_1$ and $T_2$. We compose the alignment $\gamma$ for $T$ by concatenating the sub-alignments, i.e., $\gamma{=}\gamma_1{\cdot}\gamma_2$.

### 5.6 Approximating on Parallel Operator

According to the semantics, we must define a sub-trace for each subtree, i.e., $\langle (T_1, \sigma_1), (T_2, \sigma_2) \rangle$. In contrast to the sequence operator, $\sigma_1{\cdot}\sigma_2{=}\sigma$ does *not* generally hold. The splitting/assignment w.r.t. Eq. (3) can be defined as an ILP. In general, each activity can be assigned to one of the subtrees independently.

For example, assume $\sigma{=}\langle c, a, d, c, b \rangle$ and $T \mathrel{\widehat{=}} \wedge\big(\rightarrow(a, b), \circlearrowleft(c, d)\big)$ with subtree $T_1 \mathrel{\widehat{=}} \rightarrow(a, b)$ and $T_2 \mathrel{\widehat{=}} \circlearrowleft(c, d)$. Below we assign the activities to subtrees.

$$\begin{array}{ccccccc} \langle & c, & a, & d, & c, & b & \rangle \\ & T_2 & T_1 & T_2 & T_2 & T_1 & \end{array}$$

Based on the assignment, we create two sub-traces: $\sigma_1{=}\langle a, b \rangle$ and $\sigma_2{=}\langle c, d, c \rangle$. Assume that $\gamma_1 \mathrel{\widehat{=}} \langle (a, a), (b, b) \rangle$ and $\gamma_2 \mathrel{\widehat{=}} \langle (c, c), (d, d), (c, c) \rangle$ are the two alignments eventually obtained. To compose an alignment for $T$, we have to consider the assignment. Since the first activity $c$ is assigned to $T_2$, we extract the corresponding alignment steps from $\gamma_1$ until we have explained $c$. The next activity in $\sigma$ is an $a$ assigned to $T_1$. We extract the alignment moves from $\gamma_1$ until we explained the $a$. We iteratively continue until all activities in $\sigma$ are covered. Finally, we obtain an alignment for $T$ and $\sigma$, i.e., $\gamma \mathrel{\widehat{=}} \big\langle (c, c), (a, a), (d, d), (c, c), (b, b) \big\rangle$.

### 5.7 Approximating on Loop Operator

We calculate $m{\in}\{1, 3, 5, \dots\}$ sub-traces that are assigned alternately to the two subtrees: $\langle (\sigma_1, T_1), (\sigma_2, T_2), (\sigma_3, T_1), \dots, (\sigma_{m-1}, T_2), (\sigma_m, T_1) \rangle$ s.t. $\sigma{=}\sigma_1{\cdot}\dots{\cdot}\sigma_m$. Thereby, $\sigma_1$ and $\sigma_m$ are always assigned to $T_1$. Next, we visualize all possible splitting positions for the given trace: $\langle |_1\, a_1\, |_2\, |_3\, a_2\, |_4 \dots |_{2n-1}\, a_n\, |_{2n} \rangle$. If we split at each position, we obtain $\big\langle \big(\langle\rangle, T_1\big), \big(\langle a_1 \rangle, T_2\big), \big(\langle\rangle, T_1\big), \dots, \big(\langle a_n \rangle, T_2\big), \big(\langle\rangle, T_1\big) \big\rangle$. The optimal splitting/assignment w.r.t Eq. (3) can be defined as an ILP.

Composing an alignment is similar to the sequence operator. In general, we obtain $m$ sub-alignments $\langle \gamma_1, \dots, \gamma_m \rangle$, which we concatenate, i.e., $\gamma{=}\gamma_1{\cdot}\dots{\cdot}\gamma_m$.

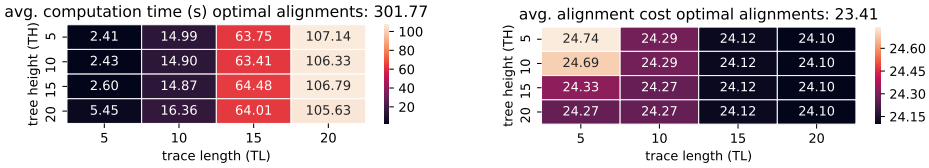
## 6 Evaluation

This section presents an experimental evaluation of the proposed approach.

We implemented the proposed approach in PM4Py[3], an open-source process mining library. We conducted experiments on real event logs [17,18]. For each log, we discovered a process tree with the Inductive Miner infrequent algorithm [10].

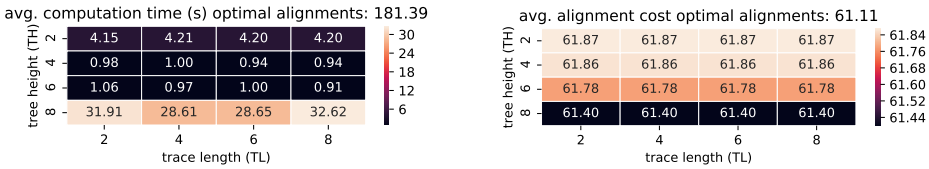
In Figures 5 and 6, we present the results. We observe that our approach is on average always faster than the optimal alignment algorithm for all tested parameter settings. Moreover, we observe that our approach never underestimates

[3] https://pm4py.fit.fraunhofer.de/




(a) Avg. computation time (s) (b) Avg. alignment costs

Fig. 5: Results for [17], sample: 100 variants, tree height 24, avg. trace length 28




(a) Avg. computation time (s) (b) Avg. alignment costs

Fig. 6: Results for [18], sample: 100 variants, tree height 10, avg. trace length 65

the optimal alignment costs, as our approach returns a valid alignment. W.r.t. optimization problems for optimal splittings/assignments, consider parameter setting TH:5 and TL:5 in Figure 5. This parameter setting results in the highest splitting along the tree hierarchy and the computation time is the lowest compared to the other settings. Thus, we conclude that solving optimization problems for finding splittings/assignments is appropriate. In general, we observe a good balance between accuracy and computation time. We additionally conducted experiments with a decomposition approach [15] (available in ProM[4]) and compared the calculation time with the standard alignment implementation (LP-based) [12] in ProM. Consider Table 2. We observe that the decomposition approach does not yield a speed-up for [17] but for [18] we observe that the decomposition approach is about 5 times faster. In comparison to Figure 6a, however, our approach yields a much higher speed-up.

## 7 Conclusion

We introduced a novel approach to approximate alignments for process trees. First, we recursively split a trace into sub-traces along the tree hierarchy based

[4] http://www.promtools.org/

Table 2: Results for decomposition based alignments

| Approach | [17] (sample: 100 variants) | [18] (sample: 100 variants) |
|---|---|---|
| decomposition [15] | 25.22 s | 20.96 s |
| standard [12] | 1.51 s | 103.22 s |

on a gray-box view on the respective subtrees. After splitting, we compute optimal sub-alignments. Finally, we recursively compose a valid alignment from sub-alignments. Our experiments show that the approach provides a good balance between accuracy and calculation time. Apart from the specific approach proposed, the contribution of this paper is the formal framework describing how alignments can be approximated for process trees. Thus, many other strategies besides the one presented are conceivable.

## References


1. W. M. P. van der Aalst, *Process Mining - Data Science in Action*. Springer, 2016.
2. W. M. P. van der Aalst, A. Adriansyah, and B. F. van Dongen, "Replaying history on process models for conformance checking and performance analysis," *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 2, no. 2, 2012.
3. J. Carmona, B. F. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking - Relating Processes and Models*. Springer, 2018.
4. W. L. J. Lee, H. M. W. Verbeek, J. Munoz-Gama, W. M. P. van der Aalst, and M. Sepúlveda, "Recomposing conformance: Closing the circle on decomposed alignment-based conformance checking in process mining," *Inf. Sci.*, vol. 466, 2018.
5. M. F. Sani, S. J. van Zelst, and W. M. P. van der Aalst, "Conformance checking approximation using subset selection and edit distance," in *CAiSE 2020*, ser. LNCS, vol. 12127. Springer, 2020.
6. F. Taymouri and J. Carmona, "An evolutionary technique to approximate multiple optimal alignments," in *BPM 2018*, ser. LNCS, vol. 11080. Springer, 2018.
7. ——, "Model and event log reductions to boost the computation of alignments," in *SIMPDA 2016*, vol. 1757. CEUR-WS.org, 2016.
8. M. Bauer, H. van der Aa, and M. Weidlich, "Estimating process conformance by trace sampling and result approximation," in *BPM 2019*, ser. LNCS, vol. 11675. Springer, 2019.
9. S. J. J. Leemans, "Robust process mining with guarantees," Ph.D. dissertation, Department of Mathematics and Computer Science, 2017.
10. S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour," in *BPM Workshops 2013*, ser. LNBIP, vol. 171. Springer, 2013.
11. D. Schuster, S. J. van Zelst, and W. M. P. van der Aalst, "Incremental discovery of hierarchical process models," in *RCIS 2020*, ser. LNBIP, vol. 385. Springer, 2020.
12. A. Adriansyah, "Aligning Observed and Modeled Behavior," Ph.D. dissertation, Eindhoven University of Technology, 2014.
13. B. F. van Dongen, "Efficiently computing alignments - using the extended marking equation," in *BPM 2018*, ser. LNCS, vol. 11080. Springer, 2018.
14. B. F. van Dongen, J. Carmona, T. Chatain, and F. Taymouri, "Aligning modeled and observed behavior: A compromise between computation complexity and quality," in *CAiSE 2017*, ser. LNCS, vol. 10253. Springer, 2017.
15. W. M. P. van der Aalst, "Decomposing Petri Nets for Process Mining: A Generic Approach," *Distributed and Parallel Databases*, no. 4, 2013.
16. V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8, 1966.
17. B. F. van Dongen, "BPI Challenge 2019. Dataset," 2019.
18. B. F. van Dongen and F. Borchert, "BPI Challenge 2018. Dataset," 2018.