Temporal Network Representation of Event Logs for Improved Performance Modelling in Business Processes

Arik Senderovich[1], Matthias Weidlich[2], and Avigdor Gal[1]

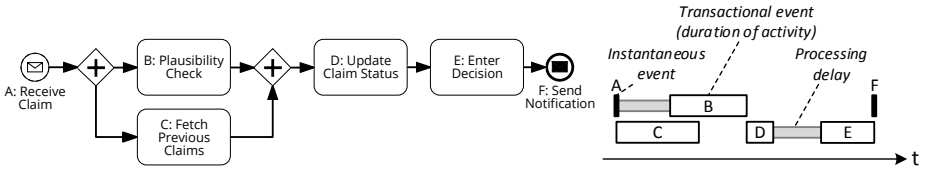
[1] Technion – Israel Institute of Technology
{sariks@,avigal@ie.}technion.ac.il
[2] Humboldt-Universität zu Berlin
matthias.weidlich@hu-berlin.de

Abstract. Analysing performance of business processes is an important vehicle to improve their operation. Specifically, an accurate assessment of sojourn times and remaining times enables bottleneck analysis and resource planning. Recently, methods to create respective performance models from event logs have been proposed. These works are severely limited, though: They either consider control-flow and performance information separately, or rely on an ad-hoc selection of temporal relations between events. In this paper, we introduce the Temporal Network Representation (TNR) of a log, based on Allen's interval algebra, as a complete temporal representation of a log, which enables simultaneous discovery of control-flow and performance information. We demonstrate the usefulness of the TNR for detecting (unrecorded) delays and for probabilistic mining of variants when modelling the performance of a process. In order to compare different models from the performance perspective, we develop a framework for measuring performance fitness. Under this framework, we provide guarantees that TNR-based process discovery dominates existing techniques in measuring performance characteristics of a process. To illustrate the practical value of the TNR, we evaluate the approach against three real-life datasets. Our experiments show that the TNR yields an improvement in performance fitness over state-of-the-art algorithms.

1 Introduction

Modern process-aware information systems (PAIS) support the design, enactment, and analysis of business processes in various domains [1]. Based on a formalisation of the supported business process in terms of a process model, they control how the execution of a set of activities is coordinated to reach a certain outcome for an instance of the process. The operation of business processes can be improved by modelling their performance. Specifically, an accurate assessment of key performance measures, such as sojourn times and remaining times, enables bottleneck analysis and optimised resource planning [2].

Recently, to enable performance analysis of business processes, methods that construct process models from event logs that contain transactional data have been proposed [3,4]. Yet, these methods consider control-flow and performance information separately [5,6,7]. They first create a process model that captures causal dependencies between activities (commonly referred to as *discovery*), which is later annotated with performance details (referred to as *enhancement*). Hence, any bias introduced in control-flow discovery carries over to the performance analysis.



(a) (b)

Fig. 1: Claim handling process (a); common actual execution of activities (b).

To illustrate the problem implied by this 2-step approach, we consider a claim handling example, where discovery may yield the BPMN model in Figure 1a. Annotating the model with activity durations, however, does not capture delays between actual activity executions. This potentially yields inaccuracies when conducting performance analysis. That is, once a claim is received (A), a system may automatically fetch previous claims (C). Yet, the plausibility check (B), supposed to be done in parallel, involves a knowledge worker, who is not available immediately. Hence, the start of the activity is delayed (see Figure 1b). Similarly, after the automatic status update (D), another staff member needs to enter the decision (E), which also introduces a delay.

In transactional event logs that record the start *and* end of activity execution, delays are directly visible for individual process instances, as shown in Figure 1b. However, such an instance may represent noise in the event log, which raises the question of how to consider delays on the model-level. When constructing a process model for performance analysis, the observed delays of individual instances need to be generalised.

The challenge of incorporating delays in the construction of performance models has been recognised in the literature. Specifically, Tsinghua-α [8] and variants of the Inductive Miner [4] incorporate performance details by considering temporal relations between the start and end of activity executions. Yet, these approaches are limited in two ways: (i) They take an ad-hoc decision on the type of temporal relation to consider in model discovery (e.g., to distinguish interleaved and concurrent execution of activities [4]); and (ii) they require a model to represent a single temporal relation per pair of activities (e.g., two activities are *always* interleaved or concurrent [4]).

In this paper, to overcome the above limitations, we introduce the Temporal Network Representation (TNR) of an event log as a formalism that is grounded in Allen's interval algebra [9]. The TNR is a compact representation of all (pairwise) temporal relations between activity executions as observed in the event log. As such, it generalises different notions of dependency graphs commonly used in process model discovery and enables us to incorporate performance information in terms of processing delays in model discovery. Our contributions and the structure of the paper are summarised as follows:

- *The Temporal Network Representation (TNR) of an event log:* Following an introduction of preliminaries (Section 2), in Section 3, we present the TNR of transactional event logs. The TNR generalises common representations of event logs.
- *Inductive Mining with the TNR:* In Section 4, we propose an algorithm to integrate *delay unfolding* in inductive mining, exploiting the TNR to include processing delays explicitly. We then show how the TNR enables *probabilistic variant mining*, which handles noisy event logs, but preserves performance details in the discovered model.

– *Measuring Performance Fitness:* Section 5 introduces a framework for measuring performance fitness between an event log and a model. We also show that under this framework, TNR-based inductive mining is guaranteed to discover unbiased models.

To demonstrate the practicality of the TNR, we evaluated our approach with three real-world healthcare datasets. As detailed in Section 6, TNR based reasoning yields up-to 40% improvement in performance fitness with respect to existing approaches. Finally, we discuss related work in Section 7, before concluding in Section 8.

2 Preliminaries

This section reviews preliminaries for our work in terms of event logs, process trees as a formalism for process modelling, and Allen's algebra to reason on temporal intervals.

Event Logs. We adopt a notion of a transactional event log that relates events to their activity labels (activities, for short), start times, and completion times. Let $\mathcal{E}$ be the universe of events produced by an information system and let $\mathcal{A}$ be the set of supported activities. Then, by $e.a \in \mathcal{A}$, $e.s \in \mathbb{R}_0^+$, and $e.c \in \mathbb{R}_0^+$, we denote the activity that corresponds to the event, its start time, and completion time, respectively.

A case $\xi \in 2^{\mathcal{E}} \setminus \emptyset$ is a finite set of events, assuming that no event may occur in more than one case and that a case comprises at least one event. An event log $L \subseteq 2^{\mathcal{E}}$ is a set of cases. Table 1 presents an example event log for the claim handling process in Figure 1. Note that some of the events are instantaneous (i.e., have a duration of 0). We denote by $A_\xi \subseteq \mathcal{A} \times \mathbb{N}^+$ the multi-set of activities that appear in ξ, namely $A_\xi = \{(e.a, k) | e \in \xi\}$ with k being the frequency of $e.a$ in ξ.

Table 1: Example event log for the claim handling process.

Case	Activity	Start	Complete
1	A: Receive Claim	9:05	9:05
1	C: Fetch Previous Claim	9:05	9:10
1	B: Plausibility Check	9:08	9:20
1	D: Update Claim Status	9:20	9:22
1	E: Enter Decision	9:40	12:05
2	A: Receive Claim	10:23	10:23
2	C: Fetch Previous Claim	10:23	10:34
3	A: Receive Claim	10:25	10:25
3	B: Plausibility Check	10:25	10:28
3	C: Fetch Previous Claim	10:25	10:30
2	B: Plausibility Check	10:30	10:55
3	D: Update Claim Status	10:30	10:30
2	D: Update Claim Status	10:55	10:55
2	E: Enter Decision	11:10	11:28
2	F: Send Notification	11:28	11:28
1	F: Send Notification	12:05	12:05

Process Trees. To represent the process executed by an information system, we adopt the notion of a process tree [10] that is enriched with time information. Traditionally, a process tree encodes the control-flow of a process in terms of its possible traces, i.e., sequences of activity executions. We recall the intuition behind process trees and refer the reader to [10] for a complete formalisation of their syntax and semantics.

An untimed process tree is a rooted tree, in which the leaf nodes are activities in $\mathcal{A}_\tau = \mathcal{A} \cup \{\tau_1, \ldots, \tau_n\}$ with τ_i, $1 \leq i \leq n$, denoting *silent activities* that cannot be observed during the execution of the process (but which may have different durations, so that they need to be distinguished from one another). All non-leaf nodes are control-flow operators, denoted by $\mathcal{O}$. Common control-flow operators are sequence ($\rightarrow$), exclusive choice ($\times$), concurrency ($\wedge$), interleaving ($||$) and structured loops ($\circlearrowleft$). Figure 2 shows the process tree for the BPMN model in Figure 1a. Semantics of a process tree is defined

by recursively constructing a set of traces: For a leaf node labelled with $a \in \mathcal{A}$, the set of traces contains a single trace, $\{\langle a \rangle\}$, whereas it contains the empty trace $\{\langle\rangle\}$ for a silent activity. Semantics of a non-leaf node is formalised by a language function that joins the traces of the subtrees of the node. For instance, the set of traces of the exclusive choice operator is given by the union of the trace sets of its children.

We extend process tress by adding durations to leaf nodes. Each activity $a \in \mathcal{A}_\tau$ is assigned a duration of the (potentially silent) execution of a, which comes from a cumulative distribution function (CDF) D_a. This induces a timed semantics of the process tree in terms of sequences of events. From a trace of the untimed process tree, a set of events is constructed by drawing a duration from D_a for each activity $a \in \mathcal{A}_\tau$ of the trace and constructing the start time and completion time as follows: the start time is the completion time of the event for the previous activity in the trace (or 0, if the activity is the first one) and the completion time is the start time plus the duration. This way we model instantaneous activities (with a constant duration of 0) and processing delays (silent activities of a certain duration).

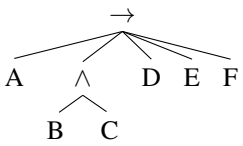


Fig. 2: Process tree of the claim handling process.

As another extension to the common model of process trees, we consider its enrichment with branching probabilities. For our purposes, it suffices to assign a probability distribution P_o to each n-ary exclusive choice operator $o \in \mathcal{O}$, so that P_o models the occurrence probabilities of the n children of the operator.

Allen's Interval Algebra. To reason about temporal relations of events, Allen presented an interval algebra [9] that defines 13 relations between two intervals. Each of them formalises a different partial order of the start and completion times of interval events, see Figure 3. Instantiating these relations for the above notion of events, for instance, a pair of events $x, y \in \mathcal{E}$ is in the *overlaps* relation, if and only if, $x.s < y.s < x.c < y.c$.

The interval relations are mutually exclusive and partition the Cartesian product of events. As shown in Figure 3, each relation between two events, except *equals*, has a counterpart that holds for the reversed pair of events. To avoid this kind of redundancy, in the remainder, we consider the following 7 out of the 13 relations: *precedes*, *meets*, *overlaps*, *is finished by*, *contains*, *starts*, and *equals*. The set of these relations is denoted by $\mathcal{R}$.

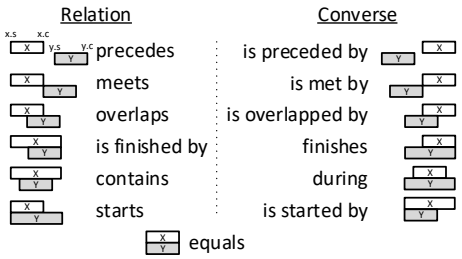


Fig. 3: Allen's interval relations, see [11].

3 The Temporal Network Representation of an Event Log

This section introduces our notion of the temporal network representation (TNR) of an event log. It is based on Allen's interval relations to capture the temporal information in the log. We further discuss how the TNR can be collapsed to obtain commonly used models of event logs, namely the directly-follows graph and the concurrency graph.

3.1 Definition

The TNR is grounded in the notion of *temporal evidence*, which is needed when lifting the interval relations from events to activities. Since in an event log, there may be many pairs of events related to the same pair of activities, the temporal evidence captures the frequency of a particular interval relation being observed among the respective events.

Definition 1 (Temporal Evidence) *Temporal evidence is a tuple* $(R, f) \in \mathcal{R} \times \mathbb{N}^+$ *with* R *being an interval relation and* f *being its frequency.*

Given an event log, its TNR is a directed graph where nodes are activities and edge labels assign temporal evidence to the respective pairs of activities.

Definition 2 (Temporal Network Representation (TNR)) *Let* L *be an event log. Its Temporal Network Representation is a directed, edge-labelled graph* $G = (V, E, \lambda)$*, such that*

- $V = \bigcup_{\xi \in L} \bigcup_{e \in \xi} e.a$*, the nodes are all activities of events of cases in the log;*
- $E = \{(v_1, v_2) \in V \times V \mid \exists\, \xi \in L : \exists\, e_1, e_2 \in \xi : e_1.a = v_1 \ \wedge\ e_2.a = v_2\}$*, edges are defined between all pairs of activities that occur jointly in cases;*
- $\lambda : E \to 2^{\mathcal{R} \times \mathbb{N}^+}$*, with* $\lambda(d) \mapsto (R, f)$*,* $R \in \mathcal{R}$*, and* $f = |\{(e_1, e_2) \in \mathcal{E} \times \mathcal{E} \mid \exists\, \xi \in L : e_1, e_2 \in \xi : (e_1.a, e_2.a) = d \ \wedge\ (e_1, e_2) \in R\}|$*, the edge labelling maps temporal evidence as observed in the log to edges.*

The TNR of an event log can be constructed incrementally upon the addition of a new case to an event log. Considering the new information from the log may introduce additional vertices, additional edges, or increase the frequency of some temporal evidence.

For the log in Table 1, the TNR is shown in Table 2. Here, the time of an instantaneous event is considered as a completion time when deriving the interval relations.

Table 2: Matrix representation of the TNR of the event log in Table 1.

	A	**B**	**C**	**D**	**E**	**F**
A		{(precedes,2), (meets,1)}	{(meets,3)}	{(precedes,3)}	{(precedes,2)}	{(precedes,2)}
B			{(starts,1)}	{(precedes,1), (meets,2)}	{(precedes,2)}	{(precedes,2)}
C		{(overlaps,2)}		{(precedes,2), (meets,1)}	{(precedes,2)}	{(precedes,2)}
D					{(precedes,2)}	{(precedes,2)}
E						{(is finished by,2)}
F						

3.2 Projections on the TNR

Since the TNR captures all pairwise temporal relations between activities, it generalises existing models of event logs. These models are typically defined as dependency graphs, in which the edges encode a particular temporal relation. Prominent examples used in discovery algorithms such as the Inductive Miner [10], α-Miner [12], or the Heuristic Miner [13] include the direct-follows graph and the concurrency graph. In the direct-follows graph, assuming that it is grounded in completion times of transactional events, a directed edge between activities x and y encodes that there exists a case $\xi \in L$ with two events $e_1, e_2 \in \xi$, such that $e_1.a = x$, $e_2.a = y$, $e_1.c < e_2.c$, and there is no event $e_3 \in \xi$ with $e_1.c < e_3.c < e_2.c$. The concurrency graph, in turn, contains an

undirected edge for each pair of activities x and y, for which there exists a case $\xi \in L$ and events $e_1, e_2 \in \xi$ with $e_1.a = x$, $e_2.a = y$, and $e_1.s \leq e_2.s \leq e_1.c \leq e_2.c$.

These graphs may be derived from the TNR by projections. A TNR projection is a function that maps a TNR $G = (V, E, \lambda)$ to another TNR $G' = (V', E', \lambda')$, such that $V = V'$, $E \subseteq E'$, whereas the labelling λ' of G' is not constrained.

We first illustrate the derivation of the directly-follows graph [10], assuming that it is grounded in the completion times of activities. This requires two projections:

(1) For each edge, the temporal evidence for the relations *precedes*, *meets*, *overlaps*, and *contains* is aggregated and considered as part of the *precedes* relation. That is, the frequencies of all these relations are summed up and yield the new frequency of the *precedes* relation. We then remove all edges having a frequency of 0 for *precedes*.
(2) On the TNR that contains only edges with temporal evidence related to *precedes*, we conduct a transitive reduction. We are left with the directly-follows graph.

In the same manner, we can also derive the concurrency graph as used by the life-cycle variant of the Inductive Miner [4]. To this end, the *overlaps*, *is finished by*, *contains*, *starts*, and *equals* relations are aggregated, yielding a new *overlaps* relation. Then, all edges not having temporal evidence related to *overlaps* are removed.

4 Inductive Mining with the TNR

In this section, we show how the TNR can be used to enhance discovery of process models via inductive mining. We first introduce how the TNR is used to make processing delays explicit, before elaborating on the actual construction of a process tree (Section 4.1). Then, we propose probabilistic variant mining based on the TNR to handle noisy event logs, while preserving performance details (Section 4.2).

4.1 Delay-Aware Inductive Mining

Delay Unfoldings on the TNR. The TNR indicates processing delays by means of the *precedes* interval relation. If the TNR contains an edge between activities x and y with temporal evidence for *precedes*, it means that there is a case in the log in which the start and completion times of two transactional events that represent the occurrence of x and y are ordered, but the occurrence of y does not start immediately after the occurrence of x completes—there is a processing delay between x and y.

To make such processing delays explicit, we define a transformation of the TNR, referred to as delay unfolding. In essence, it inserts a *delay activity* between any two activities for which there is an edge with temporal evidence for the *precedes* interval relation. This delay activity is then linked to the respective activities in terms of temporal evidence for the *meets* relation, with the intuition being that this activity represents the gap between the occurrences of the original activities.

However, an activity x will be in the *precedes* relation with any other activity that starts after x completes. Therefore, we insert a delay activity between two activities x and y solely if there does not exist an activity z, whose occurrence can be seen as the reason for the time gap between the completion of x and the start of y. The situation when a delay-driven gap does not exist between activities x and y would be characterised by one of the following cases:

– There is an activity z that starts after or with the completion of x, while y starts after or with the completion of z, both are manifested as relations $\mathcal{R}_{after} = \{precedes, meets\}$; or
– there is an activity z that starts before the completion of x (temporal evidence is given as $\mathcal{R}_{over} = \{overlaps, is\ finished\ by, contains, starts, equals\}$), while y starts after or with the start of z (all relations in $\mathcal{R}$).

Using the above sets of temporal relations, we formally define the transformation of delay unfolding as follows:

Definition 3 (Delay Unfolding) *Given a TNR $G = (V, E, \lambda)$, the delay unfolding yields a new TNR $G' = (V', E', \lambda')$, such that:*

– *$V' = V \cup V_\delta$, where V_δ contains a node $\delta_{(x,y)}$ for each edge $d = (x, y) \in E$ with temporal evidence $(\text{precedes}, f) \in \lambda(d)$, $f > 0$, if there do not exist edges $d_x = (x, z), d_y = (z, y) \in E$ with temporal evidences $(R_x, f_x) \in \lambda(d_x)$, $(R_y, f_y) \in \lambda(d_y)$, $f_x, f_y > 0$, and either $R_x, R_y \in \mathcal{R}_{after}$, or $R_x \in \mathcal{R}_{over}$ and $R_y \in \mathcal{R}$;*
– *$E' = E \cup E_\delta$, where $E_\delta = \{(x, \delta_{(x,y)}), (\delta_{(x,y)}, y) \mid \delta_{(x,y)} \in V_\delta\}$ connects the new nodes from V_δ with the source and target of the respective edges; and*
– *$\lambda'(d) = \{(R, f) \in \lambda(d) \mid R \neq precedes\}$ for original edges $d \in E_{V_\delta} = \{(x, y) \mid \delta_{(x,y)} \in V_\delta\}$ for which unfolding was applied; $\lambda'(d) = \lambda(d)$ for all other edges $d \in E \setminus E_{V_\delta}$; and $\lambda'(d) = \{(meets, f)\}$ for $(x, \delta_{(x,y)}), (\delta_{(x,y)}, y) \in E_\delta$, $d = (x, y) \in E$ and $(precedes, f) \in \lambda(d)$, for the new edges.*

Figure 4 illustrates the delay unfolding for a part of the example in Table 1. Here, a delay activity δ is introduced between A and B, representing the time gap indicated by the temporal evidence for the *precedes* relation. For all other edges with evidence for *precedes*, there are other possible reasons for the respective time gaps. Considering the edge between B and D as an example: while there is temporal evidence for a delayed start of D after B (the *precedes* relation), the temporal evidence of edges between B and C, and C and D, indicates that this delay may stem from the execution of C.

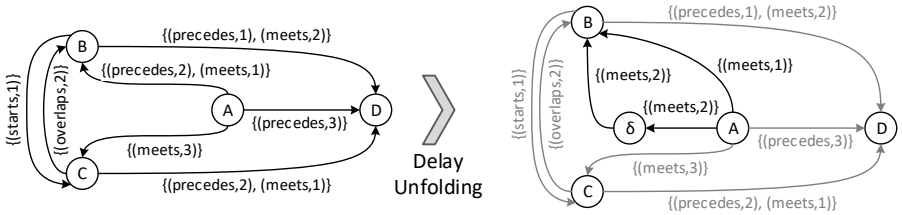


Fig. 4: Delay unfolding for a part of the TNR for the log in Table 1.

Model Construction. Once the delay unfolding on the TNR has made processing delays explicit, a timed process tree is derived. Here, we exploit the idea of inductive mining (IM), which is a constructive approach to process discovery [4,10,14]. In essence, inductive mining proceeds as follows: given a directly-follows graph, it recursively identifies cuts in the graph that separate its components, induce a partition of the log, and yield the control-flow operators of the process tree. For each identified component, this procedure is repeated until trivial components (single activities) are obtained.

We adopt this general approach for the TNR after delay unfolding and rely on the existing algorithms to detect cuts, while integrating the handling of processing delays:

(1) By means of TNR projections, a directly-follows graph and concurrency graph is derived from the TNR as outlined in Section 3.2. Then, a process tree is built using the IM for transactional event logs [4], which relies on the concurrency graph to distinguish interleaved from concurrent execution of activities. The resulting process tree contains the delay activities introduced as part of the delay unfolding.

(2) For all activities $a \in \mathcal{A}$ of the obtained process tree, we fit a cumulative distribution function CDF D_a to model the duration. For activities that are observed in the log, we fit the distribution based on all observed durations of events, $\{e.c - e.s \mid e \in \bigcup_{\xi \in L} \xi \wedge e.a = a\}$. For a delay activity $\delta_{(x,y)}$, the samples to which the distribution is fitted are given as $\{e_2.s - e_1.c \mid e_1, e_2 \in \xi \wedge \xi \in L \wedge e_1.a = x \wedge e_2.a = y\}$, i.e., the durations between the completion of the preceding activity and the start of the succeeding activity. Every exclusive choice operator is enriched with the corresponding occurrence probabilities.

(3) Any delay activity $\delta_{(x,y)}$ is labelled as a distinguished silent activity, $\delta_{(x,y)}.a = \tau_i$ for a unique $i \in \mathbb{N}$, to capture that it does not carry any application semantics.

For the above example, the structure of the timed process trees that results from this procedure is shown in Figure 5. The model features two silent activities τ_1 and τ_2 that represent common processing delays in handling claims, as outlined already for a single case in Figure 1b. The CDFs assigned to these silent activities model the durations of these delays.

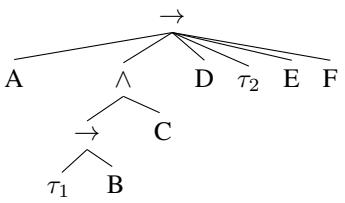


Fig. 5: Process tree with delay activities.

4.2 Probabilistic Variant Mining

In this part, we consider another angle to utilise the TNR in model discovery: it can guide the handling of noise in the event log. We observe that common approaches to process model discovery strive for understandable models. As a consequence, when filtering noise in the event log, they tend to represent solely the most commonly observed behavioural relations between activities in the log. From the viewpoint of performance modelling, this self-imposed restriction is not needed. We therefore argue that, in order to improve the representational bias, a model discovered for performance analysis may incorporate different behavioural relations explicitly, if there is enough evidence for them. These different relations may then be weighted following a probabilistic model.

Below, we introduce *probabilistic variant mining* (PVM), which uses the TNR to handle noise in inductive mining of process trees. PVM comprises two steps: (1) a *preprocessing* step, where a discrete-valued cumulative distribution function is computed for behavioural relations based on the TNR; (2) a *variant construction* step, where this distribution function is used to introduce choices between subtrees in the resulting model.

Preprocessing. Consider a TNR $G = (V, E, \lambda)$ that represents an event log L. The PVF relies on the frequencies defined by the temporal evidence to compute a discrete-valued cumulative distribution function (dv-CDF) over the interval relations, $F(d) : \mathcal{R} \rightarrow$

$[0,1], d \in E$. Let $\lambda(d) = \{(R_1, f_1), \ldots, (R_k, f_k)\}$ be the temporal evidence of edge $d \in E$. Without loss of generality, we order the elements (R_i, f_i), $1 \leq i \leq k$, such that $f_1 \leq f_2 \leq \ldots f_k$. The dv-CDF of edge e for the i-th relation is then defined as:

$$F(d)(R_i) = F(d)(R_{i-1}) + \frac{f_i}{\sum_{j=1}^{k} f_j}, \quad \text{with } F(d)(f_0) \text{ defined as } 0.$$

Variant Construction. Following the general approach of inductive mining of process trees (as recalled in Section 4.1, yet potentially without handling of processing delays), the control-flow operators of a process tree are identified by iteratively detecting cuts in a dependency graph. If such cuts cannot be detected immediately, edges of the graph may be considered as noise and filtered according to a user-defined noise threshold [14]. In any case, cuts are detected based on a deterministic projection of the dependency graph. In contrast, PVM defines a probabilistic means to identify cuts based on different TNR projections to obtain a direct-follows graph. In addition to the construction of a direct follows graph outlined in Section 3.2, we also consider a construction solely from relations that define an empty intersection of intervals (i.e., the set of relations $\mathcal{R}_{after}$ as defined above). If the resulting graphs give rise to different cuts, PVM constructs an exclusive choice operator in the process tree, which embeds all subtrees obtained from the different cuts. Hence, no information is lost in the process. The branching probabilities of this exclusive choice are assigned based on the dv-CDF function F as determined in the preprocessing step: each choice is assigned the aggregated probability of the respective pairwise relations between activities.

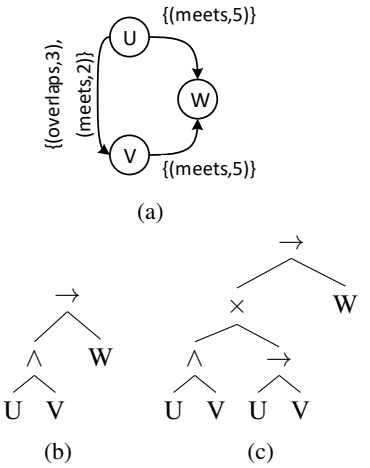


Fig. 6: Example TNR (a); trees obtained with traditional inductive mining (b) and with PVM (b).

For illustration, consider the TNR given in Figure 6a, which defines temporal evidence for U and V for both, *overlaps* and *precedes*. For this setting, the Inductive Miner for transactional event logs [4] would construct the model given in Figure 6b, meaning that the relations between B and C are interpreted as concurrent execution. With PVM, it will be noticed that the graph created from relations that define an empty intersection of intervals also yields a sequence cut. Thus, the result would be the model in Figure 6c; with the branching probabilities of the $\times$ operator set to $\frac{3}{5}$ for the concurrent ($\wedge$) case, and $\frac{2}{5}$ for the sequence case ($\rightarrow$). Clearly, this model is less understandable for users. Yet, focusing on performance modelling, it more accurately captures the behaviour encoded in the TNR and, thus, shall improve the model from the performance perspective.

5 Performance Fitness and Theoretical Guarantees

To evaluate algorithms for performance-driven process discovery, such as the one introduced earlier, this section presents a framework for measuring performance fitness.

First, we define performance measures, loss functions, and the notion of *performance replay*, which enable us to quantify the distance between an event log and a process tree (Section 5.1). We then prove that TNR-based inductive mining comes with guarantees on the performance fitness with respect to the discovered model (Section 5.2).

5.1 Framework for Measuring Performance Fitness

We consider performance fitness with respect to a performance measure (e.g., the sojourn time of a case) and a loss function (e.g., the bias). This pair is defined as follows.

Definition 4 (Performance Measure, Loss Function) *A performance measure* $\psi : 2^{\mathcal{E}} \setminus \emptyset \rightarrow \mathbb{R}_0^+$ *is a function that maps a set of events to a real-value. A loss function* $l : \mathbb{R}_0^+ \times \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$ *maps a pair of performance measures to a real-value.*

Having defined ψ and l, we aim at a distance measure between an event log and a model. To this end, we introduce the procedure of *performance replay* between a case $\xi = \{e_1, \ldots, e_n\} \in L$ and a process tree T. It is based on the assumption that T contains all activities referenced in events of cases $\xi \in L$. In other words, we do not consider replay for noise-filtered process trees. Performance replay involves the following steps:

(1) We first match the events of ξ to the process tree T. While there may be multiple leaf nodes to which an event could be matched, we note that matching needs to ensure consistency of the temporal relation between events and the semantics of the process tree operators. That is, this matching can be seen as the reverse operation of constructing a process tree from the TNR, see Section 4.1. As a result, we obtain a subtree T_ξ that contains a set of operators (without exclusive choice) and leaf nodes, so that each event in ξ is matched to one of the leaf nodes.

(2) The subtree T_ξ is then played-out into a replayed case $\xi_T = \{e_1^T, \ldots, e_n^T \mid e_i^T.a = e_i.a\}$. It contains the same number of events as ξ and every replayed event e_i^T refers to the same activity as the corresponding event e_i in ξ. Activities that appear in the subtree T_ξ, but are without matching events in ξ, are omitted from ξ_T via a projection of T_ξ on these activities.

(3) For the replayed case, we then construct durations. That is, for event e_i^T, the duration x_i^T is sampled from the CDF D_a of the respective activity $a = e_i^T.a$.

(4) Based on the durations, the start and completion times of the replayed case are set. To this end, we follow the partial order induced by the operators in T_ξ. The first events have a start time of $e_i^T.s = 0$ and a completion time of the sampled duration $e_i^T.c = e_i^T.s + x_i^T$. Subsequent events start at the maximal completion time of preceding events (in terms of the partial order induced by T_ξ), while their completion time is again determined based on their start time and sampled duration.

As an example, we consider the replay of a case $\xi = \{U_0^{10}, V_0^7, W_{10}^{15}\}$ (subscripts and superscripts correspond to start and end times, respectively) on the process tree in Figure 6b. The replay procedure will match the events to the subtree representing concurrent execution of U and V, which yields a replayed case $\xi_T = \{U, V, W\}$. The durations for U, V, W are then sampled, e.g., as $x_U = 7, x_V = 1, x_W = 4$. Respecting the partial order induced by the subtree, the start and completion times of U, V are set first, before the start time of W is set to be the maximum of completion times of U and V. As a result, the following replayed case is created $\xi_T = \{U_0^7, V_0^1, W_7^{11}\}$.

Performance replay is based on a stochastic model. As such, the *replayed event log*, defined as $L_T = \{\xi_T \mid \forall\ \xi \in L\}$, is different for every enactment of the replay procedure for L over T. To make the event log and the model comparable, we thus use the following statistical approach. We conduct performance replay K times, independently, and consider every replayed log L_T to be a sample from T. This results in a sequence of K replayed logs indexed $L_T^{(1)}, \ldots, L_T^{(K)}$. Then, we define performance fitness between an event log L and a process tree T, as a statistical comparison between L and the sequence of K replayed logs $L_T^{(1)}, \ldots, L_T^{(K)}$.

To simplify notation, we assume that the event log L contains N cases with each case denoted by $\xi_i, i = 1, \ldots, N$; the corresponding replayed cases in the k-th event log are denoted by $\xi_T^{(i,k)}$. We are now ready to define the *performance fitness score*, which quantifies the distance between an event log and a process tree.

Definition 5 (Performance Fitness Score (PFS)) *Given an event log L, a sequence of replayed logs $\{L_T^{(k)}\}, k = 1, \ldots, K$, a performance measure ψ, and a loss function l, the performance fitness score (PFS) between L and T is given by,*

$$S_{K,N}(L, T, \psi, l) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{K} \sum_{k=1}^{K} l(\psi(\xi_i), \psi(\xi_T^{(i,k)})). \tag{1}$$

5.2 Guarantees on Performance Fitness for TNR-based Inductive Mining

It turns out that inductive mining with TNR-based delay unfolding and probabilistic variant mining comes with guarantees on the performance fitness of the discovered model. To capture these guarantees, we first need the notion of the *expected performance fitness score*, defined as follows:

Definition 6 (Expected Performance Fitness Score (E-PFS)) *Let ξ be a case of an event log L and let ξ_T be the corresponding replayed case. The expected performance fitness score (E-PFS) is*

$$S(L, T, \psi, l) = \mathbb{E}[l(\psi(\xi), \psi(\xi_T))]. \tag{2}$$

The randomness in $l(\psi(\xi), \psi(\xi_T))$ stems from the fact that we observe an arbitrary case from the event log. Assuming that $\psi(\xi_i)$ are independent and identically distributed (i.i.d.) samples from $\psi(\xi)$ and that $\psi(\xi_T^{(i,k)})$ are i.i.d. samples from $\psi(\xi_T)$, the PFS, $S_{K,N}(L, T, \psi, l)$, is an unbiased estimator of $S(L, T, \psi, l)$, since it estimates an expected value by using the sampled mean [15].

Below, we instantiate the performance fitness framework with the sojourn time of a case $\xi = \{e_1, \ldots, e_n\} \in L$, i.e., $\psi'(\xi) = \max_{e \in \xi} e.c - \min_{e \in \xi} e.s$, as a performance measure; and the bias, $l'(x, y) = x - y$, as the loss function. Then, we show that a process tree discovered by inductive mining using TNR-based techniques results in an unbiased process tree (in terms of the E-PFS) with respect to the originating event log.

Theorem 1 *Let L be an event log and let T be a process tree discovered from L using TNR-based delay unfolding and PVM. For performance fitness in terms of sojourn times of a case and the bias loss, the corresponding E-PFS is unbiased: $S(L, T, \psi', l') = 0$.*

Theorem 1 implies that $S_{K,N}(L, T, \psi', l') \to 0$ as the two sample sizes, N (number of cases in the event log) and K (number of replays), increase. Below, we provide a proof sketch, while the full proof can be found in an extended version of this paper.[3]

Proof Sketch 1 *We wish to prove that for an arbitrary ξ,*

$$\mathbb{E}[\psi'(\xi) - \psi'(\xi_T)] = \mathbb{E}[\psi'(\xi)] - \mathbb{E}[\psi'(\xi_T)] = 0. \tag{3}$$

To this end, we write the expression for the difference between the two sojourn times $\psi'(\xi) - \psi'(\xi_T)$ and show that both can be written as the sum of three components: sequential activity durations, sequential delays, and concurrent set durations. Since the replayed case and the log case respect the operators in T and their durations come from the same probability distributions, we show that the expectation of these three components are equal for both ξ and ξ_T, which proves Equation 3, and the Theorem.

Furthermore, by using similar arguments, we can show that any process tree T that does not explicitly consider delays or is derived by threshold-based noise filtering will have a positive bias in terms of sojourn times, when delays have a positive expected value.

6 Evaluation

The above theoretical guarantees are based on several assumptions that may not hold in practice. For example, Theorem 1 holds if the execution times of activities are independent and identically distributed (i.id.), which must not be the case in real-life business processes. Hence, this section evaluates the usefulness of the TNR from a practical point of view. Our experiments show, based on three real-world datasets, that detecting processing delays and probabilistic variant mining improve performance analysis based on the discovered models. In the experiments, we compare the presented approach against models discovered by state-of-the-art inductive miners [4]. Below, we provide details on the datasets (Section 6.1), outline the experimental setting and procedure (Section 6.2), before presenting the main results (Section 6.3).

6.1 Datasets

We utilise three data sources: two event logs that stem from two different processes of DayHospital, a large cancer outpatient hospital in the United States, and an event log that comes from the Rambam hospital, a general hospital in Haifa, Israel.[4]

The first dataset, named 'Consult', corresponds to a patient consultation process in DayHospital. This process involves several procedures including blood draw, physical examination, vital signs, and consulting with a health provider. The process is typically very sequential. The second dataset, named 'Chemo', comes from a chemotherapy treatment process in DayHospital. It is a hybrid manufacturing-service process. Specifically, in order for a patient to receive chemotherapy, they must go through activities such as blood draw, examination, and chemotherapy infusion. During this process the

[3] https://hu.berlin/TNR-extended

[4] Data is available at http://seeserver.iem.technion.ac.il/databases/HomeHospital/

relevant chemotherapeutic drugs is manufactured. The chemotherapy process exhibits concurrency due to the need to serve the patient and produce their medications. For the DayHospital experiments, we used six months of data (03/2014 - 09/2014).

The third dataset, named 'Rambam', originates from a general hospital comprising multiple departments, such as emergency and internal ward. The available data includes process data in terms of department names, start times, and completion times, thereby providing a high-level description of a patient's journey through the hospital as reflected in the information system. For a thorough description of the process and its corresponding data see [16]. For the Rambam hospital, we used a month's worth of data (04/2014).

6.2 Experimental Setting and Procedure

Below, we instantiate the performance fitness framework by setting its building blocks (performance measure ψ and loss function l), and describe the experimental procedure.

Setting. As our performance measure ψ, we chose the sojourn time, which is the total time a case spends in the process. As the loss function, we considered the squared loss, i.e., $l(x, y) = (x - y)^2$. We mine the baseline models via the Inductive Miner–life cycle (IMlc) [4]. Our evaluation scenarios comprise three controlled variables: (1) the dataset, (2) whether to detect delays (TNR approach vs. IMlc), and (3) whether to use probabilistic variant mining (PVM vs. noise filtering).

Procedure. The experimental procedure involved the quantification of the performance fitness score (PFS), namely $S_{K,N}(L, T, \psi, l)$, per scenario. In our case, the PFS corresponds to the average loss in sojourn times. To this end, we discovered and enriched models in correspondence to the 12 aforementioned scenarios. For example, to assess the scenario of the 'Consult' dataset, with delay detection, and probabilistic variant mining, we construct a process tree based on the TNR by applying both delay unfolding and PVM. Next, the model was simulated, case-by-case, with 30 runs per scenario, and the sojourn time of each case was recorded. Lastly, we quantified the empirical root-mean squared error (RMSE) to asses the loss.

6.3 Results

Table 3 summarises our experimental results in terms of the RMSE measure for sojourn time estimation over the 12 scenarios. The 'Delays' column gets values of 'Detected' (if delay unfolding was used), or 'NDetected', otherwise. The 'PVM' value of the 'Noise Handling' column corresponds to probabilistic variant mining (Section 4.2), while 'Happy-Path' corresponds to a 20% noise filtering of variants based on the IMlc. We show the average RMSE across 30 runs and the average sojourn times of cases (in minutes for DayHospital, in hours for Rambam). To scale the accuracy of estimation, we also provide the percentage of error out of the average sojourn time.

For DayHospital, we observe that PVM dominates the deterministic approach by IMlc ($40\% - 60\%$ improvement in estimation error). This points toward the ability of the TNR-based approach to improve performance measurement. Further, delay detection improves performance fitness only when combined with probabilistic mining (up to 6%

Table 3: Experimental results: scenarios and estimated performance fitness score

Dataset	Noise Handling	Delays	Avg Sojourn Time	PFS(RMSE)	Prop. to AVG
Consult	PVM	Detected	73	65	89%
	PVM	NDetected	73	69	95%
	Happy-Path	Detected	66	86	130%
	Happy-Path	NDetected	66	85	129%
Chemo	PVM	Detected	240	172	72%
	PVM	NDetected	240	178	74%
	Happy-Path	Detected	222	233	105%
	Happy-Path	NDetected	222	229	103%
Rambam	PVM	Detected	113	233	206%
	PVM	NDetected	113	234	207%
	Happy-Path	Detected	96	249	259%
	Happy-Path	NDetected	96	250	260%

improvement in accuracy); it does not significantly improve estimation when a 'Happy-Path' model is discovered, as the data is filtered first and the TNR receives a biased version of the event log. This result is explained by the fact that deterministic noise filtering removes unlikely paths, which are typically longer and involve more delays. Thus, the TNR is less likely to contain delays. Rambam hospital scenarios suffer from low estimation accuracy across all scenarios due to large variance in patient sojourn times. Yet, the results described for DayHospital are repeated across the Rambam scenarios.

7 Related Work

Our work falls under the field of process mining [3], as it provides a temporal representation for extracting process models from event data. Closest to our work are discovery algorithms that rely on transactional data to extract control-flow relations, such as Tsinghua-α [8], Inductive Miner–life cycle [4], and the Heuristic Miner++ [17].

In this paper, we focus on discovery of timed models for performance analysis of business processes. This topic was the subject of numerous works in process mining [5,6,18]. Commonly, the proposed techniques treat control-flow discovery, and temporal aspects of the event log separately, which results in sub-optimal results when computing performance measures. To overcome this limitation, we propose a new formalism, namely the Temporal Network Representation (TNR), which is based on Allen's interval algebra [9]. Additional formalisms to represent temporal aspects of information systems have been proposed in the literature. These include: generalised interval algebra, temporal networks, fuzzy temporal knowledge, and time ontology methods, see [19] for an extensive survey. In this work, we selected Allen's algebra as it provides a complete representation of all pairwise relations between time intervals that is simple and transparent [20].

Another line of work, namely predictive process monitoring, aims at online forecasts of key performance indicators (KPIs), e.g., the remaining time, and the next event [21]. Our work is mainly concerned with the ability to reconstruct performance measures as they were observed in the event log, and thus focuses on post-mortem analysis.

This paper proposes two techniques that enhance performance-driven discovery based on the TNR, namely delay unfolding and probabilistic variant mining. Related to the former, a method for detecting (unrecorded) queueing delays from event logs with

missing temporal information is presented in [22]; it models resource availability via an activity-life cycle representation. The TNR provides a more general approach to delay detection, as it does not require a priory knowledge of resource behaviour.

Probabilistic variant mining relates to recent attempts to handle noise in process discovery, based either on event log filtering [10,13,23], or model abstraction [24,25]. While these works rely on deterministic reasoning, requiring a user to decide on over or under representation of process variants in the model, the TNR enables for a probabilistic framework, which better reflects process heterogeneity in the discovered model.

To assess performance-oriented fitness, we proposed the performance fitness score (PFS) that is derived using a stochastic replay of events logs over process trees. The idea to calculate fitness based on a replay procedure was first introduced in [26], and later extended based on the notion of *alignments* [27]. However, these two approaches do not address the situations when the discovered models are stochastic, i.e., comprising time distributions and branching probabilities. The *second-pass* approach presented in [5] involves simulating a (discovered) stochastic model to obtain a sample of event logs. These simulated logs are then compared to the originating event log that was used for discovery, to check whether the discovered model is a good representation of the event log. However, the comparison is performed without a formal notion of model-log distance (or similarity). In our approach, we propose the PFS as a statistic that measures the distance between event logs and process trees. Further, we show that under probabilistic assumptions, the TNR produces unbiased model with respect to the PFS.

8 Conclusion

Targeting the improvement of business processes based on performance analysis, we introduced a novel model of event logs, namely the temporal network representation (TNR). It captures the temporal relations between activities, which is an essential aspect of performance-oriented process model discovery. To demonstrate the effectiveness of the TNR, we introduced two applications. First, we proposed delay unfolding as a means to detect unrecorded processing delays. Second, we presented probabilistic variant mining (PVM) to preserve performance information, while handling noise in event logs. Further, we developed a framework for assessing performance fitness of discovered models. We have shown that under this framework, TNR-based inductive mining is guaranteed to result in unbiased models with respect to the original event log. We evaluated the approach with three real-world datasets from the healthcare domain. We have been able to show an up-to 40% percent improvement in sojourn time estimation, when combining delay detection and PVM.

In future work, we aim at enriching the TNR with workload information to separate delays stemming from resource queueing and those originating from synchronisation.

References

1. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer (2013)
2. Senderovich, A., Weidlich, M., Yedidsion, L., Gal, A., Mandelbaum, A., Kadish, S., Bunnell, C.A.: Conformance checking and performance improvement in scheduled processes: A queueing-network perspective. Inf. Syst. **62** (2016) 185–206

3. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
4. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Using life cycle information in process discovery. In: BPM Workshops. LNBIP 256, Springer (2015) 204–217
5. Rozinat, A., Mans, R., Song, M., van der Aalst, W.M.P.: Discovering simulation models. Information Systems **34**(3) (2009) 305–327
6. Rogge-Solti, A., Weske, M.: Prediction of Remaining Service Execution Time Using Stochastic Petri Nets with Arbitrary Firing Delays. In: ICSOC. LNCS 8274, Springer (2013) 389–403
7. Senderovich, A., Weidlich, M., Gal, A., Mandelbaum, A.: Queue mining - predicting delays in service processes. In: CAiSE. LNCS 8484, Springer (2014) 42–57
8. Wen, L., Wang, J., van der Aalst, W.M., Huang, B., Sun, J.: A novel approach for process mining based on event types. Journal of Intelligent Information Systems **32**(2) (2009) 163–190
9. Allen, J.F.: Maintaining knowledge about temporal intervals. CACM **26**(11) (1983) 832–843
10. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - A constructive approach. In: ATPN. LNCS 7927, Springer (2013) 311–329
11. Alspaugh, T.A.: Software support for calculations in Allen's interval algebra. (2005)
12. Van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. on Knowledge and Data Engineering **16**(9) (2004) 1128–1142
13. Weijters, A., van Der Aalst, W.M., De Medeiros, A.A.: Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven, Tech. Rep. WP **166** (2006) 1–34
14. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: BPM Workshops. LNBIP 171, Springer (2013) 66–78
15. Bickel, P.J., Doksum, K.A.: Mathematical statistics: basic ideas and selected topics. Vol. 2, CRC Press (2015)
16. Armony, M., et al.: On patient flow in hospitals: A data-based queueing-science perspective. Stochastic Systems **5**(1) (2015) 146–194
17. Burattin, A.: Heuristics miner for time interval. In: Process Mining Techniques in Business Environments. Springer (2015) 85–95
18. van der Aalst, W.M.P., Schonenberg, M., Song, M.: Time prediction based on process mining. Information Systems **36**(2) (2011) 450–475
19. Vila, L.: A survey on temporal reasoning in artificial intelligence. Ai Com. **7**(1) (1994) 4–28
20. Freksa, C.: Temporal reasoning based on semi-intervals. Art. Intel. **54**(1-2) (1992) 199–227
21. Maggi, F.M., Francescomarino, C.D., Dumas, M., Ghidini, C.: Predictive monitoring of business processes. In: CAiSE. LNCS 8484, Springer (2014) 457–472
22. Senderovich, A., Leemans, S.J.J., Harel, S., Gal, A., Mandelbaum, A., van der Aalst, W.M.P.: Discovering queues from event logs with varying levels of information. In: BPM Workshops. LNBIP 256, Springer (2015) 154–166
23. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In BPM. LNCS 4714, Springer (2007) 328–343
24. Fahland, D., Van Der Aalst, W.M.: Simplifying discovered process models in a controlled manner. Information Systems **38**(4) (2013) 585–605
25. Senderovich, A., Shleyfman, A., Weidlich, M., Gal, A., Mandelbaum, A.: P3-folder: Optimal model simplification for improving accuracy in process performance prediction. In BPM. LNCS 9850, Springer (2016) 418–436
26. Rozinat, A., van der Aalst, W.M.: Conformance checking of processes based on monitoring real behavior. Information Systems **33**(1) (2008) 64–95
27. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.P.: Alignment Based Precision Checking. In: BPM Workshops. LNBIP 132, Springer (2012) 137–149