Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)

# Timed Partial Order Inference Algorithm

**Kandai Watanabe[1]*, Georgios Fainekos[2], Bardh Hoxha[2], Morteza Lahijanian[1], Danil Prokhorov[2], Sriram Sankaranarayanan[1], and Tomoya Yamaguchi[3]**

[1] University of Colorado Boulder,
[2] Toyota Motor North America, Research and Development,
[3] Woven Planet
kandai.watanabe@colorado.edu, georgios.fainekos@toyota.com, bardh.hoxha@toyota.com,
morteza.lahijanian@colorado.edu, danil.prokhorov@toyota.com, srirams@colorado.edu,
tomoya.yamaguchi@woven-planet.global

## Abstract

In this work, we propose the model of timed partial orders (TPOs) for specifying workflow schedules, especially for modeling manufacturing processes. TPOs integrate partial orders over events in a workflow, specifying "happens-before" relations, with timing constraints specified using guards and resets on clocks – an idea borrowed from timed-automata specifications. TPOs naturally allow us to capture event ordering, along with a restricted but useful class of timing relationships. Next, we consider the problem of mining TPO schedules from workflow logs, which include events along with their time stamps. We demonstrate a relationship between formulating TPOs and the graph-coloring problem, and present an algorithm for learning TPOs with correctness guarantees. We demonstrate our approach on synthetic datasets, including two datasets inspired by real-life applications of aircraft turnaround and gameplay videos of the Overcooked computer game. Our TPO mining algorithm can infer TPOs involving hundreds of events from thousands of data-points within a few seconds. We show that the resulting TPOs provide useful insights into the dependencies and timing constraints for workflows.

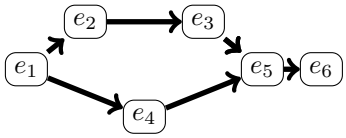
## Introduction

Workflows appear in diverse areas, including business processes (Agrawal, Gunopulos, and Leymann 1998; Datta 1998), software engineering (Cook and Wolf 1998), and factory pipelines (Choudhary, Harding, and Tiwari 2009). The individual events in a workflow, such as the start/end of a particular task or the achievement of an intermediate subgoal, are ordered according to a strict partial order that specifies that some event $e_i$ always *happens before* another event $e_j$. Such partial orders have been used to represent plans in classic AI planning algorithms (Russell and Norvig 2010). Beyond partial orders, we often have timing constraints between events in a workflow that place bounds on the time when an individual event occurs or the time elapsed between two events in the workflow. In this paper, we tackle the key problems of specifying these timing constraints in a succinct manner and mining such schedules from data that consists of sequences of time-stamped events, each representing an execution of the workflow. Such a specification enables us to implement workflow monitoring algorithms, understand sources of timing uncertainties in workflows and optimize the workflow in order to realize cost savings. Beyond monitoring, timing models enable optimal scheduling in real-time by observing the timings of in-progress and completed tasks in order to plan future tasks on-the-fly.

In this paper, we introduce a model called Timed Partial Order (TPO), and propose an algorithm to infer a TPO from timestamped event sequences. The TPO model integrates partial orders with timing constraints that are expressed with clocks, whose idea comes from timed automata specifications (Alur and Dill 1994). The clocks act as timers that express bounds on the time intervals between pairs of events. We first introduce the TPO model and present an analysis of its expressivity. We show that TPOs can succinctly capture a complex set of timing constraints by checking assertions over clocks and selectively resetting these clocks when certain events happen. In particular, we identify structural constraints that yield a restricted class called *race-free* TPOs. We show that race-free TPOs correspond precisely to difference constraints involving time intervals between events. We demonstrate an algorithm that translates a system of explicit constraints on the timing between events into a race-free TPO specification. Next, we solve the problem of mining race-free TPOs from timestamped event sequences. Our approach first mines partial order specification and timing constraints from the data, translating these constraints into race-free TPO specifications. In our TPO model, we allow setting/resetting clocks to enforce the timing constraints just as in the timed-automaton model (Alur and Dill 1994). This allows a procedural approach to monitoring or enforcing these constraints during task execution. To this end, it is desirable to minimize the number of clocks used in order to control for model complexity. However, finding the minimum number of clocks is NP-hard. We examine heuristic algorithms for solving this problem.

We present an evaluation of our approach on a combination of synthetic benchmarks to show that our algorithm can process large numbers of event sequences in a matter of sec-

*This work was primarily performed when the author was an intern at Toyota NA R&D.




| | |
|---|---|
| $e_1$ | Car Arrived |
| $e_2$ | Add Glue Primer |
| $e_3$ | Grip/Lift Window |
| $e_4$ | Clean Dust |
| $e_5$ | Place Window |
| $e_6$ | Release Grippers |

Figure 1: Partial Order for a windshield installation task in an automobile manufacturing facility. Events $e_1, \ldots, e_5$ represent events such as "car arrived" ($e_1$) or the commencement of various tasks such as "clean dust" ($e_4$).

onds, and provides succinct TPO specifications in terms of the number of clocks needed. Next, we demonstrate our approach on two examples inspired by real-life applications: a model of events involved in the workflow for commercial aircraft turnaround and an analysis of the multiplayer computer game Overcooked, as played by beginner and expert players. In both of these examples, we use TPO mining to produce specifications that can yield useful insights about the nature of the workflows in question.

**Example 1.** *Figure 1 shows the set of events that define the (simplified) process of placing a windshield on a car in an automobile manufacturing facility. The events are described along with a directed graph that represents the partial order between events. For instance, the edge from $e_2$ to $e_3$ specifies that the event $e_2$ must always precede $e_3$ for a (successful) windshield installation. However, like many tasks in an assembly line, there are timing constraints that must be respected. Some of the timing constraints for the windshield installation task are summarized in the table below:*

| *ID* | *Interval* | *Cons.* | *Remark* |
|---|---|---|---|
| $C_1$ : | $e_2 \to e_5$ | $\leq 40s$ | *Glue appl. to place window* |
| $C_2$ : | $e_5 \to e_6$ | $\geq 30s$ | *Min. glue setting time* |
| $C_3$ : | $e_1 \to e_4$ | $\leq 5s$ | *Max. time to start cleaning* |
| $C_4$ : | $e_1 \to e_6$ | $\leq 100s$ | *Max. end-to-end time.* |

*$C_1$ enforces that once the glue primer is applied to the window ($e_2$), the window must be placed on the car ($e_5$) within 40 seconds. Similarly, $C_2$ states that once the window is placed on the car, a wait of at least 30 seconds is required for the glue to set before releasing the grippers.*

In this paper, we first describe the model of *timed partial orders*, which combines partial orders between events with ideas from timed automata to represent timing constraints. Next, we show how, given a dataset of time-stamped events, we can mine the timed partial order specification.

## Related Work

Learning a task from log data is known as *Workflow Mining* or *Process Mining*. Most approaches to process mining focus on modeling the order in which tasks are performed, but do not capture timing constraints. As such, process mining has been widely used in the industry (Van der Aalst 2013). It is supported by commercial tools such as Disco, Celonis, and Process Gold; and open source tools such as ProM, Apromore and pm4py (Berti, Van Zelst, and van der Aalst 2019). Under the hood, these tools implement algorithms such as the $\alpha$-algorithm that outputs a Petri Net (Petri 1962), $\alpha+$ algorithm that can handle loops (Van der Aalst, Weijters, and Maruster 2004), and an algorithm that can handle duplicate events (Herbst 2000). Various types of $\alpha$-algorithm were introduced (Van Dongen, Alves de Medeiros, and Wen 2009) to overcome some of its limitations. Other approaches are also introduced such as region-based approaches (Van der Aalst et al. 2010; Carmona, Cortadella, and Kishinevsky 2008) that can express more complex control-flow structures and heuristic mining (Weijters and Van der Aalst 2003), fuzzy mining (Günther and Van Der Aalst 2007), query-based mining (Esparza, Leucker, and Schlund 2010) that can handle incomplete data and genetic process mining (de Medeiros, Weijters, and van der Aalst 2007) that can handle noise. The problem of learning timing constraints has been studied, as well. Berlingerio et al. (2009) focuses on inferring "typical transition time" between two events by counting the number of steps between them. Sciavicco, Zavatteri, and Villa (2021) mine *Conditional Simple Temporal Network with Uncertainty and Decisions* (CSTNUD) models from log data. CSTNUD are temporal networks (Cf. (Dechter, Meiri, and Pearl 1991)), wherein timing differences between events are represented as "durations". These durations can also be viewed as a single clock that resets at every transition. This representation is frequently used in works such as Verwer, de Weerdt, and Witteveen (2012). However, their expression is limited to a single clock, whereas our approach uses multiple clocks that need not be reset at each transition. Multiple clocks are necessary in order to capture more complex timing constraints that are frequently seen in our examples and case-studies. Moreover, their method assumes that dependencies between events are manually provided in the log, e.g., event $E$ happens 5 seconds after $A$, and only depends on *one* event. In contrast, our method can mine the structure at the same time and allows events to depend on multiple events. Moreover, we enforce timing constraints with "clocks" for easier interpretations and faster computation when planning.

Automata such as timed-automata (Alur and Dill 1994) can be used to model workflow schedules. Researchers have extended automata learning techniques to learn timed automata that can capture timing constraints. For instance, Verwer, de Weerdt, and Witteveen (2012) extended the Evidence-Driven State Merging (EDSM)-based algorithms originally proposed by Gold (1978) to learn from data with timestamps and estimate a real-time timed automaton whose edges are labeled with time duration. Although the algorithm is fast, it can only infer simple time constraints. An et al. (2020) also extended the L$^*$-based approach of Angluin (1987), and Tappler, Aichernig, and Lorber (2022) formulated the problem such that it can be solved by Satisfiability Modulo Theories (SMT). However, the former assumes a perfect oracle, and the latter is often very slow due to the nature of the SMT solver, which often leads to exponential time. The genetic algorithm-based approach (Tappler et al. 2019) is fast and gives a good solution if it finds one, but has

no optimality guarantees. All these approaches can estimate some types of Timed Automata, which are very expressive models, but are disadvantageous due to the fundamental difficulty of learning timed automata from trace data.

## Timed Partial Orders

In this section, we first define the preliminary concepts of a timed partial order and show how it can specify timing constraints between events in a manufacturing workflow.

We model workflows as a timed sequence involving a fixed number of *events*. These events may include the start/finish of a given task, or the achievement of a certain physical condition, e.g., the temperature of the water has exceeded 100°C. We assume that the set of events are fixed *a priori*. Furthermore, we assume that repetitions of events are *disambiguated* by giving them unique labels.

Let $\Pi = \{e_1, \ldots, e_n\}$ be the set of events. A given *run* of the workflow is defined by a *timed trace* $\tau : \Pi \to \mathbb{R}_{\geq 0}$ that maps each event with a non-negative timestamp, i.e.,

$$\tau = \{e_1 \mapsto t_1, e_2 \mapsto t_2, \ldots, e_n \mapsto t_n\},$$

wherein $t_i \geq 0$ denotes the timestamp for event $e_i \in \Pi$. With an abuse of notation, we also use $\tau$ as a set.

A timed-trace $\tau$ induces an ordering over the events according to increasing time stamps, and thus may also be viewed as a sequence: $(\sigma_1, t^{(1)}), \ldots, (\sigma_n, t^{(n)})$, wherein each $\sigma_i \in \Pi$ denotes a unique $i^{th}$ event in the trace with corresponding time stamp $t^{(i)}$ and furthermore, $t^{(1)} < t^{(2)} < \cdots < t^{(n)}$. Below, we formulate models for the possible timed traces corresponding to runs of a workflow.

A *(strict) Partial Order* (PO) $P$ is a relation $\prec$ on a set $\Pi$ that is irreflexive, asymmetric, and transitive. We write $e_i \preceq e_j$ if $e_i \prec e_j$ or $i = j$. If $e_i \prec e_j$ holds, then for any timed trace $\tau$ we will require that $\tau(e_i) \leq \tau(e_j)$. We now describe the model of timed POs.

**Definition 1** (Timed Partial Order)**.** *A timed partial order (TPO) is specified by a directed-acyclic graph (DAG) $G : (\Pi, \prec)$ describing a strict partial order over $\Pi$ augmented with the following:*

1. *A finite set of* clocks $C = \{c_1, \ldots, c_m\}$,
2. *A* guard map $G$ *that maps each event $e_i$ to a guard condition, which is a conjunction of the form $G(e_i) : \bigwedge_{j=1}^{n_i} c_j \bowtie a_j$ wherein $c_j \in C$ denotes a clock, $\bowtie \in \{\leq, \geq\}$, and $a_j \in \mathbb{R}_{\geq 0}$ is a non-negative constant,*
3. *A* reset map $R : \Pi \to 2^C$ *that associates each event $e_i$ with a subset of clocks $R(e_i) \subseteq C$ that are to be reset to 0 whenever event $e_i$ is encountered.*

A valuation $\nu : C \to \mathbb{R}_{\geq 0}$ assigns each clock $c_i \in C$ to a non-negative number $\nu(c_i)$. A given valuation $\nu$ can be advanced in time by a fixed $\delta \geq 0$ to yield a new valuation $\nu' := \nu \oplus \delta$ such that $\nu'(c_j) = \nu(c_j) + \delta$ for all $c_j \in C$. Likewise, given a valuation $\nu$ and a subset of clocks $\hat{C} \subseteq C$, we denote the valuation $\nu' := \mathsf{reset}(\nu, \hat{C})$ as that obtained by setting each clock $c \in \hat{C}$ to be 0: $\nu'(c) = \begin{cases} 0 & c \in \hat{C} \\ \nu(c) & c \notin \hat{C} \end{cases}$. Let $\nu_0$ represent a fixed special initial valuation wherein $\nu_0(c_j) = 0$ for all clocks and let $t^{(0)} = 0$. Timestamps represent global time since the inception of the process whereas clocks measure the time elapsed since their last reset.

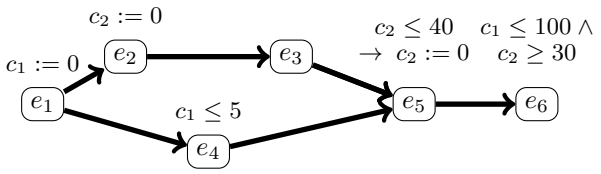



Figure 2: TPO for the car windshield installation workflow.

**Definition 2** (Semantics of Timed Partial Orders)**.** *A* run *of a timed-partial order is a sequence of triples*

$$\rho : (\sigma_1, t^{(1)}, \nu_1), \ldots, (\sigma_n, t^{(n)}, \nu_n),$$

*wherein, each $\sigma_i \in \Pi$, $\sigma_i \neq \sigma_j$ for $i \neq j$, and each $\nu_j$ is a valuation of clocks $C$.*

1. *Time stamps are non-decreasing: $t^{(1)} \leq t^{(2)} \leq \cdots \leq t^{(n)}$. Let $\Delta t^{(j)}$ denote the difference $t^{(j)} - t^{(j-1)}$ for $j \in \{1, \ldots, n\}$ (note that $t^{(0)} = 0$).*
2. *The sequence $\sigma_1, \ldots, \sigma_n$ is a linearization of the partial order $\prec$: if $\sigma_a \prec \sigma_b$ holds then $a < b$.*
3. *For each $j \in \{1, \ldots, n\}$, the valuation given by $\nu_{j-1} \oplus \Delta t^{(j)}$ satisfies the guard condition $G(\sigma_j)$.*
4. *The valuation $\nu_j$ must equal $\mathsf{reset}(\nu_{j-1} + \Delta t^{(j)}, R(\sigma_j))$, i.e, we allow time $\Delta t^{(j)}$ to elapse and then reset the clocks in $R(\sigma_j)$.*

A timed trace $\tau$ viewed as a sequence $(\sigma_1, t^{(1)}), \ldots, (\sigma_n, t^{(n)})$ is *compatible* with a timed partial order specification iff there exists a run of the form $(\sigma_1, t^{(1)}, \nu_1), \ldots, (\sigma_n, t^{(n)}, \nu_n)$.

**Example 2.** *Figure 2 shows the TPO specification for the windshield installation task in Example 1. Two clocks $c_1, c_2$ are used to enforce the constraints $C_1, \ldots, C_4$ from that example. First, we use clock $c_1$ which is reset to zero at the very beginning when event $e_1$ occurs. The guard $c_1 \leq 5$ at event $e_4$ checks that the timing interval between $e_1$ and $e_4$ is at most 5 time units ($C_3$). Likewise, the guard $c_1 \leq 100$ at event $e_6$ ensures constraint $C_4$. The clock $c_2$ is reset first when event $e_2$ occurs. The guard $c_2 \leq 40$ associated with $e_5$ enforces constraint $C_1$. The clock is then reset to zero as part of the same event and then further the guard $c_2 \geq 30$ associated with $e_6$ ensures constraint $C_2$.*

### Expressivity of TPOs

We will now examine the expressivity of TPOs. In general, TPOs enforce timing constraints using clocks. We derive key insights into the nature of these timing constraints. Furthermore, we define useful structural restrictions to TPOs that make the problem of reasoning about their behavior easier. This will pave the way for the TPO mining algorithm that will be presented in the subsequent section.

**Example 3** (Limits on Expressivity)**.** *Suppose, for the windshield installation task in Example 1, we wish to add a constraint – the time taken to clean the windshield (time elapsed*

*between events $e_4$ and $e_5$) must be less than the time taken to add the glue primer to the window (time elapsed between events $e_2$ and $e_3$). Such a constraint compares the intervals between two sets of events. As such, this will not be expressible in the formalism of TPOs. The reason (in part) is that we disallow the guards to compare the values of clocks.*

For the remainder of this section, let us fix a TPO with events $\Pi = \{e_1, \ldots, e_n\}$ and partial order relation $\prec$, clocks $C = \{c_1, \ldots, c_m\}$, guard map $G$ and resets $R$. We will represent a "generic" timed trace $\tau : \{e_1 \mapsto t_1, \ldots, e_n \mapsto t_n\}$ as a vector $(t_1, \ldots, t_n) \in \mathbb{R}^n$, wherein $t_i$ denotes the time at which the event $e_i$ occurs.

**Definition 3** (Constraints representing a TPO)**.** *An assertion $\varphi[t_1, \ldots, t_n]$ involving $t_1, \ldots, t_n$ represents a TPO iff every timed trace $\tau : \{e_1 \mapsto t_1, \ldots, e_n \mapsto t_n\}$ compatible with the TPO satisfies $\varphi$, and conversely, every timed trace satisfying $\varphi$ is compatible with the TPO.*

For the partial order $\prec$, let $\psi_{POR}$ denote the assertion

$$\psi_{POR} : \bigwedge_{e_i \prec e_j} t_j - t_i \geq 0 \ \wedge \ \bigwedge_{j=1}^{n} t_j \geq 0 \qquad (1)$$

expressing the timing constraints of $\prec$.

**Example 4.** *The TPO in Ex. 2 is represented by constraints:*

$$\left( \begin{array}{c} t_1 \leq t_2 \leq t_3 \leq t_5 \ \wedge \ t_1 \leq t_4 \leq t_5 \leq t_6 \ \wedge \ t_5 - t_2 \leq 40 \\ \wedge \ t_6 - t_5 \geq 30 \ \wedge t_4 - t_1 \leq 5 \ \wedge \ t_6 - t_1 \leq 100 \end{array} \right)$$

*The first row represents the relation $\prec$ from the partial order whereas the last row represents the timing constraints $C_1, \ldots, C_4$ discussed in Example 1. Note that (a) TPOs provide a more succinct representation of the timing constraints; and (b) they also specify a procedure to monitor the timed-trace by maintaining some clocks, checking guards on them upon events and resetting the clocks as specified by the TPO. Timing constraints on the other hand require us to potentially store the times of each and every event in the trace.*

Let us consider another example below.

**Example 5.** *Consider a TPO over 3 events $e_1, e_2, e_3$ with the $\prec = \emptyset$. In other words, there is no fixed "happens-before" order between these events.*

$$\underset{(e_1)}{c_1 \leq 1 \rightarrow \ c_1 := 0} \qquad \underset{(e_2)}{c_1 \leq 1 \ \rightarrow \ c_1 := 0} \qquad \underset{(e_3)}{c_1 \leq 1 \ \rightarrow c_1 := 0}$$

*The TPO has a single clock $c_1$, with each event having a guard $c_1 \leq 1$ and resetting the clock $c_1$ to 0. The set of admissible timed traces ensure that: (a) the events $e_1, \ldots, e_3$ may happen in arbitrary order; (b) the first event must appear within 1 time unit of the start of the process; (c) each subsequent event must happen within 1 time unit of the previous event. The following constraints represent this TPO:*

$$\begin{array}{l} (t_1 \leq t_2 \leq t_3 \ \Rightarrow \ t_1 \leq 1 \ \wedge \ t_2 - t_1 \leq 1 \ \wedge \ t_3 - t_2 \leq 1) \wedge \\ (t_1 \leq t_3 \leq t_2 \ \Rightarrow \ t_1 \leq 1 \ \wedge \ t_3 - t_1 \leq 1 \ \wedge \ t_2 - t_3 \leq 1) \wedge \\ \ldots \\ (t_3 \leq t_2 \leq t_1 \ \Rightarrow \ t_3 \leq 1 \ \wedge \ t_2 - t_3 \leq 1 \ \wedge \ t_1 - t_2 \leq 1) \end{array}$$

*Each of the 3! clauses correspond to a linearization of the partial order which leads to different constraints between intervals over successive events as dictated by the TPO.*

To avoid the need to reason over linearization of the underlying partial order, we define a structurally restricted class of TPOs that we call "race-free TPOs". The race-freedom here denotes that the timing constraints remain independent of the actual order in which the events occur. We say that an event $e_i$ in a TPO is *dependent* on a clock $c_j$ iff the guard for $e_i$ refers to $c_j$ or $c_j$ is reset by $e_i$.

**Definition 4** (Race-Free TPOs)**.** *A TPO is said to be race-free iff for every clock $c \in C$ and two different events $e_i, e_j$ that are dependent on c, either $e_i \prec e_j$ or $e_j \prec e_i$. In other words, events that are independent according to the partial order (parallel events) do not refer to the same clock.*

Note that the TPO in Example 2 is clearly race-free. For instance, events $e_2, e_5$ refer to clock $c_2$ but $e_2 \prec e_5$. This can be checked for every clock and every pair of events that are dependent on that clock. However, the TPO in Example 5 is not race-free. Events $e_1, e_2, e_3$ all refer to clock $c_1$ but we do not have any precedence relationship between any of them.

We will now establish the key theorem that characterizes the timing constraints corresponding to race-free TPOs.

**Theorem 1.** *A (race-free) TPO can be represented by a conjunction of inequalities of the form:*

$$\bigwedge_{i,j \text{ s.t. } e_i \prec e_j} (t_j - t_i) \in [\ell_{j,i}, u_{j,i}] \ \wedge \ \bigwedge_{j=1}^{n} t_j \in [a_j, b_j] \,,$$

*wherein $\ell_{i,j} \geq 0, a_j \geq 0$ form lower bounds and $u_{j,i}, b_j \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ are upper bounds that can be non-negative real numbers as well as $+\infty$.*

Full proofs are provided in the appendix. Briefly, the theorem holds because in a race-free TPO, a clock $c$ that is reset at some event $e_i$ and subsequently referred to at event $e_k$ (without intervening reset) ensures that $e_i \prec e_k$. Therefore, the clock $c$ at any point refers to the time difference $t_k - t_i$. Therefore, guards on clocks translate into constraints involving differences $t_k - t_i$. However, if a clock is not reset, it measures the time elapsed since the start of the process. A guard on such a clock is simply a constraint on $t_k$.

The "race-free" assumption limits our ability to compare timings of independent events. For instance, even if the wheels on a car can be mounted in parallel to each other, we may have a constraint that the left front and left rear wheels are attached no more than a minute apart from each other.

**Constructing Timed Partial Orders From Constraints** The main insight behind our paper lies in proving the converse of Theorem 1. Let us fix a set of events $\Pi = \{e_1, \ldots, e_n\}$ and a partial order $\prec$ between them.

**Theorem 2.** *Given timing constraints of the form*

$$\varphi : \bigwedge_{j=1}^{n} t_j \in [a_j, b_j] \ \wedge \ \bigwedge_{e_i \prec e_j} (t_j - t_i) \in [\ell_{i,j}, u_{i,j}] \,, \quad (2)$$

*wherein $a_j, \ell_{i,j} \geq 0$ and $b_j, u_{i,j} \in \mathbb{R}_{\geq 0} \cup \{\infty\}$, there is a race-free TPO that represents the timing constraints $\varphi$.*

The proof of this theorem lies in the procedure we will now present to synthesize such a TPO involving three major steps: (a) Remove *redundant* timing constraints from $\varphi$

to obtain an irredundant representation $\tilde{\varphi}$; (b) Construct a clock allocation graph $G$ from $\tilde{\varphi}$; (c) solve a graph coloring problem on $G$ and (d) translate the graph coloring result into clocks, clock guards and resets for the TPO.

**Example 6.** *Consider once again the windshield installation task from Example 1. Ignoring $C_1, \ldots, C_4$, let us instead consider the following timing constraints:*

$$\varphi : \left( \begin{array}{c} t_3 - t_1 \in [10, \infty] \ \wedge \ t_5 - t_1 \in [0, 15] \wedge \\ t_5 - t_3 \in [0, 5] \ \wedge \ t_6 - t_5 \in [0, 8] \wedge \\ t_5 - t_4 \in [5, \infty] \ \wedge \ t_6 - t_4 \in [4, 10] \end{array} \right) \quad (3)$$

**Redundancy Elimination and Simplification** First, we introduce a fictitious initial event $e_0$ which always happens at fixed time $t_0 = 0$ such that $e_0 \prec e_j$ for all $j \in \{1, \ldots, n\}$. The constraints in Eq. (2) are now written as:

$$\bigwedge_{e_i \prec e_j} \ell_{i,j} \leq (t_j - t_i) \wedge (t_j - t_i) \leq u_{i,j} \, . \quad (4)$$

Next, we eliminate redundant constraints of two types: (a) any constraint of the form $t_j - t_i \bowtie a$ that is implied by the conjunction of the remaining constraints; (b) trivial constraints with $\ell_{i,j} = 0$ or $u_{i,j} = \infty$;. The result of redundancy elimination may be witten as:

$$\tilde{\varphi} : \bigwedge_{i,j} \bigwedge_{k} (t_j - t_i) \bowtie l_{i,j,k}, \text{ wherein } \bowtie \in \{\leq, \geq\} \, . \quad (5)$$

where $k$ iterates over all inequalities that involve $t_j - t_i$. Additionally, for each inequality $t_j - t_i \bowtie l_{i,j,k}$, the corresponding events must satisfy $e_i \prec e_j$. We provide further details on redundancy elimination in the subsequent section.

**Example 7.** *Consider the constraint* (3) *in Example 6. The constraint $t_5 - t_3 \leq 5$ is redundant since we can infer it from the two constraints $t_3 - t_1 \geq 10$ and $t_5 - t_1 \leq 15$. Removing the trival and redundant constraints yields*

$$\tilde{\varphi} : \left( \begin{array}{c} t_3 - t_1 \geq 10 \ \wedge \ t_5 - t_1 \leq 15 \ \wedge \ t_6 - t_5 \leq 8 \\ t_5 - t_4 \geq 5 \ \wedge \ t_6 - t_4 \in [4, 10] \end{array} \right) . \quad (6)$$

**Allocating Clocks to Enforce Constraints** In order to enforce a constraint of the form $t_j - t_i \bowtie l_{i,j,k}$ using clocks: 1. Reset a "dedicated" clock $c_i$ at the same instant when event $e_i$ occurs; 2. Add the guard $c_i \bowtie l_{i,j,k}$ for event $e_j$. In effect, $c_i$ measures time elapsed since event $e_i$. When event $e_j$ happens, its value equals $t_j - t_i$. In fact, the clock $c_i$ can be used to enforce multiple conjunctions of the form $t_{j_1} - t_i \bowtie a_1 \ \wedge \ \cdots \ \wedge \ t_{j_l} - t_i \bowtie a_l$ since the structure of $\tilde{\varphi}$ (Eq. (5)) guarantees that $e_i \prec e_{j_1}, \ldots, e_i \prec e_{j_l}$. Thus, the modified strategy is as follows:

1. Write $\tilde{\varphi}$ as $\tilde{\varphi}_0 \wedge \cdots \tilde{\varphi}_n$, wherein $\tilde{\varphi}_i$ collects all inequalities in $\tilde{\varphi}$ of the form $(t_k - t_i) \bowtie a_{k,i}$.
2. If $\tilde{\varphi}_i$ is not empty, then allocate a dedicated clock $c_i$ that is reset at event $e_i$. In special case, since event $e_0$ is fictitious, we allocate the clock $c_0$ but do not reset it.
3. For the inequality $(t_k - t_i) \bowtie a_{k,i}$ in $\tilde{\varphi}$, add the conjunction $c_i \bowtie a_{k,i}$ to the guard $G(e_k)$ for event $e_k$.

Thus, the scheme so far constructs a TPO with at most $n + 1$ clocks. Since $n$ can be quite large ($\sim 500$) for some manufacturing workflows, we wish to minimize the number of clocks to reduce the complexity of the overall TPO.

**Example 8.** *Continuing from Example 7. Following the technique presented thus far, we split the constraint $\tilde{\varphi}$ (Cf.* (6)*) into three parts (underlining is for emphasis) given by $\tilde{\varphi}_1 = (t_3 - \underline{t_1} \geq 10 \ \wedge \ t_5 - \underline{t_1} \leq 15)$, $\tilde{\varphi}_4 = (t_5 - \underline{t_4} \geq 5 \ \wedge \ t_6 - \underline{t_4} \geq 4 \ \wedge \ t_6 - \underline{t_4} \leq 10)$ and $\tilde{\varphi}_5 = (t_6 - \underline{t_5} \leq 8)$. We allocate three clocks $c_1, c_4, c_5$ to track these three sets of constraints, respectively. Clock $c_1$ is reset at event $e_1$, $c_4$ at event $e_4$ and $c_5$ at event $e_5$. The following guards are added:*

| *Constraint* | *Guard* |
|---|---|
| $t_3 - t_1 \geq 10 \ \wedge \ t_5 - t_1 \leq 15$ | $c_1 \geq 10@e_3, \ c_1 \leq 15@e_5$ |
| $t_5 - t_4 \geq 5 \ \wedge \ t_6 - t_4 \in [4, 10]$ | $c_4 \geq 5@e_5, (c_4 \in [4, 10])@e_6$ |
| $t_6 - t_5 \leq 8$ | $c_5 \leq 8@e_6$ |

**Minimizing Clocks in the TPO** In order to reduce the number of clocks, we ask the following question: under what conditions can we "reuse" the clock $c_i$ corresponding to event $e_i$ for a different event $e_j$?

For any given event $e_i$, let $e_k$ be the event that is maximal according to the precedence relation $\prec$ such that a timing constraint of the form $(t_k - t_i) \bowtie a_{k,i}$ exists in $\tilde{\varphi}$. If no such inequality exists in the first place, then clock $c_i$ would not exist in the first place. We can reuse the clock $c_i$ for any "later" event $e_j$ such that $e_k \preceq e_j$, since the last time clock $c_i$ is used is at event $e_k$. Recall, from TPO semantics in Def. 2, that clocks are reset only *after* the guards are checked.

We construct a *clock allocation graph* $G_c$, an undirected graph whose vertices are the clocks considered so far.

1. Corresponding each clock $c_i$, we compute its latest guarded event $L(i)$ as follows:
   (a) Let $E_i = \{e_k \mid \text{inequality } \ t_k - t_i \bowtie a \text{ in } \tilde{\varphi}\}$.
   (b) Set $\mathrm{L}(i) = \mathsf{sup}_{\prec}(E_i)$, the supremum in $E_i$ according to the $\prec$ order. We observe that clock $c_i$ can be reused after event $L(i)$ has occurred.
2. Add an undirected edge $(c_i, c_j)$ whenever $L(i) \not\preceq e_j$ and $L(j) \not\preceq e_i$ enforcing that $c_i$ and $c_j$ be kept distinct.

Recall that the graph coloring problem seeks to assign one of $m$ colors to each vertex of an undirected graph so that no two vertices connected by an edge have the same color. The main idea behind minimizing clock usage is to examine the optimal coloring of the graph and whenever two nodes $c_i, c_j$ are the same color, we can substitute the use of clock $c_j$ by $c_i$. This process ensures that we use as many clocks as the number of colors used in graph coloring.

**Theorem 3.** *If the clock allocation graph $G_c$ can be colored using $m$ colors, then we can construct a TPO with at most $m$ clocks to represent the timing constraints in $\tilde{\varphi}$.*

**Example 9.** *Continuing with Example 8, we compute the clock allocation graph $G_c$ with vertices $\{c_1, c_4, c_5\}$. Note that $L(1) = e_5$, $L(4) = e_6$, and $L(5) = e_6$ as defined above. Thus, according to the construction above, the clock allocation graph has two edges $\{(c_1, c_4), (c_4, c_5)\}$. This can be colored with two colors and in particular nodes $c_1$ and $c_5$ have the same color. This denotes that the clock $c_5$ can be replaced with $c_1$ everywhere.*

Note that the problem of checking if a graph $G$ may be colored using $m \geq 3$ colors is known to be NP-complete (Garey and Johnson 1979). Nevertheless, graph coloring has been studied for numerous applications including notably register allocation for compilers and scheduling

problems (Chaitin et al. 1981; Lotfi and Sarin 1986). We may employ a simple greedy algorithm for graph coloring (Brélaz 1979) that guarantees that the number of colors is bounded by $1 + \Delta(G_c)$, wherein $\Delta(G_c)$ denotes the maximum number of neighbours for any vertex in $G_c$.

## Mining TPO from Timed Traces

Given timed traces $\tau_1, \ldots, \tau_n$ by observing some workflow with a fixed set of events $\Pi = \{e_1, \ldots, e_n\}$, we wish to synthesize a TPO such that all the timed traces in the given data $D$ are compatible with the TPO. To do so, requires identifying the partial order $\prec$, the clocks, guards and resets.

Our proposed approach proceeds in three steps: (a) identify the partial order information from the timed traces; (b) compute the tightest possible timing constraints of the form (2) that includes all the data; and (c) mine a TPO from the timing constraints in step (b) using the algorithm described in the previous section. Note that steps (a) and (b) are based on well-known techniques that will be briefly recalled in this section. We briefly describe these steps, concluding with a description of our implementation.

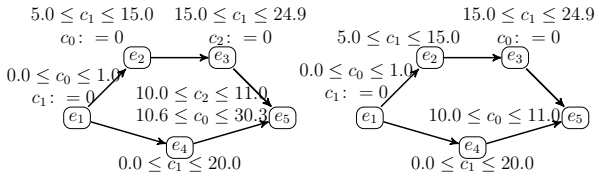
**Partial Order Identification** In order to identify the partial order $\prec$, we set $e_i \prec e_j$ for events $e_i, e_j \in \Pi$ iff in all the timed traces the event $e_i$ precedes $e_j$.

**Formulating Timing Constraints** In order to formulate timing constraints, we translate each timed trace in the data into a vector $(t_1, \ldots, t_n)$. Next, we consider bounds of two types: (a) Upper/lower bounds on each event time $t_i$ by itself to yield intervals $t_i \in [a_i, b_i]$, and (b) Upper/lower bounds on the time difference $t_j - t_i$ whenever $e_i \prec e_j$ holds.

Whereas the interval bounds are simply the maximum and minimum values in the data, there are two drawbacks: (a) The process assumes that all upper bounds are finite since we can never infer a constraint of the form $t_j - t_i \in [a, \infty)$ from the data. However, there are statistical tests from *extreme value theory* that can identify whether a distribution has an infinite support (Haan and Ferreira 2006). The application of these techniques relies on having a large volume of data. (b) The bounds themselves depend intimately on the amount of data and the sampling method. To mitigate this, we refer the reader to ideas from conformal prediction that allow us to bloat the intervals obtained from data appropriately to achieve a prediction with some associated confidence (Balasubramanian, Ho, and Vovk 2014).

**Redundancy Elimination** To eliminate redundancies, we formulate a series of optimization problems involving the constraints in Equation (2). We iterate through each inequality $(t_j - t_i) \leq u_{i,j}$ (alternatively, $t_j - t_i \geq l_{i,j}$ ) from the system in some order and carry out the following steps:

1. Remove the selected inequality and set the objective to maximize (alternatively, minimize) $t_j - t_i$ subject to the remaining constraints.
2. If the resulting optimal value is $> u_{i,j}$ (alternatively, $< l_{i,j}$) then the constraint is irredundant, and needs to be added back to the problem.
3. Otherwise, the constraint is removed once and for all.




(a) NEAREST (b) SOUND

Figure 3: Mined TPOs using the NEAREST and SOUND heuristics.

The optimization problem in question is a linear programming problem that can be solved quite efficiently for the special class of difference constraints encountered here. However, the result of redundancy elimination varies, depending on the order of constraints in which we process the inequalities. For example, the constraint $t_5 - t_1 \leq 15$ in Example 7 can be removed instead of $t_5 - t_3 \leq 5$. The problem of finding the set of irredundant constraints of the least cardinality is known to be NP-hard following a reduction from the minimum equivalent graph problem (Garey and Johnson 1979). Therefore, we consider various heuristics for deciding the order in which the constraints are to be considered.

1. NEAREST: Consider constraints $t_j - t_i$ in increasing order of the number of intermediate events between $e_i$ and $e_j$: i.e, $|\{e_k \mid e_i \prec e_k \prec e_j\}|$.
2. DISTANT: Consider constraints $t_j - t_i$ in decreasing order of the number of intermediate events between $e_i$ and $e_j$.
3. RANDOM: Consider constraints in a randomized order.
4. SOUND: The SOUND algorithm starts from the last node, checks for all time constraints $t_j - t_i \bowtie a$ that are dependent on $t_i$. If ALL of them are redundant, then we remove all the constraints, or else, if ANY of them are required, we keep all the constraints, because the clock at node $i$ is going to be required anyways.

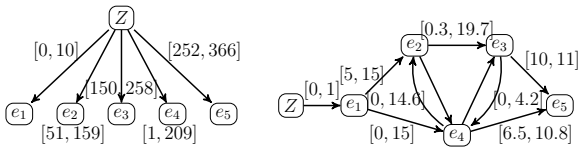
We remark that there is no good approach to choose between these heuristics, other than empirically testing each of them and selecting the best one.

**Example 10.** *Consider Example 1 with the new time constraints* $t_1 \in [0, 1]$, $t_2 - t_1 \in [5, 15]$, $t_3 - t_1 \in [15, 25]$, $t_4 - t_1 \in [0, 20]$, *and* $t_5 - t_3 \in [10, 11]$. *We randomly sampled 1000 timed traces that satisfy these constraints and used the procedure described in this section to construct TPOs, as shown in Figure 3. The TPO to the left uses the* NEAREST *heuristic for redundancy elimination requiring three clocks, whereas the TPO to the right uses the* SOUND *heuristic using just two clocks to explain the same data.*

**Time Complexity** Let $n = |\Pi|$. The algorithm solves a linear program (LP) at each step whose time complexity is bounded by a polynomial over $n$. In the worst case, there are $O(n^2)$ pairs of time constraints (LP problems to solve).

## Experiments

In this section, we evaluate our approach for mining TPOs from timed trace data. First, we compare our method against the most closest work, the CSTNUD mining algorithm. Next, we run an ablative study on a benchmark to evaluate




(a) Global (b) Local

Figure 4: Outputs by the CSTNUD mining algorithm assuming (Left) all the events depend on the global clock $Z = 0$ and (Right) each event depends on the previous event. The constraints $e_4 \rightarrow e_3 : [0, 18.2]$ and $e_2 \rightarrow e_4 : [0, 13.2]$ are not visualized due to the limited space.

the clock allocation performance against a few heuristics. Subsequently, we show interesting results on two datasets inspired from real-life applications: aircraft turnaround and Overcooked game. Our evaluation focuses on how well the resulting model can provide human interpretable insights into the dependencies between the various tasks in the workflow. In particular, we do not classify between different outcomes such as success/failure due to the lack of such labels in our dataset.

**Comparison against a CSTNUD mining algorithm** [1] As a comparison to Example 1, we ran the algorithm proposed by (Sciavicco, Zavatteri, and Villa 2021). Their method requires human annotations on event relationships, so we prepared two different result that can easily be derived from the raw data: 1) all events depending on the global clock (called "global") and each event depending on the previous event (called "local") and showed the resulting graphs in Figure 4. The globally-dependent graph entirely depends on the global clock and cannot mine the relationships between events. As a result, the timings constraints (durations) between events tend to become large. Whereas the locally-dependent graph is shown to mine the event relationships well but with unnecessary edges such as $e_2 - e_4$ and $e_3 - e_4$. In other words, they cannot handle parallelized tasks like ($e_2$, $e_3$) pair and $e_4$. Moreover, the CTSNUD mining algorithm can only mine relationships between neighboring edges that result in a one-clock model whereas our model can be viewed as a generalization to multiple-clock graphs.

**Analysis on Synthetic TPO Data** In this experiment, we seek to understand (a) how the running time of our procedure scales with increasing number of events/traces; and (b) the number of clocks generated by the various redundancy elimination heuristics. We generated a bunch of random TPOs according to a method described in the appendix. The TPOs vary according to the number of events $n$. For each TPO, we randomly sampled 1000 traces.

Figure 5 shows the average number of clocks identified over ten runs of this procedure. The number of clocks increases as the number of events increases. The heuristics yield a similar number of clocks. Computation time did not depend on the choice of the heuristics, but it varied on how many times the LP optimizations were called and how fast

[1] https://github.com/matteozavatteri/cstnud-miner

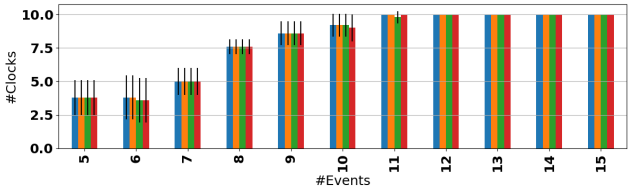



Figure 5: The number of clocks identified by the different heuristic algorithms: DISTANCE (blue), NEAR (orange), RANDOM (green), SOUND (red).

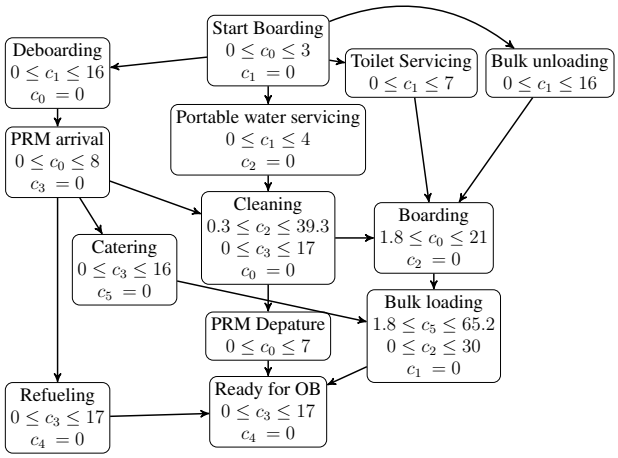



Figure 6: Mined TPO for aircraft turnaround.

they found the solution. The more eliminations happen at the earlier stage, the fewer constraints remain in later LP. Often, however, the extra cost of upfront elimination does not provide enough of a payoff in the later stages. This is very much dependent on the nature of the data and constraints.

**Aircraft Turnaround Example** Next, we evaluate our algorithm on the processes involved in the turnaround of an aircraft at a gate. Aircraft turnaround is a critically important process that affects the operating costs of airlines. It involves a series of tasks such as deboarding, cleaning, refueling and boarding with happens-before orders. For instance, cleaning must be performed after deboarding. However, refueling can be performed in parallel with cleaning. We defined a TPO of the aircraft turnaround operations using data synthesized from the information presented in Nosedal Sánchez and Piera Eroles (2018). In particular, we use *average time that led to delays* as the maximum time for each operation. Timestamps were sampled from truncated normal distributions, as specified by Nosedal Sánchez and Piera Eroles (2018). We synthesized 1000 timed traces and evaluated our algorithm against the SMT-based timed automata inference algorithm (Tappler, Aichernig, and Lorber 2022). The SMT-based algorithm did not terminate over the original data set (timed out due to expensive calls to SMT solvers). We had to reduce the data size to just 20 timed traces in order to get the algorithm to run. However, the result fails to capture the timing constraints present in the original problem. The

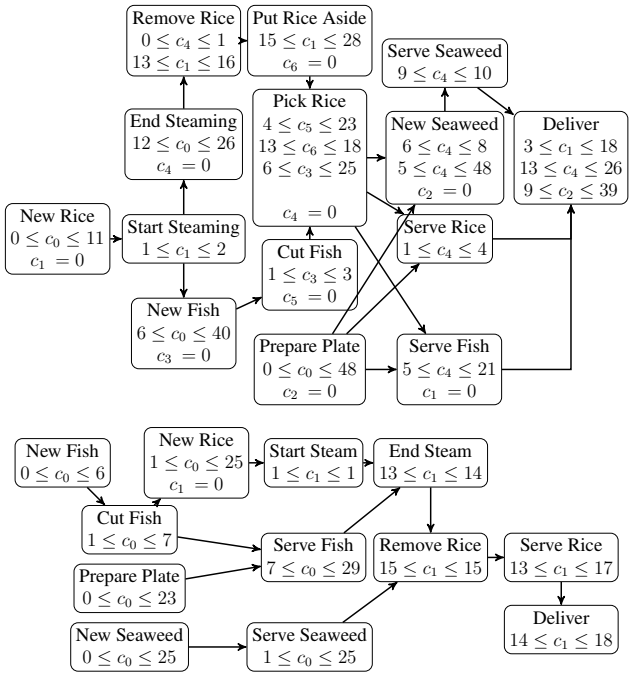



Figure 7: Overcooked gameplay analysis: (Top) TPO specification for a beginner player, (Bottom) TPO specification of a professional player.

RTI+ algorithm (Verwer, de Weerdt, and Witteveen 2012) also fails to run on the reduced data.

In contrast, the TPO mined using our algorithm shown in Figure 6 respects all original precedence orders, along with the maximum time duration for each operation. Furthermore, notice that the algorithm identified new precedence orders such as "Toilet servicing" happens-before "Boarding" due to the relation between the two time constraints. The TPO just requires 6 clocks in all. This example shows that our algorithm can be applied to realistic scenarios.

**Overcooked Example** Overcooked is a multiplayer game that simulates a busy restaurant kitchen, requiring players to collaborate on producing numerous plates of food according to a fixed recipe against timing constraints. We tested our algorithm on publicly available game play videos on YouTube. We analyzed the difference between the beginner[2] versus professional[3] gamers on Overcooked 2. The task of the game is to repeatedly make Sushi plates from seaweed, sliced fish, and cooked rice, serve them to customers in a target area. There is no serving order between the three ingredients, but there are strict orders on how each ingredient is prepared. We manually annotated the video with event labels describing the actions of the players. We were able to identify eight timed traces in the beginner's play and collected the same number of traces from the professional's play. The mined TPOs are shown in Figure 7.

In both cases, strict orders are correctly identified. For example, a fish must be cut before serving on a plate. The difference between the two is the order of the parallelizable tasks. Interestingly, the beginners start with *cooking rice* and then *cutting fish*, whereas the professionals start with *cutting fish* and then *cooking rice*. This is because beginners prepare dish one by one, and hence they must start with rice, which takes the longest time to prepare, whereas the professionals cook in a batch and the fish comes first in this strategy.

In terms of time constraints, strict constraints are imposed, such as (1) fish/seaweed must be cut/served immediately after a new object is taken out of the box (2) rice must be steamed for about 13-16 seconds in both figures. In contrast, a new object (plate, rice, and fish) can be taken out of the box at any time in the scene (with a large bound in $c_0$). Furthermore, a time constraint between serving fish and rice (Serve Rice $4 \leq c_1 \leq 14$) in Figure 7 (bottom) specifies that parallel tasks must take similar times, so the dish can be delivered immediately after (Deliver $1 \leq c_4 \leq 2$). This experiment shows that our TPO mining algorithm provides an interpretable representation for a task solely based on a small amount of data.

## Discussion

**Pipeline Workflow: Repetitive Events** In our problem setting, we assumed that the data log contains a neatly classified set of traces and each trace contains only a unique set of events. However, in reality, a log consists of a set of mixed traces sorted by timestamps and there could be multiple occurrences of the same event in a trace. For example, an automotive assembly line manufactures multiple cars concurrently and the same events (e.g., install a door) appear multiple times in each trace. To split the log into a set of traces, we need to identify the counts of each event appearing in a trace $x \in \mathbb{Z}^n_{>0}$ and split the log accordingly. To do so, we formulate the problem as an integer programming. Given a number of products being manufactured $k$ and a log, minimize $x$, such that, $k \cdot I \cdot x + I \cdot y = b, x > 0, y >= 0$ and $y < k$, where $y \in \mathbb{Z}^n_{\geq 0}$ is a vector of variables representing the remaining events in a trace and $b \in \mathbb{Z}^n_{>0}$ is the counts of events in the log. Then, we greedily split the log from the top with each trace containing $x$ counts of events.

**Loops** In our formulation, we cannot model loops in the partial order graph. However, in reality, a data log can come from a workflow that requires certain events to loop. For, example, cracking three eggs can be represented as repetitions of *NewEgg* and *Crack* events. To learn a TPO from such data, the naive way is to relabel the repetitive events with unique events, e.g., *NewEgg1*, *NewEgg2*. Once the partial among other events are identified, then the repeated events can be folded. For example, *Crack1* can be relabeled back too *Crack* and add an edge to *NewEgg* to form a loop.

## Conclusions

We have introduce and analyzed the expressivity of TPOs, leading to a procedure for mining TPOs from data. Experiments demonstrate how mining TPOs from process data can yield useful insights for important workflows inspired by real-life manufacturing processes.

[2]https://www.youtube.com/watch?v=jTrenjjZDtA&t=668s

[3]https://www.youtube.com/watch?v=YcnpWo4Y01M&t=60s

## Acknowledgments

We gratefully acknowledge comments from the anonymous reviewers that allowed us to improve our presentation. This work was funded in part by US National Science Foundation under award numbers CNS-1932068 and CNS-1932189.

## References

Agrawal, R.; Gunopulos, D.; and Leymann, F. 1998. Mining process models from workflow logs. In *EDBT*, 467–483. Springer.

Alur, R.; and Dill, D. L. 1994. A Theory of Timed Automata. *TCS*, 126(2): 183–235.

An, J.; Chen, M.; Zhan, B.; Zhan, N.; and Zhang, M. 2020. Learning one-clock timed automata. In *TACAS*, 444–462. Springer.

Angluin, D. 1987. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2): 87–106.

Balasubramanian, V. N.; Ho, S.-S.; and Vovk, V. 2014. *Conformal Prediction for Reliable Machine Learning*. Morgan Kaufmann.

Berlingerio, M.; Pinelli, F.; Nanni, M.; and Giannotti, F. 2009. Temporal mining for interactive workflow data analysis. In *KDM*, 109–118.

Berti, A.; Van Zelst, S. J.; and van der Aalst, W. 2019. Process mining for python (PM4Py): bridging the gap between process-and data science. *arXiv preprint arXiv:1905.06169*.

Brélaz, D. 1979. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4): 251–256.

Carmona, J.; Cortadella, J.; and Kishinevsky, M. 2008. A region-based algorithm for discovering Petri nets from event logs. In *BPM*, 358–373. Springer.

Chaitin, G. J.; Auslander, M. A.; Chandra, A. K.; Cocke, J.; Hopkins, M. E.; and Markstein, P. W. 1981. Register allocation via coloring. *Computer Languages*, 6(1): 47–57.

Choudhary, A. K.; Harding, J. A.; and Tiwari, M. K. 2009. Data mining in manufacturing: a review based on the kind of knowledge. *Journal of Intelligent Manufacturing*, 20(5): 501–521.

Cook, J. E.; and Wolf, A. L. 1998. Discovering models of software processes from event-based data. *ACM Trans. on Soft. Engg. (TOSEM)*, 7(3): 215–249.

Datta, A. 1998. Automating the discovery of as-is business process models: Probabilistic and algorithmic approaches. *Information Systems Research*, 9(3): 275–301.

de Medeiros, A. K. A.; Weijters, A. J.; and van der Aalst, W. M. 2007. Genetic process mining: an experimental evaluation. *Data mining and knowledge discovery*, 14(2): 245–304.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial intelligence*, 49(1-3): 61–95.

Esparza, J.; Leucker, M.; and Schlund, M. 2010. Learning workflow petri nets. In *Intl. Conf. Appl. and Theory of Petri Nets*, 206–225. Springer.

Garey, M. R.; and Johnson, D. S. 1979. *Computers and Intractability: A guide to the theory of NP-Completeness*. W.H.Freeman.

Gold, E. M. 1978. Complexity of automaton identification from given data. *Information and control*, 37(3): 302–320.

Günther, C. W.; and Van Der Aalst, W. M. 2007. Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In *BPM*, 328–343. Springer.

Haan, L.; and Ferreira, A. 2006. *Extreme Value Theory*. Springer.

Herbst, J. 2000. A machine learning approach to workflow management. In *European conference on machine learning*, 183–194. Springer.

Lotfi, V.; and Sarin, S. 1986. A graph coloring algorithm for large scale scheduling problems. *Computers & Operations Research*, 13(1): 27–32.

Nosedal Sánchez, J.; and Piera Eroles, M. A. 2018. Causal analysis of aircraft turnaround time for process reliability evaluation and disruptions' identification. *Transportmetrica B: Transport Dynamics*, 6(2): 115–128.

Petri, C. 1962. Kommunikation mit automaten (phd thesis). *Institut für Instrumentelle Mathematik, Bonn, Germany*.

Russell, S.; and Norvig, P. 2010. *Artificial intelligence: a modern approach*. Prentice Hall series in artificial intelligence. Prentice Hall, 3rd edition.

Sciavicco, G.; Zavatteri, M.; and Villa, T. 2021. Mining CSTNUDs significant for a set of traces is polynomial. *Information and Computation*, 281: 104773.

Tappler, M.; Aichernig, B. K.; Larsen, K. G.; and Lorber, F. 2019. Time to learn–learning timed automata from tests. In *FORMATS*, 216–235. Springer.

Tappler, M.; Aichernig, B. K.; and Lorber, F. 2022. Timed Automata Learning via SMT Solving. In *NASA Formal Methods Symposium*, 489–507. Springer.

Van der Aalst, W.; Weijters, T.; and Maruster, L. 2004. Workflow mining: Discovering process models from event logs. *IEEE Trans. KDM*, 16(9): 1128–1142.

Van der Aalst, W. M. 2013. Process mining in the large: a tutorial. *European Business Intelligence Summer School*, 33–76.

Van der Aalst, W. M.; Rubin, V.; Verbeek, H.; van Dongen, B. F.; Kindler, E.; and Günther, C. W. 2010. Process mining: a two-step approach to balance between underfitting and overfitting. *Software & Systems Modeling*, 9(1): 87–111.

Van Dongen, B. F.; Alves de Medeiros, A.; and Wen, L. 2009. Process mining: Overview and outlook of petri net discovery algorithms. *Petri Nets and Other Models of Concurrency II*, 225–242.

Verwer, S.; de Weerdt, M.; and Witteveen, C. 2012. Efficiently identifying deterministic real-time automata from labeled data. *Machine learning*, 86(3): 295–333.

Weijters, A.; and Van der Aalst, W. M. 2003. Rediscovering workflow models from event-based data using little thumb. *Integrated Computer Aided Engineering*, 10(2): 151–162.