Mining Timing Constraints from Event Logs for Process Model

Zhenyu Zhang[1], Chunhui Guo[2], Shangping Ren[1]
[1]Department of Computer Science, San Diego State University, San Diego, CA 92182, USA
[2]Department of Computer Science, California State University, Los Angeles, CA 90032, USA
zzhang4430@sdsu.edu, cguo4@calstatela.edu, sren@sdsu.edu

***Abstract*—Process mining is a technique for extracting process models from event logs. Event logs contain abundant information related to an event such as the timestamp of the event, the actions that triggers the event, etc. Much of existing process mining research has been focused on discoveries of process models behind event logs. How to uncover the timing constraints from event logs that are associated with the discovered process models is not well-studied. In this paper, we present an approach that extends existing process mining techniques to not only mine but also integrate timing constraints with process models discovered and constructed by existing process mining algorithms. The approach contains three major steps, i.e., first, for a given process model constructed by an existing process mining algorithm and represented as a workflow net, extract a time dependent set for each transition in the workflow net model. Second, based on the time dependent sets, develop an algorithm to extract timing constraints from event logs for each transition in the model. Third, extend the original workflow net into a time Petri net where the discovered timing constraints are associated with their corresponding transitions. A real-life road traffic fine management process scenario is used as a case study to show how timing constraints in the fine management process can be discovered from event logs with our approach.**

I. Introduction

Over the last decade, process mining has emerged as a new research field that uses available data to understand a process and improve it when possible. Given an event log, the goal of process mining is to extract process knowledge (e.g., process models) in order to discover, monitor, and improve the real processes[1]. Process mining has been applied successfully in many application domains, such as in banking, insurance, e-government, and medical, to name a few.

The starting point for process mining is event logs. An event log contains information about a working process as it takes place. Each event in such a log reflects an activity which is a well-defined step in the process and is related to a particular trace of a process instance. An event log may also contain other information related to the event such as the executor of the event, the timestamp of the event, and the recorded data related to the event.

One of the most challenging problems in process mining is to discover a structured process model from a set of traces in event logs. Several research groups have been working on techniques for automated process discovery based on event logs. The goal of many process mining algorithms, such as Alpha algorithm [2], region-based approaches [3], [4], heuristic approach [5], to name a few, is to construct a process model from a set of event logs. However, these algorithms focus only on the functional aspects of a process, the timing information as to *when* an action must take place is neglected.

However, the timing information can be critical in many domains, such as in patient care. For instance, ionized calcium test must be tested within 10 minutes of its collection. Blood specimens received in a lab after 10 minutes need to be re-collected for accurate results. These timing constraints need to be enforced in order to provide safe and effective patient care. However, process models uncovered by most existing process mining techniques only reveal the underline structures of an actual process, the timing constraints, such as the 10 minute limit in ionized calcium test example, are not reflected in the process models.

In this paper, we present an approach that extends existing process mining techniques to not only mine but also integrate timing constraints with a workflow or process model constructed by any existing process mining algorithm. The approach contains three major steps, i.e., first for a given workflow model constructed by an existing process mining algorithm and represented by a workflow net (which belongs to a subset of Petri net) and, extract a time dependent set associated with each transition in the workflow model. Second, based on the time dependent sets, develop an algorithm to extract implicit timing constraints from event logs for each transition in the model. Third, extend the original workflow model into a time Petri net where the discovered timing constraints are associated with their corresponding transitions. Fig. 1 depicts the architecture of our approach. A real-life road traffic fine management process [6] is used as a case study to investigate the effectiveness and validity of the approach. The case study provides the evidence that the approach is able to discover timing constraints of an actual workflow process from event logs.

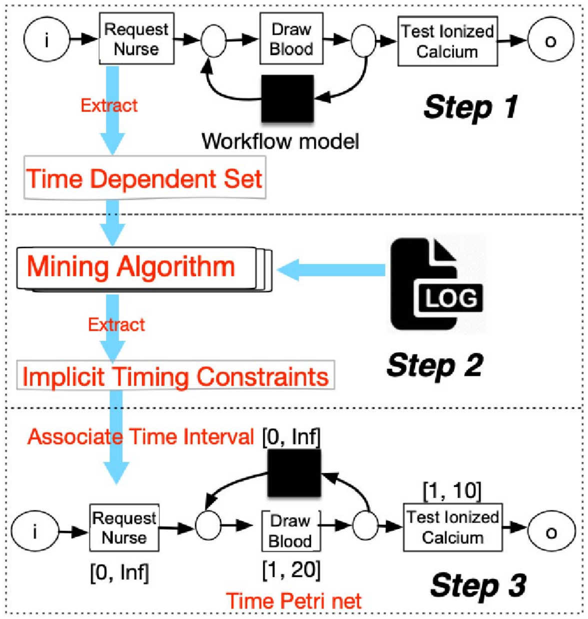


Fig. 1. Mining and Integrating Timing Constraints in Workflow Models

The paper is organized as follows. Section II introduces definitions and notations used in the rest of the paper. We present an approach to extract time dependent set in Section III. Section IV develops the algorithm to mine the timing constraints from event logs based on time dependent sets and integrate the timing constraints into the workflow net in the form of the time Petri net. Section VI discusses the related work of process mining. A real-life road traffic fine management process scenario is performed in Section V to investigate the effectiveness and validity of the approach. We conclude in Section VII.

978-1-7281-7303-0/20/$31.00 ©2020 IEEE
DOI 10.1109/COMPSAC48688.2020.0-139

II. Process Models

In this section, we introduce definitions and notations that will be used in the later sections of this paper. Some of the definitions are cited here for self-containment.

Definition 1 (Petri net [7]). *A Petri net N is a tuple (P, T, F), where*

- *P is a finite set of places;*
- *T is a finite set of transitions, $P \cap T = \emptyset$; and*
- *$F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs.*

Notation 1 ($\bullet t$). *Given a Petri net $N = (P, T, F)$ and a transition $t \in T$, we use notation $\bullet t$ to represent a set of places immediately before transition t, i.e., $\bullet t = \{p | p \in P \wedge (p, t) \in F\}$.*

Notation 2 ($t\bullet$). *Given a Petri net $N = (P, T, F)$ and a transition $t \in T$, we use notation $t\bullet$ to represent a set of places immediately after transition t, i.e., $t\bullet = \{p | p \in P \wedge (t, p) \in F\}$.*

The state of a Petri net $N = (P, T, F)$ is represented by its markings which is a distribution of tokens on places P. A marking of N is defined by a mapping function $m : P \to \mathbb{N}$, where $\mathbb{N}$ is the natural number set. A place p is *marked* by a marking m if $m(p) > 0$.

The execution semantics of a Petri net $N = (P, T, F)$ are defined by transition firings which specify the enabling conditions and the marking transformation of the Petri net. A transition $t \in T$ is enabled by a marking m if m marks all places in its pre-set $\bullet t$, i.e.,

$$\forall p \in \bullet t : m(p) > 0. \tag{1}$$

The firing of an enabled transition t transforms the marking m to m' as below:

$$m'(p) = \begin{cases} m(p) - 1 & \texttt{if} \quad p \in \bullet t \wedge p \notin t\bullet \\ m(p) + 1 & \texttt{if} \quad p \notin \bullet t \wedge p \in t\bullet \\ m(p) & \texttt{otherwise} \end{cases} \tag{2}$$

We use the Petri net shown in Fig. 2 to explain the above concepts. In the graphical representation, *places*, *transitions*, *arcs*, and *tokens* are represented by circles, squares, arrows, and black dots, respectively. The Petri net N shown in Fig. 2 is defined as (P, T, F), where $P = \{p_1, p_2, p_3, p_4\}$, $T = \{t_1, t2\}$, and $F = \{(t_1, p_1), (p_1, t_2), (t_2, p_2), (t_2, p_3), (t_2, p_4), (p_2, t_1)\}$. The current marking is $m = \{(p_1, 3), (p_2, 0), (p_3, 0), (p_1, 1)\}$. Based on formula (1), the transition t_2 is enabled by the marking m. After the firing of t_2, according to formula (2), the new marking becomes $m' = \{(p_1, 2), (p_2, 1), (p_3, 1), (p_1, 2)\}$.

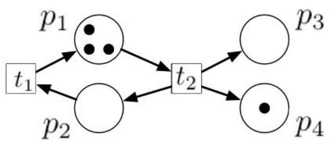


Fig. 2. A Petri net Example

The time Petri net [8] extends Petri net by specifying a time interval for every transition to constrain firing duration. The definition is as follows.

Definition 2 (Time Petri Net [8]). *A time Petri net N is a tuple (P, T, F, I) such that (P, T, F) is a Petri net and I associates each transition $t \in T$ with a static firing time interval, i.e., $I : T \to \{[a, b] \in \mathbb{R}^* \times (\mathbb{R}^* \cup +\infty) | a \leq b\}$, where $\mathbb{R}^*$ is the set of all non-negative real numbers.*

Given a time Petri net $N = (P, T, F, I)$, the firing time interval $I(t) = [a, b]$ for transition $t \in T$ specifies the earliest firing time a and the latest firing time b after the transition t is enabled. For instance, suppose the transition t is enabled at time instance τ, t can only be fired during time interval $[\tau + a, \tau + b]$. If the transition t is not fired during $[\tau + a, \tau + b]$, then t becomes disabled.

If a transition t's firing time interval is $I(t) = [0, +\infty]$, it indicates that t can be fired immediately after enabling and never becomes disabled if the N's state does not change. Hence, if $\forall t \in T : I(t) = [0, +\infty]$, the given time Petri net $N = (P, T, F, I)$ has equivalent execution semantics with Petri net $N' = (P, T, F)$.

Note that the process model mined by most existing algorithms [9], [10], [4] is a subclass of the Petri net named a *workflow* net with three further constraints: (1) there is one and only one *input place* where a process starts; and (2) there is one and only one *output place* where the process ends; and (3) all elements are on a path from the input place to the output place. The formal definition is given below.

Definition 3 (*Workflow net* [11]). *A net $N = (P, T, F)$ is a workflow net, if it is a Petri net and satisfies the following constraints:*

- *There is one and only one input place i, i.e.*
  1) $\exists i \in P$, s.t. $\forall t \in T, (t, i) \notin F$,
  2) $\exists i_1, i_2 \in P$, $(\forall t \in T, (t, i_1) \notin F \wedge (t, i_2) \notin F) \to i_1 = i_2$.
- *There is one and only one output place o, i.e.*
  1) $\exists o \in P$, s.t. $\forall t \in T, (o, t) \notin F$,
  2) $\exists o_1, o_2 \in P$, $(\forall t \in T, (o_1, t) \notin F \wedge (o_2, t) \notin F) \to o_1 = o_2$
- *For a pseudo transition $\xi \notin T$, the net $(P, T \cup \{\xi\}, F \cup \{((o, \xi), (\xi, i)\})$ is a strongly connected net.*

It is worth pointing out that the three constraints do not change neither syntax nor execution semantics of Petri nets and time Petri nets. Since our work is based on workflow net derived by existing process mining algorithms, we represent the process model in terms of time Petri net that follows the three constraints listed above.

III. Extracting Time Dependent Set from a Workflow Net

The event logs considered in this paper is similar to the one defined in [12]. In particular, for a given activity set Σ, an event entry e in an event log records an activity happening in the operation of a process. It not only records the activity itself, which is a well-defined step in the process, but also contains other information, such as the timestamp of when the activity takes place. An event trace σ is a finite sequence of event entries ordered by their occurrence time. An event log L is a set of traces. We assume event logs do not have noise, i.e. every event entry is a truthful recording of an activity. In addition, we assume the event log has sufficiently large number of event entries. We give the definitions of event entry, event trace, and event log as follows. Suppose the workflow net $N = (P, T, F)$ is mined from an event log L. The transition set in N represents the activity set recorded in the event log L, i.e., $T = \Sigma$. Hence, we also use the notation t to represent an activity in the event log L.

Definition 4 (Event Entry). *Given an activity set Σ, an event entry e is a tuple (t, τ), where $t \in \Sigma$ is an activity and τ is the timestamp of t.*

Definition 5 (Event Trace). *An event trace σ is a finite sequence of event entries $e_1, e_2, \ldots, e_i, \ldots, e_n$ satisfying $\forall 0 < i < n : \tau_i < \tau_{i+1}$, where $e_i = (t_i, \tau_i)$.*

Definition 6 (Event Log). *An event log L is a set of event traces $\{\sigma\}$.*

For an illustration purpose, we consider a simplified event log shown in Table I. This log contains information about five traces. Note that the ordering of events within a trace is relevant, while the ordering of events among different traces is of no importance. The log shows that for **trace** 1 and 3, the tasks A, B, C, D were executed. For **trace** 2 and 4, the tasks A, C, B, D were executed. For **trace** 5 the tasks A, E ,and D were executed. Each trace starts with the

TABLE I
AN EXAMPLE OF EVENT LOGS

Trace Identifier	Task Identifier	Timestamp
Trace 1	A	08/05/2019 : 08:15
Trace 2	A	08/05/2019 : 08:24
Trace 3	A	08/05/2019 : 09:30
Trace 1	B	08/05/2019 : 10:24
Trace 3	B	08/05/2019 : 10:24
Trace 2	C	08/05/2019 : 10:26
Trace 1	C	08/05/2019 : 10:25
Trace 4	A	08/05/2019 : 11:45
Trace 2	B	08/05/2019 : 11:46
Trace 2	D	08/05/2019 : 12:23
Trace 5	A	08/05/2019 : 13:14
Trace 4	C	08/05/2019 : 13:17
Trace 1	D	08/05/2019 : 13:19
Trace 3	C	08/05/2019 : 14:09
Trace 3	D	08/05/2019 : 14:29
Trace 4	B	08/05/2019 : 14:43
Trace 5	E	08/05/2019 : 15:22
Trace 5	D	08/05/2019 : 15:45
Trace 4	D	08/05/2019 : 16:10

execution of A and ends with the execution of D. In addition, we also observe that if task C is executed, task B is also executed. However, for some traces, task C is executed before task B, while for some other traces, it is the other way around.

In the following, we will first use an example to illustrate the steps to extract the time dependent set for each transition in a workflow net derived from a given event log file. We will then give an algorithm that automates the steps.

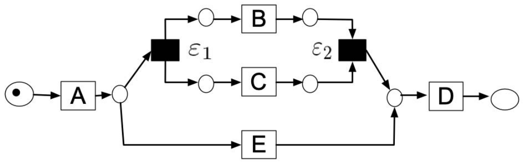


Fig. 3. A process model corresponding to the event log

Consider the event log shown in Table I, we apply an existing process mining algorithm, for instance, the inductive miner algorithm developed by [13], on the event log. The obtained workflow net is depicted in Fig 3. As shown in Fig 3, the workflow net starts at task A and finish after task D. In the workflow net constructed by the inductive miner algorithm, there are invisible transitions represented in black rectangles, such as ε_1 and ε_2, in Fig 3. Invisible transitions are only for routing purposes. They are produced by process mining algorithms, not represent recorded activities in event logs.

Before we show how to extract a time dependent set of a transition, we first give its formal definition.

Definition 7 (Time Dependent Set)**.** *Given a workflow net $N = (P, T, F)$, where $T = A \cup \Gamma$, $A \cap \Gamma = \emptyset$. A is a set of non-invisible transitions, and Γ is a set of invisible transitions. The time dependent relation set $\Theta(t)$ of a transition $t \in A$ is defined as follows:*

$$\begin{aligned}\Theta(t) =&\{a \in A | (a \bullet \cap \bullet t \neq \emptyset) \vee \\ &(\exists \varepsilon_1, ..., \varepsilon_i \in \Gamma : a \bullet \cap \bullet \varepsilon_1 \neq \emptyset \wedge \varepsilon_1 \bullet \cap \bullet \varepsilon_2 \neq \emptyset \\ &\wedge \cdots \wedge \varepsilon_{i-1} \bullet \cap \bullet \varepsilon_i \neq \emptyset \wedge \varepsilon_i \bullet \cap \bullet t \neq \emptyset)\}\end{aligned}$$

Example 1. *The workflow net shown in Fig. 3 has two invisible transitions ε_1 and ε_2 and five non-invisible transitions. According to Definition 7, the time dependent set of the transition D in the workflow net is $\Theta(D) = \{E, B, C\}$. As the output place of invisible transition ε_2 is also the input place of target transition D, we need to trace back to until we find the visible transitions that connect to the invisible transition ε_2, which are transitions B and C. Hence, the time dependent set of transition D is $\{E, B, C\}$*

Algorithm 1 gives the procedure of extracting time dependent set of the transition x in the workflow net. The time complexity of algorithm 1 is $O(N^3)$, where N is the number of transitions in the workflow net.

Algorithm 1: EXTRACTING TIME DEPENDENT SET OF A TRANSITION

```
Data: a workflow net $N = (P, T, F)$, a transition $x$
Result: The time dependent set $\Theta$ of transition $x$
begin
    for each transitions $t \in T$ do
        if $t$ is non-invisible transition then
            $A = A \cup \{t\}$
        else
            $\Gamma = \Gamma \cup \{t\}$
    for each transition $t_0 \in T$ do
        if $\bullet x \cap t_0 \bullet \neq \emptyset$ then
            $\Theta(x) = \Theta(x) \cup \{t_0\}$
    repeat
        Define break condition: $condition = False$ for each
         transition $t_1 \in \Theta(x)$ do
            if $t_1 \in \Gamma$ then
                $condition = True$ $\Theta(x) = \Theta(x) - \{t_1\}$ for
                 each transition $t_2 \in T$ do
                    if $\bullet t_1 \cap t_2 \bullet \neq \emptyset$ then
                        $\Theta(x) = \Theta(x) \cup \{t_2\}$
    until condition
```

IV. MINING TIMING CONSTRAINTS FROM EVENT LOGS

In this section, we design an algorithm to automatically mining timing constraints for every transition based on the time dependent set and represent the workflow model associated with timing constraints by time Petri nets.

Given a workflow $N = (P, T, F)$ mined from an event log L, we first construct a time Petri net $N' = (P, T, F, I)$ where $\forall t \in T$: $I(t) = [0, +\infty]$. The constructed time Petri net N' has the same execution semantics as N as explained in Section II. The next step is to mine the firing time interval for every transition t from event logs and update the corresponding $I(t)$.

As presented in Section III, the time dependent set $\Theta(t)$ of a transition t contains all transitions that have immediate timing impact on the transition t. We use the following rules to determine the timing constraints for each transition in a workflow net from the workflow net's corresponding event logs: if a transition directly follows the workflow net's input place, i.e., $\Theta(t) = \emptyset$, we set its timing constraint as $I(t) = [0, +\infty]$; otherwise, $I(t) = [a, b]$ where a and b are determined as following:

- **Earliest Firing Time:** the transition t's earliest firing time (a) is the minimum value of time between an occurrence of t and its most recent occurrence of any transition $t' \in \Theta(t)$ that occurs before t in the same event trace, i.e., $a = \min\{(\tau_c - \tau_r) | e_c = (t, \tau_c) \wedge e_r = (t_r, \tau_r) \wedge \Phi(e_c, e_r) = \texttt{True}\}$, where

$$\begin{aligned}\Phi(e_c, e_r) =& \forall \sigma \in L : \forall e_i(t_i, \tau_i) \in \sigma : t_i = t \wedge \\ &\tau_c = \tau_i \wedge (\exists e_j(t_j, \tau_j) \in \sigma : 0 < j < i \wedge \\ &t_j \in \Theta(t) \wedge \tau_j = \tau_r \wedge (\forall e_k(t_k, \tau_k) \in \sigma : \\ &j < k < i \wedge t_k \notin \Theta(t)))\end{aligned} \tag{3}$$

- **Latest Firing Time:** the transition t's latest firing time (b) is the maximum value of time between an occurrence of t and its most recent occurrence of any transition $t' \in \Theta(t)$ that occurs before t in the same event trace, i.e., $b = \max\{(\tau_c - \tau_r) | e_c = (t, \tau_c) \wedge e_r = (t_r, \tau_r) \wedge \Phi(e_c, e_r) = \texttt{True}\}$, where $\Phi(e_c, e_r)$ is given in formula (3).

Theorem 1. *Given an event log $L = \{\sigma | \sigma = e_1(t_1, \tau_1), \ldots, e_i(t_i, \tau_i), \ldots, e_n(t_n, \tau_n)\}$ and corresponding time Petri net $N' = (P, T, F, I)$ with $I(t) = [a, b]$ derived by the mining rules for each $t \in T$, all timestamps of corresponding activity t in log L satisfy the timing constraint $I(t) = [a, b]$.*

Proof. We use contradiction to prove the theorem.

For transition t, suppose there exist event entries $e = (t, \tau)$ and $e' = (t', \tau')$ in the same event trace σ, and the transition t's timing constraint $\mathcal{C}(t)$ determined by event entries e and e' is $\tau - \tau' < a$. Because the event entries e and e' determine the timing constraint $\mathcal{C}(t)$ for t, $\Phi(e, e') = \texttt{True}$ holds, where $\Phi(e, e')$ is defined in formula (3). The timing constraint $\mathcal{C}(t) < a$ contradicts the **Earliest Firing Time** rule, i.e., $a = \min\{(\tau_c - \tau_r) | e_c = (t, \tau_c) \wedge e_r = (t_r, \tau_r) \wedge \Phi(e_c, e_r) = \texttt{True}\}$. Hence, transition $t's$ timing constraint must be larger than or equal to a, i.e., $\mathcal{C}(t) \geq a$.

Similarly, suppose there exist event entries $e = (t, \tau)$ and $e' = (t', \tau')$ which determines the transition t's timing constraint $\mathcal{C}(t)$ as $\tau - \tau' > b$. Since the event entries e and e' determine the timing constraint $\mathcal{C}(t)$, $\Phi(e, e') = \texttt{True}$ holds, where $\Phi(e, e')$ is defined in formula (3). The timing constraint $\mathcal{C}(t) > b$ contradicts the **Latest Firing Time** rule, i.e., $b = \max\{(\tau_c - \tau_r) | e_c = (t, \tau_c) \wedge e_r = (t_r, \tau_r) \wedge \Phi(e_c, e_r) = \texttt{True}\}$. Hence, transition $t's$ timing constraint must be smaller than or equal to b, i.e., $\mathcal{C}(t) \leq b$.

Therefore, $a \leq \mathcal{C}(t) \leq b$ holds, i.e., all timestamps of activity t in event log L satisfy the timing constraint interval $I(t) = [a, b]$. ☐

It is worth pointing out that the timing constraints defined by these rules may not be tight. We will give an example to explain a possible reason for its looseness in Section V.

Algorithm 2 gives the detailed steps of applying the rules to automatically mine and update the firing timing constraints for every transition in a workflow net model. The time complexity of Algorithm 2 is $O(K \times M \times N^2)$, where K is the number of transitions in the process model, M is the number of traces in the event log, and N is the number of events in a trace. We illustrate Algorithm 2 in Example 2 with the event log given in Table I.

Example 2. *The Petri net N mined from the the event log L in Table I is depicted in Fig. 3. We first construct a time Petri net $N' = (P, T, F, I)$ where $\forall t \in T : I(t) = [0, +\infty]$. According to Algorithm 1, the time dependent set of N' is $\Theta = \{\Theta(A) = \emptyset, \Theta(B) = \{A\}, \Theta(C) = \{A\}, \Theta(D) = \{B, C, E\}, \Theta(E) = \{A\}\}$.*

We take transitions A and D as examples to illustrate different scenarios of applying Algorithm 2. As $\Theta(A) = \emptyset$, based on Lines 2-4, Algorithm 2 does not update the firing time interval of transition A, i.e., $I(A) = [0, +\infty]$.

The time dependent set of transition D is $\Theta(D) = \{B, C, E\}$. We apply Lines 7-16 of Algorithm 2 for every trace in the event log L. For instance, the trace $\sigma_1 = \{A, B, C, D\}$ only contains one occurrence of event D at time "08/05/2019 : 13:19". Based on Lines 9-15, we locate the event C occurring at "08/05/2019 : 10:25" and calculate the following results $\Delta = 174$, $a = 174$, and $b = 174$ with minutes as the time units. After applying the procedure to other four traces, the final results are $a = 20$ and $b = 174$, i.e., $I(D) = [20, 174]$.

Similarly, we apply Algorithm 2 to other three transitions and get the following results $I(B) = [54, 202]$, $I(C) = [92, 339]$, and $I(E) = [128, 128]$. The updated time Petri net is shown in Fig. 4.

Algorithm 2: MINING TIMING CONSTRAINTS

```
Data: The constructed time Petri net $N' = (P, T, F, I)$, the
      corresponding event log $L = \{\sigma\}$, and the
      corresponding time dependent set $\Theta = \{\Theta(t) | t \in T\}$
Result: The time Petri net $N' = (P, T, F, I)$ with $I$ updated
begin
    for each $t \in T$ do
        if $\Theta(t) = \emptyset$ then
            CONTINUE
    Define temporary variables $a = +\infty$ and $b = -\infty$  for
      each $\sigma = e_1(t_1, \tau_1), \ldots, e_i(t_i, \tau_i), \ldots, e_n(t_n, \tau_n) \in L$
      do
        for each $1 \leq i \leq n$ do
            if $t_i = t$ then
                $j = i - 1$
            while $j \geq 1$ do
                if $t_j \in \Theta(t)$ then
                    $\Delta = \tau_i - \tau_j$
                    $a = \min(a, \Delta)$, $b = \max(b, \Delta)$
                    BREAK
                $j--$
        $I(t) = [a, b]$
```

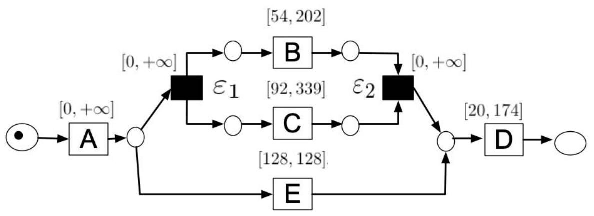


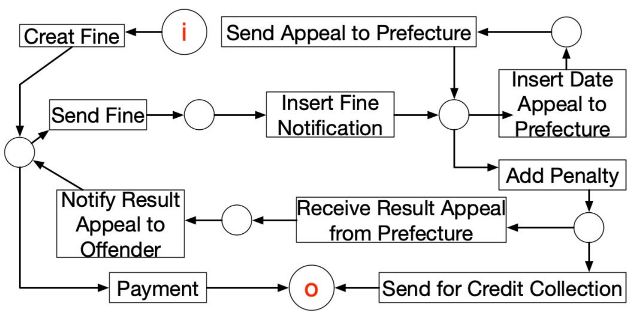
Fig. 4. Updated time Petri net

The proposed techniques of mining timing constraints from event logs for a given workflow net model is implemented on an existing open-source process mining framework *pm4py* [14]. The implementation of the technique is available at https://github.com/stevenzzy9/timing-Constraints-Mining.

V. Road Traffic Fine Management Process Case Study

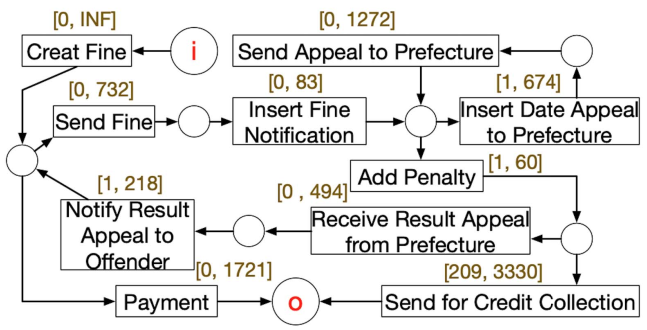
In this section, we apply the technique developed in earlier sections to a road traffic fine management process used in Italy. By the Italian law, a traffic fine management process starts with the *Create Fine* activity. A fine notification must be sent within 90 days since its creation. After being notified, the offender can either pay the fine or appeal the fine within 60 days of notification. If the fine is paid, the corresponding case is closed. The system maintains a case for up to 5 years from the corresponding offense is committed. The snapshot of the event log was taken in June 2013 [15]. After filtering out all open (unclosed) cases, the event log contains 145,800 event traces. Among these traces, 41 % of the traces end after two events, and 51% of the traces have five or more events. In addition, 62 % of the traces take longer than 100 days to finish. Fig. 5 depicts the process model mined by an open source SIMPLE algorithm [16].

We first apply Algorithm 1 to extract the time dependent sets for all transitions in the workflow net given in Fig. 5. The time dependent sets are generated by the tool we developed based on Algorithm 1. Second, using the time dependent sets, we mine the timing constraints from the event log by applying Algorithm 2. The execution results are captured and depicted in Fig. 6. The process model associated



Creat Fine
i
Send Appeal to Prefecture
Insert Date Appeal to Prefecture
Send Fine
Insert Fine Notification
Add Penalty
Notify Result Appeal to Offender
Receive Result Appeal from Prefecture
Payment
o
Send for Credit Collection

Fig. 5. Road Traffic Fine Management Process Model



[0, INF]
Creat Fine
i
[0, 1272]
Send Appeal to Prefecture
[0, 83]
[1, 674]
[0, 732]
Send Fine
Insert Fine Notification
Insert Date Appeal to Prefecture
Add Penalty
[1, 60]
[1, 218]
[0 , 494]
Notify Result Appeal to Offender
Receive Result Appeal from Prefecture
[0, 1721]
[209, 3330]
Payment
o
Send for Credit Collection

Fig. 7. The Workflow Net with Timing Constraints

with mined timing constraints is represented as a time Petri net and depicted in Fig. 7.

```
Create Fine : [0, INF]
Send Fine : [0, 732]
Insert Fine Notification : [0, 83]
Payment : [0, 1721]
Insert Date Appeal to Prefecture : [1, 674]
Send Appeal to Prefecture : [0, 1272]
Receive Result Appeal from Prefecture : [0, 494]
Add penalty : [1, 60]
Notify Result Appeal to Offender : [1, 218]
Send for Credit Collection : [209, 3330]
```

Fig. 6. Timing Constraints

From the mined timing constraints shown in Fig. 7, we can conclude that except the *Send for Credit Collection* transition, the latest firing times of all other transitions are less than 5 years. The latest firing time of the *Insert Fine Notification* transition is 83 days which obeys Italian law's 90-day notification restrict.

If an offender does not pay the fine on time or if the prefecture denies a offender's appeal, the system sends the offender's information to the credit card company to automatically deduct the fine. According to the law, the fine notification and offender response takes up to 90 and 60 days, respectively. Hence, the *Send for Credit Collection* activity must occur after 150 days. The mined timing constraint for *Send for Credit Collection* activity is $[209, 3330]$ which complies with the law and reflects the actual scenario. The time difference between 150 and 209 is the system processing time for *Send for Credit Collection* activity and/or the appeal process time by prefecture.

As shown in Fig. 5, the *Payment* activity has two pre-activities: *Create Fine* and *Notify Result Appeal to Offender*, which represent two scenarios of the *Payment* activity. The first scenario indicated by pre-activity *Create Fine* is that an offender directly pays the fine after the fine creation. The second scenario represented by pre-activity *Notify Result Appeal to Offender* is that an offender appeals against the fine and pays the fine after the appeal is denied. According to the Italian law, the timing constraint for activity *Payment* under the first scenario is $[0, 60]$, but the law does not specify when the *Payment* must happen if the appeal is denied. However, our algorithm does not distinguish these two scenarios and the mined timing constraint $[0, 1721]$ for the *Payment* activity covers all cases of both scenarios. Hence, it may be loose for the first scenario. Our further work is to extend the proposed approach to tighten the timing constraints.

VI. Related Work

Process mining is to discover, monitor and improve real processes from information recorded while the real processes are in operation. The research on process mining started over a decade ago [17], [18], [19], [20], [21], [22], [23], [24], [25]. In [18], Cook and Wolf investigated process mining issues in the context of software engineering processes. They developed three methods for process discovery from a single event stream, namely, using neural networks, purely algorithmic approach, and Markovian approach. The authors consider the purely algorithmic and the Markovian approaches are most promising. The purely algorithmic approach builds a finite state machine from an event stream. The Markovian approach uses a mixture of algorithmic and statistical methods and is able to deal with noise, i.e., rare occurrences. Cook and Wolf later extended their work to concurrent processes [19]. They proposed specific metrics (entropy, event type counts, periodicity, and causality) and used these metrics to discover process models from event streams. In addition, Cook and Wolf provided a quantitative measurement to quantify the discrepancies between a process model and the actual behaviors recorded in an event stream. However, their work is based on a single event stream.

Agrawal et.al [17] developed process mining approaches that are based on multiple event traces, or event logs. One of the most well-known process discovery algorithms based on event logs is the α-algorithm [2]. The α-algorithm scans the event log for particular patterns. For example, if activity a is followed by b but b is never followed by a, then it is assumed that there is a causal dependency between a and b. The α-algorithm is simple and efficient. However, it has difficulty to model a process that contains a loop with only one or two activities. Neither can it handle noises (rare occurrences) in the event logs. Region-based approaches are able to express more complex structures. For practical applications of process discovery it is essential that noise or uncompleted traces are handled well. A few process discovery algorithms focus on addressing these issues, such as the heuristic mining [5], fuzzy mining [10], and genetic process mining [26].

In many application domains, such as in patient care, timing constraints in a process are critical. Unfortunately, not much work is done in the area of mining non-functional behaviors, such as timing constraints, from event logs. One exception is Aalst's work [27] which learns statistical time information from event logs that relates to how long it takes to complete a process at any give state. Different from Aalst's work, we focus on the timing constraints on each transitions, i.e., the time interval in which a transition must take place.

Compared with process mining research which focuses more on extracting process-related insights from event logs, business project management research often focuses on extracting path-related insights from process models, such as identifying a path that may impact a project's completion time. It is often built upon given project/process models, such as the work of finding the longest-duration path in Activity on Edge (AOE) nets [28], [29], [30] and analyzing the actual project plan based on AOE nets [31], [32], while process mining is to discover the process models.

VII. Conclusion

In this paper, we have presented an approach that extends existing process mining techniques to not only mine but also integrate timing constraints with a workflow or process model constructed by any existing process mining algorithm. In particular, we first introduced the concept of time dependent set for each transition in a workflow net model and developed an algorithm that automatically finds the sets from a given workflow net model. Based on the time dependent sets, we developed an algorithm to extract implicit timing constraints from event logs for each transition in the model and represent the process model in terms of time Petri net in which we assign each transition with the appropriate timing constraints mined from event logs. The algorithm is implemented based on the open-source process mining framework *pm4py*. A real-life road traffic fine management process scenario is used as a case study to investigate the effectiveness and validity of the approach. The evaluation results show that the algorithm is able to discover the timing constraints of an actual workflow process from event logs. However, the case study also shows that although the obtained timing constraints are correct, i.e., the timestamps of all event entries satisfy the timing constraints, they may be not be tight. One of our future work is to tighten the constraints. Currently, our work is based on the assumption that there is no noise in the event logs. The other line of future work is to improve our the mining algorithm to tolerate data noise.

Acknowledgement

The work is supported in part by NSF CNS 1842710.

References

[1] Ronny S Mans, Wil MP van der Aalst, and Rob JB Vanwersch. Healthcare processes. In *Process Mining in Healthcare*, pages 11–15. Springer, 2015.

[2] Wil Van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

[3] Marc Solé and Josep Carmona. Process mining from a basis of state regions. In *International Conference on Applications and Theory of Petri Nets*, pages 226–245. Springer, 2010.

[4] Robin Bergenthum, Jörg Desel, Robert Lorenz, and Sebastian Mauser. Process mining based on regions of languages. In *International Conference on Business Process Management*, pages 375–383. Springer, 2007.

[5] AJMM Weijters, Wil MP van Der Aalst, and AK Alves De Medeiros. Process mining with the heuristics miner-algorithm. *Technische Universiteit Eindhoven, Tech. Rep. WP*, 166:1–34, 2006.

[6] Felix Mannhardt, Massimiliano De Leoni, Hajo A Reijers, and Wil MP Van Der Aalst. Balanced multi-perspective checking of process conformance. *Computing*, 98(4):407–437, 2016.

[7] Jörg Desel and Wolfgang Reisig. *Place/transition Petri Nets*, pages 122–173. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[8] Sea Ling and Heinz Schmidt. Time petri nets for workflow modelling and analysis. In *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*, volume 4, pages 3039–3044. IEEE, 2000.

[9] AJMM Weijters and Wil MP van der Aalst. Process mining: discovering workflow models from event-based data. In *Belgium-Netherlands Conf. on Artificial Intelligence*. Citeseer, 2001.

[10] Christian W Günther and Wil MP Van Der Aalst. Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In *International conference on business process management*, pages 328–343. Springer, 2007.

[13] Alejandro Bogarín Vega, Rebeca Cerezo Menéndez, and Cristobal Romero. Discovering learning processes using inductive miner: A case study with learning management systems (lmss). *Psicothema*, 2018.

[11] Wil MP Van der Aalst, AJMM Weijters, and Laura Maruster. Workflow mining: Which processes can be rediscovered. Technical report, BETA Working Paper Series, WP 74, Eindhoven University of Technology, Eindhoven, 2002.

[12] Wil Van Der Aalst. *Process mining: discovery, conformance and enhancement of business processes*, volume 2. Springer, 2011.

[14] *State-of-the-art-process mining in Python*, 2020. https://pm4py.fit.fraunhofer.de/.

[15] *Road Traffic Fine Management Process Data*, 2015. https://data.4tu.nl/repository/uuid:270fd440-1057-4fb9-89a9-b699b47990f5.

[16] AK Alves De Medeiros, Boudewijn F van Dongen, Wil MP Van der Aalst, and AJMM Weijters. Process mining: Extending the α-algorithm to mine short loops. 2004.

[17] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. Mining process models from workflow logs. In *International Conference on Extending Database Technology*, pages 467–483. Springer, 1998.

[18] Jonathan E Cook and Alexander L Wolf. Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 7(3):215–249, 1998.

[19] Jonathan E Cook and Alexander L Wolf. Event-based detection of concurrency. In *ACM SIGSOFT Software Engineering Notes*, volume 23, pages 35–45. ACM, 1998.

[20] Jonathan E Cook and Alexander L Wolf. Software process validation: quantitatively measuring the correspondence of a process to a model. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 8(2):147–176, 1999.

[21] Joachim Herbst. A machine learning approach to workflow management. In *European conference on machine learning*, pages 183–194. Springer, 2000.

[22] Joachim Herbst. Dealing with concurrency in workflow induction. In *European Concurrent Engineering Conference. SCS Europe*. Citeseer, 2000.

[23] Joachim Herbst and D Karagiannis. An inductive approach to the acquisition and adaptation of workflow models. In *Proceedings of the IJCAI*, volume 99, pages 52–57. Citeseer, 1999.

[24] Joachim Herbst and Dimitris Karagiannis. Integrating machine learning and workflow management to support acquisition and adaptation of workflow models. *Intelligent Systems in Accounting, Finance & Management*, 9(2):67–92, 2000.

[25] Marco K Maxeiner, Klaus Küspert, and Frank Leymann. Data mining von workflow-protokollen zur teilautomatisierten konstruktion von prozeßmodellen. In *Datenbanksysteme in Büro, Technik und Wissenschaft*, pages 75–84. Springer, 2001.

[26] Wil MP Van der Aalst, AK Alves De Medeiros, and AJMM Weijters. Genetic process mining. In *International conference on application and theory of petri nets*, pages 48–69. Springer, 2005.

[27] Wil MP Van der Aalst, M Helen Schonenberg, and Minseok Song. Time prediction based on process mining. *Information systems*, 36(2):450–475, 2011.

[28] Jin-Shing Yao and Feng-Tse Lin. Fuzzy critical path method based on signed distance ranking of fuzzy numbers. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 30(1):76–82, 2000.

[29] Cedell Alexander, Donna Reese, and James Harden. Near-critical path analysis of program activity graphs. In *Proceedings of International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 308–317. IEEE, 1994.

[30] Xiao-guang Zhang, Jian Cao, Shen-sheng Zhang, and Qian Fu. Critical path analysis for structure optimization of workflow. *JOURNAL-SHANGHAI JIAOTONG UNIVERSITY-CHINESE EDITION-*, 38(1):0029–33, 2004.

[31] Li Xin Dai and Zong Li. Study on application of critical path to project time management. In *Advanced Materials Research*, volume 328, pages 368–371. Trans Tech Publ, 2011.

[32] Wei Dong, Zhiqiang Zhan, and Xue-song Qiu. A runtime-restricted strategy for highly parallel scheduling human resource in change management. In *2012 IEEE Symposium on Computers and Communications (ISCC)*, pages 000753–000758. IEEE, 2012.