Conformance Checking on Timed Automaton Process Models

Sohei Ito[1][a] and Kento Hamae[2]
[1]*School of Information and Data Sciences, Nagasaki University, 1-14 Bunkyo-machi, Nagasaki City, Japan*
[2]*FCC Techno, 4-25-30 Ohashi, Minami-ku, Fukuoka City, Japan*

Keywords: Conformance Checking, Timed Automaton, Business Process Model, Process Mining.

Abstract: Conformance checking is a kind of process analysis methods that evaluate the difference between modeled behavior and recorded behavior of a process. Usual conformance checking evaluates such difference (called *fitness*) based on only the order of executed tasks. Therefore, one cannot correctly evaluate fitness for process executions that violate the task completion time required by the model. In this paper, we consider process models with time constraints given as timed automata, and propose a method for evaluating fitness of timed traces with timed automaton process models. We implement two algorithms to compute the fitness we proposed, namely a naive exhaustive search and A* algorithm, and show experimental results with simple process models.

1 INTRODUCTION

A process is a collection of tasks (or activities) organized to achieve a certain goal. Processes appear everywhere, and business processes in particular are a prime example. Since organizations are usually composed of many business processes, improving their efficiency is one of the social issues that have been addressed over the years.

A method called *process mining* (van der Aalst, 2016) has been attracting attention for the systematic design, evaluation, and improvement of business processes. Process mining is a collective term for methods that use process models and process execution records (called *event logs*) to perform various analyses. Event logs come from many sources: a database system such as patient data in a hospital, a transaction log of a trading system, an ERP system, etc.

Conformance checking is one of the methods of process mining. It quantitatively evaluate how well a process model conforms to the actual behavior (i.e. event log). Process models are usually obtained through various abstractions, and thus it is not uncommon for errors to be introduced during the modeling process. Moreover, processes themselves vary due to changes in the environment and technological updates. For these reasons, conformance checking is useful to analyze the difference between the model and the process executions.

Several methods have been proposed for conformance checking (Weidlich et al., 2011a; Zha et al., 2010; Weidlich et al., 2011b; Polyvyanyy et al., 2014; Rozinat, 2010; vanden Broucke et al., 2014b; vanden Broucke et al., 2014a; Adriansyah, 2014; Adriansyah et al., 2011a; Adriansyah et al., 2011b; van der Aalst et al., 2012). However, these methods only use activity names for conformance checking, and thus only interested in whether the order of activities conform to the model. In general, however, event logs contain a lot of information, such as timestamps for each activity and unique attributes for each activity type (e.g., “price” for the activity “send quote”). If a process model determines its behavior depending on such attributes, it is desirable to check conformance considering not only the order of activities but also attribute values of activities. For example, in a negotiation process in a trading company (e.g. (Ito et al., 2018)), a salesperson should send a quote within a set time limit, and the quoted price should be within a reasonable bound. It is desirable to check conformance of such constraints besides the correctness of task execution order.

Therefore, we propose a method for checking conformance between process models with the time attribute and timed traces (a trace is an execution of a process). The reason why we only focus on the time attribute is that almost all event logs contain timestamps. For the formalism of process models, we adopt the *timed automaton*. We regard locations of a timed

[a] https://orcid.org/0000-0002-7029-664X

Ito, S. and Hamae, K.
Conformance Checking on Timed Automaton Process Models.
DOI: 10.5220/0012548800003690
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 26th International Conference on Enterprise Information Systems (ICEIS 2024) - Volume 2*, pages 545-556
ISBN: 978-989-758-692-7; ISSN: 2184-4992
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

automaton as activities. Traces are sequences of pairs of an activity and a timestamp (i.e. completion time of the activity), that is, an event log consists of *timed* traces. Since an execution of a timed automaton is a sequence of pairs of locations and times, we can naturally regard timed traces as executions of a timed automaton.

Our proposed method of conformance checking consists of two steps. The first step is to evaluate conformance based only on the order of activities. We call this the *order fitness*, which takes a value between 0 and 1. For this, we adjust an alignment-based conformance checking proposed for Petri net models (Adriansyah et al., 2011b) to finite automaton models. This method finds a matching between an model execution and a trace. The next step is to evaluate the *time fitness* (also takes a value between 0 and 1) of the matching. The time fitness, which we newly define, evaluates how well the timestamps in a trace conform to the constraints given in the model.

The alignment-based method searches the optimal model execution (the execution having the smallest difference from the trace) to compute fitness values. Although there are several such optimal executions in general, they all have the same order fitness value. However, as we see later, such optimal executions possibly have different time fitness values. Therefore, if one wants to find the best time fitness value, one needs to search all possible optimal executions. Such exhaustive search may be computationally expensive if a model is large. Therefore, we implement A* algorithm as a more efficient algorithm, which is also used in (Adriansyah et al., 2011b), as well as a naive exhaustive search algorithm and compare the difference of performance and the obtained best fitness values using simple process models as an experiment.

This paper is organized as follows: Section 2 gives a technical preliminary. Section 3 formally defines the fitness of our conformance checking on timed automaton process models. Section 4 states the algorithms to compute the fitness values. Section 5 presents experimental results of the proposed method. Section 6 gives related work. The final section concludes the paper.

2 PRELIMINARY

In this section, we formally define the event log and the timed automaton process model. An *event log* is a record or a history of executions of a process. An event log consists of many *cases*. A case is an instance of a process and corresponds to a sequence of events (*trace*). An event consists of an *activity* and several *attributes*. In this paper, the only attribute we consider is the timestamp. Each event occurs in only one case.

Let A be the set of activities.

Definition 1 (Event log). $L_A = (E, C, \alpha, \beta, \tau, \succ)$ *is an* event log *over the set A of activities, where E is a set of* events, *C is a finite set of* cases, $\alpha : E \to A$ *is a function that assigns an* activity *to each event,* $\beta : E \to C$ *is a function that relates events to cases,* $\tau : E \to \mathbb{R}_{\geq 0}$ *is a function that gives the timestamp of an event, and* $\succ \subseteq E \times E$ *is a strict total order on E. Let* $\ell \in C$ *be a case identifier. We write* E_ℓ *for the* trace *of* ℓ*, which is a sequence of events* $\langle e_1, \ldots, e_{|E_\ell|} \rangle$*, where* $\beta(e_i) = \ell \Leftrightarrow e_i \in E_\ell$ *for all* $1 \leq i \leq |E_\ell|$ *and* $e_i \succ e_j$ *for all* $1 \leq i < j \leq |E_\ell|$*. We extend* α *to a sequence of events, i.e.,* $\alpha(E_\ell) = \langle \alpha(e_1), \ldots, \alpha(e_{|E_\ell|}) \rangle$ *is the sequence of activities obtained from* E_ℓ*. We write* $\mathsf{last}(E_\ell)$ *for the last event of* E_ℓ*, namely* $e_{|E_\ell|}$*.*

The process model is formally defined as a timed automaton.

Definition 2 (TA-model). *A* timed automaton process model (TA-model *in short) is given as a tuple* $(L, l_0, z, A, t, T, \gamma)$ *where L is a finite set of locations,* $l_0 \in L$ *is the initial location,* $z \in L$ *is the final location, A is a finite set of activities, t is a clock,* $T \subseteq L \times B(V) \times L$ *is a set of transitions,* $B(V)$ *is a set of clock constraints (*guards*) given as an interval* $d_1 < t < d_2$ $(d_1, d_2 \in \mathbb{R}_{\geq 0})$*, and* $\gamma : L \to A$ *is a function that assigns an activity to each location. For a transition* $(p, g, q) \in T$*, we denote* $G(p,q) = g$*. There is no* $s \in L$ *and* $g \in B(V)$ *such that* $(z, g, s) \in T$*, i.e., there is no successor location of z.*

We give an example process model in Figure 1.

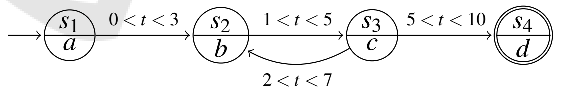

Figure 1: An example process model. Locations are represented as circles, where location names are written above and activity names are written below. The guards of transitions are written above arrows.

Definition 3 (Execution). *An* execution *of a TA-model* $(L, l_0, z, A, t, T, \gamma)$ *is a sequence* $\langle (l_0, t_0), (l_1, t_1), \ldots, (l_n, t_n) \rangle$*, where* l_0 *is the initial location,* $l_n = z$ *(i.e. the final location), and for all* $1 \leq i < n$*,* $(l_i, g_i, l_{i+1}) \in T$ *and* t_i *satisfies* g_i*.*

For example, $\langle (s_1, 1), (s_2, 2), (s_3, 6), (s_4, 8) \rangle$ is an execution of the model in Figure 1.

An execution $\langle \ldots (l_i, t_i), (l_{i+1}, t_{i+1}), \ldots \rangle$ means that the process finishes the activity of l_i at time t_i and moves to location l_{i+1}. Therefore, the timestamp of the final location is not constrained by any guard (because there is no transition from it). One may regard the activity in the final location as just a mark of the end of the process.

For a TA-model $M = (L, l_0, z, A, t, T, \gamma)$, we can obtain an ordinary finite automaton model (called *FA-model*) $M' = (L, l_0, z, A, T', \gamma)$ by forgetting the clock and guards. That is, $T' \subseteq L \times L$ is obtained from T by ignoring guards.

Let $M = (L, l_0, z, A, T, \gamma)$ be a FA-model. A sequence $\sigma = \langle s_0, s_1, ..., s_n \rangle$ is said to be an *execution* of M if $s_0 = l_0$, $s_n = z$, and for all $i \in \{0, ..., n-1\}$, $(s_i, s_{i+1}) \in T$. We write $\mathsf{exe}(M)$ for the set of *executions* of M.

Before concluding this section, we give some remarks about our definitions. In the standard definition of the timed automaton, it has possibly many clocks, and a transition has a set of clocks to be *reset*. This means that when a transition takes place, the clocks in the set are reset to 0 (in general, they can be assigned arbitrary values). We could include the reset of clocks in our definition of the TA-model. In our setting, however, reset is not important because we are not interested in executing the model but in evaluating fitness between the model and the execution, for which only guards on transitions matter. With the reset of clocks, the values of timestamps in an execution do not necessarily increase: for example, the trace $\langle (s_1, 5), (s_2, 3), \ldots \rangle$ is completely fine as an execution of TA-model in Figure 1. We can interpret such execution that the clock was reset between the activity a and b.

Another device that the standard timed automaton has is the *invariant*. Each location has an invariant, which is a condition for the system to stay in the location. For example, if the location s_2 in Figure 1 has the invariant $t < 5$, the system can stay at s_2 as long as $t < 5$ (and thus if $t \geq 5$, the system must exit s_2). For evaluating fitness, one may also be interested in how much the invariants are satisfied during the execution. For simplicity, however, we omit the invariants because they can be treated similarly to the guards.

The definition of executions of a TA-model also differs from the standard definition of runs of a timed automaton. In the standard definition, runs are just a sequence of the pair of locations and valuations of clock $\langle (l_0, u_0), (l_1, u_1), \ldots \rangle$, where (l_{i+1}, u_{i+1}) can be obtained from (l_i, u_i) by two possible transitions: the delay transition and the action transition. The delay transition represents that the system stays at the same location for some time (thus, $l_{i+1} = l_i$). On the other hand, the action transition makes the system to change its location according to the transition T and does not cause a time lapse. Thus, l_{i+1} is one of the successors of l_i in the automaton. Therefore, $\langle (s_1, 0), (s_1, 1.5), (s_2, 1.5), (s_2, 4), \ldots \rangle$ is a run of the timed automaton in Figure 1, but $\langle (s_1, 1), (s_2, 2), (s_3, 6), (s_4, 8) \rangle$ is not because $(s_1, 1) \rightarrow (s_2, 2)$ is neither the delay transition nor the action transition. In our setting, we regard a trace in an event log as an execution of a process model, which is a sequence of events $\langle e_1, e_2, \ldots \rangle$ and each event e_i is a pair (a_i, t_i), where a_i is an activity and t_i is a timestamp. Therefore, we adopt such traces as executions of a TA-model.

3 CONFORMANCE CHECKING ON TIMED AUTOMATON PROCESS MODELS

Conformance checking is a method for evaluating the difference between an event log and a process model. The difference is evaluated by the fitness, which is expressed as a value between 0 and 1. If the event log and the process model do not match at all, the fitness value is 0, and if they match perfectly, the fitness value is 1. If the finesses is greater than 0 and less than 1, there are two possible interpretations. (1) There is a problem on the process model side. The actual business process is not correctly modeled, i.e., there is room for improvement in the process model. (2) There is a problem on the event log side. The actual execution of the process contains some kind of violation, i.e., the operations are not being performed as assumed in the process model. In this case, it may lead to the discovery of inefficient operations.

In this section, we give a method for evaluating the fitness between a TA-model $M = (L, l_0, z, A, t, T, \gamma)$ and a trace E_ℓ in an event log $L_A = (E, C, \alpha, \beta, \tau, \succ)$. The steps for this is as follows:

1. Find an optimal alignment between M' (FA-model obtained from M) and $\alpha(E_\ell)$ (the sequence of activities obtained from E_ℓ), then compute the fitness value of the alignment (we call it the *order fitness*).
2. For the alignment obtained in Step 1, we compute the *time fitness* by evaluating how much the guards are satisfied in it.
3. We take the average of the order and time fitness values obtained in Step 1 and 2, respectively, as the fitness value between M and E_ℓ.

The order fitness can be obtained by adjusting the definition of the fitness based on alignments of Petri net models proposed in (Adriansyah et al., 2011b). We will define the time fitness by quantitatively evaluate how much the timestamps deviate from the intervals given as guards in Section 3.2.

For the fitness between M and L_A (the entire event log), we take average of the fitness between M and E_ℓ for all $\ell \in C$, as usual.

3.1 Order Fitness

In this section, we formally define the order fitness between a FA-model and a trace.

First, we introduce *instances* of FA-models. An instance is a straight line automaton that corresponds to an execution of an automaton.

Definition 4. *Let $M = (L, l_0, z, A, T, \gamma)$ be a FA-model over the activity set A. An* instance $I = (LI, TI, \rho)$ *of M is defined as:*

- *LI is a set of location instances,*
- *$TI \subseteq LI \times LI$ is a set of transition instances,*
- *$\rho : LI \to L$ assigns location instances to locations in L,*

where I satisfies the following:

1. *for all $(x,y) \in TI$, $(\rho(x), \rho(y)) \in T$,*
2. *for all $x \in LI$, $|\mathsf{succ}(x)| \leq 1$,*
3. *for all $x \in LI$, if $|\mathsf{pred}(x)| = 0$ then $\rho(x) = l_0$,*

where $\mathsf{succ}(x) = \{x' \mid (x,x') \in T\}$, $\mathsf{pred}(x) = \{x' \mid (x',x) \in T\}$.

Example 1. Let $I = (LI, TI, \rho)$, where $LI = \{s'_1, s'_2, s'_3, s''_2, s''_3, s'_4\}$, $TI = \{(s'_1, s'_2), (s'_2, s'_3), (s'_3, s''_2), (s''_2, s''_3), (s''_3, s'_4)\}$, $\rho = [s'_1 \mapsto s_1, s'_2 \mapsto s_2, s''_2 \mapsto s_2, s'_3 \mapsto s_3, s''_3 \mapsto s_3, s'_4 \mapsto s_4]$. Then, I is an instance of the FA-model of Figure 1, which corresponds to the execution $\langle s_1, s_2, s_3, s_2, s_3, s_4 \rangle$.

Note that instances may correspond to *partial* executions, which do not end at the final location.

If an instance of a FA-model matches a trace, we call it a *matching instance*, which is formally defined as:

Definition 5 (Matching instance)**.** *Let $L_A = (E, C, \alpha, \beta, \tau, \succ)$ be an event log over the activity set A and $\ell \in C$. Let E' be a prefix of E_ℓ. Let $M = (L, l_0, z, A, T, \gamma)$ be a FA-model and $I = (LI, TI, \rho)$ be an instance of M. Let $\mu : E' \rightharpoonup LI$ be a partial function that assigns events to location instances, and μ induces the bijection from $\mathsf{dom}(\mu) \subseteq E'$ to $\mathsf{rng}(\mu) \subseteq LI$. If the following two conditions hold, I is said to* match *E' by μ and is called a* matching instance*:*

1. *for all $e_1, e_2 \in \mathsf{dom}(\mu)$, if there is a path from $\mu(e_1)$ to $\mu(e_2)$ in I, then $e_1 \succ e_2$,*
2. *for all $e \in \mathsf{dom}(\mu)$, $\rho(\mu(e)) \in \{s \in L \mid \gamma(s) = \alpha(e)\}$.*

For an event e, we denote $\mathsf{loc}(e) = \rho(\mu(e))$. This means that in the matching instance, the execution of the event e is regarded as the execution of the activity of the location $\mathsf{loc}(e)$ in the model. For $p' \in LI$, we write $\mathsf{next}(p')$ for the unique q' such that $(p', q') \in TI$ if it exists.

The condition 1 says that the order of events in the prefix is preserved in the instance. The condition 2 says that an event in the prefix and its corresponding location (related through maps μ and ρ) must have the same activity.

We write $(I_{E'}, \mu)$ for a matching instance that matches the trace E' by μ ($I_{E'}$ is an automaton instance), and write $\mathcal{I}_{E'}$ for the set of all such instances.

Example 2. Let $I = (LI, TI, \rho)$ be the automaton instance in Example 1. Let $E_1 = \langle e_1, e_2, e_3, e_4, e_5 \rangle$, where $\alpha(e_1) = a$, $\alpha(e_2) = b$, $\alpha(e_3) = c$, $\alpha(e_4) = b$ and $\alpha(e_5) = d$. Let $\mu_1 = [e_1 \mapsto s'_1, e_2 \mapsto s'_2, e_3 \mapsto s'_3, e_5 \mapsto s'_4]$. Then, (I, μ_1) is a matching instance. Let $E_2 = \langle e_1, e_2, e_5 \rangle$ and $\mu_2 = [e_1 \mapsto s'_1, e_2 \mapsto s'_2, e_5 \mapsto s'_4]$. Then, (I, μ_2) is also a matching instance.

In Example 2, we can see $e_4 \notin \mathsf{dom}(\mu_1)$. This is interpreted that the event e_4 is *inserted* in the execution of the process, i.e., e_4 should not have been executed. On the other hand, $s'_3 \notin \mathsf{rng}(\mu_2)$. This is interpreted that the activity c (related to the location s''_3) is *skipped* in the execution of the process, i.e., c should have been executed. Therefore, matching instances can represent the difference between a trace and a model execution, which is quantitatively evaluated as the *order fitness*.

Definition 6 (Order fitness)**.** *Let $M = (L, l_0, z, A, T, \gamma)$ be a FA-model over the activity set A. Let E_ℓ be a trace from an event log $L_A = (E, C, \alpha, \beta, \tau, \succ)$. Let $I = (LI, TI, \rho)$ be an automaton instance of M and (I, μ) be a matching instance. Let $LI^s = LI - \mathsf{rng}(\mu)$ and $E^i_\ell = E_\ell - \mathsf{dom}(\mu)$. Furthermore, let $\kappa^s : A \to \mathbb{N}_{>0}$ and $\kappa^i : A \to \mathbb{N}_{>0}$ be functions that give the* skip costs *and the* insert costs *of activities, respectively. Then, the* order fitness *of (I, μ) is defined as:*

$$f_{order} = 1 - \frac{\mathsf{cost}}{\mathsf{base}},$$

where

$$\mathsf{cost} = \sum_{s' \in LI^s} \kappa^s(\gamma(\rho(s'))) + \sum_{e \in E^i_\ell} \kappa^i(\alpha(e)),$$

$$\mathsf{base} = \min_{\sigma \in \mathsf{exe}(M)} \sum_{s \in \sigma} \kappa^s(\gamma(s)) + \sum_{e \in E_\ell} \kappa^i(\alpha(e)).$$

In this definition, the set LI^s represents the set of skipped activities in the trace E_ℓ that should have been executed according to the model, and the set E^i_ℓ represents the set of inserted events in the trace

E_ℓ that should not have been executed according to the model. We call such mismatches of the model execution and the trace execution as *misalignments*. To each activity the costs representing the penalty of skipping and inserting are assigned by functions κ^s and κ^i, respectively. The value base is the sum of the minimal skip cost of model executions (regarded as all model activities are skipped) and the insert cost of the trace (regarded as all trace activities are inserted). This is the worst cost among the possible alignments between the model and the trace. The value cost take weighted sum for each misalignment, where each weight is either the skip cost or the insert cost. Therefore, the order fitness is 1 minus the ratio of the cost of misalignments to the worst cost. If there is no misalignment, the order fitness is 1, and if the model does not match the trace at all, the order fitness is 0. Therefore, the order fitness necessarily takes a value between 0 and 1.

3.2 Time Fitness

In this section, we formally define the time fitness between a TA-model and a trace. We quantitatively evaluate how much a given matching instance satisfies the time constraints specified in the model.

As in the definition of the TA-model, we only consider guards of the form $l < t < u$. For the guard g of the form $l < t < u$, we define $\underline{g} = l$ and $\overline{g} = u$.

Definition 7 (Time fitness)**.** *Let $M = (L, l_0, z, A, t, T, \gamma)$ be a TA-model over the activity set A. Let E_ℓ be a trace from an event log $L_A = (E, C, \alpha, \beta, \tau, \succ)$. Let $I = (LI, TI, \rho)$ be an automaton instance of M' (the FA-model obtained from M) and (I, μ) be a matching instance. Let $D = \mathsf{dom}(\mu) - \{\mathsf{last}(E_\ell)\}$. If D is non-empty, the* time fitness *of (I, μ) for M is defined as:*

$$f_{time} = \frac{1}{|D|}\sum_{e \in D}\frac{f(e)}{F(e)}$$

where

$$\begin{aligned} f(e) &= \overline{G(\mathsf{loc}(e), \mathsf{loc}(\mathsf{next}(e))} \\ &\quad - \underline{G(\mathsf{loc}(e), \mathsf{loc}(\mathsf{next}(e))} \\ F(e) &= \max(\tau(e), \overline{G(\mathsf{loc}(e), \mathsf{loc}(\mathsf{next}(e)))}), \\ &\quad - \min(\tau(e), \underline{G(\mathsf{loc}(e), \mathsf{loc}(\mathsf{next}(e)))}). \end{aligned}$$

If D is empty, $f_{time} = 1$.

We give an intuitive explanation of this definition. Suppose the execution of the event e in the trace corresponds to the execution of location p in the model and the guard of the corresponding transition is $l < t < u$ (note that although there are possibly multiple transitions from p, the location corresponding to the next event $\mathsf{next}(e)$ in the matching instance uniquely determines the transition). Then, the timestamp recorded in e is $\tau(e)$. If $l < \tau(e) < u$, the guard of this transition is satisfied. In this case, $f(e) = u - l$ and $F(e) = \max(\tau(e), u) - \min(\tau(e), l) = u - l$. Hence, we have $\frac{f(e)}{F(e)} = 1$, that is, the execution of e perfectly conforms to the model. On the other hand, if $\tau(e) < l$, then $F(e) = u - \tau(e)$, and thus $\frac{f(e)}{F(e)} = \frac{u-l}{u-\tau(e)} < 1$. The smaller $\tau(e)$ becomes than l, the smaller the value of $\frac{f(e)}{F(e)}$ is, which results in smaller f_{time}. For example, suppose $l = 10$ and $u = 20$. If $\tau(e) = 5$ then $\frac{20-10}{20-5} = \frac{10}{15} = \frac{2}{3}$. If $\tau(e) = 0$, then $\frac{20-10}{20-0} = \frac{10}{20} = \frac{1}{2}$. Therefore, simply speaking, the value $\frac{f(e)}{F(e)}$ is the ratio of the length of the interval to the sum of the length of the interval and the distance between $\tau(e)$ and the interval.

The average of this value for each event in the trace is the time fitness f_{time} of the matching instance.

Example 3. We consider the TA-model in Figure 1 and the trace $\langle (a,2), (b,6), (c,7), (b,9), (d,10) \rangle$. Let the matching instance $((LI, TI, \rho), \mu)$ be as follows: $LI = \{a', b', c', b'', c'', d'\}$, $TI = \{(a', b'), (b', c'), (c', b''), (b'', c''), (c'', d')\}$, $\rho(a') = s_1$, $\rho(b') = \rho(b'') = s_2$, $\rho(c') = \rho(c'') = s_3$, $\rho(d') = s_4$. $\mu((a,2)) = a'$, $\mu((b,6)) = b'$, $\mu((c,7)) = c'$, $\mu((b,9)) = b''$, $\mu((d,10)) = d'$. There is no activity corresponding to c'' in LI; the activity c is skipped once in the trace. Then, we have $D = \mathsf{dom}(\mu) - \{\mathsf{last}(E)\} = \{(a,2), (b,6), (c,7), (b,9)\}$. The value of the time fitness of this instance is:

$$\begin{aligned} &\frac{1}{4}\Bigg(\frac{\max(0,3) - \min(0,3)}{\max(2,3) - \min(2,0)} \\ &\quad + \frac{\max(1,5) - \min(1,5)}{\max(6,5) - \min(6,1)} \\ &\quad + \frac{\max(2,7) - \min(2,7)}{\max(7,7) - \min(7,2)} \\ &\quad + \frac{\max(1,5) - \min(1,5)}{\max(9,5) - \min(9,1)}\Bigg) \\ =&\frac{1}{4}(1 + 0.8 + 1 + 0.5) = 0.825 \end{aligned}$$

In the above example, the activity c is skipped in the trace, which is not taken account in the computation of the time fitness value. In general, misaligning events and activities in the matching instance are not taken into account in the computation of the time fitness value. This is because such misalignments are taken into account in the order fitness. Another reason is that we cannot give reasonable timestamp values to such misalignments due to lack of corresponding transitions. Our treatment of misalignments is similar to

Leoni et al.'s conformance checking (de Leoni et al., 2012), which compares attribute values specified in the model with recorded values in the trace.

When a timestamp takes the value precisely equal to a boundary value of a guard, the time fitness value for such event is evaluated to 1. In the above example, for the third event $(c,7)$ in the trace, its time fitness value is $\frac{\max(2,7)-\min(2,7)}{\max(7,7)-\min(7,2)} = 1$. Since the guard is $2 < t < 7$ and the timestamp is 7, this event does not logically satisfy the guard. In practice, however, the probability that the timestamp precisely takes a boundary value can be negligible. Moreover, although both cases $t = 7$ and $t = 7 + 10^{-100}$ are logically false, we can think that both cases conform to the model almost equally. Therefore, we defined that if the timestamp takes a boundary value of a guard, the corresponding fitness value is 1.

3.3 Total Fitness

The *total fitness* of a matching instance is the average of the order fitness value and the time fitness value of the instance.

Definition 8 (Total fitness)**.** *Let $M = (L,l_0,z,A,t,T,\gamma)$ be a TA-model over the activity set A, $M' = (L,l_0,z,A,T',\gamma)$ be the FA-model obtained from M. Let $L_A = (E,C,\alpha,\beta,\tau,\succ)$ be an event log and E_ℓ be a trace of a case $\ell \in C$. Let (I,μ) be a matching instance of E_ℓ and M'. Let f_{order} and f_{time} be the order fitness value and the time fitness value of (I,μ), respectively. Then the* total fitness *(or simply* fitness*) of (I,μ) is defined as:*

$$fitness = \frac{f_{order} + f_{time}}{2}$$

4 COMPUTATION OF FITNESS VALUES

The previous section defined the fitness of a matching instance. However, there are possibly many matching instances for a single trace. For example, if the process model has a loop, there is a matching instance for each number of iteration of the loop. Let us consider the model in Figure 1. There are automaton instances corresponding to execution $\langle a,b,c,d\rangle$, $\langle a,b,c,b,c,d\rangle$, $\langle a,b,c,b,c,b,c,d\rangle$, and so on. Each automaton instance yields different matching instance. When checking conformance of a trace to the model, which execution should we choose for evaluating the fitness? The answer is an *optimal* matching instance. For example, if the trace is $\langle a,b,d\rangle$, the execution of the model in Figure 1 most similar to it is $\langle a,b,c,d\rangle$. Thus, it is natural to think that the trace intended to execute this path in the model, but unfortunately a mistake happened. It is not reasonable to think that the trace intended to execute $\langle a,b,c,b,c,d\rangle$ nor $\langle a,b,c,b,c,b,c,d\rangle$, because if the trace really intended to such executions (i.e., loop iterations), at least b or c should occur more than once. Therefore, we have to find the most similar execution to the trace for evaluating the fitness, i.e. an optimal matching instance. In other words, an optimal matching instance represents the most similar execution of the model to reality (the trace recorded in the event log).

This section articulates the computation of the fitness values between a trace and a process model. For this, we need to define the notion of *optimal matching instances*. Furthermore, we show how we can find such optimal matching instances efficiently using A* algorithm.

4.1 Optimal Matching Instance

We defined the order fitness in Definition 6. Here we define the cost of a matching instance, and based on it we define *optimal matching instances*.

Definition 9 (Cost of matching instance, optimal matching instance)**.** *Let $M = (L,l_0,z,A,T,\gamma)$ be a FA-model model over the activity set A. Let $L_A = (E,C,\alpha,\beta,\tau,\succ)$ be an event log over A and E_ℓ be the trace of a case $\ell \in C$. Let E' be a prefix of E_ℓ. Furthermore, let $\kappa^s : A \to \mathbb{N}_{>0}$ and $\kappa^i : A \to \mathbb{N}_{>0}$ be functions that give the* skip costs *and the* insert costs *of activities, respectively. Then, the* cost *$\delta_n : \mathcal{J}_{E'} \to \mathbb{N}$ of a matching instance is defined by:*

$$\delta_n((I_{E'},\mu)) = \sum_{s_i \in LI^s} \kappa^s(\gamma(\rho(s_i))) + \sum_{e \in E'^i} \kappa^i(\alpha(e))$$

where $I_{E'} = (LI,TI,\rho)$, $LI^s = LI - \mathsf{rng}(\mu)$ and $E'^i = E' - \mathsf{dom}(\mu)$.

An optimal matching instance *of E' for M is a matching instance in $\mathcal{J}_{E'}$ that has the minimum cost.*

Note that the cost of a matching instance appears as cost in the definition of the order fitness $f_{order} = 1 - \frac{\mathsf{cost}}{\mathsf{base}}$ in Definition 6. Since the value base is constant (i.e., does not depend on a matching instance), optimal matching instances have the maximum value of the order fitness.

We define *the order fitness of a trace E_ℓ* to be the order fitness of an optimal matching instance in $\mathcal{J}_{E_\ell}$.

Note that there are possibly many optimal matching instances for the same trace. Consider the trace $\langle a,b,c,b,d\rangle$ for the process model in Figure 1, and assume that every activity has the same skip cost and the insert cost. Then, there are two possible optimal matching instances: one corresponding to the model execution $\langle a,b,c,d\rangle$ (the second b in the trace is inserted) and one corresponding to the model execution $\langle a,b,c,b,c,d\rangle$ (the second c in the model execution is skipped in the trace). To compute the order fitness of a trace, we need to find at least one optimal matching instance.

4.2 Search for an Optimal Matching Instance

To find an optimal matching instance for a given trace and a process model, we define the search space graph whose vertices are matching instances. For this, we first define the transition relation and the distance between matching instances.

For an automaton instance $I = (LI, TI, \rho)$ and $S \subseteq LI$, We write $I \cap S$ for the automaton instance $(LI \cap S, TI\restriction_S, \rho\restriction_S)$, where $TI\restriction_S = \{(l_1,l_2) \in TI \mid l_1 \in S \wedge l_2 \in S\}$, and $\rho\restriction_S$ is the restriction of ρ to S.

Definition 10 (Transition relation between matching instances)**.** *Let $M = (L,l_0,z,A,T,\gamma)$ be a FA-model over the activity set A. Let $L_A = (E,C,\alpha,\beta,\tau,\succ)$ be an event log over A and E_ℓ be the trace of a case $\ell \in C$. Let E_1 and E_2 be prefixes of E_ℓ. Let $I_1 = (LI_1,TI_1,\rho_1)$ and $I_2 = (LI_2,TI_2,\rho_2)$ be automaton instances of M, and let $(I_1,\mu_1) \in \mathcal{I}_{E_1}$ and $(I_2,\mu_2) \in \mathcal{I}_{E_2}$ be matching instances. Then, $(I_1,\mu_1) \blacktriangleright (I_2,\mu_2)$ if and only if one of the following hold:*

- *$E_2 = E_1 \cdot e$ and there is $s_i \in LI_2 - LI_1$ such that $LI_2 = LI_1 \cup \{s_i\}$, $\mu_2(e) = s_i$, $\forall e' \in E_1.\mu_1(e') = \mu_2(e')$, and $I_1 = I_2 \cap LI_1$. In other words, E_2 is obtained from E_1 by appending the event e, and a corresponding activity execution in the model is also added to I_2. This means that the trace execution and the model execution coincide.*
- *$E_2 = E_1$ and there is $s_i \in LI_2 - LI_1$ such that $LI_2 = LI_1 \cup \{s_i\}$, $\forall e' \in E_2.\mu_1(e') = \mu_2(e')$, and $I_1 = I_2 \cap LI_1$. In other words, an activity corresponding to s_i is added to I_2. This means that an activity of the model is skipped in the trace E_2.*
- *$E_1 = E_2 \cdot e$ and $\forall e' \in E_1.\mu_1(e') = \mu_2(e')$ and $\mu_2(e)$ is undefined. In other words, E_2 is obtained from E_1 by appending the event e, but no corresponding activity is added to I_2. This means that an event is inserted to the trace E_2.*

If $(I_1,\mu_1) \blacktriangleright (I_2,\mu_2)$, then (I_2,μ_2) is obtained by executing the same activity in the trace and the model, or only the model executes an activity (and thus the automaton instance grows) and the trace does not change (skip), or only the trace executes an activity (insert) and the automaton instance does not change.

We define the distance for such transitions between matching instances (technically, it is defined for every pair of matching instances).

Definition 11 (Distance between matching instances)**.** *Let $M = (L,l_0,z,A,T,\gamma)$ be a FA-model over the activity set A. Let $L_A = (E,C,\alpha,\beta,\tau,\succ)$ be an event log over A and E_ℓ be the trace of a case $\ell \in C$. Let E_1 and E_2 be prefixes of E_ℓ. Let I_1 and I_2 be automaton instances of M, and let $(I_1,\mu_1) \in \mathcal{I}_{E_1}$ and $(I_2,\mu_2) \in \mathcal{I}_{E_2}$ be matching instances. We define the distance $\delta((I_1,\mu_1),(I_2,\mu_2))$ between (I_1,μ_1) and (I_2,μ_2) as:*

$$\delta((I_1,\mu_1),(I_2,\mu_2)) = \delta_n((I_2,\mu_2)) - \delta_n((I_1,\mu_1)) + |E_2| - |E_1|$$

Now we define the search space graph of matching instances based on the above definitions. Here, we write $\sigma < \sigma'$ to mean that the sequence σ is a prefix of the sequence σ'.

Definition 12 (Search space graph)**.** *Let $M = (L,l_0,z,A,T,\gamma)$ be a FA-model over the activity set A. Let $L_A = (E,C,\alpha,\beta,\tau,\succ)$ be an event log over A and E_ℓ be the trace of a case $\ell \in C$. The* search space graph *of matching instances of E_ℓ for M is $G = (V,W,\zeta)$ where*

- *$V = \bigcup_{E' < E_\ell} \mathcal{I}_{E'}$ is the set of vertices,*
- *$W = \{(v_1,v_2) \in V \times V \mid v_1 \blacktriangleright v_2\}$ is the set of edges, and*
- *ζ is a weight of edges, and defined as $\zeta((v_1,v_2)) = \delta(v_1,v_2)$ for every $(v_1,v_2) \in W$.*

The sequence $v_1,v_2,\ldots,v_n$ is called a path *of G if $v_i \blacktriangleright v_{i+1}$ for all $1 \leq i < n$. For a path $\pi = v_1,v_2,\ldots,v_n$ of G, the* length *of π is defined as:*

$$\mathsf{len}(\pi) = \sum_{i=1}^{n} \delta(v_1,v_2).$$

Then, the optimal matching instance search problem is defined as:

Definition 13 (Optimal matching instance search problem)**.** *Let $G = (V,W,\zeta)$ be the search space graph of E_ℓ for $M = (L,l_0,z,A,T,\gamma)$. Let $v_{src} = ((\emptyset,\emptyset,\rho_0),\mu_0) \in \mathcal{I}_{\langle\rangle}$ (here ρ_0 and μ_0 are the empty maps) be the initial vertex. Let $V_{trg} = \{((LI,TI,\rho),\mu) \in \mathcal{I}_{E_\ell} \mid \rho(\mathsf{last}((LI,TI,\rho))) = z\}$ be the set of target vertices, where $\mathsf{last}((LI,TI,\rho))$ is the location instance l such that $\mathsf{succ}(l) = \emptyset$ in (LI,TI,ρ). Then, the* optimal matching instance search problem *is a problem to find the shortest path $v_{src},\ldots,v_{goal}$ where $v_{goal} \in V_{trg}$.*

Remark 1. Adriansyah et al.'s definition of the target vertices (Adriansyah et al., 2011b) includes automaton instances that do not end with the final location z. In this case, if a trace partially executes the model correctly, its order fitness value is 1. On the other hand, our target vertices are only those that execute the model till the final location. In our definition, the order fitness value of a trace that correctly executes the model but not to the end is less than 1. Since traces in an event log are regarded as executions of the process from the initial activity to the final activity, our definition is more natural.

Theorem 1. *If $v_{src}, ..., v_{goal}$ be the shortest path in the search space graph of Definition 13, then v_{goal} is an optimal matching instance.*

Proof. Let Π be the set of all paths from v_{src} to some vertex in V_{trg} in the search space graph. Let $\pi \in \Pi$ be a path $\langle (I_1,\mu_1), ..., (I_n,\mu_n) \rangle$, where $(I_i,\mu_i) \in \mathcal{I}_{E_i}$, $E_i < E_\ell$ for all $i < n$ and $E_n = E_\ell$. Since $\delta((I_i,\mu_i),(I_{i+1},\mu_{i+1})) = \delta_n((I_{i+1},\mu_{i+1})) - \delta_n((I_i,\mu_i)) + |E_{i+1}| - |E_i|$, the length of π is

$$\begin{aligned}\mathsf{len}(\pi) = &\sum_{k=1}^{n-1} \delta((I_k,\mu_k),(I_{k+1},\mu_{k+1})) \\ =&\delta_n((I_n,\mu_n)) - \delta_n((I_1,\mu_1)) + |E_n| - |E_1|.\end{aligned}$$

Here we have $\delta((I_1,\mu_1)) = 0 = |E_1|$ since $(I_1,\mu_1) = v_{src}$. Therefore, $\mathsf{len}(\pi) = \delta_n((I_n,\mu_n)) + |E_n| = \delta_n(\mathsf{last}(\pi)) + |E_\ell|$. Hence,

$$\begin{aligned}\arg\min_{\pi\in\Pi} \mathsf{len}(\pi) =& \arg\min_{\pi\in\Pi}(\delta_n(\mathsf{last}(\pi)) + |E_\ell|) \\ =& \arg\min_{\pi\in\Pi} \delta_n(\mathsf{last}(\pi)),\end{aligned}$$

because $|E_\ell|$ is a constant. Assume that π is a matching instance that gives the minimum $\mathsf{len}(\pi)$, that is, $\pi = \langle (I_1,\mu_1), ..., (I_n,\mu_n) \rangle$ is the shortest path. Clearly, for all $v \in V_{trg}$, there is a path from v_{src} in the search space graph. Therefore, $\min_{\pi\in\Pi} \delta_n(\mathsf{last}(\pi)) = \min_{v\in V_{trg}} \delta_n(v)$. Then, from the above formula, its last matching instance $\mathsf{last}(\pi) = v_{goal}$ takes the minimum cost $\delta_n(v_{goal})$ among the vertices in V_{trg}, which is equal to the value of cost in the definition of the order fitness (Definition 6). Therefore, v_{goal} takes the maximum order fitness, i.e., it is an optimal matching instance. □

4.3 Search Algorithm

If we are only interested in the order fitness, it is sufficient to find one optimal matching instance, because every optimal matching instance has the same order fitness value. In this research, however, we are also interested in the time fitness, and the total fitness value is the average of the order fitness value and the time fitness value. Since each optimal instance may have different time fitness values, each optimal instance has a different total fitness value. As we mentioned in Section 3, when checking conformance, we need to find the model execution most similar to the trace (reality). Therefore, to find the best total fitness value, we need to find all possible optimal matching instances.

Remark 2. The matching instance that has the best total fitness value among all matching instances might not be optimal. In our definition, however, the order fitness of a *trace* is defined as the order fitness of an *optimal matching instance*. Therefore, even if there may be a matching instance that is not optimal but has a better total fitness value (due to the better time fitness value), it is not regarded as the fitness value of the trace.

In general, the set of vertices of a search space graph is infinite. For example, if a process model has a loop, we can iterate the loop arbitrarily many times. That is, we can skip the activities in the loop as many times as possible. However, since we assume that the skip cost (given by the function κ^s) of activities are positive, such arbitrary number of iteration just deteriorate the order fitness. Thus, we can find an optimal matching instance in finite steps by a suitable search algorithm.

As we mentioned above, we need to find all possible optimal matching instances when we are to find the best fitness value for a given trace and a process model. Therefore, we need to perform the exhaustive search. We can do this by breadth-first search (BFS) algorithm. To use BFS in finding all possible optimal matching instances, we first do usual BFS on the search space graph and find one optimal matching instance. Then, we find the minimum length of the path to the target nodes. Using this value, we exhaustively search all possible paths shorter than or equal to this value (recall that the length of the path corresponds to the cost of the matching instance of the final vertex of the path). We call this method the *exhaustive search*.

Clearly, the exhaustive search is not efficient. As the size of a process model or the length of a trace grows, computation of the fitness value will eventually be intractable. Therefore, we utilize A* algorithm as a more efficient algorithm, which is also used in alignment-based conformance checking (Adriansyah et al., 2011b). This algorithm finds an arbitrary optimal matching instance and compute the fitness value for it. Of course, it may not give the best total fitness value, that is, it approximately computes the best total fitness value for a given trace and a process model.

To utilize A* algorithm in searching an optimal matching instance, we need to define a suitable heuristic function. For a search space graph (V, W, ζ)

of Definition 12, we define the heuristic function $h : V \to \mathbb{N}$ as $h(v) = |E_\ell| - |E'|$ for $v \in \mathcal{J}_{E'}$. $h(v)$ underestimates the distance from the vertex v to some vertex in V_{trg}. That is, for all $v_{trg} \in V_{trg}$, $h(v) \leq \zeta(v, v_{trg})$ holds (Adriansyah et al., 2011b). With this h, we define the evaluation function f as $f(v) = g(v) + h(v)$, where $g(v)$ returns the minimum cost from v_{src} to v, which is the current position in the search.

To ensure that A* algorithm finds one of the target vertices, we need to prove the following:

1. At least one target vertex is reachable,
2. The heuristic function actually gives an underestimation of the distance to the target vertices, and
3. The evaluation function is monotonously increasing.

These claims are true as in (Adriansyah et al., 2011b).

5 EXPERIMENTAL RESULTS AND DISCUSSION

This section shows experimental results of the proposed conformance checking algorithms for some simple TA-models and traces. We give the comparison of the exhaustive search algorithm and A* algorithm with respect to the obtained matching instance, the order and time fitness values, and the number of the searched vertices and the computation time.

5.1 Experiment

We implemented both algorithm in Python. In our implementation, a timed automaton is given in .xml format used in UPPAAL[1] (Larsen et al., 1997), a model checker for timed automata, and a trace is given as a text file consisting of a sequence of pairs of alphabets (activities) and numerals (timestamps). For simplicity, the skip cost and insert cost of every activity are set to 1. We performed the experiment on the computer with Core i9-13900 32GB RAM.

We gave three TA-models in this experiment.

The first model contains only one loop (Figure 2). For this model, we checked the conformance against the trace $\langle(a,10),(b,15),(c,25),\ (b,15),(d,0)\rangle$. The result is presented in Table 1.

The second model has a loop that contains a branch (Figure 3). For this model, we checked the conformance against the trace $\langle(a,5),(b,10),(c,5),$ $(e,15),(b,10),(e,15)\rangle$. The result is presented in Table 2.

[1]https://uppaal.org/

The third model has a branch followed by a loop (Figure 4). For this model, we checked the conformance against the trace $\langle(a,10),(d,15),(e,10),$ $(d,10),(f,0)\rangle$. The result is presented in Table 3. Note that in this trace the activities inside the branch (b and c) are skipped.

5.2 Discussion

The results of the three experiments show that the order fitness is the same for each optimal instance found by the exhaustive search algorithm, but the time fitness is different for each optimal instance. In other words, the experiments confirmed that even the optimal matching instances have different time fitness values. Since A* algorithm approximately finds the total fitness value, it fails to find the best values in the first and third experiments, as shown in Table 1 and 3. Therefore, we confirmed that A* algorithm does not necessarily find the best fitness value.

We analyze the third experiment a bit deeper. For this model and the trace, the order fitness value is the same whether the activity b or c is interpreted as skipped. However, when we consider the timestamp, since the timestamp of a is 10, if the activity c is interpreted as skipped, this does not give penalty by the guard on the transition from a to c. On the other hand, interpreting b as skipped gives penalty by the guard on the transition from a to b. Hence, the interpretation of c as skipped gives better fitness value. We can see this fact from Table 3. In the optimal instances found by the exhaustive search, the c-skipped interpretation (the second and the forth instances) give better time fitness values than the b-skipped interpretation (the first and the third instances).

The result of conformance checking can be interpreted in two ways: either the process model is normative and the trace (what happened in reality) is wrong, or the trace is valid and the process model is incomplete. In the former interpretation, the execution path in the process model with the best fitness value is the intended execution of the trace (reality). The latter interpretation would suggest that the path with the best fitness value gives a hint to improve the process model. In either case, the optimal path, which was difficult to identify only by the order fitness, can be identified with our proposed conformance checking method, enabling effective analysis of identifying errors in the trace or parts in the model deviating from reality.

As for the computation time, A* algorithm is about 15-200 times faster than the exhaustive search algorithm. Since real business processes are generally much more complex than the models used in these

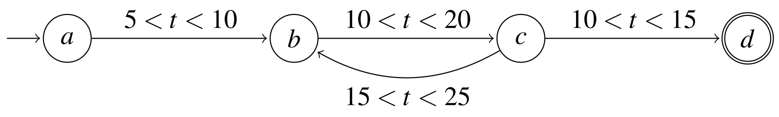

Figure 2: The first TA-model. The location names designate the activity names.

Table 1: The result of conformance checking for the model in Figure 2 and the trace $\langle(a,10),(b,15),(c,25),(b,15),(d,0)\rangle$.

Algorithm	Optimal instance	f_{order}	f_{time}	*fitness*	# of searched vertices	Time
A*	$\langle a,b,c,d\rangle$	0.889	0.778	0.883	40	5.174×10^{-4}
Exhaustive	$\langle a,b,c,d\rangle$	0.889	0.778	0.833	355	8.066×10^{-3}
	$\langle a,b,c,b,c,d\rangle$	0.889	1.000	**0.944**		

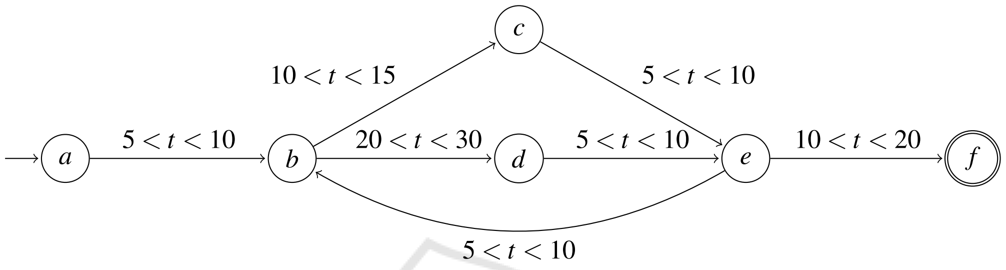

Figure 3: The second TA-model. The location names designate the activity names.

Table 2: The result of conformance checking for the model in Figure 3 and the trace $\langle(a,5),(b,10),(c,5),(e,15),(b,10),(e,15)\rangle$.

Algorithm	Optimal instance	f_{order}	f_{time}	*fitness*	# of searched vertices	Time
A*	$\langle a,b,c,e,b,c,e,f\rangle$	0.818	0.917	**0.867**	147	2.562×10^{-3}
Exhaustive	$\langle a,b,c,e,b,c,e,f\rangle$	0.818	0.917	**0.867**	3478	5.435×10^{-1}
	$\langle a,b,c,e,b,d,e,f\rangle$	0.818	0.833	0.826		

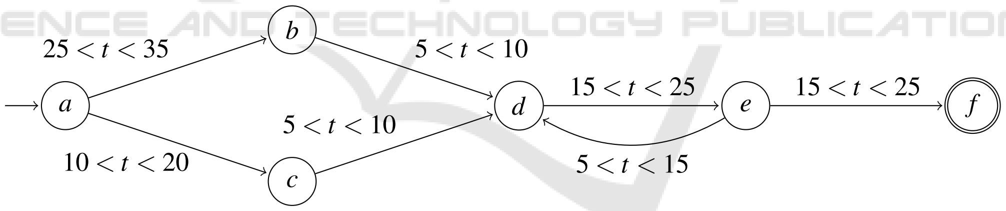

Figure 4: The third TA-model. The location names designate the activity names.

Table 3: The result of conformance checking for the model in Figure 4 and the trace $\langle(a,10),(d,15),(e,10),(d,10),(f,0)\rangle$.

Algorithm	Optimal instance	f_{order}	f_{time}	*fitness*	# of searched vertices	Time
A*	$\langle a,b,d,e,f\rangle$	0.800	0.689	0.744	99	1.428×10^{-3}
Exhaustive	$\langle a,b,d,e,f\rangle$	0.800	0.689	0.744	988	5.105×10^{-2}
	$\langle a,c,d,e,f\rangle$	0.800	0.889	0.845		
	$\langle a,b,d,e,d,e,f\rangle$	0.800	0.767	0.783		
	$\langle a,c,d,e,d,e,f\rangle$	0.800	0.917	**0.858**		

experiments, we need an efficient algorithm like A* that also takes the time fitness into account to find the best matching instance. Since the problem of finding optimal matching instances is known to be NP-hard (de Leoni and van der Aalst, 2013), some approximative algorithm is desired even for computing the order fitness.

6 RELATED WORK

As we mentioned in Section 1, most conformance checking method only consider the order of activities. Only a few researches considering attributes or timestamps are known.

Leoni et al. proposed a method for conformance checking considering attributes of events (de Leoni et al., 2012). In their research, a process model has assignments of constants to variables (attributes), and when activities are executed, the value of the variables are updated according to the assignments. The alignment of a trace to a model is defined as usual. The fitness is evaluated based on the number of activities that are not misaligned but their attribute values do not coincide with each other. That is, their method only uses the number of discrepancies of attribute values. On the other hand, our proposed method evaluate the degree of discrepancy, not just the number of discrepancies.

Leoni et al.'s approach is later extended to handle infinite data domains and customizable balance between the control-flow perspective and the data perspective (Mannhardt et al., 2016). Our approach first considers the control-flow perspective, then evaluate the data perspective. Therefore, one possible future direction is to enable such balancing in our proposed method.

Giacomo et al. proposed a method for aligning timed traces to a formula of Metric Temporal Logic (MTL_f) (Giacomo et al., 2021). For a given timed trace and MTL_f formula where the trace does not satisfy the formula, their method finds the trace obtained by editing the original trace with the lowest editing cost to satisfy the formula. Their research does not focus on computing the fitness values. If one defines it, it is necessarily based on the number of deletion and insertion. Thus, the fitness essentially equal to the order fitness. Furthermore, their method demands NONELEMENTARY time for computation, making it difficult to put into practice.

Grohs and Rehse (Grohs and Rehse, 2023) proposed a method for correlating process conformance with trace attributes in order to explain the potential cause of deviations. They creates a regression tree to find which attributes contribute to lower and higher fitness values. The purpose of utilizing attribute values is, thus, different from our approach.

7 CONCLUSIONS

In this paper, we proposed a conformance checking on timed automaton process models, which takes time attributes of events into account. Our proposed method first finds an optimal matching instance (i.e., alignment between a model and a trace). In the obtained matching instance, we evaluate how much each timestamp deviates from the interval represented by the corresponding guard. Therefore, the fitness is given by the order fitness, which is evaluated based on the number of misalignments, and the time fitness, which is evaluated based on how much the timestamps satisfy the guards.

We implemented our method using the exhaustive search algorithm and A* algorithm. The exhaustive search algorithm finds the best fitness value at the cost of performance. A* algorithm runs faster than the exhaustive search but finds an arbitrary optimal matching instance to compute the fitness value, thus may not give the best fitness value.

By considering time attributes, we demonstrated that we can uniquely determine the most similar execution of a model to a given trace, which is indistinguishable from other optimal executions in terms of the order fitness. Thus, our proposed method is useful for identifying an error in the trace or in the process model.

There are several future directions for extending our method. In this paper, we only consider the time attribute. It is natural to include other numerical attributes such as the amount of a resource consumed at the event (in a manufacturing process), or the price of a quote (in a negotiation process). By including such attributes, we can evaluate more precise fitness of traces and event logs.

Another direction is to extend our guard expressions; we only have the form $d_1 < t < d_2$. In general, a guard is given as a conjunction of such intervals. Therefore, we need to modify Definition 6 to handle such formulas.

We also need to develop an efficient search algorithm in computing the fitness value considering the time fitness.

Finally, we are to carry out extensive experiments using real-life event logs and process models. The problem is, however, the lack of timed automaton process models in real-life examples. One attempt is to discover a skeleton of the model using well-established model discovery algorithm, then manually modify and annotate necessary information to it (Ito et al., 2021). It is strongly desired to develop some process discovery algorithm that constructs appropriate timed automaton process models from event logs.

ACKNOWLEDGMENTS

This work was supported by JSPS KAKENHI Grant Number JP21K11756. The authors thank Atsushi Takayanagi for helpful comments.

REFERENCES

Adriansyah, A. (2014). *Aligning Observed and Modeled Behavior*. PhD thesis, Technische Universiteit Eindhoven.

Adriansyah, A., Sidorova, N., and van Dongen, B. F. (2011a). Cost-based fitness in conformance checking. In *11th International Conference on Application of Concurrency to System Design, ACSD 2011, Newcastle Upon Tyne, UK, June 20-24, 2011*, pages 57–66.

Adriansyah, A., van der Aalst, W. M. P., and van Dongen, B. F. (2011b). Conformance checking using cost-based fitness analysis. In *15th IEEE International Enterprise Computing Conference, EDOC 2011, Helsinki, Finland, August 29-Spetember 2, 2011*, pages 55–64.

de Leoni, M. and van der Aalst, W. M. P. (2013). Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming. In Daniel, F., Wang, J., and Weber, B., editors, *Business Process Management - 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings*, volume 8094 of *Lecture Notes in Computer Science*, pages 113–129. Springer.

de Leoni, M., van der Aalst, W. M. P., and van Dongen, B. F. (2012). Data- and resource-aware conformance checking of business processes. In Abramowicz, W., Kriksciuniene, D., and Sakalauskas, V., editors, *Business Information Systems - 15th International Conference, BIS 2012, Vilnius, Lithuania, May 21-23, 2012. Proceedings*, volume 117 of *Lecture Notes in Business Information Processing*, pages 48–59. Springer.

Giacomo, G. D., Murano, A., Patrizi, F., and Perelli, G. (2021). Timed trace alignment with metric temporal logic over finite traces. In Bienvenu, M., Lakemeyer, G., and Erdem, E., editors, *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online event, November 3-12, 2021*, pages 227–236.

Grohs, M. and Rehse, J.-R. (2023). Attribute-based conformance diagnosis: Correlating trace attributes with process conformance. In Montali, M., Senderovich, A., and Weidlich, M., editors, *Process Mining Workshops*, pages 203–215, Cham. Springer Nature Switzerland.

Ito, S., Vymětal, D., and Šperka, R. (2021). Process mining approach to formal business process modelling and verification: a case study. *Journal of Modelling in Management*, 16(2):602–622.

Ito, S., Vymětal, D., Šperka, R., and Halaška, M. (2018). Process mining of a multi-agent business simulator. *Computational & Mathematical Organization Theory*, 24(4):500–531.

Larsen, K. G., Petterson, P., and Yi, W. (1997). UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1/2):134–152.

Mannhardt, F., de Leoni, M., Reijers, H. A., and van der Aalst, W. M. P. (2016). Balanced multi-perspective checking of process conformance. *Computing*, 98(4):407–437.

Polyvyanyy, A., Weidlich, M., Conforti, R., Rosa, M. L., and ter Hofstede, A. H. (2014). The 4c spectrum of fundamental behavioral relations for concurrent systems. In *32th International Conference in Application and Theory of Petri Nets and Concurrency, PETRI NETS 2014, Tunis, Tunisia, June 23-27, 2014. Volume 8489 of Lecture Notes in Computer Science*, pages 210–232. Springer.

Rozinat, A. (2010). *Process Mining Conformance and Extension*. PhD thesis, Technische Universiteit Eindhoven.

van der Aalst, W. M. (2016). *Process Mining: Data Science in Action*. Springer.

van der Aalst, W. M. P., Adriansyah, A., and van Dongen, B. F. (2012). Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, 2(2):182–192.

vanden Broucke, S. K. L. M., Munoz-Gama, J., Carmona, J., Baesens, B., and Vantheienen, J. (2014a). Event-based real-time decomposed conformance analysis. In *On the Move to Meaningful Internet Systems, OTM 2014, Amantea, Italy, October 27-31, 2014*, pages 345–363.

vanden Broucke, S. K. L. M., Weerdt, J. D., Vantheienen, J., and Baesens, B. (2014b). Determining process model precision and generalization with weighted artificial negative events. *IEEE Trans. Knowl. Data Eng.*, 26(8):1877–1889.

Weidlich, M., Polyvyanyy, A., Desai, N., Mendling, J., and Weske, M. (2011a). Process compliance analysis based on behavioural profiles. *Inf. Syst.*, 36(7):1009–1025.

Weidlich, M., Polyvyanyy, A., Mendling, J., and Weske, M. (2011b). Causal behavioural profiles – Efficient computation, applications, and evaluation. *Fundam. Inform.*, 113(3–4):399–435.

Zha, H., Wang, J., Wen, L., Wang, C., and Sun, J. (2010). A workflow net similarity measure based on transition adjacency relations. *Computers in Industry*, 61(5):463–471.