Timed Alignments

Thomas Chatain
LMF, ENS Paris-Saclay,
CNRS, Université Paris-Saclay, Inria
Gif-sur-Yvette, France
thomas.chatain@ens-paris-saclay.fr
ORCID 0000-0002-1470-5074

Neha Rino
LMF, ENS Paris-Saclay,
CNRS, Université Paris-Saclay, Inria
Gif-sur-Yvette, France
neha.rino@ens-paris-saclay.fr

***Abstract*—The subject of this paper is to study conformance checking for timed models, that is, process models that consider both the sequence of events in a process as well as the timestamps at which each event is recorded. Time-aware process mining is a growing subfield of research, and as tools that seek to discover timing related properties in processes develop, so does the need for conformance checking techniques that can tackle time constraints and provide insightful quality measures for time-aware process models. In particular, one of the most useful conformance artefacts is the alignment, that is, finding the minimal changes necessary to correct a new observation to conform to a process model. In this paper, we set our problem of timed alignment and solve two cases each corresponding to a different metric over time processes.**

***Index Terms*—Conformance checking, Alignments, Timestamps, Time Petri nets**

I. Introduction

Process mining studies vast systems through their event logs, and seeks to extract meaningful ways to model the underlying patterns or processes that govern the behaviour of the system in order to better understand it, or predict it's future behaviour [1]. Once such a process model is obtained, through machine learning or otherwise, it is natural to ask how one is sure the obtained model is a reasonable approximation of the system's behaviour, especially given the lack of explainability in the domain of machine learning. Conformance checking is the art of judging the performance of a process model by relating the modelled behaviour and the observed behaviour of a process, without depending on the origin of the model [2]. Observed behaviour comes in the form of traces in an event log, a sequence of events that occur during the functioning of the system, while process models are blueprints that describe what the underlying process of the system is expected to look like. Measures of how well a model reflects system behaviour include fitness, precision, generalisation, and simplicity. We often do not want the model to precisely generate all possible future system behaviour, but neither should it simply regurgitate the event log and accept no new system behaviours. Hence, what is often sought is a process model that can, up to some small error factor, approximate any reasonable future system behaviour.

In Arya Adriansyah's seminal thesis [3], we obtain the notion of an alignment, that is, the minimal series of corrections needed to transform an observed trace into the execution of the process model that most closely mimics it. It is often seen as an execution of the synchronised product of the process model and a *trace model* i.e. a simple model that captures exactly one event log trace. This gives us a series of edits, usually insertions or deletions, that transform the observed trace into a process trace. Alignments thereby pinpoint exactly where inevitable deviations from expected behaviour occur, and the more distant the aligning word of a model is from its observed trace, the worse the model is at reflecting real system behaviour.

Process models can be represented using a variety of formal objects, including Petri nets. Assuming the event logs are a list of words over a finite alphabet (the set of possible discrete events), the problem of calculating the alignment has been extensively studied [3] [4]. The notion of distance used on these words used is usually either Hamming distance or Levenshtein's edit distance. It is natural to want to study explicitly timed systems, as by considering events along with their timestamps when mining processes, we can glean information about the minimum delay between two events, the maximum duration the system takes to converge upon a state, or check deadlines, all of which are highly relevant in real world applications [5] [6] [7]. In addition, one may want to predict the timestamps of processes [8]. Time-aware process mining seeks to study both what the underlying processes govern system behaviour are, and what time constraints they obey [9] [10] [11]. In the process mining community, there are ways to use existing process model notation in order to denote time constraints. BPMN 2.0 comes equipped with *timer events* and can record absolute, relative, and cyclical time constraints. For our purposes, we use time Petri nets, that is, Petri nets augmented with the ability to record and check the duration it takes to fire a transition once enabled, using which one can impose certain constraints on the relationships between

Neha Rino was funded by the International Master's Scholarships Program IDEX of Université Paris-Saclay.

the timestamps of different events.

As time-aware process mining grows popular, new quality measures and conformance checking techniques must be developed that are sensitive to temporal constraints, but so far in the study of alignments as a conformance checking artefact, we notice that the process model used is never time-aware. For this, one needs to define distance functions that can meaningfully compare and separate different time processes. This paper seeks to provide a framework by which to do the very same, and set and solve the alignment problem for time-aware processes.

In this paper, we start in Section III by formally setting the alignment problem and, proposing three different distance functions over timed words. In Section IV-B we look at aligning under d_t, and give a quadratic time alignment algorithm in the case of a structurally restricted class of models (sequential causal processes). For all other types of models, in Section IV-A we present an encoding as a linear programming problem, albeit with slower worst-case time complexity. Finally in section V we study d_θ, which utilises the structure of the process model in its calculations, and we present a straightforward algorithm for solving the alignment in this case, whose time complexity is linear in the size of the causal process and the transition set of the model. The second setting, and the notion of delay edits that we proposed, provide new insight into the nature of time processes and their properties.

The proofs of the lemmas and theorems are available in [12].

II. Preliminaries

We represent events as pairs (a, t) where $a \in \Sigma$ is the name of the action, and t denotes the time at which said action was taken.

Definition 1. *A timed trace is a sequence $\gamma \in (\Sigma \times \mathbb{R}^+)^*$ of timed events, seen as a timed word.*

We will often ignore the untimed parts of timed words, i.e., the projection onto Σ^*, leaving just the timestamps, a sequence belonging to $\mathbb{R}^{+*}$. The timed process model we use here is a labeled time Petri net.

Definition 2 (Labelled Time Petri Net)**.** *A labeled time Petri net (or TPN) is a tuple $N = (P, T, F, SI, \Sigma, \lambda, M_0, M_f)$, where P, T are disjoint sets of places and transitions respectively, $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, $SI : T \to \mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \{\infty\})$ is the static interval function, where $SI(t) = (Eft(t), Lft(t))$ such that Eft stands for earliest firing time, and Lft for latest firing time, $\lambda : T \to \Sigma$ is the labeling function, labeling transitions with actions from the action set Σ, and $M_0, M_f : P \to \mathbb{N}$ are the initial and final markings.*

Given a transition $t \in T$ we define the pre-set of t as ${}^\bullet t = \{p \in P | (p, t) \in F\}$ and its post-set similarly is defined as $t^\bullet = \{p \in P | (t, p) \in F\}$, (the presets and post-sets of places are defined similarly). A transition t of a TPN is *enabled* at marking M iff $\forall p \in {}^\bullet t : M(p) > 0$. The set of all enabled transitions at a marking M is denoted by *Enabled(M)*.

A *state* of a TPN $N = (P, T, F, SI, \Sigma, \lambda, M_0, M_f)$ is a pair $S = (M, I)$, where M is a marking of N and $I : Enabled(M) \to \mathbb{R}^+$ is called the *clock function*. The initial state is $(M_0, \mathbf{0})$, where $\mathbf{0}$ is the zero function.

A transition t can fire from state $S = (M, I)$ after delay $\theta \in \mathbb{R}^+$ iff t is enabled at M, and $I(t) + \theta \in [Eft(t), Lft(t)]$ and for every other t' enabled at M, $I(t') + \theta \leq Lft(t')$. This firing is denoted $(M, I)[t\rangle(M'', I')$ with the new state (M'', I') defined as follows:

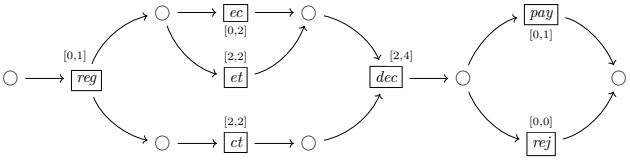
$$M'' = M' \cup t^\bullet \text{ where } M' = M \setminus {}^\bullet t$$

$$I'(t) = \begin{cases} I(t) + \theta & \text{If } t \in Enabled(M'') \cap Enabled(M') \\ 0 & \text{If } t \in Enabled(M'') \setminus Enabled(M') \end{cases}$$

A valid execution of the model begins at the initial marking, fires a sequence of transitions and reaches M_f, with any clock function I.

Definition 3 (Language $\mathcal{L}(N)$ of a TPN)**.** *A word $w = (a_0, a_1, \dots a_n) \in \Sigma^*$ is in the language of the labeled TPN $\mathcal{L}(N)$ if there is a firable sequence of pairs, of transitions and their timestamps, $(t_0, t_1 \dots t_n) \in (T, \mathbb{R}^+)^*$ such that $(\lambda(t_0), \lambda(t_1), \dots, \lambda(t_n)) = w$ and they transform the initial marking into the final one, that is, for some clock function I on M_f, $(M_0, \mathbf{0})[t_0, t_1, \dots t_n\rangle(M_f, I)$.*

Example 1. *Consider the following example (adapted from [11]) of a TPN N, modelling an airline refund office:*



The action labels are as follows : Register request (reg), Examine thoroughly (et), Examine casually (ec), Check ticket (ct), Decide (dec), Pay (pay), and Reject (rej).

The firing sequence $(reg, 1)(ec, 2)(ct, 3)(dec, 7)(rej, 7)$ is a valid execution of N. The initial marking only has reg enabled, and registering the request at time 1 updates the marking by removing the token from reg's preplace and filling its two post-places, thereby enabling ec, et, and ct. Now ec fires at 2, satisfying the constraint, filling one preplace of dec. Then ct fires, having been enabled for 2 units of time. Now, finally, dec is enabled, and fires after 4 units of having been enabled, and rej is fired immediately after.

Now we will want to build some vocabulary to help us talk about individual executions of timed words on TPNs, that take the form of branching paths.

Definition 4 (Causal Net)**.** *A causal net $CN = (B, E, G)$ is a finitary, acyclic net where $\forall b \in B : |b^\bullet| \leq 1 \wedge |{}^\bullet b| \leq 1$. B (conditions) and E (events) are the vertices, and G denotes the edges.*

Hence, multiple elements of B or E map to the same element in P or T respectively, a notion that is expressed via the map p defined below.

Definition 5 (Homomorphism)**.** *Let N be a TPN with place set P and transition set T, and $CN = (B, E, G)$ be a causal net. A mapping $p : B \cup E \to P \cup T$ is a homomorphism if $p(B) \subseteq P$, $p(E) \subseteq T$ and $\forall e \in E$ the restriction of p to ${}^\bullet e$ is a bijection between ${}^\bullet e$ and ${}^\bullet p(e)$ and the restriction of p to $e^\bullet$ is a bijection between $e^\bullet$ and $p(e)^\bullet$ and the restriction of p to $Min(CN)$ is a bijection between $Min(CN)$ and M_0, where $Min(CN)$ is the set of vertices of CN that are minimal under the G pre-ordering.*

Definition 6 (Causal Process)**.** *A causal process of a TPN is a pair (CN, p) where CN is a causal net and p is a homomorphism from CN to TPN.*

Using p, elements of CN are identified with their corresponding *originals* in the TPN, and as a result, any causal process corresponds uniquely to an untimed run on the untimed version of a given TPN.

Definition 7 (Timing Function)**.** *A timing function $\tau : E \to \mathbb{R}$ is a function from events of a causal process into time values.*

III. The Alignment Problem in Timed Settings

The alignment problem, given a trace from an event log and a process model, involves finding a valid execution of the model that is closest to the trace under some metric.

Definition 8 (The General Alignment Problem)**.** *Given a process model N denoted by a TPN and a timed trace σ we wish to find a timed word $\gamma \in \mathcal{L}(N)$ such that $d(\sigma, \gamma) = \min_{x \in \mathcal{L}(N)} d(\sigma, x)$ for some distance function d on timed words.*

In the untimed setting, this is viewed as a problem of minimizing cost over a series of edit moves, either insertions (a model move) or deletions (a trace move). When aligning timed words, clearly there are two crucial aspects to the problem, which are, how one deals with deviations in the action labels themselves (the action labels), versus how one deals with deviations in just the timing properties of the word (the timestamps). The alignment problem has been extensively studied for the untimed case [3] [4], but the timed setting is more complex.

Example 2. *Consider the process model of Example 1 and an observed trace $(reg, 1) \cdot (ec, 2) \cdot (ec, 2) \cdot (ct, 3) \cdot (dec, 8) \cdot (rej, 8)$. Clearly, one shouldn't examine casually twice so the ec is extraneous, and also, the dec event fires too late, it should have waited at most four time units after the firing of event ct. Secondly, consider an observed trace of the form $(reg, 1) \cdot (ec, 2) \cdot (ct, 3) \cdot (dec, 7) \cdot (rej, 8)$. The control flow run itself is valid, but with the timestamps, we notice that the last transition is invalid. We could either edit the rej action to a pay one and leave the timestamp alone, or, edit the timestamp to get $(rej, 7)$. Timestamps are attached to the letters they belong to, so in a sense, firing event rej at time 7 as compared to firing event pay at time 8 are not easily comparable, as the time constraints on different events are often different. In this manner, it is not clear how meaningful it would be to define edit moves that transform timestamps that are attached to different tasks.*

This is why, we first restrict our attention to the case where the untimed part of σ does match the model, that is to say, if $\pi_1(\sigma)$ has a valid causal process (CN, p) over N. Clearly this case of the problem needs to be solved if the general timed alignment problem is to be solved, and we argue that this case is interesting and complex in its own right.

This ensures that we can compare the timing constraints on analogous parts of the process, and hence reason meaningfully about timing deviances from the model. This gives us the following problem :

Definition 9 (The Purely Timed Alignment Problem)**.** *Given a process model N denoted by a TPN and a timed trace w, with timing function $\pi_2(w) = \sigma$ and a valid causal process of the untimed part $\pi_1(w)$, we wish to find a valid timing function γ such that $d(\sigma, \gamma) = \min_{x \in \mathcal{L}(N)} d(\sigma, x)$for some distance function d on timing functions.*

Hence, for sections III, IV and V of this paper, we will ignore the untimed parts of the words, i.e we assume the word σ we wish to align comes with a valid causal process (CN, p) of the model, and seek to find timing functions on said causal process that are both valid, and minimize distance to σ under the metric of choice.

We shall revisit the general timed alignment problem in section VI, and show how our methods can be adapted using existing techniques to provide approaches for solving the general problem.

A. Edit Moves on Timing Functions

Now we come to the problem of deciding how to compare two timing functions over the same causal process, and quantify how close they are to each other.

Much like Levenshtein's edit distance, popularly used in the untimed case of the alignment problem, we view the definition of these distances as an exercise in cost minimisation over the set of all transformations between two words. In order to formalise the same, we need to define what the valid moves of such a transformation could be. We define *moves* as functions that map one timing function over (CN, p) to another. What sort of functions are useful notions of transformation on a timed system?

Example 3. *Let us go back to example 1, and study the process model N presented there. Say we had the following words that did not fit the model, and we wished to analyse how best to modify them to fit them back into the model.*

We start with $\sigma = (reg, 1)\cdot(ct, 2)\cdot(ec, 2)\cdot(dec, 6)\cdot(rej, 6)$. One candidate for the closest valid execution would be $\gamma = (reg, 1) \cdot (ct, 3) \cdot (ec, 2) \cdot (dec, 6) \cdot (rej, 6)$, and it feels reasonable to say that the cost for aligning σ to this γ is 1. A way to arrive at this conclusion is by noticing that on trying to execute σ's firing sequence only the guard for ct fails. If ct were shifted to fire at 3 instead, the whole run would execute without a hitch. This sort of local, almost typographical error can often happen in systems, and it is the simplest kind to fix.

There is, however, another way to look at it. A closest valid timing function would try to preserve the positions of reg and ec but the moment it tries to fire ct, σ runs early. By pushing ct forward, it should take longer for the decision to be made and the rejection to happen, as those events depended on the checking of the ticket being done first. This causes a cascading chain of errors where if ct is moved forward to fire at 3, to preserve the relative relationships between ct and its successors, we get the trace $\gamma' = (reg, 1)\cdot(ct, 3)\cdot(ec, 2)\cdot(dec, 7)\cdot(rej, 7)$, and in this sense, γ' is closer to σ as it preserves more relative gaps. As we just knocked one event a little later by 1 unit, we can view this as a cost 1 edit too. . This is a slightly more complex type of error, but it reflects the real life fact of cascading errors, and hence is an important type of edit to consider when diagnosing such cascading faults.

Based on the above example, we naturally arrive at two types of moves.

We define a *stamp move* as a move that translates the timing function only at a point, i.e., that edits a single element of the timestamp series τ.

Definition 10 (Stamp Move)**.** *Given a timing function $\gamma : E \to \mathbb{R}$, formally, we define this as :*

$\forall x \in \mathbb{R}, e \in E : stamp(\gamma, x, e) = \gamma'$ *where*

$$\forall e' \in E : \gamma'(e') = \begin{cases} \gamma(e') + x & e' = e \\ \gamma(e') & otherwise \end{cases}$$

The next type of move we describe is the more novel and interesting *delay move*. Here, we seek to leverage the structure of the process model itself, by reflecting the causal relationships the pre-order G causes. If an event is G-reachable from another, that means the first event is strictly in the causal history of the second so changing its timestamp should have consequences for all of its causal descendents, while leaving any causally unrelated (non-G-reachable) events undisturbed. Hence, a delay move at e will preserve relative relationships between timestamps in the future, at the cost of shifting the timestamp of every causal descendent of e by the same amount.

Definition 11 (Delay Move)**.** *Given a timing function $\gamma : E \to \mathbb{R}^n$, we define a delay move applied to it as follows :*

$\forall x \in \mathbb{R}, e \in E : delay(\gamma, x, e) = \gamma'$ *where*

$$\forall e' \in E : \gamma'(e') = \begin{cases} \gamma(e') + x & e' \geq_G e \\ \gamma(e') & otherwise \end{cases}$$

The cost of a move is the magnitude $|x|$ in the above definitions, and the cost of a sequence of moves is the sum of the costs of the moves. Armed with these definitions, three natural notions of distance can be constructed.

Definition 12 (Stamp Only Distance : d_t)**.** *Given any two timing functions τ_1, τ_2 over the same causal process (CN, p), we define the stamp-only distance d_t as follows :*

$d_t(\tau_1, \tau_2) = \min\{cost(m) | m \in Stamp^*, m(\tau_1) = \tau_2\}$

Definition 13 (Delay Only Distance : d_θ)**.** *Given any two timing functions τ_1, τ_2 over the same causal process (CN, p), we define the delay-only distance d_θ as follows :*

$d_\theta(\tau_1, \tau_2) = \min\{cost(m) | m \in Delay^*, m(\tau_1) = \tau_2\}$

Definition 14 (Mixed Moves Distance : d_N)**.** *Given any two timing functions τ_1, τ_2 over the same causal process (CN, p), we define the mixed move distance d_N as follows*

$d_N(\tau_1, \tau_2) = \min\{cost(m) | m \in (Stamp \cup Delay)^*, m(\tau_1) = \tau_2\}$

Example 4. *Consider the following example :*

Now for this N, let the observed trace $\sigma = (3, 4, 5) \notin \mathcal{L}(N)$.

The best d_t alignment for the example in the diagram below is $\gamma = (1, 3, 4)$ with minimum cost $d_t(\sigma, \gamma) = 4$.

The best $d_t = \theta$ alignment for the example in the diagram below is also $\gamma = (1, 3, 4)$, but this time with minimum cost $d_\theta(\sigma, \gamma) = 3$, evidenced by the move sequence $(delay(-2, 1)delay(+1, 2))$.

And lastly, the best d_N alignment for the example in the diagram below is also $\gamma = (1, 3, 4)$, the sequence of moves being one stamp and one delay move at the start, $m = stamp(-1, 1)delay(-1, 1)$, and now with minimum cost $d_N(\sigma, \gamma) = 2 < \min\{d_t(\sigma, \gamma), d_\theta(\sigma, \gamma)\}$.

IV. Results and Algorithm : Stamp Edits

Lemma 1 (Stamp only distance : d_t)**.** *d_t as defined above is equivalent to the Manhattan distance or taxicab distance between two points of $\mathbb{R}^n$, where $|E| = n$, say $\mathbf{t} = (t_1, t_2, \ldots t_n)$ and $\mathbf{s} = (s_1, s_2, \ldots s_n)$, defined as*

$$d_t(\mathbf{t}, \mathbf{s}) = \sum_{i=1}^{n} |t_i - s_i|$$

This lemma also allows us to restate the alignment problem for d_t, as we see below.

A. Casting Alignment as a Linear Programming Problem

The alignment problem in the stamp-only setting can be viewed as a problem of convex optimization, minimizing the stamp only cost from a fixed observed trace σ, which by Lemma 1 is $cost(\tau|_i) = \sum_{j=1}^{i} |\tau_j - \sigma_j|$ over the set of valid timestamp series τ for the model N. What remains is to show that the space of valid runs of the model is a convex set, as shown in [13].

Hence, we claim that using an efficient encoding of the target function using the firing domain formulation

described above, the problem can be cast as a linear programming problem, and LPP solvers can solve this in good time in practice.

Theorem 2. *Given a bounded TPN N, an observed trace σ, and its causal process $CN = (B, E, G)$, the alignment problem can be viewed as seeking the vector $(\gamma_1, \gamma_2, \dots \gamma_{|E|})$ that minimizes the quantity $\sum_{e \in E} |\gamma_e - \sigma_e|$ over the set $\gamma \in \mathcal{L}(N)$ which can be cast as a linear programming problem.*

By the above theorem we see that the alignment problem for d_t is easily cast as a linear progamming problem, and can be solved by any linear programming solver. However this approach does not take into account the specific nature of the problem, and LPP methods can be slow in the worst-case.

This bound is not necessarily tight, but either improving it or proving its tightness are left for future work. While the general case seems to be hard to solve efficiently, there is a subclass of process models for which the problem seems more tractable, and moreover, has a quadratic time solution.

B. A Direct Algorithm for Sequential Models

We are given a process model N, an observed trace σ, and its underlying sequential causal process $CN = (B, E, G)$ with homomorphism p for the untimed run of σ on N. We express the cost of aligning a prefix of σ to some timing function τ on the truncation of the sequential causal process up to the ith transition, call it *cost*.

$$cost(\tau|_i) = \sum_{j=1}^{i} |\tau_j - \sigma_j| = |\tau_i - \sigma_i| + cost(\tau|_{i-1})$$

Observe that the minimum cost to align σ depends on exactly two things, the cost of aligning the last stamp, and the minimum cost of aligning the $n-1$ length prefix of σ, such that the last and second last timestamps obey the last static interval constraint. This suggests that the minimal cost of aligning prefixes of σ to prefixes of the model can be recursively calculated as a function of the last timestamp in the alignment of a given prefix.

Let $g_i(d) = \min\{cost(\tau|_i) | \tau|_i \in \mathcal{L}^i(N) \wedge \tau_i = d\}$.

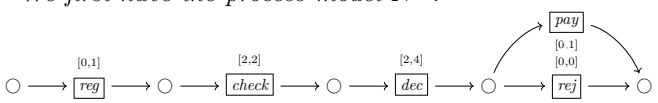
Now, the recursive relation g functions obey is :

$$g_{i+1}(d) = |d - \sigma_{i+1}| + \min_{d - d' \in [a,b]} g_i(d')$$

If we can calculate the family of functions $\forall i \leq n : g_i$ efficiently, then the minimal cost for aligning σ to N is simply $\min_d g_n(d)$.

Example 5. *We take a moment to look at an example of the computation of this family of functions we proposed by considering d_t-alignment on a simplification of our net from 1, collapsing checking the ticket and the examination steps into one check step :*

We first have the process model N' :



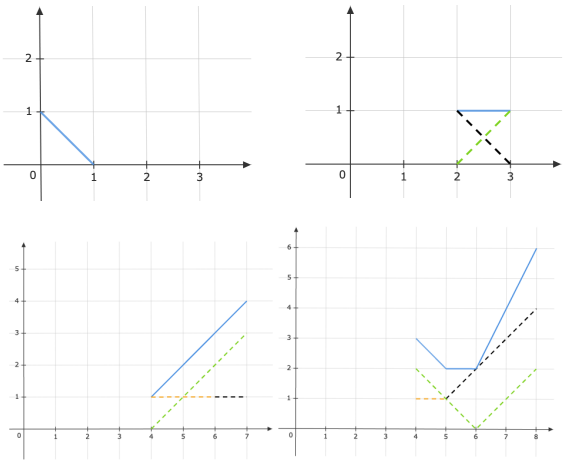


Figure 1. Illustration of a run of the algorithm on Example 5.

The observed trace is $\sigma = (reg, 1) \cdot (check, 2) \cdot (dec, 4) \cdot (pay, 6)$. We begin to compute the family of functions g_i, starting with $g_1(x) = |1 - x|$ in the domain $[0, 1]$. Utilising the recursion, we proceed to compute g_2. The new domain is $[2, 3]$, a translation of the previous one because the guard on the second transition is the point $[2, 2]$. Now, we want the sum of both $|x-2|$ and $\min_{x' - x \in [2,2]} g_1(x')$. The first of these functions, i.e., the contribution of the last timestamp to the error, is pictured in figure 1 in green dashed lines. The second component is pictured in black dashed lines, and is a translation of g_1, as $\min_{x' - x \in [2,2]} g_1(x') = g_1 - 2$.

We construct g_3 similarly, by summing the contribution of the last timestamp, $|x - 4|$, to $\min_{x' - x \in [2,4]} g_2(x')$. We now find an orange segment in the second component. This is used to demarcate any additional segments in $\min_{x' - x \in [a_i, b_i]} g_{i-1}(x')$ not present in g_{i-1}. We see the effect of the min *operation is to attach a segment with the same length as the ith static interval constraint, at the lowest point of g_{i-1}. We continue similarly : The next black graph is a translation of the previous blue one, with an orange line segment or the same length as the next static interval constraint inserted at the lowest point, and the next green graph is $|x - \sigma_i|$ in the new domain, and summing these two gives the next blue graph. As these are all piecewise linear graphs, summing is simply incrementing the slopes of all the segments of the black-orange curve by one if they're to the right of σ_i, and by minus one if they're to the left, and translating the height up as needed.*

The procedure for constructing g_4 is identical, and now finally we see that the minimum cost of 2 is achieved in g_4 by paying at time 6. Now, we know the contribution of the $(pay, 6)$ to the error is 0, so this means we can deduce the corresponding value in g_3 must have been 2, achieved when dec fires at 5. We can backtrack in this manner to build an optimal alignment, such as in this case, $(reg, 1)(check, 3)(dec, 5)(pay, 6)$.

Algorithm 1 d_t Algorithm

Input : σ, SI
Output : $cost = \min_{x \in \mathcal{L}(N)} d_t(x, \sigma)$, $\gamma \in \mathcal{L}(N)$ such that $d_t(\gamma, \sigma) = cost$
Object : $graph = \{(left, slope, right) | graph[i].left = graph[i-1].right\}$
procedure StampOnlyAlgo(σ, SI)
  Initialise System
  $[a, b] \leftarrow SI[i]$
  $graph \leftarrow \{(a, 0, b)\}$
  graphAddMod($graph, \sigma[0]$)
  $graphlist \leftarrow \{graph\}$
  $i \leftarrow 1$
  $cost \leftarrow |a - \sigma[0]|$
  while $i < |sigma|$ **do**
    $[a, b] \leftarrow SI[i]$
    graphMin($graph, a, b$)
    graphAddMod($graph, \sigma[i]$)
    $graphlist.append(graph)$
    $cost \leftarrow cost + |a - \sigma[i]|$
  end while
  $i \leftarrow 0$
  $s \leftarrow graph[0].slope$
  while $graph[i].slope < 0$ **do**
    $s \leftarrow graph[i].slope$
    $cost \leftarrow cost + s * (graph[i].right - graph[i].left)$
    $i \leftarrow i + 1$
  end while
  $\gamma \leftarrow$ backTrack($\sigma, graphlist, cost, SI$)
  Return : $cost, \gamma$
end procedure

With this context, we present the following algorithm that progressively calculates the graphs of g_i (represented as lists of segments, each characterised by their slopes and x-projections of endpoints) each round, and claim it is both correct and efficient. The algorithm precisely executes the above procedure, where the subfunction graphMin takes $g_i(d)$ and outputs

$$\min_{d-d' \in [a,b]} g(d') = \begin{cases} g(d-a) & l + a \leq d < m + a \\ g(m) & m + a \leq d \leq m' + b \\ g(d+b) & m' + b < d \leq h + b \end{cases}$$

Which is exactly the process of taking the last blue graph and creating the next orange-black graph, and subfunction graphAddMod takes $\min_{d-d' \in [a,b]} g(d')$ and adds $|d - \sigma_{i+1}|$ to it, i.e., adding the green component. Lastly, the backTrack algorithm backtracks through the list of graphs $\{g_i\}_n$ constructed to reverse engineer a word that aligns with the minimal cost calculated.

Theorem 3. *Algorithm 1 is correct, i.e., it outputs a valid execution $\gamma \in \mathcal{L}(N)$ which minimizes the distance $d_t(\gamma, \sigma)$ to the trace σ.*

Remark 1. *Note that the subfunctions graphMin and graphAddMod both run through the list of segments of the graph once each, and hence are linear in the sizes of their inputs, and Algorithm 1 calls each of these subfunctions once for each letter of the observed trace, each time on a graph with size linear in the current prefix. So the cost calculation section of Algorithm 1 has quadratic time complexity in the size of the input.*

The subfunction backTrack runs a binary search on each stored graph in graphlist to find a trace in the language that does indeed give the minimum cost, letter by letter, and so overall takes $O(n \log(n))$ time, keeping the overall time complexity $O(n^2)$ where $n = |\sigma|$.

V. Results and Algorithm : Delay Edits

Delay edits provide a new way of thinking about transformations over time-series, and inspire a new way of viewing timing functions over causal processes. There is of course the standard definition, $\tau : E \to \mathbb{R}^+$ that assigns to each event a timestamp that records exactly when the event occurs.

Instead, thinking along the lines of durations between events occurring, we will often benefit from considering the following representation when speaking about delay moves, defined as to view a timed word not in terms of its absolute timestamps, but by the delays between them.

Definition 15 (Flow Function)**.** *Given a causal process (CN, p) over N, where $CN = (B, E, G)$, and a (not necessarily valid) timing function $\tau : E \to \mathbb{R}^+$, we first define the flow function of τ, $f_\tau : E \to \mathbb{R}^+$ such that*

$$f_\tau(e) = \begin{cases} \tau(e) & {}^\bullet\bullet e = \emptyset \\ \tau(e) - \tau(e') & e' \in {}^\bullet\bullet e, \\ & \tau(e') = \max_{e'' \in {}^\bullet\bullet e} \{\tau(e'')\} \cup \{0\} \end{cases}$$

Example 6. *Consider again the model in example 1. Say we want to align $w = (reg, 1)(et, 2)(ct, 5)(dec, 6)(rej, 8)$ to N. Consider the values the clock takes during a faulty "execution" $reg \to 1, et \to 1, ct \to 4, dec \to 1, rej \to 2$. This is the flow function.*

Note that e' as defined here is the latest causal predecessor of e . Hence, f_τ as defined produces exactly the time durations that the guards of each transition in the model checks, i.e. the clock function values during the run.

As defined, we see that if a word is in the language of the model, then its f function maps events to values that lie within the event's static interval constraint. This condition is unfortunately not sufficient due to urgency, but it is quite close to the exact condition necessary for a word to be in the language of a time Petri net.

Also note that given the underlying causal process and the resulting f_τ, we can reconstruct τ quite straightforwardly as $\forall e \in E : \tau(e) = \sum_{e' \leq_G e} f_\tau(e')$.

Lemma 4 (Delay only distance : d_θ)**.** *d_θ as defined previously is equivalent to the following formulation directly using the flow function: $d_\theta(\tau_1, \tau_2) = \sum_{e \in E} |f_{\tau_1}(e) - f_{\tau_2}(e)|$.*

The flow function is hence a dual representation of timing functions, as if a timing function traditionally labels its

transition nodes with timestamps, the flow function labels edges leading up to transitions with the duration since said transition was enabled. This gives us a new way to formulate the alignment problem in terms of minimising distance from the flow vector of σ.

Example 7. *Let us return to example 6. Say we now wish to align w to N. We recall that we calculated its flow function to be*
$reg \to 1, et \to 1, ct \to 4, dec \to 1, rej \to 2$. *Pick the closest flow values in the model possible, naively. The resulting flow function is* $reg \overset{1}{\mapsto}, et \overset{2}{\mapsto}, ct \overset{2}{\mapsto}, dec \overset{2}{\mapsto}, rej \overset{0}{\mapsto}$ *From which we can reconstruct the closest word* $u = (reg, 1)(et, 3)(ct, 3)(dec, 5)(rej, 5)$.

Clearly if one tries to move any component of u's flow function any closer to that of w, the word will no longer be in the language.

We could reason this way because d_θ allows one to locally edit traces without affecting membership in the language of the TPN, as both the firing rule and delay edits concern themselves only with the duration of time elapsed between a transition's enabling and its firing. This relationship is especially clear in extended free choice TPNs, where the only events that determine the fireability of an event are its causal history and events with the exact same preset which ensures the interval constraint. Hence, we present Algorithm 2, which minimizes the delay cost at each transition locally, while still overall ensuring membership in the language of the model. The main step in the algorithm is line 12, the rest just ensures validity.

Algorithm 2 Local d_θ Algorithm

1: **Input :** N, $CN = (B, E, G), p$, $\sigma : E \to \mathbb{R}^+$
2: **Output :** $\gamma : E \to \mathbb{R}^+$ such that γ is a valid timing function and $d_\theta(\gamma, \sigma) \leq d_\theta(x, \sigma)$ for all valid x
3: $Curr = Min(CN)$
4: $Enabled = \{t | {}^\bullet t \subseteq p(Curr)\}$
5: $FD = \{(t, Eft(t), l, 0) | t \in Enabled, l = \min_{\bullet t' = \bullet t} Lft(t')\}$
6: $Soon = \{(t, eft, l, toe) \in FD | l + toe \text{ is minimal in } FD\}$
7: **while** $E \neq \emptyset$ **do**
8: Pick $(t, eft, l, toe) \in Soon, t \in p(E)$
9: $E \leftarrow E \setminus \{e | p(e) = t\}$
10: $e \leftarrow p^{-1}(t)$
11: $Curr \leftarrow \{Curr \setminus \{{}^\bullet e\}\} \cup \{e^\bullet\}$
12: $f_\gamma(e) = \text{argmin}_{x \in [eft, l]} |x - f_\sigma(e)|$
13: **for all** $t' \in T \wedge {}^\bullet t' = {}^\bullet t$ **do**
14: $Enabled \leftarrow Enabled \setminus \{t'\}$
15: $FD \leftarrow FD \setminus \{(t', e', l', toe')\}$
16: **end for**
17: **for all** $t' \in T \wedge t'^\bullet = t^\bullet$ **do**
18: $Enabled \leftarrow Enabled \cup \{t'\}$
19: $eft' \leftarrow Eft(t')$
20: $l' \leftarrow \min_{\bullet t'' = \bullet t'} Lft(t'')$
21: $toe' \leftarrow toe + f_\gamma(e)$
22: $FD \leftarrow FD \cup \{(t', eft', l', toe')\}$
23: **end for**
24: $Soon = \{(t, eft, l, toe) \in FD | l + toe \text{ is minimal in } FD\}$
25: **end while**
26: **return** γ

Theorem 5. *Algorithm 2 is correct, i.e., it outputs a valid execution $\gamma \in \mathcal{L}(N)$ which minimizes the distance $d_\theta(\gamma, \sigma)$ to the trace σ.*

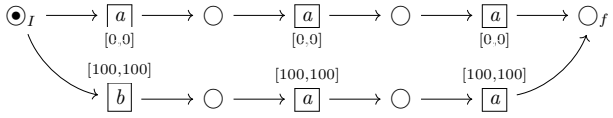
Remark 2. *Note that Algorithm 2 runs through the events of the causal process E exactly once, and each time runs through the set of currently enabled transitions, and the total set of transitions, and hence its time complexity is linear in the size of the input $|CN|$, and the size of the transition set, i.e, $O(|E||T|)$.*

VI. The General Timed Alignment Problem

Now that we have provided methods to tackle the purely timed alignment problem for two metrics, we return to the general timed alignment problem. As our examples throughout have described, the purely timed alignment problem does not so much ignore the action labels as assume that they do not need editing. As a result, one approach for solving the general timed alignment problem would be to begin by completely ignoring the timed aspect, and aligning only the untimed part of the word w and the process model N (now reduced to the underlying untimed Petri net) using the $\mathbb{A}^*$ algorithm [3], which can be used to get a valid control flow, that is, a valid causal process. This, for us, is an untimed word w' which can then be given a timing function based on that of w, repeating or deleting timestamps for any insertion or deletion moves. This yields a valid causal process and a potentially invalid timing function over it, which is then cast as a purely timed alignment problem.

This approach is quite naive, as it may have a tendency to exaggerate the importance of action label errors. For example, consider the following scenario :

Example 8. *Consider the following net N_3 where the final marking is presumed to be the sink place:*



Now, let the observation be $(a, 100)(a, 100)(a, 100)$. The alignment the above approach would provide is $(a, 0)(a, 0)(a, 0)$. But if we assign even a tenth of the cost of action edits to timestamp edits, clearly the closer word in the language was $(b, 100)(a, 100)(a, 100)$.

The issue here is the tradeoff between timestamp edits and action label edits. A better approach would assign one cost c_A to action edits and another c_T to time edits, and seek to minimise their sum. A potential fix to our previous idea would be to essentially retain the above approach, but instead of just doing it for the best untimed alignment, take a selection of good untimed alignments (up to some

allowable threshold), and try to align timestamps for each untimed candidate, and then minimise the total cost over this set. This alleviates some of the bias towards preserving action labels by allowing for action deviations in case the timestamp alignment proves particularly easy, but this is still a naive approach to the wider problem, and we hope to find better solutions in the future.

VII. Implementation

We have implemented the stamp only alignment algorithm in python, available at https://github.com/NehaRino/TimedAlignments. This algorithm is quadratic in time complexity, and runs well in practice, as seen below

Trace Length	10	100	1000
Running Time (seconds)	0.001	0.1	9.8

VIII. Perspectives and Conclusion

In this paper, we posed the alignment problem for timed processes, devised three metrics for studying alignments for timestamp sequences, and solved the purely timed alignment problem for the first two metrics proposed. As far as we know, this is the first step in conformance checking for time-aware process mining, and much further work can be inspired from this point. The alignment problem for the third metric d_N is a first future direction. Secondly, for both metrics studied here the class of models for which the alignment problem was solved efficiently are structurally restricted (being sequential causal processes for d_t and extended free choice time Petri nets for d_θ) and it would be interesting to see how to broaden their scope to larger classes of process models. Thirdly, further investigation in the general timed alignment problem is necessary, as our proposed approach here is rather rudimentary and can certainly be improved. Lastly, there are a number of other conformance artefacts that can be set and studied in the timed setting, such as anti-alignments [14], and one can better develop all such conformance checking methods to account for timed process models.

References

[1] W. M. P. van der Aalst, *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.

[2] J. Carmona, B. F. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking - Relating Processes and Models*. Springer, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-99414-7

[3] A. Adriansyah, "Aligning observed and modeled behavior," Ph.D. dissertation, Technische Universiteit Eindhoven, 2014.

[4] M. Boltenhagen, T. Chatain, and J. Carmona, "A discounted cost function for fast alignments of business processes," in *BPM 2021, Proceedings*, ser. LNCS, vol. 12875. Springer, 2021, pp. 252–269. [Online]. Available: https://doi.org/10.1007/978-3-030-85469-0_17

[5] S. Cheikhrouhou, S. Kallel, N. Guermouche, and M. Jmaiel, "The temporal perspective in business process modeling: a survey and research challenges," *Service Oriented Computing and Applications*, vol. 9, pp. 75–85, 2014.

[6] J. Eder, E. Panagos, and M. Rabinovich, "Time constraints in workflow systems," in *CAiSE*, 1999.

[7] A. Nguyen, S. Chatterjee, S. Weinzierl, L. Schwinn, M. Matzner, and B. Eskofier, *Time Matters: Time-Aware LSTMs for Predictive Business Process Monitoring*, 03 2021, pp. 112–123.

[8] W. Aalst, H. Schonenberg, and M. Song, "Time prediction based on process mining," *Inf. Syst.*, vol. 36, pp. 450–475, 04 2011.

[9] A. Rogge-Solti, R. Mans, W. M. P. van der Aalst, and M. Weske, "Repairing event logs using timed process models," in *On the Move to Meaningful Internet Systems: OTM 2013, Proceedings*, ser. LNCS, vol. 8186. Springer, 2013, pp. 705–708. [Online]. Available: https://doi.org/10.1007/978-3-642-41033-8_89

[10] R. Conforti, M. L. Rosa, A. H. M. ter Hofstede, and A. Augusto, "Automatic repair of same-timestamp errors in business process event logs," in *BPM 2020, Proceedings*, ser. LNCS, vol. 12168. Springer, 2020, pp. 327–345. [Online]. Available: https://doi.org/10.1007/978-3-030-58666-9_19

[11] W. M. P. van der Aalst and L. F. R. Santos, "May I take your order? - on the interplay between time and order in process mining," in *Business Process Management Workshops - BPM 2021 International Workshops*, ser. Lecture Notes in Business Information Processing, A. Marrella and B. Weber, Eds., vol. 436. Springer, 2021, pp. 99–110. [Online]. Available: https://doi.org/10.1007/978-3-030-94343-1_8

[12] T. Chatain and N. Rino, "Timed Alignments," 2022, hal-03703712.

[13] B. Barthomieu and M. Menasche, "A state enumeration approach for analyzing time petri nets," 1982. [Online]. Available: https://projects.laas.fr/tina/papers/atpn82.pdf

[14] T. Chatain, M. Boltenhagen, and J. Carmona, "Anti-alignments - measuring the precision of process models and event logs," *Inf. Syst.*, vol. 98, p. 101708, 2021. [Online]. Available: https://doi.org/10.1016/j.is.2020.101708