Timed Anti-Alignments for Acyclic Marked Graphs

Stefano Bavaro[1], Thomas Chatain[2] and Boudewijn F. van Dongen[1]

[1] *Eindhoven University of Technology, Eindhoven, The Netherlands*
[2] *Université Paris-Saclay, ENS Paris-Saclay, CNRS, Laboratoire Méthodes Formelles, Gif-sur-Yvette, France*

Abstract

This paper addresses conformance checking in a time-aware setting, that relates the timing of recorded events in the log with the time constraints of the process model specified as a Time Petri Net. To evaluate whether a model reflects the observed executions, several quality criteria were proposed.

We define the Timed Anti-Alignment Problem as finding model traces whose timing most deviate from observed log traces. Anti-alignments, as witnesses for imprecision of the model, were proposed in untimed settings to measure precision.

We solve the Purely Timed Anti-Alignment Problem for Acyclic Time Marked Graphs. By framing the problem as an optimization task with linear constraints, we enable the use of efficient Linear Programming solvers. Finally, we test our implementation's performance to understand its practicability.

Keywords

Conformance Checking, Petri Nets, Anti-Alignments, Linear Programming, Precision

1. Introduction

Processes are crucial for organizations, coordinating activities needed for delivering products and services. The increase in event data collection has raised the need for process analysis techniques. Process Mining uses event data to discover processes, check compliance, analyze bottlenecks, compare process variants, and suggest improvements [1]. Conformance checking techniques evaluate how well a process model represents an actual process. Real executions, recorded in event logs, are collections of events executed during system operations: process models abstract these operations. The metrics that assess model accuracy include fitness, precision[2], generalization, and simplicity. Alignments are essential for the first three metrics, relating the model to an observed trace by providing the run that most closely resembles it [3]. This paper focuses on Anti-Alignments [4], which identify the most deviating behaviors not seen in the log. For models that should strictly adhere to specific behaviors (e.g., banking, healthcare), the absence of highly deviating Anti-Alignments and their early identification may be crucial. This conformance checking tool has been first introduced in [5] as a way to complement the already existing notion of alignment, i.e. the run of a process model the most similar to a given log trace. Later in the same year, in [6] the authors expanded on the utility of anti-alignments, showing how they can be used to measure two fundamental process mining metrics, i.e. precision (highly deviating anti-alignments indicate a loss in precision) and generalization [4] [6], which remains the main use of this conformance checking tool. In untimed settings, Anti-Alignments have been defined using both Hamming and Levenshtein distances [4]. For the Levenshtein distance, an implementation through a SAT encoding is provided in [7]. By introducing a discount factor into the distance calculation, a more efficient algorithm was later developed in [8], allowing for practical yet approximate computation with reduced complexity. However, Anti-Alignments have only been explored in untimed settings so far. Our study is instead situated in the field of Time-Aware Process Mining, which focuses on identifying timing-related properties in processes, such as the minimum delay between events or the maximum duration for the system to reach a specific state. This field not only seeks to understand the processes governing system behavior but also the time constraints they obey [9] [10] [11]. Various established process model notations already exist to incorporate time constraints. In this paper we use Time Petri

ATAED'25, International Workshop on Algorithms & Theories for the Analysis of Event Data, 2025
s.bavaro@student.tue.nl (S. Bavaro); thomas.chatain@ens-paris-saclay.fr (T. Chatain); b.f.v.dongen@tue.nl (B. F. v. Dongen)

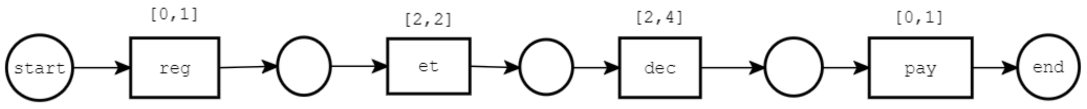


Figure 1: Example of a Sequential ATMG.

Nets (TPNs) [12], where each transition t has an interval $[a, b]$ of possible firing delays: if transition t was last enabled at time θ, then t can not fire before $\theta + a$ and must fire by $\theta + b$, unless it becomes disabled. Firing a transition takes no time to complete [13]. Hence, TPNs record and check the duration for firing transitions, imposing constraints on the relationships between the timestamps of different events. In [14], it is shown how to describe symbolically the possible execution times for events of a poset execution of a TPN. The case of extended free choice TPNs was studied in [15]. As Time-Aware Process Mining grows, it becomes necessary to adapt quality measures and conformance checking artifacts to consider temporal constraints. While in untimed contexts the computation of alignments has been widely studied [16] [17] [18], the first attempt to extend this work to timed settings was made only in [19]: the problem was addressed using two novel distance metrics and focusing exclusively on the temporal aspects of traces. The authors later expand the work to include a solution for an additional distance metric on timed traces [20]. Today, the problem of Anti-Alignments in timed settings remains unaddressed in current research: this paper aims at being the first step in this direction. We first define the General Timed Anti-Alignment Problem (GTAAP), i.e. finding the most distant traces from a log using distance functions that consider both time and activities. However, to eventually approach such problem, we initially focus on the reduced Purely Timed Anti-Alignment Problem (PTAAP), where the untimed part of traces is not considered. We consider two distance functions, Stamp-Only and Delay-Only [19]. We restrict the study to TPNs with no choice between activities and no loops, i.e., Acyclic Time Marked Graphs (ATMGs). For this class of models, each trace consists of the same activities, allowing us to ignore its untimed part to focus on Purely Timed Anti-Alignments. We present the following example to provide an initial understanding of the notion of Purely Timed Anti-Alignments.

Example 1. The process in Figure 1 describes a simple refunding process of an airline (adapted from [21]): it starts with the registration of a client, the examination of their ticket, and the final decision and payment from the airline. Considering only the activity information, this model produces one trace variant, $\sigma = (reg, et, dec, pay)$. However, a TPN produces "timed traces", where the timestamps for each activity must comply with the model constraints. Therefore, variants differ in terms of timestamp information. An example of a timed log accepted by this TPN is:

$$L = \left\{ \begin{array}{c} \langle(\text{reg}, 0), (\text{et}, 2), (\text{dec}, 4), (\text{pay}, 4)\rangle \\ \langle(\text{reg}, 0), (\text{et}, 2), (\text{dec}, 4.5), (\text{pay}, 5)\rangle \\ \langle(\text{reg}, 0.5), (\text{et}, 2.5), (\text{dec}, 5), (\text{pay}, 5.2)\rangle \end{array} \right\}$$

Ignoring the activity information (as it is the same for all traces) and only considering timestamp sequences, the log becomes:

$$L = \left\{ \begin{array}{c} (0, 2, 4, 4) \\ (0, 2, 4.5, 5) \\ (0.5, 2.5, 5, 5.2) \end{array} \right\}$$

Using the Stamp-Only distance (Manhattan distance for timestamp sequences), we aim to find a Purely Timed Anti-Alignment, i.e., a timestamp sequence accepted by the TPN and the most distant from the log. The distance from the log of a candidate Anti-Alignment is its minimal distance from any trace in the log. Since in this example the traces are relatively close to each other (not sparse in the search space), the intuitive approach is to maximize the timestamp value for each activity. Thus, the anti-alignment is the timed trace $(\text{reg}, 1), (\text{et}, 3), (\text{dec}, 7), (\text{pay}, 8)$. This trace being far from all the traces recorded in the log, it can be considered as a witness for imprecision and used in a precision metric.

In the provided example, due to its simplicity, we could find a solution using common sense and manual checks. However, this is not feasible for all cases. In this paper, we solve the PTAAP by framing it as an optimization problem and using only linear constraints. As the problem is non-linear, we derive equivalent linear formulations, obtaining a Mixed Integer Programming (MIP) problem. This allows us to use more efficient Linear Programming solvers. We test the performance of our implementation to assess its practicability, evaluating both accuracy and time requirements for increasingly complex instances. Our results indicate lower time requirements and improved accuracy compared to a brute-force approach. The paper is organized as follows. Section 2 provides the preliminary definitions: TPNs, functions to work with timestamps, ATMGs, and the distances used. Section 3 formalizes the GTAAP and the PTAAP, with relevant examples. Section 4 briefly summarizes the characteristics of the linear reformulation of the problem. Section 5 reports experiments and their results. Section 6 concludes the paper and suggests future research directions.

2. Preliminaries

2.1. Timed process modeling

Definition 1 (Timed trace and Timed events). A timed trace is a sequence $\sigma \in (\Sigma \times \mathbb{R}^+)^*$ of timed events. In other words, we represent events as pairs (a, τ) where $a \in \Sigma$ is the label of the action and τ denotes the time at which said action was taken.

Definition 2 (Timed event log). A timed event log L is a multiset of timed traces $\sigma \in (\Sigma \times \mathbb{R}^+)^*$.

The timed process model used here are Time Petri Nets.

Definition 3 (Time Petri Net [12]). A *Time Petri Net* (TPN) is a tuple $N = (P, T, F, SI, \Sigma, \lambda, M_0, M_f)$, where P is the set of places, T is the set of transitions (with $P \cap T = \emptyset$), $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, $SI : T \to (\mathbb{R}^+) \times (\mathbb{R}^+ \cup \{\infty\})$ is the static interval function, with $SI(t) = (Eft(t), Lft(t))$, where Eft stands for Earliest Firing Time and Lft for Latest Firing Time (therefore $Eft(t) \leq Lft(t)$), $\lambda : T \to \Sigma$ is the labelling function that labels 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 $t \in T$, the pre-set of t is ${}^\bullet t = \{p \in P | (p, t) \in F\}$ and its post-set is $t^\bullet = \{p \in P | (t, p) \in F\}$ (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*. Every enabled transition hence has an implicit clock, measuring how long it has been since its most recent enabling. 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 a delay of $\theta \in \mathbb{R}^+$ iff t is enabled at M, and $I(t) + \theta \in [Eft(t), Lft(t)]$, and $\forall t' \in Enabled(M),\ I(t') + \theta \leq Lft(t')$. This firing is denoted $(M, I)[t\rangle(M', I')$ with new state (M', I') defined as follows:

$$M'(p) = \begin{cases} M(p) + 1 & p \in t^\bullet \setminus {}^\bullet t \\ M(p) - 1 & p \in {}^\bullet t \setminus t^\bullet \\ M(p) & otherwise \end{cases}$$

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

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. In this paper, we consider TPNs with only one token in every starting place.

Definition 4 (Language of a Time Petri Net)**.** A timed trace $\sigma = (a, \tau)^n \in (\Sigma \times \mathbb{R}^+)^n$ is in the language of (or is accepted by) a TPN, i.e. $\sigma \in \mathcal{L}(N)$, if there is a fireable sequence of transitions $(t_0, t_1, ..., t_n) \in T^n$ such that $\langle(\lambda(t_0), \tau_0), (\lambda(t_1), \tau_1), ..., (\lambda(t_n), \tau_n)\rangle = \sigma$ 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 2. The TPN in Figure 2 (adapted from [21]) models the refunding process of an airline. An acceptable timed trace for it is $\langle(reg, 1), (ex, 1.5), (cid, 3), (ct, 3), (dec, 5)\rangle$. Initially, actions reg and cid are enabled: the request is registered at time 1, so the marking is updated by removing the token from reg's pre-place and filling its two post-places, thus enabling ex and ct. After 0.5 time units, at time 1.5, the action ex is triggered, fulfilling the constraint and populating one pre-place of dec. Later, the actions cid and ct fire in parallel after being enabled for 3 and 2 units of time respectively, at time 3: as dec is now enabled, it is executed after at time 5, after two time units, and the final marking is reached.

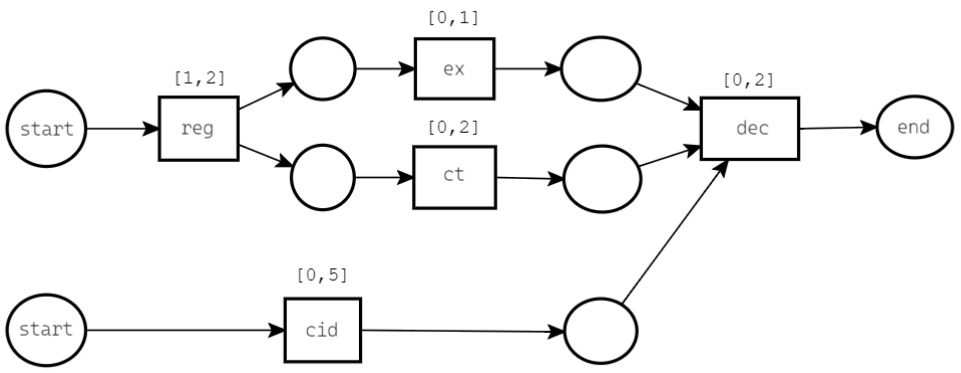


Figure 2: Example of an ATMG. The labels assigned are: Register request (reg), Examine request (et), Check ticket (ct), Check ID (cid), Decide (dec).

As in this paper we will mostly consider timestamp sequences, we also define some functions that will be necessary to operate directly on sequences of timestamps.

Definition 5 (Timing function)**.** Given a TPN N and a trace $\sigma = \langle(a_1, \tau_1), ..., (a_n, \tau_n)\rangle \in \mathcal{L}(N)$, we define the timing function for σ as $\gamma_\sigma : \{1, 2, ..., n\} \to \mathbb{R}^+$ s.t. $\gamma_\sigma(i) = \tau_i$. The set of all acceptable timing functions for the TPN N will be denoted as Γ_N, i.e. $\Gamma_N = \{\gamma_\sigma | \sigma \in \mathcal{L}(N)\}$.

Definition 6 (Timing sequence)**.** Given a TPN N and a timing function $\gamma \in \Gamma_N$, we define the timing sequence obtained from γ using the bijective function $\tau : \Gamma_N \to \mathcal{R}^n$ s.t. $\tau(\gamma) = (\gamma(1), \dots, \gamma(n))$. In other words, $\tau(\gamma_\sigma) = \pi_2(\sigma)$, i.e. the projection of the timestamps of the trace σ.

Example 3. Given the TPN and the trace σ in Example 2, the corresponding timing function is $\gamma_\sigma(1) = 1, \gamma_\sigma(2) = 1.5, \gamma_\sigma(3) = 3, \gamma_\sigma(4) = 3, \gamma_\sigma(5) = 5$. Consequently, the timing sequence is $\tau(\gamma_\sigma) = (1, 1.5, 3, 3, 5)$.

In order to retrieve the relationships between transitions, i.e. retrieve all the transitions that contribute to enabling a transition t and all the transitions that are, at least partly, enabled by the firing of a transition t, we define:

Definition 7 (Parent and Children Transitions)**.** Given a TPN N and one of its transitions t, we define its parent transitions, i.e. set of transitions that, when firing, contribute to enabling t, and its children transitions, i.e. the set of transitions that, when t fires, become at least partly enabled, respectively as:

$$pre_T(t) = \bigcup_{p \in {}^\bullet t} {}^\bullet p \qquad\qquad post_T(t) = \bigcup_{p \in t^\bullet} p^\bullet$$

To ease notation and improve readability, we will denote these two functions also as ${}^\bullet_T t$ and $t^\bullet_T$ respectively. We also define the recursive counterparts of these functions as:

$$pre^*_T(t) = {}^\bullet_T t \cup \bigcup_{t_j \in {}^\bullet_T t} pre^*_T(t_j) \qquad\qquad post^*_T(t) = t^\bullet_T \cup \bigcup_{t_j \in t^\bullet_T} post^*_T(t_j)$$

In this paper we use a sub-class of TPNs, Acyclic Time Marked Graphs (ATMGs), i.e. an acyclic TPN where each place has no more than one input and one output transition.

Example 4. Two examples of ATMG are in Examples 1 and 2. While the first ATMG allows only for sequential behaviour, the second one introduces parallelism at the level of the transition cid, executable in every moment before the execution of dec, and at the level of transitions ex and ct, that can be executed in different orders or at the same time.

As we are mainly interested in the timestamp information of timed traces, it is useful to map them to points in a n-dimensional space. Since the traces produced by ATMGs contain always the same activities, but possibly with a different execution order , we want to define a unique index for every transition to identify the related timestamp in a timing sequence (n-dimensional point). This implies that, when considering only timing sequences, the execution order of the events will not influence how we map activities to the sequence of timestamps. We will call this function the *ordering function*. We assume that each ATMG comes with a bijective ordering function $ord : T \rightarrow \{1, ...|T|\}$, so that the timestamp for the event originated by transition t, with $ord(t) = i$, is the value of the i-th dimension of the mapped point. To improve readability, we refer to specific transitions using the index assigned, i.e. t_i is the transition related to the event in the i-th position of an ordered trace.

Example 5. Given the ATMG in Example 2, a possible ordering is: $ord(reg) = 1$, $ord(ex) = 2$, $ord(ct) = 3$, $ord(cid) = 4$, $ord(dec) = 5$. Given a point $x = (x_1, x_2, x_3, x_4, x_5) \in \mathbb{R}^5$, each value x_i is the timestamp of the action generated by the transition with index i. An example trace $\sigma = \langle (reg, 2), (ex, 3), (ct, 4), (cid, 2), (dec, 4) \rangle$ is then mapped to the point $(2, 3, 4, 2, 4)$. The ordering of the indexes assigned to transitions might not reflect the ordering of transition executions: as such ordering is not the same for all traces, it has to be agreed beforehand.

To further simplify notation, some functions defined over transitions will equivalently be defined over indexes, and vice versa (this is possible when a bijective ordering function is defined for a TPN, i.e. when neither choice nor loops are allowed, which is the case for ATMGs). For example, the function pre_T, previously defined over transitions, can be applied to indexes to return the indexes of resulting transitions: ${}^{\bullet}{}_T\, i = \{j \in 1, ..., d \mid t_j \in {}^{\bullet}{}_T\, t_i\}$.

2.2. Distances on Timing Functions

As the Anti-Alignment problem in this paper is restricted to the time relationships, we hereby describe the two types of distances that we will consider, defined by Rino and Chatain in [19]. To define them, we make use of the definition of *moves*, i.e. functions that map one time sequence to another. In [19], two types of moves are described:

- *Stamp Move*: changes the timestamp for one event in a trace, i.e. edits one value τ of a timestamp sequence.
- *Delay Move*: changes the timestamp for one event in the trace and potentially shifts the timestamps of the following actions, i.e. edits one or more values τ of a timestamp sequence.

Definition 8 (Stamp Move)**.** Given an ATMG timing function $\gamma : K = \{1, 2, ..., d\} \rightarrow \mathbb{R}^+$, $\forall x \in \mathbb{R}, \forall i \in K$, a Stamp Move is a function $stamp_\gamma(x, i) = \gamma'$ where,

$$\forall j \in K,\ \gamma'(j) = \begin{cases} \gamma(j) + x & j = i \\ \gamma(j) & otherwise \end{cases}$$

Example 6. Given the ATMG N and the timing function γ related to the trace σ of Example 5, a Stamp Move can be $stamp_\gamma(4, 2) = \gamma'\ s.t.\ \tau(\gamma') = (2, 7, 4, 2, 4)$. The resulting trace is not accepted by N, as the timestamp of activity ex goes outside the interval of values that it can take.

While a Stamp Move is purely local (when it is applied for the i-th activity in a trace it does not imply a derailment of the rest of the system), a Delay Move will instead preserve relative relationships in the future, potentially shifting the timestamp of every causal descendent of the action on which it is applied. However, the cost of a Delay Move is not necessarily propagated equally to all the following activities. In fact, it depends on the *Flow Function*, an alternative representation of timing function that, instead of assigning timestamps to a transition, labels each transition with the duration since it was enabled.

Definition 9 (Flow function)**.** Given an ATMG timing function $\gamma : K = \{1, \ldots, d\} \to \mathbb{R}^+$, the flow function of γ is defined as $f_\gamma : K \to \mathbb{R}^+$ s.t.

$$f_\gamma(i) = \begin{cases} \gamma(i) & {}^{\bullet}{}_T\, t_i = \emptyset \\ \gamma(i) - \max_{k \in {}^{\bullet}{}_T\, i} \gamma(k) & otherwise \end{cases}$$

Definition 10 (Delay Move)**.** Given an ATMG timing function $\gamma : K = \{1, 2, ..., d\} \to \mathbb{R}^+, \forall x \in \mathbb{R}, \forall i \in K$, a Delay Move is a function $delay_\gamma(x, i) = \gamma'$, where

$$\forall j \in K,\ \gamma'(j) = \begin{cases} \max_{k \in {}^{\bullet}{}_T\, j} \gamma'(k) + f_\gamma(j) & j \in post^*_T(i) \\ \gamma(i) + x & j = i \\ \gamma(j) & otherwise \end{cases}$$

Example 7. Given the ATMG and the timing function γ related to the trace σ of Example 5, the corresponding *Flow Function* would be $f_\gamma(1) = 2, f_\gamma(2) = 1, f_\gamma(3) = 2, f_\gamma(4) = 2, f_\gamma(5) = 0$. Examples of Delay Moves (results are written as timing sequences instead of timing functions) are:

1. $delay_\gamma(1, 1) = (3, 4, 5, 2, 5)$
2. $delay_\gamma(0.5, 2) = (2, 3.5, 4, 2, 4)$
3. $delay_\gamma(2, 2) = (2, 5, 4, 2, 5)$

In the first move, the most straightforward, the same delay is passed on to all the following actions.
In the second move, the delay applied to action ex does not affect the rest of the trace. This is because, even after the delay, the maximum timestamp of the parents of transition dec remains that of transition ct ($\gamma'(2) = 3.5 < 4 = \gamma'(3)$).
In the third move, the delay applied to action ex affects the rest of the trace because the maximum timestamp of the parents of transition dec changes. However, action dec is not delayed by the same amount as ex. Its new value is the new maximum previous timestamp, 5, plus the value of the flow function for dec, which is 0.

Distance functions for time-aware scenarios can be considered as a cost minimisation problem over the set of all moves between two traces. Specifically, given two ATMG timing functions γ_1, γ_2 over the same set $K = \{1, 2, ..., d\}$, we consider two distance functions:

- Stamp-Only distance: the minimum cost for a sequence of Stamp Moves that transforms γ_1 into γ_2.
- Delay-Only distance: the minimum cost for a sequence of Delay Moves that transforms γ_1 into γ_2.

However, formulating these distances as optimization problems is impractical. A more feasible way to compute them for ATMGs has been presented in [19].

Lemma 1 (Stamp-Only distance : d_t)**.** *Given two ATMG timing functions γ_1, γ_2 for the same ATMG N, with $|T| = d$, the Stamp-Only distance $d_t(\gamma_1, \gamma_2)$ is the Manhattan distance or L1 Norm between the two timing sequences, i.e.:*

$$d_t(\gamma_1, \gamma_2) = \sum_{i=1}^{d} |\gamma_1(i) - \gamma_2(i)| = ||\tau(\gamma_1) - \tau(\gamma_2)||_1$$

Lemma 2 (Delay-Only distance : d_θ). *Given two ATMG timing functions γ_1, γ_2 for the same ATMG N, with $|T| = d$, the Delay-Only distance $d_\theta(\gamma_1, \gamma_2)$ is the Manhattan distance or L1 Norm between the two timing sequences, modified by applying the flow function to them, i.e.:*

$$d_\theta(\gamma_1, \gamma_2) = \sum_{i=1}^{d} |f_{\gamma_1}(i) - f_{\gamma_2}(i)|$$

Example 8. Given two timing functions accepted by the ATMG in Example 5, expressed as timing sequences and with the same ordering of the example, $\tau(\gamma_1) = (1, 2, 1, 0, 3)$ and $\tau(\gamma_2) = (2, 2, 3, 3, 5)$, it results that

- $d_t(\gamma_1, \gamma_2) = |1 - 2| + |2 - 2| + |1 - 3| + |0 - 3| + |3 - 5| = 8$
- $d_\theta(\gamma_1, \gamma_2) = |1 - 2| + |1 - 0| + |0 - 1| + |0 - 3| + |1 - 2| = 7$

The fact that both distances can be represented as Manhattan distances is a significant finding that makes the solution to the PTAAP similar for both.

3. The Anti-Alignment Problem in Timed Settings

The Anti-Alignment Problem, given a log and a TPN, involves finding the valid run(s) of the model that are the farthest from the log for a given distance metric.

Definition 11 (The General Timed Anti-Alignment Problem (GTAAP)). Given a Time Petri Net N and a timed log $L \subseteq \mathcal{L}(N)$, find a timed trace $\alpha \in \mathcal{L}(N)$ such that $d(\alpha, L) = \max_{\alpha \in \mathcal{L}(N)} d(\alpha, L)$, where $d(\alpha, L) = \min_{\sigma \in L} d(\alpha, \sigma)$ for some distance function d on timed traces.

In untimed settings, the problem involves identifying the model run(s) that maximize the cost over the log, i.e. identifying series of moves (insertions or deletions) with the highest cost. This untimed version of the problem has been explored in [22]. However, the problem becomes more complex with timed traces, as the challenge lies in finding model runs that deviate significantly from the observed traces in both action labels and timestamps, necessitating a distance metric that covers both aspects. To eventually approach the GTAAP, one crucial yet unexplored step is to find anti-alignments only for the timed part of a trace. Consequently, the distance functions we consider to find anti-alignments are initially solely based on the timestamps of the traces (e.g. Stamp-Only d_t and Delay-Only d_θ distances as defined previously). Thus, the problem is reduced as follows.

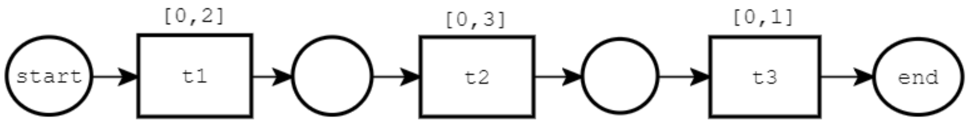
Definition 12 (Purely Timed Anti-Alignment Problem (PTAAP)). Given a Time Petri Net N and a timed log $L \subseteq \mathcal{L}(N)$, where the timing function for a timed trace $\sigma \in L$ is denoted as γ_σ, find the valid timing function(s) $\gamma \in \Gamma_N$ s.t. $d(\gamma, L) = \max_{\gamma \in \Gamma_N} d(\gamma, L)$, where $d(\gamma, L) = \min_{\sigma \in L} d(\gamma_\sigma, \gamma)$ for some distance d on timing functions.

In this paper, we solve the PTAAP, considering the Stamp-Only d_t and Delay-Only d_θ distances, specifically for ATMGs. These restictions are motivated by two main factors.
First, solving the GTAAP would need an approach that incorporates additional constraints to ensure mutual coherence between Untimed Anti-Alignments (UAAs) and Purely Timed Anti-Alignments (PTAAs) while maximizing a combined distance metric (e.g. weighted sum of activity-based and timestamp-based distances). In fact, the GTAAP cannot be effectively decomposed into two independent sub-problems (finding the furthest UAA and PTAA separately), because the two solutions may not be consistent with each other. For example, the UAA may not follow the temporal constraints derived from the PTAA in models that allow for parallelism. A naive approach, such as first determining the UAA and then assigning a compliant timestamp sequence that maximizes the distance from the log, risks overemphasizing one aspect (e.g., activities) at the expense of the other (e.g., timestamps), resulting in biased or suboptimal outcomes. In previous attempts to define edit distances between timed words, e.g. in [23], the edit distance is indeed computed by first identifying an optimal sequence of edit

operations that aligns the untimed parts of the words and then by minimizing the distance between the corresponding timestamp sequences.
Secondly, in [19], the Stamp-Only and Delay-Only distances were defined only for timing functions over the same causal process, i.e., for identical untimed runs on the untimed version of a TPN. This ensures meaningful comparisons of timestamps, i.e. by computing their distances only if related to the same activities. While this constraint does not pose issues when searching for Purely Timed Alignments, where the search space is confined to the timing functions allowed by a single trace's constraints, searching for Purely Timed Anti-Alignments instead requires exploring all timing functions allowed by the model. Consequently, it is necessary to restrict the class of models to those that only produce traces with the same activity information.

We now provide examples of problem instances and their solutions to illustrate the problem and expected behaviors.

Example 9. Suppose we have the simple ATMG N below, which models a simple sequential process. Note that traces will be considered just for their timestamp information, i.e. as points in a space, as described previously.



We hereby present some possible logs accepted by this model and the corresponding solution to the PTAAP, using the Stamp-Only distance as it is the most intuitive and does not require any transformation to the traces.

Event Log 1: $L = \{\sigma_m\} = \{(0,0,0)\}$
Suppose we have a log with only one point σ_m, the minimal point of the model, obtained by executing every transition at the earliest possible time (σ_m s.t. $\forall t_i \in T, f_{\gamma_{\sigma_m}}(i) = Eft(t_i)$). As we seek a point accepted by N that is farthest from the log (i.e., from σ_m), it is clear that the solution to the PTAAP is the maximal point $\sigma_M = (2,5,6)$, obtained by executing every transition at the latest possible time (σ_M such that $\forall t_i \in T, f_{\gamma_{\sigma_M}}(i) = Lft(t_i)$).

Event Log 2: $L = \{\sigma_m, \sigma_M\} = \{(0,0,0),(2,5,6)\}$
Suppose we have a log with two points, the minimal and maximal points. We seek a point accepted by N that is most distant from the log, i.e. from both points simultaneously. This point will be "between" the two, with half the maximal distance in the space. For example, a solution is $\sigma_s = (1, 2.5, 3)$, where $d_t(\sigma_s, \sigma_m) = d_t(\sigma_s, \sigma_M) = \frac{d_t(\sigma_m,\sigma_M)}{2} = 6.5$. Any point accepted by N that is farther from one log point will be closer to the other, reducing the distance from the whole log and therefore non-optimal.

Event Log 3: $L = \{\sigma_1, \sigma_2, \sigma_3\} = \{(0,0,0),(0,1,2),(1,4,5)\}$
Suppose we have a log consisting of the three points above. The optimal solution could be in the set of equidistant points for each pair of neighboring log points, or on the edges of the search space, i.e. either "between" σ_1 and σ_2, "between" σ_2 and σ_3, or at the maximal point $\sigma_M = (2,5,6)$. The latter is the furthest point from σ_3 without getting closer to σ_2.

Given that $d_t(\sigma_1, \sigma_2) = 3$ is smaller than $d_t(\sigma_2, \sigma_3) = 7$, we can discard the first option. Since $d_t(\sigma_3, \sigma_M) = 3$ is less than half the distance between σ_2 and σ_3, the optimal point is the one equidistant from σ_2 and σ_3, i.e. the point $\sigma_s = (2, 2.75, 2.75)$, with $d_t(\sigma_s, \sigma_2) = d_t(\sigma_s, \sigma_3) = 4.5$. Notably, for the first dimension (activity), the optimal timestamp is greater than those for both σ_2 and σ_3 ($2 > 1 > 0$), as it maximizes the distance from both traces but still within the constraints of the model. However, decisions of this kind are not always as straightforward, as the timestamp taken by a transition executed early then influences the timestamps of the following events. Therefore, a dimension-wise optimization does not necessarily guarantee the best (farthest) timestamp sequence.

For ATMGs allowing for parallelism, the timestamps taken by parallel activities will not influence each other, and only their maximal timestamp will have an impact on the timestamps of the rest of the trace.

For the Delay-Only distance, instead, dimension-wise optimization is possible since timestamp sequences are considered as their flow function form, i.e. the value taken for an activity does not influence the search space for the following.

3.1. Complexity Considerations

The PTAAP can be seen as a version of the Largest Empty Sphere problem [24], where the question is to find a hypersphere of largest radius whose interior does not contain any of a finite set of points (in a d-dimensional space) given as input; or equivalently, to find a point (the center of the sphere) which maximizes its distance to the set of input points, where the distance to the set of points is understood as the distance to the closest point of the set. The search space is usually delimited by bounds, e.g. the convex hull of the input points.

In our case, the input points are the time sequences in the log, their dimension is the length of the traces, and the distance is Manhattan (or L^1) distance.

The Largest Empty Sphere problem has been studied, specially in small dimensions where it can be solved with good complexity (typically $n\log(n)$ in dimension 2 with the Euclidian distance) using techniques based on Voronoi diagrams. The complexity grows exponentially with the dimension [25]. In higher dimensions, the Largest Empty Sphere becomes indeed a non-convex optimization problem (for our case, with the Manhattan distance, one sees easily the non-convexity coming from the absolute values in the definition of the distance).

For these reasons, we propose to encode our problem as a non-convex optimization problem and rely on the performances of the solvers.

4. Solving the Purely Timed Anti-Alignment Problem

The Purely Timed Anti-Alignment Problem can be viewed as an optimization problem, maximizing the chosen distance from a fixed set of observed traces L, over the set of valid timestamp series $S(N)$ for the model N. The characteristics of such a search space $S(N)$ change depending on whether the chosen ATMG allows for parallelism or not.

Therefore, the optimization problem at hand, given an ATMG N, with $|T| = d$, a timed log $L \subseteq \mathcal{L}(N)$ and using the Stamp-Only distance function, is the following:

$$\begin{aligned} \text{maximize} \quad & \min_{\sigma\in L} \sum_{i=1}^{d} |\gamma_\sigma(i) - x_i| \\ \text{subject to} \quad & x \in S(N) \end{aligned}$$

As for the search space $S(N)$, it results that, given an ATMG N with $|T| = d$ equipped with a certain ordering function ord, each component x_i of a d-dimensional point (timestamp sequence) $x = (x_1, \dots, x_d) \in S(N) \subset \mathbb{R}^d_+$ can take values in the intervals defined by the bounding function:

$$B(i) = \begin{cases} SI(t_i) & {}^\bullet{}_T\, t_i = \emptyset \\ \max_{j\in {}^\bullet{}_T\, i} x_j + SI(t(i)) & otherwise \end{cases}$$

where the operation $+$ between an interval and a number is defined here as $[a_1, a_2]+\alpha = [a_1+\alpha, a_2+\alpha]$. As the function returns an interval, for readability we will express its lower bound and upper bound as $B_l(i)$ and $B_u(i)$ respectively. In practice, this constraint ensures that the execution of transition t_i occurs after the completion of its latest parent transition, if any, by an interval specified by the lower bound $Eft(t)$ and upper bound $Lft(t)$.

When considering the Delay-Only distance instead, there are two differences. First, timestamp sequences are mapped to their flow function values (so that the distance d_θ can be represented in the form of the Manhattan distance) and the search space $S(N)$ will be much simpler, as the boundaries for each dimension can now only range within the interval defined for the corresponding transition.

Therefore, the problem becomes the following:

$$\begin{aligned} \text{maximize} \quad & \min_{\sigma \in L} \sum_{i=1}^{d} |f_{\gamma_\sigma}(i) - x_i| \\ \text{subject to} \quad & Eft(t_i) \leq x_i \leq Lft(t_i),\ \forall i \in \{1, ..., d\} \end{aligned}$$

The approach used to solve such optimization problem involves Linear Programming (LP). Linear Programming is the process of minimizing/maximizing a linear objective function subject to a finite number of linear equality and inequality constraints [26]. However, to describe the characteristics of many optimization problems it is sometimes necessary to adopt a set of non-linear terms. Finding an optimal solution for a non-linear problem in acceptable computational time is still a big challenge in the optimization theory [27]: instead, in comparison, linear forms require significantly less computational time [28]. Therefore, one of the techniques often used to solve optimization problems with non-linear terms consists in replacing the latter with equivalent linear formulations, at the cost of often increasing the size of the problem [29]. This is the case also for the Purely Timed Anti-Alignment Problem. Specifically, equivalent linear formulations are needed to express:

- the fact that this is a max-min problem, i.e. a problem where the goal is to maximize the minimum of the objective function for all potential scenarios;
- the absolute value contained in the objective function;
- the maximum function contained in the definition of the search space $S(N)$ for the distance d_t.

We use established techniques [29] to retrieve linear formulations for these functions. As the structure of an ATMG can differ, we also found a simpler reformulation (in terms of number of variables and constraints) for models that allow only for sequential events (no parallelism). An in-depth explanation of the different linear reformulations obtained is contained in the Appendix.

5. Implementation and Experimental Analysis

We implemented the Linear Programming solution in Python[1] using PuLP [30], a modeling library that uses Python syntax for constraints and supports various solvers, including the default CBC solver [31] that we used. The experiments were executed on Google Colaboratory [32], using a virtual environment with an Intel Xeon CPU @ 2.20GHz and 12 GB of available RAM.

To assess our implementation's practicality, we generated various problem examples, varying both the model dimension (d transitions) and log cardinality (n traces). The chosen parameter values are:

- Cardinality $|L|$: {10, 100, 1000}
- Dimension $|T|$: {5, 15, 30}

These values reflect realistic scenarios, as process models typically do not reach extremely high complexity in the number of transitions, but logs often contain many traces and events.

As we found different linear formulations depending on whether parallelism is allowed or not in the model, we tested them separately to study the difference in efficiency. We therefore considered two types of models, sequential ATMGs (as in Example 9) and ATMGs with parallelism: since for the second type different structures are allowed, the chosen one is shown in Figure 3.

We automatically generated models and logs of increasing complexity as follows:

[1]Available at: `github.com/StefanoBavaro/TimedAntiAlignments`

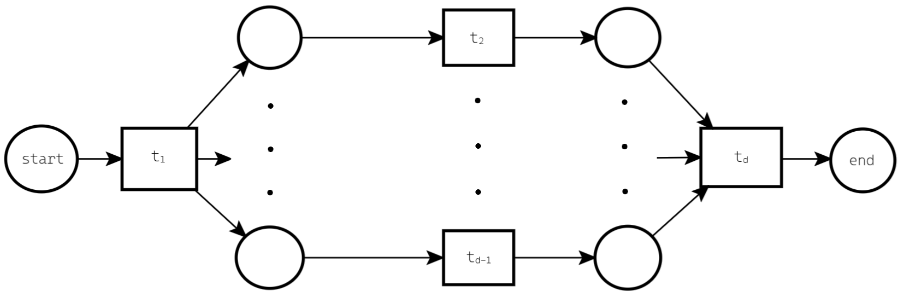


Figure 3: Structure of ATMGs with parallelism used in experiments: given d dimensions, $d-2$ transitions are in parallel.

- Given a model dimension d, an ATMG with d transitions is generated, following the structures mentioned above. For each transition t, the timestamp interval $SI(t)$ is set using a random process. The lower bound is a random number between 0 and 5, rounded to two decimal places ($Eft(t_i) \sim \text{Uniform}(0,5)$). The upper bound is calculated by adding another random number between 0 and 5, also rounded, to the lower bound ($Lft(t_i) = Eft(t_i) + \alpha, \alpha \sim \text{Uniform}(0,5)$). This creates intervals with variable ranges, better representing real time relationships.
- Given the generated model N and some cardinality n, a log with n points is created by randomly assigning valid values (rounded to two decimal places) for each dimension, ensuring each point is accepted by N.

For every combination of dimension, cardinality and type of ATMGs, we compute the time needed for the solver to find the optimal point, the number of variables (#V) and the number of constraints (#C). As for the last two measures, we can identify three cases by observing the linear formulations:

1. Sequential ATMGs for both distances and ATMGs allowing parallelism and with the structure in Fig 3 for distance d_θ, that share the number of constraints and variables, i.e. $\#V = 1+3(n\times d)+d$ and $\#C = n + 3(n \times d) + 2d$.
2. ATMGs allowing parallelism with the structure in Fig 3 with distance d_t, for which $\#V = 1 + 3(n \times d) + 4d - 5$ and $\#C = 1 + 3(n \times d) + 8d - 12$.
3. ATMGs allowing parallelism with no specific structure. In this case, the determining factors are the amount of non-starting transitions $|I_P|$ (with $I_P = \{i \in \{1, ..., d\} \mid {}^\bullet{}_T\, t_i \neq \emptyset\}$) and the number of parent transitions for each of them. We computed the lower and upper bounds for such metrics in this case, as the exact value depends on these variable factors, obtaining that $1+3(n\times d)+d \leq \#V \leq 1+3(n\times d)+d+|I_P|+|I_P|^2$ and that $n+3(n\times d)+2d \leq \#C \leq n+3(n\times d)+2d+3|I_P|^2$.

Note that, for the metric $\#C$, the amounts reported are the ones computed by our implementation, that do not consider variables being binary or positive as constraints as they can just be declared as such.

Results (the elapsed time is expressed in seconds, rounded to the nearest integer) for the two structures of models are reported in Tables 1 and 2.

As expected, time requirements significantly increase with the growth of variables and constraints. More time is usually needed to solve the problem when considering the Delay-Only distance for Sequential ATMGs: as the amount of variables and constraints do not change for the two distances, a possible factor might be the change in magnitude of the search space. Instead, for the second type of ATMGs, it usually takes less time to solve the problem when considering the Delay-Only distance: this can be explained by the lower amount of variables and constraints for the two distance settings. However, to interpet these results, it is necessary to consider the high variability in the performance of MIP solvers [33], influenced by factors like variable declaration order and search space shape. Hence, the reported time needed to solve different instances of the PTAAP offers insights into the solver behavior

Table 1
Elapsed time (in seconds), #V and #C for Sequential ATMGs

Card. $\|L\|$	Dist.	Dimension $\|T\|$								
		5			15			30		
		Time	#V	#C	Time	#V	#C	Time	#V	#C
10	d_t	1	156	170	7	466	490	7	931	970
	d_θ	1			5			4		
100	d_t	54	1506	1610	69	4516	4630	449	9031	9160
	d_θ	52			575			>6h		
1000	d_t	2459	15006	16010	5833	45016	46030	>6h	90031	91060
	d_θ	7605			>6h			>6h		

Table 2
Elapsed time (in seconds), #V and #C for ATMGs as Figure 3

Card. $\|L\|$	Dist.	Dimension $\|T\|$								
		5			15			30		
		Time	#V	#C	Time	#V	#C	Time	#V	#C
10	d_t	1	166	188	20	506	568	164	1116	1138
	d_θ	1	156	170	4	466	490	4	931	970
100	d_t	41	1516	1628	2639	4556	4708	>6h	9116	9229
	d_θ	35	1506	1610	1177	4516	4630	>6h	9031	9160
1000	d_t	10492	15016	16028	>6h	45056	45109	>6h	90116	90229
	d_θ	10264	15006	16010	>6h	45016	46030	>6h	90031	91060

across different problem settings, but cannot precisely indicate solve times for further similar instances with the same controlled parameters, as also the specific time constraints and the log provided can have an impact.

To provide a baseline for comparison, we also implemented a Brute-Force solver and tested it against the LP solver. The brute force approach builds the search space recursively by splitting the range of possible timestamps into evenly spaced values for each dimension. Each value is then used to compute the timestamps for the following dimensions in the point, until the final dimension is reached and the complete point is added to the search space. The computational complexity, both in time and space, increases with the granularity of the timestamp intervals: finer splits improve accuracy but require more computational resources. Every point in such search space is a possible solution: the solver computes the distance from the log for each point and selects the maximum as the optimal distance. The results (execution time in seconds, rounded to the nearest integer) of both solvers for both types of ATMGs are presented in Tables 3. The interval subdivision value for the brute-force search space is 5.

Table 3
Comparison of execution times (in seconds) between LP and Brute-Force (BF) Solvers for Sequential ATMGs (left) and ATMGs as Figure 3 (right).

Card. $\|L\|$	d	Dimension $\|T\|$					
		5		10		11	
		LP	BF	LP	BF	LP	BF
10	d_t	1	0	2	234	4	1185
	d_θ	**1**	**0**	1	191	2	1024
100	d_t	25	0	288	306	190	1659
	d_θ	54	0	289	267	197	1412

Card. $\|L\|$	d	Dimension $\|T\|$					
		5		10		11	
		LP	BF	LP	BF	LP	BF
10	d_t	2	0	**13**	**364**	**6**	**1856**
	d_θ	1	0	2	185	**1**	**759**
100	d_t	60	0	1266	425	**2084**	**2192**
	d_θ	**50**	**0**	**480**	**205**	515	1060

The brute-force approach performs significantly worse for increasing dimensions, as evidenced by the steep growth in execution time in the models with 10 and 11 dimensions. In fact, such explosion, also in terms of RAM requirements, prevented the exploration of the BF solver beyond 11 dimensions. It can also be noticed that execution time doesn't scale as drastically with cardinality as with dimensions: this is because computing distances is not as computationally intensive as building the search space. Finally, it is worth noticing that in some cases (in bold), the distance found by the LP solver was greater (therefore better) by some decimals than the one found through the BF approach: this might be due to insufficient discretization of the BF search space, which, if increased, would require even more time and space resources.

6. Conclusion and Future Work

In this paper, we introduced time-aware anti-alignments and solved the Purely Timed Anti-Alignment Problem for Acyclic Time Marked Graphs using the Stamp-Only and Delay-Only distances. Our LP implementation of the problem, reformulated as a Mixed-Integer Programming problem, notably outperformed a brute-force solver as model complexity increased. Alongside [19] and [20], our work contributes to advancing conformance checking in time-aware process mining. Solving the Purely Timed Anti-Alignment Problem is key to potentially defining new time-aware metrics using timed anti-alignments, e.g. for precision or generalization as done in untimed settings. Future research could explore solving the problem for other distance functions, such as the Mixed-Moves distance [19]. The work can also be extended to models that include loops or choice: this would probably require new ways of computing the Stamp-Only and Delay-Only distances, so that to compare timestamps related to different sequences of activities. Finally, this is another step towards tackling the General Timed Anti-Alignment Problem, which would require additional strategies to balance the contributions of activity-based and timestamp-based distances while ensuring coherence between the purely-timed and untimed optimal anti-alignments.

Declaration on Generative AI

During the preparation of this work, the authors did not use any GenAI tool.

References

[1] W. Van Der Aalst, W. van der Aalst, Data science in action, Springer, 2016.

[2] N. Tax, X. Lu, N. Sidorova, D. Fahland, W. M. P. van der Aalst, The imprecisions of precision measures in process mining, Inf. Process. Lett. 135 (2018) 1–8. URL: https://doi.org/10.1016/j.ipl.2018.01.013. doi:10.1016/J.IPL.2018.01.013.

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

[4] T. Chatain, M. Boltenhagen, J. Carmona, Anti-alignments - measuring the precision of process models and event logs, Inf. Syst. 98 (2021) 101708. doi:10.1016/j.is.2020.101708.

[5] T. Chatain, J. Carmona, Anti-alignments in conformance checking – the dark side of process models, in: F. Kordon, D. Moldt (Eds.), Application and Theory of Petri Nets and Concurrency, Springer International Publishing, Cham, 2016, pp. 240–258.

[6] B. F. van Dongen, J. Carmona, T. Chatain, A unified approach for measuring precision and generalization based on anti-alignments, in: International conference on business process management, Springer, 2016, pp. 39–56.

[7] M. Boltenhagen, T. Chatain, J. Carmona, Optimized sat encoding of conformance checking artefacts, Computing 103 (2021) 29–50. URL: https://doi.org/10.1007/s00607-020-00831-8. doi:10.1007/s00607-020-00831-8.

[8] M. Boltenhagen, T. Chatain, J. Carmona, An a*-algorithm for computing discounted anti-alignments in process mining, in: 2021 3rd International Conference on Process Mining (ICPM), IEEE, 2021, pp. 25–31.

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

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

[11] W. M. P. van der Aalst, 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, volume 436 of *Lecture Notes in Business Information Processing*, Springer, 2021, pp. 99–110. doi:10.1007/978-3-030-94343-1_8.

[12] P. Merlin, A study of the recoverability of computer systems, Ph.D. Thesis, Computer Science Dept., University of California (1974).

[13] B. Berthomieu, M. Menasche, An enumerative approach for analyzing time petri nets, in: IFIP 9th World Computer Congress, 1983.

[14] T. Aura, J. Lilius, A causal semantics for time petri nets, Theoretical Computer Science 243 (2000) 409–447. URL: https://www.sciencedirect.com/science/article/pii/S0304397599001140. doi:https://doi.org/10.1016/S0304-3975(99)00114-0.

[15] T. Chatain, C. Jard, Back in time petri nets, in: V. Braberman, L. Fribourg (Eds.), Formal Modeling and Analysis of Timed Systems, Springer Berlin Heidelberg, 2013.

[16] W. L. J. Lee, H. Verbeek, J. Munoz-Gama, W. M. van der Aalst, M. Sepúlveda, Recomposing conformance: Closing the circle on decomposed alignment-based conformance checking in process mining, Information Sciences 466 (2018) 55–91. URL: https://www.sciencedirect.com/science/article/pii/S0020025518305413. doi:https://doi.org/10.1016/j.ins.2018.07.026.

[17] D. Reißner, R. Conforti, M. Dumas, M. La Rosa, A. Armas-Cervantes, Scalable conformance checking of business processes, in: H. Panetto, C. Debruyne, W. Gaaloul, M. Papazoglou, A. Paschke, C. A. Ardagna, R. Meersman (Eds.), On the Move to Meaningful Internet Systems. OTM 2017 Conferences, Springer International Publishing, Cham, 2017, pp. 607–627.

[18] F. Taymouri, J. Carmona, Computing alignments of well-formed process models using local search, ACM Trans. Softw. Eng. Methodol. 29 (2020). URL: https://doi.org/10.1145/3394056. doi:10.1145/3394056.

[19] T. Chatain, N. Rino, Timed alignments, in: 4th International Conference on Process Mining, ICPM 2022, IEEE, 2022, pp. 112–119. doi:10.1109/ICPM57379.2022.9980687.

[20] N. Rino, T. Chatain, Timed alignments with mixed moves, in: Business Process Management Workshops, Springer Nature Switzerland, 2024, pp. 186–197.

[21] W. M. P. van der Aalst, L. Santos, May i take your order?, in: A. Marrella, B. Weber (Eds.), Business Process Management Workshops, Springer International Publishing, 2022, pp. 99–110.

[22] T. Chatain, M. Boltenhagen, J. Carmona, Anti-alignments—measuring the precision of process models and event logs, Information Systems 98 (2021) 101708. doi:https://doi.org/10.1016/j.is.2020.101708.

[23] K. Chatterjee, R. Ibsen-Jensen, R. Majumdar, Edit distance for timed automata, HSCC '14, Association for Computing Machinery, New York, NY, USA, 2014, p. 303–312. URL: https://doi.org/10.1145/2562059.2562141. doi:10.1145/2562059.2562141.

[24] G. T. Toussaint, Computing largest empty circles with location constraints, Int. J. Parallel Program. 12 (1983) 347–358.

[25] J. Shi, Y. Yoshitsugu, A D.C. Approach to the Largest Empty Sphere Problem in Higher Dimension, Springer US, Boston, MA, 1996, pp. 395–411. doi:10.1007/978-1-4613-3437-8_25.

[26] H. Karloff, Linear programming, Springer Science & Business Media, 2008.

[27] A. M. Fathollahi-Fard, L. Woodward, O. Akhrif, Sustainable distributed permutation flow-shop scheduling model based on a triple bottom line concept, Journal of Industrial Information Integration 24 (2021) 100233. doi:https://doi.org/10.1016/j.jii.2021.100233.

[28] G. Pauer, Árpád Török, Binary integer modeling of the traffic flow optimization problem, in the case of an autonomous transportation system, Operations Research Letters 49 (2021) 136–143. doi:https://doi.org/10.1016/j.orl.2020.12.004.

[29] M. Asghari, A. M. Fathollahi-Fard, S. Mirzapour Al-E-Hashem, M. A. Dulebenets, Transformation and linearization techniques in optimization: A state-of-the-art survey, Mathematics 10 (2022) 283.

[30] S. Mitchell, M. OSullivan, I. Dunning, Pulp: a linear programming toolkit for python, The University of Auckland, Auckland, New Zealand 65 (2011).
[31] J. Forrest, R. Lougee-Heimer, Cbc user guide, in: Emerging theory, methods, and applications, INFORMS, 2005, pp. 257–277.
[32] Google, Google colaboratory, 2024. URL: https://colab.research.google.com/.
[33] A. Lodi, A. Tramontani, Performance variability in mixed-integer programming, in: Theory driven by influential applications, INFORMS, 2013, pp. 1–12.

A. Appendix

We start by presenting the reformulation for the Stamp-Only distance, which is the most straightforward case as no transformation has to be performed on traces. We later present how to adapt the solution to the Delay-Only Distance. The initial form of the problem is then:

$$\begin{aligned} \text{maximize} \quad & \min_{\sigma \in L} \sum_{i=1}^{d} |\gamma_\sigma(i) - x_i| \\ \text{subject to} \quad & x \in S(N) \end{aligned}$$

Note that in the further formulations, to ease notation, we will identify the timing function related to a trace σ_k as γ_k rather than γ_{σ_k}.
In the following sections, we detail every reformulation needed, showing how each non-linear constraint can be transformed into a linear form.

A.1. Maximin reformulation

The problem at hand is a Maximin problem, i.e. a problem where the goal is to maximize the minimum of the objective function for all potential scenarios. In fact, we aim at maximizing the distance with the timed log, which by definition is the minimum distance from any point in the log. We therefore introduce a new objective function, represented by the only variable z. The decision variables x_i are now used to set z, through multiple linear inequalities, as lower than the distance from any point in the log L. This way, an upper bound is set for z so that, by maximizing it, the minimum distance from any point in the log is necessarily obtained. The first transformation is therefore the following:

$$\begin{aligned} \text{maximize} \quad & z \\ \text{subject to} \quad & z \leq \sum_{i=1}^{d} |\gamma_k(i) - x_i|, \ \forall k \in \{1, ..., n\} \\ & x \in S(N) \end{aligned}$$

where n is the cardinality of the log L.

A.2. Absolute Values Reformulation

To reformulate the absolute value function, it is needed to express the values of every absolute difference $|\gamma_k(i) - x_i|$. To do so, we introduce new positive variables $\text{diff}^+_{k,i}$ and $\text{diff}^-_{k,i}$ for each element of the log and each dimension of the solution space, i.e. $\forall i \in I_D = \{1, ..., d\}$ and $\forall k \in I_L = \{1, ..., n\}$. By using binary variables $b_{k,i}$, we also set one of the two variables $\text{diff}^+_{k,i}$ and $\text{diff}^-_{k,i}$ to be 0 (Constraints 3 and 4). The constant M is used to prevent any of these variables from becoming infinite: we set it as the maximum possible distance between two points in the search space, i.e. the distance between the minimal trace σ_m and the maximal trace σ_M, $d_t(\sigma_m, \sigma_M)$. With these constraints, we split $\gamma_k(i) - x_i$ into the possibly positive and negative results (Constraint 2). If $\gamma_k(i) - x_i$ is positive, its value will be taken by $\text{diff}^+_{k,i}$; vice versa, if $\gamma_k(i) - x_i$ is negative, the absolute value will be taken by $\text{diff}^-_{k,i}$. Therefore, it follows that $|\gamma_k(i) - x_i| = \text{diff}^+_{k,i} + \text{diff}^-_{k,i}$, which is indeed substituted in Constraint 1.

The reformulated linear programming problem incorporating these transformations is then:

$$\begin{aligned} \text{maximize} \quad & z \\ \text{subject to} \quad & \end{aligned}$$

$$z \leq \sum_{i=1}^{d} \text{diff}^+_{k,i} + \text{diff}^-_{k,i}, \quad \forall k \in I_L \tag{1}$$

$$\text{diff}^+_{k,i} - \text{diff}^-_{k,i} = \gamma_k(i) - x_i, \quad \forall i \in I_D, \forall k \in I_L \tag{2}$$

$$0 \leq \text{diff}^{+}_{k,i} \leq M \cdot b_{k,i}, \quad \forall i \in I_D, \forall k \in I_L \tag{3}$$
$$0 \leq \text{diff}^{-}_{k,i} \leq M \cdot (1 - b_{k,i}), \quad \forall i \in I_D, \forall k \in I_L \tag{4}$$
$$b_{k,i} \in \{0, 1\}, \quad \forall i \in I_D, \forall k \in I_L \tag{5}$$
$$x \in S(N) \tag{6}$$

This reformulation transforms the original problem with absolute value functions into a Mixed Integer Programming (MIP) problem (due to the presence of binary variables, considered as integers) that can be efficiently solved using linear programming solvers.

A.3. Formulation of Search Spaces

Finally, we specify the constraints of the search space $S(N)$, i.e. the constraints for each decision variable x_i. For sequential ATMGs, the search space can be further simplified than the one presented in the paper. Given a sequential ATMG N with $|T| = d$ equipped with a standard ordering function ord that trivially follows the causal relationships of the transitions, each component x_i of a d-dimensional point (timestamp sequence) $x = (x_1, \ldots, x_d) \in S(N) \subset \mathbb{R}^d_+$ can take values in the intervals defined by the bounding function:

$$B(i) = \begin{cases} SI(t_i) & {}^{\bullet}{}_{T}\, t_i = \emptyset \\ x_{i-1} + SI(t_i) & otherwise \end{cases}$$

where the operation $+$ between an interval and a number is defined here as $[a_1, a_2] + \alpha = [a_1 + \alpha, a_2 + \alpha]$. As the function returns an interval, for readability we will express its lower bound and upper bound as $B_l(i)$ and $B_u(i)$ respectively. In practice, these constraints ensure that the execution of transition t_i occurs after the completion of the preceding transition, if any, by an interval specified by the lower bound $Eft(t)$ and upper bound $Lft(t)$.

With this function, we can now explict (for sequential ATMGs) the constraint $x \in S(N)$ that we momentarily left out in the previous formulations in the form of the new Constraint 6. Therefore, the problem will now be:

$$\text{maximize} \quad z$$
$$\text{subject to}$$
$$z \leq \sum_{i=1}^{d} \text{diff}^{+}_{k,i} + \text{diff}^{-}_{k,i}, \quad \forall k \in I_L \tag{1}$$
$$\text{diff}^{+}_{k,i} - \text{diff}^{-}_{k,i} = \gamma_k(i) - x_i, \quad \forall i \in I_D, \forall k \in I_L \tag{2}$$
$$0 \leq \text{diff}^{+}_{k,i} \leq M \cdot b_{k,i}, \quad \forall i \in I_D, \forall k \in I_L \tag{3}$$
$$0 \leq \text{diff}^{-}_{k,i} \leq M \cdot (1 - b_{k,i}), \quad \forall i \in I_D, \forall k \in I_L \tag{4}$$
$$b_{k,i} \in \{0, 1\}, \quad \forall i \in I_D, \forall k \in I_L \tag{5}$$
$$B_l(i) \leq x_i \leq B_u(i), \ \forall i \in \{1, ..., d\} \tag{6}$$

When dealing with ATMGs with paralellism, the search space $S(N)$ changes as defined previously (Section 4). Additional complexity arises as now transitions can have multiple parent transitions: therefore, it is needed to retrieve only the maximum value among their timestamps. For transitions without parents, no reformulation is needed (Constraint 6) and the constraint is the same as for sequential ATMGs. Instead, when a transition t_i has any parent transition (for readability, $I_P = \{i \in \{1, ..., d\} \mid {}^{\bullet}{}_{T}\, t_i \neq \emptyset\}$ is the set of indexes of transitions with any parent transition), we need to reformulate the max function for both inequalities (i.e. both endpoints of the interval).

The inequality $\max_{j\in\bullet_T i} x_j + Eft(t_i) \leq x_i$ can be directly modeled by defining the same for each parent transition t_j instead of just the maximal (Constraint 7): the inequality with the maximum term x_j will be a stricter constraint, therefore dictating the lower bound for x_i.

The second inequality $x_i \leq \max_{j\in\bullet_T i} x_j + Lft(t_i)$ requires additional variables to express the max function[29]. We introduce a variable maxVar$_i$ for every transition t_i to act as a proxy for the maximum timestamp of the transition's parents (Constraint 12). First, the lower bound of every maxVar$_i$ is set as the maximal timestamp of parent transitions (Constraint 8) as done previously. The upper bound, which has to be the same as the lower bound, is set using binary variables $\beta_{i,j}$ whose sum is 1 (Constraint 10). Constraint 9 prevent any maxVar$_i$ from becoming infinite and, as the only assignment of binary variables that make the problem feasible is the one for which only the binary variable related to the maximum of the parents timestamps is set to 1, maxVar$_i$ will take exactly the maximal value of parents transactions.

Finally, the problem will now be:

$$\text{maximize} \quad z$$
$$\text{subject to}$$

$$z \leq \sum_{i=1}^{d} \text{diff}^{+}_{k,i} + \text{diff}^{-}_{k,i}, \quad \forall k \in I_L \tag{1}$$

$$\text{diff}^{+}_{k,i} - \text{diff}^{-}_{k,i} = \gamma_k(i) - x_i, \quad \forall i \in I_D, \forall k \in I_L \tag{2}$$

$$0 \leq \text{diff}^{+}_{k,i} \leq M \cdot b_{k,i}, \quad \forall i \in I_D, \forall k \in I_L \tag{3}$$

$$0 \leq \text{diff}^{-}_{k,i} \leq M \cdot (1 - b_{k,i}), \quad \forall i \in I_D, \forall k \in I_L \tag{4}$$

$$b_{k,i} \in \{0, 1\}, \quad \forall i \in I_D, \forall k \in I_L \tag{5}$$

$$B_l(i) \leq x_i \leq B_u(i), \; \forall i \in I_D \; s.t \; \underset{T}{\bullet} t_i = \emptyset \tag{6}$$

$$Eft(t_i) + x_j \leq x_i, \; \forall i \in I_P, \forall j \in \underset{T}{\bullet} i \tag{7}$$

$$x_j \leq maxVar_i, \; \forall i \in I_P, \forall j \in \underset{T}{\bullet} i \tag{8}$$

$$maxVar_i \leq x_j + M \cdot (1 - \beta_{j,i}), \; \forall i \in I_P, \forall j \in \underset{T}{\bullet} i \tag{9}$$

$$\sum_{j\in\bullet_T i} \beta_{j,i} = 1, \; \forall i \in I_P \tag{10}$$

$$\beta_{j,i} \in \{0, 1\}, \; \forall i \in I_P, \forall j \in \underset{T}{\bullet} i \tag{11}$$

$$x_i \leq Lft(t_i) + maxVar_i, \; \forall i \in I_P \tag{12}$$

A.4. Anti-Alignments for Delay-Only distance

We here present the full reformulation of the PTAAP for ATMGs considering the Delay-Only distance rather than the Stamp-Only distance. The two main main differences are presented already in Section 4, and correspond to Constraints 2 and 6. Therefore, the formulation of such problem is:

$$\text{maximize} \quad z$$
$$\text{subject to}$$

$$z \leq \sum_{i=1}^{d} \text{diff}^{+}_{k,i} + \text{diff}^{-}_{k,i}, \quad \forall k \in I_L \tag{1}$$

$$\text{diff}^{+}_{k,i} - \text{diff}^{-}_{k,i} = f_k(i) - x_i, \quad \forall i \in I_D, \forall k \in I_L \tag{2}$$

$$0 \leq \text{diff}^{+}_{k,i} \leq M \cdot b_{k,i}, \quad \forall i \in I_D, \forall k \in I_L \tag{3}$$

$$0 \leq \text{diff}^{-}_{k,i} \leq M \cdot (1 - b_{k,i}), \quad \forall i \in I_D, \forall k \in I_L \tag{4}$$

$$b_{k,i} \in \{0, 1\}, \quad \forall i \in I_D, \forall k \in I_L \tag{5}$$

$$Eft(t_i) \leq x_i \leq Lft(t_i), \ \forall i \in I_D \tag{6}$$

Due to the transformation of the traces given as input, the optimal point obtained in this case needs to be mapped back to the original timestamp values, i.e. by reverting the effect of the flow function. To do that, one can simply sum the values of the found optimal point according to the transitions relationships. Therefore, given the obtained optimal sequence, which can as well be represented as a timing function γ_{result}, the timing function γ_σ of the non-transformed trace σ is:

$$\gamma_\sigma(i) = \begin{cases} \gamma_{result}(i) & {}^{\bullet}_{T}\, t_i = \emptyset \\ \gamma_{result}(i) + \max_{k \in {}^{\bullet}_{T}\, t_i} \gamma(k) & otherwise \end{cases}$$