

Model Repair and Conformance Checking of Time-Aware Models

Shubh Sharma^{1,2}, and Thomas Chatain²

¹Chennai Mathematical Institute, ²LSV, ENS Paris-Saclay

Abstract

The subject of this paper is to implement model repair 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 the process develop, so does the need for conformance checking techniques that give insightful quality measure, for the purpose of this paper, we use alignments as witness for the model being unfit, we then give algorithms improving the models

Keywords Time Petri Nets, Model Repair, Conformance Checking

1. Introduction

1.a. Process Mining and Model Repair

Process Mining is a family of techniques used to analyze event data in order to understand and improve the system. It studies the system through their event logs and seeks to extract meaningful ways to model the underlying process to understand the system better or to predict its future behaviour (Aalst, 2016).

The first step: *discovery*, in process mining is the discovery of a model one expects to model the system through techniques like machine learning. But the unexplainability of the approaches that ML takes naturally makes one question if the produced model approximates the target system well enough.

This is where *Conformance Checking* comes in. Conformance Checking is a set of techniques used to compare a *process model* with an event log and rate it over some parameters like:

- **Fitness** : Does the model exhibit the behaviours specified in the logs?
- **Precision** : Does the model deviate a lot from the behavior specified in the logs?
- **Generalization** : Does the model correctly predict behavior of the system outside the given logs?
- **Simplicity** : Is the model the simplest model that describes the log accurately?

Once we measure how well the model conforms to the given set of logs, we move to the next step that is *enhancement*: where an existing process model is extended or improved using information about the actual process record in some event log.

Another important part of Process Mining is Performance Analysis where the goal is to analyze and improve the execution of the model to use less time and resources and improve its performance data.

Model Repair is a special case of **Enhancement** that deals with improving the model to more accurately fit any discrepancies due to events in the system that happen after the model has been constructed. The improvement metrics are usually one of the 4 mentioned above, this paper focuses on the fitness of the model.

1.b. Time Aware Models

Process Models are represented by formal objects. Petri-Nets offer a graphical means to represent concurrent systems and a formal semantics for their execution. The setup is similar to the one in (Chatain & Rino, 2022) where an event is represented by a letter from a finite alphabet (a set of possible discrete events). Logs are represented by the set of timed words over the alphabet, which is a list of events along with the timestamps on which the event occurred. The notion of distance between words, which will give our conformance metric, will be similar to the Levenshtein's edit distance where we find the quickest way to go from one timed-word to another using a given set of edit actions.

We will be using Time Petri Nets, which are a variant of Petri Nets that can check the duration it takes to fire a transition once it's enabled, restricting the set of timed-words it accepts, this can be used to construct relationships and constraints between events and the timestamps at which they can be taken as seen in the logs.

2. Preliminaries

2.a. Time Petri Nets

We represent an event as pairs (a, t) where $a \in \Sigma$ is the action and t denotes the time at which the 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 part of the word, i.e. project it on to the time component leaving a word in $(\mathbb{R}^+)^*$.

The Process Model we use here is a Time Petri Net.

Definition 2: A *Labelled Timed Petri Net* (or TPN) is a tuple $\mathcal{N} = \langle P, T, F, SI, \Sigma, \lambda, M_0, M_f \rangle$ where

- P and T are disjoint sets of places and transitions respectively.
- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation.
- $SI : T \rightarrow \mathbb{I}$ is the static interval function, $SI(t) = (st(t), en(t))$ where
 - $st(t)$ is the smallest valid time interval from the enabling of t to its firing.
 - $en(t)$ is the largest valid time interval from the enabling of t to its firing.
- $\lambda : T \rightarrow \Sigma$ is a labelling function for the transition with actions from the action set Σ
- M_0 and $M_f : P \rightarrow \mathbb{N}$ are the initial and final markings.

Given a transition $t \in T$ we define

- The Pre-set of t from as $\bullet t = \{p \in P \mid (p, t) \in F\}$
- The Post-set as $t^\bullet = \{p \in P \mid (t, p) \in F\}$ (we define pre-set and post-set if places similarly).
- We say that a transition t is enabled at a marking M if $\forall p \in \bullet t, M(p) > 0$
- The set of all enabled transitions in M is given by $Enabled(M) = \{t \in T \mid t \text{ is enabled in } M\}$.

Definition 3: The *state* (or *configuration*) of a TPN $\mathcal{N} = \langle P, T, F, SI, \Sigma, \lambda, M_0, M_f \rangle$ is a pair $S = (M, I)$ where M is a marking and $I : \text{Enabled}(M) \rightarrow \mathbb{R}^+$ is the clock function keeping track of the time since each transition was enabled. We set the initial state to be $(M_0, \mathbf{0})$ where $\mathbf{0}$ is the zero-function.

A transition t is said to be **fireable** after a delay θ from a state $S = (M, I)$ if t is enabled in M and $I(t) + \theta \in SI(t)$

The update to the marking and time function are defined below:

Definition 4: (Firing Rule) When a transition t fires after a delay θ from state $S = (M, I)$, the new state $S' = (M', I')$ is given as follows:

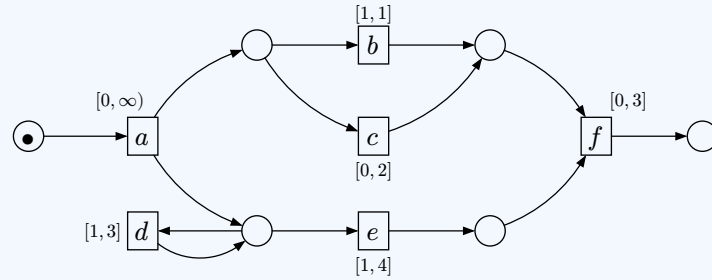
$$M' = (M \setminus \bullet t) \cup t \bullet$$

$$I'(t) = \begin{cases} I(t) + \theta & \text{If } t \in \text{Enabled}(M') \\ 0 & \text{If } t \in \text{Enabled}(M') \text{ and } t \notin \text{Enabled}(M) \\ \text{Undefined} & \text{Otherwise} \end{cases}$$

This is also denoted as $S[t]S'$

A valid execution of the model starts at the initial marking M_0 , fires a sequence of transitions and ends at the final marking M_f .

Example 1: Consider the following example of a Time Petri Net N :



One possible execution of N would be for the firing sequence

$$w = (a, 1)(b, 2)(d, 3)(e, 4)(f, 5)$$

The initial marking only has a enabled and firing a moves the token to places that enable b, c, d and e . Then transition b at time 2 is fired, which puts a token in one of the places consumed by transition f . at time 3, d is fired followed by e at 4 and now f is enabled and is fired after a second of wait.

Now we can define the language of the Time Petri Net as follows

Definition 5: A word $w = a_0 a_1 \dots a_n \in \Sigma^*$ is in the *language of the Labelled Time Petri Net* $\mathcal{L}(\mathcal{N})$ if there is a fireable sequence of transitions $(t_0, t_1 \dots t_n)$ such that $\lambda(t_0, t_1 \dots t_n) = w$ and if the sequence of transitions is taken from the initial state M_0 , it will end at the final transition M_f .

$$(M_0, \mathbf{0})[t_0 t_1 \dots t_n](M_f, I)$$

2.b. Helper Definitions

To help with defining some of the things that will be used further, we will use **causal nets** which are like unfoldings of a Petri Net that will make definitions and procedures about walking through the Petri Net easier.

Definition 6: A *Causal Net* $CN := \langle B, E, G \rangle$ is a finitary, acyclic net where

$$\forall b \in B : |b^\bullet| \leq 1 \wedge |\bullet b| \leq 1$$

This definition reads as “A Petri net where each place has at most 1 in-transition and at most 1 out-transition”. We can also think of this as taking the original petri net and everytime we see a state with multiple out-transitions we copy the state and the net constructed till now once for each transition, we do the same for out-transitions.

Once we construct a Causal Net for a Petri Net we need to connect the execution of the Causal Net with that of the Petri Net. This will be done using a homomorphism.

Definition 7: A mapping $p : B \cup E \rightarrow P \cup T$ is a *homomorphism* if:

- $\forall e \in E, p(e^\bullet) = p(e)^\bullet$
- $\forall e \in E, p(\bullet e) = \bullet p(e)$
- $M_{0(\text{causal net})} = p(M_{0(\text{Petri net})})$

We will use a Causal Net and a homomorphism together as

Definition 8: A *Causal Process* of a Time Petri Net \mathcal{N} is a pair (CN, p) where CN is a causal net and p is a homomorphism from CN to \mathcal{N} .

Using p , the elements of CN are identified with their corresponding elements in \mathcal{N} . As a result, any run in the Causal Process corresponds uniquely to an untimed run in a Timed Petri Net. To also associate time stamps with our Causal Process we define

Definition 9: A *Timing Function* $\tau : E \rightarrow \mathbb{R}^+$ is a function from events of a causal process into time values.

2.c. Conformance Metric

Conformance Checking tries to measure how well a process model mimics the system, some of the metrics used for that are defined below.

Definition 10: Given a process model \mathcal{N} and a log L we define the *fitness* of \mathcal{N} with respect to L as

$$\text{fitness}(\mathcal{N}, L) = 1 - \max_{\sigma \in L} \text{dist}^*(\sigma, \mathcal{L}(\mathcal{N}))$$

Here dist^* is some normalized distance between traces, some options are defined later.

The fitness of the model is high if all of the observed behaviors in the logs are closely captured by the model.

Definition 11: Given a process model \mathcal{N} and a log L we define the *precision* of \mathcal{N} with respect to L as

$$\text{precision}(\mathcal{N}, L) = 1 - \max_{w \in \mathcal{L}(\mathcal{N})} \text{dist}^*(L, w)$$

We have that the precision of the model is high if it does not exhibit behavior that deviates too much from the observed logs.

3. Conformance Checking and Model Repair in Timed Setting

The Problem of Model Repair is, given an event log, a process model and some budget, compute the edits that can be made to the model under the budget to improve the conformance of the model to the system by some metric. If we let a Time Petri Net be our process model and fitness be our conformance metric then the problem can be stated as :

Model Repair of Time Petri Net (General)

Given a process \mathcal{N} denoted by a Time Petri Net, a log L and a budget β , we wish to find an edit of the $\mathcal{N} \rightarrow \mathcal{N}'$ that can be implemented under the given budget constraint and optimally increases the fitness.

The two ways in which the model can be imperfect fitness is to have traces in the log such that

- $\text{Untime}(L) \not\subseteq \text{Untime}(\mathcal{L}(\mathcal{N}))$, i.e there are traces where the sequence of events is not captured by \mathcal{N}
- There exists a trace whose untimed version is in the language, but the timestamps do not match with any word in the language of \mathcal{N}

Example 2: Consider the Process Model in **Example 1** and consider the the following observed traces

- $\sigma_1 = (a, 0)(a, 1)(b, 2)(d, 3), (e, 3)(f, 5)$
 - Clearly, there is no trace in the process model that has more than 1 a , which means the structure of the model itself needs to be updated by adding/removing states and transitions.
- $\sigma_2 = (a, 1)(b, 1)(d, 3)(e, 4)(f, 5)$
 - The sequence of transitions in σ_2 can happen in the model but for transition b we need to wait for at least 1 unit. Changing the timestamp for that transition to 2 gives a trace that has a run in the petri net.
- $\sigma_3 = (a, 1)(d, 1)(d, 2)(e, 4)(f, 5)$
 - This trace is also not possible in the model as transition b or c must be fired to enable transition f . This can be fixed by relabelling transition b or c .

In the untimed setting, this problem is viewed as minimizing cost over a series of edit moves which are either insertions or deletions to the model. For the timed case there are two aspects that need to be improved, which are mentioned above. This problem has been studied for the untimed case, but the timed settings is more complex.

Also, in practice, a large set of malfunctionings can be modeled as temporal anomalies (a slowing down of a conveyor belt speed due to wear, a shorter duration of a work phase due to an incorrect handling of the operator, a causal change in a timer duration, etc.) and the problem is a pre-requisite for the general case of dealing with all kinds of errors. In this paper we will be focusing on the purely timed version of the model repair problem. i.e where the only anomalies that are fixed are temporal ones (All traces that are not in the language of the model will have an issue similar to σ_2 in [Example 2](#))

Model Repair of Time Petri Net (Purely Timed)

Given a process \mathcal{N} denoted by a Time Petri Net, a log L and a budget β , we wish to find an edit of the $\mathcal{N} \rightarrow \mathcal{N}'$ that can be implemented under the given budget constraint and optimally increases the fitness. We also have the constraint that $\forall \sigma \in L, \text{UnTime}(\sigma)$ gives a valid causal process for \mathcal{N} .

To properly formalize the problem we need definitions for editing out petri net and conformance for which we need to define out distance functions.

3.a. Edits and Distances

Our notion of distance is similar to that of Levenstein's edit distance where we are given a set of edit actions and we try to go from one trace to another in the shortest way, representing in some sense how different 2 traces are, there are 2 options that are considered usually

Definition 12: (Stamp Edit) Given a timing function $\gamma : E \rightarrow \mathbb{R}^+$, we define the a stamp move as:

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

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

i.e we edit the timestamp at which a particular transition (e) was taken by x . These edits only affect a single transition, and can represent a reading error in the model which needs to be corrected without affecting the other timestamps.

Another natural edit move to consider is :

Definition 13: (Delay Edit) Given a timing function $\gamma : E \rightarrow \mathbb{R}^+$, we define the a delay move as:

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

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

Here the relation \geq_G can be thought of as a causal one, i.e if e must happen for e' to happen then $e \leq_G e'$.

Intuitively, this edit represents a change in the duration one waits before taking a transition, this is why timestamps of all subsequent transitions are also changed by the same amount.

Using these 2 distance we can define out notion of distance. We assign a cost to each edit move say for both delay and stamp edits we say that the cost of an edit is the same as the change x , using that we can define the following 3 definitions:

Definition 14: (*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 distance as

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

Definition 15: (*Delay Only Distance d_θ*) Given any two timing functions τ_1, τ_2 over the same causal process (CN, p) , we define the stamp distance as

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

Definition 16: (*Mixed Moves Distance d_N*) Given any two timing functions τ_1, τ_2 over the same causal process (CN, p) , we define the stamp distance as

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

4. Results

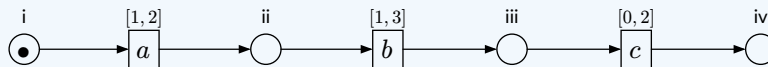
4.a. Sequential Petri Nets and Delay-Only Distance

We first focus out attention to a restricted version of problem where [Definition 15 14](#).

- The model in question will be a **Sequential Time Petri Net**, which means that, there is a dedicated start state and a dedicated end state, and each transition, connects one state, to one other state in a way that the underlying graph looks like a line graph, which the start and end states acting as the two ends of the graph.
- The problem is restricited to a **Purely Timed Problem**, which means that the sequence of transitions represent the sequence of events in the system correctly, but the timestamps might not be accurate.
- The metric for measure which will be used is going to be **Delay-Only Distance**,
- For the edits to the mode, we cost x unit of the budget whenever any bound of a time range of a transition is changed by x .
- The conformance metric used is *fitness* but here we define it as $-(\max_{\sigma \in L} \text{dist}_\theta(\sigma, \mathcal{L}(\mathcal{N})))$ which can be easily converted to the normalized distance used in the original definition.
- Another thing to note is that for a Sequential Petri Net is in bijection with it's Causal Process, hence we will not make a distinction here.

Example 3: We start with a simple example and informally go over the procedure

Consider the following Time Petri Net \mathcal{N}



We are also given the following log

$$L = \left\{ \begin{array}{l} [(a, 1) (b, 5) (c, 9)], \\ [(a, 0) (b, 5) (c, 7)] \end{array} \right\}$$

And we are given the budget $\beta = 2$.

Our goal is to edit the model by making a change of at most $\beta = 2$ to the boundaries of the transitions in order to minimize the distance of the logs from the model. The Procedure will go as follows:

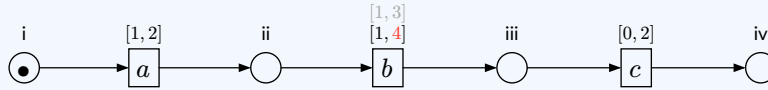
- The transitions only keep track of the delay that is done to trigger them, so it will be easier to look at traces with wait times before the previous transitions rather than the next one, so we construct the following

$$F = \left\{ \begin{array}{l} [(a, 1) (b, 4) (c, 4)], \\ [(a, 0) (b, 5) (c, 2)] \end{array} \right\}$$

And now it is easier to see that neither of the traces have a corresponding run in N , also, the distance of each transition is now easy to compute, for both the transitions it's 3. So the fitness is -3 and we need to reduce the distance of the model from each of the traces to improve the fitness of N .

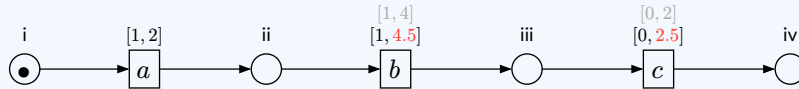
Trace 1 takes transition b and c too late, where as Trace 2 takes transition b late and takes transition a too early. This means that increasing the upper bound of transition b reduce the distance of both traces, and hence will be optimal, so we spend our budget on b .

After spending 1 unit of budget though the model changes to



And now trace 1 takes the transition b at the correct time. If we continue to spend out budget entirely on b then we will not reduce the distance of trace 1 from the model anymore and hence will not change the fitness of the model.

So we need to rethink our distribution of the budget. We need to reduce the distance of both the traces from the model, hence we need to spend budget in a way that improves fitness. One way to do it optimally would be to split our leftover budget evenly between transition b and c . And finally ending with the following model.



Now we can stop as we have consumed all our budget and the distance of each trace from the model is 1.5.

Note that this is not the only optimal solution, reducing the lower bound of transition a instead of increasing the upper bound b in step to also leads to the same improvement in fitness.

4.a.i. Plan:

- Our fitness depends on the distances of the log traces from the model, so we start with computing those. During that we find the traces with maximal distance, which are the ones that affect fitness.
- Find how changing each transition will affect the found subset, we have a limited budget, so we must find the edits to the transition that improves the fitness the most, we use linear programming to find it.
- Once the optimal change is found, we keep changing the transitions until one of the 3 things happen
 - We run out of budget
 - Our model becomes perfectly fit (log lies entirely in the language of the model)
 - we unequally reduce the distance of the log entries, so at some point, one of the traces not considers, becomes significant, at which point we go back to step 1

4.a.ii. Finding the Distance between the Model and the Log Traces:

During the execution of the model repair algorithm, we keep updating the model, which means that the set of furthest log traces might change, hence we need to keep track of all the traces.

We define the distance between a trace and a model as the minimum distance between the trace and any word of the model.

Definition 17: The *flow function* (or simply *flow*) of a trace is $f : (\Sigma, \mathbb{R}^+)^*$ which keeps track of the delay between successive events and is defined for $\tau = \tau_1\tau_2\dots\tau_n$ as $f = f_1f_2\dots f_n$ where

$$f_i := \begin{cases} \tau_1 & i = 1 \\ \tau_i - \tau_{i-1} & i \in [2\dots n] \end{cases}$$

Let the sequence of transitions in \mathcal{N} be $T = \{t_1, t_2, t_3\dots t_n\}$ where each $\text{SI}(t_i) = \langle s_i, e_i \rangle$. Given a trace $\tau = \tau_1\tau_2\dots\tau_n$ we can say it's distance from \mathcal{N} can be given by the following.

Given a trace, we first find its flow and then we can use that to easily compute it's distance from the model in the following way.

Algorithm 1: Dist (\mathcal{N} : net, f : flow):

```

1   $n := f.\text{length}()$ 
2   $\text{dist} := 0$ 
3  for  $i$  in  $1\dots n$ :
4    if  $f_i < s_i$                                 ▸ Transition taken too early
5       $\text{dist} := \text{dist} + (s_i - f_i)$ 
6    else if  $f_i > e_i$                              ▸ Transition taken too late
7       $\text{dist} := \text{dist} + (f_i - e_i)$ 
8    else                                           ▸ Transition taken on time
9       $\text{dist} := \text{dist} + 0$ 
10 return  $\text{dist}$ 
```

This can be done for each trace in L .

Now that we have a the set of logs we find the subset of the that are furthest away from the model which can simply be given by

$$L_{\max} = \{\sigma \in L \mid \forall \sigma' \in L, \text{Dist}(\sigma, \mathcal{N}) \geq \text{Dist}(\sigma', \mathcal{N})\} \quad (1)$$

4.a.iii. Finding the Optimal Changes to Transitions:

We want to minimize our budget for a given change in the fitness of the model, there are a few things that we need to keep in mind for that.

- The fitness is only affected by log traces that are furthest away from the model.
- If we want to make a change to transition, which affects some of the furthest traces, but not all, it will not change the fitness, as the unaffected trace is still equally far away. So we would like to divide the budget to deal with multiple log traces at once.

Note that we can say that improvement in fitness is \min of improvements in each trace. And improvement in a trace τ is just the sum of budgets assigned to the transitions that affect the distance of τ .

- For dealing with all boundaries at once, we define $B = \{s_i \mid \langle s_i, e_i \rangle \in T\} \cup \{e_i \mid \langle s_i, e_i \rangle \in T\}$
- The above statement about boundaries affecting traces can be formalized as

$$\text{is_affected_by}(\tau, b) = \begin{cases} \top & b = e_i \text{ and } e_i < \tau_i \\ \top & b = s_i \text{ and } s_i > \tau_i \\ \perp & \text{otherwise} \end{cases} \quad (2)$$

We can rephrase our problem of finding an optimal distribution as a linear program, we define use the following definitions for it:

- for each trace $\tau_i \in L'$ we have a variable tr_i which represents how close the trace gets after making the edit.
- for each $b \in B$ we have the variable ch_b which represent the portion of the budget assigned to that bound. Then we get the following equation for each tr_i .

$$\text{tr}_i = \sum_{\substack{b \in B \\ \text{is_affected_by}(\tau_i, b) = \top}} \text{ch}_b \quad (3)$$

And we can measure the overall improvement by the variable improvement which can be given the constraint for each $\tau_i \in L'$

$$\text{improvement} \leq \text{tr}_i \quad (4)$$

Now for our linear program we just need condition

$$\max(\text{improvement}) \quad (5)$$

There are 3 extra constraints that we need to put that the algorithm does not ask us to spend an infinite amount of budget, these constraints correspond to the conditions when we need to stop spending the budget.

- We cannot spend more than the budget

$$\sum_{b \in B} \text{ch}_b \leq \beta \quad (6)$$

- We not to re-evaluate the updates each time an edit to a transition makes a trace be valid at some point.
 - To do that, for all $\tau \in L$, we define its distance $d_{\tau, b}$ from a bound b as
 - 0 if not affected by it.

- Distance between f_j and t_j where f is the flow function of τ and $t_j = \langle -, b \rangle$ or $\langle b, - \rangle$
- And $\forall \tau \in L$ and $\forall b \in B$, if $\text{is_affected_by}(\tau, b)$ we add the constraint

$$d_{\tau, b} \geq \text{ch}_b \quad (7)$$

- We need to consider new transitions when they join the set of furthest transitions
 - For every $\tau_i \in L$ (note: previously we were only dealing with L') we define

$$\text{tr}_i = \sum_{\substack{b \in B \\ \text{is_affected_by}(\tau_i, b) = \top}} \text{ch}_b \quad (8)$$

and

$$D - \text{improvement} \geq \text{Dist}(\tau_i, \mathcal{N}) - \text{tr}_i$$

where D is the maximum distance of a log trace from \mathcal{N} .

Finding a solution to the above linear program gives an edit for the petri net, and the change in the fitness which is improvement.

4.a.iv. *Editing the Petri Net:*

Now we go over all ch_b

- If $b = e_i$, then we set $e_i \leftarrow e_i + \text{ch}_b$
- If $b = s_i$, then we set $s_i \leftarrow s_i - \text{ch}_b$
- We also set $D \leftarrow D - \text{improvement}$
- We update the budget $\beta \leftarrow \beta - \sum_{b \in B} \text{ch}_b$

And we keep repeating this process until our budget goes down to zero.

References

- Aalst, W. van der. (2016). *Process Mining: Data Science in Action* (2nd ed.). Springer Publishing Company, Incorporated.
- Chatain, T., & Rino, N. (2022). Timed Alignments. *2022 4th International Conference on Process Mining (ICPM)*, 0, 112–119. <https://doi.org/10.1109/ICPM57379.2022.9980687>