Open Access

Model Repair and Conformance Checking of Time-Aware Models

Shubh Sharma [1, 2], and **Thomas Chatain** [2]

[1] Chennai Mathematical Institute, [2] LSV, 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 researchh, 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 it's future behaviour (Aalst, 2016). It is a multi-step process that happens as follows:

Discovery: Discovery in process mining is the process of constructing a model as a an attempt to describe the working of a given system. This is a well studied domain and it is common to see tools like Machine Learning used here. But the unexplainability of the approaches that processes like ML take naturally makes one questions if the produced model approximates the target system well enough.
This is where **Conformance Checking** comes in. Conformace Checking is a set of techniques used to comapre a *process model* with an event log and rate it over some parameters like:

- Fitness : Does the model exhibit the beahviours specified in the logs?
- Precision : Does the model deviate a lot from the behavior specified in the logs?
- Generalization : Does the 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 **Enahncement**: where and existing process model is extended or improved using information about the actual process record in some event log.

Research Internship

Published Jul 04, 2024

Another important part of Process Mining is **Performance Analysis** where the goal is to analyze and improve the exectution of the model to use less time and resources an 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 improvment 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 are offer a graphical means to represent concurrent systems and a formal semantics for their exectution. 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 is represented by the set of timed words over the alphabet, which is a list of events along with the timestamps on which the event occured. The notion of distance between words, which will give our conformance metric will be similar to the Levenstein'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 $\mathcal{N} = \langle P, T, F, \mathrm{SI}, \Sigma, \lambda, M_0, M_f \rangle$ 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.
- $\mathrm{SI} : T \to \mathbb{I}$ is the static interval function, $\mathrm{SI}(t) = (\mathrm{st}(t), \mathrm{en}(t))$ where
  - $\mathrm{st}(t)$ is the smallest valid time interval from the enabling of t to its firing.
  - $\mathrm{en}(t)$ is the largest valid time interval from the enabling of t to its firing.
- $\lambda : T \to \Sigma$ is a labelling function for the transition with actions from the action set Σ
- M_0 and $M_f : P \to \mathbb{N}$ are the initial and final markings.

Given a transition $t \in T$ we define

- The Pre-set of t 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 $\mathrm{Enabled}(M) = \{t \in T \mid t \text{ is enabled in } M\}$.

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

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:

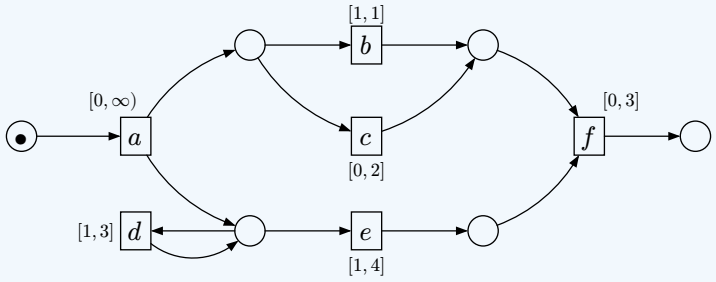
$$M' = (M \setminus {}^\bullet t) \cup t^\bullet$$

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

This is also denoted as $S[t\rangle S'$

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 langauge of the Time Petri Net as follows

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

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

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 though the Petri Net easier.

Definition 6: A *Causal Net* $\mathrm{CN} := \langle B, E, G\rangle$ 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 $p : B \cup E \to P \cup T$ is a *homomorphism* if:

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

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) wher 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* $\tau : E \to \mathbb{R}^+$ is a function from events of a causal process into time values.

Another useful way to capture the relaton between a trace and a causal net would be to look at the amount of time a transitions has to wait before it is triggered, this is defined using the flow function

Definition 10: Given a causal process (CN, p) and a timing function $\tau : E \to \mathbb{R}^+$, one can define a flow functions $f_\tau : E \to \mathbb{R}^+$ as :

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

2.c. Conformance Metric

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

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

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

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

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

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 me 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 $\mathcal{N} \to \mathcal{N}'$ 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

- $\mathrm{Untime}(L) \not\subseteq \mathrm{Untime}(\mathcal{L}(\mathcal{N}))$, 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 langauge, 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 possbile 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 problems is veiwed as minimzing 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 malfuctionings 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 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 Nets (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 $\mathcal{N} \to \mathcal{N}'$ that can be implemented under the given budget constraint and optimally increases the fitness. We also have the constraint that $\forall \sigma \in L, \mathrm{Untime}(\sigma)$ 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 13: (*Stamp Edit*) Given a timing function $\gamma : E \to \mathbb{R}^+$, we define the a stamp move as:

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

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

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 corrected without affecting the other timestamps.

Another natural edit move the consider is :

Definition 14: (*Delay Edit*) Given a flow function $\eta : E \to \mathbb{R}^+$, we define the a delay move as:

$$\forall x \in \mathbb{R}, e \in E : \text{delay}(x, e)(\eta) = \eta' \text{ where}$$
$$\forall e' \in E : \eta'(e) = \begin{cases} \eta(e') + x & e' = e \\ \eta(e') & \text{otherwise} \end{cases}$$

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 our 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 15: (*Stamp Only Distance* d_t) Given any two timing functions (or flow 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 16: (*Delay Only Distance* d_θ) Given any two timing functions (or flow 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 17: (*Mixed Moves Distance* d_N) Given any two timing functions (or flow functions) τ_1, τ_2 over the same causal process (CN, p), we define the stamp distance as

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

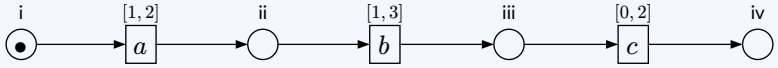
3.b. Alignments

In (Adriansyah, 2014), the notion of alignment was defined as the minimal series of corrections need to transform a log trace to match a trace closest to it in the language of the process model. This idea was extended further in (Chatain & Rino, 2022), timed alignments are the members of the language of a process model that only differ from a timed trace in the time stamps and can be modified to fit the trace with the least amount of steps.

Given a log trace and a model, we can find an alignment, which is the trace in the model that can be converted to the given log trace with the least amount of edits. Hence, the idea behind all of the following sections would be to find an alignment for each log trace, and try to edit it to match the trace, and whenever the edit takes the alignment out of the language of the model, we extend the model to accomodate it.

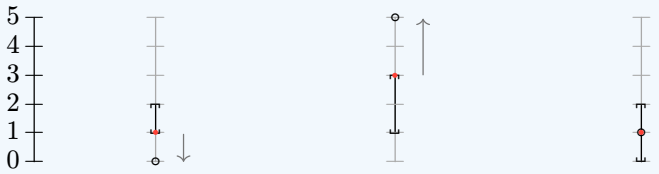
Example 3: The following is a visual representation of the above idea.

Consider the following Time Petri Net N

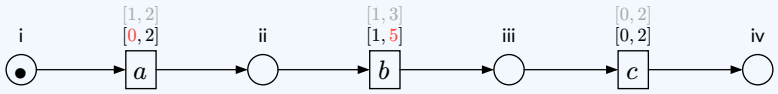


Along with the flow function $f = (a, 0)(b, 5)(c, 1)$ and the notion of distance used is **Delay-Only** distance.

This word does not belong to the language of the model and it's alignment would be the flow function $a_f = (a, 1)(b, 3)(c, 1)$, both of these can be seen in the following diagram



Here $\circ$ is used to donate the flow function of the f, while $\bullet$ is used to denote the flow function of a_f and the intervals represent the static intervals of the 3 transitions. We improve the fitness of the model by getting the a_f closer to f by changing the intervals of N to get the following Time Petri Net N':



4. Results

4.a. Extended Free Choice Time Petri Nets with Delay-Only Distance

During the process of editing the model, the alignment will keep changing, to make sure that we can freely change the flow function at a point, without having to worry about the fireability of other transitions that not causally linked to the current one we focus out attention to **Extended Free Choice Petri Nets**

Definition 19: A Time Petri Net is *Extended Free Choice* iff, for all two transitions t and t' we have that ${}^\bullet t \cap {}^\bullet t' \neq \emptyset \Rightarrow {}^\bullet t = {}^\bullet t'$

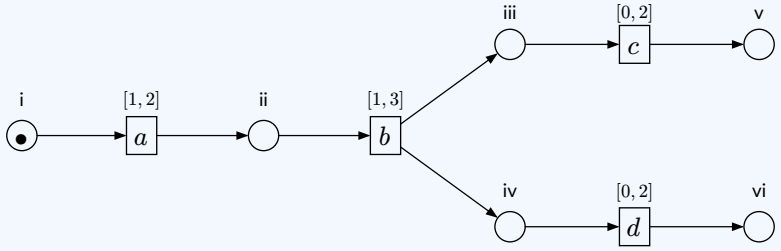
The problem dicussed here would be The Purely Timed Model Repair Problem, so the untimed log is a subset of the untimed language of the petri net.

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.

Example 4: 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

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

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:

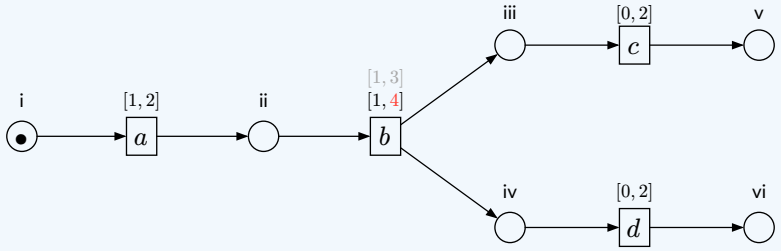
- The transitions only keep track of the delay that is done to trigger them, so it will be easier to look at traces with wait times before the previous transitions rather than the next one, so we construct the following

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

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.

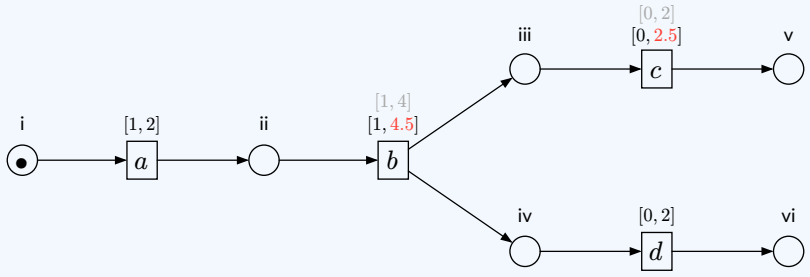
Trace 1 takes transition b and c too late, where as Trace 2 takes transition b late and takes transition a too early. This means that increasing the upper bound of transition b reduce the distance of both traces, and hence will be optimal, so we spend our budget on b.

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. ***The*** unfit ***function:***

With Extened Free Choice Petri nets, we get a conflict free subset of petri nets, which means that we can freely alter the delay we wait before taking a transition without having to worry about the fireability of other transitions that are not causally linked to the current one, this makes it easy to compute the alignment of a given trace. The edits we make to a given petri net and how it affects the *fitness* of the model is captured by the unfit function.

For each transition, the bounds of its static interval can be edited in 2 ways, increasing the upper bound and reducing the lower bound, so given a petri net with n transitions, every edit made to the petri net can be represented as a vector in $\mathbb{R}^{2n}$. We represent the initial configuratio n of the net $\mathcal{N}$ as $\vec{0}$ and a vector $\vec{v}$ represents the petri net $\mathcal{N}'$ derived by editing $\mathcal{N}$ as follows: If the k^{th} transition of $\mathcal{N}$ is $[a, b]$ then the k^{th} transition of $\mathcal{N}'$ is $[a - 2\vec{v}_{2k-1}, b + \vec{v}_{2k}]$.

Here $\vec{v}_m$ is the m^{th} component of v where indexing starts at 1.

More specifically, the unfit function has the type $(\mathbb{R}^+)^{2n} \to \mathbb{R}$ and takes in a vector in $(\mathbb{R}^+)^{2n}$ which represents an edited petri net, and returs the maximum distance of any log trace from the language of the model.

The following helper function $d' : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ will be useful: $d'(a, b) = \max(0, b - a)$

First we define the unfit function for the case where there is just 1 trace in the log. Let the trace be τ and it's flow function be f. Here the unfit function is the same as the distance function.

For each transition t_i of the petri net, we define $d_i\left(\vec{a}, \vec{b}\right) = d'\left(\vec{a}(i), \vec{b}(i)\right)$. Where $\vec{a}(i)$ is the i^{th} component of $\vec{a}$.

Now that we have defined it for each component we let $D\left(\vec{a}, \vec{b}\right) = \sum_{i=1}^{n} d_{i\left(\vec{a}, \vec{b}\right)}$

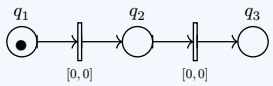
Theorem 2: Given a vector $\vec{a}$ representing an edit on the petri net $\mathcal{N}$ producing $\mathcal{N}'$ and a flow function f in $\mathcal{F}'$, $D(a, f)$ is precisely dist_θ between the edited net and f.

Definition 18: Given a net $\mathcal{N}$ with constant $[0, 0]$ static interval functions for all transitions and a log $\mathcal{L}$, the unfit *function* can be defined as follows.

$$\mathrm{unfit}_{(\mathcal{N},\mathcal{L})}(a) = \max_{f \in \mathcal{F}'} D(a, f) \tag{1}$$

Corollary 3: $\mathrm{unfit}_{(\mathcal{N},\mathcal{L})}(a)$ is negation of the fitness of $\mathcal{N}$ with respect to $\mathcal{L}$.

Example 5: Consider the following net $\mathcal{N}$ with 2 transitions



And the following set of flow functions for it

$$F = \begin{Bmatrix} f_1 &= [2 & 0], \\ f_2 &= [0 & 2], \\ f_3 &= [1.5 & 1.5], \end{Bmatrix}$$

And the following are the graphs of the unfit functions

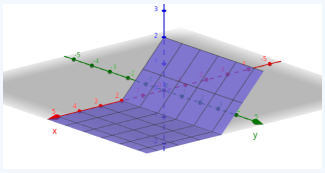


Image 1: unfit function for f_1

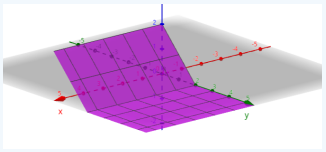


Image 2: unfit function for f_2

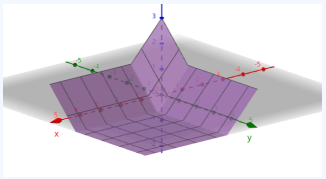


Image 3: unfit function for f_3

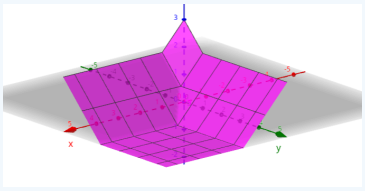


Image 4: unfit function for the entire problem

4.a.ii. *Gradient Descent:*

Theorem 4: Some properties of the unfit functions:

The unfit functions turns out to have a ton of nice properties.

- The function d_i is continuous and a piecewise function, linear in each part.
- It is also a convex function.
- Since unfit us just multiple d_i functions combined using $\max$ and summation. which means it's also continuous, piecewise linear, and a convex function.
- Also the domain of the function in general is $(R^+)^n$, but it can be restricted to all vectors such that the sum of their components (p_1 norm) $\leq \beta$ to give the problem a budget. In their scenarios the input space is a convex set.

These properties make it really good for gradient descent. Hence we will choose that as our strategy to solve the problem, it also precisely matches an intuitive direct startegy of finding the way to distribute the budgets to make changes optimal locally.

Gradient Descent is a greedy mathematical optimization techniques that involves starting at a point in the input space and moving in the opposite direction of the gradient(in the direction where the function decreases at the highest rate), until a local minimum is reached.

For our case picking Gradient Descent is a good option because:

- The function is piecewise linear, which means each part the gradient is constant and can be computed easily using linear programming.
- The function is convex with a convex set as it's input space, which means that there is only 1 set of global minima which is also convex and no other critical point.
- The input space can be bounded. If the fitness of the model is $-k$ then the we can bound the input space to all vectors whose p_1 norm is less than $k * |L|$. This is also a closed set, hence compact, which means the points that evaluate the global minima are in the set.

Therefore we can give the following algorithm for computing the model repair problem

- We start at the point $\mathbf{0}$
- At every step we find the direction of steepest descent and we keep moving in that direction:
  - Consume the entire budget
  - A new trace becomes a trace with maximum distance from the net
  - Some trace with maximum distance from net has a transition which is taken at a time delay accepted by the edited petri net.
- Whenever we reach such a point, we do recompute the gradient and keep repeating this and the previous step.
- We stop when the bugdet runs out or when the model becomes fit.

4.a.iii. *Computing the Solution:*

We have defined our unfit function as the maximum of the distances of the log traces from the model. Let $\mathcal{F}_{\max}$ be the set of flow functions with maximum distance from $\mathcal{N}$.

We now use linear programming to both compute the direction of steepest descent and the amount of budget to be spent in it before a recomputation is required.

We will be using the following list of variables.

- For every t in the set of transitions of the petri net (denoted by T), we have a variable b_t holding the budget assigned to t.
- For every f in $\mathcal{F}$ we have imp_f denoting the reduction in distance of f from $\mathcal{N}$ because of the edit.
- $\mathrm{improvement}$, holding the total change in the fitness of the model, hence our goal will be to maximize this.
- spend, holding the total amount of budget spent.

The following constants will be helpful in writing the equations

- We let $\vec{a} = \mathbf{0}$
- β, which is the total budget available
- $\text{un-fitness} = \mathrm{unfit}_{(\mathcal{N},\mathcal{L})}(\vec{a})$
- For each $f \in \mathcal{F}$ we have $d_f = D(\vec{a}, f)$, that is the distance of f from $\mathcal{N}$

- for each $f \in \mathcal{F}$, and $e \in E$ we let $d_{f,e} = f(e)$
- For each $f \in \mathcal{F}$ and $e \in E$ we have $\text{affects}_{e,f}$ which is
  - 0 if t is the i^{th} transitions and $d_i(\vec{a}, f) = 0$
  - 1 otherwise

The goal of linear program is

$$\text{Maximize}(\text{improvement}) \tag{2}$$

Under the constraints:

- Improvement in a flow function is the total budget assigned to the transitions which affect it, for each $f \in \mathcal{F}$ we have

$$\text{imp}_f = \sum_{t \in T} b_t \times \text{affects}_{t,f} \tag{3}$$

- Total improvement is the least improvement of all of the flow functions in $\mathcal{F}_{\max}$, so for each $f \in \mathcal{F}_{\max}$

$$\text{improvement} \leq \text{imp}_f \tag{4}$$

- We restrict the amount of budget we spend to exactly 1 unit, this does not give the final edit, but the direction in which we make it.

$$1 = \sum_{t \in T} b_t \tag{5}$$

The variables b_t can be used to compute the direction of steepest descent, but we also add the following constrainst to calculate the largest "step size" after which we need to do recomputation.

- We need to recompute when a new flow function needs to be added to $\mathcal{F}_{\max}$ so for all $f \in \mathcal{F}$ we compute the amount of budget that can be spent as

$$\lambda_f = \frac{d_f - \text{un-fitness}}{\text{imp}_f - \text{improvement}} \tag{6}$$

- We need to recompute each time the effects variables become outdated, so for all $i \in [1...n]$ let t_i be the i^{th} transition and forall $f \in \mathcal{F}$ we have

$$\lambda_{f,i} = \frac{d_{f,i}}{b_{t_i}} \tag{7}$$

We now consider λ the minimum of all positive $\lambda_f, \lambda_{f,i}$ and the budget β, that is the amount of budget that can be spent safely.

Once we get values of b_{t_i} after solving the linear program, we set $\vec{a}(i) := \vec{a}(i) + \lambda b_{t_i}$ where t_i is the i^{th} transition. We also set $\beta := \beta - \lambda$. After fixing those 2, we recompute all the other constants and do the linear programming again till we consume the entire budget or un-fitness becomes 0.

4.a.iv. *Proof of correctness:*

The proof of correctness involves showing that we always reach the minimum of the unfit function.

Lemma 1: let y be a point where the unfit function reaches its minimum in the domain, then from any point x where the function does not reach its minimum, if the algorithm is on x then it has not terminated.

From the construction of the algorithm, we get that the algorithm does not terminate iff there exists a v and an $\varepsilon > 0$ such that $\forall 0 < \varepsilon' \leq \varepsilon, \mathrm{unfit}(x + \varepsilon' v) < \mathrm{unfit}(x)$.

Let $v = y - x$ and we pick $\varepsilon = 1$ and the above condition is met by convexity of unfit. $\square$

Now we just need to prove that the algorithm terminates:

- Each iteration of the algorithm, which involves a linear optimization problem which takes finite amount of time, and an edit which takes a finite amount of time.
- We divide the input space into finitely many regions as follows:
  - first, for each boundary and trace, we have 2 regions, one where the trace respects the static intverval of the boundary, and one where it doesn't. So for n transitions, we get atmost 2^n regions. And we consider the common refinement of the partitions created using the process by each trace. Whenever we change a partition during gradient descent, we need to start another iteration of steps.
  - Within a partition, each the unfit function for each trace behaves like a linear map, so for each pair of traces τ_1 and τ_2, each part can be further divided into 2 parts, where the unfit function for τ_1 is greater, and where it isn't. This gives us that there are only finitely many parts.

  1. Here we have that each iteration takes us to a differnt part, by construction.
  2. None of the parts as defined in the first bullet are visited twice, as each region can be given a vector $v_k \in \{0, 1\}^{2n}$ where the i^{th} component is 1 if the i^{th} transition affects the k^{th} trace, and we switch regions only me converting a 1 to a 0 for some vector, this can only happen a finite amount of time.
  3. withing each region, the function is a $\max$ of linear functions, and hence gradient descent takes finitely mant steps.

  So the algorithm must terminate.

This proof works for Extended Choice Free Petri Nets as they are conflict free, chaging the time at which a transition is taken does not effect the fireability of any other transitions.

References

Aalst, W. van der. (2016). *Process Mining: Data Science in Action* (2nd ed.). Springer Publishing Company, Incorporated.

Adriansyah, A. (2014). *Aligning observed and modeled behavior*. https://api.semanticscholar.org/CorpusID:64418472

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