Master Thesis Proposal

Kefang Ding

February 24, 2019

Title:	Process Enhancement by Incorporating Negative Instances
Author:	Kefang Ding
Supervisor:	Professor Dr. ir. Wil van der Aalst ir. Sebastiaan J. van Zelst
Degree:	Master of Science

Abstract

Model repair in process mining aims to improve existing process model according to actual event log. Event log is divided into positive and negative instances based on given KPIs. However, the current repair technologies use only positive instances, while negative instances are ignored. This results in a less precision model. This article focuses on incorporating both positive and negative instances to repair model, in order to provide a model with better precision.

Firstly, a directly-follows graph is created from an existing process model in form of Petri net, positive and negative instances of event log. Then, the directly-follows graph is transferred to a process tree, which is used to generate the final model in Petri net. To improve the precision of Petri net, long-term dependency is analyzed and added to Petri net.

In comparison to current technologies, the methods proposed in this articles provide better result with respect on precision in most cases.

1 Introduction

Research in process mining has gained much interest over the past decade[1, 2]. The objective of process mining is to support the analysis of business process and provide valuable insights on processes. According to [1], techniques of process mining are divided into three categories: process discovery, conformance checking and process enhancement. Process discovery techniques focus on deriving process models from event logs of the information system, aiming to improve the understanding of real business process. Conformance checking analyzes the deviations between process models and observed behavior of its execution. Enhancement adapts and improves existing process models by extending the model with more data perspectives or repairing the existing model to better reflect observed behaviors.

Due to the increasing availability of detailed event logs of information systems, process mining techniques have recently enabled wider applications of process mining in organizations around the world[1]. After applying process discovery on the information system, a process model is fixed for organizations to guide the execution of business. However,in real life, business processes often encounter exceptional situations where it is necessary to execute process differing from the predefined model. The organizations need to adapt the existing process model to reflect the reality. Basically, one can apply process discovery techniques again to obtain a new model from event log. Yet, the discovered model tend to have less similarity with the original model[3]. As shown in [3], there is a need to change an existing model similar to the original model while replaying the current process execution. Here comes the model repair.

Model repair belongs to process enhancement and stands between process discovery and conformance checking. It analyzes the workflow deviations between event log and process model, and fix the deviations by adding sub processes on the model. As known, business in organizations is goal-oriented and aims to have high performance according to a set of Key Performance Indicators(KPIs), for example, average conversion time for the sales, payment error rate for the finance. However, there are little research on applying the process mining with consideration of performance[2]. It points the contribution of [4] to combine performance into process mining. In [4], the event log is firstly divided into positive instances and negative instances with respect to specific KPIs. Then, the positive instances are used to repair the model. In this way, negative instances are simply ignored, which results in a model with less precision.

As an example, there is an model presented in Figure 1 (a), where A is followed directly by B. During its execution in real life, an event log is generated:

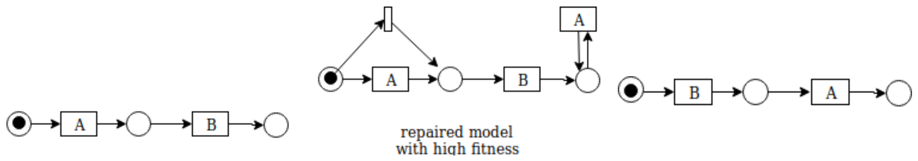
$$< A, B >^{55}, < B, A >^{105}$$

When considering KPIs performance, the log is divided into a positive and a negative set:

$$Positive\ examples : < A, B >^{5}, < B, A >^{100}$$

$$Negative\ examples : < A, B >^{50}, < B, A >^{5}$$

After applying current model repair techniques in [5], the process model is repaired using all examples. A can be skipped and duplicated later in a self-loop. In [4] methods, only the positive examples are taken for the model repair. Yet, since both $< A, B > and < B, A >$ contributes to good performance, the repaired model keeps the same like in Figure 1(b).



(a) original process model (b) process model with high fitness (c) process model with good KPI

Figure 1: example for model change under model repair

However, it's obvious that $< A, B >$ often leads to bad performance and therefore should be excluded. The Figure 1(c) shows the expected model with incorporating the negative information. This model reinforces positive examples and avoids negative instances, which provides us a more accurate view of the business process.

The demand to repair model with incorporating negative instances appears. The concrete problem showing with demand is as follows. Given an input of one existing process model M, an event log L and KPIs, how to improve current process enhancement techniques by incorporating negative information within process model repair, and generate a process model to enforce the positive instances while blocking the negative instance. Therefore, the repaired model provides a better way to understand and execute the real business process.

This paper tries to provide a solution for it. The reminder is organized in the following order. Section 2 recalls the basic notions on process mining and list the preliminary to solve the problem. The next section lists our methods are introduced and formal definitions are given. In Implementation Section, the details of algorithms are given. Later, we evaluate our methods with simulated data and real data respectively and list the results. Subsequently, the discussion on this paper is presented. At last section, a conclusion is drawn on the paper.

2 Related Work

Process enhancement focuses to extend or improve an existing process model by using actual event log[1]. Besides extending the model with more data perspectives, repair is another type of enhancement. It modifies the model to reflect observed behavior while keeping the model as similar as possible to original model.

In [3], model repair is firstly introduced into process enhancement. By using conformance checking, the deviations of alignment are detected. The consecutive log moves is collected in the form of subtraces at specific location Q. Later, the subtraces are grouped into sublog that share the same location Q for subprocess discovery. In the earlier version in [3], the sublogs are obtained in a greedy way, while in [5], sublogs are gathered by using ILP miner to guarantee the fitness. Additional Subprocesses and loops are introduced into the existing model to ensure the fitness, which also brings variants of execution paths into the model.

Later, compared to [3, 5], where all deviations are incorporated in model repair, [4] considers model performance into model repair. An observation instance is built to represent the type of log moves given trace and KPI. Subsequently, a classification tree will be constructed from the set of observation instances to cluster traces of event log. Then, the techniques in [5] are applied into event log with positive traces to repair model.

As described above, the state-of-the-art repair techniques are based only on positive instances, meanwhile the negative instances are neglected. Without negative instances, it is difficult to balance the fitness and precision of those model. Few researches give a try on incorporating negative information into process discovery. [6] analyzes the available events set before and after one position and generates artificial negative events based on the complement of those event sets. Next, Inductive Logic Programming is applied to detect the preconditions for each activity. Those preconditions are then converted to petri net after applying a pruning and post-process step.

Similar work on model discovery based on negative information are published later. In [7], the author improves the method by assigning weights on artificial events with respect to unmatching window, in order to offer generalization on model. [8] presents a supervised approach based on SMT to incorporate negative information for model discovery.

However, the field of model repair which considers the negative information is new.

To incorporate the negative information, simulated data are used, to limit the choices of going..

Compared to this, our approach is innovative mainly in the following aspects.

- Incorporate the negative information into model repair. Unlike the methods mentioned before

- Analyze the long-term dependency in the model to provide a preciser result.

- Analyze Model on Trace level. All activities constituting a trace are considered.

3 Preliminary

Copy the existing document and paste here..

3.1 Definitions Related To Dfg-Method

In this part, we provides concepts related to the dfg-method which is based on directly-follows graph. A directly-follows graph as used in [9], represents the directly-follows relation of activities in event log. For instance, if there are traces of $\langle ..., A, B, ... \rangle$ in event log, one edge (A,B) is added into directly-follows graph. By cutting directly-follows graph under different conditions, Inductive Miner[9, 10] discovers a process model. Unlike this process, we adapt Inductive Miner to repair model by using existing model, and event log with labels.

Definition 3.1 (Cardinality in directly-follows graph)**.** *Given a directly-follows graph G(L) derived from an event log L, the cardinality of each directly-follows relation in G(L) is defined as:*

- $Cardinality(E(A, B))$ *is the frequency of traces with* $\langle ..., A, B, ... \rangle$.
- *Start node A cardinality* $Cardinality(Start(A))$ *is the frequency of traces with begin node A.*
- *End node B cardinality* $Cardinality(End(A))$ *is the frequency of traces with end node B.*

From the positive and negative event log, we can get directly-follows graphs, respectively $G(L_{pos})$ and $G(L_{neg})$. Each edge of $G(L_{pos})$ and $G(L_{neg})$ has a cardinality to represent the strength of this directly-follows relation. However, when the existing model is transformed into directly-follows graph $G(L_{ext})$, there is no point to assign cardinality on each edge. So we just set 1 to cardinality of each edge.

To incorporate all information from $G(L_{pos})$, $G(L_{neg})$ and $G(L_{ext})$, we define weight for each directly-follows relation in graph.

Definition 3.2 (Weight of directly-follows relation)**.** *Given a directly-follows graph G(L), the weight of each directly-follows relation is defined as*

$$Weight(E(A, B)) = \frac{Cardinality(E(A, B))}{Cardinality(E(A, *))}$$

for start activities A, we have

$$Weight(Start(A)) = \frac{Cardinality(Start(A))}{Cardinality(Start(*))}$$

Similarly for end activities B, we have

$$Weight(End(B)) = \frac{Cardinality(End(B))}{Cardinality(End(*))}$$

E(A,) means all edges with source A, E(*,B) means all edges with target B, Start(*) represents all start nodes, and End(*) represents all end nodes.*

After defining the weights of each directly-follows relation, for each directly-follows relation, there are three weights from G_{pos}, G_{neg} and G_{ext}. The following strategy assigns new weight to directly-follows relation to new generated directly-follows graph G_{new}.

Definition 3.3 (Assign new weights to graph G_{new}). *there are three weights from G_{pos}, G_{neg} and G_{ext}, the new weight is*

- *For one directly-follows relation,*

  $$Weight(E_{G_{new}}(A,B)) = Weight(E_{G_{pos}}(A,B)) + Weight(E_{G_{ext}}(A,B)) - Weight(E_{G_{neg}}(A,B))$$

- *For start activities A, we have*

  $$Weight(Start_{G_{new}}(A)) = Weight(Start_{G_{pos}}(A)) + Weight(Start_{G_{ext}}(A)) - Weight(Start_{G_{neg}}(A))$$

- *For end activities B, we have*

  $$Weight(End_{G_{new}}(A)) = Weight(End_{G_{pos}}(A)) + Weight(End_{G_{ext}}(A)) - Weight(End_{G_{neg}}(A))$$

After assigning all the weight to directly-follows relation in G_{new}, we filter out all directly-follows relation in G_{new} with weight less than 0. Then, we transform the G_{new} into process tree bu using Inductive Miner for the next stage.

3.2 Definitions Related To Add Long-term Dependency

Example 1 Consider event log L with labels

$$L = [\langle A, C, E\rangle^{50,pos}, \langle B, C, D\rangle^{50,pos}, \langle A, C, D\rangle^{50,neg}].$$

$\langle A, C, E\rangle^{50,pos}$ means there are 10 traces $\langle A, C, E\rangle$ labeled as positive in event log. Similarly, $\langle A, C, D\rangle^{50,neg}$ represents there are $\langle A, C, D\rangle$ traces at number 50 in event log which have negative labels.

After applying the dfg-algorithm, a model as shown in Figure 2 is discovered. In event log L, B and D has long-term dependency, and A is expected to support only the execution of E, since $< A, C, D >$ is negative and $< A, C, E >$ is positive. However, the model doesn't express those constraints. Obviously, long-term dependency relates the choices structure in process model, such as exclusive choice, loop and or structure. Due to the complexity of or and loop structure, only the long-term dependency in exclusive choice is considered.

The inputs for this algorithm are,

- Repaired model in process tree
- Event log with positive and negative labels

The output of this algorithm is:

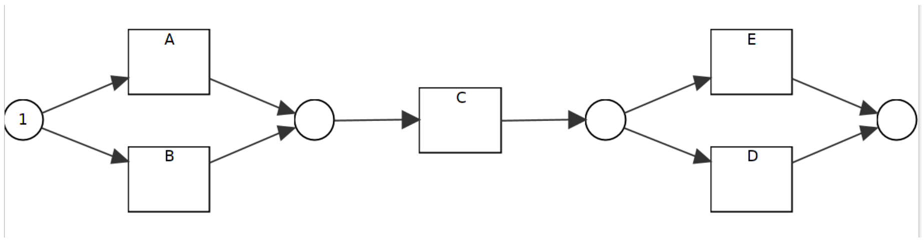


Figure 2: Process model generated from dfg-algorithm

- Repaired model in petri net with long-term dependency

Process tree, as one input for the algorithm, is one common model to interpret business process in process mining. It's a block-structured tree. To specify the process tree with respect to long-term dependency, the following definitions are in need. Firstly, the definitions related to tree are reviewed.

Definition 3.4 (Tree)**.** *Let $\mathscr{E}$ be a finite set of entities, a tree is a collection of entities called nodes, which are connected by edges. A tree T is,*

- *t, with $t \in \mathscr{E}$, t has no outgoing edges*
- *$t(T_1, T_2, ..., T_n)$, with $t \in \mathscr{E}, i, n \in \mathbb{N}, i \leq n, T_i$ is a tree.*

T_i is a child or subtree of $t(T_1, T_2, ..., T_n)$, $t(T_1, T_2, ..., T_n)$ is one parent of T_i, which can be expressed in $P(t(T_1, T_2, ..., T_n), T_i)$. The root of tree is the node without any parent; A tree has only one root. A leaf node is the node which has no children nodes.
For a node in a tree, its ancestor and descendant are defined as:

Definition 3.5 (Ancestor Relation Anc(A,t))**.** *An ancestor of a node t in a tree is a node A, written as $Anc(A, t) \Rightarrow True$, if those conditions hold,*

- *A is a parent of t, written as $P(A, t) \Rightarrow True$, or*
- $\exists t_1, t_2..t_n, n \in \mathscr{E}, i < n, P(A, t_1) \wedge P(t_i, t_{i+1}) \wedge P(t_n, t) \Rightarrow True$

The ancestor of root is empty, while leaf nodes has no descendants. Based on this, we define the ancestors set of a node s.

Definition 3.6 (Ancestors of a node a)**.** *The ancestors set of a node s, Ancestors(s), is defined as:*

$$Ancestors(A) = \{t | Anc(t, s) \Rightarrow True\}$$

Accordingly, descendant relation is given for node t and node s, Des(s,t). If node s is the ancestor of t, then t is a descent of s. $Anc(s, t) \Rightarrow Dec(t, s)$; The set of descendants of node t is Descendants(t).

Definition 3.7 (Least Common Ancestor)**.** *A least common ancestor for node s and node t in a tree is a node n, where*

$$Anc(n, s) \wedge Anc(n, t) \wedge \exists! m Anc(n, m) \wedge Anc(m, s) \wedge Anc(m, t)$$

In process tree, all the leaves are activities in business process, and the middle nodes are operators which represents the relations of all its children nodes[**?**, 9]. This paper uses four operators in context of long-term dependency. The four relations. $\{\rightarrow, \times, \wedge, \circlearrowright\}$ are considered.

Next, we only focus on exclusive xor structure on long-term dependency. As known, long-term dependency is associated with choices. In xor block, it means the choices of each xor branch in xor block. For sake of convenience, we define the xor branch.

$Q = \times(Q_1, Q_2, ..Q_n)$, Q_i is one xor branch with respect to Q, rewritten as $XORB_{Q_i}$ to represent one xor branch Q_i in xor block, and record it $XORB_{Q_i} \in XOR_Q$. For each branch, there exists the begin and end nodes to represent the beginning and end execution of this branch, which is written respectively as Begin($XORB_{Q_i}$) and End($XORB_{Q_i}$).

For convenience of analysis, two properties of xor block, purity and nestedness are demonstrated to express the different structures of xor block according to its branches.

Definition 3.8 (XOR Purity and XOR Nestedness)**.** *The xor block purity and nestedness are defined as following:*

- *A xor block XOR_Q is pure if and only $\forall XORB_X \in XOR_Q, XORB_X$ has no xor block descent, which is named as pure xor branch. Else,*
- *A xor block XOR_Q is nested if $\exists XOR_X, Anc(XOR_Q, XOR_X) \rightarrow True$. Similarly, this xor branch with xor block is nested.*

In the Figure3, xor block Xor(c1,c2) are pure and not nested, since all the xor branches are leaf node, but xor block Xor(a,Seq(b,Xor(c1,c2))) is impure and nested with Xor(c1,c2).

Long-term dependency researches on the dependency of choices in xor block, with observation, actually on the pure xor branch, because nested xor branch has multiple choices, which affect the execution of later process. For two arbitrary pure xor branches, to have long-term dependency, they firstly need to satisfy the conditions: (1) they have an order;(2) they have significant correlation. The order of xor branch follows the same rule of node in process tree which is explained in the following.

Definition 3.9 (Order of nodes in process tree)**.** *Node X is before node Y, written in $X \prec Y$, if X is always executed before Y. In the aspect of process tree structure, $X \prec Y$, if the least common ancestor of X and Y is a sequential node, and X positions before Y.*

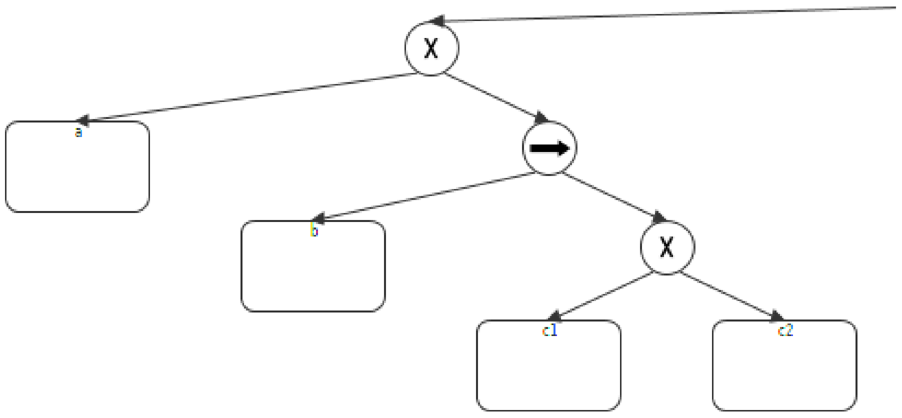


Figure 3: XOR branch variants

The correlation of xor branches is significant if they always happen together. To define it, several concepts are listed at first.

Definition 3.10 (Xor branch frequency)**.** *Xor branch $XORB_X$ frequency in event log l is $F_l(XORB_X)$, the count of traces with the execution of $XORB_X$. For multiple xor branches, the frequency in event log l is defined as the count of traces with all the execution of xor branches $XORB_{Xi}$, written as*

$$F_l(XORB_{X1}, XORB_{X2}, ..., XORB_{Xn})$$

.

After calculation of the frequency of the coexistence of multiple xor branches in positive and negative event log, we get the supported connection of those xor branches, and define the correlation.

Definition 3.11 (Correlation of xor branches)**.** *For two pure xor branches, $XORB_X \prec XORB_Y$, the supported connection is given as*

$$SC(XORB_X, XORB_Y) = F_{pos}(XORB_X, XORB_Y) - F_{neg}(XORB_X, XORB_Y)$$

. If $SC(XORB_X, XORB_Y) > lt - threshold$, then we say $XORB_X$ and $XORB_Y$ have significant correlation.

I did some introduction, it is proven that in some cases, we can't deal with it. When the $XORB\_S \neq LT\_S || XORB\_T \neq LT\_T$.

Another way to avoid this problem is to add duplicated events, but the problem stays the same, so if we want to keep the model fit, we add new event for the discovered long-term dependency, the original, we keep it in the model?? But it is not precise!!! It allows too many choices there, but the model is sound, because we add events on model, we produce and consume tokens from duplicated events, and consumes it later.. Still, it is not so right. But if we choose the events before, we decides the events later, it's true... helps a little...

How to prove it ?? By induction. We use process trees as one internal result in our approach in two factors: (1) they are sound by construction, and can be transferred into sound Petri net models; (2) they are block-structured. which benefits the detection of exclusive relations in model.

- the original model is sound
- after adding one long-term dependency by duplicating the event, we make sure
  duplicated events are added into the xor branches, if it's chosen, then it consumes one token, at this xor branches, then it produces two tokens, one of which is put back again into the To prove the added places and duplicated events between two xor branches will not violate the soundness of model.
-
  - adding duplicated events of one long-term dependency does not violate the soundness of model.
  - To connect the source and target of long-term dependency which are the duplicated events will not violate the soundness.
    One extra place is added to connect the source and target. After executing the source, one token is generated in this place; Due to long-term dependency, only this target is triggered by this token and it consumes this token. No extra token is introduced into this model, so the model keeps sound.
- While keeping the original event in the model, the model is with less precision.
  So I want to solve this problem, and make it preciser by considering the original events into model... One drawback exists there, still... If we use the old and then the

It's about 12 pages. But one problem, some basic information, we forget to give .. Maybe, we can put the related work later!!!

4 Implementation

We can give some screen shoots and describes the work here..

For the implementation, we need to consider the correctness of our methods.

4.1 Detect Directly-follows Graph

4.2 Add Long-term Dependency

This article is used to explain the difficulty I confront about proving the soundness of generated models. In the first phrase of algorithm, the existing model, positive event log and negative event log are used to generate a new directly-follows graph. Based on Inductiver Miner, this graph is transformed into a sound petri net without long-term dependency.

In the next phase, our algorithm focuses on detecting and adding long-term dependency in Petri net. We define, if the supported connection on the set pair of xor branches is over a threshold, pair has significant correlation.Therefore, this pair has long-term dependency.

During the implementation, it comes clear that supported connection only on the positive and negative event log is not enough, since the existing model can keep some directly-follows relations about xor branches which do not show in the positive event log or shows only in negative event log. Consequently, when we detect this long-term dependency on those xor branches, there is no evidence of long-term dependency on those xor branches. It results in an unsound model, since those xor branches can't get fired to consume the tokens generated from the choices before.

5 Problem Description

In this section, we use one simple example to deepen our understanding of this problem. Given one sound model shown in Fig 4 ready for adding long-term dependency on it.

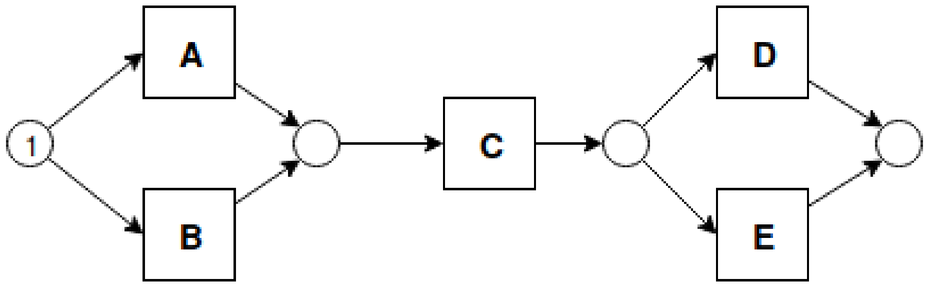


Figure 4: Simpe Example

There are 7 sorts of long-term dependency that is able to happen in this model as listed in the following. Before this, we need to define some concepts at the sake of convenience.

Definition 5.1 (Source and Target of Long-term Dependency)**.** *We define the source set of long-term dependency is* $LT_S := \{X|\exists X, X \rightsquigarrow Y \in LT\}$*, and target set is* $LT_T := \{Y|\exists Y, X \rightsquigarrow Y \in LT\}$*.*
For one xor branch $X \in XORB_S$*, the target xor branch set relative to it with long-term dependency is defined as:* $LT_T(X) = \{Y|\exists Y, X \rightsquigarrow Y \in LT\}$ *Respectively, the source xor branch relative to one xor branch in target is* $LT_S(Y) = \{X|\exists X, X \rightsquigarrow Y \in LT\}$

At the same time, we use $XORB_S$ and $XORB_T$ to represent the set of xor branches for source and target xor block.

1. $LT = \{A \rightsquigarrow D, A \rightsquigarrow E, B \rightsquigarrow D, B \rightsquigarrow D\}$.
   $LT_S = \{A, B\}, LT_T = \{D, E\}, |LT| = |XORB_S| * |XORB_T|$, which means long-term dependency has all combinations of source and target xor branches.

2. $LT = \{A \rightsquigarrow D, A \rightsquigarrow E, B \rightsquigarrow E\}$.
   $LT_S = \{A, B\}, LT_T = \{D, E\}$ $LT_S = XORB_S$ and $LT_T = XORB_T, |LT| < |XORB_S| * |XORB_T|$. it doesn't cover all combinations. But for one xor branch $X \in XORB_S, LT_T(X) = XORB_T$, it has all the full long-term dependency with $XORB_T$.

3. $LT = \{A \rightsquigarrow D, B \rightsquigarrow E\}$.
   $LT_S = \{A, B\}, LT_T = \{D, E\}$ $LT_S = XORB_S$ and $LT_T = XORB_T, |LT| < |XORB_S| * |XORB_T|$. For all xor branch $X \in XORB_S, LT_T(X) \subsetneq XORB_T$, none of xor branch X has long-term dependency with $XORB_T$.

4. $LT = \{A \rightsquigarrow D, B \rightsquigarrow D\}$.
   $LT_S = XORB_S, LT_T \subsetneq XORB_T$. There exists at least one xor branch $Y \in XORB_T$ which has no long-term dependency on it.

5. $LT = \{A \rightsquigarrow D, A \rightsquigarrow E\}$.
   $LT_S \subsetneq XORB_S, LT_T = XORB_T$. There exists at least one xor branch in source $X \in XORB_S$ which has no long-term dependency on it.

6. $LT = \{A \rightsquigarrow E\}$.
   $LT_S \subsetneq XORB_S, LT_T \subsetneq XORB_T$. There exists at least one xor branch in source $X \in XORB_S$ and one xor target xor branch which has no long-term dependency on it.

7. $\emptyset$. There is no long-term dependency on this set.

For situation 1, it's full connected and xor branches can be chosen freely. So there is no need to add explicit connection on model to represent long-term dependency, therefore the model keeps the same as original. For Situation 2 and 3, if $LT_S = XORB_S, LT_T = XORB_T$, then we can create an sound model by adding silent transitions. If not, then we need to use the duplicated transitions to create sound model. But before, we need to prove its soundness. For situation 4, 5 and 6, there is no way to prove the soundness even by adding duplicated events. For situation 7, I don't think it exists... but if the negative instances has strong effect on it, and make all the connection disappear in long-term dependency but xor branches kept after dfg method??? To prove the model sound, we need to prove the four conditions.

- safeness. Places cannot hold multiple tokens at the same time
- proper completion. If the sink place is marked, all other places are empty.

- option to complete. It is always possible to reach the final marking just for the sink place/
- no dead part. For any transition there is a path from source to sink place through it.

6 Discussion of Multiple Implementations

Until now, multiple methods have been tried to add long-term dependency while trying to keep model sound. Next, we introduce those methods, and give proof if it change its soundness by adding long-term dependency for two xor block in Petri net.

6.1 Only Silent Transitions without Duplicated Events

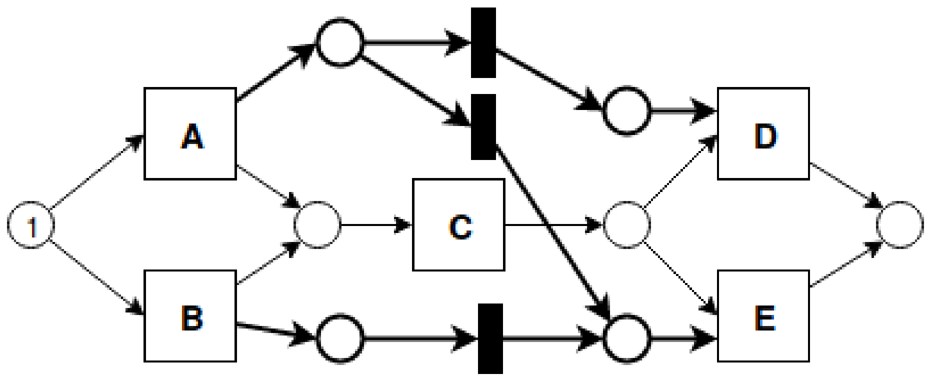
We add long-term dependency on model by injecting silent transitions and extra places into Petri net. An example is given to describe this method. This method can not ensure the soundness of model with long-term dependency for situation 4 and 5.
Proof In Fig **??**, it violated the proper completion, and in Fig **??**, it can not fire D, E if B is chosen to execute at first.

6.2 Full Duplicated Events Without Silent Transitions

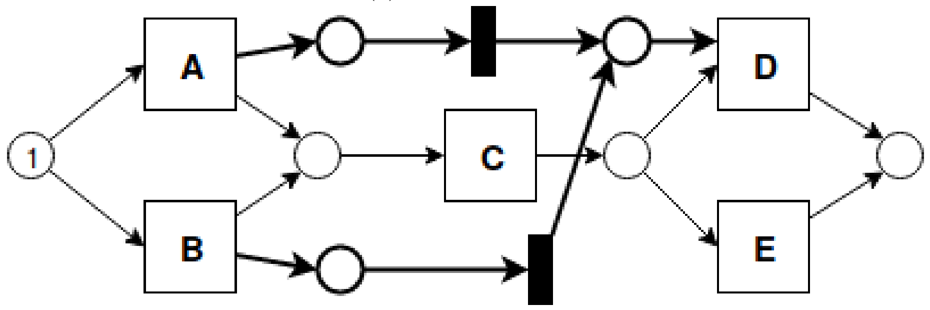
For xor branches which are related to long-term dependency, we duplicate those xor branches in the original xor block, and then connect them together by adding extra place. An example is given to briefly explain it. In this method, for every item in $LT = \{A \rightsquigarrow D, A \rightsquigarrow E, B \rightsquigarrow D\}$, we duplicate the source and target in corresponding xor block, and then connect the source and target by one explicit place. This method doen not keep the model sound.
Proof: If original transitions at source xor block are chosen, the execution path is limited to the original transitions in target xor block. It keeps the model sound. If the duplicated transitions at source xor block are chosen (which is colored), two tokens are generated, one for the xor-joint place, one for the extra place. When it comes to target xor block, two kind of transitions are enabled, one is the original transitions, one is the transitions with long-term dependency. But if original transitions are chosen, it violates the proper completion condition for soundness.

6.3 Semi Duplicated Events Without Silent Transitions

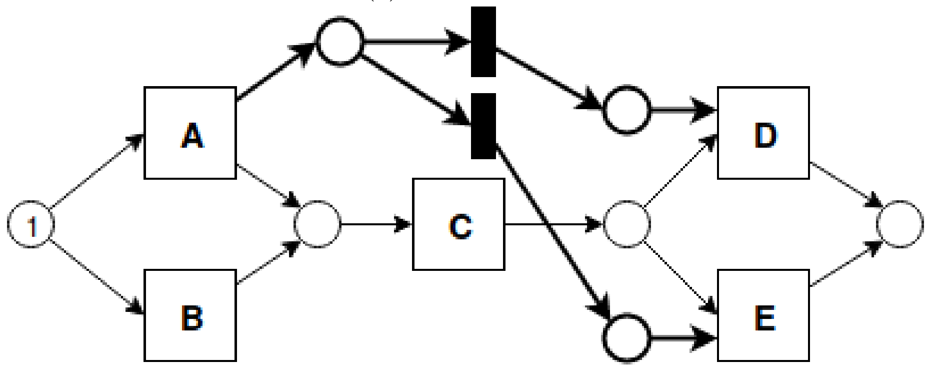
This method uses the original transition to express long-term dependency, only adding duplicated events if existing ones are already used for expressing long-term dependency. To express Situation 2 and 3, it create sound model, but for situation 4,5 and 6, it can't guarantee the soundness as shown in the Fig 7b.



(a) For situation 2



(b) For situation 4



(c) For situation 5

Figure 5: Silent Events for Long-term Dependency

Because the transitions in target xor block are not enabled if B is chosen to execute.

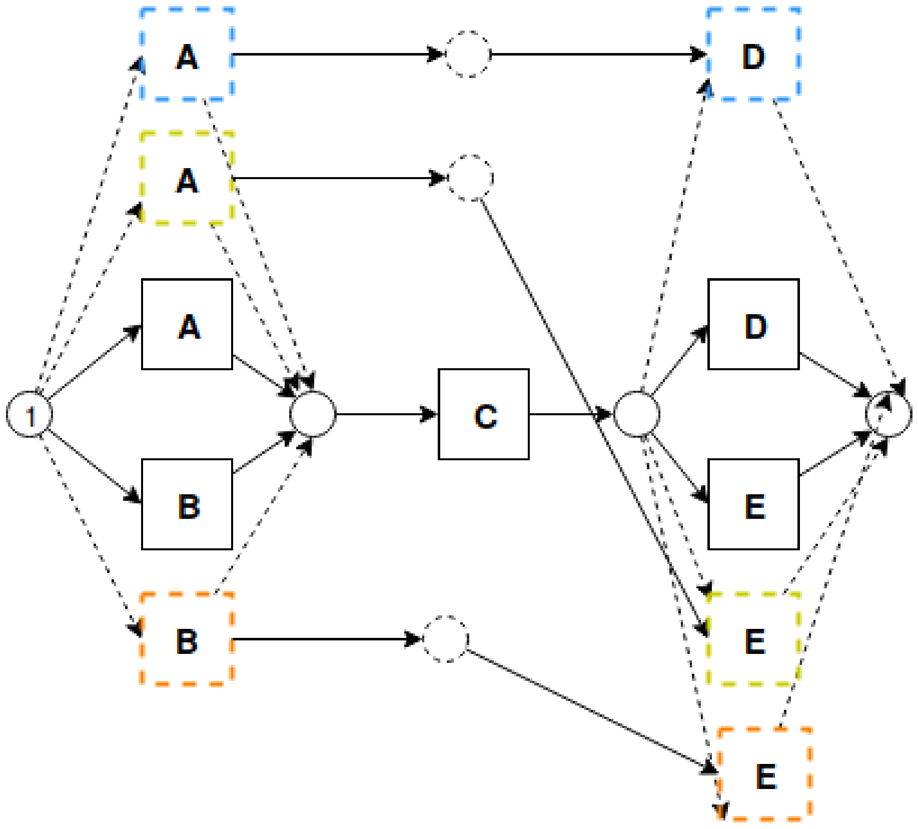


Figure 6: Full Duplicated Events for Long-term Dependency For situation

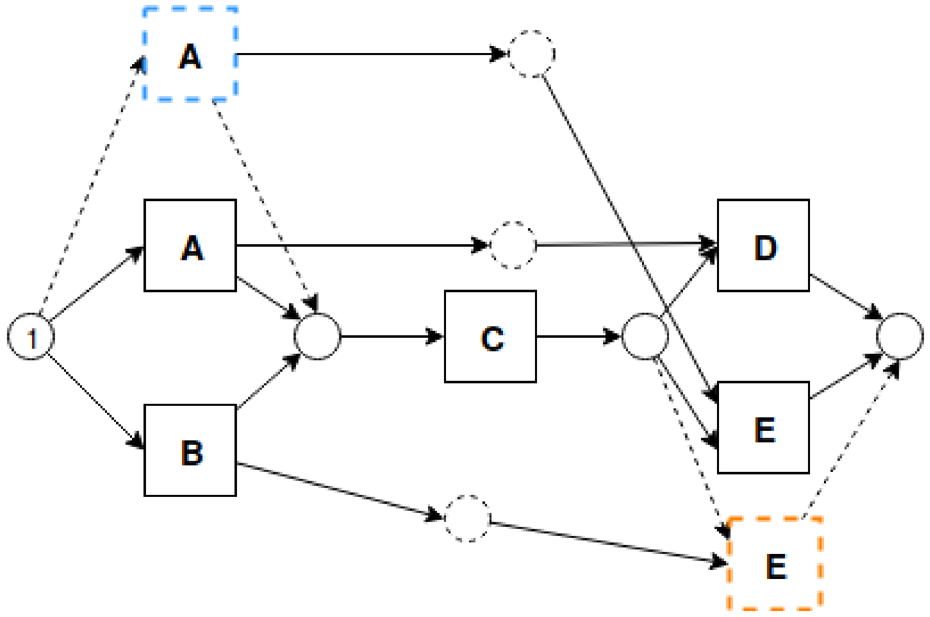
7 Solution Trials

To make the generated Petri net sound, there are two thinking ways, one is to make sure that only situation 2 and 3 happen when generating those models. One is to propose a new method to keep it sound. In the next subsection, we try to solve this problem by giving constraints to only allow situation 1 2, and 3 available.

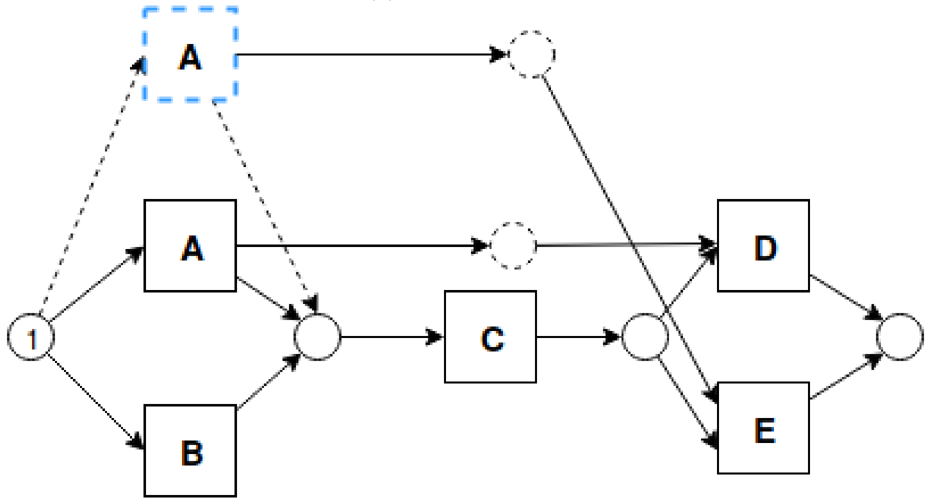
7.1 Add Constraints

By adding constraints, we make sure only situations 1,2,3 happen when adding long-term dependency. It requires that after dfg method considers the negative, positive and existing model to delete the unused xor branches for long-term dependency. Before doing it, we recover some definitions for long-term dependency detection. The definition 3.11 is rephrased into weight.

Definition 7.1 (Rephrased Correlation of xor branch)**.** *The correlation for two*



(a) For situation 2



(b) For situation 5

Figure 7: Semi Duplicated Events for Long-term Dependency For situation

branches is expressed into

$$Wlt(XORB_X, XORB_Y) = Wlt_{ext}(XORB_X, XORB_Y) + Wlt_{pos}(XORB_X, XORB_Y)$$

$$-Wlt_{neg}(XORB_X, XORB_Y)$$

, where $Wlt_{ext}(XORB_X, XORB_Y) = \frac{1}{|XORB_{Y*}|}$, $|XORB_{Y*}|$ *means the number of possible directly-follows xor branche set* $XORB_{Y*} = \{XORB_{Y1}, XORB_{Y2}, ...XORB_{Yn}\}$ *after* $XORB_X$.

$Wlt_{pos}(XORB_X, XORB_Y) = \frac{F_{pos}(XORB_X, XORB_Y)}{F_{pos}(XORB_X, *)}$,
$Wlt_{neg}(XORB_X, XORB_Y) = \frac{F_{neg}(XORB_X, XORB_Y)}{F_{neg}(XORB_X, *)}$,

The $F_{pos}(XORB_X, XORB_Y)$ and $F_{neg}(XORB_X, XORB_Y)$ are the frequency of the coexistence of $XORB_X$ and $XORB_Y$, respectively in positive and negative event log.

With this rephrased definition, to make the model sound, we need to prove, if there is a xor branch $XORB_Y$ in the generated process tree, there must exist one long-term dependency related to it, $\exists XORB_X, Wlt(XORB_X, XORB_Y) >$ lt-threshold. We formalize this problem. Else, the model can't be sound!!

Proposition 7.1. *Given a process tree, a pair of xor branch set,* (B_A, B_B) *with* $B_A = XORB_{X1}, XORB_{X2}, ...XORB_{Xm}, B_B = XORB_{Y1}, XORB_{Y2}, ...XORB_{Yn}$, *the obligatory part between* B_A *and* B_B *is marked M, it is to prove::*
$\forall XORB_Y \in B_B$, *if* $W(M, XORB_Yj) >$ *threshold,*
then there exists one $XORB_Xi \in B_A$ *with*

$$Wlt(XORB_Xi, XORB_Yj) > lt\text{-}threshold$$

.

Given a simplified scenes, it is listed in Fig 8. M is an obligatory path from the set $\{X1, X2, ..Xm\}$ to $\{Y1, Y2, ..Yn\}$. If there exists directly-follows relation of M and Y1, then there must exist one long-term dependency of Xi and Y1.

The definition of $W(M, XORB_{Yj})$ is reviewed below.

Definition 7.2 (Assign new weights to graph G_{new})**.** *there are three weights from* G_{pos}, G_{neg} *and* G_{ext}, *the new weight is*

- *For one directly-follows relation,*

  $$W(E_{G_{new}}(A, B)) = Weight(E_{G_{pos}}(A, B)) + Weight(E_{G_{ext}}(A, B)) - Weight(E_{G_{neg}}(A, B))$$

- *Given a directly-follows graph G(L), the weight of each directly-follows relation is defined as*

  $$Weight(E(A, B)) = \frac{Cardinality(E(A, B))}{Cardinality(E(A, *))}$$

When prove by contradiction, we assume that the opposite proposition is true. If it shows that such an assumption leads to a contradiction, then the original proposition is valid.

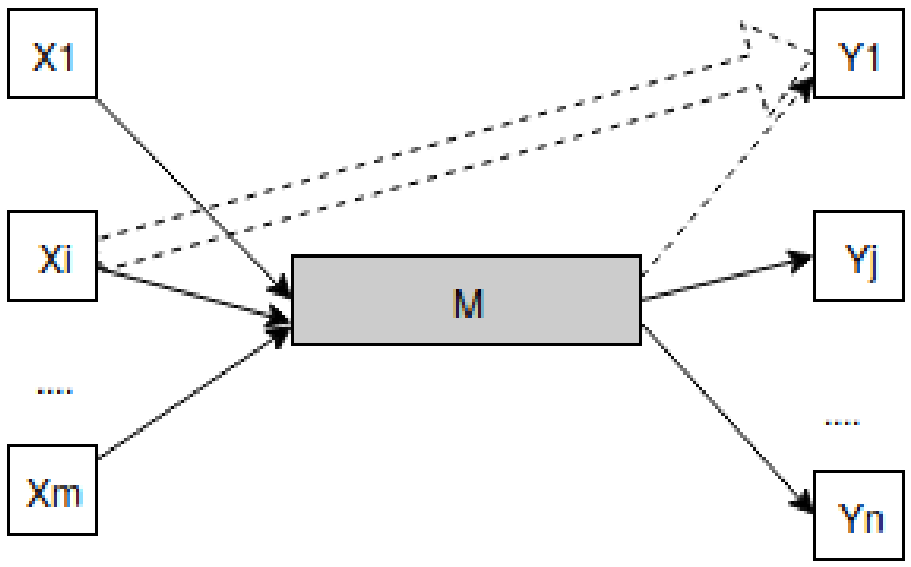


Figure 8: Simplified Graph MOdel

Proposition 7.2. *for one xor branch $XORB_Y$, if $W(M, XORB_{Yj}) > threshold$, there exists no one $XORB_{Xi}$ with*

$$Wlt(XORB_{Xi}, XORB_{Yj}) < lt\text{-}threshold$$

.

Or we change to another thinking way to get the relation of threshold and lt-threshold, such that we have the theorem valid. Then we rephrase the question into

Proposition 7.3 (Another way of thinking)**.** *Given a process tree, a pair of xor branch set, (B_A, B_B) with $B_A = \{XORB_{X1}, XORB_{X2}, ...XORB_{Xm}\}, B_B = \{XORB_{Y1}, XORB_{Y2}, ...XORB_{Yn}\}$, the obligatory part between B_A and B_B is marked M. If,*
for one xor branch $XORB_{Yj}$, if $W(M, XORB_{Yj}) > threshold$,
there exists one $XORB_Xi$ with

$$Wlt(XORB_{Xi}, XORB_{Yj}) > lt\text{-}threshold$$

What is the relation of threshold and lt-threshold??

If we expand the theorem, we need to prove

Proposition 7.4 (Relation of threshold and lt-threshold)**.** *What is the relation* *of threshold and lt-threshold, to make the following proposition valid. If*

$$W(M, XORB_{Yj}) > threshold$$
$$W(M, XORB_{Yj}) = Weight(E_{G_{pos}}(M, XORB_{Yj}))$$
$$+Weight(E_{G_{ext}}(M, XORB_{Yj})) - Weight(E_{G_{neg}}(M, XORB_{Yj}))$$
$$\frac{1}{|Y*|} + \frac{\sum_{Xi} Cardinality(M, Yj|Xi)}{\sum_{Xi} Cardinality(M, Y*|Xi)} - \frac{\sum_{Xi} Cardinality(M, Yj|Xi)\prime}{\sum_{Xi} Cardinality(M, Y*|Xi)\prime > threshold}$$

Then, exist one Yj with

$$Wlt(Xi, Yj) > \textit{lt-threshold}$$
$$Wltext(Xi, Yj) + Wltpos(Xi, Yj) - Wltneg(Xi, Yj) > \textit{lt-threshold}$$
$$\frac{1}{|Y*|} + \frac{Cardinality(M, Yj|Xi)}{Cardinality(M, Y*|Xi)} - \frac{Cardinality(M, Yj|Xi)\prime}{Cardinality(M, Y*|Xi)\prime} > \textit{lt-threshold}$$

Or ***there is a contradiction when all Yj***

$$Wlt(Xi, Yj) < \textit{lt-threshold}$$
$$\sum_{Xi} Wlt(Xi, Yj) < |X*| \bullet \textit{lt-threshold}$$
$$\frac{|X*|}{|Y*|} + \sum_{Xi} \frac{Cardinality(M, Yj|Xi)}{Cardinality(M, Y*|Xi)} - \sum_{Xi} \frac{Cardinality(M, Yj|Xi)\prime}{Cardinality(M, Y*|Xi)\prime} < |X*| \bullet \textit{lt-threshold}$$

$Cardinality(M, Yj|Xi)$ means the frequency of coexistence of M and Yj given Xi in the trace in positive, while $Cardinality(M, Yj|Xi)\prime$ represents the frequency in negative. $Cardinality(M, Y*|Xi)$ is the sum frequency of set $Y1, ..Yj, ..Yn$, it equals to

$$Cardinality(M, Y*|Xi) = \sum_{Yi} Cardinality(M, Yj|Xi)$$

If we set them into zero, there is a lot of existing edges kept into the old method with no evidence in event log to support the connection.

7.2 New Method Proposal

we can add silent transition to make the model sound??? To combine the duplicated events and silent transitions. But how is it and how to prove its soundness??

The problem is around situation 4 ,5 and 6. If we have the negative evidence to avoid the long-term dependency. For situation 4, $LT = \{A \rightsquigarrow D, B \rightsquigarrow D\}$. It shows in those two situations.

- If E is kept due to the existing model, and no positive and negative evidence is given for long-term dependency, then Wlt is positive to add long-term dependency on it.
- If long-term dependency is prohibited due to the negative instances, so we need to delete the execution of E by substituting it by silent transitions,

and then deleted from the model. Further, after deleting those events, we check the long-term dependency again. If it have full long-term dependency combination, no need to explicitly address them in the model. Else, the explicit long-term dependency is kept in the model.

We can solve it by adding silent transitions in the xor blocks to represent the long-tem dependency.

More details here are around 6 pages

8 Evaluation

9 Discussion

Some tests are given, one is simulated data, and also some examples are given to explain the advantages. Later, real data is listed here..

It's about 3 pages or 4 pages with graphs.

10 Conclusion

One page is enough.

References

[1] W. Van Der Aalst, *Process mining: discovery, conformance and enhancement of business processes*, vol. 2. Springer, 2011.

[2] M. Ghasemi and D. Amyot, “Process mining in healthcare: a systematised literature review,” 2016.

[3] D. Fahland and W. M. van der Aalst, “Repairing process models to reflect reality,” in *International Conference on Business Process Management*, pp. 229–245, Springer, 2012.

[4] M. Dees, M. de Leoni, and F. Mannhardt, “Enhancing process models to improve business performance: a methodology and case studies,” in *OTM Confederated International Conferences” On the Move to Meaningful Internet Systems”*, pp. 232–251, Springer, 2017.

[5] D. Fahland and W. M. van der Aalst, “Model repairaligning process models to reality,” *Information Systems*, vol. 47, pp. 220–243, 2015.

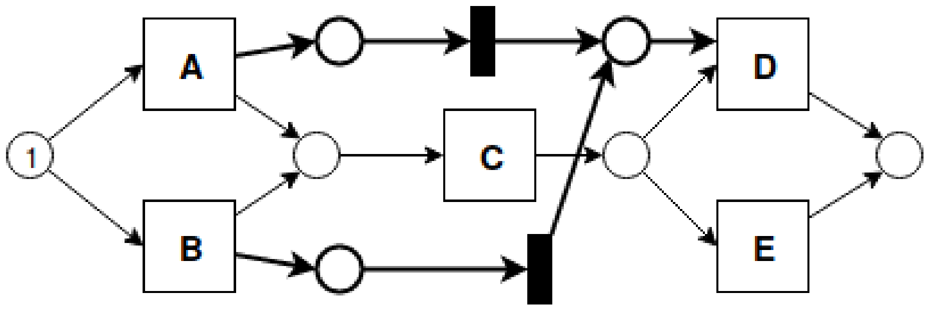
[6] S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens, “Robust process discovery with artificial negative events,” *Journal of Machine Learning Research*, vol. 10, no. Jun, pp. 1305–1340, 2009.

[7] S. K. vanden Broucke, J. De Weerdt, J. Vanthienen, and B. Baesens, “Determining process model precision and generalization with weighted artificial negative events,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 8, pp. 1877–1889, 2014.

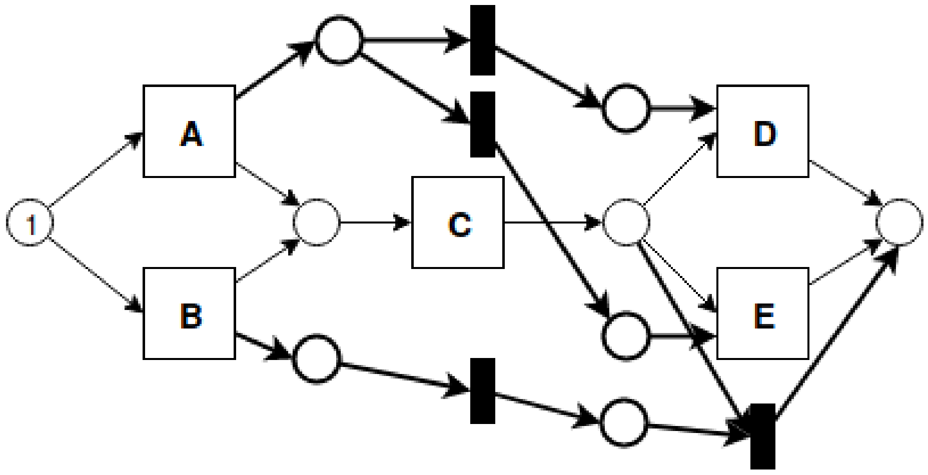
[8] H. Ponce-de León, J. Carmona, and S. K. vanden Broucke, “Incorporating negative information in process discovery,” in *International Conference on Business Process Management*, pp. 126–143, Springer, 2016.

[9] S. J. Leemans, D. Fahland, and W. M. van der Aalst, “Discovering block-structured process models from event logs-a constructive approach,” in *International conference on applications and theory of Petri nets and concurrency*, pp. 311–329, Springer, 2013.

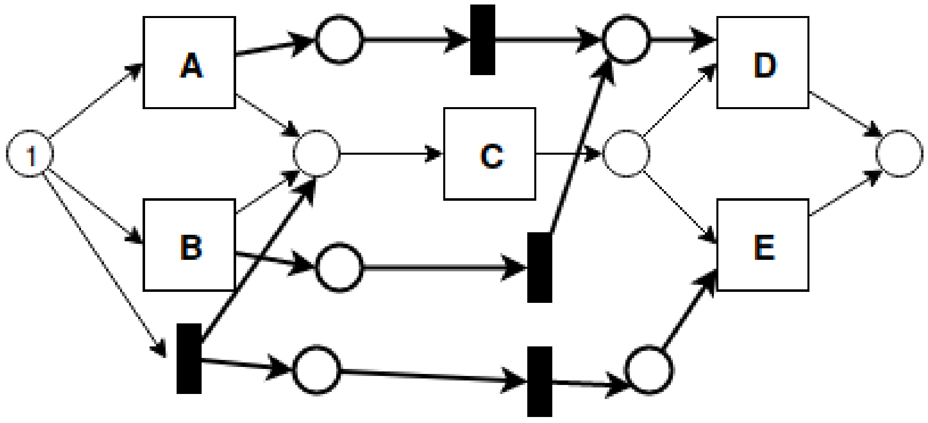
[10] S. J. Leemans, D. Fahland, and W. M. van der Aalst, “Discovering block-structured process models from incomplete event logs,” in *International Conference on Applications and Theory of Petri Nets and Concurrency*, pp. 91–110, Springer, 2014.



(a) For situation 4



(b) For situation 4



(c) For situation 5

Figure 9: Silent Events for Long-term Dependency