PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
SCHOOL OF ENGINEERING

A FEEDBACK-BASED FRAMEWORK FOR PROCESS ENHANCEMENT OF CAUSAL NETS

NICOLÁS JAVIER PIZARRO DE LA FUENTE

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Advisor:
MARCOS SEPÚLVEDA FERNÁNDEZ

Santiago de Chile, July 2015

PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
SCHOOL OF ENGINEERING

A FEEDBACK-BASED FRAMEWORK FOR PROCESS ENHANCEMENT OF CAUSAL NETS

NICOLÁS JAVIER PIZARRO DE LA FUENTE

Members of the Committee:
MARCOS SEPÚLVEDA FERNÁNDEZ
JORGE MUÑOZ GAMA
BERNHARD HITPASS HEYL
SERGIO MATURANA VALDERRAMA

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Santiago de Chile, July 2015

To my family and friends. Without your support none of this would be possible.

ACKNOWLEDGEMENTS

I would like to thank professors Marcos Sepúlveda and Jorge Muñoz Gama for their valuable guidance throughout this research.

I am also grateful to my friends Gustavo Pizarro, Mauricio Arriagada, Andrea Vásquez and Gabriel Vidal for their help and continued support.

Finally, I want to thank my parents, Paulina and Jaime, my siblings, Beatriz, Daniela and Vicente, my family, friends and all those who have encouraged me to expand my knowledge and pursue my life goals.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iv

LIST OF FIGURES vii

LIST OF TABLES viii

ABSTRACT ix

RESUMEN x

1. INTRODUCTION 1
   1.1. Technological overview 1
   1.2. Process mining 4
      1.2.1. Event logs 5
      1.2.2. Process models 6
      1.2.3. Discovery, Conformance and Enhancement 8
      1.2.4. Process mining perspectives 9
   1.3. Challenges of current algorithms 10
   1.4. A Feedback-based Framework for Process Enhancement using Causal nets 11
   1.5. Thesis Outline 12

2. PRELIMINARIES 14
   2.1. Multisets, traces and logs 14
   2.2. Process models and causal nets 14

3. DUAL-ACTIVITY FEEDBACK ENHANCEMENT 20
   3.1. Causality ($\rightarrow$) 20
   3.2. Parallelism ($||$) 21
   3.3. Indifference (#) 23

4. ACTIVITY-BLOCK FEEDBACK ENHANCEMENT 25

4.1. Introduction to SESEs and RPST 25
4.2. Node-extended RPST 27
4.3. SESE-based Node Collapsing 28
4.4. RPST-aided SESE Collapse 31

5. STRATEGIES FOR FEEDBACK RECOMMENDATION 33
5.1. Log-based Feedback Recommendation 33
5.1.1. Fitting Traces and Common Paths 33
5.1.2. Unfitting traces and Deviation Points 33
5.1.3. Decomposition-based Alignments 35
5.2. Model-based Feedback Recommendation 35

6. IMPLEMENTATION AND EXPERIMENTAL EVALUATION 37
6.1. ProM Implementation 37
6.2. Experimental Evaluation 38
6.2.1. Performance evaluation 38
6.2.2. User evaluation 39

7. RELATED WORK 42

8. CONCLUSIONS AND FUTURE WORK 45

References 46

LIST OF FIGURES

1.1 Moore's law. 1
1.2 Storage cost over time. 2
1.3 Gartner Hype Cycle for Emergent Technologies. 3
1.4 Process mining categories. 4
1.5 UML diagram of XES classes. 6
1.6 Process model notations. 8
1.7 Process mining categories. 9
1.8 Theory/reality gap in process mining. 10
1.9 Process model for car technical reviews. 11
1.10 Iterative Feedback Enhancement. 12
2.1 Causal net example. 14
3.1 Enhanced c-net with a dual-activity operation. 23
4.1 Directed multi-graph. 25
4.2 SESE decomposition. 27
4.3 RPST. 27
4.4 Node-extended RPST. 28
4.5 Collapsed SESEs. 29
4.6 Enhanced c-net with a activity-block operation. 31
6.1 FeedbackRepair plugin in ProM. 37
6.2 User interface of FeedbackRepair plugin in ProM. 38
6.3 Users evaluation. 40

LIST OF TABLES

1.1 Event log for a car technical review process. 5

2.1 C-net input/output functions. 16

5.1 Deviation point count. 35

5.2 Binding count. 36

6.1 Time results for performance evaluation. 39

6.2 Results for user evaluation. 40

ABSTRACT

Process mining techniques aim to generate useful information from event logs. For instance, using process discovery tools, a process model indicating the order of activities can be generated from an event log. However, event logs are generally biased, as they represent just a sample of all possible behaviour. Even though some repair techniques have been developed, they still rely on log information to work. As no other data-source allows current algorithms to improve the generated models, this paper focuses on providing a user-powered model repair framework. This is done by an iterative approach, defining a set of operations that allows users to provide additional information that can be used to enhance an existing model.

This work uses causal nets as a base process modeling notation. User feedback can be expressed using three binary operations: causality ($\rightarrow$), parallelism ($\|$) and indifference ($\#$). Each one of them represents the link between two activities according to the user. For large and complex processes, this paper also proposes a sub-process collapsing approach based on *SESE* (Single Entry Single Exit). The resulting model complies with user-provided operations, while maintaining the initial process structure. This approach has been implemented in the ProM framework and has been tested with several event logs.

Keywords: Process Mining, Process Enhancement, Process Repair, User Feedback, Causal Nets, SESE, RPST.

RESUMEN

Las técnicas de minería de procesos se utilizan para extraer información útil desde los logs de eventos. Por ejemplo, usando herramientas de descubrimiento de procesos, es factible generar un modelo de proceso que indica el orden de ejecución de las distintas actividades. Sin embargo, los logs de eventos no son siempre objetivos, dado que contienen sólo una muestra de todo el comportamiento posible. Aún cuando se han desarrollado algunas técnicas que permiten reparar estos problemas, ellas se siguen basando en la información del log para funcionar. Dado que ninguna otra fuente de datos permite a los algoritmos existentes mejorar los modelos generados, este trabajo se centra en la creación de una metodología de reparación de modelos de proceso en base a comentarios de usuarios. Esto se realiza iterativamente, definiendo un conjunto de operaciones que permiten a los usuarios proporcionar información adicional, la cual a su vez permite reparar un modelo existente.

Este trabajo utiliza redes causales (causal nets) como notación base sobre la cual incorporar los comentarios del usuario, usando tres operadores binarios: causalidad ($\rightarrow$), paralelismo ($\|$) e indiferencia ($\#$). Cada uno de ellos representa la relación entre dos actividades según un usuario. Para procesos complejos y con un mayor número de actividades, este trabajo propone además una agrupación mediante subprocesos basada en SESE (Single Entry Single Exit - Entrada Unica Salida Unica). El modelo resultante satisface las operaciones entregadas por los usuarios, manteniendo a su vez la estructura inicial del proceso. Esta metodología ha sido implementada en el programa ProM y fue validada mediante pruebas de usuario.

Palabras Claves: Minera de Procesos, Mejoramiento de Procesos, Reparación de Procesos, Redes Causales, SESE, RPST.

1. INTRODUCTION

Eric Schmidt, executive chairman of Google, stated in 2010 that ”every two days now we create as much information as we did from the dawn of civilization up until 2003. Thats something like five exabytes of data”. This enormous amount of data has no value by itself. In order to make full use of it, we need to process and transform it into information. This task may appear to be nearly impossible, but technology has become a powerful ally in this matter.

1.1. Technological overview

Computing capabilities are increasing exponentially. Gordon Moore, founder of Intel Corporation and Fairchild Semiconductor, stated in 1965 that the number of components per integrated circuit would double each year. This was called *Moore's law* (see Figure 1.1). Even though a decade later he realized that this doubling phenomenon actually happened every two years, his prediction was by far the most accurate, as no one would have thought such an explosive growth at the time was possible.

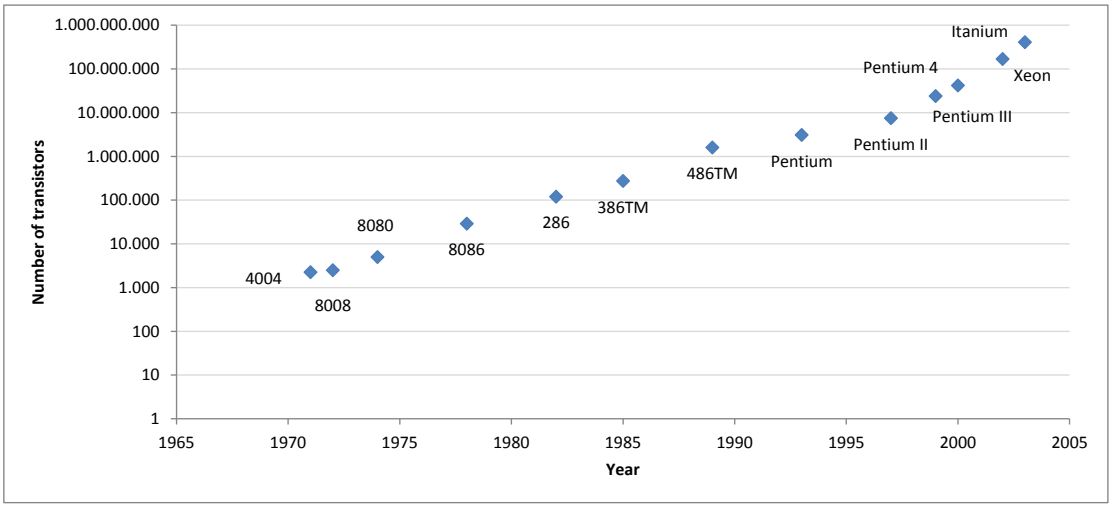


FIGURE 1.1. Moore's law: Number of transistors of popular Intel processors[1].

[1]Compiled information retrieved from Intel website (http://www.intel.com/pressroom/kits/quickrefyr.htm).

This technology evolution also made storage hardware cost to decrease every year (see Figure 1.2). Over the past 30 years the cost per gigabyte has gone down to half every 14 months on average. A major milestone was achieved during 2004, where the cost per gigabyte went below the one dollar threshold. Notice that for Figures 1.1 and 1.2, the y-axis is presented on a logarithmic scale.

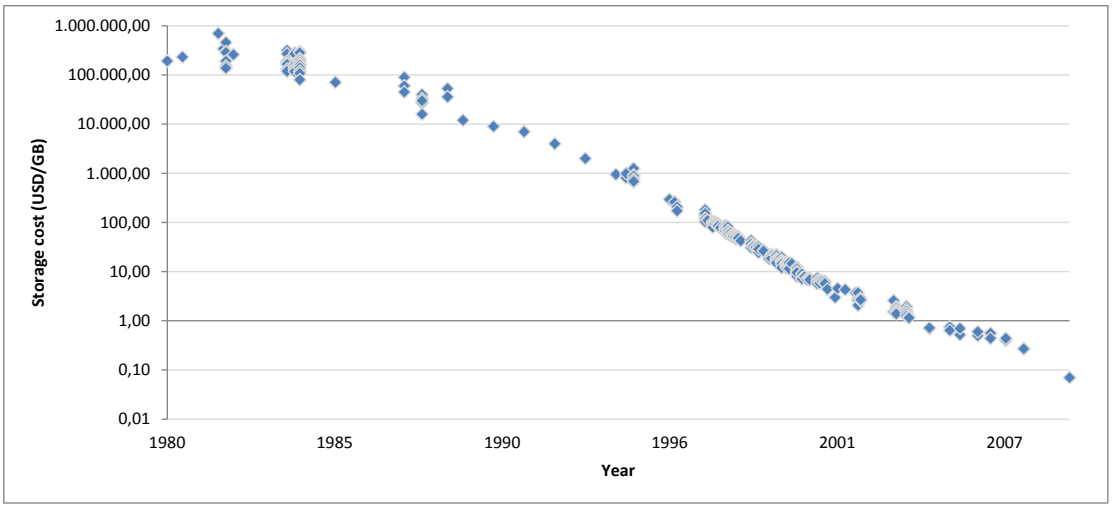


FIGURE 1.2. Storage cost over time[2].

This has enabled the possibility to store every piece of data and to transform it into information using computing capabilities. The discipline in charge of analyzing and producing useful information from data is known as *data science*. In the context of large data sources, applying data science algorithms is known as *big data*.

Enterprises are not strangers to this phenomenon, as their information systems are also recording large amounts of data related to their internal processes. Some companies are currently adopting some of these techniques, but most of them are not mature enough. As Gartner Hype Cycle for Emergent Technologies shows on Figure 1.3, data science and big data are still on early stages of adoption. However, more and more enterprises are willing to adopt these techniques, because they know that accuracy in data science lead to more

[2]Compiled information retrieved from Historical Notes about the Cost of Hard Drive Storage Space (http://ns1758.ca/winch/winchest.html).

confident decision making. Better decisions can mean greater operational efficiency, cost reductions or reduced risk at the long run.

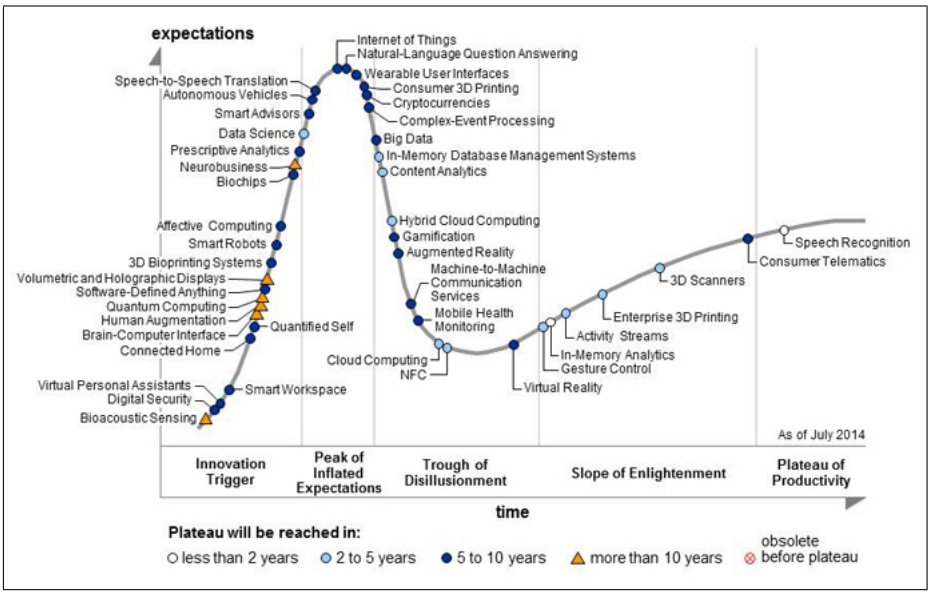


FIGURE 1.3. Gartner Hype Cycle for Emergent Technologies.

1.2. Process mining

During the past decade, several common problems of *Business Process Management (BPM)* have been approached by a discipline called *Process Mining*. Process mining is a branch of *data mining*, both of which are part of the *data science* discipline. While data science reviews all types of databases, process mining seeks to apply diverse algorithms only to databases of operational processes information, also known as *event logs*.

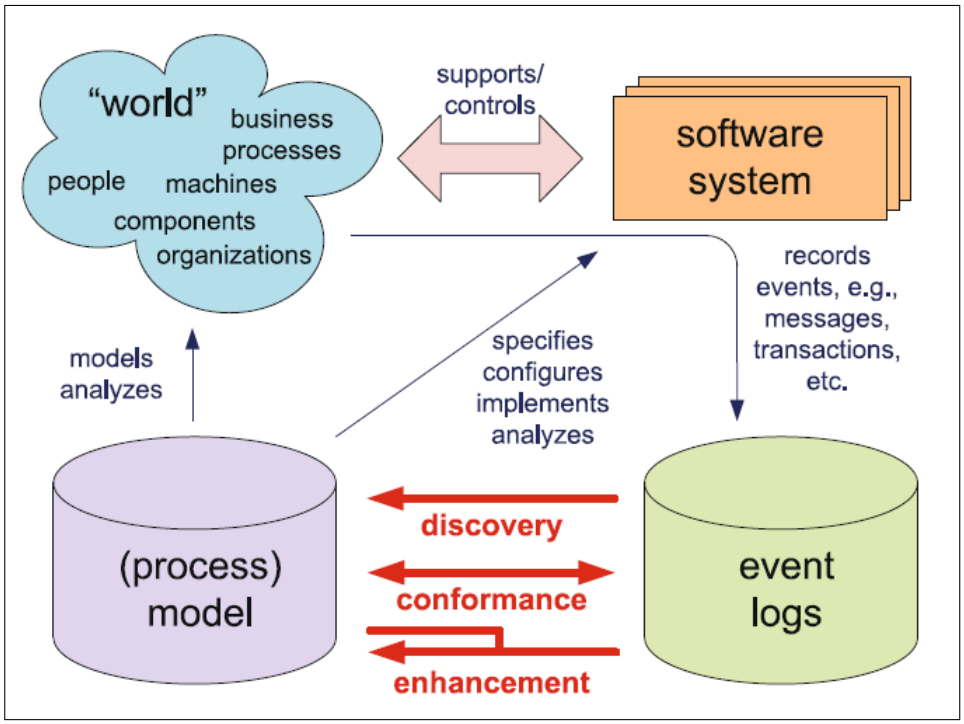


FIGURE 1.4. Process mining categories and their interaction with processes[3].

As Figure 1.4 shows, process mining establishes links between the actual processes and their data on the one hand and process models on the other hand. The idea is to discover, monitor and improve real processes by extracting knowledge from these event logs.

[3]Extracted from (Aalst,2011b).

1.2.1. Event logs

Event logs are the input for almost every process mining algorithm. Event logs contain information about executions of a process and may answer the question ”who made what and when?”. Minimum data contains the following fields.

- Case identification, also known as instance or trace (e.g. 101)
- Activity (e.g. Review performance analysis)
- Timestamp (e.g. 05-06-2015 07:43:22)
- Resource (e.g. Nicolás, Marcos or Jorge)

However, many other fields can be attached to an event, like cost (i.e. monetary cost of performing an activity) or completion time (i.e. elapsed time from start to end). An example event log can be found in Table 1.1.

TABLE 1.1. Event log for a car technical review process.

Case Id.	*Activity*	*Timestamp*	*Resource*
001	Register vehicle	05-06-2015 07:43:22	Nicolás
001	Generate invoice	05-06-2015 08:10:15	Jorge
002	Register vehicle	05-06-2015 08:12:42	Marcos
001	Generate payment	05-06-2015 08:13:55	Jorge
002	Illumination test	05-06-2015 08:20:36	Nicolás
002	Suspension test	05-06-2015 08:25:59	Jorge
...	...	...	...
986	Car delivery	30-06-2015 17:45:28	Marcos

Event logs are stored in a format called XES (Extended Event Stream). XES is an XML-based standard for event logs known by its simplicity, flexibility and extensibility.

As Figure 1.5 shows, an XES document (i.e., XML file) contains one log consisting of any number of traces. Each trace describes a sequential list of events corresponding to a particular case. The log, its traces, and its events may have any number of attributes.

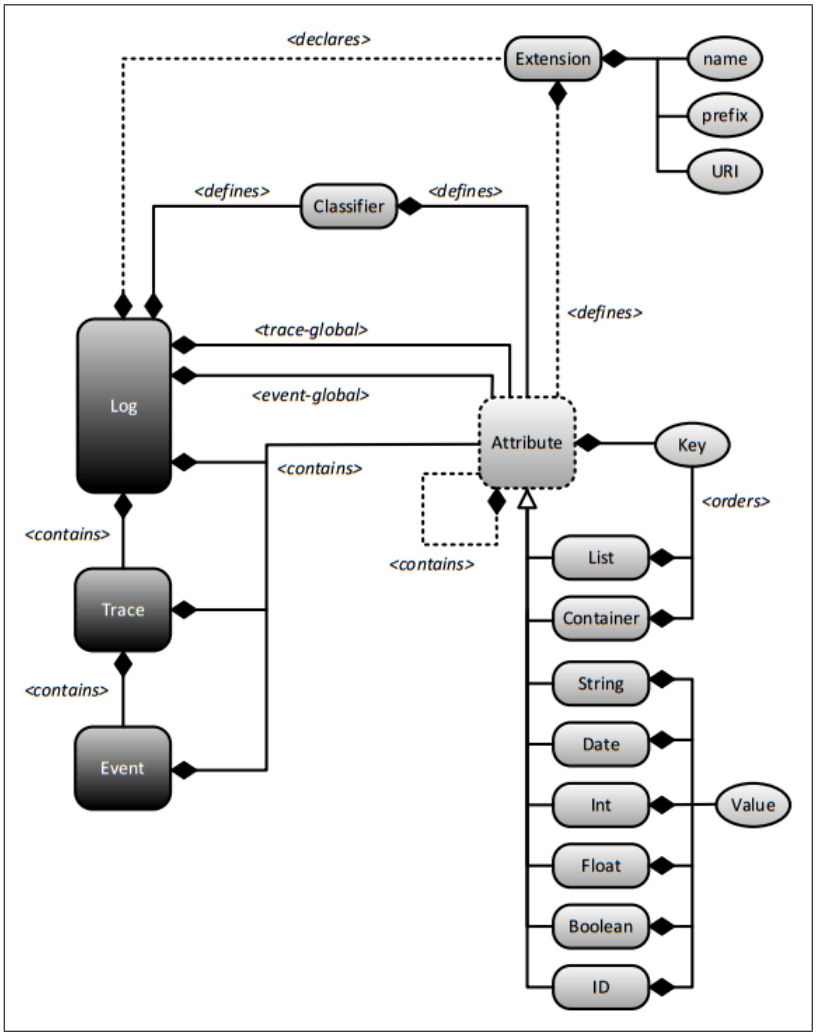


FIGURE 1.5. UML diagram of XES classes[4].

Big companies have *Enterprise Resource Planning (ERP)*, *Customer Relationship Management (CRM)* or other information systems that are able to store this information. This kind of software is called *Process-Aware Information Systems (PAIS)*.

1.2.2. Process models

Process models are used in every company for a variety of purposes. Their main goal is to decide the order in which activities must be executed. According to Aalst (2011b), some of the most important uses for models are listed as follows.

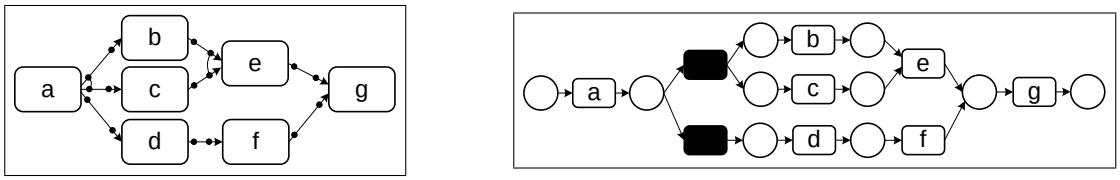
[4]Extracted from XES website (http://www.xes-standard.org/_media/xes/xesstandarddefinition-2.0.pdf).

- **Insight**: while making a model, the modeler is triggered to view the process from various angles.
- **Discussion**: the stakeholders use models to structure discussions.
- **Documentation**: processes are documented for instructing people or certification purposes (e.g. ISO 9000 quality management).
- **Verification**: process models are analyzed to find errors in systems or procedures (e.g., potential deadlocks).
- **Performance analysis**: techniques like simulation can be used to understand the factors influencing response times, service levels, etc.
- **Animation**: models enable end users to simulate different scenarios and thus provide feedback to the designer.
- **Specification**: models can be used to describe a PAIS before it is implemented and can hence serve as a contract between the developer and the end user.
- **Configuration**: models can be used to configure a system.

As we will show later, one of the process mining categories is *discovery*: mining for a process model given a certain event log. This model contains the order of execution of activities in all possible paths. Currently, several process model notations are being used. In this section, we will review two of them: *Petri nets* and *causal nets*.

A Petri net is a bipartite graph consisting of *places* and *transitions*. Each place can contain an unlimited number of *tokens*. The state of a Petri net is determined by the distribution of these tokens over places and is referred to as its *marking*. Transitions are used to signal the occurrence of an activity. However, a transition only can be triggered if there is at least one token on each of its input places. In Figure 1.6 (b), transitions are symbolized by squares and circles denote places.

Causal nets, shown in Figure 1.6 (a), are a graph where nodes represent activities and edges represent the relation between them. In this notation, process flow is identified by *bindings*, which are denoted by sets of dots over the edges.



(a) Example causal net. (b) Example Petri net.

FIGURE 1.6. Different representations of the same process. Black transitions are called invisible or silent transitions, as its occurrence is irrelevant in the process.

At present, the majority of process mining theory is formulated using Petri nets. Its simplicity makes them able to generate several guarantees that are desirable in a process model. However, this simplicity works against them when expressing more complex behavior (e.g. existence of *invisible/silent* activities).

Many other notations have been developed during the last decade, including EPCs, BPMN, YAWL and causal nets, also known as *c-nets*. C-nets use declarative semantics instead of a local firing rule (i.e. tokens on Petri nets). This way a larger fraction of models is considered to be correct, as the behaviour of a c-net is restricted to valid sequences. A detailed evaluation on c-nets as representational bias is presented in (Aalst,2011a). This compact representation is what makes c-nets more suitable than other languages in the context of process mining.

1.2.3. Discovery, Conformance and Enhancement

Process mining can be divided into three main categories: *process discovery*, *conformance checking* and *process enhancement*. While process discovery tries to generate a model from an event log, conformance checking highlights differences between reality (AS IS) and a theoretical model (TO BE). Finally, process enhancement tries to either include additional information to a process model (*extend*) or to provide corrections to a model (*repair*). Each category and its inputs and outputs are listed on Figure 1.7.

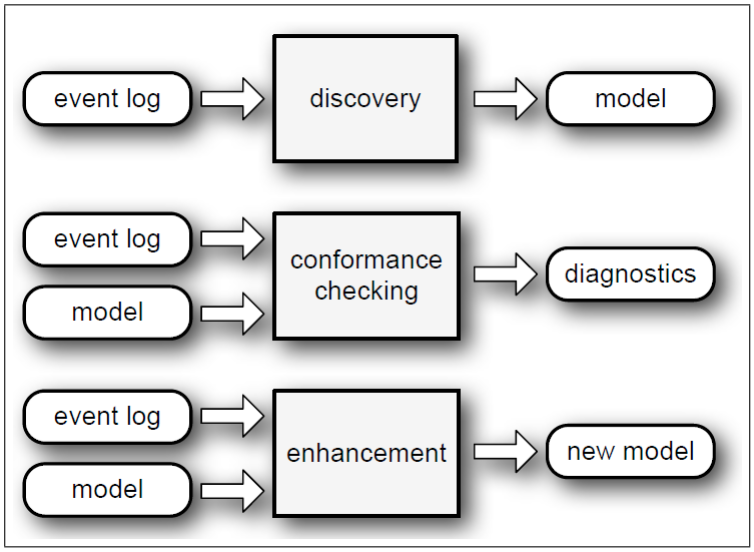


FIGURE 1.7. Process mining categories[5].

1.2.4. Process mining perspectives

Both discovery and conformance checking techniques generally aim to retrieve or check the order of activities in a model, which is known as the *control-flow perspective*. However, these categories are not limited to control-flow, as we could also discover a social network or check the validity of a organizational graph. All perspectives in process mining are listed as follows.

- **Control-flow perspective** focuses on the order of activities, providing insights of all possible paths in a process.
- **Organizational perspective** focuses on information about resources in the log and how they are related (e.g. groups of people working together).
- **Case perspective** focuses on properties of cases. For example, it may be interesting to predict the outcome of a process based on the cost of some key activity.
- **Time perspective** focuses in the timing and frequency of events, discovering bottlenecks, measuring service levels and predicting remaining process time of running cases.

[5]Extracted from (Aalst, Adriansyah, Medeiros, et al.,2011).

Note that the different perspectives are partially overlapping and non-exhaustive. However, they provide a good characterization of the aspects that process mining aims to analyze.

1.3. Challenges of current algorithms

Process discovery has a variety of algorithms, and the selection of the most appropriate one usually depends on many log factors, such as the number of activities, number of traces and log ”noise”, among others. Moreover, each algorithm generates a model in a different language. Most of them provide Petri nets, which is the most common form of representing a process. However, many other representations are also being used, in particular causal nets because of their simplified semantics. Therefore, it is well-accepted that there is no silver bullet algorithm in the field of process discovery.

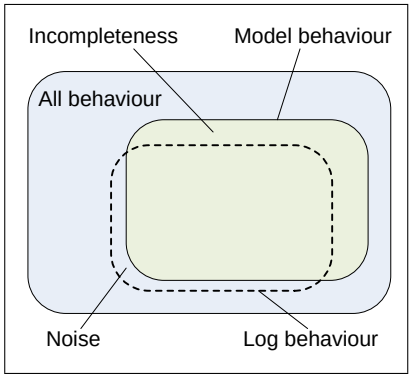


FIGURE 1.8. Theory/reality gap in process mining.

Typically, state-of-the-art process discovery techniques relies just on log information in order to generate a process model. However, event logs contain just a sample of all possible behaviour and are not exempt from errors. If a sequence of activities (later defined as a *trace*) is not present on an event log, that does not mean that sequence is not feasible. Figure 1.8 represents the gap between model and log. In an ideal world, the log behaviour (i.e. real behaviour) should match the model behaviour (i.e. theoretical behaviour), however this is not the case. Even if noise is not significant, it is always present because of inherent system flaws.

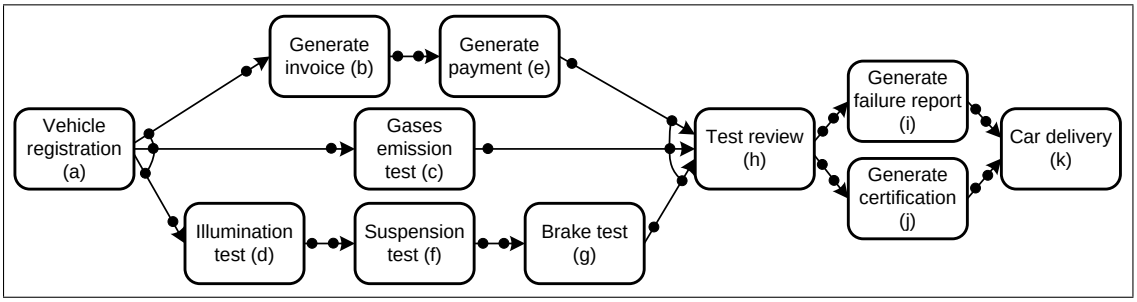


FIGURE 1.9. Process model for car technical reviews.

Situations where not all allowed behaviour is present on an event log are not infrequent (i.e. incompleteness). One of the most common cases is when two activities (d and f) can be performed concurrently but one of them requires significant less amount of time (d). Generally, a user will always perform short activity first, logging always d before f. For instance, Figure 1.9 represents a car technical review process. In this model, the illumination test is always done before the suspension test even though they can be done concurrently.

State-of-the-art process repair techniques also relies on an event log to perform its corrections, and as this information suffers from incompleteness, all behaviour cannot be amended. Some repair techniques require a reference model to work, but this requires to build an entire model to perform the most minimal correction. Therefore, most process repair techniques are unable to perform amendments such as required on Figure 1.9, where the illumination test (d) can be performed concurrently with the suspension test (f).

1.4. A Feedback-based Framework for Process Enhancement using Causal nets

Even though some process repair techniques are able to perform the correction needed in the previous example, they require to build a reference model, and this is a rather tedious task. In order to perform corrections in a more straightforward way, this work proposes an iterative feedback-based approach to process enhancement. Our framework provides a set of operations between activities that modify a process model in order to adjust to the user feedback. These feedback operations are *causality* ($\rightarrow$), *parallelism* ($\|$) and *indifference*

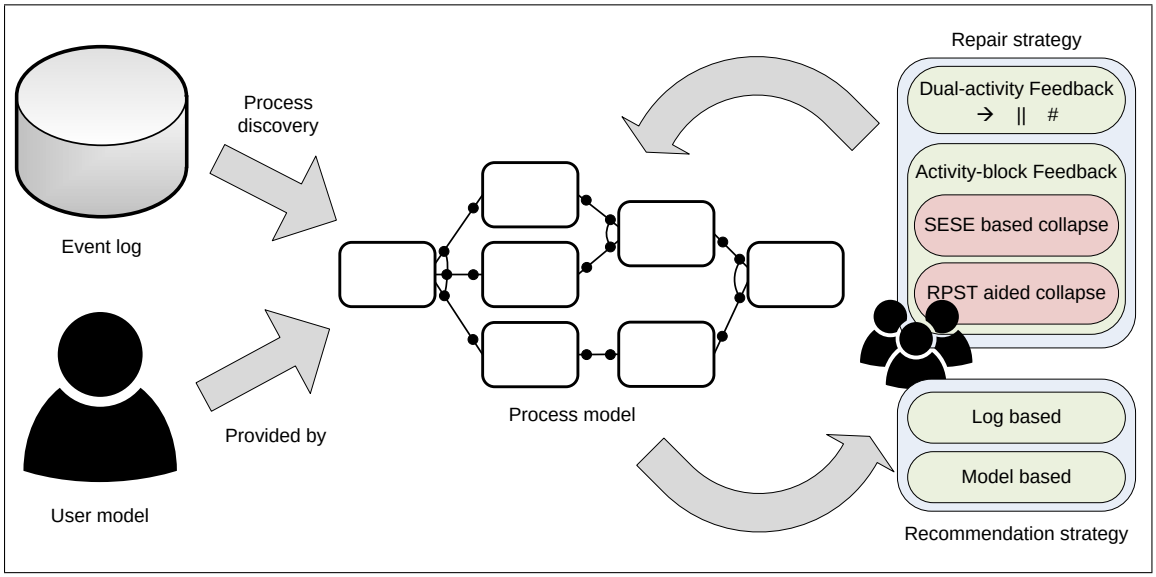


FIGURE 1.10. Iterative Feedback Enhancement.

(#). Our idea is to use process users, analysts, managers or others resources involved in a process as a new data source for process repair.

As Figure 1.10 shows, this framework provides a simple set of operations that will allow users to modify an existent model. Feedback relationships were based on the α-miner algorithm (Aalst, Weijters, & Maruster,2004), where causality and concurrency relations were defined. Please note that those definitions were made over Petri nets, so we will need to redefine them over c-nets. In order to simplify process models and extend the scope of our framework we also propose a *SESE* (Single Entry Single Exit) based node collapsing using a hierarchical tree called *RPST* (Refined Process Structure Tree). This collapsing technique allows to apply feedback relationships over group of activities. Finally, the framework also provides feedback recommendations that highlight specific sections of the process model. These identified sections might contain problems that are reported to the user, who can perform corrections using feedback operations.

1.5. Thesis Outline

This work is structured as follows. Section 2 formalizes basic notions on c-nets, SESE and RPST. Sections 3 and 4 explains how to deal with dual-activity and activity-block

feedback. Section 5 presents some strategies for automatic feedback recommendation. Section 6 shows our framework implementation using the ProM framework along with experimental evaluation. Finally, section 7 reviews all related work in this field and section 8 concludes this document.

2. PRELIMINARIES

In this section we will introduce some mathematical preliminaries along with model definitions.

2.1. Multisets, traces and logs

A multiset (or a bag) is a set in which elements of a set can appear more than once. Formally, a multiset is a tuple (S, c) where S is a set and $c : S \to \mathbb{N}$ is a cardinal function that assigns a positive integer to each element in the set. We denote $\mathbb{B}(X)$ as the set of all multisets that can be created using elements of set X. Generally, multisets are written used square brackets. For instance $[a^5, b^2, c]$ is a multiset where a appears five times, b two times and c one time.

A trace is a sequence of activities. Given a finite sequence $\rho = \langle a_1, a_2, a_3, ..., a_n \rangle$, its length is denoted by $|\rho| = n$ and the element at position i (i.e. a_i) is denoted as ρ_i. For example, $\langle a, b, c, e, g \rangle$ is a trace. Note that sequences are written using angle brackets.

An event log is simply a multiset of traces. For instance, $[\langle a, b, c, e, g \rangle^3, \langle a, c, b, e, g \rangle^4, \langle a, d, f, g \rangle^2]$ is an event log.

2.2. Process models and causal nets

In this section we will define c-nets, which are the main model used along this paper. Causal nets are originally defined in (Aalst, Adriansyah, & Dongen,2011).

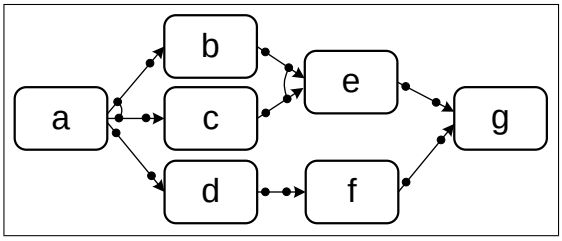


FIGURE 2.1. Causal net example.

Definition 2.1. *Causal nets*

A c-net is a graph where vertexes represent activities and edges represent causality between them. Formally, a c-net is a tuple C = (A, a_i, a_o, D, I, O) where:

- *A is a set of activities.*
- *$a_i \in A$ is the initial activity.*
- *$a_o \in A$ is the final activity.*
- *$D \subseteq A \times A$ is the dependency relation.*
- *$AS = \{X \subseteq \mathcal{P}(A) \mid X = \{\emptyset\} \vee \emptyset \notin X\}$ is the set of all possible bindings*[1].
- *$I : A \to AS$ defines the input set of each activity.*
- *$O : A \to AS$ defines the output set of each activity.*

where

- $D = \{(a_1, a_2) \in A \times A \mid a_1 \in \bigcup_{a \in I(a_2)} a\}$
- $D = \{(a_1, a_2) \in A \times A \mid a_2 \in \bigcup_{a \in O(a_1)} a\}$
- $\{a_i\} = \{a \in A \mid I(a) = \{\emptyset\}\}$
- $\{a_o\} = \{a \in A \mid O(a) = \{\emptyset\}\}$

Figure 2.1 represents a causal net. Each binding on output edges represents a possible outcome flow, whereas each binding on input edges represents start conditions. Formally, its components are defined as follows:

- $A = \{a, b, c, d, e, f, g\}$
- $a_i = a$, $a_o = g$
- $D = \{(a, b), (a, c), (a, d), (b, e), (c, e), (d, f), (e, g), (f, g)\}$
- I and O mapping functions are shown in the following table.

[1] $\mathcal{P}(A)$ denotes the powerset of A (i.e. the set of all subsets of A). Please note that either this set only contains the emptyset or the emptyset is not present on this set

TABLE 2.1. Input and Output functions for Figure 2.1.

Input (I)	*Activity*	*Output (O)*
$\{\emptyset\}$	a	$\{\{b, c\}, \{d\}\}$
$\{\{a\}\}$	b	$\{\{e\}\}$
$\{\{a\}\}$	c	$\{\{e\}\}$
$\{\{a\}\}$	d	$\{\{f\}\}$
$\{\{b, c\}\}$	e	$\{\{g\}\}$
$\{\{d\}\}$	f	$\{\{g\}\}$
$\{\{e\}, \{f\}\}$	g	$\{\emptyset\}$

As Figure 2.1 shows, process flow on causal nets is defined by its bindings. An activity binding is a tuple (a, as^I, as^O) denoting the occurrence of activity a with input binding as as^I and output binding as^O. For instance, (e, $\{b, c\}$, $\{g\}$) denotes the occurrence of activity e in Figure 2.1 while being preceded by b and c, and succeeded by g.

Definition 2.2. *Binding*

Let C = (A, a_i, a_o, D, I, O) be a c-net. $B = \{(a, as^I, as^O) \in A \times \mathcal{P}(A) \times \mathcal{P}(A) \mid as^I \in I(a) \wedge as^O \in O(a)\}$ *is the set of activity bindings. A binding sequence* σ *is a sequence of activity bindings, i.e.,* $\sigma \in B^*$.

B^* denotes the set of all binding sequence over B, including the empty sequence. A possible binding sequence for Figure 2.1 would be $\sigma_{ex} = \langle (a, \{\emptyset\}, \{b, c\}), (b, \{a\}, \{e\}), (c, \{a\}, \{e\}), (e, \{b, c\}, \{g\}), (g, \{e\}, \{\emptyset\}) \rangle$.

Function $\alpha : B^* \rightarrow A^*$ projects binding sequences onto activity sequences, i.e., the input and output bindings are abstracted and only the activity names are retained. For the previous example, $\alpha(\sigma_{ex}) = \langle a, b, c, e, g \rangle$.

A binding sequence is valid if a predecessor activity and successor activity always "match" their bindings. In other words, an occurrence of activity x with y in its output binding needs to be followed by an occurrence of activity y with x in its input binding. In order to formalize the notion of validity we need to first define the concept of state. States are multisets of pending obligations.

Definition 2.3. *State*

Let C = (A, a_i, a_o, D, I, O) be a c-net. Its state space $S = \mathbb{B}(A \times A)$ is composed of states that represent multisets of pending obligations. Function $\psi : B^* \to S$ is defined inductively as follows.

$$\psi(\langle\rangle) = []$$
$$\psi(\sigma + (a, as^I, as^O)) = (\psi(\sigma) \setminus (as^I \times \{a\})) \cup (\{a\} \times as^O)$$

for any binding sequence[2] *$\sigma + (a, as^I, as^O) \in B^*$. $\psi(\sigma)$ is the state after executing binding sequence σ.*

Initially there are no pending obligations. If activity binding $(a, \{\emptyset\}, \{b, c\})$ occurs in Figure 2.1, then $\psi(\langle\rangle + (a, \{\emptyset\}, \{b, c\})) = (\psi(\langle\rangle) \setminus (\emptyset \times \{a\})) \cup (\{a\} \times \{b, c\}) = ([] \setminus []) \cup [(a, b), (a, c)] = [(a, b), (a, c)]$. State $[(a, b), (a, c)]$ denotes the obligation to execute b and c using input bindings involving a. Input bindings remove pending obligations whereas output bindings create new obligations.

A valid binding sequence is a binding sequence that starts with the initial activity a_i, ends with the final activity a_o, only removes obligations that are pending, and ends without any pending obligations. This conditions are formalized in the following definition.

[2]For the sake of clarity, + operator is intended to append a binding to the end of a binding sequence.

Definition 2.4. *Valid sequence*

Let C = (A, a_i, a_o, D, I, O) be a c-net and $\sigma = \langle (a_1, as_1^I, as_1^O), (a_2, as_2^I, as_2^O), ..., (a_n, as_n^I, as_n^O) \rangle \in B^*$ *a binding sequence. σ is a valid binding sequence of C iif:*

- $a_1 = a_i, \quad a_n = a_o$
- $a_k \in A \setminus \{a_i, a_o\} \quad \forall k \in \mathbb{N} \mid 1 < k < n$
- $\psi(\sigma) = []$
- $(as_k^I \times \{a_k\}) \leq \psi(\sigma_{k-1}) \quad \forall k \in \mathbb{N} \mid 1 \leq k \leq n$

where $\sigma_k = \langle (a_1, as_1^I, as_1^O), ..., (a_k, as_k^I, as_k^O) \rangle$.

$V(C)$ is the set of all valid sequences of C.

For instance, binding sequence σ_{ex} defined as $\langle (a, \{\emptyset\}, \{b, c\}), (b, \{a\}, \{e\}), (c, \{a\}, \{e\}), (e, \{b, c\}, \{g\}), (g, \{e\}, \{\emptyset\}) \rangle$ is a valid sequence.

$$\psi(\langle\rangle) = []$$

$$\psi(\langle (a, \{\emptyset\}, \{b, c\}) \rangle) = [(a, b), (a, c)]$$

$$\psi(\langle (a, \{\emptyset\}, \{b, c\}), (b, \{a\}, \{e\}) \rangle) = [(a, c), (b, e)]$$

$$\psi(\langle (a, \{\emptyset\}, \{b, c\}), (b, \{a\}, \{e\}), (c, \{a\}, \{e\}) \rangle) = [(b, e), (c, e)]$$

$$\psi(\langle (a, \{\emptyset\}, \{b, c\}), (b, \{a\}, \{e\}), (c, \{a\}, \{e\}), (e, \{b, c\}, \{g\}) \rangle) = [(e, g)]$$

$$\psi(\langle (a, \{\emptyset\}, \{b, c\}), (b, \{a\}, \{e\}), (c, \{a\}, \{e\}), (e, \{b, c\}, \{g\}), (g, \{e\}, \{\emptyset\}) \rangle) = []$$

Sequence σ_{ex} indeed starts with initial activity $a_i = a$, ends with final activity $a_o = g$, only removes obligations that are pending and ends without any pending obligations since $\psi(\sigma_{ex}) = []$.

The concept of *soundness* has been defined for Petri nets and other process model notations. A process model is *sound* if it is free of deadlocks, livelocks and other anomalies.

Since the semantics of causal nets only consider valid sequences (i.e. invalid sequences are not considered part of the behavior), we only need to check that there are valid sequences and that all parts of the c-net can be activated by such valid sequences.

Definition 2.5. *Soundness*

A c-net C = (A, a_i, a_o, D, I, O) is sound iif

- $\forall a \in A, as^I \in I(a) \exists \sigma \in V(C) \wedge as^O \subseteq A \mid (a, as^I, as^O) \in \sigma$
- $\forall a \in A, as^O \in O(a) \exists \sigma \in V(C) \wedge as^I \subseteq A \mid (a, as^I, as^O) \in \sigma$

In other words, every activity and its input and output sets have a valid sequence in which its bindings are used. Finally, we will say a trace *fits* a c-net if the activity sequence generated by some valid binding sequence is equal to the trace. Equivalently, we will say that the model *replays* this trace.

Definition 2.6. *Fitting trace*

Let C be a c-net and L a log with $t \in L$ as one particular trace. We will say t is a fitting trace iif:

$$\exists \sigma \in V(C) \mid \alpha(\sigma) = t$$

3. DUAL-ACTIVITY FEEDBACK ENHANCEMENT

This section focuses on defining feedback operations for *dual* activities. We refer as dual-activity enhancement when provided feedback is limited to two activities, and not to a group of them. For each operation, we provide a definition together with an algorithm to force its compliance.

3.1. Causality ($\rightarrow$)

Definition 3.1. *Causality*

Let x and y be activities from a c-net (i.e $x, y \in A$). We will say that y is caused by x ($x \rightarrow y$) when:

- $\exists as^O \in O(x) \mid Y \in as^O \wedge \exists as^I \in I(y) \mid X \in as^I$
- $\forall a \in A \setminus \{x, y\}, \forall as^O \in O(a) \quad X \in as^O \rightarrow Y \notin as^O$
- $\forall a \in A \setminus \{x, y\}, \forall as^I \in I(a) \quad Y \in as^I \rightarrow X \notin as^I$

In other words, the first condition verifies that the two activities are in a sequential order. The two other conditions ensure that no other activities have X and Y in their input binding or output binding simultaneously. This is necessary in order to prevent parallelism.

To force two non-causal activities from a c-net to become one caused by the other one, we propose the following strategy.

The *contains* operator is quite intuitive, as an activity input or output set will contain an activity if any of its sets contains that activity. Remember that both the input and output sets are sets of sets of activities. Formally, given a c-net C = (A, a_0, a_i, D, I, O) and activities $a, b \in A$, we will say that $O(a)$ *contains* b *iff* $\exists x \in O(a) \mid b \in x$.

Algorithm 1 Force causality ($x \rightarrow y$)

```
1:  procedure FORCECAUSALITY(ACTIVITY X, ACTIVITY Y, C-NET C)
2:      if O(x) !contains y then
3:          O(x) ← O(x) ∪ {{y}}
4:      if I(y) !contains x then
5:          I(y) ← I(y) ∪ {{x}}
6:      for all a ∈ A \ {x, y} do
7:          for all as^I ∈ I(a) do
8:              if x ∈ as^I and y ∈ as^I then
9:                  as^I ← as^I \ {x}
10:         for all as^O ∈ O(a) do
11:             if x ∈ as^O and y ∈ as^O then
12:                 as^O ← as^O \ {y}
13:     return C = (A, a_i, a_o, D, I, O)
```

3.2. Parallelism ($\|$)

Definition 3.2. *Parallelism*

Let x and y be activities from a c-net (i.e $x, y \in A$). We will say that x and y are parallel ($x \parallel y$) when:

- $\forall a \in A \setminus \{x, y\}, \forall as^O \in O(a) \quad x \in as^O \leftrightarrow y \in as^O$
- $\forall a \in A \setminus \{x, y\}, \forall as^I \in I(a) \quad y \in as^I \leftrightarrow x \in as^I$
- $\forall as^O \in O(x) \quad y \notin as^O \qquad \forall as^O \in O(y) \quad x \notin as^O$
- $\forall as^I \in I(x) \quad y \notin as^I \qquad \forall as^O \in I(y) \quad x \notin as^I$

The first two conditions verifies that if one activity is present on an input or output set, the other activity has to be present as well. (i.e. input and output sets are equals). The remaining conditions ensure that one activity is not present on the other activity input or output sets. This is necessary in order to prevent causality.

To force parallelism on two non-parallel activities from a c-net, we propose the following algorithm.

Algorithm 2 Force parallelism ($x \parallel y$)

procedure FORCEPARALLELISM(ACTIVITY X, ACTIVITY Y, C-NET C)
  for all $a \in A \setminus \{x, y\}$ **do**
    for all $as^I \in I(a)$ **do**
      if $x \in as^I$ and $y \notin as^I$ **then**
        $as^I \leftarrow as^I \cup \{y\}$
      else if $x \notin as^I$ and $y \in as^I$ **then**
        $as^I \leftarrow as^I \cup \{x\}$
    for all $as^O \in O(a)$ **do**
      if $x \in as^O$ and $y \notin as^O$ **then**
        $as^O \leftarrow as^O \cup \{y\}$
      else if $x \notin as^O$ and $y \in as^O$ **then**
        $as^O \leftarrow as^O \cup \{x\}$
  $I_{xy} \leftarrow I(x) \cup I(y)$
  $O_{xy} \leftarrow O(x) \cup O(y)$
  for all $as^I \in I_{xy}$ **do**
    if $x \in as^I$ **then**
      $as^I \leftarrow as^I \setminus \{x\}$
    else if $y \in as^I$ **then**
      $as^I \leftarrow as^I \setminus \{y\}$
  for all $as^O \in O_{xy}$ **do**
    if $x \in as^O$ **then**
      $as^O \leftarrow as^O \setminus \{x\}$
    else if $y \in as^O$ **then**
      $as^O \leftarrow as^O \setminus \{y\}$
  $I(x), I(y) \leftarrow I_{xy}$
  $O(x), O(y) \leftarrow O_{xy}$
  return $C = (A, a_i, a_o, D, I, O)$

We can now apply parallelism operator to our process in Figure 1.9, assuming that the illumination test (d) can be performed concurrently with the suspension test (f). Forcing $d \parallel f$, results in Figure 3.1.

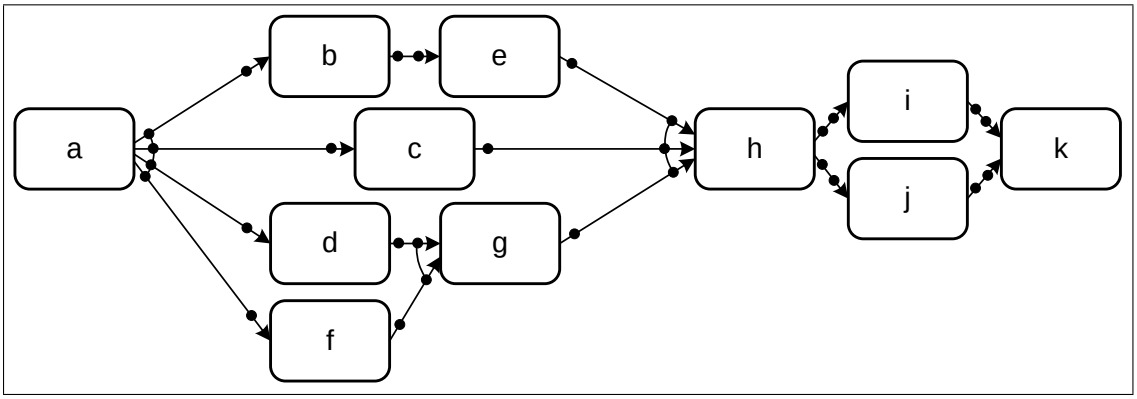


FIGURE 3.1. Enhanced c-net with a dual-activity operation.

3.3. Indifference (#)

Definition 3.3. *Indifference*

Let x and y be activities from a c-net (i.e $x, y \in A$). We will say that x and y are indifferent ($x\#y$) when:

$$x\#y \leftrightarrow \neg(x \rightarrow y) \wedge \neg(y \rightarrow x) \wedge \neg(x \parallel y)$$

To force indifference on two causal or parallel activities from a c-net, we propose the following strategy, which basically suppresses each activity from the other activity input and output set.

Algorithm 3 Force indifference ($x \# y$)

procedure FORCEINDIFFERENCE(ACTIVITY X, ACTIVITY Y, C-NET C)
  for all $as^I \in I(x)$ **do**
    if $y \in as^I$ **then**
      $as^I \leftarrow as^I \setminus \{y\}$
  for all $as^I \in I(y)$ **do**
    if $x \in as^I$ **then**
      $as^I \leftarrow as^I \setminus \{x\}$
  for all $as^O \in O(x)$ **do**
    if $y \in as^O$ **then**
      $as^O \leftarrow as^O \setminus \{y\}$
  for all $as^O \in O(y)$ **do**
    if $x \in as^O$ **then**
      $as^O \leftarrow as^O \setminus \{x\}$
  return $C = (A, a_i, a_o, D, I, O)$

As stated previously, please note that operations defined above work locally (hence called dual-activity enhancement). In order to expand the scope of this framework, we will use the concepts of SESE and RPST as a way to apply dual-activity operations over sets of activities, and not just a pair of them. In other words, we will be able to say $d \parallel \{f, g\}$ if f and g have some special properties. This kind of repair will be called *activity-block* enhancement.

4. ACTIVITY-BLOCK FEEDBACK ENHANCEMENT

This section defines the concept of SESE and RPST, along with our strategy to deal with feedback involving sets of activities. SESEs are subsets of edges over a graph that have a single entry and a single exit, while a RPST is a hierarchical tree that identifies all SESEs within a model.

4.1. Introduction to SESEs and RPST

To formalize the concepts of SESE and RPST, we need to first define a *multi-graph*.

Definition 4.1. *Multi-graph*

A multi-graph $G = (V, E, l)$ consists of two disjoint sets V and E of vertexes and edges, respectively. l is a mapping that assigns a pair of vertexes to each edge (i.e. $l : E \to V \times V$). Note that two vertexes can be connected by more than one edge (hence the term multi-graph). Whether $V \times V$ is an ordered pair or an unordered pair, we will call G directed or undirected multi-graph.

Please note that is it possible to generate a directed multi-graph $G = (V, E, l)$ from a c-net $C = (A, a_i, a_o, D, I, O)$ using $V = A$ and D to generate E and l. For instance, Figure 4.1 shows a directed multi-graph example, generated from the c-net in Figure 1.9.

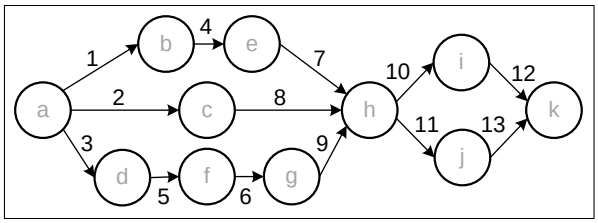


FIGURE 4.1. Directed multi-graph.

Given a multi-graph and a subset of its edges, we can distinguish each node depending on the graph structure. A node can either be *interior* if it is connected only to nodes in the subset or *boundary* if it is connected to nodes outside the subset.

Definition 4.2. *Interior, boundary, entry and exit nodes*

Let $G_S = (V_S, S)$ be a connected sub-graph of G formed by a set of edges $S \subseteq E$ and the vertexes $V_S = Ind(S)$[1] induced by S.

A node $x \in V_S$ is interior with respect to G_S iff it is connected only to nodes in V_S;otherwise x is a boundary node of G_S. A boundary node y of G_S is an entry of G_S iff no incoming edge of y belongs to S or if all outgoing edges of y belong to S. A boundary node y of G_S is an exit of G_S iff no outgoing edge of y belongs to S or if all incoming edges of y belong to S.

A SESE is a particular subset of edges, which has only one input and one output. This kind of subset is particularly useful in some contexts, as it allows to group or divide several regions of a graph.

Definition 4.3. *SESE*

$S \subseteq E$ is a SESE (Single-Exit-Single-Entry) of graph $G = (V, E)$ iff G_S has exactly two boundary nodes: one entry and one exit. A SESE is trivial if it is composed of a single edge. S is a canonical SESE of G if it does not partially overlap with any other SESE of G, i.e., given any other SESE S' of G, they are nested ($S \subseteq S'$ or $S' \subseteq S$) or they are disjoint ($S \cap S' = \emptyset$)

An RPST is a hierarchical tree of canonical SESEs. SESE decomposition is a well-studied problem. Work in (Polyvyanyy, Vanhatalo, & Völzer,2010) provides the latest algorithm to create an RPST for a given graph.

Definition 4.4. *RPST*

The Refined Process Structured Tree (RPST) of G is the tree composed by the set of all its canonical SESEs, such that, the parent of a canonical SESE S is the smallest canonical SESE that contains S. The root of the tree is the entire graph, and the leaves are the trivial SESEs.

[1] $Ind(R) = \bigcup_{(a,b)\in K}\{a, b\}$ where $K = \bigcup_{v\in R} l(v)$

For instance, Figure 4.2 shows all SESEs from Figure 4.1. The final RPST is presented in Figure 4.3.

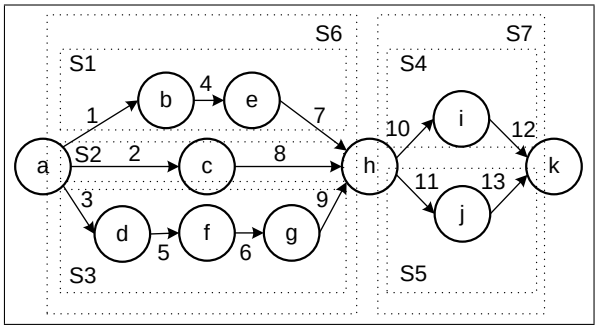


FIGURE 4.2. SESE decomposition.

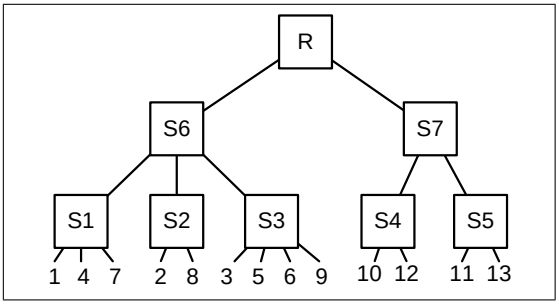


FIGURE 4.3. RPST.

Please note that RPST is computed over edges (not over nodes), which constitutes a challenge for this approach, as user feedback is defined over activities. In other words, we need to be able to group activities and not edges.

4.2. Node-extended RPST

To tackle this issue we need to extend RPST to nodes. While maintaining RPST structure, we will replace edges as leaves of the tree. An activity will be present on a node of the tree if all of its edges are present on the same node (i.e. input and output edges).

Definition 4.5. *Node-extended RPST*

Given a multi-graph $G = (V, E, l)$ and its RPST, an activity $x \in V$ is present on the node R of a node-extended RPST iif:

$$E_x \subseteq R \quad \textit{where} \quad E_x = \{e \in E \mid l(e) = (x, z) \vee l(e) = (z, x)\}$$

For instance, if we extend the RPST on Figure 4.3 we would obtain the results presented in Figure 4.4.

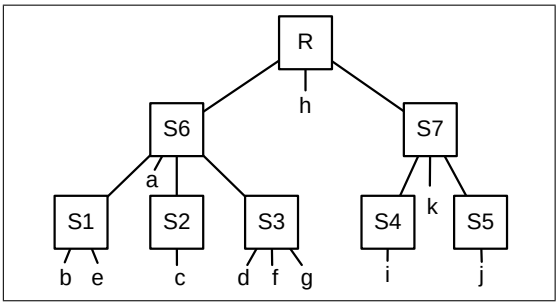


FIGURE 4.4. Node-extended RPST.

4.3. SESE-based Node Collapsing

To avoid locality problems with algorithms presented in section 3 we propose a *SESE-based node collapsing* approach. Our main idea is to use SESEs as a way to generate sub-processes and then apply our feedback algorithms in resulting model.

Node collapsing can be done by two methods: user identification or RPST-aided. In the first one, a node can be collapsed if a user can visually identify a SESE. If this is not possible, we will use our node-extended RPST to find one. At this point, SESEs are only defined as a group of edges of a graph, so we need to extend the concept of SESEs to c-nets.

Definition 4.6. *C-net Extended SESE*

Let $C = (A, a_i, a_o, D, I, O)$ be a c-net and $A' \subseteq A$. A' is a SESE iff

- $\exists! s_i \in A' \mid O(s_i) \subseteq \mathcal{P}(A') \wedge I(s_i) \not\subset \mathcal{P}(A')$
- $\exists! s_o \in A' \mid I(s_o) \subseteq \mathcal{P}(A') \wedge O(s_o) \not\subset \mathcal{P}(A')$
- $I(a) \subseteq P(A') \wedge O(a) \subseteq \mathcal{P}(A') \qquad \forall a \in A' \setminus \{a_i, a_o\}$

where $\mathcal{P}(X)$ denotes the powerset of X.

Activities s_i and s_o are called *input* and *output* activity respectively.

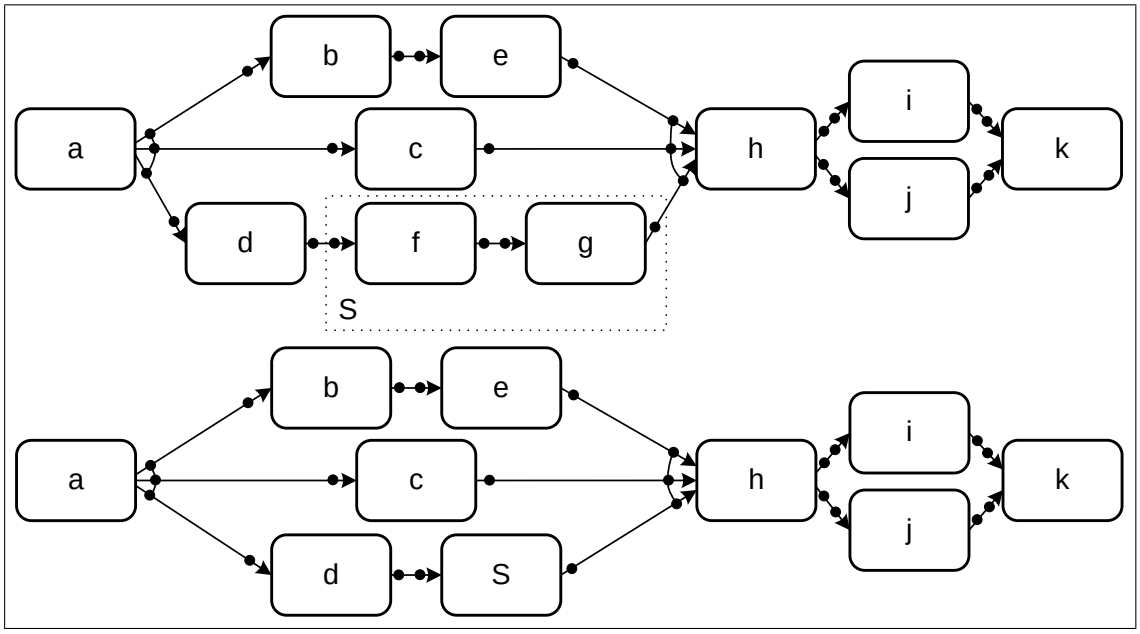


FIGURE 4.5. Collapsed SESEs.

In other words, A' is a SESE if all its activities are connected to other activities in A', except for two cases, the *input* and *output* activities, which can have inputs and outputs outside A', respectively. Algorithm 4 shows a way to check whether a subset of activities is a SESE.

Algorithm 4 Check SESE

```
1:  procedure CHECKSESE(C-NET C, ACTIVITY[] A')
2:      input, output ← null
3:      A'' ← A \ A'
4:      for all a ∈ A' do
5:          for all b ∈ A'' do
6:              if I(a) contains b then
7:                  if input is not null then return false
8:                  else
9:                      input ← a
10:             if O(a) contains b then
11:                 if output is not null then return false
12:                 else
13:                     output ← a
14:     return true
```

Once algorithm 4 is used to check for a SESE, we are able to generate a new c-net, replacing all activities within A' for a new macro-activity. We can then apply causality, parallelism and indifference operators on this resulting c-net. An example of this transformation is shown on Figure 4.5. Note that the new macro activity (S) input set is equal to SESE input activity (f) input set. Same thing happens with the output set of S and g.

Formally, let $C = (A, a_i, a_o, D, I, O)$ be a c-net and $A' \subseteq A$ a SESE with s_i and s_o as input and output activities. A SESE collapsed c-net can be processed using algorithm 5.

Algorithm 5 Collapse SESE

procedure COLLAPSESESE(C-NET C, ACTIVITY[] A')
  $A'' \leftarrow A \setminus A'$
  for all $a \in A''$ **do**
    for all $as^O \in O(a)$ **do**
      if $s_i \in as^O$ **then**
        $as^O \leftarrow (as^O \setminus \{s_i\}) \cup \{S\}$
    for all $as^I \in I(a)$ **do**
      if $s_o \in as^I$ **then**
        $as^I \leftarrow (as^I \setminus \{s_o\}) \cup \{S\}$
  $I(S) \leftarrow I(s_i)$
  $O(S) \leftarrow O(s_o)$
  if $a_i \in A'$ **then**
    $a_i \leftarrow S$
  if $a_o \in A'$ **then**
    $a_o \leftarrow S$
  return $C_{A'} = (A'' \cup \{S\}, a_i, a_o, D, I, O)$

Back to our example, collapsing the suspension test (f) and brake test (g) and then applying $d \parallel \{f, g\}$, we would obtain the c-net presented in Figure 4.6.

Notice that information about the process is not lost after collapsing a group of activities, as this collapse can be restored after the operation is completed. In other words, we are using c-net extended SESEs only as a way to apply dual-activities operators over group of activities. C-net extended SESEs have an input and output activity that can replace the macro activity's input and output bindings when uncollapsing.

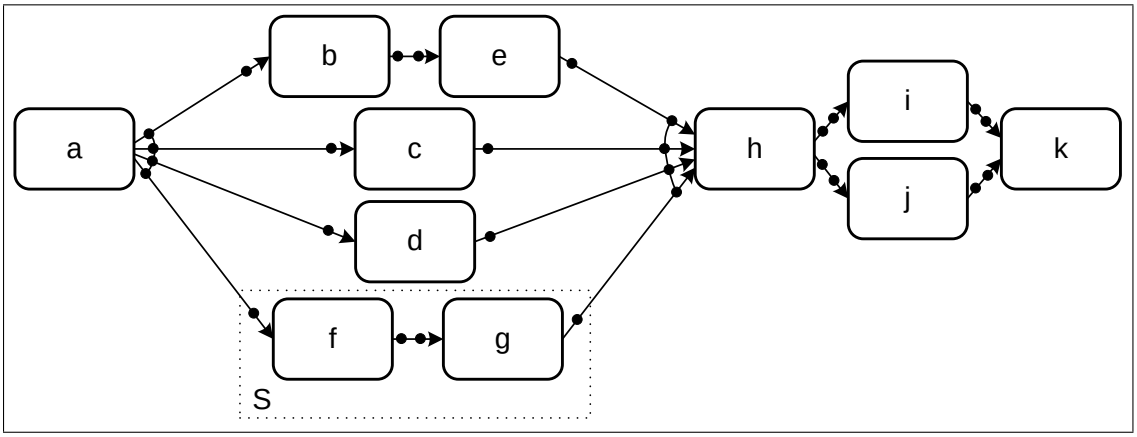


FIGURE 4.6. Enhanced c-net with a activity-block operation.

4.4. RPST-aided SESE Collapse

SESE-based node collapsing presented in section 4.3 is useful whenever a user is able to identify a c-net extended SESE. As in many cases this visual identification is not possible, we propose an *aided SESE collapse*, based on the extended RPST shown in section 4.2. This way, the algorithms can help a user dealing with a complex process model to find a c-net extended SESE that contains certain activity. Given two distinct activities of a c-net, our goal is to identify two non-overlapping SESEs, each one containing one of the activities. We can then proceed to collapse these SESEs using algorithm 5 and then apply dual-activity feedback operators over resulting collapsed activities. Our search strategy is presented in algorithm 6.

In this example, function $getParent$ returns the set of all activities contained in the parent node. For instance, in Figure 4.4 if R is the node-extended RPST then $getParent(f, R) = \{d, f, g\}$.

The main problem of this approach is that not every node from a node-extended RPST is a SESE as presented in definition 4.6. It is easily verifiable that node *S7* in Figure 4.4, composed by activities *i*, *j* and *k*, is not a c-net extended SESE (activities *i* and *j* have inputs from activity *h*, which is outside the set).

Algorithm 6 Aided SESE

```
1:  procedure AIDEDSESE(ACTIVITY X, ACTIVITY Y, NODE EXTENDED RPST R)
2:      S_x ← getParent(x, R)
3:      S_y ← getParent(y, R)
4:      while !checkSESE(S_x) do
5:          S_x ← getParent(S_x, R)
6:      while !checkSESE(S_y) do
7:          S_y ← getParent(S_y, R)
8:      if S_x ⊂ S_y then
9:          S_y ← {y}
10:     else if S_y ⊂ S_x then
11:         S_x ← {x}
12:     else if S_x = S_y then
13:         S_x ← {x}
14:         S_y ← {y}
15:     return S_x, S_y
```

In order to find a SESE in which an activity is contained, we need to first validate if the subset is indeed a c-net extended SESE. If it is not a c-net extended SESE, we can try its parent node. Worst case scenario would be to reach the root node, without finding a better c-net extended SESE. Even though this approach is not exhaustive, it is fast and provide at least one solution to our SESE-finding problem. Please note that if two activities share one RPST node, the algorithm is not able to find a SESE, as in many cases the solution may not be unique. For instance, activities *d* and *g* share node *S3*. Two possible SESEs containing these activities are $\{d, f\}$ and g or d and $\{f, g\}$. Since this problem has two solutions, the proposed algorithm return simply returns each activity as a SESE (i.e. $\{d\}$ and $\{g\}$).

The presented c-net extended SESE finding approach is not exhaustive, and this is due to our c-net extended RPST construction. As Figure 4.4 shows, if a user asks to find a c-net extended SESE for activity k, our current algorithm would return $\{a, b, c, d, e, f, g, h, i, j, k\}$, which is indeed a c-net extended SESE but not minimal. Subset $\{h, i, j, k\}$ is not found even though it is a valid c-net extended SESE. This is due because input and output activities are not present in an node-extended RPST until their incoming edges belong to that node.

5. STRATEGIES FOR FEEDBACK RECOMMENDATION

Direct feedback can be a good alternative when process experts have an idea of where the model can be enhanced. As this might not be the case, we propose some ideas to guide users. Our strategies for feedback recommendation aims to highlight specific subsets of a model where problems may occur and ask users to repair these subsets. Please note that these strategies work only as a warning sign, since it is up to the user to repair or disregard feedback recommendations.

5.1. Log-based Feedback Recommendation

Several strategies can be adopted in order to generate feedback recommendations. The first proposed strategy requires an event log together with the process model to work. Log-based recommendations are divided into two categories, one using the event log's fitting traces and another one using non-fitting traces.

5.1.1. Fitting Traces and Common Paths

In large process models, the repair process may be a time-consuming task. In order to focus users with limited time on more common process paths, we propose using fitting traces event log to identify most common traces (hence more common process paths). Our strategy is to highlight these specific paths to the user. The main idea behind this strategy is to focus just on significant sections of the process model.

5.1.2. Unfitting traces and Deviation Points

Another recommendation technique relies on *deviation points*. A deviation point is an activity in a process model in which the model cannot replay a trace. For this case, we use non-fitting traces present in the event log to check for deviations. This identification is not an easy task, since many deviation points may occur in the same trace.

In order to identify deviation point we need to introduce *alignments*. An alignment tries to "match" a trace and a process model. For instance, trace t defined as $t = \langle a, b, d, g, e, h, i, j, k \rangle$ has the following alignment with the process model on Figure 1.9.

log	a	b	d	$\gg$	g	e	$\gg$	h	i	j	k
model	a	b	d	f	g	e	c	h	i	$\gg$	k

The top row of each alignment corresponds to "moves in the log" and the bottom row correspond to "moves in the model". If a move in the model cannot be mimicked by a move in the log, then a "$\gg$", denoting a deviation point, appears in the top row. The symmetric situation (a move in the log that cannot be mimicked by a move in the model) can also happen and is denoted analogously. Please note that in a fitting trace there is no deviation points. For more details on alignments for conformance checking the reader is referred to (Bose & Aalst,2012;Aalst, Adriansyah, & Dongen,2012).

Since each of the "$\gg$" signs denote a deviation point, it is necessary to develop a highlight strategy for feedback recommendation. For this purposes, we propose to count the number of mismatch per activity, after aligning each trace in the event log. For example, using the event log $L = [\langle a, b, d, g, e, h, i, j, k \rangle^5, \langle a, b, e, c, d, g, h, i, k \rangle^2, \langle a, b, e, d, f, c, g, h, j, k \rangle]$ with the process model in Figure 1.9, results in the following alignments.

a	b	d	$\gg$	g	e	$\gg$	h	i	j	k
a	b	d	f	g	e	c	h	i	$\gg$	k

a	b	e	c	d	$\gg$	g	h	i	k
a	b	e	c	d	f	g	h	i	k

a	b	e	d	f	c	g	h	j	k
a	b	e	d	f	c	g	h	j	k

Finally, counting the number of deviation points for each trace results in Table 5.1. Please note that the last trace is a fitting trace, since no deviation points are found. Notice that only three alignments are presented but each aligned trace has multiple occurrences in the event log.

TABLE 5.1. Deviation point count.

Activity	*f*	*c*	*j*
Deviation points	*5+2=7*	*5*	*5*

In this case, the user would be reported about activity f. However, given the large number of activities that may suffer from deviation points in a real event log, a threshold based on a percentage of the number of activities (e.g. 5%) should be used to report only the most relevant mismatches.

5.1.3. Decomposition-based Alignments

Alignment techniques are extremely challenging from a computational point of view. Traces in the event log need to be mapped to paths in the model. A model may have many paths and the traces may have an arbitrary amount of deviating points. Although the algorithms have demonstrated to be of great value for managing small or medium-sized problem instances, they are often unable to handle problems of industrial size. In this case, we propose to use decomposition techniques like the ones presented in (Munoz-Gama, Carmona, & Aalst,2013a,2013b). These algorithms divide a process model into multiple components, based on SESEs using RPST, and then process alignments over these components. This divide-and-conquer approach has been found to be extremely useful on large process model contexts, reducing computation time in an order of magnitude. After processing alignments with this decomposition approach, we propose the feedback recommendation strategy presented in the previous section, based on counting deviation points.

5.2. Model-based Feedback Recommendation

Another recommendation strategy is based on model behaviour. Given an initial process model, or after each feedback iteration, we propose to search for deadlocks inside the c-net. Even though some amending algorithms have been proposed in order to fix these situations (Solé & Carmona,2013), our idea is to highlight these specific problems and alert

the user. Note that deadlocks can be present on the initial process model or generated by an erroneous feedback operation.

Model complexity is also a perspective that can be used for model-based feedback recommendation. These techniques intend to highlight most complex sections on a process model. This is done by counting each activity input and output bindings (i.e. the sum of the cardinality of the input and output function). This indicator reflects the complexity each activity in terms of the number of different possible flows or paths. If a certain activity has a high indicator value, several paths arrive to or are issued from this activity. A problem on those specific bindings may result in a much significant error. For instance, using Figure 1.9, the binding count results in Table 5.2.

TABLE 5.2. Binding count.

Activity	h	b	c	d	e	f	g	i	j	k	a
$\|I\| + \|O\|$	*3*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*1*

In this case, the user would be reported about activity h. However, given the large number of activities in a real event log, a threshold based on a percentage of the number of activities (e.g. 5%) should be used to report only the most relevant activities.

6. IMPLEMENTATION AND EXPERIMENTAL EVALUATION

6.1. ProM Implementation

Algorithms described in 3 and 4 have been implemented in a ProM plugin called *FeedbackRepair*, available on version 6.4 nightly builds. Some screenshots are shown in Figures 6.1 and 6.2.

As Figure 6.2 shows, all operators are in the bottom of the screen. Process users have to select an activity followed by an operator and another activity. Finally, pressing update will generate the resulting model.

This plugin takes as parameters an *heuristics net* and an event log. Heuristic nets are a ProM implementation of causal nets and can be easily generated by the Heuristic Miner or Flexible Heuristics Miner (Weijters & Ribeiro,2011) algorithms. In order to generate RPSTs, this plugin uses the Java library jBPT (Java Business Process Technologies). This library contains tools for process model analysis that support research on the design, execution, and evaluation of processes, including RPST calculation, which is described in Polyvyanyy et al. (2010).

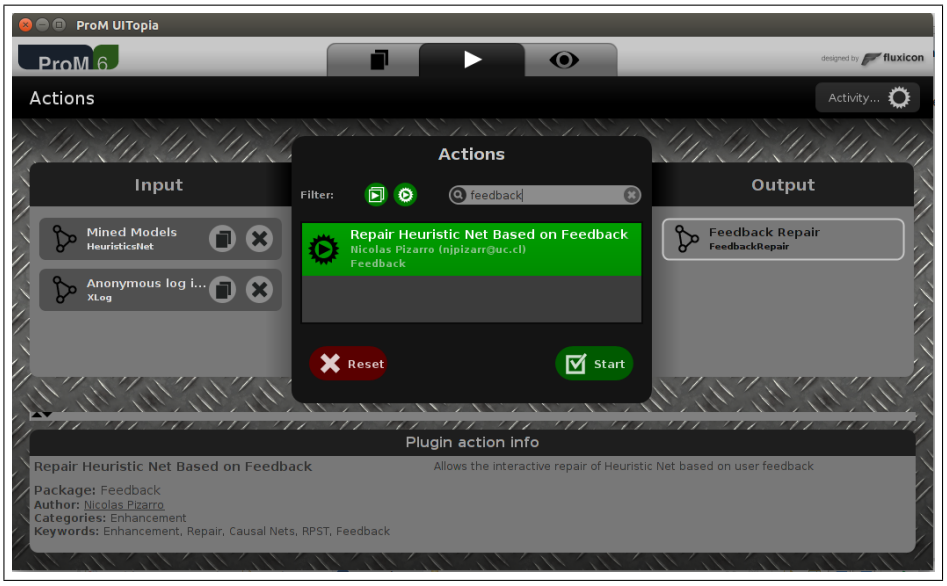


FIGURE 6.1. FeedbackRepair plugin in ProM.

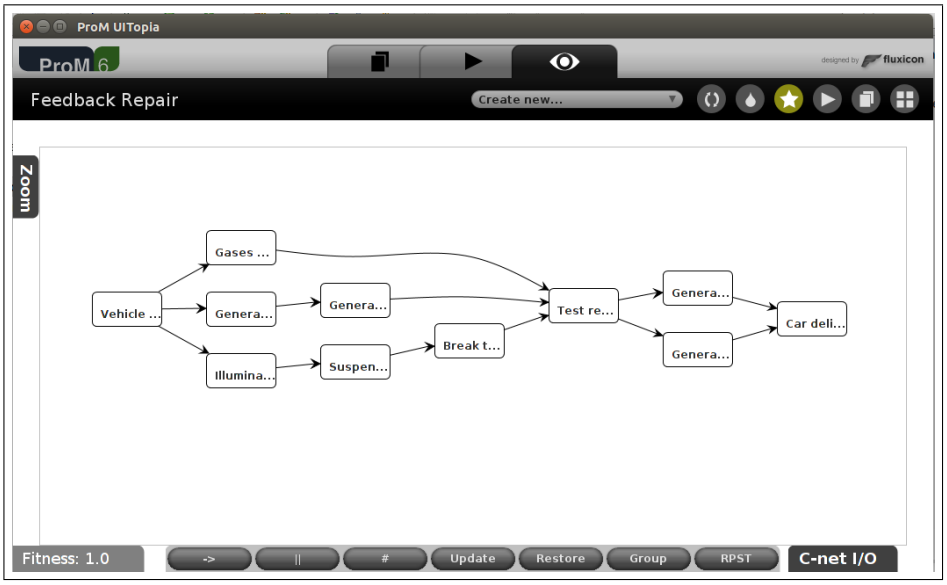


FIGURE 6.2. User interface of FeedbackRepair plugin in ProM.

6.2. Experimental Evaluation

This section focuses on testing the proposed algorithms using diverse approaches. First step is a performance evaluation, where algorithms were applied to a large real case process model. Then, we performed a user evaluation, where we tested whether generated models are in line with user's intuition.

6.2.1. Performance evaluation

For performance evaluation purposes, we took a c-net model of a bank transfer process, consisting in 113 activities and 150 edges (i.e. $|A| = 113$ and $|D| = 150$). This model was generated by applying the Flexible Heuristic Miner algorithm to the bank's event log[1]. For this process model, we have simulated 50 SESE collapse, 50 causality operations ($\rightarrow$), 50 parallelism operations ($\|$) and 50 indifference operations ($\#$). Results are shown on Table 6.1.

As Table 6.1 shows, most of computation time is used processing RPSTs and node-extended RPSTs. This calculation is needed at every step because each iteration modifies

[1]http://dx.doi.org/10.4121/uuid:c1d1fdbb-72df-470d-9315-d6f97e1d7c7c

TABLE 6.1. Time results for performance evaluation.

Operation	*Avg. time (s)*	
	C-net mod.	*RPST calc.*
Causality ($\rightarrow$)	0.12	3.2
Parallelism ($\|\|$)	0.11	3.6
Indifference ($\#$)	0.11	3.1
SESE collapse	0.15	3.4
Total	**0.11**	**3.5**

the existing model, and consequently its RPST. However, RPST computation (Polyvyanyy et al.,2010) is also quite fast, as this algorithm has linear complexity (i.e. $\mathcal{O}(n)$). This shows that even in large models, the defined algorithms are completed in a time that allows the user to provide several feedback operations in matter of seconds.

6.2.2. User evaluation

For user evaluation purposes, we performed an experiment that was divided in two steps. In the first one, we proposed a running example in which we asked users about their proposals to deal with log incompleteness on a specific small process model. The second step consisted on applying some specific operations over a large model using The FeedbackRepair plugin on ProM.

The experiment involved 12 subjects, all with a strong knowledge on business process modeling and process mining: 4 subjects had expertise on process mining research, while 8 had a more application focus on the matter.

In the first experiment, we presented the problem of process discovery/repair related to log incompleteness, without giving users any key about the purpose of our framework. We asked them to manually repair a process model given certain problem. Results can be seen on Table 6.2, where 71% of the generated models matched the output of the proposed algorithms.

We also asked users to identify the type of relationship a domain expert could give about a set of activities. The concept of parallelism and causality came immediately (85% of cases), but indifference was not brought up to discussion, possibly because it is easier

TABLE 6.2. Results for user evaluation.

Model	*Correct*	*Incorrect*	*Success rate*
m01	9	3	75%
m02	8	4	67%
Total	**17**	**7**	**71%**

to say how two activities are related and not how they are not. Interestingly, some users also suggested that grouping might be a useful user insight if provided by a process expert, since this would allow a macro point-of-view of the process.

This led to the question about how activities could be grouped. In this category, only a few of the most experienced researchers noticed that all groupings were SESEs. However the most remarkable fact, was that a couple of the other users suggested that this grouping could be done hierarchically, the same way as RPST work.

Regarding the second experiment, we presented our framework to users, together with a process model. We asked them to perform several operations over the model, in order to familiarize them with the Feedback Repair plugin. After all operations were completed and models were analyzed, we provided a set of statements about the application and asked users to rate their level of agreement with each of them. Approval was measured using a -2 (strongly disagree) to 2 (strongly agree) Likert scale. Results are shown in Figure 6.3.

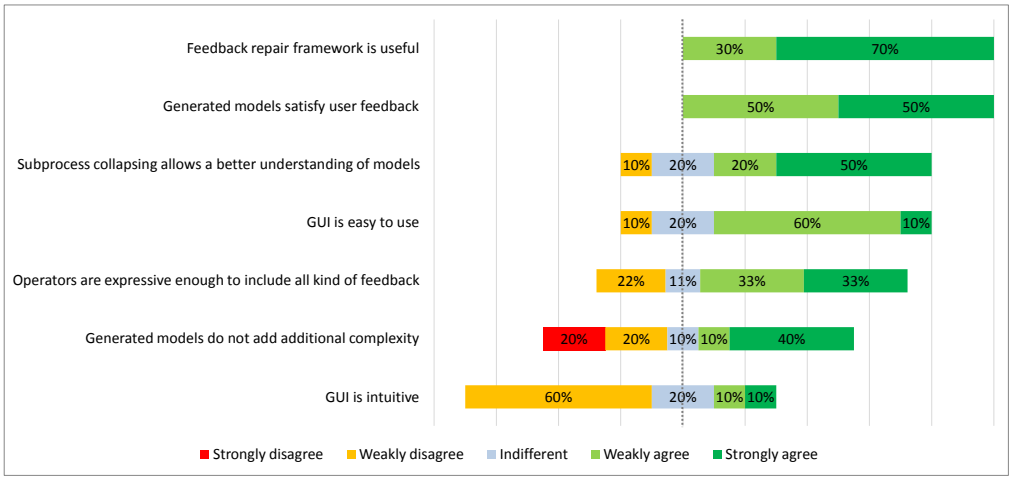


FIGURE 6.3. Users evaluation.

Results show that most users consider this framework to be useful (70% strongly agree), and that proposed operations allows to include all sorts of feedback (66% strongly or weak agree) on a specific model. Moreover, users agree that the generated models satisfy their feedback (50% strongly agree) and that sub-process collapsing was helpful to provide a better understanding of model processes. However, they remain neutral to the fact that the framework incorporated additional complexity to the model (50% disagree, 40% agree). Regarding the user interface, users opinion was that it was not intuitive enough (60% weakly disagree), but once that everything was explained, they stated that is was easy to use (60% weakly agree).

7. RELATED WORK

Process repair is being tackled from multiple perspectives. On one hand, several correction algorithms have been developed using event log information. This approach is useful when the process model is not generated using discovery techniques over the same log, since there is not new information being added. This makes these algorithms not able to deal with log incompleteness. For this perspective, several conformance checking techniques are being used. In (Fahland & Aalst,2015), the authors divide non-replayable traces into different logs and then modify the existing model, replacing each nonconformity section by its repaired sub-process. In the same perspective, other initiatives use an iterative process discovery approach. In (Kindler, Rubin, & Schäfer,2005), process discovery is made based on logs of Document Version Management Systems. Each iteration adds additional information, as more traces are now present on the logs. Result is produced by merging successive models. Work in (Sun, Li, Peng, & Sun,2007) proposes an incremental workflow mining algorithm based on ordering and independence relationships.

On the other hand, corrections are also performed based on a reference model. The main drawback of this approach is that a complete reference model is necessary even to perform the most minimal correction. In (Li, Reichert, & Wombacher,2009) authors propose a metric based on the number of differences between activities to compare model similarity. Algorithms in (Dongen, Desel, & Aalst,2012) try repairing a model aggregating ”runs”, which are mappings of Petri nets. A mix with previous perspective is presented in (Buijs, Rosa, Reijers, Dongen, & Aalst,2012), where the algorithm takes a log and a reference model to discover a model that represents the log but is also ”similar” to the reference model.

Some algorithms seeks to correct specific behaviour, for example deadlocks, livelocks or other process anomalies. This algorithms require just a process model to work since no additional behaviour is being added. For instance, work in (Gambini, Rosa, Migliorini, & Hofstede,2011) proposes a soundness approach in order to ensure deadlocks are not present on a process model. A similar idea is presented in (Lohmann,2008), where deadlocks

are fixed based on edit distance. In (Solé & Carmona,2013), authors propose an *SMT amendment* for an initial c-net. This is achieved through dividing the process into several components and applying first-order logic to find a feasible solution. The main focus of this approach is to improve model fitness, precision and simplicity. This work followed (Solé & Carmona,2012), where *SMT discovery* was proposed. Approach of (Fahland & Aalst,2011,2013) introduces a post-processing step after discovery aiming to simplify a model while balancing overfitting and underfitting. Work in (Bose & Aalst,2009) proposes to cluster process instances in order to deal with unstructured logs.

Finally, some new techniques perform a *log repair* instead of model repair. These techniques perform corrections on log traces before any process discovery algorithm is attempted. Some of this techniques allow to add user-provided knowledge in this pre-processing step. However, the main drawback of this approach is that user-provided information is given ”a priori”, since users have no idea of the generated model and has to provide all their information before process discovery. For instance, the work in (Rogge-Solti, Mans, Aalst, & Weske,2013a,2013b) propose a log repair. A similar approach is presented in (Leoni, Maggi, & Aalst,2015), where alignments are used to check conformance on declarative models and then perform log repair. In (Dumas & Garcia-Banuelos,2015), authors propose event structures as a common representation of process models and event logs, merging all knowledge into a single data-source.

It is important noticing that user feedback has already been proposed for BPM in the context of model matching (i.e. identification of correspondences between process models). Model matching is also a common use case, since it allows to search for a ”similar” model in a process model repository. In (Aalst,2013), this use case is listed as ”select model from collection”, and not in the ”repair model” category. In (Klinkmüller, Leopold, Weber, Mendling, & Ludwig,2014), authors provide a way of finding similarities between two process models using user feedback. Even though this approach does not fit the context of process repair, it uses a similar technique to debug matching algorithms, since it relies on SESEs as a way to match group of activities and RPST as a way to find suitable SESEs.

Note that even though the methodology presented in this work has a different approach to the perspectives recently presented, they are not exclusive. It is possible to use these techniques together. For instance, section 5 proposes a way to use deadlock detection after a user feedback operation in order to check for possible anomalies.

8. CONCLUSIONS AND FUTURE WORK

This work addresses for the first time the problem of repairing a process model without direct use of logs or reference models. Instead, we propose a framework in which users can provide their knowledge about the process using simple operators. We also provide some guidelines that can aid a less experienced user to focus on some specific section of the model.

Experimental evaluation shows that the proposed algorithms are computed in matter of seconds, even for large process models. Moreover, this framework has been found to be useful for most users and that the proposed operations allows to incorporate desired behaviour on the models.

Future work includes an exhaustive algorithm to retrieve all c-net extended SESEs as defined before. At this point we are suggesting a single c-net extended SESE, which works as a proof of concept but needs further investigation in order to maximize the expressiveness of this framework. We have to also work on a quantitative metric to measure resulting model fitness, since if measured against a log, resulting model will generally underperform fitness and precision-wise. Finally, further work is required on the implemented UI. Other diagram libraries should be considered, as current c-net visualization plugins does not offer enough expressiveness.

References

Aalst, W. M. P. van der. (2011a). Do petri nets provide the right representational bias for process mining? In *Workshop applications of region theory 2011 (art 2011)* (pp. 85–94).

Aalst, W. M. P. van der. (2011b). *Process mining - discovery, conformance and enhancement of business processes*. Springer.

Aalst, W. M. P. van der. (2013). Business process management: A comprehensive survey. *ISRN Software Engineering*, *2013*.

Aalst, W. M. P. van der, Adriansyah, A., & Dongen, B. F. van. (2011). Causal Nets: A Modeling Language Tailored towards Process Discovery. In J. Katoen & B. König (Eds.), *CONCUR 2011* (Vol. 6901, pp. 28–42). Springer.

Aalst, W. M. P. van der, Adriansyah, A., & Dongen, B. F. van. (2012). Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, *2*(2), 182–192.

Aalst, W. M. P. van der, Adriansyah, A., Medeiros, A. K. A. de, Arcieri, F., Baier, T., Blickle, T., et al. (2011). Process mining manifesto. In F. Daniel, K. Barkaoui, & S. Dustdar (Eds.), *Business process management workshops - BPM 2011 international workshops, clermont-ferrand, france, august 29, 2011, revised selected papers, part I* (Vol. 99, pp. 169–194). Springer.

Aalst, W. M. P. van der, Weijters, T., & Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, *16*(9), 1128–1142.

Bose, R. P. J. C., & Aalst, W. M. P. van der. (2009). Trace clustering based on conserved patterns: Towards achieving better process models. In S. Rinderle-Ma, S. W. Sadiq, & F. Leymann (Eds.), *Business process management workshops, BPM 2009 international workshops, ulm, germany, september 7, 2009. revised papers* (Vol. 43, pp. 170–181). Springer.

Bose, R. P. J. C., & Aalst, W. M. P. van der. (2012). Process diagnostics using trace alignment: Opportunities, issues, and challenges. *Inf. Syst.*, *37*(2), 117–141.

Buijs, J. C. A. M., Rosa, M. L., Reijers, H. A., Dongen, B. F. van, & Aalst, W. M. P. van der. (2012). Improving business process models using observed behavior. In P. Cudré-Mauroux, P. Ceravolo, & D. Gasevic (Eds.), *Data-driven process discovery and analysis - second IFIP WG 2.6, 2.12 international symposium, SIMPDA 2012, campione d'italia, italy, june 18-20, 2012, revised selected papers* (Vol. 162, pp. 44–59). Springer.

Dongen, B. F. van, Desel, J., & Aalst, W. M. P. van der. (2012). Aggregating causal runs into workflow nets. *T. Petri Nets and Other Models of Concurrency*, *7400*, 334–363.

Dumas, M., & Garcia-Banuelos, L. (2015). Process mining reloaded: Event structures as a unified representation of process models and event logs. In *Application and theory of petri nets and concurrency - 36th international conference, PETRI NETS 2015, brussels, belgium, june 21-26, 2015. proceedings.* Springer.

Fahland, D., & Aalst, W. M. P. van der. (2011). Simplifying mined process models: An approach based on unfoldings. In S. Rinderle-Ma, F. Toumani, & K. Wolf (Eds.), *Business process management - 9th international conference, BPM 2011, clermont-ferrand, france, august 30 - september 2, 2011. proceedings* (Vol. 6896, pp. 362–378). Springer.

Fahland, D., & Aalst, W. M. P. van der. (2013). Simplifying discovered process models in a controlled manner. *Inf. Syst.*, *38*(4), 585–605.

Fahland, D., & Aalst, W. M. P. van der. (2015). Model repair - aligning process models to reality. *Inf. Syst.*, *47*, 220–243.

Gambini, M., Rosa, M. L., Migliorini, S., & Hofstede, A. H. M. ter. (2011). Automated error correction of business process models. In S. Rinderle-Ma, F. Toumani, & K. Wolf (Eds.), *Business process management - 9th international conference, BPM 2011, clermont-ferrand, france, august 30 - september 2, 2011. proceedings* (Vol. 6896, pp. 148–165). Springer.

Kindler, E., Rubin, V., & Schäfer, W. (2005). Incremental workflow mining based on document versioning information. In M. Li, B. W. Boehm, & L. J. Osterweil (Eds.), *Unifying the software process spectrum, international software process workshop, SPW 2005, beijing, china, may 25-27, 2005, revised selected papers* (Vol. 3840, pp. 287–301). Springer.

Klinkmüller, C., Leopold, H., Weber, I., Mendling, J., & Ludwig, A. (2014). Listen to me: Improving process model matching through user feedback. In S. W. Sadiq, P. Soffer, & H. Völzer (Eds.), *Business process management - 12th international conference, BPM 2014, haifa, israel, september 7-11, 2014. proceedings* (Vol. 8659, pp. 84–100). Springer.

Leoni, M. de, Maggi, F. M., & Aalst, W. M. P. van der. (2015). An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. *Inf. Syst.*, *47*, 258–277.

Li, C., Reichert, M., & Wombacher, A. (2009). Discovering reference models by mining process variants using a heuristic approach. In U. Dayal, J. Eder, J. Koehler, & H. A. Reijers (Eds.), *Business process management, 7th international conference, BPM 2009, ulm, germany, september 8-10, 2009. proceedings* (Vol. 5701, pp. 344–362). Springer.

Lohmann, N. (2008). Correcting deadlocking service choreographies using a simulation-based graph edit distance. In M. Dumas, M. Reichert, & M. Shan (Eds.), *Business process management, 6th international conference, BPM 2008, milan, italy, september 2-4, 2008. proceedings* (Vol. 5240, pp. 132–147). Springer.

Munoz-Gama, J., Carmona, J., & Aalst, W. M. P. van der. (2013a). Conformance checking in the large: Partitioning and topology. In F. Daniel, J. Wang, & B. Weber (Eds.), *Business process management - 11th international conference, BPM 2013, beijing, china, august 26-30, 2013. proceedings* (Vol. 8094, pp. 130–145). Springer.

Munoz-Gama, J., Carmona, J., & Aalst, W. M. P. van der. (2013b). Hierarchical conformance checking of process models based on event logs. In J. M. Colom & J. Desel

(Eds.), *Application and theory of petri nets and concurrency - 34th international conference, PETRI NETS 2013, milan, italy, june 24-28, 2013. proceedings* (Vol. 7927, pp. 291–310). Springer.

Polyvyanyy, A., Vanhatalo, J., & Völzer, H. (2010). Simplified computation and generalization of the refined process structure tree. In M. Bravetti & T. Bultan (Eds.), *Web services and formal methods - 7th international workshop, WS-FM 2010, hoboken, nj, usa, september 16-17, 2010. revised selected papers* (Vol. 6551, pp. 25–41). Springer.

Rogge-Solti, A., Mans, R., Aalst, W. M. P. van der, & Weske, M. (2013a). Improving documentation by repairing event logs. In J. Grabis, M. Kirikova, J. Zdravkovic, & J. Stirna (Eds.), *The practice of enterprise modeling - 6th IFIP WG 8.1 working conference, poem 2013, riga, latvia, november 6-7, 2013, proceedings* (Vol. 165, pp. 129–144). Springer.

Rogge-Solti, A., Mans, R., Aalst, W. M. P. van der, & Weske, M. (2013b). Repairing event logs using timed process models. In Y. T. Demey & H. Panetto (Eds.), *On the move to meaningful internet systems: OTM 2013 workshops - confederated international workshops: OTM academy, OTM industry case studies program, acm, ei2n, isde, meta4es, orm, sedes, sincom, sms, and SOMOCO 2013, graz, austria, september 9 - 13, 2013, proceedings* (Vol. 8186, pp. 705–708). Springer.

Solé, M., & Carmona, J. (2012). An smt-based discovery algorithm for c-nets. In S. Haddad & L. Pomello (Eds.), *Application and theory of petri nets - 33rd international conference, PETRI NETS 2012, hamburg, germany, june 25-29, 2012. proceedings* (Vol. 7347, pp. 51–71). Springer.

Solé, M., & Carmona, J. (2013). Amending c-net discovery algorithms. In S. Y. Shin & J. C. Maldonado (Eds.), *Proceedings of the 28th annual ACM symposium on applied computing, SAC '13, coimbra, portugal, march 18-22, 2013* (pp. 1418–1425). ACM.

Sun, W., Li, T., Peng, W., & Sun, T. (2007). Incremental workflow mining with optional patterns and its application to production printing process. *International Journal of Intelligent Control and Systems*, *12*(1), 45–55.

Weijters, A. J. M. M., & Ribeiro, J. T. S. (2011). Flexible heuristics miner (FHM). In *Proceedings of the IEEE symposium on computational intelligence and data mining, CIDM 2011, part of the IEEE symposium series on computational intelligence 2011, april 11-15, 2011, paris, france* (pp. 310–317). IEEE.