University of Padova

Department of Department of Mathematics

Master Thesis in Data Science

Conformance Checking of Large Process Model: An Approach based on Decomposition

Supervisor
Massimiliano de Leoni
University of Padova

Master Candidate
Jiawei Ma

Academic Year
2021-2022

Acknowledgments

I would first like to thank my supervisor, Professor Massimiliano de Leoni, who supported me at the beginning during the internship, giving me suggestions and encouragement in spite of some not expected difficulties, then, he guided throughout this thesis work.

I would also like to thank Data Science master course council and all teaching professors to keep the courses up to date, always bringing out cutting-edge research results. I really enjoyed and learnt a lot during these years of study, I firmly believe that the acquired knowledge are valuable assets for my future.

Contents

Listing of figures

Listing of tables

1 Introduction

Process Mining is a very active research area in recent years and this interest is still increasing, attracting attention in both academic world and industry, due to its enormous potentials in improving any *process* occurring around us. Interestingly, sometimes we do not even realize that we are surrounded by processes–from withdrawal of cash in bank to booking a doctor's appointment in hospital. Process mining is considered an intersection of many studies fields such as computer science, data mining, economics and etc. In fact, it is often seen as a link between *data science* and *process science*. A canonical division of the subject consists of three study directions: (1) *Process Models Discovery*, (2) *Conformance Checking* and (3) *Process Enhancement and improvement*.

The rush to digitalization all over the world is creating new processes, consequently, data to be stored in somewhere, this phenomena refers to the notable *"Big Data"*. This data explosion is challenging traditional process mining algorithms in all the aforementioned study areas, the objective of this thesis is to tackle one of these challenges that is the evaluation of large process models (conformance checking).

Conformance checking is the problem to pinpoint deviations between how processes are executed in reality and how processes are expected to be performed according to norms, regulations and protocols. The executions are recorded in event logs, while the expected behaviors are encoded in a process model. The complexity of the problem is exponential with respect to the size of the model, this makes the problem not scale when models become very large. To keep the problem tractable, one can decompose the model into parts for which conformance checking is carried out.

Early works related to decomposed conformance checking had already obtained satisfactory results, nevertheless, they are still lacking in performance for some use cases–for example, when the number of sub-models is unavoidably big. In this thesis we propose a novel *sub-solutions merging algorithm* which aims to overcome some weaknesses of the state-of-the-art methods and to be a valid alternative to them. The thesis is organized in a way that we first provide process mining fundamentals and go in-depth with the decomposed conformance checking theory, then, we present our algorithm by formally defining it and proving its correctness. Finally, we test the new algorithm's performance by using large synthetic dataset generated by an appropriate tool, discussing pros and cons of the solution proposed.

2 Preliminaries

In this first chapter we aim to provide some process mining fundamentals which include historical developments, objectives, basic notations and definitions, they are useful in the next chapters.

2.1 Data Science

Data science is an emerging interdisciplinary science due to the data/information explosion phenomenon occurring in these days. As observed in [1] a data scientist is often asked to answer a variety of data-driven questions:

- *What happened (history)?*
- *Why did it happen (analysis)?*
- *What will happen (prediction)?*
- *What is the best that can happen (Recommendation)?*

We would like add also that this is a very broad study field , differently from natural sciences it is more **application-oriented** science so that the *domain knowledge* in most of the cases play a crucial role.

The basic knowledge required in Data Science are partially overlapping (sub)disciplines, the Fig. 2.1 gives a illustration of this multidisciplinarity. Note that there are knowledge from computer science (e.g. algorithms and databases) but also economics and psychology related subjects are essential.

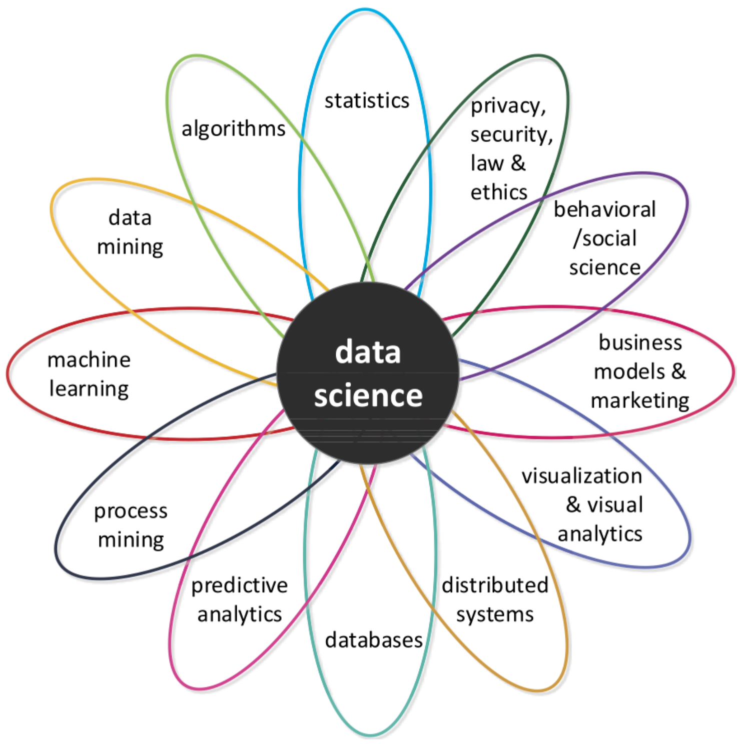


Figure 2.1: Intersection of other disciplines from numerous study areas that provide basis for becoming a data scientist. Source: [1].

2.2 Process Mining

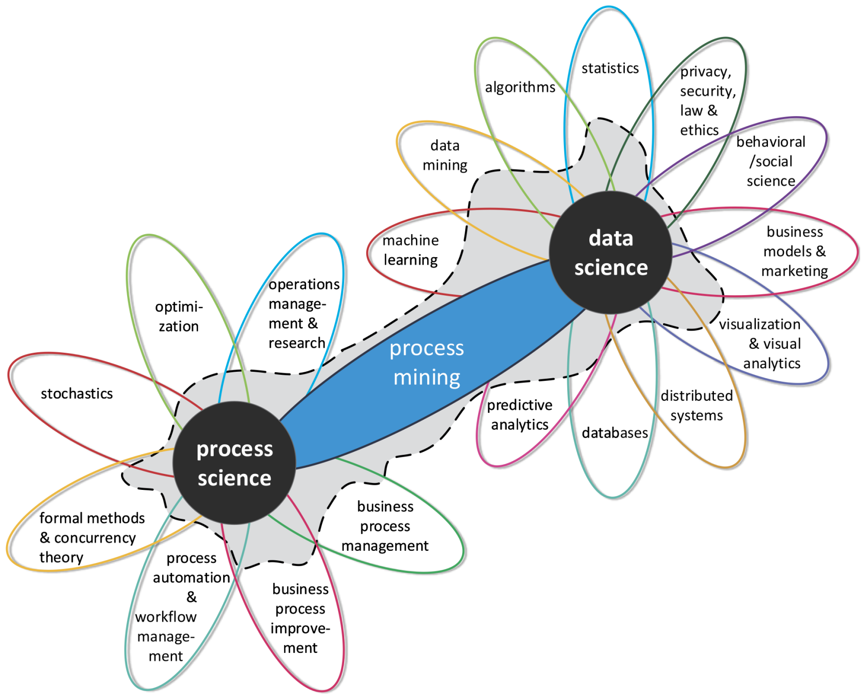
Process mining belongs to a wider science named as *process science* and a possible definition is the following, given in book [1]:

> *We use the umbrella term "process science" to refer to the broader discipline that combines knowledge from information technology and knowledge from management sciences to improve and run operational processes.*

Then, it is also provided a definition for process mining:

> *process mining can be viewed as the link between data science and process science.*

This bridge between the two sciences is well-illustrated in Fig. 2.2. Once again we deduce that a process mining expert should have a strong background knowledge of the field/sector he/she is working in.



algorithms
statistics
privacy, security, law & ethics
behavioral /social science
data mining
machine learning
data science
business models & marketing
visualization & visual analytics
optimi-zation
operations manage-ment & research
process mining
stochastics
predictive analytics
databases
distributed systems
process science
business process management
formal methods & concurrency theory
process automation & workflow manage-ment
business process improve-ment

Figure 2.2: A possible collocation of process mining, it is considered a link between data science and process science.

2.2.1 In a nutshell

Early core business processes were simple and often manual:

- To create a product you would procure materials from a supplier, manufacture it and store it;
- To fulfil a purchase, you would receive an order, retrieve the product, package it up and ship it out;
- To pay a supplier, you would receive an invoice, arrange the payment, and send confirmation.

But as businesses have digitized every aspect of working life into IT systems, core processes have become complex operational machinery in and of them- selves—too fast, frequent, interconnected and distributed to manage manually. Modern information systems such as ERP

(Enterprise Resource Planning) systems (SAP, Or- acle, etc) and BPM (Business Process Management) systems support processes in different organizations.

To concretize the ideas and emphasize the importance of the process science it is convenient to make some examples. Processes are made up of several temporally ordered activities, some probable examples of activities can be withdrawal of cash from an ATM, a doctor adjusting an X–ray machine, a citizen applying for a driver's license, submission of a tax declaration, and receipt of an e–ticket number by a traveler. Speaking about common business processes they include financial processes such as *Purchase-to-Pay (P2P) and Order-to-Cash (O2C)*, in addition to these there are many others that are brought into play on a daily basis:

- **Purchase-to-Pay (P2P)**: Refers to the business processes that start from purchasing/receiving a good/service to pay the bills, for example raw material acquisition;
- **Order-to-Cash (O2C)**: They are common sales processes so they begin with receiving an order and finish with release payment receipt from customer side (clear invoice);
- **Accounts Payable (AP)**: Refers to an account that represents a company's obligation to pay off a short-term debt to its creditors or suppliers, the business processes that involve this type of operation have the same name. These processes are quite complex because they vary significantly according to the business sector the companies belong to;
- **Accounts Receivable (AR)**: are, in simple words, any money that your customers or clients owe you for a service or product they bought on credit;
- **Procurement Processes (PP)**: are processes of purchasing of raw materials, goods and services necessary for the operation of a production activity;
- **Order Management (OM)**: This system generates huge amount of data, most of them are real-time and have to be elaborated quickly. Those data are of vital importance for the business and so gaining insight into the generator processes would be very profitable for companies;
- **Inventory Management (IM)**: This is also a core component in a enterprise management system, especially for those companies working in the commerce sector. Here each product is tracked from arriving in warehouse to being shipped out, when dealing with large number of materials monitoring processes' inefficiency can lead to a big loss for the business.

These are just few examples, of course. In addition, different companies define certain processes differently depending on their needs, the systems they use and other factors.

By their nature, *processes are not static*, nor do they always follow a standard path. Even the most accurate systems can incur errors, and over time these deviations can become a (potentially damaging) "new rule" . A successful application of process mining in the manufacturing industry is the case of ASML (the leading manufacturer of wafer scanners in the world) [7], in which process mining algorithms could answer many business questions in complex environments as the wafer scanner qualification phase, moreover, concrete suggestions could be yielded for process improvement.

2.2.2 Process Mining Manifesto

One of the cornerstone works in this field is *Process Mining Manifesto* [8], the same paper defines the word "manifesto" as following:

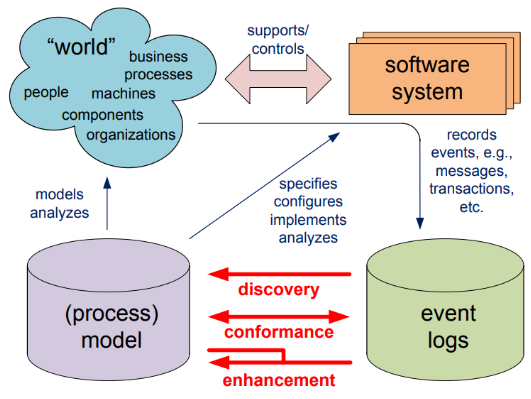
> *A manifesto is a "public declaration of principles and intentions" by a group of people. . . . . . The goal of this task force is to promote the research, development, education, implementation, evolution, and understanding of process mining.*

The key contribution is to *determine the macro division of the research directions* at that time and in the future, in this way researchers can focus on a specific area of their interest:

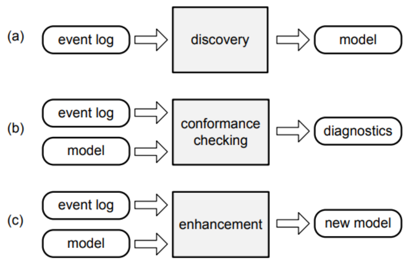
1. **Discovery**: Organizations use procedures to handle cases that are recorded by information systems, it could happen that some of the cases include informal actions or not following the standard company protocols. Therefore, it is important to discover/mine actual processes. Starting point for process mining is an *event log*, it consists of *traces* which are a sequence of *events* (e.g. purchasing a good till to pay the bills in a P2P process). Process discovery techniques produce *process models* (e.g. *petri nets*) based on event logs, one of the first and simplest algorithms is α-*algorithm* [9]. Formal definition of the previous concepts will be stated latter in this chapter;

2. **Conformance Checking**: When built process models we need to evaluate them, so we take a model and an event log as input then the modelled behavior and the observed behavior (i.e., event log) are compared. Essentially, conformance checking is necessary, for example

    - to identify deviating cases,
    - for auditing purposes,
    - to judge the quality of a discovered process model,
    - as a starting point for model enhancement or process adjustment.



(a) The standard process mining projects' workflow. The principal research directions are colored in red.



(b) Inputs and output of process mining algorithms in each three major studying areas.

Figure 2.3: The input, output and types of Process Mining. Source: [2].

This is the area we have to focus on and in this thesis we illustrate issues and possible solutions related to conformance checking challenges;

3. **Enhancement**: This is a *cyclic procedure* in which we discover a model and do model-log conformance checking, after identified model deviations we go to the discovery step to compute an improved model, then we repeat the cycle until we obtain a satisfactory result. In this phase we aim also to enrich the process with information about other *no control-flow perspectives*, such as *organizational perspective* to mine social network (handover of work) and *time perspective* which focuses on waiting times in-between activities, for more details see [1].

The typical process mining project workflow is depicted in Fig. 2.3 where we can see all elements and concepts we've just described above.

Observed behavior and modelled behavior:

When doing conformance checking there are four quality dimensions for comparing model and log which need to be considered:

- **Fitness**: A model with good fitness means that most of the behavior of the event log can be observed in the model. This is the model metric of our interest in this thesis;
- **Simplicity**: It measures the complexity of a model, this can be thought as number of events/activities embedded in the model;

- **Precision**: A model is precise if this allows "not too much" behavior, i.e. it can reproduce all event log but no more;
- **Generalization**: Very similar to the homonym metric when dealing with statistical models, also here a model should also generalize and not restrict behavior to just the data in the dataset.

The job of a data scientist is to balance all these metrics, since due to their definition very often high value in one dimension means penalizing the opposite one, i.e. *a perfectly fitting model may not be simple and a precise model with not enough data has few generalization power.* For a detailed discussion on these four quality of process discovery algorithms see [10].

2.3 Notations and Definitions

In this chapter, we present the preliminary definitions that will be used later in the thesis. Basic concepts such as sets and multisets will be recalled. Then, we present concepts commonly used in process mining such as events, trace, event logs and process models.

2.3.1 Fundamental Notations

What follow are basic notations in process mining, mathematical formalities are necessary to be more precise/without ambiguity in theorem statements and useful for proving theoretical results.

Definition 2.1 (Multisets). Let X be a set, a *multiset* of X is a mapping $M : X \to \mathbb{N}$. $\mathcal{B}(X)$ denotes the set of all multisets over X. Let M and M' be multisets over X. M contains M' if and only if $\forall_{x \in X} M(x) \geqslant M'(x)$, this inclusion is denoted by $M \geqslant M'$. The union of M and M' is denoted $M + M'$ that is $\forall_{x \in X} (M + M')(x) = M(x) + M'(x)$. The difference between M and M' is denoted $M - M\prime$ and is defined by $\forall_{x \in X} (M - M')(x) = \max(M(x) - M'(x), 0)$.

Remark. it is easy to see that $(M - M') + M'$ only if $M \geqslant M'$. A multiset can be thought as a set containing elements with repetition, e.g. $[x^2, y^2, z]$.

Definition 2.2 (Projection on sequences and multisets). Let X be a set and $X' \subseteq X$ be a subset of X, let $\sigma \in X^\star$ be a *sequence* over X and let $M \in \mathcal{B}(X)$ be a multiset over X. With $\pi_{X'}(\sigma)$ we denote the *projection of σ on X'* that is a sequence with the only elements in X', e.g. $\pi_{\{x,z\}}(\langle x, x, y, y, z \rangle) = \langle x, x, z \rangle$.Similarly, with $\pi_{X'}(M)$ we denote the *projection of M on X'*, e.g. $\pi_{\{x,z\}}([x^3, y^2, z^3]) = [x^3, z^3]$.

Sometimes, the projection is denoted by the symbol $\upharpoonright_X$, e.g. $\upharpoonright_{\{x,z\}}([x^3, y^2, z^3]) = [x^3, z^3]$.

Definition 2.3 (Function domains and ranges). Let $f \in X \rightharpoonup X'$ be a partial function [*]. $dom(f) \subseteq X$ denotes the set of all elements from X that are mapped onto some value in X' by f. With $rng(f) \subseteq X'$ we denote the set of all elements in X' that are mapped onto by some value in X, in symbols, $rng(f) = \{f(x)|\, x \in dom(f)\}$.

2.3.2 Business Process Modeling and Petri Nets

Regarding process models, there are many different process modelling languages some examples are

- Business Process Modelling Notation (BPMN);
- Event-Driven Process Chains (EPCs);
- Unified Modeling Language (UML);
- Yet Another Workflow Language (YAWL).

In [1] there are more information about these models. *Petri nets* are very simple and mathematically well-studied models to represent the processes, so in this thesis we use them to illustrate some examples.

Definition 2.4 (Petri net). A *Petri net* is a tuple $N = (P, T, F)$ with P the set of places, T the set of transitions, $P \cap T = \emptyset$ and a set of arcs or flow relations is defined as $F = (P \times T) \cup (T \times P)$.

Places are typically visualized by circles, whereas transitions are typically visualized by squares or rectangles. Taking as an example the Petri net in Fig. 2.4, we can denote this net $N_2 = (P_1, T_1, F_1)$ where the set of places is $P_2 = \{p_1, p_2, \cdots, p_{19}\}$, the set of transitions is $T_2 = \{t_1, t_2, \cdots, t_{18}\}$ and the set of arcs is $F_2 = \{(p_1, t_1), (t_1, p_2), \cdots, (t_{18}, t_{19})\}$.

The state of a Petri net is called a *marking*, and corresponds to a multiset of places. A marking is typically visualized by putting as many so-called *tokens* (black dots) at a place as the place occurs in the marking. So it is very intuitive to give the following

[*]A partial function is a standard function without specifying the it is domain that is a subset of X.

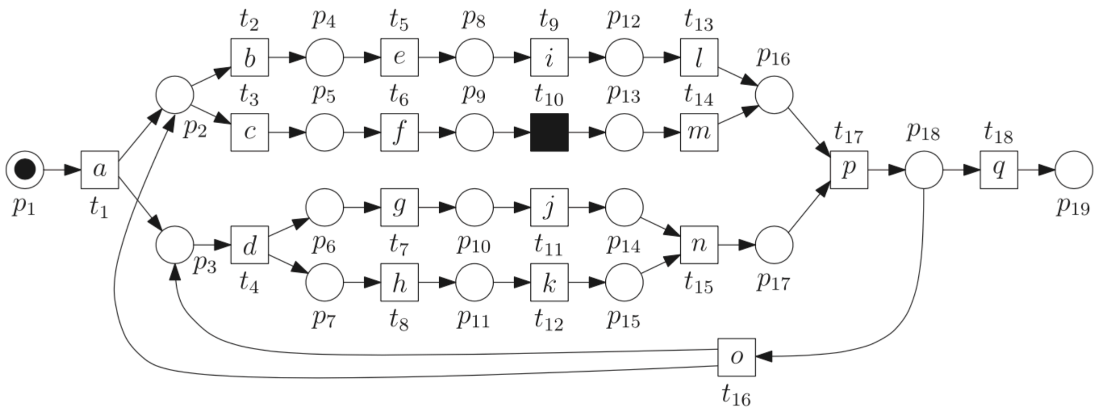


Figure 2.4: An example of Petri nets which will be used in later chapters. This system net SN_2 models a bank transfer process in which a client makes a bank transfer from one account (sender account) to another account (receiver account). The receiver account can be of a local bank or an overseas bank. Source: [3]

Definition 2.5 (Marking, transition firing and reachability)**.** Let $N = (P, T, F)$ be a Petri net. A *Marking* M is a multiset of places ($M \in \mathcal{B}(P)$).
For a node $n \in P \cup T$ (it can be a place or a transition), the set of *input nodes* is denoted by $\bullet n = \{n' | \, (n', n) \in F\}$, vice versa, the set of *output nodes* is $n\bullet = \{n' \; (n, n') \in F\}$.
A transition $t \in T$ is *enabled* by a marking M if and only if $M \geqslant \bullet t$. The *firing* of an enabled transition t in marking M is denoted as $(N, M)[t\rangle(N, M')$, where $M' = (M - \bullet t) + t\bullet$, i.e. the transition t fires consuming one token form each of input places in $\bullet t$ and producing one token at each of the output places in $t\bullet$.
A marking M' is *reachable* from a marking M if and only if there exists a sequence of transitions $\sigma = \langle t^1, t^2, \cdots, t^n\rangle$ such that $\forall_{0 \leqslant i < n}(N, M^i)[t^{i+1}\rangle(N, M^{i+1})$ with $M^0 = M$ and $M^n = M'$.

Since real-world processes either have an human-readable name or a company code for each of the activities executable in them, then it was given the following

Definition 2.6 (Labelled Petri net and invisible transitions)**.** A *labelling function* is $l \in T \rightharpoonup \mathcal{U}_A$, where $\mathcal{U}_A$ is some universe of activity labels and a labelled Petri net is defined as $N = (P, T, F, l)$. A transition t is called *invisible* if and only if it is not mapped to any activity label by the labelling function. Otherwise, transition t is visible and corresponds to an observable activity $a = l(t)$.

The labels are used to link transitions in the Petri net to activities in an activity log. The reachability condition can be rewritten for a labelled Petri net: $(N, M)[\sigma_v \rhd (N, M')$ if and only if there exists a sequence σ such that $(N, M)[\sigma\rangle(N, M')$ with $l(\sigma) = \sigma_v$. Always taking as an example the Petri net in Fig. 2.4 we have that $N_2 = (P_1, T_2, F_2, l_2)$ with l_2 a labelling function that maps transition t_1 onto activity label a, t_2 onto b, etc. N_2 is a labelled Petri net and t_{10} is an invisible transition.

Typically, processes have an initial state and a well-defined final state, similarly, when replaying an activity log on a Petri net, the Petri net needs to have an initial marking to start with, and final markings to conclude whether the replay has reached a proper final marking. We say that a firing sequence is *complete* when it starts from a marking only containing the initial place and ends at a marking only containing the final places (e.g. for the Petri net N_2,$\langle t_1, t_3, t_6, t_{10}, t_{14}, t_4, t_7, t_8, t_{11}, t_{12}, t_{15}, t_{17}, t_{18}\rangle$ is a complete firing sequence).

Definition 2.7 (System net). A *system net* is a triplet $SN = (N, I, O)$ where $N = (P, T, F, l)$ is a labelled Petri net, $I \in \mathcal{B}(P)$ is the initial marking and $O \in B(P)$ is the final marking. *universe of system nets* is denoted by $\mathcal{U}_{SN}$.

The net $SN_2 = (N_2, I_2, O_2)$ in Fig. 2.4 is a system net, with an initial marking $I_1 = [p_1]$ and a final marking $O_1 = [p_{19}]$.

Some other notations on system nets are useful:

Definition 2.8. Let $SN = (N, I, O) \in \mathcal{U}_{SN}$ be a system net with $N = (P, T, F, l)$.

- $T_v(SN) = dom(l)$ is the set of visible transitions in SN;
- $A_v(SN) = rng(l)$ (range) is the set of observable activities in SN;
- $T_v^u(SN) = \{t \in T_v(SN) | \forall_{t' \in T_v(SN)} l(t) = l(t') \Rightarrow t = t'\}$ is the set of unique visible transitions in SN;
- Similarly, $A_v^u(SN) = \{l(t) | \, t \in T_v^u(SN)\}$ is the set of corresponding unique observable activities in SN.

2.3.3 Event Logs

The following is a fragment of an anonymized real-life event log contains events of sepsis cases from a hospital. Sepsis is a life threatening condition typically caused by an infection. One case represents the pathway through the hospital and the events were recorded by the ERP system of the hospital:

```
<trace>
<string key="concept:name" value="A"/>
        <event>
                <boolean key="InfectionSuspected" value="true"/>
                <string key="org:group" value="A"/>
                <boolean key="DiagnosticBlood" value="true"/>
                <boolean key="DisfuncOrg" value="true"/>
                <boolean key="SIRSCritTachypnea" value="true"/>
                <boolean key="Hypotensie" value="true"/>
                <boolean key="SIRSCritHeartRate" value="true"/>
                <boolean key="Infusion" value="true"/>
                <boolean key="DiagnosticArtAstrup" value="true"/>
                <string key="concept:name" value="ER Registration"/>
                <int key="Age" value="85"/>
                <boolean key="DiagnosticIC" value="true"/>
                <boolean key="DiagnosticSputum" value="false"/>
                <boolean key="DiagnosticLiquor" value="false"/>
                <boolean key="DiagnosticOther" value="false"/>
                <boolean key="SIRSCriteria2OrMore" value="true"/>
                <boolean key="DiagnosticXthorax" value="true"/>
                <boolean key="SIRSCritTemperature" value="true"/>
                <date key="time:timestamp"
                value="2014-10-22T11:15:41.000+02:00"/>
                <boolean key="DiagnosticUrinaryCulture" value="true"/>
                <boolean key="SIRSCritLeucos" value="false"/>
                <boolean key="Oligurie" value="false"/>
                <boolean key="DiagnosticLacticAcid" value="true"/>
                <string key="lifecycle:transition" value="complete"/>
                <string key="Diagnose" value="A"/>
                <boolean key="Hypoxie" value="false"/>
                <boolean key="DiagnosticUrinarySediment" value="true"/>
                <boolean key="DiagnosticECG" value="true"/>
        </event>
        <event>
```

```
                <float key="Leucocytes" value="9.6"/>
                <string key="org:group" value="B"/>
                <string key="lifecycle:transition" value="complete"/>
                <string key="concept:name" value="Leucocytes"/>
                <date key="time:timestamp"
                value="2014-10-22T11:27:00.000+02:00"/>
        </event>
        <event>
                <float key="CRP" value="21.0"/>
                <string key="org:group" value="B"/>
                <string key="lifecycle:transition" value="complete"/>
                <string key="concept:name" value="CRP"/>
                <date key="time:timestamp"
                value="2014-10-22T11:27:00.000+02:00"/>
        </event>
        ............
</trace>
<trace>
............
```

Intuitively, the event log contains as many *traces* as the information systems have recorded at the time of data extraction. Each trace consists of *ordered events* which need to refer to a single process instance, often referred to as *cases*. Then, each case is enriched by many *attributes*, examples of typical attribute names are activity, timestamp, costs, and resource, here we have a lot of domain-specific parameters which are *variables* and processes with data are called *data-aware processes* [11]. Finally, note that there is an attribute named "lifecycle:transition" which indicate the state of an event, example values are start, schedule, complete, etc.

Keeping in mind the concrete example, we give the formal definitions for the above concepts.

Definition 2.9 (Traces). Let $SN = (N, I, O) \in \mathcal{U}_{SN}$ be a system net. $\phi(SN) = \{\sigma_v | \, (N, I)[\sigma_v \triangleright (N, O)\}$ is the set of visible traces starting in marking I and ending in marking O. $\phi_f(SN) = \{\sigma | \, (N, I)[\sigma\rangle(N, O)\}$ is the corresponding set of complete firing sequences.

Definition 2.10 (Event log). $\mathcal{U}_A$ denotes the universe of activities. An *event log* L is a multiset of activity sequences/traces.

3
Decomposed Conformance Checking

In this chapter we enter into detail about conformance checking, decomposed conformance checking when data increase considerably and the traditional algorithms are not able to handle such amount of data to be processed. We recall that the conformance checking quality metric studied throughout this thesis is fitness.

3.1 Alignment-based conformance checking

Conformance checking is one of the four main areas in Process Mining as we have already mentioned in Chapter 2. Essentially, the goal is to evaluate the quality of discovered Petri nets and identify process bottlenecks according to the four quality dimensions: simplicity, fitness,generalization and precision. Many quantitative approaches are available and two of them are very popular: *token replay and alignment*. We preferred the **alignment-based** approach but we give also an overview of the other technique.

3.1.1 Token Replay

Replay simply means that we use both event log and a process model as input then we compare the behaviors observed in the log and behaviors reproducible by the model. In Chapter 2 we introduced the concept of *firing sequence* and token replay approach consists of taking a trace

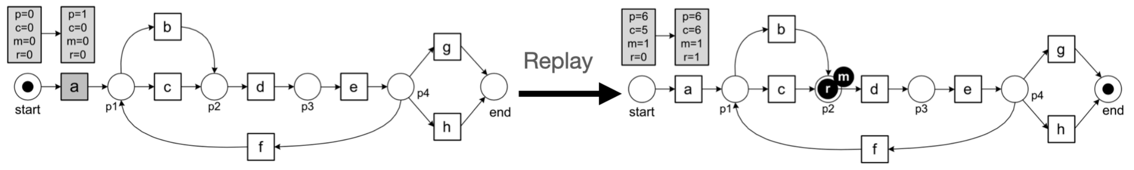


Figure 3.1: This an example of token replay of a trace $\sigma = \langle a, d, c, e, h \rangle$ on a system net SN. We skipped intermediate steps and go to calculate the final fitness: $fitness(\sigma, SN) = \frac{1}{2}\left(1 - \frac{1}{6}\right) + \frac{1}{2}\left(1 - \frac{1}{6}\right) = 0.8333$. Source: [1].

from the log and try to replay this in the model. When replaying we take into account four measures:

- *Produced tokens* (p);
- *Consumed tokens* (c);
- *Missing tokens* (m);
- *Remaining tokens* (r).

Then, given a system net SN and a trace σ, we define *token replay fitness* as:

$$fitness(\sigma, SN) = \frac{1}{2}\left(1 - \frac{m}{c}\right) + \frac{1}{2}\left(1 - \frac{r}{p}\right).$$

Note that if $m = 0$ then $\left(1 - \frac{m}{c}\right) = 1$ and if $r = 0$ then $\left(1 - \frac{r}{p}\right) = 1$. So, we have that $0 \leq fitness(\sigma, SN) \leq 1$, one trace perfectly fits a model ($fitness = 1$) if there are no missing and remaining tokens, on the other hand, one trace does not fit a model ($fitness = 0$) if number of missing tokens equals number of consumed tokens and number of remaining tokens equals number of produced tokens ($m = c,\ r = p$). In Fig. 3.1 we show an example of token replay fitness calculation.

3.1.2 Alignment-based Conformance Checking

Token-based replay seems to be a good method to evaluate discovered models, but that is not a always-good method for any scenario, some of it is limitations are discussed in [12] and in the same paper was introduced *alignment-based* replay for the first time. In this new approach fitness of a trace is measured by "summing up costs of all missing steps for visible activities". We explain this concept by making a example. For the Petri net in Fig. 3.1 and the trace $\sigma = \langle a, d, c, e, h \rangle$ we can have the following alignment:

L	a	$\gg$	d	c	e	h
M	a	c	d	$\gg$	e	h

Where by L and M we mean *Log move* and *Model move* respectively. Intuitively, we try to mimic an activity in the model by using an activity in the trace, if one activity cannot be mimicked then we use the symbol "$\gg$" which implies *no step or log move* on the M row, on the other hand, if an activity in the trace cannot be mimicked by an activity in the model then we use the same symbol on the row L (model move). For an invisible activity in the discovered model we use the symbol "τ" on the L row. We say *synchronous move* if both model and trace agree on an activity move. Usually, the row on the top is always related to log moves and that on the bottom is related to model moves, so in these thesis we keep this convention and omit the first column from now on.

Using this simple case we can observe that there may be *infinite alignments* for a single trace, for example, in the model we can always "go back" to the place $p1$ from the place $p4$ (*loop*), looping over and over again we get a different alignment every time. it is easy to understand that there are alignments that fit most for a trace-model pair, those with minimum number of misalignments but this is not always the case (when a model move is not equivalent to a log move). We will give formal definitions for these concepts.

Definitions and notations:

We can think of a *move* as a pair (l, m) where l refers to an activity in the trace and m indicates a model activity.

Definition 3.1 (Legal moves). Let L be an event log and let A the set of activities in the log. Let $SN = (N, I, O) \in \mathcal{U}_{SN}$ be a system net with $N = (P, T, F, l)$.The set of all possible *legal moves* is defined as $A_{LM} = \{(a, (a,t))|\, a \in A \wedge t \in T \wedge l(t) = a\} \cup \{(\gg, (a,t))|\, a \in A \wedge t \in T \wedge l(t) = a\} \cup \{(\gg, (\tau, t))|\, t \in \ \wedge t \notin dom(l)\} \cup \{(a, \gg)|\, a \in A\}$.

An alignment consists of legal moves and models moves should bring the token in the initial place to one of the final places.

Definition 3.2 (Alignment). Let L be an event log and let A the set of activities in the log. Let $\sigma_L \in L$ be a log trace and $\sigma_M \in \phi_f(SN)$ be a complete firing sequence of the system net SN. An alignment of σ_L and σ_M is a not empty sequence $\gamma \in A_{LM}$ such that, ignoring all non-sync moves $\gg$, the projection on the first elements yields σ_L and the projection on the last elements yield s σ_M.

Among all possible alignments we would like to order them according to the number of non-sync moves: both model and log moves. Generally, a model move is not equivalent to a log move depending on if we are more "tollerant" to one over the other one. So, we give the following general

Definition 3.3 (Cost of alignment). The *cost function* is a function $\delta \in A_{LM} \to \mathbb{Q}_{\geq 0}$ such that synchronous moves have cost zero, i.e. $\delta(a, (a, t)) = 0$ for all $a \in A$, both model and log moves have positive costs, i.e. $\delta(\gg, (a, t)), \delta(a, \gg) > 0$ with $l(t) = a$ and $a \in A$, a move in the model also has no costs if the transition is invisible, i.e. $\delta(\gg, (\tau, t)) = 0$ if $t \notin dom(l)$. The *cost of a non-empty alignment* $\gamma \in A_{LM}$ is the sum of all costs: $\delta(\gamma) = \sum_{(a,m)\in\gamma} \delta(a, m)$.

The standard cost function that has unit cost for both model and log move is denoted by δ_1. Now we give the following straightforward

Definition 3.4 (Optimal alignment). Let L be an event log and let A the set of activities in the log. Let $SN \in \mathcal{U}_{SN}$ be a non-empty system net (i.e. $\phi(SN) \neq \emptyset$). Given a trace $\sigma_L \in L$, an alignment $\gamma \in A_{LM}$ for σ_L is *optimal* if the cost associated with aligning σ_L and the corresponding complete firing sequence of the system net $\sigma_M \in \phi_f(SN)$ is lower or equal than any other alignment for σ_L.

We use $\lambda(\sigma_L, SN) \in A \to A_{LM}$ to denote a deterministic function that map a trace to one of it is optimal alignments. As an example, we take the Petri net in Fig. 3.1, the trace $\sigma = \langle a, d, c, e, h \rangle$, the following are optimal alignments:

a	$\gg$	d	c	e	h
a	c	d	$\gg$	e	h

a	$\gg$	d	c	e	h
a	b	d	$\gg$	e	h

ù

It is possible to define an *alignment-based fitness* of a trace starting from an optimal alignment and the associated cost, we also want it to be *normalized*: this is a value between zero (maximal cost) and one (perfect fitness or cost zero). To do so we need the concept of *worst-case alignment* in which there are no synchronous moves and only “moves in model only” and “moves in log only”. Always considering the example in Fig. 3.1 we have the following worst-case alignment for the trace:

a	c	d	c	e	h	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	a	d	c	e	h

Following the usual notations we define:

- $move_M(SN) = \min_{\sigma_M \in \phi_f(SN)} \sum_{t \in \sigma_M} \delta(\gg, (l(t), t))$: this is the minimal costs of an alignment between an empty log trace and a complete firing sequence of the system net.
- $move_L(\sigma_L) = \sum_{a \in \sigma_L} \sigma(a, \gg)$: this is the costs of an alignment between σ_L and an empty model trace or this equals to the length of the trace.

Definition 3.5 (Alignment-based fitness)**.** Following the notations of this chapter, the *alignment-based fitness* is defined as:

$$fitness(\sigma_L, SN, \delta) = 1 - \frac{\delta(\lambda(\sigma_L, SN))}{move_M(SN) + move_L(\sigma_L)}.$$

Then, it is possible to extend the fitness to the overall event log as follows:

$$fitness(L, SN, \delta) = 1 - \frac{\sum_{\sigma_L \in L} \delta(\lambda(\sigma_L, SN))}{|L| \times move_M(SN) + \sum_{\sigma_L \in L} move_L(\sigma_L)}.$$

Back to the same example we have that, considering the unit cost function δ_1, $fitness(\sigma, SN, \delta_1) = 1 - \frac{2}{6+5} \approx 0.82$.

3.2 Decomposing Petri nets

Due to the boom of business investment in information technology and global digitalization in recent years, the term “Big Data” began to pick up steam around the IT journals and scientific articles. Looking at Fig. 3.2 data accumulation trend is unstoppable, the *cloud computing* seems to be a possible solution for this problem. When talking about Big Data a notable characterization of it is the so called the “four V’s of data”: *Volume, Velocity, Variety, Veracity* [13].

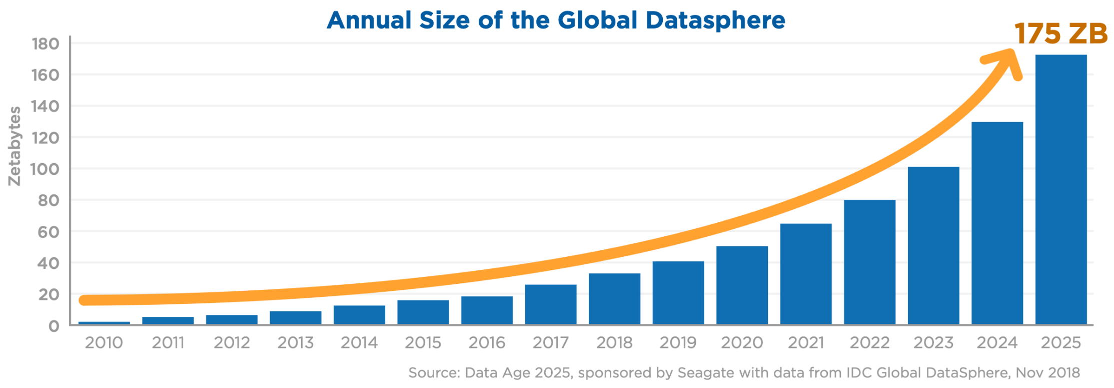


Figure 3.2: Amount of data stored over the recent years and a future trend forecast.

- The first characterize impressive amount of data to be processed;
- The second is the incredible growth rate of the data generation;
- The third refers to the different forms in which data are generated;
- The last one measures the different degrees of trustworthiness of the data.

In Process Mining large amount of data to be processed is a big challenge as well, especially during model discovery and conformance checking. Our focus is of course on the conformance checking. It is proved that the complexity of conformance checking problems are exponential when models' size increase [4]. This may make traditional conformance checking algorithms unacceptably slow or even infeasible. Fortunately, many techniques are developed to overcome this problem, both in the field of model discovery and in the field of conformance checking. We can classify the methods in two categories for decomposition-based conformance checking:

- **Case-based decomposition** (called "vertical partitioning of the event log"): Fig. 3.3. The idea is very easy to understand, the event log is divided in sub-logs or distributed over a set of compute nodes, then the computation will be do in parallel. This approach is very convenient to use when we have small process model and big event log;
- **Activity-based decomposition**: Sometimes, if the model is too big then conformance checking even a single trace would be infeasible, in this case we have to move to the *activity-based* approach. In this scenario we focus on decomposing process models (e.g. Petri nets) into sub-models afterwords we do the conformance checking for each sub-model taking into account only activity within the analysed model (i.e. trace projection on the sub-model). Finally, we combine the *local results to obtain a global conformance evaluation*. See Fig. 3.4. Comparing to the case-based method that can gain a *linear*

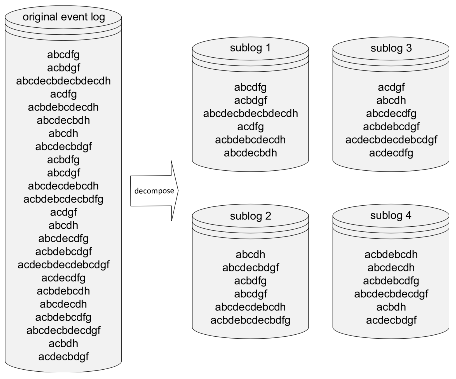


Figure 3.3: A case-based decomposition example, each sub-log can potentially processed by many compute nodes or in a single node with a processor multi-core in parallel. Doing so we can gain a linear speed-up in terms of computation time. Source: [1].

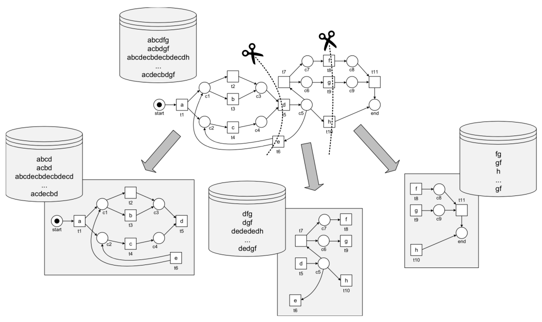


Figure 3.4: A activity-based decomposition example, we divide the conformance checking problem into smaller conformance checking problems: we properly decompose the Petri net and calculate the projection of the traces on each subnet. Then, we merge the local results to get a conformance evaluation on the overall net. In this way we can potentially reach more than linear speed-up in terms of computation time. This is our studying direction in this thesis. Source: [1].

speed-up in terms of computation time by using this activity-based approach we can potentially reach *higher speed-up rate*. Thus, **in this thesis we aim to give our contribution to improve existing activity-based decomposition algorithms.**

3.2.1 Why Decomposed Conformance Checking?

To illustrate computational issues related to conformance checking of big models and to show a case in which we can gain more than linear speed-up, we use the example in Fig. 3.5a. After a token is fired by the transition t_0 a new token is produced in the place p_2, then from this place it can be consumed by one of the n parallel transitions. We can measure the complexity of a Petri net by counting how many possible traces it admits. Thus, we have $n!$ possible traces for this very simple Petri net that is a lot more than a linear increase w.r.t n. However, if we are able to split the net into two subnets with half of the parallel transitions for each then we can execute the conformance checking simultaneously for each subnet. By doing so, the possible number of traces would be reduced to $(\frac{n}{2})!$, we show this net splitting in Fig.3.5b. To appreciate more this complexity reduction we suppose $n = 8$, then $8! = 40320$ and $(8/2)! = 4! = 24$, the difference is $40320/24 = 1680$ times.

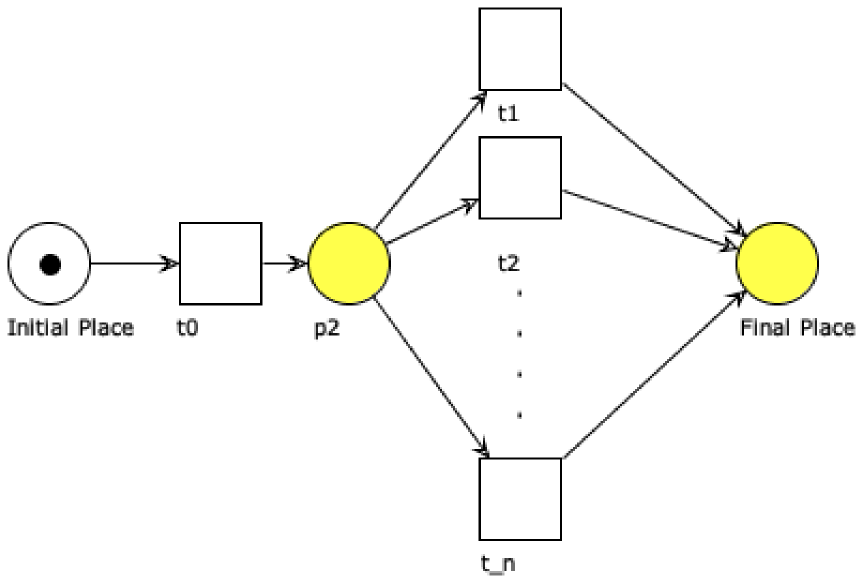
3.2.2 Valid Decomposition

Merging the local results into a global result for the original net is not easy task because we would like that the final fitness to be as much as possible close to the true value, as we can obtain by using some conformance checking techniques on the overall net.
To reach this goal, we have to

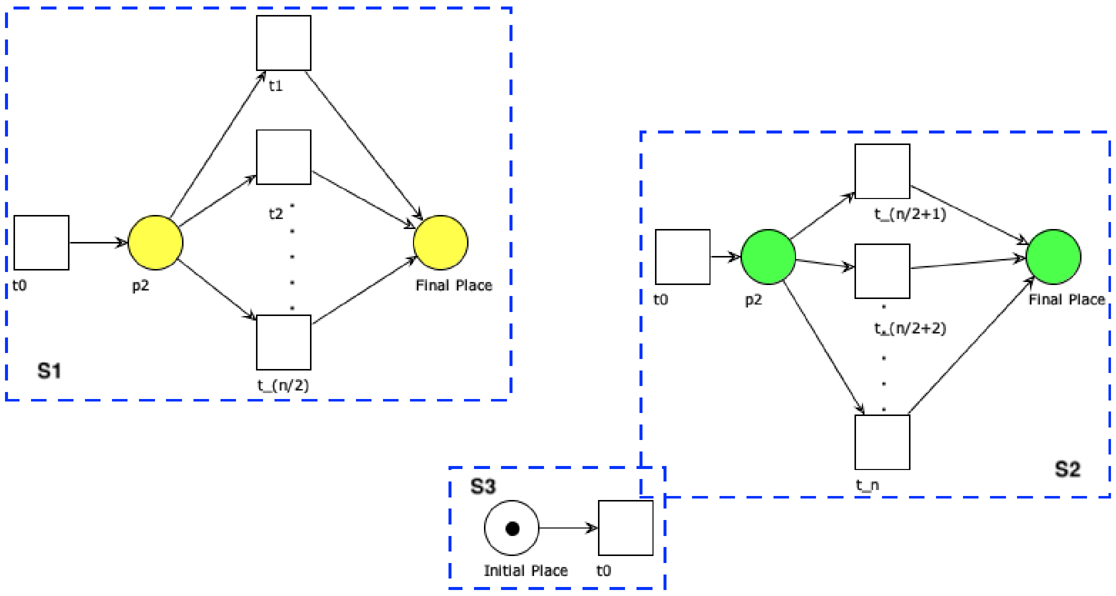
- *properly decompose the overall Petri net and*
- *contrive local results merging techniques.*

We discuss all these two topics in this chapter.

Ideally, a good decomposition is such that each subnet is *well-separated/independent* from the others, in this way we can evaluate the single subnet without consider the results of the remaining ones. If we take the decomposition example in Fig. 3.5b, that is not a good decomposition because the subnets S_1 and S_2 are not independent from each other, in particular they share the place p_2. To see this, consider the firing sequence $\langle t_0, t_n \rangle$ that is a perfect fitting trace



(a) A very simple Petri net with n parallel activities.



(b) A possible net decomposition.

Figure 3.5: A simple Petri net to show that the number of possible activity is $n!$, this is much more than linear increase w.r.t. the number of activities n. However, using the net decomposition we can incredibly reduce a lot the complexity of the initial net.

for the net in Fig. 3.5a, however, if we project this trace on the three subnets then we get respectively: $\langle t_0 \rangle$, $\langle t_0, t_n \rangle$ and $\langle t_0 \rangle$. We observe that the projected trace of the subnet S_1 cannot be perfectly replayed by the net, whereas the other projected traces are perfect fitting traces for their respective subnet, it is very hard to join them together to get a cost zero alignment for the overall net.
To overcome issues related to common places and other ambiguities in [4] the authors studied the problem and they introduced the concept of *valid decomposition* and *border activities*.

Definition 3.6 (Valid decomposition). Let $SN \in \mathcal{U}_{SN}$ be a system net with labeling function l. $D = \{SN^1, SN^2, \cdots, SN^n\} \subset \mathcal{U}_{SN}$ is a *valid decomposition* if

- $SN^i = (N^i, I^i, O^i)$ is a system net with $N^i = (P^i, T^i, F^i, l^i)$ for all $1 \leq i \leq n$,
- $l^i = l \restriction_{t^i}$ for all $1 \leq i \leq n$,
- $P^i \cap P^j = \emptyset$ for all $1 \leq i \leq j \leq n$,
- $T^i \cap T^j \subset T^u_v(SN)$ $1 \leq i \leq j \leq n$ and
- $SN = \bigcup_{1 \leq i \leq n} SN^i$.

$\mathcal{D}(SN)$ denotes the set including all valid decompositions of SN.

In addition to null intersection of all places in the net, we should ensure also that each place and invisible transition resides in just one subnet, if there are multiple transitions with the same label, they should reside in the same subnet. Only unique visible transitions can be shared among different subnets. Moreover, each edge appears in precisely one of the subnets.
We illustrate an example of valid decomposition in Fig. 3.6.In this case the valid decomposition is $D = \{S_1, S_2, S_3, S4\}$.
From the definition we observe, unlike the places, some transitions can be shared among the subnets, they are activities on the borders. This a concept strictly related to the decomposition and it is very important for developing local results merging algorithms, since we need them to satisfy a certain condition.

Definition 3.7 (Border activities). Let $SN = (N, I, O) \in \mathcal{U}_{SN}$ be a system net with $N = (P, T, F, l)$ and let $D = \{SN^1, SN^2, \cdots, SN^n\} \in \mathcal{D}(SN)$. The set of *border activities* of the valid decomposition D is defined as $A_b(D) = \{l(t) | \ t \in T^i \cap T^j for some positive indices i, j \leq n\}$.
Let $a \in rng(l)$ be an observable activity. The set of subnets containing a as a border activity is denoted by $SN_b(a, D) = \{SN^i | SN^i \in D \wedge a \in A_b(SN^i)\}$.

The valid decomposition D_1 have the border activities $A_b(D_1) = \{a, g, d\}$.

Remark. Some properties immediately follow from the definition:

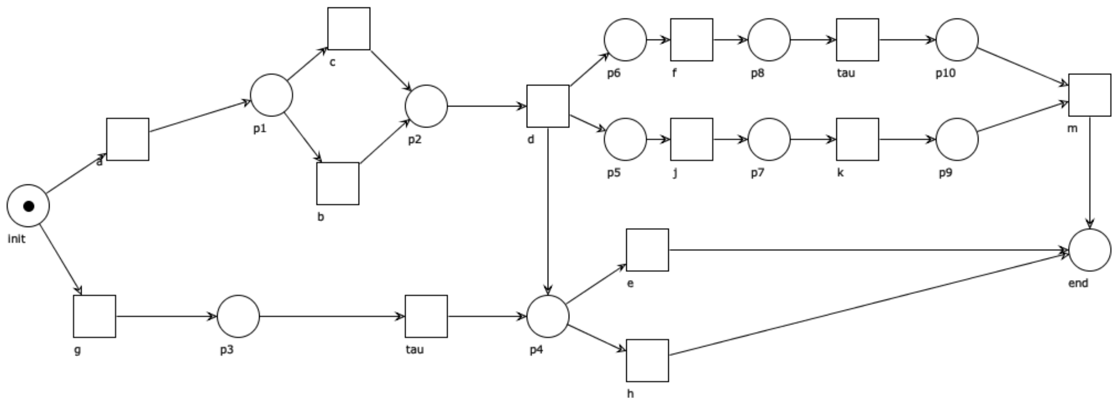
- $A_b(D) \subset A_v^u(SN)$, i.e. a border activity can only be an activity that has a unique label;
- if a is a non-unique activity then it must be that $|SN_b(a, D)| = 1$. On the other way, if a is a border activity then $|SN_b(a, D) > 1|$;

A valid decomposition guarantees that a trace that is fitting the system SN is also fitting each subnet SN^i in which SN is decomposed and vice versa. See the following theorem proved in [4]:

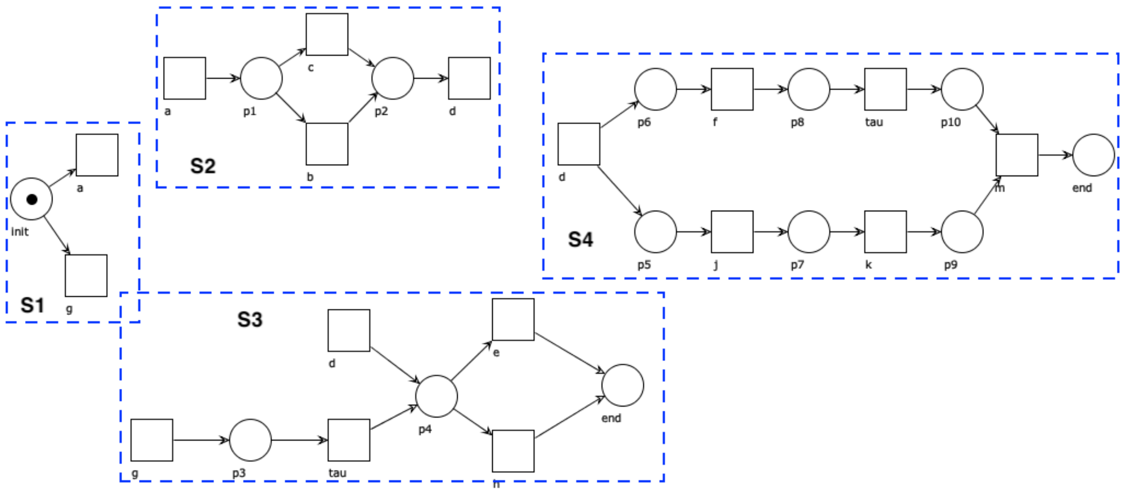
Theorem 3.8 (Decomposed conformance checking). Let L be an event log and let A the set of activities in the log. Let $SN = (N, I, O) \in \mathcal{U}_{SN}$ be a system net with $A_v(SN) = A$ and $D = \{SN^1, SN^2, \cdots, SN^n\}$ be an any valid decomposition of SN. Then, a trace $\sigma_L \in L$ is a perfect fitting trace for the system net SN if and only if $\sigma_L \upharpoonright_{A_v(SN^i)}$ is a perfect fitting trace for the subnet SN^i for all $1 \leq i \leq n$.

Proof. We adopt the same notations introduced in this chapter.
($\Rightarrow$) Let σ_M be a complete firing sequence in an optimal alignment for the trace σ_L, i.e. $(N, M_{init})[\sigma_M\rangle(N, M_{final})$. it is trivial that the projected firing sequence on each subnet $\sigma_M \upharpoonright_{A_v(SN^i)}$ is a perfect firing sequence.
($\Leftarrow$) Let σ_M^i be a complete firing sequence in an optimal alignment for the projected trace $\sigma_L \upharpoonright_{A_v(SN^i)}$ on the subnet SN^i for all $1 \leq i \leq n$, i.e. $(N, M_{init}^i)[\sigma_M^i\rangle(N, M_{final}^i)$ for all $1 \leq i \leq n$. Since, due to the definition of a valid decomposition, the different subnets only share *unique visible border transitions*, moreover, these transitions move synchronously with the corresponding activities in the trace σ_L ($l(t) = a$), then we can easily *stitch* together all sub-alignments, for example by using the alignment stitching algorithm presented in [6], the merging result will be a valid alignment and equal to σ_L (see the Chapter 5 for detailed information). $\square$

By applying this powerful theorem, given a decomposed system net and after evaluate each sub-log on the respective subnet, we can already compute the percentage of fitting traces in the overall log, without complex results analysis. The remaining traces need to be further evaluated. Consider the perfect fitting trace $\langle a, b, d, f, j, k, m, e\rangle$ for the system net SN_1, if we decompose the net as in Fig. 3.6b then we have the following optimal alignment and



(a) The system net SN_1. There two invisible activities that we denote with τ_1 and τ_2, they reside in S_4 and S_3 respectively.



(b) A valid decomposition of SN_1

Figure 3.6: An example system net SN_1 and it is a valid decomposition.

sub-alignments:

(Optimal alignment:)

$$\begin{array}{c|c|c|c|c|c|c|c} a & b & d & f & j & k & m & e \\ \hline a & b & d & f & j & k & m & e \end{array}$$

(Optimal sub-alignments:)

$$S_1\text{:}\begin{array}{|c|} a \\ \hline a \end{array} \qquad S_2\text{:}\begin{array}{c|c|c} a & b & d \\ \hline a & b & d \end{array} \qquad S_3\text{:}\begin{array}{c|c} d & e \\ \hline d & e \end{array} \qquad S_4\text{:}\begin{array}{c|c|c|c|c|c} d & f & \tau & j & k & m \\ \hline d & f & \tau_1 & j & k & m \end{array}$$

Adapted cost function:

Sometimes knowing simple percentage of fitting trace in an event log is not sufficient, we need also to know the overall cost of a single trace's optimal alignment.
Take the trace $\langle a, b, f, j, k, m, e \rangle$, optimal alignment and sub-alignments are:

(Optimal alignment:)

$$\gamma : \begin{array}{c|c|c|c|c|c|c|c} a & b & d & f & j & k & m & e \\ \hline a & b & \gg & f & j & k & m & e \end{array}$$

(Optimal sub-alignments:)

$$\gamma_1\text{:}\begin{array}{|c|} a \\ \hline a \end{array} \qquad \gamma_2\text{:}\begin{array}{c|c|c} a & b & d \\ \hline a & b & \gg \end{array} \qquad \gamma_3\text{:}\begin{array}{c|c} d & e \\ \hline \gg & e \end{array} \qquad \gamma_4\text{:}\begin{array}{c|c|c|c|c|c} d & f & \tau & j & k & m \\ \hline \gg & f & \tau_1 & j & k & m \end{array}$$

We have that the optimal alignment γ has the cost $\delta_1(\gamma) = 1$, the optimal sub-alignments have the costs $\delta_1(\gamma_1) = 0$ and $\delta_1(\gamma_2) = \delta_1(\gamma_3) = \delta_1(\gamma_4) = 1$. Intuitively, to obtain the overall cost we cannot simply sum up the sub-alignments' costs because in this way we would count multiple (three) times the same misalignment in the activity d. Motivated by this example the cost function is adapted in [3] as follows:

Definition 3.9 (Adapted cost function). Let $D = \{SN^1, SN^2, \cdots, SN^n\} \in \mathcal{D}(SN)$ be a valid decomposition of the subnet SN and $\delta \in A_{LM} \to \mathbb{Q}$ be a cost function. The *adapted cost function* $\delta_D \in A_{LM} \to \mathbb{Q}$ for decomposition D is defined as follows:

$$\delta_D(a, m) = \begin{cases} \frac{\delta(a,m)}{|SN_b(\alpha(a,m),D)|}, & \text{if } \alpha(a, m) \neq \tau \\ \delta(a, m), & \text{otherwise.} \end{cases}$$

Where the function $\alpha \in A_{LM} \to A \cup \{\tau\}$ maps a move to its associated activity, i.e. for all $t \in T$ and $a \in A$, $\alpha(a,(a,t)) = a$, $\alpha(\gg,(a,t)) = a$, $\alpha(\gg,(\tau,t)) = \tau$ and $\alpha(a,\gg) = a$.

The concept of *adapted fitness* follows the adapted cost function:

Definition 3.10 (Decomposed fitness metric). Let $SN = (N,I,O) \in \mathcal{U}_{SN}$ be a system net with $N = (P,T,F,l)$ and let $D = \{SN^1, SN^2, \cdots, SN^n\} \in \mathcal{D}(SN)$ be a valid decomposition. Let $A_v^i := A_v(SN^i)$ be the set of visible activity of the subnet SN^i for all $1 \leq i \leq n$.
The *decomposed fitness metric* of a trace $\sigma_L \in L$ under decomposition D is defines as:

$$fitness_D(\sigma_L, SN, \delta) = 1 - \frac{\sum_{i \in \{1,\cdots,n\}} \delta_D(\lambda(\sigma_L^i, SN^i))}{move_M(SN) + move_L(\sigma_L)},$$

where $\sigma_L^i := \sigma_L \upharpoonright_{A_v^i}$.
Extending the above definition to the overall event log L:

$$fitness_D(L, SN, \delta) = 1 - \frac{\sum_{\sigma_L \in L} \sum_{i \in \{1,\cdots,n\}} \delta_D(\lambda(\sigma_L^i, SN^i))}{|L| \times move_M(SN) + \sum_{\sigma_L \in L} move_L(\sigma_L)}.$$

Back to the example trace $\langle a,b,f,j,k,m,e\rangle$ of this paragraph, it is quite straightforward to compute its fitness for the unit cost function (worst-case model moves are $\langle g, \tau_2, e\rangle$): $1 - \frac{1}{2+7} \approx 0.89$. Then, $fitness_D(\langle a,b,f,j,k,m,e\rangle, SN, \delta_1) = 1 - \frac{3/3}{2+7} \approx 0.89$, this value is as same as the result calculated by doing the conformance checking on the original net SN.

Adapted cost function as a lower bound for misalignment costs:

We have already discussed about local results that do not take into account overall net structure; a sub-alignment with minimum costs for a subnet may result expensive when it is viewed globally. Thus, we can deduce that decomposed optimal costs are lower bound for the overall optimal costs.

Theorem 3.11 (Lower bound for overall optimal costs). Let $SN \in \mathcal{U}_{SN}$ be a system net and let $D = \{SN^1, SN^2, \cdots, SN^n\} \in \mathcal{D}(SN)$ be a valid decomposition. Let $A_v^i := A_v(SN^i)$ be the set of visible activity of the subnet SN^i for all $1 \leq i \leq n$.
For any trace $\sigma_L \in L$, decomposed costs given by aligning σ_L in the subnets are lower bound

for the overall costs by aligning it in the overall net:

$$\sum_{i\in\{1,\cdots,n\}} \delta_D(\lambda(\sigma_L^i, SN^i)) \leq \delta(\lambda(\sigma_L, SN)),$$

where $\sigma_L^i := \sigma_L \upharpoonright_{A_v^i}$.
Furthermore, the decomposed fitness is an upper bound for the overall fitness:

$$fitness(\sigma_L, SN, \delta) \leq fitness_D(\sigma_L, SN, \delta).$$

Proof. We prove the lower bound by contradiction, the upper bound directly followed by using the fitness definitions.
Suppose that

$$\sum_{i\in\{1,\cdots,n\}} \delta_D(\lambda(\sigma_L^i, SN^i)) > \delta(\lambda(\sigma_L, SN)).$$

And let σ_M the firing sequence obtained by projecting the last element of an optimal alignment γ for the trace σ_L, let $\gamma_1, \gamma_2, \cdots, \gamma_n$ be some optimal sub-alignments for the projected traces $\sigma_L^1, \sigma_L^2, \cdots, \sigma_L^n$. Then, it is possible, thanks to how we define a valid decomposition, to construct a new set of sub-alignments $\gamma'_1, \gamma'_2, \cdots, \gamma'_n$ that use the σ_M^i's, with $\sigma_M^i = \sigma_M \upharpoonright_{T^i}$. Applying the definition of adapted cost function it must be that

$$\delta(\lambda(\sigma_L, SN)) = \sum_{i\in\{1,\cdots,n\}} \delta_D(\gamma'_i) < \sum_{i\in\{1,\cdots,n\}} \delta_D(\lambda(\sigma_L^i, SN^i)).$$

Thus, there must be a index $\bar{i} \in \{1, 2, \cdots, n\}$ such that $\delta(\gamma'_{\bar{i}}) < \delta(\gamma_{\bar{i}})$, this is a contradiction because $\gamma_{\bar{i}}$ is supposed to be an optimal sub-alignment. □

3.2.3 Total Border Agreement

To compute the costs of an alignment the method of decomposed costs does not always work.

Let's consider again the system net SN_1 in Fig. 3.6a, the valid decomposition in Fig. 3.6b and a new trace $\sigma_1^1 = \langle a, g, b, d, c, d, f, j, e, h, m\rangle$. We change the cost function to another one that maps model moves to 10 costs and log moves to 3 costs. Then, we have the following optimal alignment and sub-alignments:

(Optimal alignment:)

$$\gamma : \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c} a & g & b & d & c & d & f & j & \gg & e & h & \tau & m \\ \hline a & \gg & b & d & \gg & \gg & f & j & k & e & \gg & \tau_1 & m \end{array}$$

(Optimal sub-alignments:)

$$\gamma_1 \colon \begin{array}{c|c} a & g \\ \hline \gg & g \end{array} \qquad \gamma_2 \colon \begin{array}{c|c|c|c|c} a & b & d & c & d \\ \hline a & b & d & \gg & \gg \end{array} \qquad \gamma_3 \colon \begin{array}{c|c|c|c|c|c} g & \tau & d & d & e & h \\ \hline g & \tau_2 & \gg & d & e & h \end{array}$$

$$\gamma_4 \colon \begin{array}{c|c|c|c|c|c|c} d & d & f & \tau & j & \gg & m \\ \hline d & \gg & f & \tau_1 & j & k & m \end{array}$$

The costs are: $\delta(\gamma) = 22$, $\delta(\gamma_1) = 3$, $\delta(\gamma_2) = 6$, $\delta(\gamma_3) = 3$ and $\delta(\gamma_4) = 13$. According to the Defnition 3.9 we have that total adapted cost is equal to $6/2$(border activity a)$+3$(activity c)$+9/3$(border activity d)$+10$(activity k)$= 19 \neq \delta(\gamma)$, this is lower bound for the true costs (see Theorem 3.11).

We note two problems related to *local optimality*; the first one is that γ_3 did not know about the mutually exclusivity between the activity a and the activity g, in fact, it contains the firing transition g that is not a correct firing for the overall net. Secondly, the border activity a has different move type in the sub-alignments γ_1 and γ_2 (log move against sync move), due to how we calculate the adapted costs we just divided the costs caused by misalignment of the activity a by the number of subnets which contain it, not caring about the fact that in γ_2 there is a sync move.

Having the property that all border activities agree in terms of move types across all subnets is a key characterization for merging sub-alignments.

Definition 3.12 (Total border agreement). Let $SN = (N, I, O) \in \mathcal{U}_{SN}$ be a system net with $N = (P, T, F, l)$ and let $D = \{SN^1, SN^2, \cdots, SN^n\} \in \mathcal{D}(SN)$ be a valid decomposition. Let $A_v^i := A_v(SN^i)$ be the set of visible activity of the subnet SN^i for all $1 \leq i \leq n$.
Let $a \in A_b(D)$ be a border activity, $a_{LM} = \{(a, (a, t)), (\gg, (a, t)), (a, \gg)\}$ denotes all legal moves for activity a (sync move, model move and log move).
Given a trace $\sigma_L \in L$, let $\gamma^1, \cdots, \gamma^m$ be optimal sub-alignments respectively for the trace in the subnets $SN^1, \cdots, SN^m \in SN_b(a, D)$ which are subnets containing a as a border activity. The set of sub-alignments $\gamma^1, \cdots, \gamma^n$ are said to be *under border agreement* on the border activity a if $\gamma^i \restriction_{a_{LM}} = \gamma^j \restriction_{a_{LM}}$, for all $SN^i, SN^j \in SN_b(a, D)$.
The set of sub-alignments $\gamma^1, \cdots, \gamma^n$ are *under total border agreement (t.b.a.)* if border agreement is achieved one by one on all the border activities in $\gamma^1, \cdots, \gamma^n$ following the order of

their occurrences across $\gamma^1, \cdots, \gamma^m$, starting with the first occurring border activity in subnet $SN^i \in D$.

The intuition that t.b.a. leads to a good sub-alignments merging condition is proved by the next theorem, presented in [3].

Theorem 3.13 (Exact value for decomposed fitness metric under t.b.a.). Let $SN = (N, I, O) \in \mathcal{U}_{SN}$ be a system net and let $D = \{SN^1, SN^2, \cdots, SN^n\} \in \mathcal{D}(SN)$ be a valid decomposition. Let $A_v^i := A_v(SN^i)$ be the set of visible activity of the subnet SN^i for all $1 \leq i \leq n$. Given a trace $\sigma_L \in L$, let $\gamma^1, \cdots, \gamma^n$ be optimal sub-alignments respectively for the trace in the subnets $SN^1, \cdots, SN^n \in D$.
Suppose $\gamma^1, \cdots, \gamma^n$ are under t.b.a. , then the decomposed fitness metric coincide with the standard fitness computed with the overall trace on the system net SN:

$$fitness(\sigma_L, SN, \delta) = fitness_D(\sigma_L, SN, \delta).$$

Moreover, if for all the log traces in L, their corresponding set of sub-alignments is under total border agreement then

$$fitness(L, SN, \delta) = fitness_D(L, SN, \delta).$$

Proof. We have only to prove that, under t.b.a., the costs and adapted costs associated with an optimal alignment and optimal sub-alignments respectively of a trace $\sigma_L \in L$ coincide.
The proof is done by using the sub-alignment stitching rules in [6]; it was proved that if there is no conflicts between two sub-alignment then we can stitch them together and the adapted costs equals the standard costs. Thus, if we prove that we can always stitch together sub-alignments under t.b.a. then we are done.
By contradiction, because we have a *valid decomposition* then the conflicts should be on the borders, but the fact that two sub-alignments that do not agree on a border activity, the subnets involved are not necessarily in order as $\gamma^1, \cdots, \gamma^n$, breaks the t.b.a. property. □

Obviously, the sub-alignments illustrated in this subsection do not satisfy t.b.a. property and, indeed, the two fitness metrics do not coincide. As an example of sub-alignments which are under t.b.a. and so Theorem 3.13 holds true, we consider again the trace $\langle a, b, f, j, k, m, e\rangle$ and the alignments:

(Optimal alignment:)

$$\gamma : \begin{array}{c|c|c|c|c|c|c|c} a & b & d & f & j & k & m & e \\ \hline a & b & \gg & f & j & k & m & e \end{array}$$

(Optimal sub-alignments:)

$$\gamma_1 \colon \begin{array}{|c|} a \\ \hline a \end{array} \qquad \gamma_2 \colon \begin{array}{c|c|c} a & b & d \\ \hline a & b & \gg \end{array} \qquad \gamma_3 \colon \begin{array}{c|c} d & e \\ \hline \gg & e \end{array} \qquad \gamma_4 \colon \begin{array}{c|c|c|c|c|c} d & f & \tau & j & k & m \\ \hline \gg & f & \tau_1 & j & k & m, \end{array}$$

we colored in green the moves involving border activities. Clearly, the t.b.a property is satisfied for the sub-alignments above.

An upper bound cost function for misalignment costs:

Sometimes we do not need to compute the exact costs/fitness but an interval in which the precise values is contained suffices, this is because we can save a lot of computation time and also we may be not interested in calculating the exact scores. In the Section 3.2.2 we defined a lower bound cost function, in the paper [3] the authors introduced a new cost function that can be easily showed to bound from above the true cost function (also here we use the usual notations):

$$cost_D(\sigma_L, SN, \delta) = \begin{cases} \sum_{i \in \{1, \cdots, n\}} \delta_D(\lambda(\sigma_L^i, SN^i)), & \text{if under t.b.a;} \\ move_M(SN) + move_L(\sigma_L), & \text{otherwise.} \end{cases}$$

3.2.4 Decomposition Algorithms

As we have been assuming to have a valid decomposition as an hypothesis all times, now we briefly show two notable Petri net decomposition techniques which return a valid decomposition set given a Petri net in input.

Maximal decomposition:

The idea of this decomposition strategy is quite easy-to-understand but the actual algorithm implementation may result tricky at first sight, refer to [4] for a detailed explanation of this method.

The objective of this technique is to *find subnets whose "size" is as small as possible*. The algorithm begins supposing that there are no isolated places, but even they may appear in some cases they can be removed, since, using a simple theoretical trick, they do not influence the final conformance checking result. Recalling that for a valid decomposition each edge will end up in precisely one subnet, the construction of the maximal decomposition is based on partitioning the edges which "connect" via undirected path involving as few as possible transitions/places, paying attention to the invisible transitions which must not be shared by multiple subnets. To satisfy the requirement that visible transitions with the same label must reside in one subnet, after partitioning the edges into sets of connected edges, the sets that share non-unique observable activities are merged.

In [4] there is proved that *maximal decomposition is valid*, moreover, a straightforward algorithm to construct a maximal decomposition is quadratic in the number of edges.
For an example of maximal decomposition see Fig. 3.7.

Signle-Entry Single-Exit (SESE) decomposition:

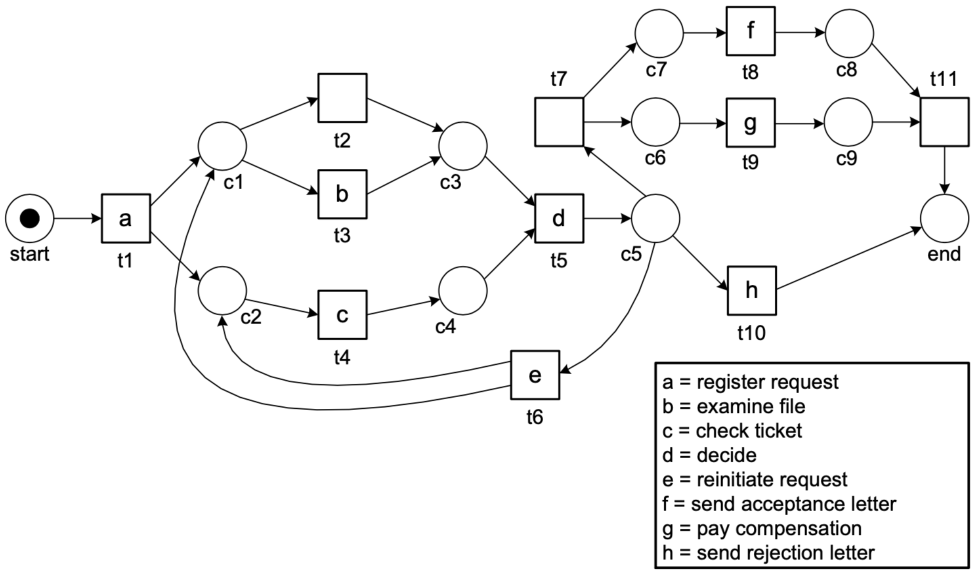
Another well-studied decomposition technique is based on *Refined Process Structure Tree (RPST)* [14], that is a hierarchical structure containing all single-entry and single-exit subnets of a model.

The starting point for building a SESE decomposition is to consider a Petri net model as a graph with only nodes, without making distinction between places and transitions (*workflow graph*).Afterwards, *a SESE of a graph* (V, E) *is a set of edges* $S \in E$ *such that the sub-graph induced by* S *has exactly two boundary nodes: one entry and one exit*. As we have just mentioned, a RPST is created on the model previously to a SESE decomposition because recent developments considerably reduced its implementation effort [15]. See the Fig. 3.8 for an example of RPST and an associated SESE decomposition.

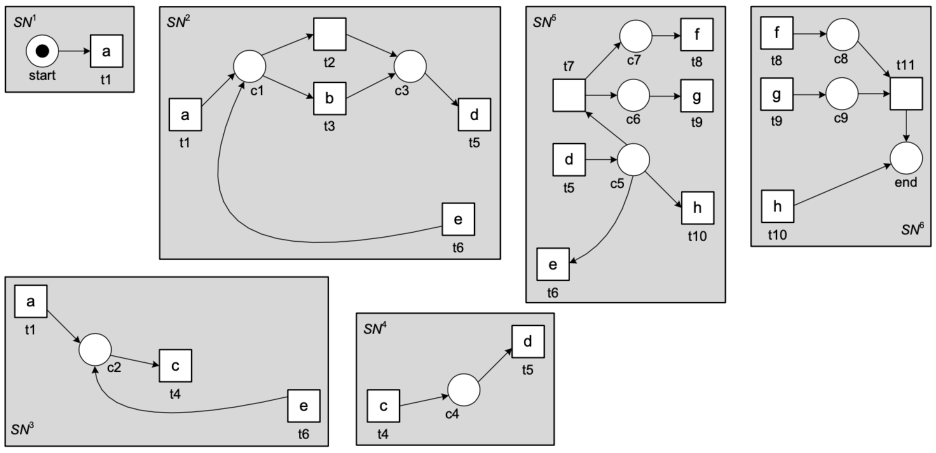
Because we do not make distinction between a place and a transition, it may happen that a place becomes a border node and this is not desirable since it does not obey the definition of a valid decomposition. To overcome this, the concept of *bridge* was introduced. Once a SESE decomposition is computed, the bridging operation consists of

1. removing boundary places together with the arcs connected to them,
2. creating additional subnets for each boundary place, they are made up of all the transitions connected with the boundary place along with the border place.

For instance, the initial SESE decomposition in Fig. 3.9b is not valid since the place p is on the borders. Then, the place is removed from the subnets S_1 and S_2 along with the arcs

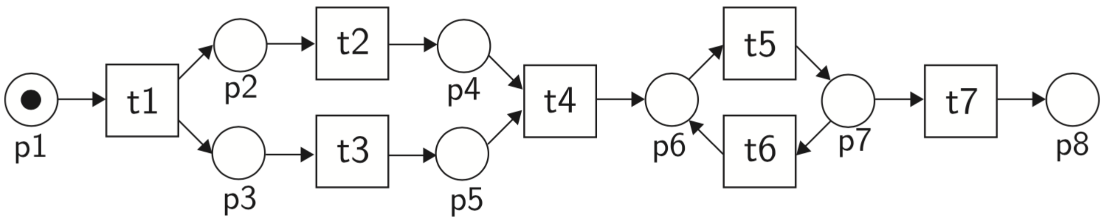


(a) The initial Petri net.

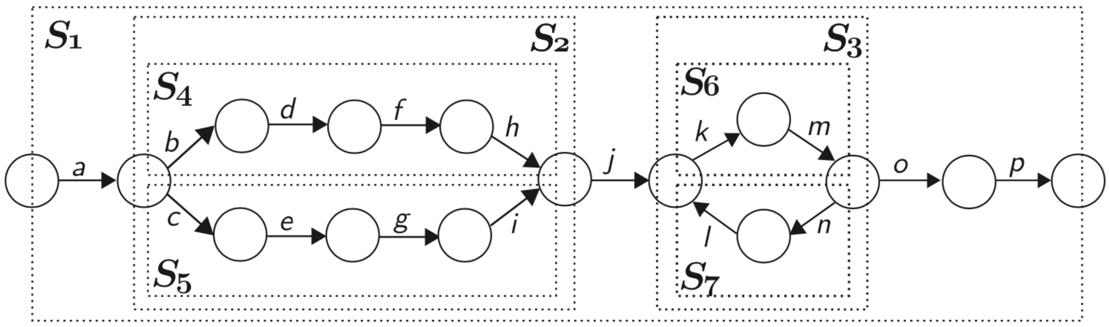


(b) The maximal decomposition of the Petri net.

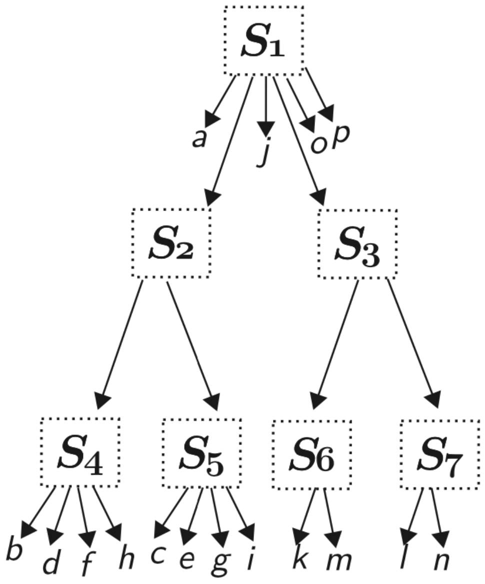
Figure 3.7: An example of maximal decomposition. Source: [4]



(a) A Petri net to be decomposed.



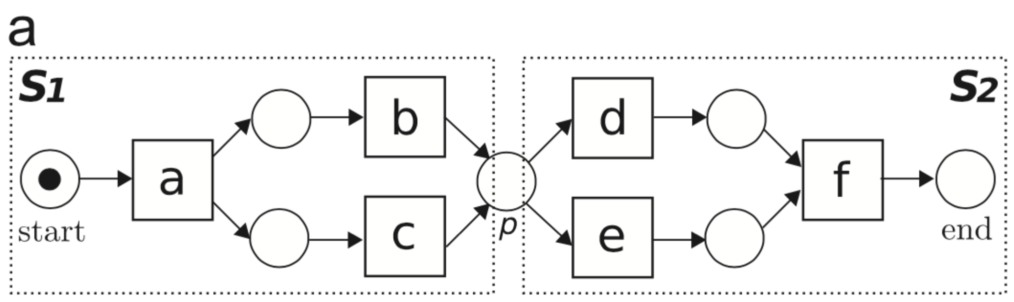
(b) Workflow graph associated with the Petri net.



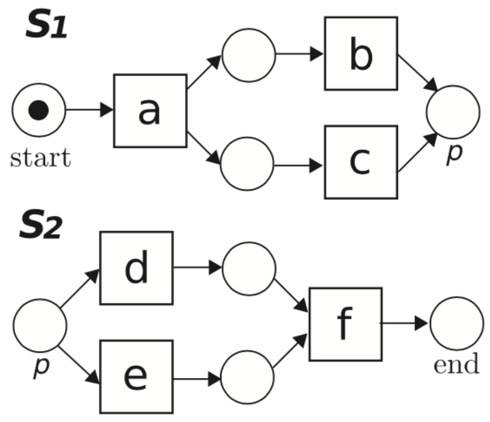
(c) RPST.

Figure 3.8: An illustration of RPST based SESE decomposition: from the transformation in workflow graph (without distinction between the places and transitions) to the computation of RPST. Source: [5].

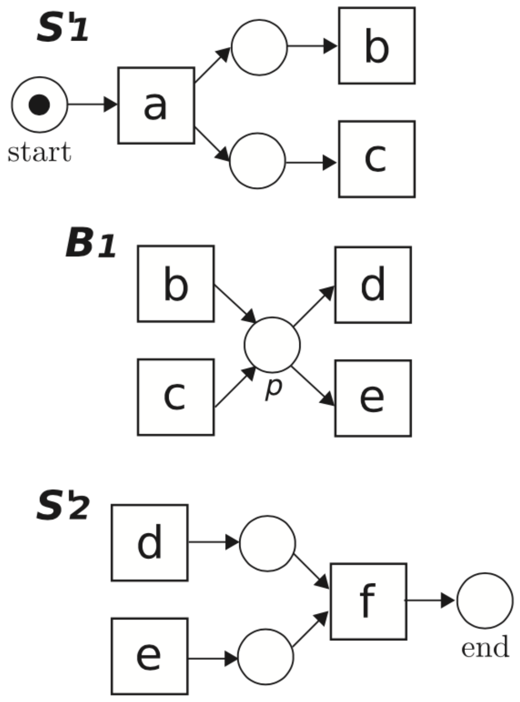
linked to it, after that, there is created a new bridging subnet B_1 which contains the place p and all transitions connected with p as has been shown in Fig. 3.9c. For detailed analysis on this decomposition approach refer to [5].



(a) The initial Petri net with a SESE decomposition.



(b) A SESE decompostion.



(c) A SESE decomposition with bridging.

Figure 3.9: An illustration of the bridging operation: a new bridging subnet B_1 is created and the final decomposition is valid. Source: [5]

4 Stepwise Stitching Decomposed Replay

The promising *recomposing conformance checking* [3] and *methods based on pseudo-alignment* [6] work very well in most of cases: they can actually accelerate models evaluation and make conformance checking possible where the traditional approaches fail. However the existing techniques have some pitfalls as discussed in Section 4.1. This chapter discusses a novel *sub-alignments merging algorithm* that aims to be a good alternative to the other decomposed conformance checking algorithms.

4.1 Stepwise Sub-alignments Stitching Algorithm

Recomposing conformance checking and pseudo-alignments based methods have both advantages and some limitations. The latter approach is very fast but obviously, the final results would be not accurate—when the total border agreement property is not satisfied the merged alignments may not be replayed in the model. About the recomposing approach the final alignments are true alignments, nevertheless this method has one major drawback that is for complex models (e.g. those with many back-loops or loops) the algorithm requires *several recomposing iterations* to reach the total border agreement condition, the consequence is that computation time sometimes could be long due to too many recomposing steps need to be pass through and also recomposed subnets may be as larger as the original model. Finally, when processing one trace in the log, if the number of border conflicts surpasses a prefixed threshold then all sub-alignments are immediately merged and the final result is a pseudo-alignment.

In this section we present our contribution in creating a new sub-alignments merging algorithm that is *single-iteration (without recomposing the subnets)* procedure. It is in the middle between the aforementioned approaches. In other words, *instead of immediately returning a pseudo-alignment when encountering border conflict issues, our algorithm tries to continue without recomposing by stitching problematic subnets, solving one conflict step by step*.

4.1.1 Algorithm

We learnt that total border agreement is required for exact fitness calculation (see Theorem 3.13), in [3] the authors presented the idea of recomposing the subnets which do not agree on a border move. Our objective is to create an algorithm that does not require multiple recomposing and restart iterations in order to possibly reduce the computation time.

Thus, we still rely on the concept of total border agreement *but instead of recomposing we try to keep the first net decomposition and find out sub-alignments for which such condition is satisfied*. In other words, for each subnets group in which there is at least one conflictual border move we strive to search *next-best sub-alignments*, in terms of misalignment costs, for which all border moves will agree. In doing so we have two problems:

1. *sub-alignments search space is generally big (we know that in some cases they are infinite legal alignments) and,*
2. *it is difficult to find out no conflictual sub-alignments for all border activities involved at once, namely, if we only focus on one border move issue separately then solving that conflict we may create a new one.*

Our solution for these problems is to solve the conflicts one at a time but *when solving a conflict we "freeze" the other border moves so that we will not create new border disagreement*. To put it in another way, when we do the problematic sub-alignments adjustment we carefully choose next-best alignments by only modifying non-border transition moves and the border transition we are analysing in that moment. Practically, given a wrong border move, each time we fix one sub-alignment with the wrong move m and try to adjust the others by setting m as the best move. Nevertheless, in order to decide which is the best border move for each border conflict *we heuristically select the one which is less "distant" from the optimal sub-alignment/sub-alignment with minimum delta costs*.

Let SN be a system net, L be an event log and $D = \{SN^1, \cdots, SN^n\} \in \mathcal{D}(SN)$ be a valid decomposition. Let δ be a cost function.

Delta costs: The *delta costs* between a trace alignment γ and other one $\bar{\gamma}$ is the difference between costs of $\bar{\gamma}$ and costs of γ, this is denoted by $\Delta_\delta(\gamma, \bar{\gamma}) := \delta(\bar{\gamma}) - \delta(\gamma)$. We also define delta costs between an alignment γ and an empty alignment to be infinite, i.e. $\Delta_\delta(\gamma, \emptyset) = +\infty$.

Next-best alignment with border move constraints (case of two subnets): Given two subnets SN^i and SN^j be two subnets of SN under the valid decomposition D, let γ^i, γ^j be alignments of SN^i and SN^j respectively. Suppose that there is a border conflict between the two alignments associated with an activity a, let m^i, m^j be legal moves associated with a in γ^i and γ^j respectively and $next_\delta(\gamma^i)$ be a *next-best sub-alignment* of SN^i w.r.t γ^i, in terms of misalignment costs and in increasing costs order, similarly for SN^j. We define *next-best sub-alignments with border constraints* $next_\delta(\gamma^i, \gamma^j; a)$ as a pair $(\bar{\gamma}^i, \bar{\gamma}^j)$ such that

- *If* $\Delta_\delta(\gamma^i, next_\delta(\gamma^i)) \neq +\infty$ *or* $\Delta_\delta(\gamma^j, next_\delta(\gamma^j)) \neq +\infty$:

  1. border moves not involving activity a are the same as in γ^i and γ^j,
  2. let $\bar{m}^i, \bar{m}^j$ be legal moves associated with a in $\bar{\gamma}^i$ and $\bar{\gamma}^j$ respectively, $\bar{m} := \bar{m}^i = \bar{m}^j$ and $\Delta_\delta(\gamma^i, next_\delta(\gamma^i)) \leq \Delta_\delta(\gamma^j, next_\delta(\gamma^j))$ if, and only if, the activity a has a legal move $\bar{m}$ in $next_\delta(\gamma^i)$, or equivalently, $\Delta_\delta(\gamma^j, next_\delta(\gamma^j)) \leq \Delta_\delta(\gamma^i, next_\delta(\gamma^i))$ if, and only if, the activity a has a legal move $\bar{m}$ in $next_\delta(\gamma^j)$.

- *Otherwise:* $next_\delta(\gamma^i, \gamma^j; a) = \emptyset$.

Next-best alignment with border move constraints (case of multiple subnets): This is a general case and we use the similar notations as before, moreover, we define $C \subset \{1, \cdots, n\}$ as the set of indices for which the referred subnets are in conflict caused by activity a. We define *next-best sub-alignments with border constraints* $next_{D,\delta}(C; a)$ as a set $(\bar{\gamma}^i)_{i \in C}$ such that

- *If there exists* $z \in C$ *such that* $\Delta_\delta(\gamma^z, next_\delta(\gamma^z)) \neq +\infty$*:*

  1. border moves not involving the activity a are the same as in $(\bar{\gamma}^i)_{i \in C}$,

2. let $(\bar{m}^i)_{i\in C}$ be legal moves associated with a in $(\bar{\gamma}^i)_{i\in C}$ respectively, $\bar{m} := \bar{m}^i = \bar{m}^j$ for all $i, j \in C$ and $\Delta_\delta(\gamma^i, next_\delta(\gamma^i)) \leq \max_{j\in C, j\neq i} \Delta_\delta(\gamma^j, next_\delta(\gamma^j))$ if, and only if, the activity a has a legal move $\bar{m}$ in $next_\delta(\gamma^i)$ for all $i \in C$, or equivalently, $\Delta_\delta(\gamma^j, next_\delta(\gamma^j)) \leq \max_{i\in C, i\neq j} \Delta_\delta(\gamma^i, next_\delta(\gamma^i))$ if, and only if, the activity a has a legal move $\bar{m}$ in $next_\delta(\gamma^j)$ for all $j \in C$.

- *Otherwise:* $next_{D,\delta}(C; a) = \emptyset$.

Note that in this *min-max case* we again heuristically decided to pick the border move type which shifts as less as possible the previous sub-alignments with lower costs.

Due to *the iterative nature of our algorithm* we call it **Stepwise Stitching Decomposed Replay** algorithm. We give the pseudo-code of the stitching algorithm for a single trace in Algorithm 1. The algorithm extension to the overall log is straightforward.

Algorithm 1 "Stepwise Stitching Decomposed Replay" algorithm for a single trace in the event log

```
Input: System net $SN$, a trace/case $\sigma_L \in L$, a valid decomposition $D = \{SN^1, \cdots, SN^n\} \in \mathcal{D}(SN)$ and a cost function $\delta$.
Output: A pseudo-alignment $\gamma$ between the trace $\sigma_L$ and the model $SN$
  A. Compute $\Gamma := (\gamma^i)_{i\in\{1,\cdots,n\}}$ optimal alignments between the subnets and the projected trace $(\sigma_L^i)_{i\in D}$.

  Let $\Lambda = \{a \in A_b(D) |$ there exist subnets containing a such that there is no between them in the border activity $a$ $(SN_b(a) \neq \emptyset)\}$.
  B.
  for $a \in \Lambda$ do
      Let $I(a) \subseteq \{1, \cdots, n\}$ be the set of indices referring to $SN_b(a)$.
      if $next_{D,\delta}(I(a); a) \neq \emptyset$ then
          Update $\Gamma$ according to $next_{D,\delta}(I(a); a)$.
      else
          break.
      end if
  end for

  C. $\gamma \leftarrow$ merge the sub-alignments in $\Gamma$ using the pseudo-alignments stitching rules in [6].
  return $\gamma$.
```

Correctness: The algorithm is certainly correct according to the output requirement (pseudo-alignment), because during each iteration in the step **B** we solve one border conflict without affecting others border moves, after this step sub-alignments in Γ may satisfy the condition of total border agreement or not, nevertheless, they can be always merged using the pseudo-alignments stitching rules presented in [6] and the result is either an alignment or a pseudo-alignment.

Termination: Both step **A** and step **C** are of course finite, for the first step we can apply already existing conformance checking algorithms. Regarding the step **B**, Γ is finite and the calculation of next-best sub-alignments is always possible, the worst-case is the empty set. We can conclude that our algorithm can be terminated for any input data since all its steps are finite.

Important remarks:

- Thanks to the idea of next-best sub-alignments as we previously defined, the step **B** should have a restricted alignments research space since we freeze border activities not involved during a comparison. *This a key difference between our new algorithm and a naive one* in which we could simply compute all possible sub-alignments combinations, of course setting certain search bounds on the conformance checker, and try to merge them, this very naive method is stopped when a merging operation succeeds.

  Always about the step **B** when iterating on the problematic border activities Γ we can arbitrary order them, the final outcome will be the same if next-best sub-alignments are not empty, otherwise, we would obtain a different pseudo-alignment according to the order we have chosen at the beginning.

- We recall that Theorem. 3.8 claims that a trace is a perfect fitting trace if and only if validly decomposed traces are perfect fitting traces for respective subnets. Since in a perfect fitting trace scenario all moves, either border moves or not, are sync moves this implies that the total border agreement property is satisfied, thus, in this case *our algorithm correctly returns optimal alignments of all perfect fitting traces*.

- We have already presented in Section 3.2.3 a misalignment costs upper bound for a single trace/case σ_L which are simply the costs of a worst-case alignment $(move_M(SN) + move_L(L))$. Applying the adapted cost function for decomposed conformance checking, if one of the termination conditions of the recomposing replay is reached then a fitness interval is returned (see Section 5.2). Since in our algorithm only pseudo-alignments or valid alignments are guaranteed (the optimality is not) then *when calculating fitness intervals the upper bound values are less or equal than the ones we would get by using the recomposing approach*, fortunately lower bounds will be always correctly returned. This is not a so big flop because, for example, if a log fitness is between $[0.90, 0.95]$ and our algorithm returns an interval that is $[0.90, 0.93]$, in addition to the indication of misalignments (pseudo-alignments), this is useful enough to identify process bottlenecks and fix them.

4.2 Running Examples

In this section we present some running examples that hopefully help to better grasp the main ideas of the new sub-alignments merging algorithm.

4.2.1 Example 1

The system net and a its valid decomposition are illustrated in Fig. 4.1. We have also the following input data:

$$\text{Trace: } \langle a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8 \rangle,$$
$$\text{Cost function: } \delta(\text{model move}) = 3, \delta(\text{log move}) = 5.$$

The following is an optimal alignment among the system net and the given trace:

$$\gamma_1 : \begin{array}{c|c|c|c|c|c|c|c|c|c|c} a_1 & \tau & a_2 & a_3 & a_4 & \tau & a_5 & a_6 & \tau & a_7 & a_8 \\ \hline t_1 & t_2 & \gg & t_4 & t_5 & t_6 & t_7 & \gg & t_9 & \gg & t_{11} \end{array}, \qquad \delta(\gamma_1) = 15.$$

Step A:

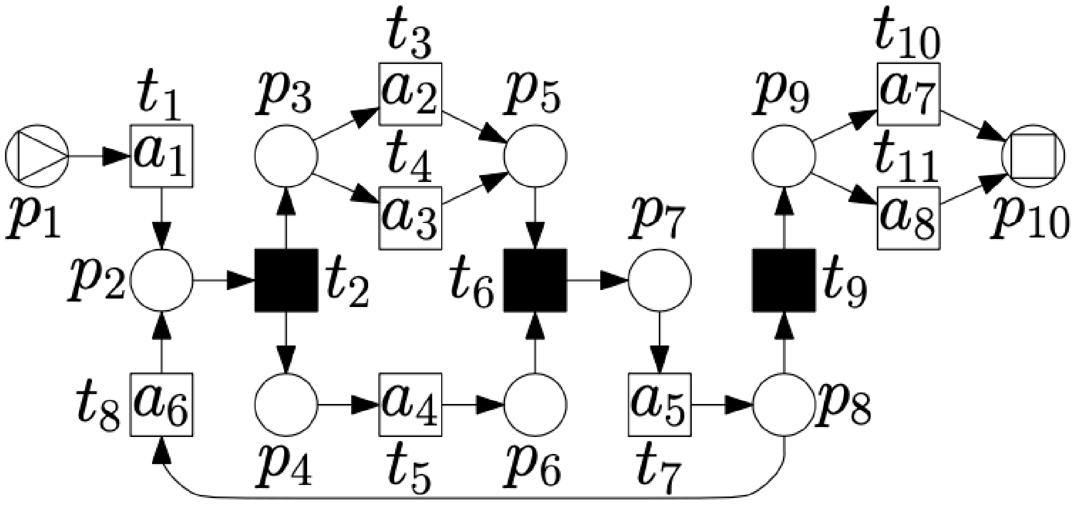
Then, we have the following optimal sub-alignments:

$$\gamma_1^a : \begin{array}{|c|} a_1 \\ \hline t_1 \end{array}, \qquad \gamma_1^b : \begin{array}{c|c|c|c|c|c} a_1 & \tau & a_2 & a_3 & a_4 & a_6 \\ \hline t_1 & t_2 & t_3 & \gg & t_5 & \gg \end{array},$$

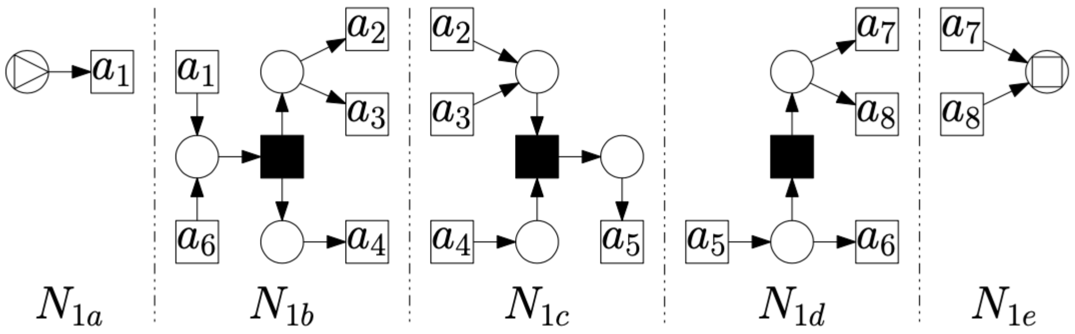
$$\gamma_1^c : \begin{array}{c|c|c|c|c} a_2 & a_3 & a_4 & \tau & a_5 \\ \hline \gg & t_4 & t_5 & t_6 & t_7 \end{array}, \qquad \gamma_1^d : \begin{array}{c|c|c|c|c|c} a_5 & a_6 & \gg & \tau & a_7 & a_8 \\ \hline t_7 & t_8 & t_7 & t_9 & t_{10} & \gg \end{array},$$

$$\gamma_1^e : \begin{array}{c|c} a_7 & a_8 \\ \hline t_{10} & \gg \end{array}.$$

The sub-alignments costs are: $\delta(\gamma_1^a) = 0$, $\delta(\gamma_1^b) = 10$, $\delta(\gamma_1^c) = 5$, $\delta(\gamma_1^d) = 8$, $\delta(\gamma_1^e) = 5$. We then colored in green the border move conflicts.



(a) The system net SN_1.



(b) A valid decomposition of SN_1

Figure 4.1: The system net SN_1 and it is a valid decomposition. Source: [6].

Step B:

Solving the conflict in the activity a_2:
Fixing the border move as in N_{1b} $(a_2|t_3)$:

$$\bar{\gamma}_1^c : \begin{array}{c|c|c|c|c|c|c|c} a_2 & a_3 & a_4 & \tau & a_5 & \gg & \gg & \gg \\ \hline t_3 & t_4 & t_5 & t_6 & t_7 & t_5 & t_6 & t_7 \end{array}, \quad \delta(\bar{\gamma}_1^c) = 9, \quad \Delta_\delta(\gamma_1^c, \bar{\gamma}_1^c) = 4.$$

Fixing the border move as in N_{1c} $(a_2|\gg)$:

$$\bar{\gamma}_1^b : \begin{array}{c|c|c|c|c|c|c} a_1 & \tau & a_2 & \gg & a_3 & a_4 & a_6 \\ \hline t_1 & t_2 & \gg & t_3 & \gg & t_5 & \gg \end{array}, \quad \delta(\bar{\gamma}_1^b) = 18, \quad \Delta_\delta(\gamma_1^b, \bar{\gamma}_1^b) = 8.$$

Since $\Delta_\delta(\gamma_1^b, \bar{\gamma}_1^b) > \Delta_\delta(\gamma_1^c, \bar{\gamma}_1^c)$ then update the sub-alignment γ_1^c.

Solving the conflict in the activity a_3:
Fixing the border move as in N_{1b} $(a_3|\gg)$:

$$\bar{\gamma}_1^c : \begin{array}{c|c|c|c|c} a_2 & a_3 & a_4 & \tau & a_5 \\ \hline t_3 & \gg & t_5 & t_6 & t_7 \end{array}, \quad \delta(\bar{\gamma}_1^c) = 5, \quad \Delta_\delta(\gamma_1^c, \bar{\gamma}_1^c) = -4.$$

Fixing the border move as in N_{1c} $(a_3|t_4)$:

$$\bar{\gamma}_1^b : \begin{array}{c|c|c|c|c|c|c|c} a_1 & \tau & a_2 & \gg & a_3 & a_4 & a_6 & \gg \\ \hline t_1 & t_2 & t_3 & t_1 & t_4 & t_5 & \gg & t_5 \end{array}, \quad \delta(\bar{\gamma}_1^b) = 11, \quad \Delta_\delta(\gamma_1^b, \bar{\gamma}_1^b) = 1.$$

Since $\Delta_\delta(\gamma_1^b, \bar{\gamma}_1^b) > \Delta_\delta(\gamma_1^c, \bar{\gamma}_1^c)$ then update the sub-alignment γ_1^c.

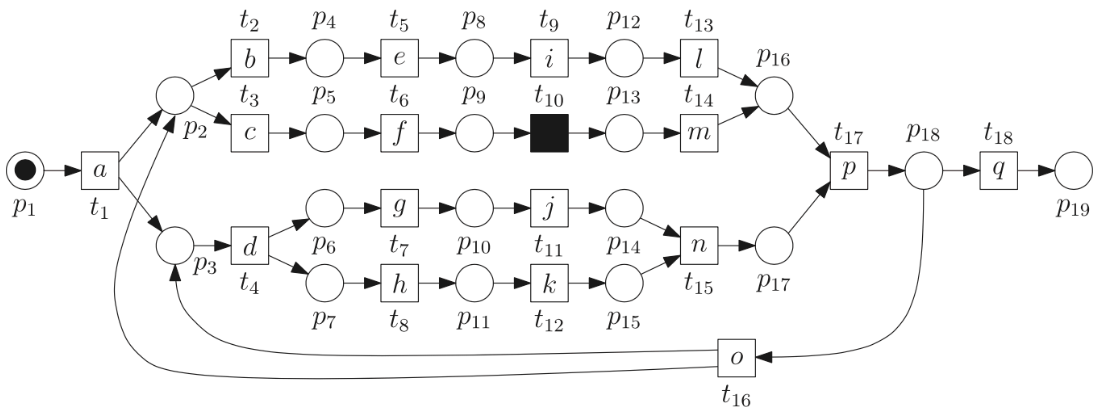
Solving the conflict in the activity a_6:
Fixing the border move as in N_{1b} $(a_6|\gg)$:

$$\bar{\gamma}_1^d : \begin{array}{c|c|c|c|c} a_5 & a_6 & \tau & a_7 & a_8 \\ \hline t_7 & \gg & t_9 & t_{10} & \gg \end{array}, \quad \delta(\bar{\gamma}_1^d) = 10, \quad \Delta_\delta(\gamma_1^d, \bar{\gamma}_1^d) = -2.$$

Fixing the border move as in N_{1d} $(a_6|t_8)$:

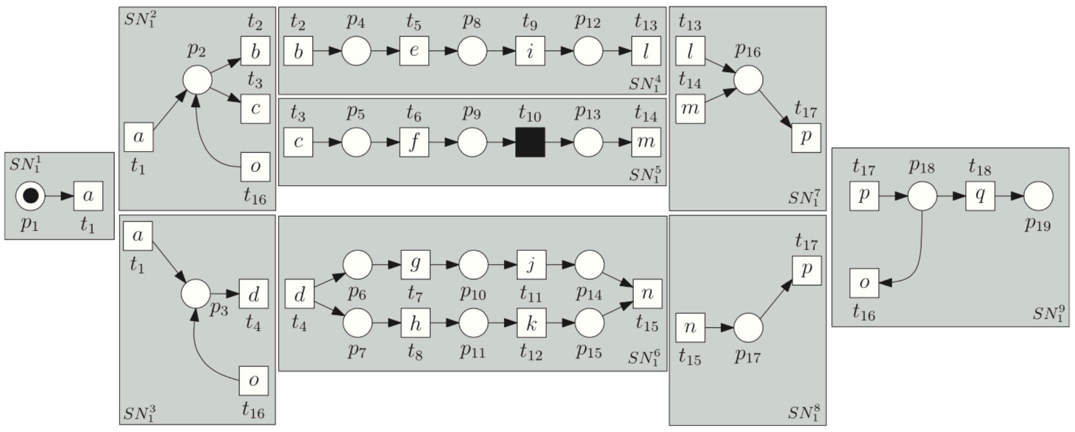
$$\bar{\gamma}_1^b : \begin{array}{c|c|c|c|c|c|c|c|c} a_1 & \tau & a_2 & a_3 & a_4 & a_6 & \gg & \gg & \gg \\ \hline t_1 & t_2 & t_3 & \gg & t_5 & t_8 & t_2 & t_3 & t_5 \end{array}, \quad \delta(\bar{\gamma}_1^b) = 14, \quad \Delta_\delta(\gamma_1^b, \bar{\gamma}_1^b) = 4.$$

Since $\Delta_\delta(\gamma_1^b, \bar{\gamma}_1^b) > \Delta_\delta(\gamma_1^d, \bar{\gamma}_1^d)$ then update the sub-alignment γ_1^d.



t_2 b p_4 t_5 e p_8 t_9 i p_{12} t_{13} l p_{16}
t_3 c p_5 t_6 f p_9 t_{10} p_{13} t_{14} m
p_1 t_1 a p_2 p_3
t_{17} p p_{18} t_{18} q p_{19}
t_4 d p_6 t_7 g p_{10} t_{11} j p_{14} n t_{15} p_{17}
p_7 t_8 h p_{11} t_{12} k p_{15}
o t_{16}

(a) The system net SN_2.



SN_1^1 SN_1^2 SN_1^3 SN_1^4 SN_1^5 SN_1^6 SN_1^7 SN_1^8 SN_1^9

(b) A valid decomposition of SN_2

Figure 4.2: The system net SN_2 and it is a valid decomposition. Source: [3].

Step C:

Now we can stitch all sub-alignments to obtain the following optimal alignment:

$$\bar{\gamma}_1 : \frac{a_1 \mid \tau \mid a_2 \mid a_3 \mid a_4 \mid \tau \mid a_5 \mid a_6 \mid \tau \mid a_7 \mid a_8}{t_1 \mid t_2 \mid t_3 \mid \gg \mid t_5 \mid t_6 \mid t_7 \mid \gg \mid t_9 \mid t_{10} \mid \gg}, \qquad \delta(\bar{\gamma}_1) = 15.$$

Note that this is a different optimal alignment w.r.t. γ_1.

4.2.2 Example 2

The system net and a its valid decomposition are illustrated in Fig. 4.2. We have also the following input data:

Trace: $\langle a, b, e, i, l, d, g, h, n, j, k, p, q \rangle$,
Cost function: $\delta(\text{model move}) = 1, \delta(\text{log move}) = 1$.

The following is an optimal alignment between the system net and the given trace:

$$\gamma_2 : \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|c} a & b & e & i & l & d & g & h & n & j & k & \gg & p & q \\ \hline a & b & e & i & l & d & g & h & \gg & j & k & n & p & q \end{array}, \qquad \delta(\gamma_2) = 2.$$

Step A:

Then, we have the following optimal sub-alignments:

$$\gamma_2^1 : \begin{array}{|c|} a \\ \hline a \end{array}, \qquad \gamma_2^2 : \begin{array}{c|c} a & b \\ \hline a & b \end{array}, \qquad \gamma_2^3 : \begin{array}{c|c} a & d \\ \hline a & d \end{array},$$

$$\gamma_2^4 : \begin{array}{c|c|c|c} b & e & i & l \\ \hline b & e & i & l \end{array}, \qquad \gamma_2^5 : \begin{array}{|c|} \\ \hline \\ \end{array}, \qquad \gamma_2^6 : \begin{array}{c|c|c|c|c|c|c} d & g & h & n & j & k & \gg \\ \hline d & g & h & \gg & j & k & n \end{array}$$

$$\gamma_2^7 : \begin{array}{c|c} l & p \\ \hline l & p \end{array}, \qquad \gamma_2^8 : \begin{array}{c|c} n & p \\ \hline n & p \end{array}, \qquad \gamma_2^9 : \begin{array}{c|c} p & q \\ \hline p & q \end{array}.$$

The sub-alignments costs are: $\delta(\gamma_2^1) = 0$, $\delta(\gamma_2^2) = 0$, $\delta(\gamma_2^3) = 0$, $\delta(\gamma_2^4) = 0$, $\delta(\gamma_2^5) = 0$,$\delta(\gamma_2^6) = 2$,$\delta(\gamma_2^7) = 0$,$\delta(\gamma_2^8) = 0$,$\delta(\gamma_2^9) = 0$. We then colored in green the border move conflicts.

Step B:

Solving the conflict in the activity n:
Fixing the border move as in SN_6 ($n| \gg, \gg |n$):

$$\bar{\gamma}_2^8 : \text{impossible}\,, \Delta_\delta(\gamma_2^8, \bar{\gamma}_2^8) = +\infty.$$

Fixing the border move as in SN_8 ($n|n$):

$$\bar{\gamma}_2^6 : \begin{array}{c|c|c|c|c|c|c|c} d & g & h & \gg & \gg & n & j & k \\ \hline d & g & h & j & k & n & \gg & \gg \end{array}, \quad \delta(\bar{\gamma}_2^6) = 4, \quad \Delta_\delta(\gamma_2^6, \bar{\gamma}_2^6) = 2.$$

Since $\Delta_\delta(\gamma_2^8, \bar{\gamma}_2^8) > \Delta_\delta(\gamma_2^6, \bar{\gamma}_2^6)$ then update the sub-alignment γ_2^6.

Step C:

Now we can stitch all sub-alignments to obtain the following optimal alignment:

$$\bar{\gamma}_2 : \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|c|c} a & b & e & i & l & d & g & h & \gg & \gg & n & j & k & p & q \\ \hline a & b & e & i & l & d & g & h & j & k & n & \gg & \gg & p & q \end{array}, \qquad \delta(\bar{\gamma}_2) = 4.$$

In this second example we obtained a true alignment (it can be replayed) but not an optimal one.

4.2.3 Example 3

The system net and a its valid decomposition are illustrated in Fig. 4.3. We have also the following input data:

$$\text{Trace: } \langle a, g, b, d, c, d, f, j, e, h, m \rangle,$$
$$\text{Cost function: } \delta(\text{model move}) = 10, \delta(\text{log move}) = 3.$$

The following is an optimal alignment between the system net and the given trace:

$$\gamma_3 : \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c} a & g & b & d & c & d & f & j & \gg & e & h & \tau & m \\ \hline a & \gg & b & d & \gg & \gg & f & j & k & e & \gg & \tau_1 & m \end{array}, \qquad \delta(\gamma_3) = 22.$$

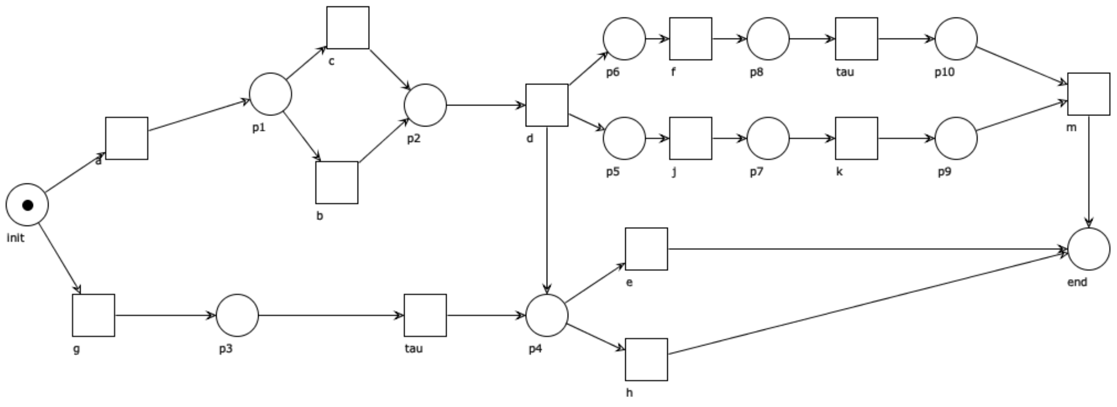
Step A:

Then, we have the following optimal sub-alignments:

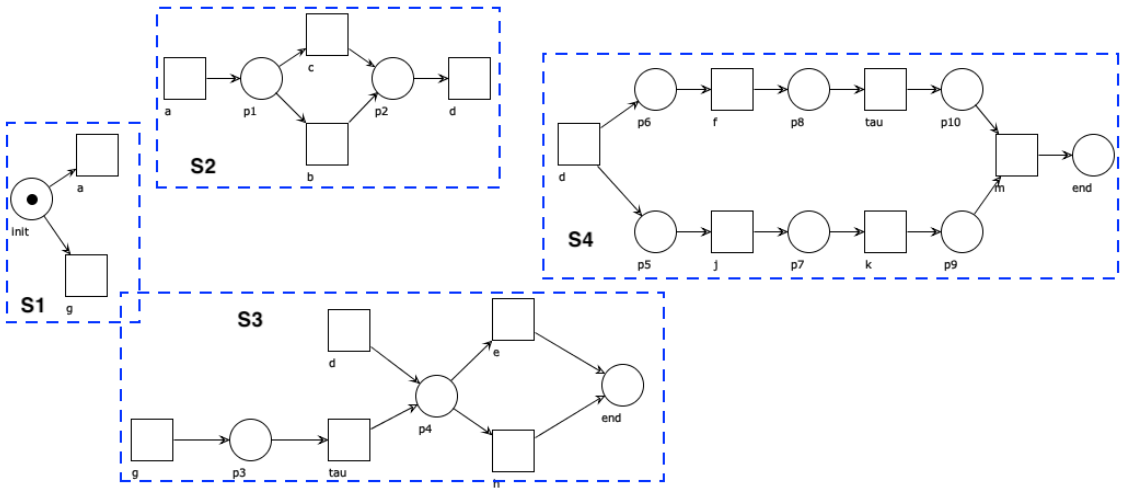
$$\gamma_3^1 : \begin{array}{c|c} a & g \\ \hline a & \gg \end{array}, \qquad \gamma_3^2 : \begin{array}{c|c|c|c|c} a & b & d & c & d \\ \hline a & b & d & \gg & \gg \end{array}$$

$$\gamma_3^3 : \begin{array}{c|c|c|c|c} g & d & d & e & h \\ \hline \gg & d & d & e & h \end{array}, \qquad \gamma_3^4 : \begin{array}{c|c|c|c|c|c|c} d & d & f & \tau & j & \gg & m \\ \hline d & \gg & f & \tau_1 & j & k & m \end{array}.$$

The sub-alignments costs are: $\delta(\gamma_3^1) = 3$, $\delta(\gamma_3^2) = 6$, $\delta(\gamma_3^3) = 3$, $\delta(\gamma_3^4) = 13$. We then colored in green the border move conflicts.



(a) The system net SN_3. There two invisible activities that we denote with τ_1 and τ_2, they reside in S_4 and S_3 respectively.



(b) A valid decomposition of SN_3

Figure 4.3: The system net SN_3 and it is a valid decomposition.

Step B:

Solving the conflict in the activity d:
Fixing the border move as in $SN_2 - SN_4$ $(d|d, d|\gg)$:

$$\bar{\gamma}_3^3 : \begin{array}{c|c|c|c|c} g & d & d & e & h \\ \hline \gg & d & \gg & \gg & h \end{array}, \quad \delta(\bar{\gamma}_3^3) = 9, \quad \Delta_\delta(\gamma_3^3, \bar{\gamma}_3^3) = 6.$$

Fixing the border move as in SN_3 $(d|d, d|d)$:

$$\bar{\gamma}_3^2 : \begin{array}{c|c|c|c|c|c} a & b & d & \gg & c & d \\ \hline a & b & d & a & c & d \end{array}, \quad \delta(\bar{\gamma}_3^2) = 10, \quad \Delta_\delta(\gamma_3^2, \bar{\gamma}_3^2) = 4.$$

$$\bar{\gamma}_3^4 : \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c} d & d & f & \tau & j & \gg & m & \gg & \tau & \gg & \gg & \gg \\ \hline d & d & f & \tau_1 & j & k & m & f & \tau_1 & j & k & m \end{array}, \quad \delta(\bar{\gamma}_3^4) = 50, \quad \Delta_\delta(\gamma_3^4, \bar{\gamma}_3^4) = 37.$$

Since $\Delta_\delta(\gamma_3^3, \bar{\gamma}_3^3) < \max(\Delta_\delta(\gamma_3^2, \bar{\gamma}_3^2), \Delta_\delta(\gamma_3^4, \bar{\gamma}_3^4))$ then update the sub-alignment γ_3^3.

Step C:

Now we can stitch all sub-alignments to obtain the following optimal alignment:

$$\bar{\gamma}_3 : \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c} a & g & b & d & c & d & f & \tau & j & \gg & e & h & m \\ \hline a & \gg & b & d & \gg & \gg & f & \tau_1 & j & k & \gg & h & m \end{array}, \quad \delta(\bar{\gamma}_3) = 22.$$

Also in this third example the final result is satisfactory, we have just calculated an overall optimal alignment.

5 Related Work

For the part of process mining overview we mainly refer to [1] which is an introduction book for anybody who wants to get started in this study field, either for a student or a professional, there are both theoretical results and practical advices. In [8, 2] best practices and challenges are analysed by the authors, most importantly, they defined the core research areas in process mining.

The seminar paper [16] provide basis of conformance checking focusing on the use of model replay to determine the fitness of an event log. In some situations alignment-based conformance checking was proved to be superior to the token-based approach [12]. In [10] the four model quality dimensions (replay fitness, precision, generalization and simplicity) are deeply analysed showing that all four quality dimensions are necessary, the major contribution was to create a configurable algorithm for a weighted average over the four quality dimension, at the end the algorithm is guaranteed to produce sound process models. While traditional conformance checking methods/tools such as [17, 18] only focus on the control-flow perspective, ignoring the other perspectives, such as data, resources and time, a multi-perspective conformance checker is presented in [19].

Discussing about decomposed conformance checking, several approaches have been proposed to decompose conformance checking in the literature [5, 6, 20]. Their experimental results showed an immense reduction in computation time over the existing monolithic approach, but these approaches only remain at the level of sub-logs and subnets, the conformance between the overall process model and event log is not computed. A complete divide et impera conformance checking approach based on the total border agreement [4] and subnets recomposing idea is proposed in [3], given that our work is mainly inspired by this paper and [6], we dedicated the next sections in this chapter for them, to write more about their relevant contributions. Decomposing large graphs into smaller fragments is a topic widely studied in the literature. Previously to the maximal decomposition studied in [4] in [21] it is shown the concept of "passages" which can be used to decompose both process discovery and conformance checking problems. SESE (Single-Entry and Single-Exit) decomposition is analysed in depth and in order to be a valid decomposition the authors proposed the idea of bridging, in which two wrongly decomposed subnets are linked together by an additional artificial subnet. Finally, data-aware process models decomposition is examined in [11] that is a natural extension of decomposition of process models without variables, specifically, the variables are considered as normal places and enriched Petri nets are then decomposed by using SESE approach.

Although in most of cases replaying sub-logs and subnets should be less computation intensive w.r.t. replaying the overall log, in [22] it has been shown there are cases in which the decomposed replay may take way more time. So, in this work it has been proposed an alternative decomposed replay which is faster than the monolithic replay even for the problematic cases.

5.1 Pseudo-alignments

Sometimes there is no need of precise alignment between a trace and the model, also approximation is accepted, so the the authors of paper [6] proposed the concept of *pseudo-alignment* for ease of sub-alignments merging. In our opinion the most important contributions in this work is the creation of two merging rules: *alignment stitching rules and pseudo-alignment stitching rules*. The first rules are applicable when the property of total border agreement is satisfied and final result is a true alignment, whereas the latter ones are more general rules and can be always applied, the outcome is a true alignment if the aforementioned property is satisfied otherwise result is a so-called pseudo-alignment.
Both two stitching rules consist of three cases:

1. when the trace and all decomposed alignments have been dealt with completely;
2. when all relevant decomposed alignments agree on an activity in the trace;
3. when all relevant decomposed alignments agree on a next model move.

The key difference between the two set of rules is that when there is no border agreement then worst-case is preferred:

- in case of activity conflicts, the most ex- pensive of the conflicting legal moves is added to the resulting pseudo alignment;
- in case of model move conflicts, one of these model moves is selected, and added to the pseudo-alignment.

5.2 Recomposing Conformance Checking

When encountering a border conflict, instead of selecting the worst-case move, it is possible to solve the conflict by recomposing the problematic subnets.

In the original paper the authors went beyond a simple subnets recomposing, they proposed an *iterative conformance checking procedure* which is depicted in Fig. 5.1. The primary objective is to have a framework that automatically merge the subnets in conflict and then restart the conformance checking process with the new decomposition, which is still a valid decomposition. Many algorithm *termination conditions* are illustrated and once early terminated the returned result will be a fitness interval which is easy to calculate as we have shown in Section 3.2.3. The termination conditions are as follows:

- All log traces have been either aligned under total border agreement or have been rejected because of the number of border conflicts is over a given threshold;
- Surpassing the overall time threshold;
- Having aligned a target percentage of traces in the log under total border agreement or the overall fitness interval value is narrow enough;
- The maximum number of iterations is reached.

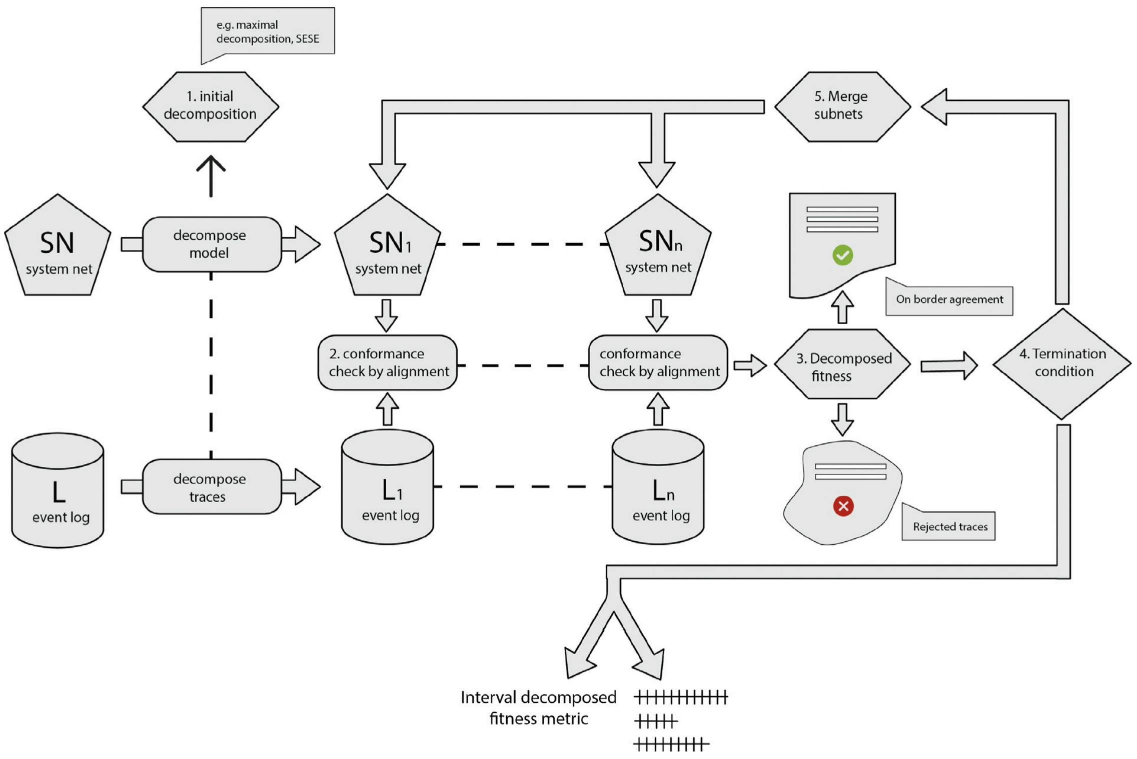


Figure 5.1: Iterative recomposing conformance checking framework. Once terminated the return value is either the exact or interval decomposed conformance metric. Source: [3].

6
Process Mining Tools

In this chapter we present tools to work with event data and to implement process mining algorithms, because we believe that theoretical results are also driven by empirical observations. We illustrate both a open-source software, which is the one we use for our later experiments, and a promising commercial platform for real-world process mining projects. Furthermore, an overview of an event data standard is provided.

6.1 XES

Any data analysis must start with data. Whereas many data format are available for event data such as simple CSV or XML documents, but until 2010 the de facto standard for storing and exchanging event logs was MXML (*Mining eXtensible Markup Language*). Due to several extensibility limitations, *XES* became the successor of MXML. In September 2010, the format was adopted by the *IEEE Task Force* on Process Mining and turned into the de facto exchange format for process mining and now it aims to be an official IEEE standard. XES is also a *tag-based* language and this standard provides a common XML format for interchange of event data between information systems. See the official web site [*] for more information.

Whole standard can be described through a meta model using UML (Unified Modeling Language) class diagram see the Fig. 6.1. Basically, an XES file contains one even log consisting of

[*] https://xes-standard.org

any number of traces which in turn includes any number of events. The log, its traces, and its events may have a certain number of attributes, nested attributes are allowed. Examples of attributes are *timestamp, resource* and other customized properties. A example of XES file has been already shown in Chapter 2.

XES supports the *classifier* concept [1] which is *a function that maps the attributes of an event onto a label used in the resulting process model,* this can be seen as the "name" of the event, thus two events are considered the identical if they have same classifier. XES allows an arbitrary number of classifier, e.g. `time:timestamp, concept:name` or `org:resource`. Programming framework we will describe later fully support XES standard and this is also the data format for experiments done in this thesis.

6.2 ProM

ProM Tool [†] is the de factor process mining framework in the academic world, it is fully open-source, highly extensible and based on JVM (Java Virtual Machine), this is the tool used in this thesis.

The Fig. 6.2 is an overview of this software and that is very useful to illustrate its main components. Basically we have the following modules:

- **ProM core:** This component includes the most important implementations of ProM, such as some fundamental definitions and algorithms (e.g. event log, Petri net or BPMN model), it helps to abstract those low-level concepts and make the framework more extensible. This ease of development should significantly lower the barriers of entry for new developers. All the other components need to communicate with the core component.

- **Plug-in Manager:** Every algorithm that has to be implemented will become a *plug-in*. As the name suggest, all plug-ins are ready-to-use subprogram which receive in input some objects (e.g. Petri net, event log ect.) and return other objects, they may be a transformation of those in input. The following is a "Hello World" plug-in example which simply print a string on the UI once started:

```
package org.processmining.plugins.gettingstarted;

import org.processmining.contexts.uitopia.annotations.UITopiaVariant;
import org.processmining.framework.plugin.PluginContext;
```

[†]http://www.promtools.org/doku.php

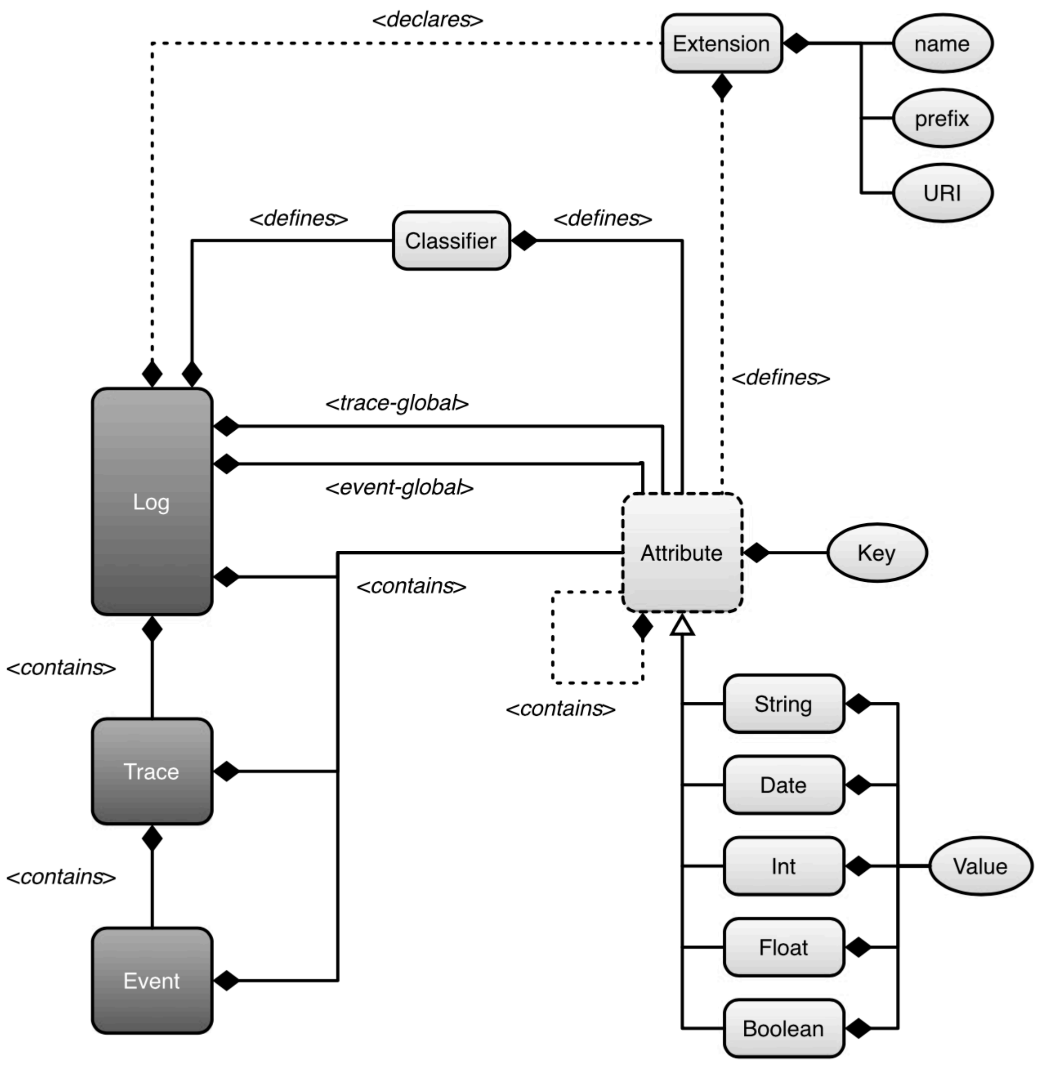


Figure 6.1: Meta model of XES standard. Source: [1]

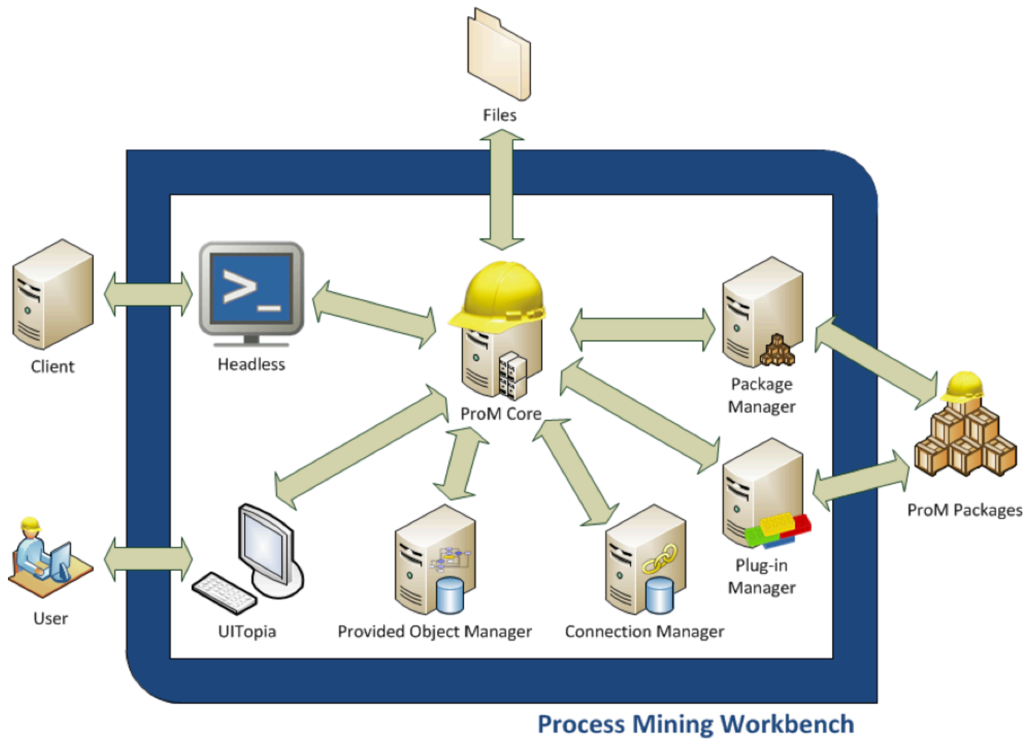


Figure 6.2: The main components of ProM, this modularization makes this framework highly extensible. Note also the separation between the actual algorithms' implement and the results visualization. Interactive user interface is provided by the library UITopia.

```
import org.processmining.framework.plugin.annotations.Plugin;

public class HelloWorld {
        @Plugin(
                name = "My Hello World Plugin",
                parameterLabels = {},
                returnLabels = { "Hello world string" },
                returnTypes = { String.class },
                userAccessible = true,
                help = "Produces the string: 'Hello world'"
        )
        @UITopiaVariant(
                affiliation = "My company",
                author = "My name",
                email = "My e-mail address"
        )
        public static String helloWorld(PluginContext context) {
                return "Hello World";
        }
}
```

Each plug-in is called with a *PluginContext* parameter. The idea of the PluginContext is that it provides all the necessary interfaces to communicate with:

- the framework;
- other plug-ins;
- the user.

Plug-ins can be executed in various contexts, such as a GUI, or a script context.

When the ProM framework is started, a main plug-in context is automatically created. This main plug-in context is used to derive new contexts from in order to execute plug-ins. When a plug-in is executed by the framework, the framework first instantiates a PluginContext object. The implementing type of this context can for example be the GUI context, or a command line context.

- **Package Manager:** The reusability is the main objective for developers, a *package* contains related methods/objects about a certain algorithm to be implemented, for example `org.processmining.decomposedreplayer` is the package in which we can find decomposed conformance checking algorithms. When the number of plug-ins/packages is rapidly increasing then a central package management system is needed and here is where the Package Manager comes into play, in order to avoid conflicts among different packages, also to have a better organization of works done so that developers do not have to write a piece of code twice.

- **Connection Manager:** The connections are also key objects in the framework, they are mapping from a set of labels to objects so that we can connect one plug-in to another by passing the results of the source method to a second algorithm or even to the same one. A example of a call to the Connection Manager is the following:

  ```
  Connection c = new MarkedNetConnection(petrinet,
          new Marking(state));
  context.addConnection(c);
  ```

  Then we can retrieve the object by calling the method:

  ```
  context.getConnectionManager().addConnection(context, c);
  ```

  In the ProM framework relations between objects are stored in connections as well. Adding connections or checking for the existence thereof is again done through the connection manager, which can be accessed always by calling `context.getConnectionManager()`.

- **Provided Object Manager:** In general, as soon as plug-ins are invoked, their results are available as *provided objects*. All objects in the framework are handled by the *provided object manager*, which can be accessed through the context. However, typically plug-ins only need a bit of the available functionality, namely to create and update provided objects. A provided object consists of a label and an object and to create one, the `createProvidedObject` method of `ProvidedObjectManager` should be called. For example:

  ```
  ProvidedObjectID id = context.getProvidedObjectManager()
          .createProvidedObject("Example string", s, context);
  ```

  Once the provided object is created, it gets an ID, which can be used to reference the object later, for example to update, or to delete it. For example:

  ```
  context.getProvidedObjectManager().deleteProvidedObject(id);
  ```

- **UITopia:** Thanks to the *modularization*, ProM 6 radically separates the UI from the internal heavy lifting. *UITopia* is a completely redesigned user interface, now users can browser objects described above within a graphical window, which are hidden from the user before the born of UITopia.

  This brand-new user interface looks as made up with three main pages:

– **Workspace:** Here is where users can select, delete, import and export objects, such as event logs, Petri nets ect. ;
– **Actions:** All installed plug-ins are available in this page. Furthermore, the UI also explicitly shows the dependencies of each plug-in (i.e., what object it requires to execute), and the list of objects that will be created after execution;
– **Visualizer:** Finally, ProM 6 separates actual objects (e.g., a Petri net model) from their visualization (i.e., the image of the Petri net on your screen). This means that, if we are no longer interested in viewing the result of your analysis, you can simply close the visualizer – the actual object will still be in your workspace.

See Fig. 6.3 to have an idea of how this UI appears to the users.

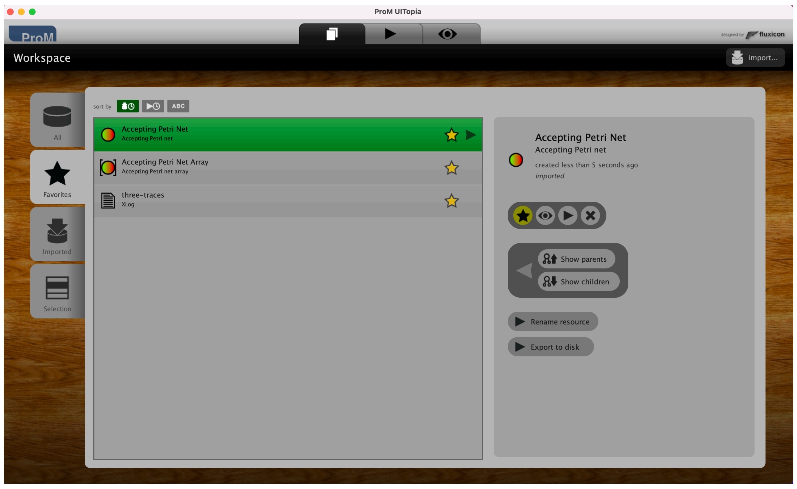
Because this is a open-source project the contribution from developers all over the world is important to keep it alive, so the community has designed a standard *development architecture* that allows programmers to work first in a local mode, through a Java code editor (*Eclipse* is the default choice since it is a integrated development environment (IDE) which offers a lot of useful coding tools/plug-ins for Java developers), then push final code to the official SVN (*Apache Subversion*) repository, this last step needs to be reviewed beforehand by ProM's maintainers. In the Fig. 6.4 all necessary steps are depicted.
For going deep into this framework refers to [23].

6.3 Celonis

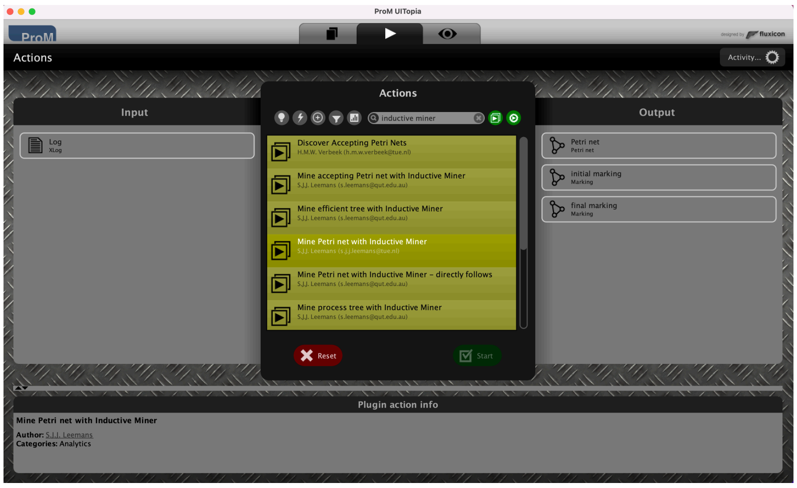
Since process mining as a young research field is gaining more and more attention from scientists and companies due to its enormous potential in industrial applications, it is important to keep eyes on its real-world developments. So, in this section we present a commercial tool for doing process mining in a non-academic setting.

6.3.1 Background

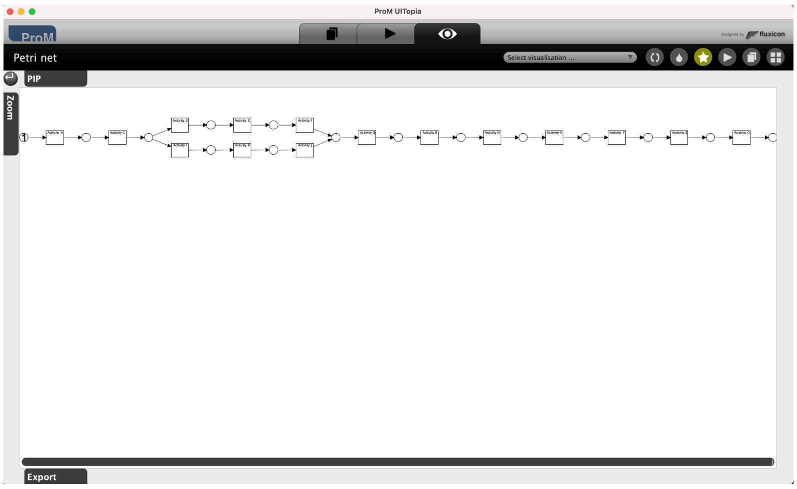
Celonis is a *commercial* process mining software developed in the homonymous German company. It was founded in 2011 by co-founders: *Martin Klenk, Bastian Nominacher, and Alexander Rinke*, in 2012 Celonis reinforced its credentials by becoming a member of the SAP HANA Startup Focus Program organized by SAP–another German tech giant in the field of *ERP (Enterprise Resource Planning)*–with which it has been in close collaboration for years, advertised as a component of SAP's software offering (*Process Mining by SAP*). Then Celonis



(a) Workspace


ProM UITopia
ProM
Actions
Activity...
Input
Log
XLog
Actions
inductive miner
Discover Accepting Petri Nets
H.M.W. Verbeek (h.m.w.verbeek@tue.nl)
Mine accepting Petri net with Inductive Miner
S.J.J. Leemans (s.leemans@qut.edu.au)
Mine efficient tree with Inductive Miner
S.J.J. Leemans (s.leemans@qut.edu.au)
Mine Petri net with Inductive Miner
S.J.J. Leemans (s.j.j.leemans@tue.nl)
Mine Petri net with Inductive Miner - directly follows
S.J.J. Leemans (s.leemans@qut.edu.au)
Mine process tree with Inductive Miner
S.J.J. Leemans (s.leemans@qut.edu.au)
Reset
Start
Output
Petri net
Petri net
initial marking
Marking
final marking
Marking
Plugin action info
Mine Petri net with Inductive Miner
Author: S.J.J. Leemans
Categories: Analytics

(b) Actions



(c) Visualizer

Figure 6.3: The main sections of the UITopia user interface.

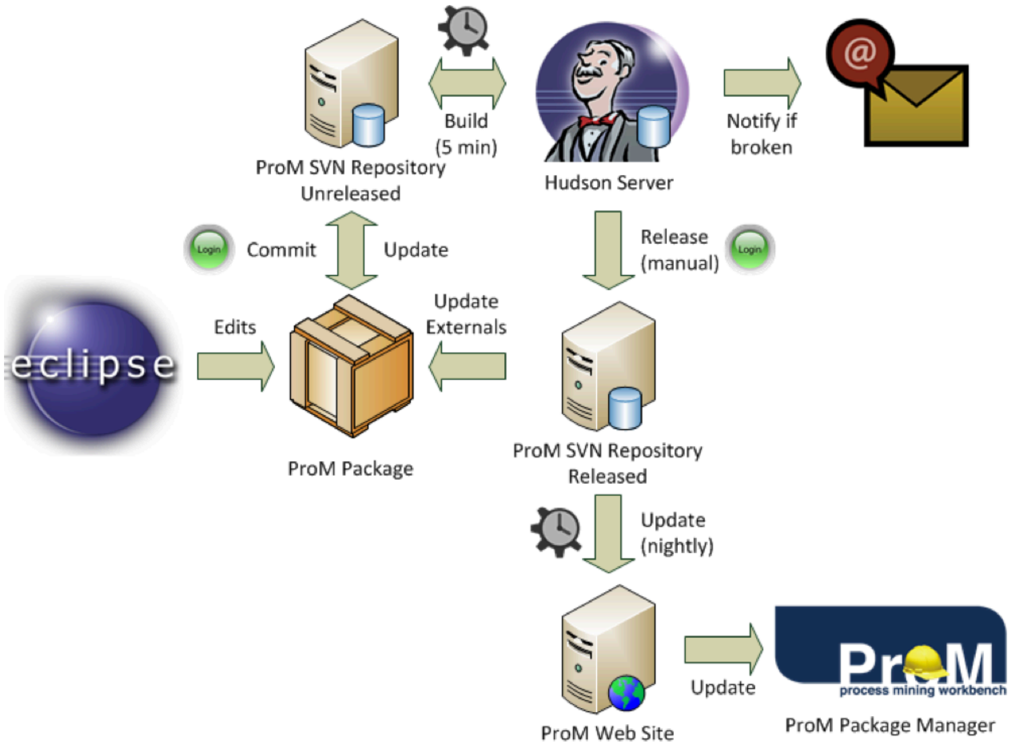


Figure 6.4: ProM packages' development life-cycle: from the local programming to their official release.

as a software product reached enough maturity so that it decided to become an independent platform with a rich process mining functionalities.

It is not so surprising to notice that many of the services that Celonis platform *EMS (Execution Management System)* provides share some similarities with other process mining softwares, especially with the open-source application we talked about in Section 6.2. Of course in a commercial product everything is simplified and made more intuitive for non-technical users, this implies missing functionalities but also better UI.

6.3.2 Platform Architecture

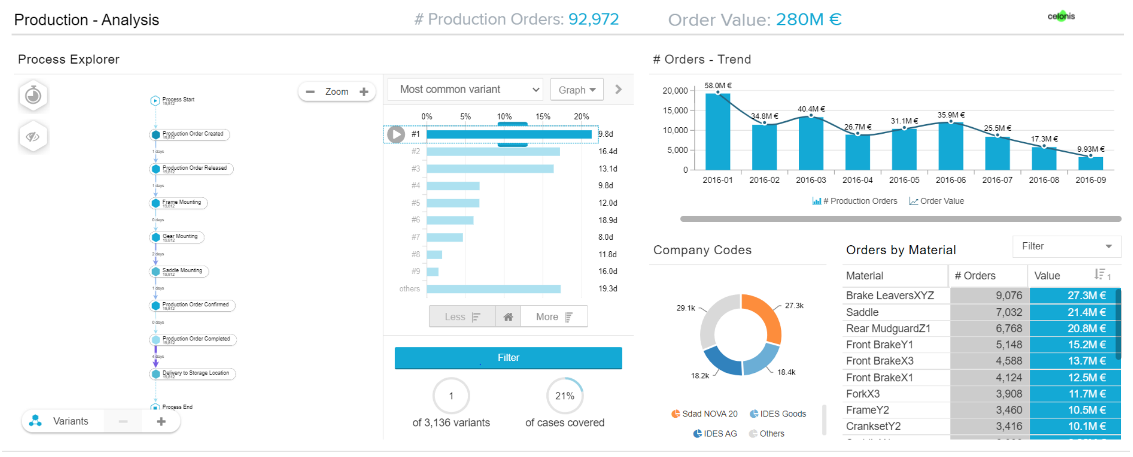
The software can be installed either in a private datacenter (*On-Premises*) or hosted in a *fully managed cloud environment*, the latter option is a *SaaS (Software as a Service)* solution and this implies no hardware infrastructure maintenance, so that is the most common use case and maybe the cheapest one since human resources for infrastructure management is very expensive.

Regarding the high-level platform architecture we can look at Fig. 6.6. Some very powerful features are available in this platform:

- **Real-time data ingestion:** Process data are updated continuously as the processes execute, this means small time lag between bottleneck discovery and taking action, in some business cases timely actions are very valuable;
- **Machine Learning Engine:** Some Celonis' services are also boosted by an internal ML engine that helps to predict users' needs;
- **Ready-to-use applications** since Celonis has been in the market for many years it has created a very complete application suite (it is still in rapid expansion) which ease difficulties for process analysts to almost immediately start to analyse their business case, without complex initialization steps. The Fig. 6.5 depicts a typical user interface for exploring process data, the UI is interactive in sense that many operations are allowed such as filtering or move/adding/modifying graphical widgets.

6.3.3 Core Components

There are two core components which are fundamental in Celonis, they represent also the *key differentiators* which separate this platform from its competitors in the market:



Orders by Material

Material	# Orders	Value
Brake LeaversXYZ	9,076	27.3M €
Saddle	7,032	21.4M €
Rear MudguardZ1	6,768	20.8M €
Front BrakeY1	5,148	15.2M €
Front BrakeX3	4,588	13.7M €
Front BrakeX1	4,124	12.5M €
ForkX3	3,908	11.7M €
FrameY2	3,460	10.5M €
CranksetY2	3,416	10.1M €

Figure 6.5: Celonis Process Analytics application illustration.

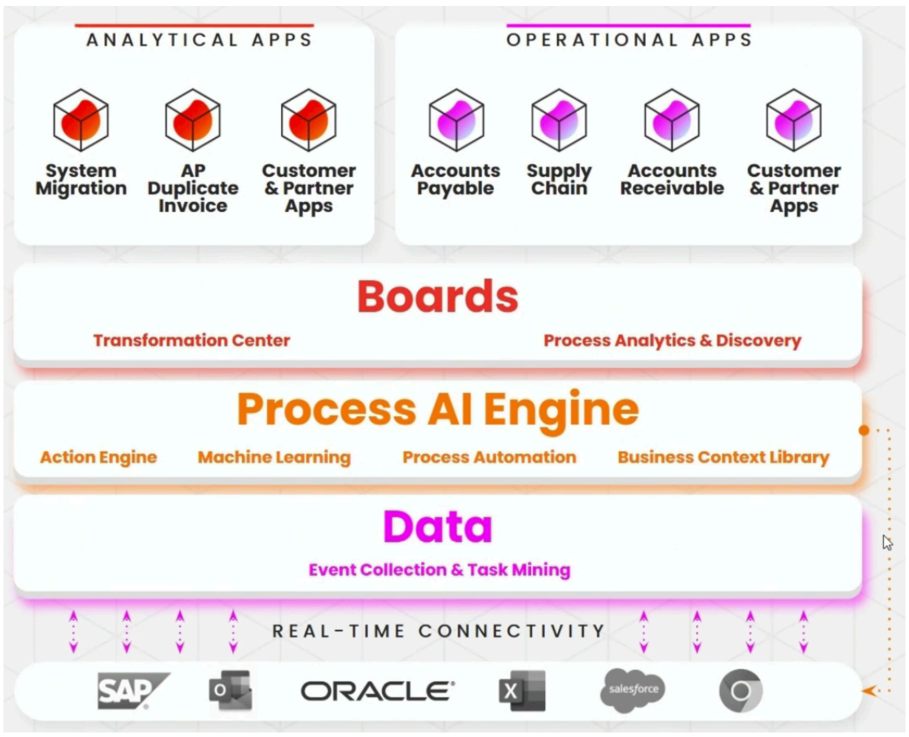


Figure 6.6: Overview of Celonis platform architecture.

Core Mining Engine:

The Core Mining Engine leverages machine learning to analyse both system logs and user interaction data to identify any bottlenecks and provide actionable insights to help resolve them, this a low-level component and it is essential for the entire Celonis platform since it has the implementation of all algorithms and concepts from newest process mining research field. It can be considered the processing engine that uses the knowledge model to determine conformance or non-conformance of the current process compared with the BPMN model (*conformance checking*).

Automation Engine:

The Automation Engine assesses the insights provided by the Core Mining Engine and takes action, efficiently automating workflows across all of the enterprises' systems. Automated tasks or actions are triggered to apply the control policy by sending tasks and instructions to workers that ensure compliance with the policy, automating actions to help workers perform their tasks more efficiently or orchestrating a series of steps to fully automate a task.

7 Experiments

This chapter is related to experiments for verifying how good is the stitching algorithm we presented in Chapter 4. In particular, we aim to test the algorithm giving it in input large process models, as the decomposed conformance checking methods are mostly useful in these cases. We will highlighte some of its advantages over the recomposing approach, but also discuss about its weaknesses.

7.1 Preliminary Steps

7.1.1 Process Data Generator: PLG2

In our experiments we leveraged on *PLG2 (Processes and Logs Generator 2)* [24] for randomly generating process models, then, there is the possibility to simulate them so that we can also create event logs *with noise*. PLG2 can be freely downloaded [*] as a JAR (Java ARchive) package and it runs on a JVM (Java Virtual Machine).

The noise can be introduced at different level, we are only interested in errors at trace level, specifically, we apply the following noise types:

- *Missing head*: the user has to specify the maximum size for a head and the software will randomly choose the actual value between one and the provided value.

[*] https://plg.processmining.it

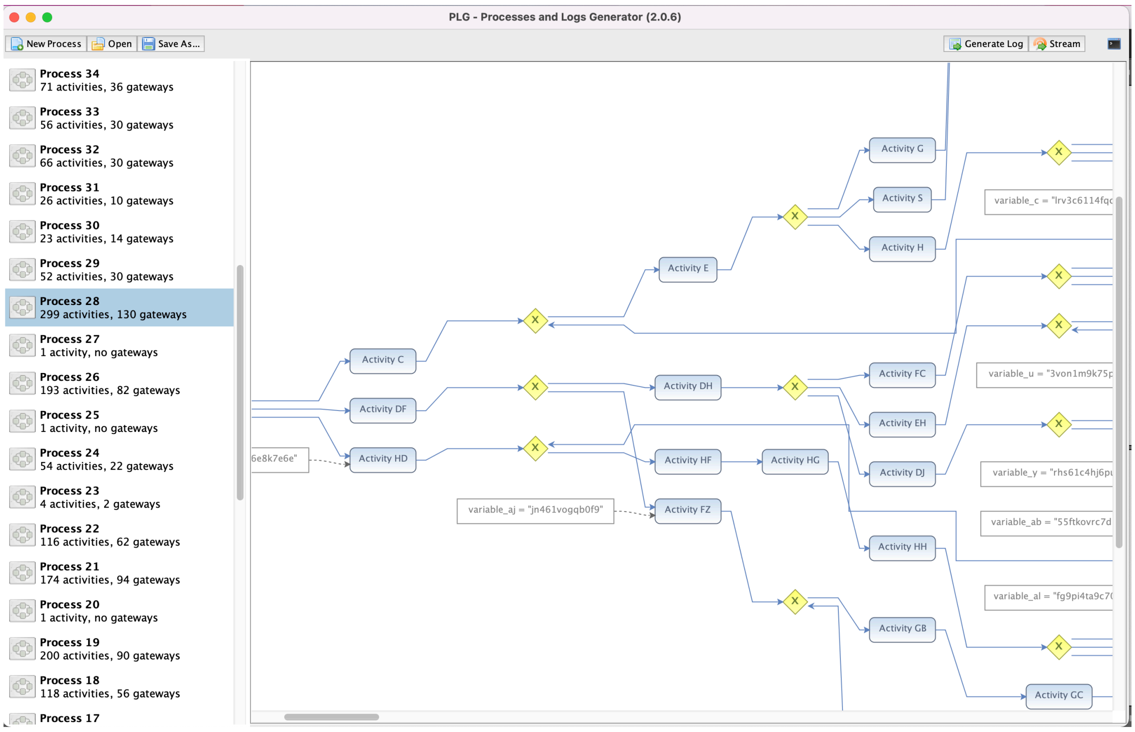

PLG - Processes and Logs Generator (2.0.6)
New Process
Open
Save As...
Generate Log
Stream
Process 34
71 activities, 36 gateways
Process 33
56 activities, 30 gateways
Process 32
66 activities, 30 gateways
Process 31
26 activities, 10 gateways
Process 30
23 activities, 14 gateways
Process 29
52 activities, 30 gateways
Process 28
299 activities, 130 gateways
Process 27
1 activity, no gateways
Process 26
193 activities, 82 gateways
Process 25
1 activity, no gateways
Process 24
54 activities, 22 gateways
Process 23
4 activities, 2 gateways
Process 22
116 activities, 62 gateways
Process 21
174 activities, 94 gateways
Process 20
1 activity, no gateways
Process 19
200 activities, 90 gateways
Process 18
118 activities, 56 gateways
Process 17
Activity G
Activity S
Activity H
Activity E
Activity C
Activity DF
Activity HD
Activity DH
Activity FC
Activity EH
Activity DJ
Activity HF
Activity HG
Activity FZ
Activity HH
Activity GB
Activity GC
variable_c = "lrv3c6114fqc
variable_u = "3von1m9k75p
variable_y = "rhs61c4hj6pu
variable_ab = "55ftkovrc7d
variable_al = "fg9pi4ta9c70
variable_aj = "jn461vogqb0f9"
6e8k7e6e"

Figure 7.1: PLG2 application UI for generating random processes and event logs.

- *Missing tail*: similar to the head missing noise but acting on the end of traces.
- *Missing event*: it is the maximum number of missing event in each trace, the tool will randomly choose the actual value between one and the provided value.
- *Alien event*: an alien event will be introduced into the trace in a random position.
- *Double event*: an event will be generated multiple times.

All these values are translated to probability form for ease of use, since absolute values need to be defined dependently on the model size.

In Fig. 7.1 it is shown the user interface of PLG2 with some random processes generated by the software. Note that number of gateways are includes also AND gates which will be transformed into place-transition pairs when exporting models as Petri nets.

7.1.2 Conformance Checker: CoCoMoT

As already mentioned in Section 4.1 for the step **A** and **B** we need to decide a conformance checking algorithm to compute optimal sub-alignments and next-best sub-alignments in case of border conflicts.

We collaborated with the authors of *CoCoMoT (Computing Conformance Modulo Theories)* framework [25] and succeeded to add a new functionality of generating multiple alignments at once, increasingly ordered according to their costs, it is the job of stitching algorithm to discard sub-alignments not satisfying the next-best property. CoCoMoT is based on *SMT* algorithmic framework [26] in which process logics are converted into *SAT-based* encodings then conformance checking problems are translated and solved through a SMT solver. In the original work CoCoMoT was proposed as a checker of multi-perspective processes but in our experiments we stay at control-flow level.

Returning an optimal sub-alignment is an easy task for CoCoMoT but compute further sub-alignments according to their costs is quite tricky. At the end, we decided to stop the algorithm by setting a *cost bound*, namely, it generates all alignments up to including prefixed cost threshold. The outputs will be written into a file and processed by the stitching algorithm.

The software is freely available [†] as an open-source project, currently it is a Python script. The tool uses *pm4py* [27] to parse traces and as back-end SMT solver it leverages *Yices 2* [28] or alternatively *Z3* [29], both written in C++ and employed by using Python bindings [‡].

7.2 Implementation Details

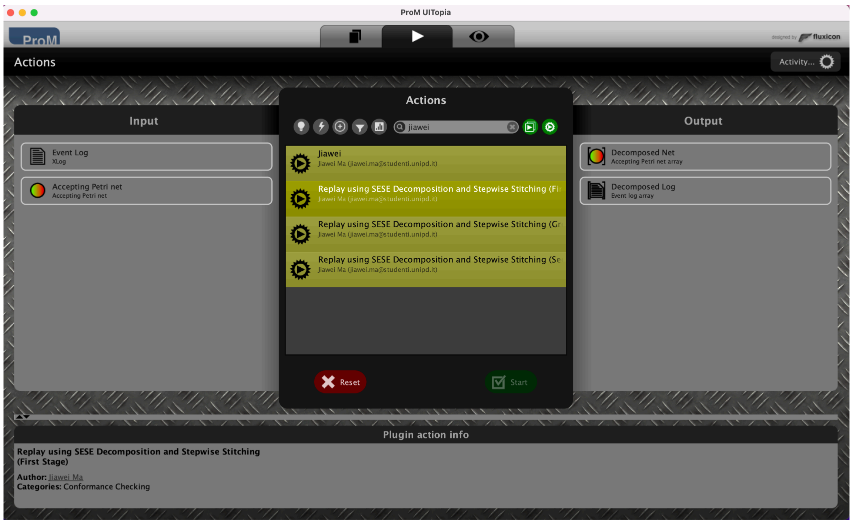
We implemented our stitching algorithm in ProM6.11. Since we require an external conformance checker then we divided the algorithm into two plug-ins: the first receives in input an event log and process model and it outputs SESE-decomposed subnets and sub-logs, the latter data are given in input to the CoCoMoT for calculating sub-alignments following the way we described in Section 7.1.2 and, finally, a second stitching plug-in stepwise merge all sub-alignments.

We coded the new plug-ins `StepwiseStitchDecomposedRepalyPlugin1,2` as subcomponents under the package `DecomposedReplayer`. We applied the same output visualizer as for the recomposing plug-in [30], for an illustration see Fig. 7.2.

All the experiments are done using a personal computer based on *macOS Monterey*, equipped with a processor *Intel Core i5, 2,4 GHz and quad-core*, *8 GB RAM*. The recomposing replayer seems to be optimized to use all cores of the processor, in order to make fair comparisons we restricted ProM to only consume one core at a time.

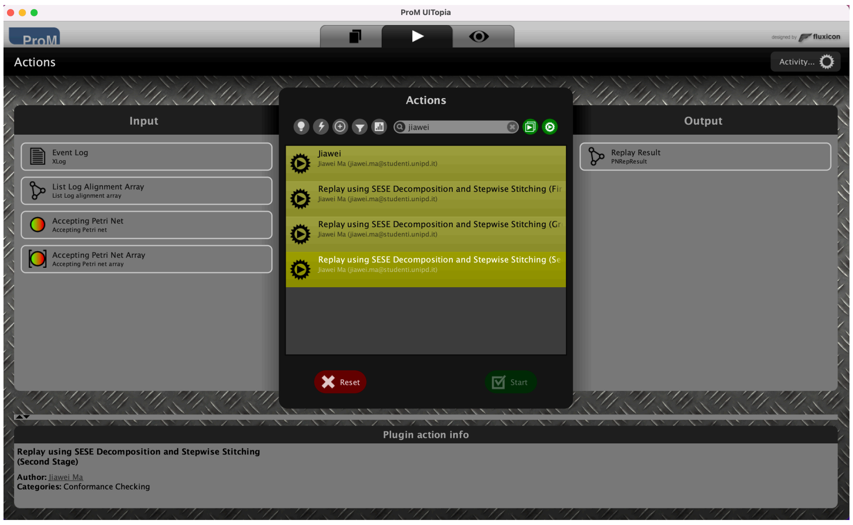
[†]https://github.com/bytekid/cocomot

[‡]https://github.com/SRI-CSL/yices2_python_bindings, https://github.com/z3prover/z3


ProM UITopia
ProM
Actions
Activity...
Input
Event Log
XLog
Accepting Petri net
Accepting Petri net
Actions
jiawei
Jiawei
Jiawei Ma (jiawei.ma@studenti.unipd.it)
Replay using SESE Decomposition and Stepwise Stitching (Fi
Replay using SESE Decomposition and Stepwise Stitching (Gr
Replay using SESE Decomposition and Stepwise Stitching (Se
Reset
Start
Output
Decomposed Net
Accepting Petri net array
Decomposed Log
Event log array
Plugin action info

Replay using SESE Decomposition and Stepwise Stitching (First Stage)
Author: Jiawei Ma
Categories: Conformance Checking

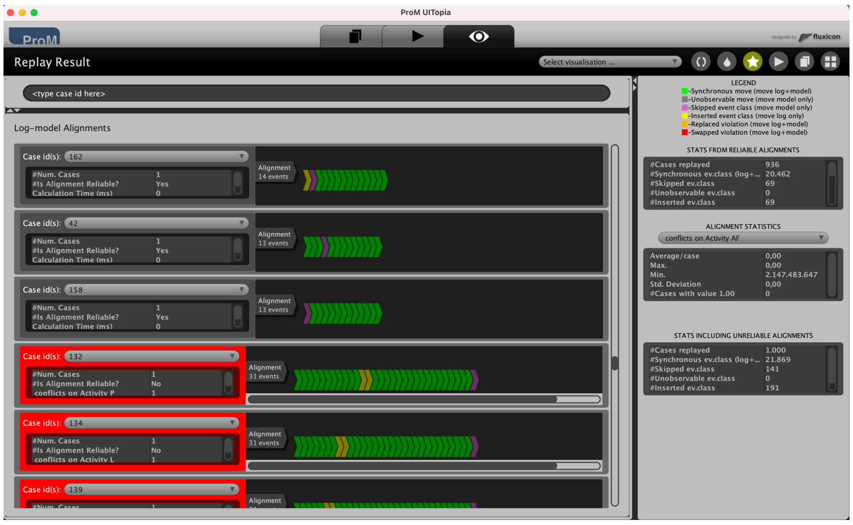
(a) First part of the stepwise stitching plug-in in ProM. In input are required an event log and an accepting Petri net (Petri net with initial and final markings), it returns SESE-decomposed nets and logs.


ProM UITopia
ProM
Actions
Activity...
Input
Event Log
XLog
List Log Alignment Array
List Log alignment array
Accepting Petri Net
Accepting Petri net
Accepting Petri Net Array
Accepting Petri net array
Actions
jiawei
Jiawei
Jiawei Ma (jiawei.ma@studenti.unipd.it)
Replay using SESE Decomposition and Stepwise Stitching (Fi
Replay using SESE Decomposition and Stepwise Stitching (Gr
Replay using SESE Decomposition and Stepwise Stitching (Se
Reset
Start
Output
Replay Result
PNRepResult
Plugin action info

Replay using SESE Decomposition and Stepwise Stitching (Second Stage)
Author: Jiawei Ma
Categories: Conformance Checking

(b) Second part of the stepwise stitching plug-in in ProM. In input are required an event log, an accepting Petri net, decomposed nets and a list of sub-alignments, it returns the replay results.

Figure 7.2: An illustration of the stepwise stitching algorithm plug-in in ProM.



Replay Result

Log–model Alignments

```
Case id(s): 162
#Num. Cases                    1
#Is Alignment Reliable?        Yes
Calculation Time (ms)          0
Alignment
14 events

Case id(s): 42
#Num. Cases                    1
#Is Alignment Reliable?        Yes
Calculation Time (ms)          0
Alignment
13 events

Case id(s): 158
#Num. Cases                    1
#Is Alignment Reliable?        Yes
Calculation Time (ms)          0
Alignment
13 events

Case id(s): 132
#Num. Cases                    1
#Is Alignment Reliable?        No
conflicts on Activity P        1
Alignment
31 events

Case id(s): 134
#Num. Cases                    1
#Is Alignment Reliable?        No
conflicts on Activity L        1
Alignment
31 events

Case id(s): 139
#Num. Cases                    1
Alignment
```

LEGEND
- 🟩–Synchronous move (move log+model)
- ▨–Unobservable move (move model only)
- 🟪–Skipped event class (move model only)
- 🟨–Inserted event class (move log only)
- 🟧–Replaced violation (move log+model)
- 🟥–Swapped violation (move log+model)

STATS FROM RELIABLE ALIGNMENTS

```
#Cases replayed                 936
#Synchronous ev.class (log+...  20.462
#Skipped ev.class               69
#Unobservable ev.class          0
#Inserted ev.class              69
```

ALIGNMENT STATISTICS

conflicts on Activity AF

```
Average/case                    0,00
Max.                            0,00
Min.                            2.147.483.647
Std. Deviation                  0,00
#Cases with value 1.00          0
```

STATS INCLUDING UNRELIABLE ALIGNMENTS

```
#Cases replayed                 1.000
#Synchronous ev.class (log+...  21.869
#Skipped ev.class               141
#Unobservable ev.class          0
#Inserted ev.class              191
```

(c) Replay results visualizer. On the left side it is possible to scroll up-down for inspecting each merged alignment, on the right hand there are statistics about the event log and replay results. Note that pseudo-alignments are highlighted with bold red border.

Figure 7.2: Cont.) An illustration of the stepwise stitching algorithm plug-in in ProM.

7.3 Data

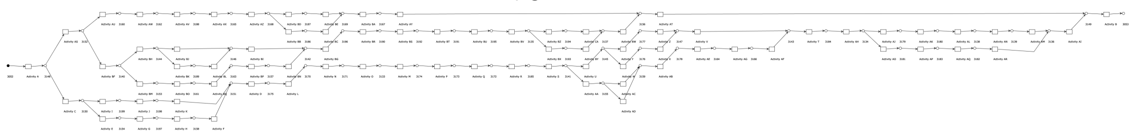
For our experiments we generated *three process models* as depicted in Fig.7.3. We simulated each model to produce *three event logs consisting of 1000 traces*: the first is with 5‰ of each noise category we described in Section 7.1.1, the second log is with 10‰ noise and the third one is noise-free. We report the data details in Table 7.1, in Fig. 7.4a and in Fig. 7.4b.

We made the models increasingly large to challenge the algorithms under evaluation, reaching with the biggest model consisting of 120 activities and 52 gateways. Even if the model associated with the datasets *synthetic_2* is bigger than the first one but the simulator returned event logs with shorter traces. We could of course "stress" more the algorithms, but the hardware available would not carry that workload.

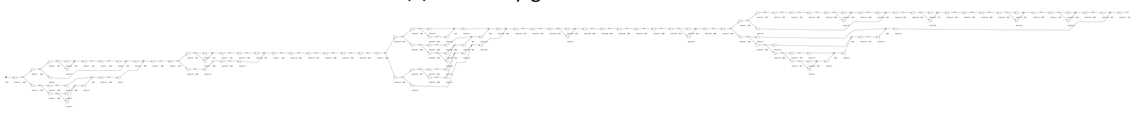
7.4 Results

During the experiments we recorded the number of subnets produced by the first stepwise stitching plug-in, that is the same as in the recomposing algorithm (in this case the initial number of subnets), we show the results in Fig. 7.5a. The amount of noise did not influence the SESE-decomposition method as we can easily understand and we also see that larger is the net

(a) Randomly generated Petri net 1.



(b) Randomly generated Petri net 2.

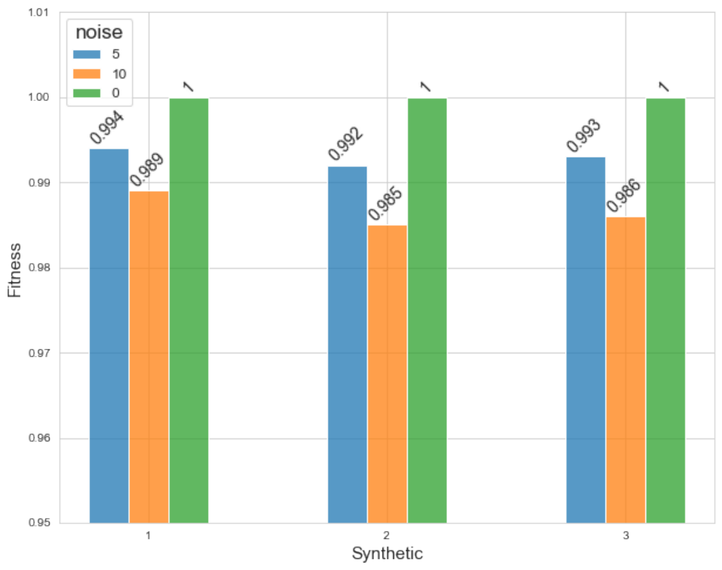


(c) Randomly generated Petri net 3.

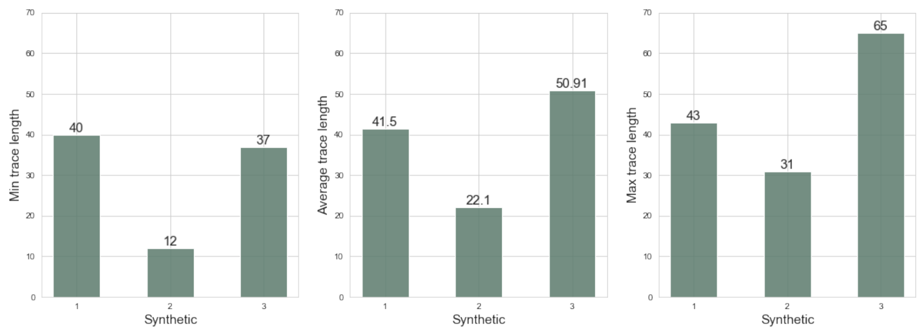
Figure 7.3: Three Petri net randomly generated by PLG2.

Synthetic Data Details			
Dataset	**N. activities**	**N. gateways**	**Noise**
Synthetic_1_5	56	30	5‰
Synthetic_1_10	56	30	10‰
Synthetic_1	56	30	0‰
Synthetic_2_5	79	38	5‰
Synthetic_2_10	79	38	10‰
Synthetic_2	79	38	0‰
Synthetic_3_5	120	52	5‰
Synthetic_3_10	120	52	10‰
Synthetic_3	120	52	0‰

Table 7.1: Dataset summary table. Noise refers to each noise category we described in Section 7.1.1.



(a) True fitness between the synthetic event logs and respective Petri nets.



(b) Event logs' traces statistics: minimum trace length, average trace length and maximum trace length.

Figure 7.4

more subnets are generated.

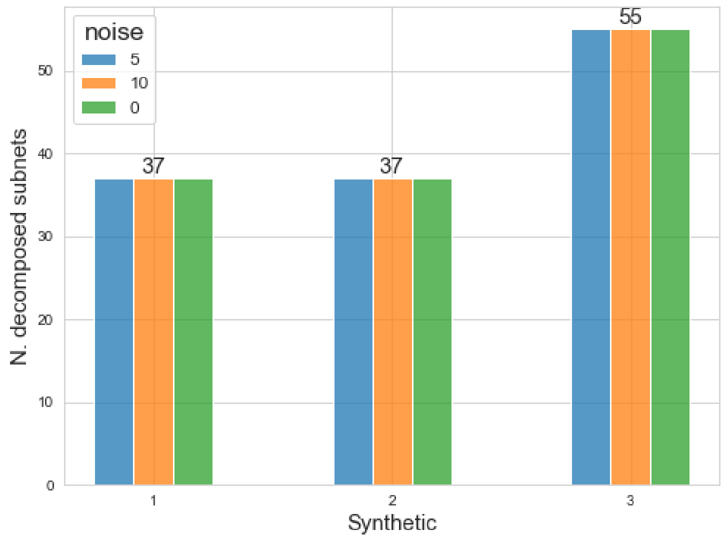
The results in Fig. 7.5b confirm our intuition that both net size and data noise trigger the recomposing step. Zero number of recomposing steps can be explained by the fact that, perfect fitting traces implies perfect fitting sub-traces (see Theorem. 3.8) and certainly there are no border conflicts among sub-alignments with only sync moves.

Looking at Fig. 7.6 we can really appreciate the speed-up offered by the stitching algorithm. To make the comparisons as fair as possible we calculated the stitching time as the sum of the time required by our stitching plug-in plus estimated time really served to find out necessary next-best sub-alignments. Specifically, to approximate the latter amount of time we have just calculated the average time for finding one sub-alignment and then we have multiplied it by average number of attempts that the stitching algorithm required to reach the non-conflict sub-alignments property. Here we underline some interesting facts:

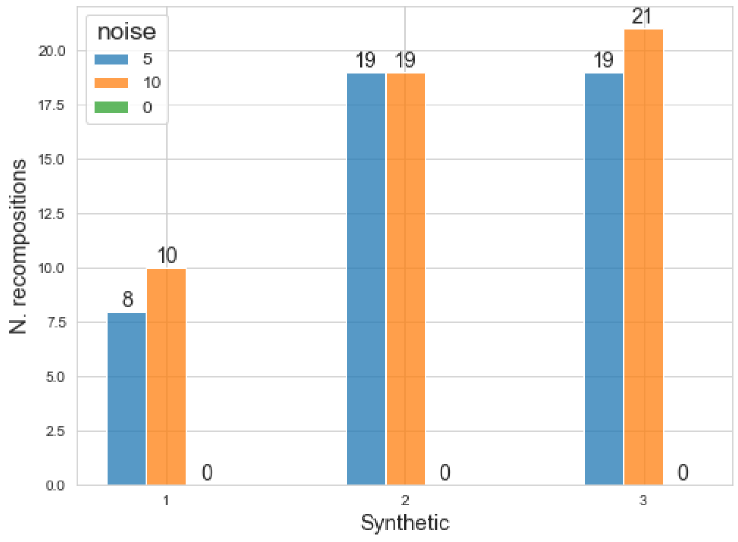
1. In all the cases the first net decomposition and replay time is negligible with respect to the total time, i.e. the recomposing algorithm spent all time to recompose and replay again, when some border conflicts were found.
2. Larger models required more computation time and more noise caused extra merging or recomposing time, except for the second case where the 5‰ noise took the most long processing time, this may be due to randomness of noise injection.
3. Stitching algorithm seems to perform better for the most noisy datasets (10‰ of noise level) with respect to the recomposing method.
4. For the dataset *synthetic_3_10* there was a speed-up of more than $5\times$, that was a great improvement in terms of computation time.
5. For perfect fitting datasets the two algorithms performed similarly (very fast) in case of the second and third dataset, except for the first synthetic dataset due to the time required by the checker.

On the other side of the coin, we also observed during the experiments that the time spent on finding sets of sub-alignments in addition to optimal ones really hampered overall performance of our algorithm, the reason is quickly guessed: CoCoMoT is not optimized to output multiple alignment along with a best one, it just returned a lot more sub-alignments than those really needed, this is what we observed during the experiments.

There is another drawback regarding the stepwise stitching approach that is valid alignments were not always guaranteed. However, It is empirically verified in Fig. 7.7 that there are significant portions of traces for which the new algorithm could not compute some valid alignments.

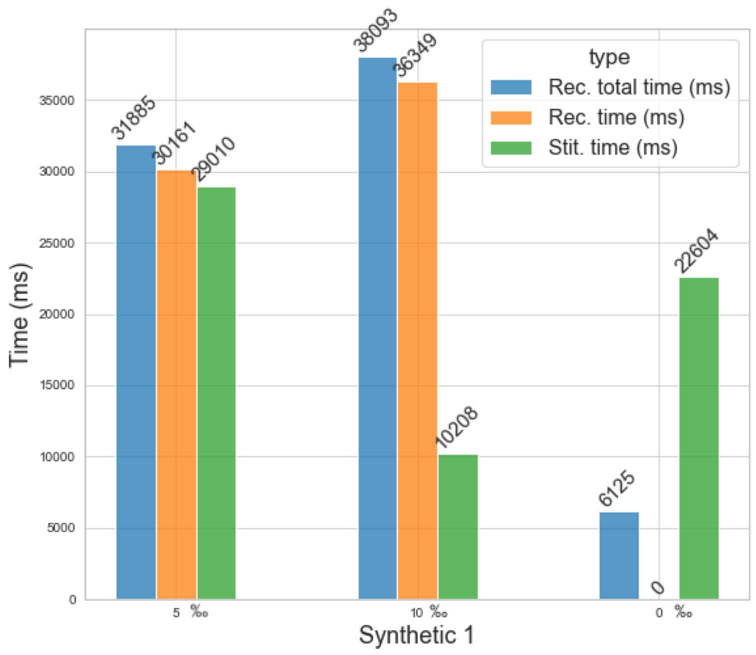


(a) Number of subnets generated by using SESE-decomposition. Both algorithms generated equal number of subnets.

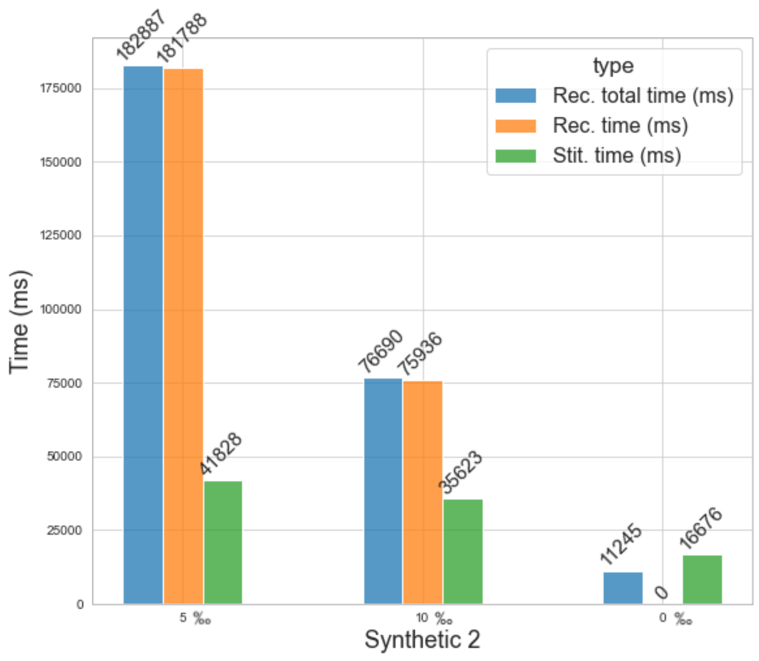


(b) Number of recomposing steps occurred for each event log.

Figure 7.5: Some statistics about recomposing method.

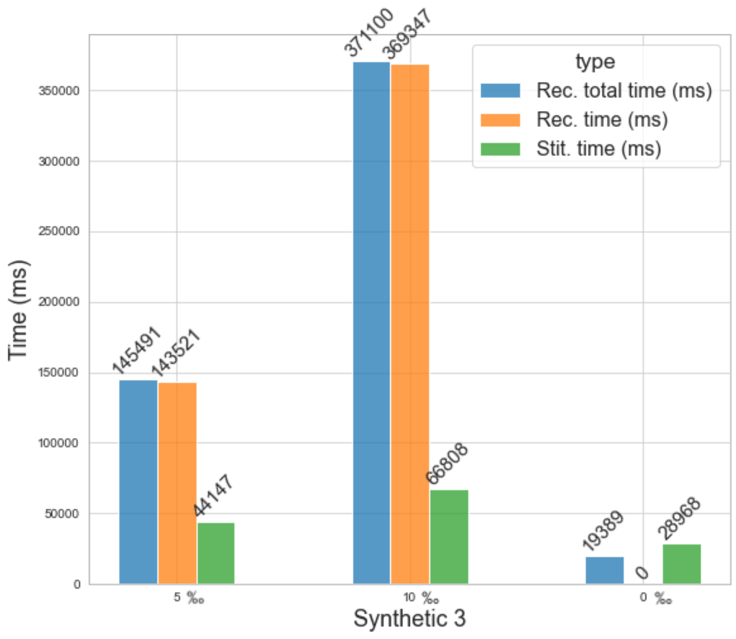


(a) It is shown the computation time of each method: *synthetic_1*.



(b) It is shown the computation time of each method: *synthetic_2*.

Figure 7.6: Summary of computation time. The total time of the recomposing algorithm is the sum of the first net decomposition and replay time plus recomposing time, that is the time spent on recomposing and replay again. To make the comparisons as fair as possible we calculated the stitching time as the sum of the time required by our stitching plug-in plus estimated time really served to find out necessary next-best sub-alignments. Specifically, to approximate the latter amount of time we have just calculated the average time for finding one sub-alignment and then we have multiplied it by average number of attempts that the stitching algorithm required to reach the non-conflict sub-alignments property.



(c) It is shown the computation time of each method: *synthetic_3*.

Figure 7.6: (Cont.) Summary of computation time. The total time of the recomposing algorithm is the sum of the first net decomposition and replay time plus recomposing time, that is the time spent on recomposing and replay again. To make the comparisons as fair as possible we calculated the stitching time as the sum of the time required by our stitching plug-in plus estimated time really served to find out necessary next-best sub-alignments. Specifically, to approximate the latter amount of time we have just calculated the average time for finding one sub-alignment and then we have multiplied it by average number of attempts that the stitching algorithm required to reach the non-conflict sub-alignments property.

In our opinion, the problem is either in setting a too restrictive bound on the checker so that our stitching algorithm has just exhausted all available sub-alignments and then it has been stopped, or the issue resides in defining next-best alignments applying the rules we discussed in Section 4.1, in this way we could have discarded useful sub-alignments. We need more time to investigate on this, perhaps different heuristics are necessary. Nevertheless, the recomposing algorithm rejected considerable amount of problematic traces as well, especially for the second dataset, arriving to reject more than half of distinct not perfectly fitting traces in the worst case.

Finally, recalling the remarks we made in Section 4.1, whereas recomposing approach either compute a single value fitness or a fitness interval in which the true fitness falls, our method in case of unsolvable border conflicts only returns an approximated fitness interval. These claims are confirmed by the results in Fig. 7.8. Fortunately, pseudo-alignments are indicated anyway so that we can further analyse them and, in a real-world conformance problem, identify possible process bottlenecks.

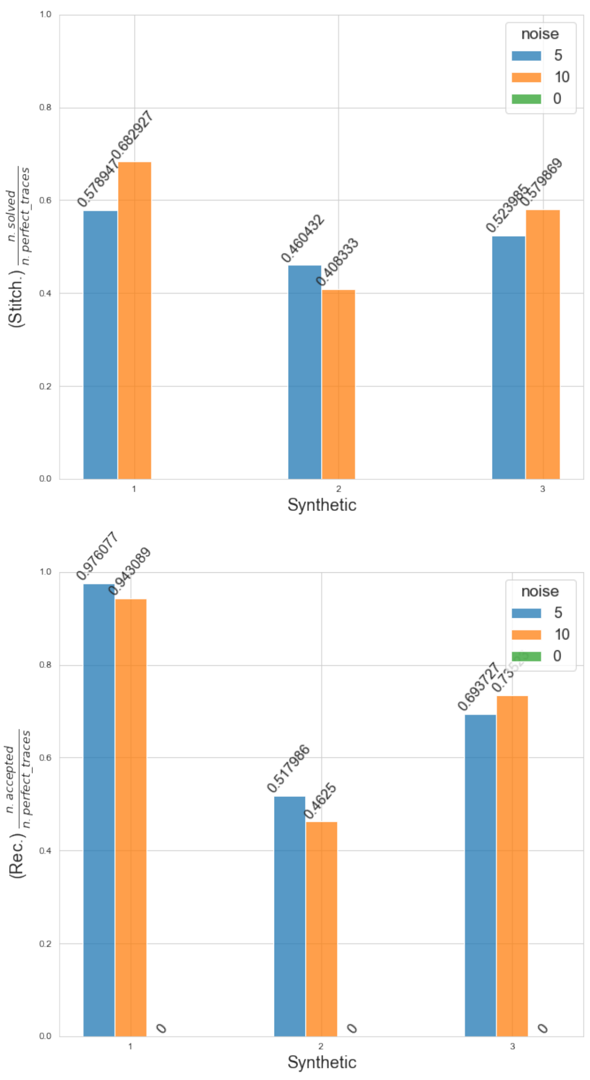
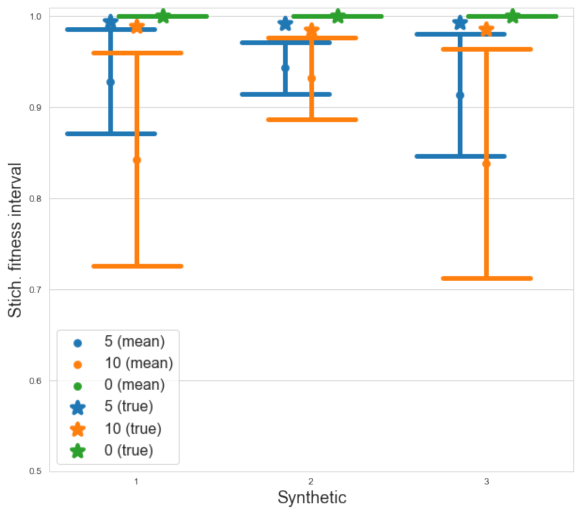


Figure 7.7: Percentage of solved and accepted traces recorded for stitching and recomposing algorithms respectively. A solved trace is a not perfect fitting trace (positive costs) for which the stitching algorithm successfully found out correct sub-alignments to be merged. We recall that a trace is rejected by the recomposing algorithm when its number of border conflicts surpasses a prefixed threshold, the number of accepted traces counts how many traces for which recomposing algorithm succeeded to align. We computed these values by firstly counting the number of distinct not perfect fitting trace for each event log, then, we compute the ratio.



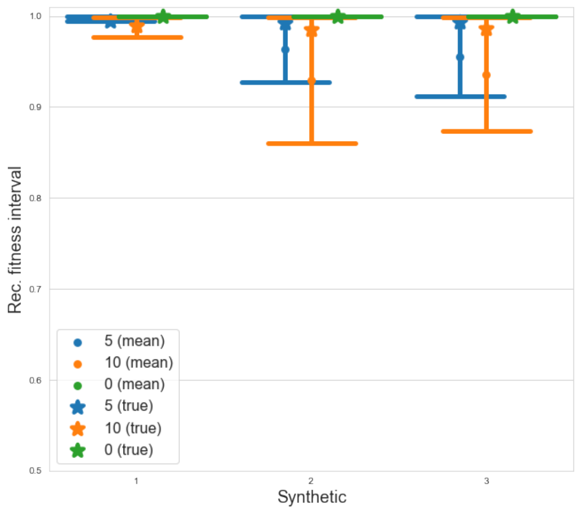


Figure 7.8: The plot on the top is related to fitness intervals computed by the stepwise stitching algorithm, on the bottom, it is shown intervals returned by the recomposing method. True fitness values are indicated by star symbols.

8 Conclusion

In this thesis we went through the state-of-the-art conformance checking with decomposition approaches while motivating, by making examples, the relevant steps. Then, we gave our contribution by proposing a novel decomposed conformance checking algorithm based on sub-alignments stepwise stitching which has some advantages with respect to the other existing methods. We formally defined our algorithm and proved its correctness, also running examples were provided for a better understanding of its main ideas.

The last part of this work is dedicated to the experiments in which we tested our algorithm by using synthetic datasets generated by a notable process data generator, this was to simulate a "Big Data" setting. The final results revealed both positive and negative aspects of the new algorithm. While the weaknesses of our algorithm might be overcame by only changing, under the same framework, some heuristics we applied, its strengths are very valuable, especially for what regarding the computation time.

Discussing about future work, firstly, we should further investigate and find the causes of the sub-alignments mismatch issue detected during the experiments: there were still significant portions of traces, with respect to the state-of-the-art method, for which our stitching algorithm could not compute a valid alignment. More experiments should also be carried out to extensively test our algorithm, using some real-world datasets. We will have to vary the initial assumptions as we have just mentioned to see if the performance will be further improved, due to the lack of time it was not possible. We can also work on the algorithm implementation side, optimizing the conformance checker (Python) for sub-alignments computation and the stitching algorithm (Java) for parallel processing, integrate them to create an integrated software for a better user experience. As a solution of this application integration issue, we can try to implement the algorithm in a different platform, based on Python (Pm4Py) instead of running it in a JVM (ProM).

References

[1] W. M. v. d. Aalst, *Process mining: data science in action*. Springer, 2016.

[2] ——, "Process mining: Overview and opportunities," *ACM Transactions on Management Information Systems (TMIS)*, vol. 3, no. 2, pp. 1–17, 2012.

[3] W. L. J. Lee, H. Verbeek, J. Munoz-Gama, W. M. v. d. Aalst, and M. Sepúlveda, "Recomposing conformance: Closing the circle on decomposed alignment-based conformance checking in process mining," *Information Sciences*, vol. 466, pp. 55–91, 2018.

[4] W. M. v. d. Aalst, "Decomposing petri nets for process mining: A generic approach," *Distributed and Parallel Databases*, vol. 31, no. 4, pp. 471–507, 2013.

[5] J. Munoz-Gama, J. Carmona, and W. M. v. d. Aalst, "Single-entry single-exit decomposed conformance checking," *Information Systems*, vol. 46, pp. 102–122, 2014.

[6] H. Verbeek and W. M. v. d. Aalst, "Merging alignments for decomposed replay," in *International Conference on Applications and Theory of Petri Nets and Concurrency*. Springer, 2016, pp. 219–239.

[7] A. Rozinat, I. S. de Jong, C. W. Günther, and W. M. v. d. Aalst, "Process mining applied to the test process of wafer scanners in asml," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 39, no. 4, pp. 474–479, 2009.

[8] W. M. v. d. Aalst, A. Adriansyah, A. K. A. d. Medeiros, F. Arcieri, T. Baier, T. Blickle, J. C. Bose, P. v. d. Brand, R. Brandtjen, J. Buijs *et al.*, "Process mining manifesto," in *International conference on business process management*. Springer, 2011, pp. 169–194.

[9] W. M. v. d. Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE transactions on knowledge and data engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[10] J. C. Buijs, B. F. v. Dongen, and W. M. v. d. Aalst, "On the role of fitness, precision, generalization and simplicity in process discovery," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2012, pp. 305–322.

[11] M. d. Leoni, J. Munoz-Gama, J. Carmona, and W. M. Van Der Aalst, "Decomposing alignment-based conformance checking of data-aware process models," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2014, pp. 3–20.

[12] W. M. v. d. W. Aalst, A. Adriansyah, and B. van Dongen, "Replaying history on process models for conformance checking and performance analysis," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 182–192, 2012.

[13] D. Laney *et al.*, "3d data management: Controlling data volume, velocity and variety," *META group research note*, vol. 6, no. 70, p. 1, 2001.

[14] J. Vanhatalo, H. Völzer, and J. Koehler, "The refined process structure tree," *Data & Knowledge Engineering*, vol. 68, no. 9, pp. 793–818, 2009.

[15] A. Polyvyanyy, J. Vanhatalo, and H. Völzer, "Simplified computation and generalization of the refined process structure tree," in *International Workshop on Web Services and Formal Methods*. Springer, 2010, pp. 25–41.

[16] A. Rozinat and W. M. v. d. Aalst, "Conformance checking of processes based on monitoring real behavior," *Information Systems*, vol. 33, no. 1, pp. 64–95, 2008.

[17] M. Weidlich, A. Polyvyanyy, N. Desai, and J. Mendling, "Process compliance measurement based on behavioural profiles," in *International Conference on Advanced Information Systems Engineering*. Springer, 2010, pp. 499–514.

[18] J. E. Cook and A. L. Wolf, "Software process validation: quantitatively measuring the correspondence of a process to a model," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 8, no. 2, pp. 147–176, 1999.

[19] M. d. Leoni and W. M. Van Der Aalst, "Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming," in *Business process management*. Springer, 2013, pp. 113–129.

[20] L. Wang, Y. Du, and W. Liu, "Aligning observed and modelled behaviour based on workflow decomposition," *Enterprise Information Systems*, vol. 11, no. 8, pp. 1207–1227, 2017.

[21] W. M. Van Der Aalst, "Decomposing process mining problems using passages," in *International Conference on Application and Theory of Petri Nets and Concurrency*. Springer, 2012, pp. 72–91.

[22] H. Verbeek, "Decomposed replay using hiding and reduction." in *PNSE@ Petri Nets*, 2016, pp. 233–252.

[23] B. F. Dongen, A. K. A. de Medeiros, H. Verbeek, A. Weijters, and W. M. v. d. Aalst, "The prom framework: A new era in process mining tool support," in *International conference on application and theory of petri nets*. Springer, 2005, pp. 444–454.

[24] A. Burattin, "Plg2: multiperspective processes randomization and simulation for online and offline settings," *arXiv preprint arXiv:1506.08415*, 2015.

[25] P. Felli, A. Gianola, M. Montali, A. Rivkin, and S. Winkler, "Cocomot: Conformance checking of multi-perspective processes via smt," in *International Conference on Business Process Management*. Springer, 2021, pp. 217–234.

[26] C. Barrett and C. Tinelli, "Satisfiability modulo theories," in *Handbook of model checking*. Springer, 2018, pp. 305–343.

[27] A. Berti, S. J. Van Zelst, and W. M. Van Der Aalst, "Process mining for python (pm4py): bridging the gap between process-and data science," *arXiv preprint arXiv:1905.06169*, 2019.

[28] B. Dutertre, "Yices 2.2," in *International Conference on Computer Aided Verification*. Springer, 2014, pp. 737–744.

[29] L. d. Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[30] W. L. J. Lee, H. Verbeek, J. Munoz-Gama, W. M. Van Der Aalst, and M. Sepúlveda, "Replay using recomposition: Alignment-based conformance checking in the large." in *BPM (Demos)*, 2017.