Process Diagnostics Using Trace Alignment: Opportunities, Issues, and Challenges

R. P. Jagadeesh Chandra Bose[a,b], Wil M. P. van der Aalst[a]

[a] *Department of Mathematics and Computer Science, University of Technology, Eindhoven, The Netherlands*
[b] *Philips Healthcare, Veenpluis 5–6, Best, The Netherlands*

Abstract

Business processes leave trails in a variety of data sources (e.g., audit trails, databases, transaction logs). Hence, every process instance can be described by a trace, i.e., a sequence of events. Process mining techniques are able to extract knowledge from such traces and provide a welcome extension to the repertoire of business process analysis techniques. Recently, process mining techniques have been adopted in various commercial BPM systems (e.g., BPM|one, Futura Reflect, ARIS PPM, Fujitsu Interstage, Businesscape, Iontas PDF, QPR PA). Unfortunately, traditional process discovery algorithms have problems dealing with less structured processes. The resulting models are difficult to comprehend or even misleading. Therefore, we propose a new approach based on *trace alignment*. The goal is to align traces in such a way that event logs can be explored easily. Trace alignment can be used to explore the process in the early stages of analysis and to answer specific questions in later stages of analysis. Hence, it complements existing process mining techniques focusing on discovery and conformance checking.

Keywords: diagnostics, conformance, alignment, execution patterns, process mining

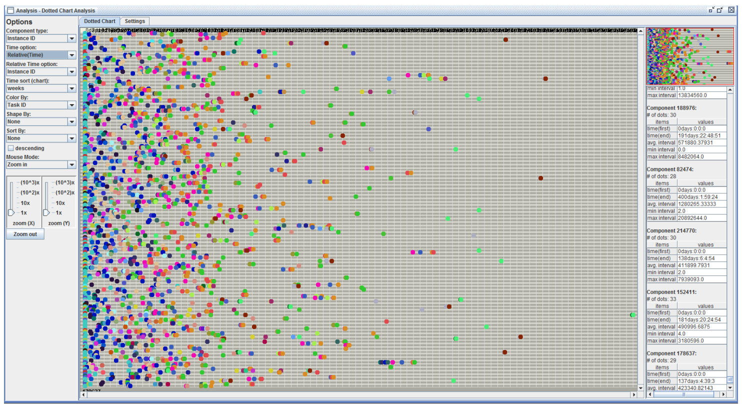
1. Introduction

Today's information systems are recording an abundance of event logs. Events may be scattered over various data sources, e.g., the database of a hospital containing information about patients, the audit trails of a BPM (Business Process Management) system, the transaction log of an airline's web service, and the job lists of a printer. By ordering events belonging to each process instance, we obtain a collection of *traces* also referred to as an *event log*. Process mining techniques can extract non-trivial knowledge and interesting insights from these event logs and exploit these for further analysis [1]. For example, it is possible to discover a process model from an event log. Process mining is not limited to control-flow and may also be used to discover other perspectives, e.g., the social network of the people working on the process or decision trees explaining the decision points in the process. Process mining is not limited to discovery and also includes conformance checking (identifying points where process instances deviate from expected and/or normative behavior) and model enhancement (e.g., enriching models based on event logs).

The lion's share of process mining research has been devoted to control-flow discovery. In this paper, we focus on complementary techniques to *diagnose processes* that are based on the *alignment of traces.* This work is inspired by practical experiences using classical control-flow discovery techniques. We (our research group) have applied process mining in over 100 organizations and our experiences show that processes tend to be less structured than expected. Moreover, event logs tend to be far from complete, i.e., just a fraction of the possible behavior indeed occurs. Therefore, it may be too ambitious to construct a process model that also says something about unseen cases, i.e., future process instances that will leave traces that were not seen before. If processes are less structured and event logs are incomplete, then it is better to *carefully inspect the event log by grouping and aligning the traces found in the event log.* First, we group similar traces in clusters [2]. Second, we visualize these clusters by aligning the traces. *By aligning traces we can see the common and frequent behavior, and distinguish this from the exceptional behavior.*

Our approach is inspired by biological sequence alignment [3]. Sequence alignment is an essential tool in bioinformatics that assists in unraveling the secondary and tertiary structures of proteins and molecules, their evolution and functions, and in inferring the taxonomic, phylogenetic or cladistic relationships between organisms, diagnoses of genetic diseases etc. [4, 5]. Process mining also deals with sequences, i.e., traces of events stored in event logs. Multiple sequence alignment has been a subject of extensive research in computational biology for over three decades. There are still many open problems e.g., dealing with ever increasing size and complexity of the data sets, misalignments and alignment errors [6, 7], obtaining accurate alignments of non-coding and non-transcribed sequences [8], integrating disparate sources of knowledge when performing alignments (knowledge-based multiple sequence alignment) [9, 10] etc. Moreover, there are various new challenges when adopting biological sequence alignment to trace alignment in the context of business processes e.g., traces in an event log in process mining need not be from a coherent set of cases and can be of different lengths, scoring matrices of substitution and insertion/deletion of activities need to be defined/derived, the size of the alphabet (number of activities) can be large of the order of a few tens or hundreds etc. The topic of trace alignment has not been explored before. We will show that it can be used to answer a variety of questions and that it is a welcome addition to the repertoire of process mining techniques.

To illustrate the importance of trace alignment, consider the dotted chart [11] in Figure 1. Every line corresponds to a process instance and every dot corresponds to an event. The color of the event indicates the activity that was executed. The horizontal dimension describes time. Note that the dots are not aligned. Hence, it is difficult to see common patterns among different cases. Events are positioned based on their timestamps rather than the activity that was executed and similarities to events in other instances. Figure 2 shows the



Options

Component type: Instance ID
Time option: Relative(Time)
Relative Time option: Instance ID
Time sort (chart): weeks
Color By: Task ID
Shape By: None
Sort By: None
descending
Mouse Mode: Zoom in

min interval	1.0
max interval	13834560.0

Component 188976:
# of dots: 30

items	values
time(first)	0days:0:0:0
time(end)	191days:22:48:51
avg. interval	571880.37931
min interval	0.0
max interval	8482064.0

Component 82474:
# of dots: 28

items	values
time(first)	0days:0:0:0
time(end)	400days:1:59:24
avg. interval	1280265.33333
min interval	2.0
max interval	20892644.0

Component 214770:
# of dots: 30

items	values
time(first)	0days:0:0:0
time(end)	138days:6:4:54
avg. interval	411899.7931
min interval	2.0
max interval	7939093.0

Component 152411:
# of dots: 33

items	values
time(first)	0days:0:0:0
time(end)	181days:20:24:54
avg. interval	490996.6875
min interval	4.0
max interval	3180596.0

Component 178637:
# of dots: 29

items	values
time(first)	0days:0:0:0
time(end)	137days:4:39:3
avg. interval	423340.82143

Figure 1: Dotted chart showing all event in one glance. Consider a dot at position (x, y) having a color c. This dot refers to an event for the process instance corresponding to y (i.e., a horizonal line) that occurred at time x (i.e., time increases from left to right). The color describes a property of the event; in this case the activity that was executed.

effect of trace alignment. Both dimensions are used in a similar fashion as before, i.e., every row corresponds to a process instance and time increases from left to right. However, now the horizonal position is based on *logical* time rather than *real* timestamps. When two rows have the same activity name in the same column, then the corresponding two events are very similar and are therefore aligned. Note that the same activity can appear in multiple columns. By reading a row from left to right, we can see the sequence of activities (i.e., the trace) that was executed for a process instance. Process instances having the same trace can be grouped into one row to simplify the diagram. Clearly, Figure 2 is much more informative than Figure 1 if one is interested in commonalities and differences among traces.

The challenge is to find an alignment that is as simple and informative as possible. For example, the number of columns and gaps should be minimized while having as much consensus as possible per column. Obtaining such high quality alignments turns out to be a challenging problem.

The remainder of this paper is organized as follows. In Section 2, we introduce the basic notations used in the paper and list the questions we would like to answer. Section 3 introduces the concept of trace alignment and discusses the techniques for finding alignments. In Section 4, we propose a framework for finding alignments over a set of traces. In Section 5, we present techniques for

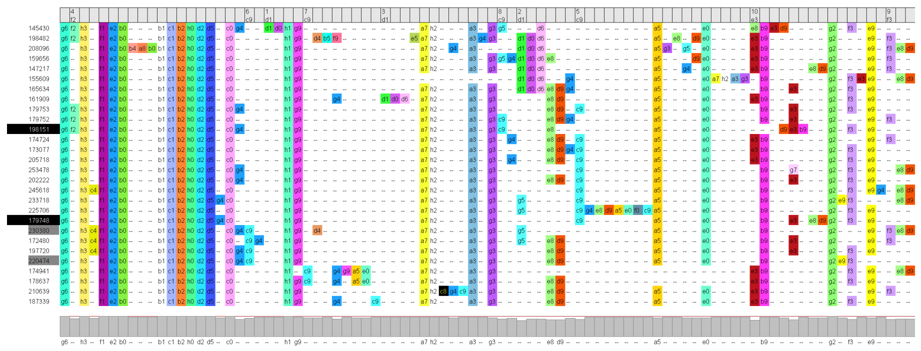


Figure 2: Traces aligned using the approach presented in this paper. Each row refers to a process instance. Columns describe positions in traces. Trace alignment aims a minimizing the number of gaps and maximizing the consensus.

refining alignments to improve alignment quality. In Section 6, we present our implementation using the process mining framework ProM. Section 7 discusses three case studies that demonstrate the applicability of our approach and the ability to answer a variety of questions. We discuss the computational complexity aspects of the proposed approach and present some quantitative results in Section 8. In Section 9, we provide an outlook on some of the opportunities and problems related to trace alignment. We discuss related work in Section 10. Finally, Section 11 concludes the paper.

2. Event Logs, Questions, and Answers

Before explaining our approach to align traces, we introduce some basic concepts and present six diagnostic questions we would like to answer using our new analysis approach.

Process mining is impossible without proper event logs. In some applications it may be challenging to extract such logs from a variety of data sources, e.g., databases, flat files, message logs, transaction logs, ERP systems, and document management systems. In other applications, this is straightforward, e.g., when using a BPM system.

The starting point for process mining is the concept of an *event*. An event refers to a *process instance*, sometimes referred to as a case. Each process instance is described by a sequence of events. An event e may have different properties, e.g., a timestamp (e occurred on *February 15th 2011 at 11:29:31 UTC*), resource information (e was executed by *Maria*), activity (e corresponds to a *credit check*), transaction type (e is a *start* event, i.e., the start of the credit check by Maria), and various data elements (e is done for customer *991133* who would like to borrow *5000* euro). All of these properties are optional. The only requirement is that events are ordered (i.e., no explicit timestamp is needed) and that each event belongs to a particular class. For analysis, we need a function that maps any event e into its class $\overline{e}$. In this paper, we assume that each event is classified based on its activity. Hence, for our earlier example $\overline{e} =$*credit check*.

Consider a *process instance* for which n events have been recorded: $e_1, e_2, \ldots e_n$. This process instance has a *trace* $\overline{e_1}, \overline{e_2}, \ldots \overline{e_n}$ associated to it. If we use activity names as a classifier, the trace corresponds to a sequence of activities. However, if we would classify events based on resource information, a trace could correspond to a sequence of person names. In the remainder, we will assume that a trace is a sequence of activities. However, we could also align traces based on the people that worked on them etc.

Using the above assumption, we can define an event log as a multi-set of traces.

Definition 1. Let Σ denote the set of *activities.* Σ^+ is the set of all non-empty finite sequences of activities from Σ. Any $T \in \Sigma^+$ is a possible *trace.* An *event log* $\mathcal{L}$ is a multi-set of traces.

For example, $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and $\mathcal{L} = \{\mathtt{abc}, \mathtt{abc}, \mathtt{abc}, \mathtt{aabbcc}, \mathtt{aabbcc}, \mathtt{ac}\}$. Event log $\mathcal{L}$ describes the traces of 6 process instances. Note that three instances have the same trace `abc` and two instances have trace `aabbcc`. Because different process instances can have the same trace, a log is multi-set of traces rather than a set.

We introduce some standard operations for traces.

Definition 2. Let $T = T(1)T(2)T(3)\ldots T(m) \in \Sigma^+$ be a trace:

- $|T| = m$ denotes the length of the trace T;
- $T(k)$ denotes the k^{th} element of trace T;
- T^k denotes the k length prefix of T, i.e., $T^k = T(1)T(2)\ldots T(k)$; and
- the set of all m-length sequences over the alphabet Σ is denoted by Σ^m.

Given an event log, we would like to answer a variety of diagnostic questions:

1. *What is the most common (likely) process behavior that is executed?:* Given an event log (i.e., multi-set of traces), it would be interesting to know which process components are essential/critical for this process. Such essential components/functions form the backbone of the process and should be conserved. Process re-design/improvement efforts should focus on improving such critical components.
2. *Where do process instances deviate and what do they have in common?:* In practice, there is often a significant gap between what is prescribed or supposed to happen, and what actually happens. There is a need to augment process diagnostics with techniques that can assist in finding

deviations by analyzing raw traces in the event logs. There are many domains/applications where this requirement is felt. Fault diagnosis, anomaly detection, diagnosis of fraudulent insurance claims are some of the applications. Given an event log containing a mix of traces where the system process functioned normally and where it malfunctioned, analyzing these traces to find deviations in malfunctioned/anomalous traces from normal traces would give cues in understanding the cause of malfunction/anomaly.

3. *Are there any common patterns of execution in the traces?:* Analyzing logs at the granularity of an individual event might not always be result yielding as one often loses the context information during such analysis. An analyst would be interested in knowing whether there are any interesting execution (behavioral) patterns in the log. The absence or presence of such patterns may indicate the cause of an anomaly (say for e.g., fraudulent insurance claim) or a security violation or a malfunction.
4. *What are the contexts in which an activity or a set of activities is executed in the event log?:* Dependencies exist between activities in a process and activity executions are expected to happen within certain contexts. There can be short-range and long-range dependencies between activities. Long-range dependencies are difficult to discover. An analyst would be interested in understanding the contexts of execution of activities and/or activity sequences.
5. *What are the process instances that share/capture a desired behavior either exactly or approximately?:* Often in diagnostics, an analyst would be interested in finding process instances that share/comply to a particular desired behavior; the desired behavior can be expressed as a manifestation of some pattern of activity sequences or some complex form (combination) of these patterns. Though temporal logic approaches can assist in addressing this problem to a certain extent by discovering process instances that capture the desired behavior exactly, one might also be interested in discovering process instances that share the desired behavior approximately.
6. *Are there particular patterns (e.g., milestones, concurrent activities etc.) in my process?:* Workflow patterns [12] refer to recurring forms/structures addressing business requirements. For example, milestones indicate specific execution points in the process model and provide a mechanism for supporting the conditional execution of a task or sub-process. An analyst would be interested in discovering the presence of, and in analyzing milestone patterns in the process event log. Similarly, discovery of process models with concurrency is one of the challenging problems in process mining. The presence of concurrent activities creates different permutations of activities in the event log that adds to the complexity of discovery algorithms. Detection of the presence of concurrent activities might also help in pre-processing the logs.

In the remainder we will show that trace alignment significantly contributes to answering these questions.

3. Trace Alignment

In this section, we formally define what trace alignment is and discuss techniques for finding optimal alignments.

Definition 3. Trace alignment over a set of traces $\mathbb{T} = \{T_1, T_2, \ldots, T_n\}$ is defined as a mapping of the set of traces in $\mathbb{T}$ to another set of traces $\overline{\mathbb{T}} = \{\overline{T_1}, \overline{T_2}, \ldots, \overline{T_n}\}$ where each $\overline{T_i} \in (\Sigma \cup \{-\})^+$ for $1 \leq i \leq n$ and

- there is an $m \in \mathbb{N}$ such that $|\overline{T_1}| = |\overline{T_2}| = \ldots = |\overline{T_n}| = m$,
- $\overline{T_i}$ is equal to T_i after removing all gap symbols "$-$",
- there is no $k \in \{1, \ldots, m\}$ such that $\forall_{1 \leq i \leq n}, \overline{T_i}(k) = -$

m in the definition above is the length of the alignment. An alignment over a set of traces can be represented by a rectangular matrix $\mathcal{A} = \{a_{ij}\}$ $(1 \leq i \leq n, 1 \leq j \leq m)$ over $\Sigma' = \Sigma \cup \{-\}$ where "$-$" denotes a gap. The third condition in the definition above implies that no column in $\mathcal{A}$ contains only gaps $(-)$. It is imperative to note that there can be many possible alignments for a given set of traces and that the length of the alignment, m, satisfies the relation $l_{max} \leq m \leq l_{sum}$ where l_{max} is the maximum length of the traces in $\mathbb{T}$ and l_{sum} is the sum of lengths of all the traces in $\mathbb{T}$.

3.1. Pairwise Trace Alignment

Before we get into the details of aligning a set of traces, let us first consider a special case of trace alignment, where the number of traces to align is 2. Aligning a pair of traces is referred to as *pair-wise* trace alignment. Let us consider the example of aligning the two traces $T_1 =$ `abcac` and $T_2 =$ `acacad`. Figure 3 depicts three of the many variants of aligning the two traces.

$\overline{T_1}$	a	b	c	a	c	-	-
$\overline{T_2}$	a	-	c	a	c	a	d

(a)

$\overline{T_1}$	a	b	c	a	c	-
$\overline{T_2}$	a	c	a	c	a	d

(b)

$\overline{T_1}$	a	b	c	a	c	-	-	-	-	-	-
$\overline{T_2}$	-	-	-	-	-	a	c	a	c	a	d

(c)

Figure 3: An example of pairwise trace alignments.

Alignment between a pair of traces, T_1 and T_2 can be considered as a transformation of the trace T_1 to T_2 or viceversa through a set of editing operations applied to one of the traces iteratively. The traces are said to be aligned after the transformation, and can be represented as a rectangular matrix as mentioned earlier. Assuming that T_1 is transformed into T_2, the following edit operations are defined for any column j in the alignment:

- the activity pair $(\mathtt{a}, \mathtt{b})$, $\mathtt{a}, \mathtt{b} \in \Sigma$, denotes the substitution of activity `a` in T_1 with activity `b` of T_2,
- the activity pair $(\mathtt{a}, -)$ denotes the deletion of activity `a` in T_1, and

- the activity pair $(-, \mathtt{b})$ denotes the insertion of activity `b` in T_1.

It is important to note that insertion and deletion operations are complementary in that an insertion in one trace can be considered as a deletion in another trace. Henceforth, we refer to insertion and deletion operations as *indel* operation. *indels* should be sensitive to the context in which the operations are performed [2]. For example, in a repair process, it is ok to have an activity `Analyze Defect` after `Register` but not after `Start Repair` (the defect should first be analyzed before performing the repair). Bose and van der Aalst [2] have considered two notions of context for indels viz., `indelRightGivenLeft` and `indelLeftGivenRight` which indicates the insertion of an activity to the immediate right of another activity or to the immediate left of another activity respectively. One can consider both the left and right contexts as well as larger subsequences of activities rather than an individual activity in defining contexts. However, this increases the computational complexity of finding alignments. We consider `indelRightGivenLeft` as the notion of indels for trace alignment.

In order to avoid edit operations that do not make sense in a certain context, a score or cost function needs to be defined for the substitution or indel operations. The substitution score is a function $S : \Sigma \times \Sigma \rightarrow \mathbb{R}$ where $S(\mathtt{a}, \mathtt{b})$ denotes the score for substitution of activity `a` with activity `b` for all $\mathtt{a}, \mathtt{b} \in \Sigma$. The `indelRightGivenLeft` score is a function $\mathcal{I} : \Sigma \cup \{-\} \times \Sigma \cup \{-\} \rightarrow \mathbb{R}$ where $\mathcal{I}(\mathtt{a}, \mathtt{b})$ denotes the score for inserting or deleting activity `b` given that the left activity is `a` for all $\mathtt{a}, \mathtt{b} \in \Sigma$. $\mathcal{I}(-, \mathtt{a}) = \mathcal{I}(\mathtt{a}, -) = \mathcal{I}(-, -) = 0$ for all $\mathtt{a} \in \Sigma$. Let $\overline{T_1}$ and $\overline{T_2}$ be the aligned traces of T_1 and T_2 and let m be the length of the alignment. It could be the case that the first activity in $\overline{T_1}$ or $\overline{T_2}$ had to be inserted/deleted. The left activity for this case does not exist; so, we define $\overline{T_1}(0) = \overline{T_2}(0) = -$. Given S and $\mathcal{I}$, *the score of a pair-wise alignment can be defined as the sum of the scores of the edit operations across all columns in the alignment.* In other words:

$$Score(\overline{T_1}, \overline{T_2}) = \sum_{j=1}^{m} e_j$$

where

$$e_j = \begin{cases} S(\mathtt{a}, \mathtt{b}) & \text{if } \overline{T_1}(j) = \mathtt{a} \text{ and } \overline{T_2}(j) = \mathtt{b} \\ \mathcal{I}(\mathtt{a}, \mathtt{b}) & \begin{cases} \text{if } \overline{T_1}(j) = \mathtt{b}, \overline{T_1}(j-1) = \mathtt{a} \text{ and } \overline{T_2}(j) = - \text{ or} \\ \text{if } \overline{T_1}(j) = -, \overline{T_2}(j) = \mathtt{b} \text{ and } \overline{T_2}(j-1) = \mathtt{a} \end{cases} \end{cases}$$

Assuming a *unit score* function where a substitution of activity pair $(\mathtt{a}, \mathtt{b})$ is associated with a score of 1 if $\mathtt{a} = \mathtt{b}$ and a score of -1 otherwise, and an indel score function, $\mathcal{I}(\mathtt{a}, \mathtt{b}) = -1$, for all $\mathtt{a}, \mathtt{b} \in \Sigma$, the alignments enumerated in Figure 3 have the scores 1, -4 and -9 respectively. A "best" alignment can be considered to be the one with the maximum score. For the above example, this corresponds to Figure 3(i). Instead, if we define the indel score function as above but change the substitution score function as $S(\mathtt{a},\mathtt{b}) = 2$ if $\mathtt{a} = \mathtt{b}$ and $S(\mathtt{a},\mathtt{b}) = 1$, otherwise, then the alignments enumerated in Figure 3 have the scores 5, 5 and -9 respectively. The maximum alignment score is 5 and there are two best alignments corresponding to Figure 3(i) and Figure 3(ii). As another example, if we define the substitution score as $S(\mathtt{a},\mathtt{b}) = 1$ if $\mathtt{a} = \mathtt{b}$ and $S(\mathtt{a},\mathtt{b}) = -1$, otherwise, and the indel score function as $\mathcal{I}(\mathtt{a},\mathtt{b}) = 1$, for all $\mathtt{a},\mathtt{b} \in \Sigma$, the alignments enumerated in Figure 3 have the scores 7, -2 and 9 respectively. The maximum score is 9 and the best alignment corresponds to Figure 3(iii).

It is imperative to note that the best scoring alignment is sensitive to the substitution and indel score functions. Furthermore, there can be more than one alignment with the maximum score. Substitution of uncorrelated/constrasting activities or insertion/deletion of activities not conforming to a context should be penalized heavily. Similarly, 'like' events should be allowed to be replaced/inserted at a minimal cost [2].

3.2. How to Compute Alignments

Figure 3 depicts just three of the many variants of aligning the two traces T_1 and T_2. In fact, the number of possible alignments for two traces of length l is $\approx (1+\sqrt{2})^{2l+1}l^{-1/2}$ [3], e.g., for two traces of length 100, the number of possible alignments is approximately 10^{77}. Therefore, it is infeasible to enumerate all possible alignments (even for moderate values of l), find their scores, and identify the best alignment.

Needleman and Wunsch [13] have proposed a dynamic programming algorithm for finding the optimal alignment between two (amino acid) sequences. The basic idea is to build up an optimal alignment using previous solutions for optimal alignments of smaller subsequences. Let T_1 and T_2 be two traces. A matrix F indexed by i and j, is constructed where the value $F(i,j)$ is the score of the best alignment between the prefix T_1^i of T_1 and the prefix T_2^j of T_2. $F(i,j)$ is constructed recursively by initializing $F(0,0) = 0$ and then proceeding to fill the matrix from top left to bottom right. It is possible to calculate $F(i,j)$ if $F(i-1,j-1)$, $F(i-1,j)$ and $F(i,j-1)$ are known. There are three possible ways that the best score $F(i,j)$ of an alignment up to T_1^i and T_2^j could be obtained: $T_1(i)$ could be aligned to $T_2(j)$, in which case $F(i,j) = F(i-1,j-1) + S(T_1(i),T_2(j))$; or $T_1(i)$ is aligned to a gap, in which case $F(i,j) = F(i-1,j) + \mathcal{I}(T_1(i-1),T_1(i))$; or $T_2(j)$ is aligned to a gap, in which case $F(i,j) = F(i,j-1) + \mathcal{I}(T_2(j-1),T_2(j))$. The best score up to (i,j) will be the largest of these three options. In other words, we have

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + S(T_1(i),T_2(j)), \\ F(i-1,j) + \mathcal{I}(T_1(i-1),T_1(i)), \\ F(i,j-1) + \mathcal{I}(T_2(j-1),T_2(j)). \end{cases} \tag{1}$$

The values along the top row (when $i = 0$) and left column (when $j = 0$) need to be handled as follows. The values $F(i,0)$ represent alignments of a prefix of T_1 to all gaps in T_2. So, we can define $F(1,0) = 0$ and for $i > 1, F(i,0) = F(i-1,0) + \mathcal{I}(T_1(i-1), T_1(i))$. Similarly, we can define $F(0,j)$. The value in the bottom right cell of the matrix, $F(|T_1|, |T_2|)$, is the best score for an alignment of T_1 and T_2. To find the alignment itself, we must find the path of choices from (1) that led to this best score, i.e., we move from the current cell (i,j) to one of the cells $(i-1,j-1), (i-1,j)$ or $(i,j-1)$ from which the value $F(i,j)$ was derived. While doing so, we add a pair of symbols onto the front of the alignment: $T_1(i)$ and $T_2(j)$ if the step was to $(i-1,j-1)$, $T_1(i)$ and the gap symbol '$-$' if the step was to $(i-1,j)$, or '$-$' and $T_2(j)$ if the step was to $(i,j-1)$. At the end we will reach the start of the matrix, $i = j = 0$. The above procedure, called *traceback*, will retrieve only one of the alignments that gives the best score; there can be cases where multiple options of (1) are equal. In these cases, an arbitrary choice is made. The set of all possible alignments for the best score can be enumerated by using graph traversal techniques.

Let us consider the two traces $T_1 = \texttt{abcac}$ and $T_2 = \texttt{acacad}$ and the unit score function. Figure 4(a) depicts the F matrix for these two traces using the unit score function. Consider the cell at row 4 and column 4, $F(4,4)$; the value for this cell corresponds to the maximum of the score at $F(3,3)$ + sub(c, a) or $F(3,4)$ + del(c,b) or $F(4,3)$ + ins(c, c). Since in the unit score function, the score of substitution of unlike activities and the score for indels is -1, we get $F(4,4) = \max\{-1,-2,0\} = 0$. The best score of the alignment is $F(6,7) = 1$. Figure 4(b) depicts the traceback procedure pertaining to the best alignment. We start with the bottom right cell at $(6,7)$ and identify the cells that led to the best score recursively until we reach the top left cell at (0,0).

3.3. Multiple Trace Alignment

Having discussed the alignment of two traces, let us move on to the alignment of a set of traces. One of the most popular scoring mechanisms for multiple sequence alignment of genomic sequences is the *sum-of-pairs* (SP) method [14, 15]. We adopt the sum-of-pairs method for trace alignment as well. Let $\overline{T_h}$ and $\overline{T_k}$ be two distinct rows extracted from a multiple trace alignment $\mathcal{A}$ (over a set of set of n traces), and let $Score(\overline{T_h}, \overline{T_k})$ be the alignment score calculated in the same way as ordinary pairwise alignment of T_h and T_k, then the SP score of a multiple trace alignment $\mathcal{A}$ is defined as

$$Score_{SP}(\mathcal{A}) = \sum_{1 \leq h < k \leq n} Score(\overline{T_h}, \overline{T_k})$$

It is possible to generalize the pairwise dynamic programming alignment approach to the alignment of n traces. However, it is impractical for more than a few traces. Assuming that the traces are all of roughly the same length l, the space complexity of the multidimensional dynamic programming algorithm is $\mathcal{O}(l^n)$ and the time complexity is $\mathcal{O}(2^n l^n)$ [16]. Multiple sequence alignment that maximizes the SP score was shown to be NP-complete [17]. Since the

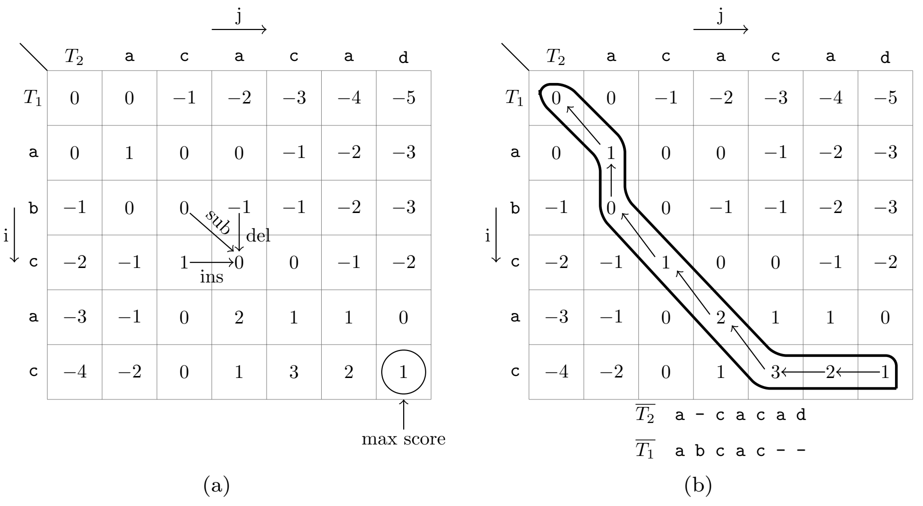


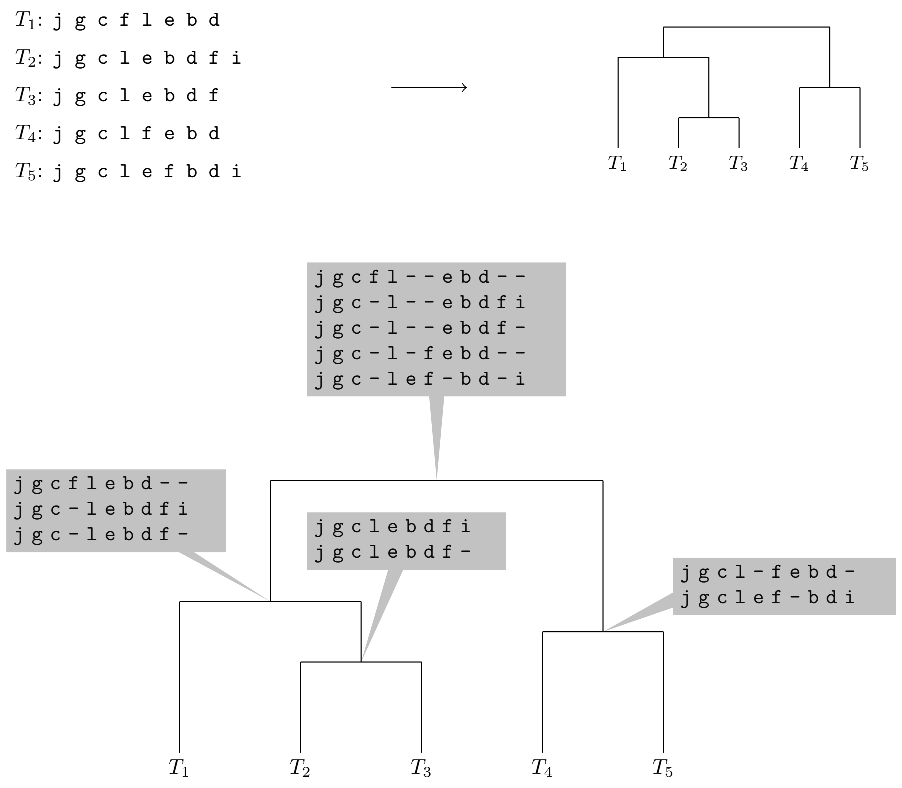
Figure 4: The F-matrix and the traceback computing an alignment between two traces using the unit score function.

computation of optimal multiple sequence alignment is prohibitively expensive, various heuristic algorithms have been proposed in the literature [18].

We adopted the most popular heuristic approach [19], viz., the progressive alignment approach [20, 21] for trace alignment. The basic idea of progressive alignment is to iteratively construct a succession of pairwise alignments. Alignment is allowed between a pair of traces, a trace and an alignment and between alignments. The selection of traces for alignment at each iteration is based on their similarity. Traces that are most similar to each other are aligned first. Once similar traces have been aligned, align the resulting clusters of traces against each other. A guide tree is built to assist this process. We use the agglomerative hierarchical clustering algorithm (AHC) [22] with minimum variance [23] as the join criteria for generating this tree. The choice of AHC is due to the fact that it produces the tree naturally as a dendrogram while the tree has to be constructed subsequently if other clustering algorithms such as k-means is used.

It is important to note that the hierarchy formed using AHC is strongly influenced by the features used to characterize the traces and the distance or similarity metrics used. What constitutes a relevant feature is largely dependent on the domain and the context of analysis [24]. Context-aware and process-centric features defined by common execution patterns are shown to out-perform the bag-of-activities and k-gram features [25]. We use the maximal repeat feature set [25], the Euclidean distance metric and the F-score similarity measure [26] in our experiments.

Figure 5 illustrates an example of the progressive alignment strategy. In this example, we consider 5 traces. A guide tree is generated using AHC. Based on the guide tree, the traces T_2 and T_3 would first be aligned using pairwise trace alignment. Next traces T_4 and T_5 would be aligned using pairwise trace alignment. Subsequently, trace T_1 is aligned with the alignment obtained from T_2 and T_3. Finally the two alignments obtained from the set of traces $\{T_1, T_2, T_3\}$ and $\{T_4, T_5\}$ are aligned.



```
T1: j g c f l e b d
T2: j g c l e b d f i
T3: j g c l e b d f
T4: j g c l f e b d
T5: j g c l e f b d i
```

```
j g c f l - - e b d - -
j g c - l - - e b d f i
j g c - l - - e b d f -
j g c - l - f e b d - -
j g c - l e f - b d - i
```

```
j g c f l e b d - -
j g c - l e b d f i
j g c - l e b d f -
```

```
j g c l e b d f i
j g c l e b d f -
```

```
j g c l - f e b d -
j g c l e f - b d i
```

Figure 5: An example of progressive alignment approach for multiple trace alignment.

While aligning an alignment $\mathcal{A}$, with another alignment $\mathcal{B}$, (1) is modified as

$$F(i,j) = \max \begin{cases} F(i-1, j-1) + \overline{S}(C^i_{\mathcal{A}}, C^j_{\mathcal{B}}), \\ F(i-1, j) + \overline{\mathcal{I}}(C^{i-1}_{\mathcal{A}}, C^i_{\mathcal{A}}), \\ F(i, j-1) + \overline{\mathcal{I}}(C^{j-1}_{\mathcal{B}}, C^j_{\mathcal{B}}). \end{cases} \tag{2}$$

where $\overline{S}(C^i_{\mathcal{A}}, C^j_{\mathcal{B}})$ denotes the score of substituting column i of alignment $\mathcal{A}$ with column j of alignment $\mathcal{B}$ and is defined as

$$\overline{S}(C^i_{\mathcal{A}}, C^j_{\mathcal{B}}) = \sum_{\mathtt{a},\mathtt{b} \in \Sigma} n^i_{\mathcal{A}}(\mathtt{a}) \cdot n^j_{\mathcal{B}}(\mathtt{b}) \cdot S(\mathtt{a}, \mathtt{b}) \tag{3}$$

where $n^i_{\mathcal{X}}(\mathtt{a})$ denotes the frequency (count) of activity $\mathtt{a}$ in column i of alignment $\mathcal{X}$. $\overline{\mathcal{I}}(C^{i-1}_{\mathcal{A}}, C^i_{\mathcal{A}})$ denotes the score of inserting column i in alignment $\mathcal{A}$ given that its left column is $i-1$ and is defined as

$$\overline{\mathcal{I}}(C^{i-1}_{\mathcal{A}}, C^i_{\mathcal{A}}) = \sum_{\mathtt{a},\mathtt{b} \in \Sigma} f^i_{\mathcal{A}}(\mathtt{a}, \mathtt{b}) \cdot \mathcal{I}(\mathtt{a}, \mathtt{b}) \tag{4}$$

where $f^i_{\mathcal{A}}(\mathtt{a}, \mathtt{b})$ is the frequency of activity $\mathtt{b}$ in column i of alignment $\mathcal{A}$ given that its neighboring activity is $\mathtt{a}$ in column $i-1$. The procedure for finding the "best" alignment is similar to that of pairwise alignment.

Note that the guide tree enables the visualization of alignments for different subsets of the traces. The alignment at the root of the tree corresponds to the alignment of all the traces in the event log whereas an alignment at any internal node of the guide tree depicts the alignment corresponding to the traces constituting the leaves of the sub-tree at the node. It is often the case that event logs contain traces capturing different execution behavior of a process and clustering assists in grouping together a coherent set of traces.

4. Framework for Trace Alignment

We propose the framework depicted in Figure 6 for trace alignment. The framework identifies the following parts:

- *Preprocess:* Preprocessing involves steps such as removal of outliers, removal of loop-constructs, abstraction of activities and transformation of log etc. The detection and removal of outliers (explained later in this section) is critical for obtaining interesting alignments.
- *Compute Scoring Matrices:* As discussed in Section 3, alignments are sensitive to the substitution and indel score functions, S and $\mathcal{I}$ respectively. We use the approach presented in [2] for deriving the substitution and indel scores from the event log.
- *Build Guide Tree:* A guide tree assists in progressive alignment of multiple traces as illustrated in Figure 5. We use the agglomerative hierarchical clustering (AHC) approach for building the guide tree. However, other approaches such as neighbor joining [27] can be used.
- *Generate Progressive Alignment:* The progressive alignment approach is used to compute the multiple trace alignment. The guide tree generated in the above step directs the growth of progressive alignment as a series of pairwise alignments.

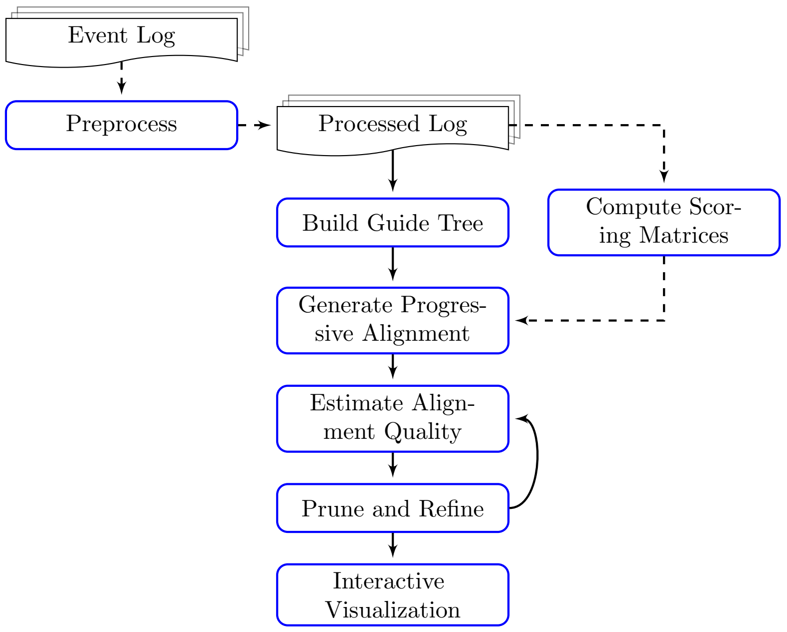


Figure 6: Framework for multiple trace alignment.

- *Estimate Alignment Quality:* Progressive alignment being a heuristic approach, the alignment that is obtained need not be optimal. Furthermore, any error in alignment done in early stages of progressive alignment cannot be undone (cf. advanced alignment techniques in Section 9). Hence it is essential to estimate the quality of an alignment. In this work, we adopt a metric based on the information score as a means for assessing the quality of an alignment. The information score of a column in an alignment is defined as $1-E/E_{max}$, where E is the entropy of activities in the column[1] and E_{max} is the maximum entropy which is equal to $\log_2(|\Sigma|+1)$.

- *Prune and Refine:* Construction of multiple trace alignment is a very complex problem, and most heuristic algorithms usually fail to generate an optimal alignment. Disturbances in an alignment can creep in from many sources thereby making the final alignment far from optimal. Disturbances here refer to the misplacement of gaps in an alignment. Efficient techniques for pruning and refining alignments need to be supported. We will discuss more about this in the next section.

[1]The entropy of a column is defined as $E = \sum_{a\in\Sigma\cup\{-\}} -p_a \log_2(p_a)$ where p_a is the probability of occurrence of a in the column.

- *Interactive Visualization:* Apart from just pictorially depicting the alignment, it is desirable to have additional interactive features for the analysts to explore into the patterns and the alignments uncovered. Features such as editing an alignment, sorting and/or filtering alignment columns based on activities of interest would all lead to gaining further insights into the execution of processes.

Though the definition of what constitutes an outlier is left open, in the current exploration, we have adopted one simple definition of outliers based on the length of the traces. It could be the case that in an event log there are certain process instances whose lengths deviate a lot from the average trace length in the log, e.g., one of the real life event logs that we analyzed has an average trace length of 47 activities (across 223 traces) while there are 5 traces with lengths above 250. Since an alignment is at least as long as the maximum trace length, such outlier traces in the log can lead to an alignment with too many gap symbols. Hence the removal of such traces is important. Note that the definition of outliers can change based on the perspective of analysis. If we are interested in finding common execution patterns or the backbone sequence of a process, the above definition of outliers may work fine. However, if we are interested in finding non-conforming traces or deviations in anomalous traces from normal traces, then the above definition might not always be appropriate. Recent efforts in detecting and dealing with outliers in process mining have been reported in [28, 29]. The efficacy of these techniques in the context of trace alignment needs to be explored.

The presence of loop constructs and the multiple invocations of a process fragment (sub-process) can all lead to variations in the lengths of the traces. Loop constructs and multiple invocations of process fragments manifest as tandem arrays and maximal repeats in the event logs respectively and can be detected efficiently in linear time [30]. One can identify such patterns and define abstractions over them and transform the log to a higher-level [30]. Trace alignment can then be applied on the transformed log.

5. Refining Alignments to Improve Alignment Quality

Variation in the lengths of the traces, the choice of scoring functions used, the method and parameter choices used in the generation of guide tree, strategies used in resolving conflicts during traceback can all lead to disturbances or misalignments. Furthermore, misalignments in earlier stages of progressive alignment percolate to later stages. It is conjectured that detecting such misalignments and refining them might improve the quality of the final alignment. Misalignments are more pronounced in cases where there are recurring patterns of common execution behavior such as the manifestation of loops and multiple invocations of a functionality.

Figure 7 depicts three scenarios of misalignment. Let P be a sequence of activities. P_i^k indicates the k^{th} instance of the manifestation of P in trace T_i. In Figure 7, there are two instances of P in trace T_1 and one instance of P in traces T_2 and T_3. It could be the case that the lone instance of P in trace T_2 is split up and aligned to the two instances of P in T_1 as depicted in Figure 7(i). Figure 7(ii) depicts the scenario where the lone instance of P in T_2 is aligned to the second instance of P (P_1^2) in T_1 where as one would have preferred it to be aligned with P_1^1. Figure 7(iii) depicts an even more undesirable situation where the lone instance of P in T_2 is aligned to the first instance of P in T_1 and the lone instance of P in T_3 gets aligned to the second instance of P in T_1. What is worse is that though P manifests in both T_2 and T_3, they do not get aligned due to their alignment conflict with T_1. In this example, a desirable alignment would be the one where the first instance of P in all the three traces are aligned together.

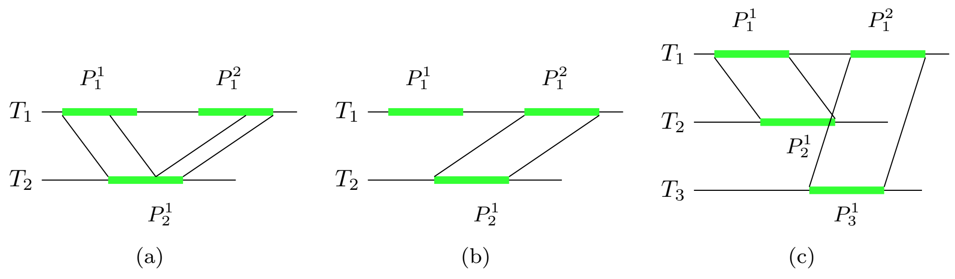


Figure 7: Scenarios of misalignment in the presence of recurring patterns of execution.

In the above examples, we have considered the scenario where the pattern P manifests exactly as is (alike) in all the traces. However, the presence of concurrent activities and/or optional activities in a process might disturb the manifestation of P in a trace. In other words, there could be instances where the concurrent/optional activities are interspersed in the manifestation of P. For example, in the trace $T =$ `jgclfebdklebdklebdi`, the concurrent activity `f` is interspersed in the manifestation of the first instance of the pattern `lebd`. Figure 8 depicts two scenarios of misalignments in the presence of concurrent/optional activities. In Figure 8(i), there exists a sequence of optional activities interleaved in P_2^1. It could be the case that the alignment of T_1 and T_2 results in the prefix of P_2^1 before the optional activities to be aligned with the prefix of P_1^1. The optional activities and the suffix of P_2^1 following the optional activities are aligned with gaps as indicated in Figure 8(i) resulting in an undesirable alignment. Instead, an ideal alignment would be the one that is depicted under 'preferred alignment' in this case. Figure 8(ii) illustrates the existence of a sequence of concurrent activities interspersed in P_2^1. A possible misalignment and the preferred alignment are also depicted in Figure 8(ii). For simplicity reasons, we illustrate the misalignment scenarios with both the traces having a single instance of P in Figure 8. However, one can imagine scenarios where multiple instances of P are manifested with uneven numbers across traces, and with some instances having an interleaved manifestation of optional/concurrent activities. In such cases, the misalignment scenarios depicted in both Figure 7 and Figure 8 are (together) possible.

.

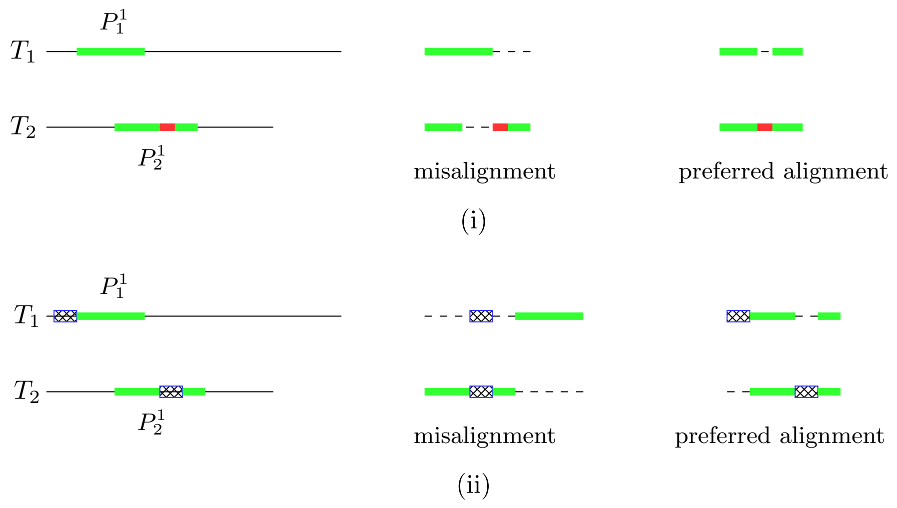


Figure 8: Scenarios of misalignment in the presence of concurrent/optional activities interspersed with recurring patterns of execution.

The information score metric defined earlier is not rich enough to capture such misalignments. Robust metrics to assess the quality of (mis-)alignment are needed. The definition of such quality metrics largely remains an open research topic in this area. We define a pattern based misalignment metric, viz., *misalignment score* in this paper.

Misalignment Score
Let $\mathbb{T} = \{T_1, T_2, \ldots, T_n\}$ be a set of traces and $\overline{\mathbb{T}} = \{\overline{T_1}, \overline{T_2}, \ldots, \overline{T_n}\}$ be the corresponding set of aligned traces of $\mathbb{T}$. Given a pattern P, let $c(P, \overline{T_i})$ denote the frequency of occurrence (considering both the exact and interspersed manifestation) of a pattern P in its corresponding original trace $T_i \in \mathbb{T}$. For any two aligned traces $\overline{T_i}$ and $\overline{T_j} \in \overline{\mathbb{T}}$, let $\overline{T_u} = \arg\min_{\overline{T} \in \{\overline{T_i}, \overline{T_j}\}} c(P, \overline{T})$ and $\overline{T_v} = \arg\max_{\overline{T} \in \{\overline{T_i}, \overline{T_j}\}} c(P, \overline{T})$. In other words, among $\overline{T_i}$ and $\overline{T_j}$, $\overline{T_u}$ represents the one that has the lesser number of instances of the pattern P while $\overline{T_v}$ represents the one with a higher number of instances of P.

Let $\mathrm{PA}(P, k, \overline{T_u}, \overline{T_v})$ denote the set of all instance numbers of pattern P in $\overline{T_v}$ to which some activity in the k^{th} instance of P in $\overline{T_u}$ is aligned to. If the k^{th} instance of P in $\overline{T_u}$ is not aligned to any instance of P in $\overline{T_v}$, i.e., it is aligned to only gaps, then $\text{PA}(P, k, \overline{T_u}, \overline{T_v}) = \{\}$. The misalignment score matrix $\mathcal{M} = \{\text{ms}_{ij}\}$ $(1 \le i, j \le n)$ of a pattern P over the set of aligned traces $\overline{\mathbb{T}}$ is defined as

$$\text{ms}_{ij} = \text{ms}_{ji} = \sum_{k=1}^{c(P,\overline{T_u})} \sum_{r \in \text{PA}(P,k,\overline{T_u},\overline{T_v})} |r - k| + \delta(k)$$

where $\delta(k) = 1$ if any activity in the k^{th} instance of pattern P in $\overline{T_u}$ is aligned to a gap or any other activity not in P in $\overline{T_v}$ and 0 otherwise. $\overline{T_u}$ and $\overline{T_v}$ correspond to that aligned trace with the lesser and higher number of instances of P respectively among $\overline{T_i}$ and $\overline{T_j}$ as defined above. The $|r - k|$ signifies the distance of misalignment. ms_{ij} indicates the degree of misalignment pertaining to all instances of pattern P in $\overline{T_u}$.

Let us look at the misalignment score metric with an example. Consider the two aligned traces

$$\overline{T_1} = \texttt{jgcl----f---ebdklebdi}$$
$$\overline{T_2} = \texttt{jgc-fleb-dklebdklebdi}$$

and the pattern $P =$ `lebd`. Activity `f` is a concurrent activity in the process and is interspersed in P in $\overline{T_1}$. $c(P, \overline{T_1}) = 2$ and $c(P, \overline{T_2}) = 3$ i.e., there are two instances of P in T_1 and three instances of P in T_2. $\overline{T_u} = \overline{T_1}$ and $\overline{T_v} = \overline{T_2}$. $\text{PA}(P, 1, \overline{T_u}, \overline{T_v}) = \{2\}$ and $\text{PA}(P, 2, \overline{T_u}, \overline{T_v}) = \{3\}$ i.e., the first instance of P in $\overline{T_u}$ is partially aligned to the second instance of P in $\overline{T_v}$ and the second instance of P in $\overline{T_u}$ is completely aligned to the third instance of P in $\overline{T_v}$. $\delta(1) = 1$ since the activity `l` in the first instance of P in $\overline{T_u}$ is aligned to a gap while $\delta(2) = 0$. $\text{ms}_{12} = |2 - 1| + \delta(1) + |3 - 2| + \delta(2) = 3$.

As another example, let us consider the pattern $P =$ `lebd` and the two aligned traces

$$\overline{T_3} = \texttt{jgclfebdklebdklebdklebdi}$$
$$\overline{T_4} = \texttt{jgclf-----ebdklebdklebdi}$$

$c(P, \overline{T_3}) = 4$ and $c(P, \overline{T_4}) = 3$ i.e., there are four instances of P in T_3 and three instances of P in T_4. $\overline{T_u} = \overline{T_4}$ and $\overline{T_v} = \overline{T_3}$. $\text{PA}(P, 1, \overline{T_u}, \overline{T_v}) = \{1, 2\}$, $\text{PA}(P, 2, \overline{T_u}, \overline{T_v}) = \{3\}$ and $\text{PA}(P, 3, \overline{T_u}, \overline{T_v}) = \{4\}$ i.e., the first instance of P in $\overline{T_u}$ is partially aligned to the both the first and second instances of P in $\overline{T_v}$ while the second and third instances of P in $\overline{T_u}$ are completely aligned to the third and fourth instances of P in $\overline{T_v}$ respectively. $\delta(1) = \delta(2) = \delta(3) = 0$. $\text{ms}_{34} = |1 - 1| + |2 - 1| + \delta(1) + |3 - 2| + \delta(2) + |4 - 3| + \delta(3) = 3$.

Once the misalignment score matrix is computed for a given alignment, the *cumulative misalignment score* is then defined to be the sum of the elements in the upper/lower triangle of the matrix (since the matrix is symmetric). An objective of a refinement technique would then be to minimize the cumulative misalignment score. In the following subsections, we propose a few techniques of refining alignments.

5.1. Global vs. Semi-global Trace Alignment

The alignment procedure described in Section 3 is also called as *global trace alignment*. Depending on the scoring functions, global trace alignment can sometimes penalize gaps at the beginning and/or end of the traces in the alignment. In order to allow gaps to be inserted at the beginning/end of any trace in an alignment, a variant of the global trace alignment called the *semi-global trace alignment* can be considered. Here the best score of the alignment is defined to be the one that is the maximum in the last row or last column of the F matrix defined in Section 3. Traceback procedure starts from that cell and proceeds until it stops at the first position it reaches in the top row or left column. Gaps can then be inserted in the appropriate trace in the positions subsequent to the maximum value cell in the last row/column and prior to the position it reached in the top row or left column. Figure 9 depicts the difference between global trace alignment and semi-global trace alignment of two traces aligned using the same scoring functions. It is easy to see that the alignment obtained using semi-global alignment is preferable to the one obtained using global-alignment.

$\overline{T_1}$	j	g	c	-	a	h	b	-	-	-	-	f	d
$\overline{T_2}$	j	g	c	f	a	h	b	d	k	a	h	b	d

(a) Global trace alignment

$\overline{T_1}$	j	g	c	-	a	h	b	f	d	-	-	-	-	-
$\overline{T_2}$	j	g	c	f	a	h	b	-	d	k	a	h	b	d

(b) Semi-global trace alignment

Figure 9: An example of global trace alignment and semi-global trace alignment.

In this example, the two traces have a common execution pattern $P =$ `ahbd`. T_1 has one instance of `ahbd` while T_2 has two instances. Global trace alignment leads to the scenario depicted in Figure 7(i) where the lone instance of `ahbd` in T_1 is split up and the two splits are aligned to different instances of `ahbd` in T_2. This problem is mitigated with the semi-global trace alignment as depicted in Figure 9(b). The misalignment score for the pattern P is $\mathrm{ms}_{12} = 1$ for the resulting alignment using the global trace alignment while it is 0 for the one obtained using semi-global trace alignment. We recommend to consider semi-global trace alignment (at any iteration of progressive alignment) in scenarios where the traces to be aligned differ vastly in their lengths (for example, due to the manifestation of loop constructs).

5.2. Block Shift Refinement

Consider the set of traces $\mathcal{L} = \{$`jgcflebdklebdklebdi`, `jgcflebdklebdi`, `jgclfebdklebdklebdi`, `jgclfebdklebdklebdklebdi`, `jgclfebdklebdi`, `jgclebfdklebdklebdi`, `jgclebdfklebdklebdi`, `jgclebdklfebdklebdi`$\}$. Figure 10(i) depicts the alignment of these 8 traces. It could be seen that there exists a common execution pattern `lebd` in $\mathcal{L}$ and that there are different instances of `lebd` among the traces. A concurrent activity `f` has an interleaved manifestation among the traces. The misalignment scenarios depicted in Figure 7 and Figure 8 can be observed in Figure 10(i). For example, the first instance of `lebd` in trace T_5 is split up and aligned with the first and third instances of `lebd` in T_4. The first instance of `lebd` in trace T_2 is aligned with the second instance of `lebd` in T_1. Figure 11(i) depicts the misalignment score matrix of the alignment in Figure 10(i). The cumulative misalignment score is 53.

```
                                                ↓
jgc-f----leb--dkl-ebdklebdi    jgc-f----lebd--kl-ebdklebdi    jgc-f----lebd--klebdklebdi
jgc----------f--l-ebdklebdi    jgc-f----lebd--kl-ebd-----i    jgc-f----lebd--klebd-----i
jgclf-----eb--dkl-ebdklebdi    jgclfebdklebd--kl-ebd-----i    jgclfebdklebd--klebd-----i
jgclfebdkleb--dkl-ebdklebdi    jgclfebdklebd--kl-ebdklebdi    jgclfebdklebd--klebdklebdi
jgcl---------f----ebdklebdi    jgclfebdklebd-------------i    jgclfebdklebd------------i
jgcl------eb-fdkl-ebdklebdi    jgcl-eb------fdkl-ebdklebdi    jgcl-eb------fdklebdklebdi
jgcl------ebdf-kl-ebdklebdi    jgcl-ebd-----f-kl-ebdklebdi    jgcl-ebd-----f-klebdklebdi
jgcl------eb--dklfebdklebdi    jgcl-ebdkl---f----ebdklebdi    jgcl-ebdkl---f---ebdklebdi

            (a)                            (b)                           (c)
```

Figure 10: An example of block shift refinement. (a) Original alignment (b) Block shifted alignment with a gap column (c) Block shifted alignment after the removal of gap column in (b).

	1	2	3	4	5	6	7	8
1	0							
2	2	0						
3	1	2	0					
4	3	4	3	0				
5	3	1	2	4	0			
6	1	2	0	3	2	0		
7	1	2	1	4	2	1	0	
8	1	2	0	3	2	0	1	0

(a) original alignment

	1	2	3	4	5	6	7	8
1	0							
2	0	0						
3	3	2	0					
4	3	2	0	0				
5	2	2	0	0	0			
6	1	1	3	3	2	0		
7	1	1	2	2	1	1	0	
8	2	3	3	2	1	2	1	0

(b) block shifted refined alignment

Figure 11: The misalignment score matrices (only the lower triangle is depicted).

It can be seen that the alignment can be improved by a mere adjustment/shifting of gap blocks. For example, the activity `f` and the first instance of `lebd` in T_2 can be shifted to the left so that `f` gets aligned to `f` of T_1 and `lebd` gets aligned to the first instance of `lebd` in T_1. Algorithm 1 presents the refinement of an alignment by shifting non-gap activities to the left. The basic idea is to consider each trace of the alignment from left to right shifting an activity at a column preceded by a block of gaps to the left most possible column in that trace where that activity manifests in any other trace of the alignment. Figure 10(ii) depicts the intermediary alignment after Step 13 of Algorithm 1. The shifting of activities has resulted in an alignment where there exists a column with only the gap symbol. Such columns can be removed. Figure 10(iii) depicts the refined alignment after block shifts. Figure 11(ii) depicts the misalignment score matrix of the block shifted refined alignment. It can be noticed that the alignment of Figure 10(iii) is better than that of Figure 10(i) (one can see that the common execution pattern `lebd` is much well conserved after refinement). Accordingly, the cumulative misalignment score of the refined alignment is 46 which is less than that of the original alignment.

Algorithm 1 Block Shift Refinement

```
Require: An alignment, $\mathcal{A}$
1: Let $m$ be the length of the alignment
2: Let $\Sigma_j$ denote the set of activities in column $j$ of $\mathcal{A}$.
3: for all aligned traces $\overline{T_i}$ in $\mathcal{A}$ do
4:   for $j = 1$ to $m$ do
5:     if there exists a block of gaps of length $p$ ($p \geq 1$) starting at $j$ in $\overline{T_i}$ then
6:       if there exists a $k$, such that $j \leq k < j + p$ and $\mathcal{A}(i, j+p) \in \Sigma_k$ then
7:         swap $\mathcal{A}(i,k)$ and $\mathcal{A}(i,j+p)$. set $j = k$.
8:       else
9:         set $j = j + p - 1$.
10:      end if
11:    end if
12:  end for
13: end for
14: Remove any column from $\mathcal{A}$ that contains only the gap symbol
```

5.3. Concurrency Pruning and Realignment

Concurrent activities manifest in mutually exclusive traces across different columns in an alignment[2]. The basis for this arises out of the fact that concurrent activities can have different contexts of execution. In the alignment of Figure 10(i), activity `f` is concurrent as it manifests in columns 5, 14, and

[2]This holds true only in scenarios where the concurrent activity is not involved in a loop construct. For cases where concurrent activities are involved in a loop, one can consider a subset of consecutive columns in an alignment demarcating an instance of the loop and then consider the manifestation of activities in that subset.

18 and there exists no trace that has `f` in more than one column. Concurrent activities are one of the primary sources for misalignments. One can improve the quality of an alignment by identifying the presence of concurrent activities and handling them in a special way. Algorithm 2 presents a procedure for pruning concurrent activities and refining an alignment[3]. This algorithm only deals with the specific case of manifestation of concurrent activities in an alignment as defined in Step 2 of Algorithm 2. Dealing with the other scenario where a concurrent activity is aligned with another (potentially concurrent) activity is relatively complex and is beyond the scope of this paper.

Algorithm 2 Concurrency Pruning and Realignment

```
Require: An alignment $\mathcal{A}$
1: Let $\mathcal{C}$ be the set of all concurrent activities in $\mathcal{A}$.
2: Let $\mathcal{C}_I \subseteq \mathcal{C}$ be the set of all concurrent activities that is aligned to either
   only itself or a gap in the alignment.
3: for all $a \in \mathcal{C}_I$ do
4:     Let $\mathcal{A}'$ be the alignment obtained from $\mathcal{A}$ after removing the columns in
       which $a$ manifests
5:     Perform block shift refinement on $\mathcal{A}'$. Let $\mathcal{A}''$ be the refined alignment.
6:     Insert the concurrent activity $a$ as columns at appropriate positions in
       the refined alignment. Let $\mathcal{A}'''$ be the new alignment.
7:     Set $\mathcal{A} = \mathcal{A}'''$.
8: end for
```

Figure 12(i) depicts the alignment obtained from Figure 10(i) after removing the columns where the concurrent activity `f` manifests (Step 4, Algorithm 2). Figure 12(ii) depicts the alignment obtained after block shift refinement on the alignment of Figure 12(i) (Step 5, Algorithm 2). As it can be seen, this alignment is the ideal alignment of the traces without activity `f`. The next step corresponds to the reintroduction of the concurrent activity `f` into the alignment.

Let $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ be a function defined over the set of trace indices, $i \in \mathbb{N}$, and alignment column indices, $j \in \mathbb{N}$, that gives the activity index $k \in \mathbb{N}$ corresponding to the trace T_i at alignment column j provided $\mathcal{A}(i,j) \in \Sigma$ (if $\mathcal{A}(i,j) = -$, then $f(i,j) = -1$). For example, for the alignment in Figure 10(i) $f(1,5) = 4$, because the activity in column 5 of the aligned trace $\overline{T_1}$ is `f` and `f` is the fourth activity in T_1. Similarly, $f(2,17) = 5$ and $f(3,19) = 11$. In the concurrency pruned and block shift refined alignment ($\mathcal{A}''$ in Step 5, Algorithm 2), we insert a column at index j in the alignment with the concurrent activity in all the traces i (and gaps for others) if and only if the activity at $T_i(f(i,j))$ corresponds to the concurrent activity. The resulting alignment after the reintroduction of the

[3]Pruning here refers to deleting the manifestation of the concurrent activity in the aligned traces

```
jgc-----leb-dklebdklebdi     jgclebdklebdklebd-----i     jgcfl-eb-d-kl-ebdklebd-----i
jgc-----------lebdklebdi     jgclebdklebd----------i     jgcfl-eb-d-kl-ebd----------i
jgcl-----eb-dklebdklebdi     jgclebdklebdklebd-----i     jgc-lfeb-d-kl-ebdklebd-----i
jgclebdkleb-dklebdklebdi     jgclebdklebdklebdklebdi     jgc-lfeb-d-kl-ebdklebdklebdi
jgcl-----------ebdklebdi     jgclebdklebd----------i     jgc-lfeb-d-kl-ebd----------i
jgcl-----eb-dklebdklebdi     jgclebdklebdklebd-----i     jgc-l-ebfd-kl-ebdklebd-----i
jgcl-----ebd-klebdklebdi     jgclebdklebdklebd-----i     jgc-l-eb-dfkl-ebdklebd-----i
jgcl-----eb-dklebdklebdi     jgclebdklebdklebd-----i     jgc-l-eb-d-klfebdklebd-----i

          (a)                          (b)                          (c)
```

Figure 12: An example of concurrent activity pruning and realignment. (a) Alignment obtained from Figure 10(a) after removing the columns where concurrent activity 'f' manifests. (b) Alignment obtained after block shift refinement of (a) (c) Alignment obtained after inserting the concurrent activity in (b).

concurrent activity `f` in all the possible traces in the alignment of Figure 12(ii) is as shown in Figure 12(iii). In this refined alignment, the k^{th} instance of `lebd` (if it exists) among the traces are aligned with each other. *The misalignment score matrix for this refined alignment is a matrix of zero's (the cumulative misalignment score is zero)* which is an improvement from that of the original (53) and the block-shifted refined alignment (46). In this log, the patterns are disturbed only by the presence of the concurrent activity `f` and an appropriate handling of the concurrent activity during refinement led to the conservation of the patterns in the alignment.

6. Realization of the Approach

The techniques and framework presented in sections 3, 4 and 5 has been implemented as plugins in ProM 6.0[4]. Figure 13 depicts the framework along with the plugins that realize each step in the framework. Some trivial preprocessing techniques such as the filtering of traces based on their length are available as filter plugins in ProM. The construction of guide tree is enabled by the 'Guide Tree Miner' plugin that implements the approaches presented in [2, 25]. The rest of the steps are handled by the 'Trace Alignment with Guide Tree' plugin.

Figure 14 depicts the screenshot of invoking the guide tree miner plugin. This plugin requires an event log as input. This plugin supports the features defined in [2, 25] and implements the agglomerative hierarchical clustering algorithm for the generation of guide tree. Apart from the guide tree, this plugin generates as output, k event logs (for a chosen number of clusters k), each containing the

[4]ProM is an extensible framework that provides a comprehensive set of tools/plugins for the discovery and analysis of process models from event logs. See `http://www.processmining.org` for more information and to download ProM.

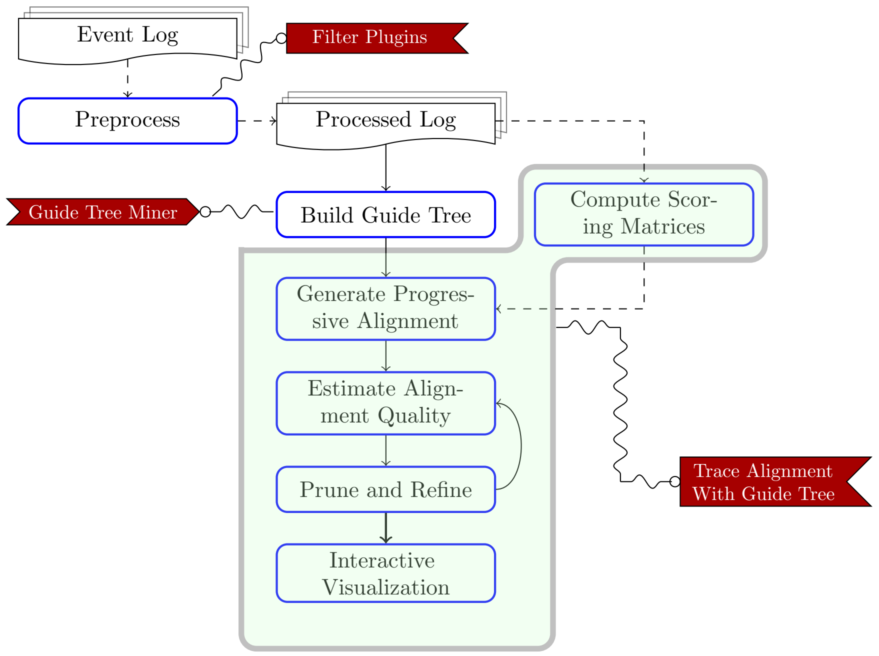
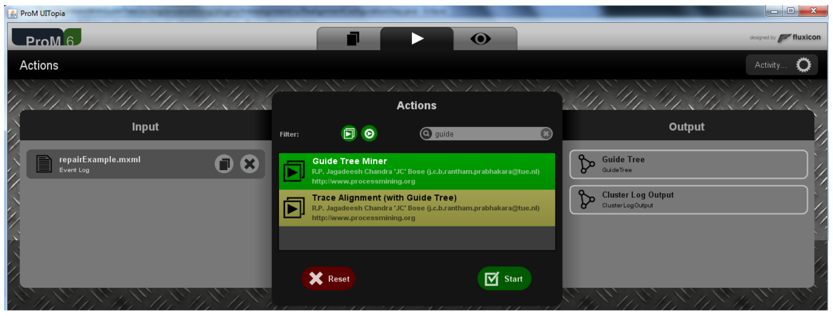


Figure 13: ProM plugins realizing the framework for trace alignment.

partitioned traces pertaining to that cluster (specifying the number of clusters k is optional and is handy in cases where the analyst has apriori knowledge about the diversity in the process/event log). Figure 15 depicts the guide tree generated by this plugin on an event log.



repairExample.mxml
Event Log

Guide Tree Miner
R.P. Jagadeesh Chandra 'JC' Bose (j.c.b.rantham.prabhakara@tue.nl)
http://www.processmining.org

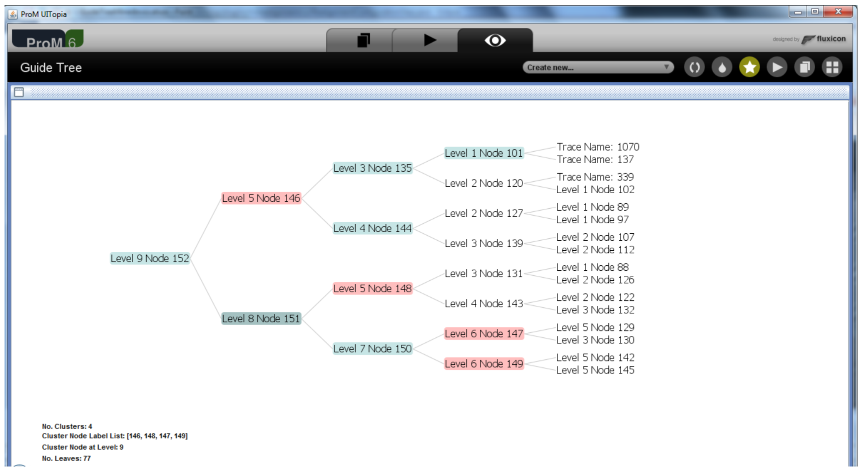
Trace Alignment (with Guide Tree)
R.P. Jagadeesh Chandra 'JC' Bose (j.c.b.rantham.prabhakara@tue.nl)
http://www.processmining.org

Guide Tree
GuideTree

Cluster Log Output
ClusterLogOutput

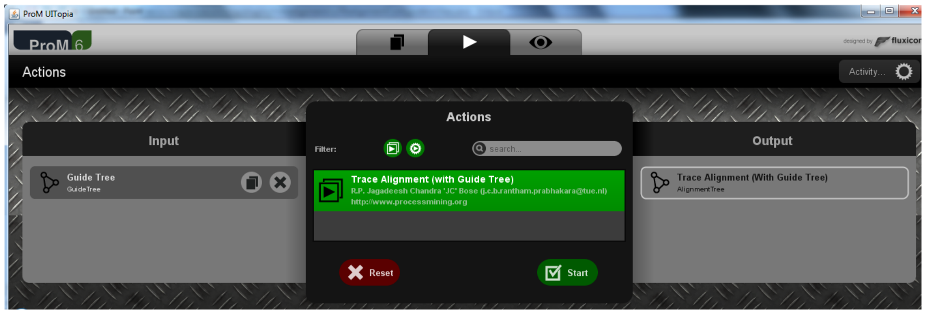
Figure 14: Invocation of the ‘Guide Tree Miner’ plugin in ProM 6.0. The plugin requires an event log as input.

Figure 16 depicts the invocation of the ‘Trace Alignment with Guide Tree’ plugin. This plugin takes as input a guide tree generated by the guide tree miner



```
No. Clusters: 4
Cluster Node Label List: [146, 148, 147, 149]
Cluster Node at Level: 9
No. Leaves: 77
```

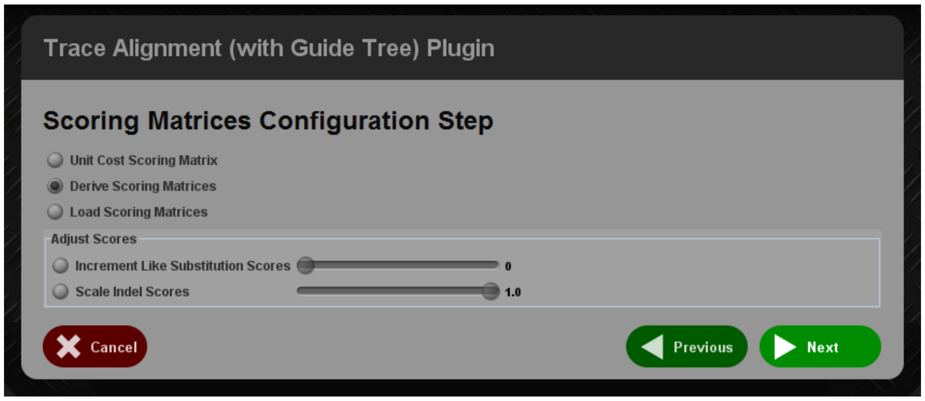
Figure 15: Visualization of the mined Guide Tree. The plugin has been configured to split the log into four clusters (the number of clusters was chosen to be four based on apriori knowledge on the process. This guide tree is for the telephone repair event log mentioned in Section 7). The pink nodes depict the root of the sub-tree corresponding to the four clusters. The tree can be expanded (up to two levels deep) by clicking a non-leaf node. Blue nodes indicate nodes that are expanded.



Trace Alignment (with Guide Tree)
R.P. Jagadeesh Chandra 'JC' Bose (j.c.b.rantham.prabhakara@tue.nl)
http://www.processmining.org

Figure 16: Invocation of the 'Trace Alignment With Guide Tree' plugin in ProM 6.0. The plugin requires a guide tree as input.

plugin. The user can configure the options for generating the substitution/indel scores and refining alignments as illustrated in Figure 17. Figure 18 depicts the visualization of the computed alignments. The alignment visualizer provides rich interactive functionality for the user to explore and gain insights into the process execution. For example, as illustrated in Figure 18, the 'View' menu has options for enabling the sorting or filtering of activities in a column of an alignment and other options pertaining to how an alignment is rendered. The analysis menu provides options for refining alignments as well as for uncovering the potential concurrent activities in the process. Concurrent activities, if any, are then notified in a separate panel. Furthermore, if an analyst wishes to manually refine an alignment, there exists a functionality to do so by editing a


Trace Alignment (with Guide Tree) Plugin

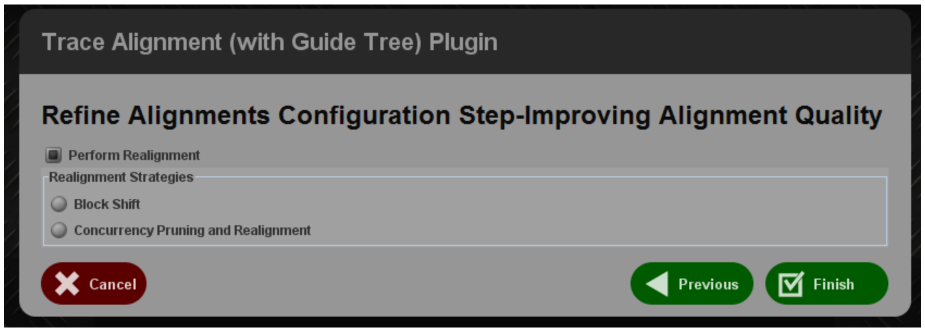
Scoring Matrices Configuration Step

- Unit Cost Scoring Matrix
- [✓ Derive Scoring Matrices]
- Load Scoring Matrices

Adjust Scores

- Increment Like Substitution Scores 0
- Scale Indel Scores 1.0

(a) Scoring matrices configuration step


Trace Alignment (with Guide Tree) Plugin

Refine Alignments Configuration Step-Improving Alignment Quality

- [✓ Perform Realignment]

Realignment Strategies

- Block Shift
- Concurrency Pruning and Realignment

(b) Refine alignments configuration step

Figure 17: A couple of configuration steps of the trace alignment with guide tree plugin.

trace in an alignment as illustrated in Figure 18. For a more detailed explanation of the functionality and usage of the plugins, the reader is referred to the respective plugin's user manual available at the ProM repository.

7. Case Studies and Discussion

We present the results of applying trace alignment on three event logs (one synthetic and two real-life) in the subsequent sections. The synthetic log captures the process of fixing telephone repairs. The goal of using a synthetic log is to show the capabilities of trace alignment on a process for which complete information is known. We also show the applicability of trace alignment and discuss the insights gained on two real-life logs, one from a rental agency and the other from a municipality.

7.1. Telephone Repair Log

The telephone repair event log [31] consists of 1104 traces, of which only 77 traces are distinct. Since duplicate traces add to the complexity of com-

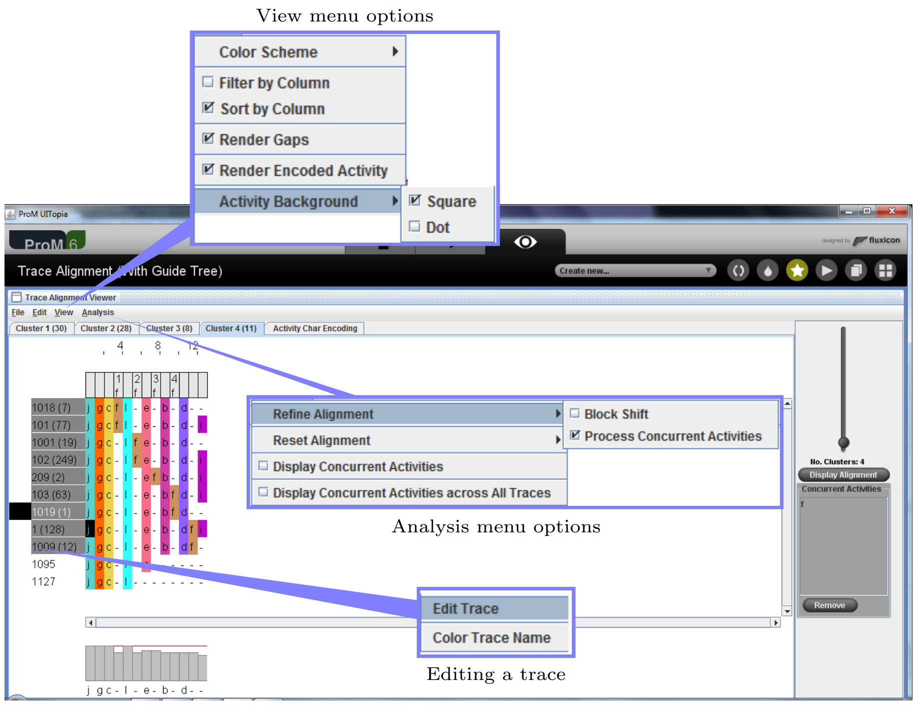


Figure 18: Trace alignment visualization with interactive features

puting an alignment without yielding any additional benefits, we applied the trace alignment on these 77 traces (but at the same time maintain the fact that there exists identical traces in the log). There are 12 event classes viz., Register, Analyze Defect-Start, Analyze Defect-Complete, Repair (Simple)-Start, Repair (Simple)-Complete, Repair (Complex)-Start, Repair (Complex)-Complete, Inform User-Complete, Restart Repair-Complete, Test Repair-Start, Test Repair-Complete and Archive Repair. This is a synthetic log for which complete information about this process is known. Figure 19 depicts the process model discovered using the Alpha mining algorithm [1]. The log consists of cases where the repair can be classified as a simple or complex one. For our discussion here, we further distinguish two types of cases based on the difficulty level of repair viz., cases where the telephone repair was easy and cases where it was difficult (in both simple and complex types). Difficult cases required multiple tries of the repair diagnosis for failing the quality assessment test after a repair. The goal of this example is to show the diagnostics for a *known* process.

As mentioned earlier, the guide tree inherently captures the notion of clustering. We have split the event log into four clusters (using the maximal repeat feature set [25] and the F-score [26] as a measure of similarity; Using this feature set and similarity metric, we were able to obtain the four clusters mentioned

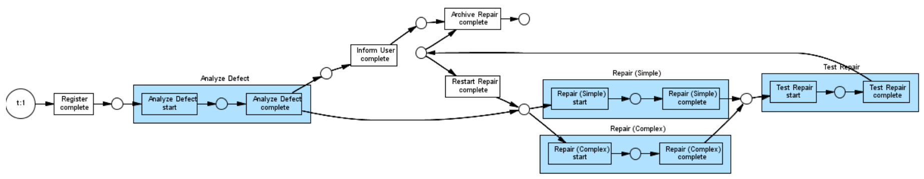


Figure 19: Process model discovered using the Alpha miner in ProM 5.2 for the telephone repair log

above) and Figure 20 depicts the trace alignment for one of the four clusters. Figure 20(i) depicts the original alignment whereas Figure 20(ii) depicts the alignment obtained after the refinement using the concurrent activity pruning and realignment technique (activity `f` is concurrent in this example). This cluster corresponds to traces where the difficulty level was easy and a complex repair procedure was done to fix the problem. The length of the alignment is 14 for this cluster. The left panel depicts the process instance identifier (as in the log) and identifiers with a grey background indicate traces that have identical duplicates. The number within parenthesis indicate the number of duplicates. For example there are 7 traces identical to process instance 1018 (corresponding to the trace `jgcflebd` in the event log) while there are no identical traces for the process instance 1127[5]. The top panel depicts a sorting component where the traces involved in the alignment can be sorted based on the activities in a column and the number in the column indicates the priority of sorting. For example in Figure 20(i), the traces are sorted based on activity `f` (which indicates the `inform user` activity) with traces having `f` in column 4 having first priority and then with those having `f` in column 7 and finally with those having `f` in column 11. The bottom panel depicts the information score metric for each column as well as a consensus sequence for the alignment. *The consensus sequence captures the major activity in each column and can be considered as a back-bone sequence for the process.* The consensus sequence for this example corresponds to `jgclebd` with an optional execution of activity `i` at the end. Columns with an information score of 1.0 indicate well conserved patterns. For example in this alignment, the columns $1-3$ depicting the encoded activity sequence `jgc` (corresponding to activities `Register-complete`, `Analyze (Defect)-start` and `Analyze (Defect)-complete`) is well conserved and appears in all the traces as the beginning subsequence.

It is obvious to see that the encoded activity `f` corresponding to `Inform User - complete` is a concurrent activity. *Concurrent activity manifests in mutually exclusive traces across different columns in the alignment.* The encoded activities `l`, `e`, `b`, `d` and `i` correspond to `Repair Complex-start`, `Repair Complex-complete`,

[5]the number of identical traces is used when computing the scores during the alignment; substitution/indels are accounted for m times if there are m identical traces

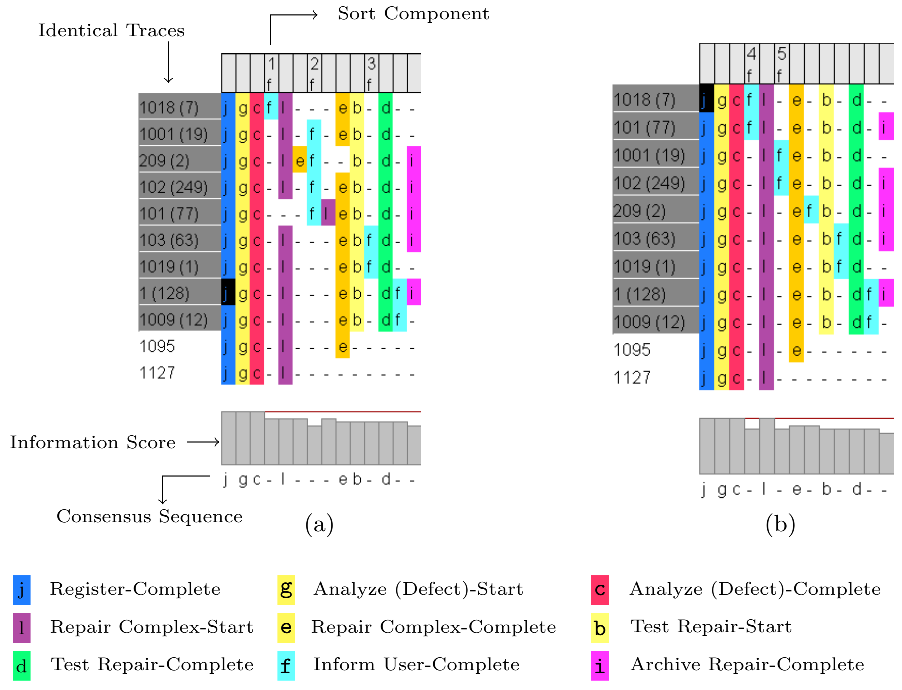
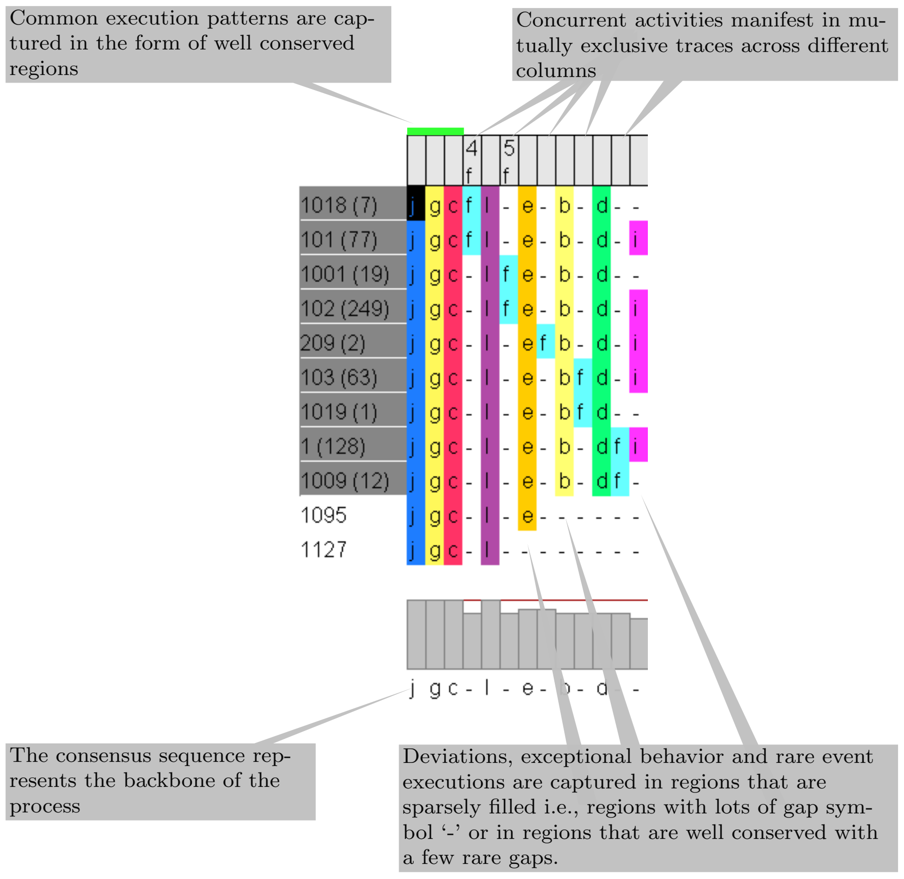


Figure 20: Trace alignment of traces in telephone repair event log for one of the clusters. (i) Original alignment (ii) Concurrency pruned and refined alignment.

`Test Repair-start`, `Test Repair-complete` and `Archive Repair-complete` respectively. Annotating the traces with additional information such as performance metrics, customer feedback etc. over the alignment might give further insights. For example, let us assume that the customer was not happy for the cases 1 and 1009, it is obvious to see that these traces differ from the rest in that the activity `f` appears quite late in these traces. It could be inferred that these customers were not timely informed about the status of their complaint and thus were not satisfied.

Trace alignment provides a complete perspective of activity executions in a log including that of long range dependencies (any dependencies between activities are reflected as common execution patterns in the traces where they manifest). Furthermore, with rich interactive visualization (such as the options of filtering columns containing an activity), trace alignment enables a flexible inspection of the log. In this fashion, trace alignment can assist in uncovering extremely interesting insights and act as probes when analyzing process execution behavior. Figure 21 summarizes how trace alignment can assist in answering some of the diagnostic questions raised in Section 2.

The rest of the 3 clusters for this event log corresponded to the following difficulty level and repair type categories: *easy* and *simple*, *difficult* and *complex*, and *difficult* and *simple/complex* where the last cluster pertained to cases where a simple repair procedure was first tried and finally a complex repair procedure was done to fix the problem.


Common execution patterns are captured in the form of well conserved regions
Concurrent activities manifest in mutually exclusive traces across different columns
4 f
5 f
The consensus sequence represents the backbone of the process
Deviations, exceptional behavior and rare event executions are captured in regions that are sparsely filled i.e., regions with lots of gap symbol '-' or in regions that are well conserved with a few rare gaps.

```
1018 (7)   j g c f l - e - b - d - -
101 (77)   j g c f l - e - b - d - i
1001 (19)  j g c - l f e - b - d - -
102 (249)  j g c - l f e - b - d - i
209 (2)    j g c - l - e f b - d - i
103 (63)   j g c - l - e - b f d - i
1019 (1)   j g c - l - e - b f d - -
1 (128)    j g c - l - e - b - d f i
1009 (12)  j g c - l - e - b - d f -
1095       j g c - l - e - - - - - -
1127       j g c - l - - - - - - - -

           j g c - l - e - b - d - -
```

Figure 21: Aligned traces for one of the clusters of the telephone repair log with annotations indicating how trace alignment can assist in answering some of the diagnostic questions.

7.2. Rental Agency Log

We have applied trace alignment on a real life log of a rental agency where the cases corresponded to the cancellation of a current rental agreement and subsequent registration of a new rental agreement. This log was provided by a large Dutch agency that rents houses and apartments (the organization has approximately 1000 employees and handles 80, 000 houses). There are 74 event classes, one event type (transaction type, complete), 210 process instances and 6100 events in this log. As we can see, this is a reasonably complex log in terms of the size of the alphabet and the number of cases. This process is less structured and is not well understood (Figure A.25(a) depicts the heuristic net and Figure A.25(b) depicts the dotted chart mined using the log. It is not easy to unravel the mined results and provide answers to some of the questions raised in this paper). We now show how trace alignment can assist in uncovering interesting insights from this log. The traces are first encoded into activity sequences where each activity is encoded as a two character sequence.

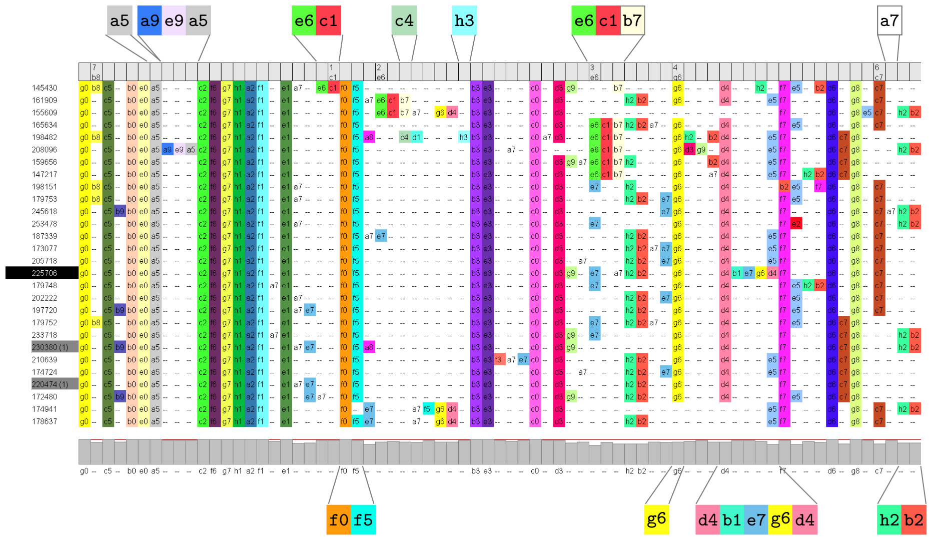


Figure 22: Trace alignment of the traces in one of the clusters of the rental agency log.

Figure 22 depicts the alignment for one of the four clusters of this log. Since the whole alignment is not legible[6], we highlight the interesting patterns/activities (that we refer to for our further discussion) at the top and the bottom of the figure. The length of the alignment is 71. At the outset, we can see certain common execution patterns in the form of well conserved regions (columns) in the alignment. For example, the activity sequence `b0e0a5` (at columns $5-7$) corresponding to the activities, `planning of first inspection`, `preparation of lease termination form`, and `is first inspection performed?` respectively is common across all the traces. *Deviations, exceptional behavior and rare event executions are captured in regions that are sparsely filled i.e., regions with lot of gap symbols* $(-)$. We will present the results of analysis of some of these devia-

[6]The actual alignment can be inspected at `www.win.tue.nl/~jcbose/MiningAlignmentAnalysis.png`

tions. It could be seen that only one of the traces (sixth trace in the alignment) has the activity subsequence `a9e9a5` in columns $8-10$. Activity `a5` in column 7 corresponds to the check, `is first inspection performed?` and the activity subsequence `a9e9a5` corresponds to the scenario where the result of the check was negative due to the fact that the tenant was not at home. `a9` corresponds to the activity of sending a letter to the tenant and `e9` corresponds to the activity of rescheduling the first inspection.

The activity sequence `f0f5` corresponding to the checks `is final inspection done?` (`f0`), and `are there new/repaired defects?` (`f5`) is well conserved across all but one of the traces . We see an exceptional activity sequence `e6c1` corresponding to the offering of a flat in one of the traces (first trace) before the activity sequence `f0f5`. It is strange that a flat was offered before the final inspection was done as in all the other traces where the flat was offered, it happened subsequent to the final inspection. Upon further inspection, we observed that though the flat was offered, the actual registration/check of the candidate corresponding to activity `b7` happened subsequent to the final inspection. Furthermore, in all the cases where the flat was offered and the candidate registered, the activity sequence `d6c1b7` was well conserved except for the first trace. Trace alignment helps us uncover such anomalies and deviations. Similarly, we notice that in only one of traces (fifth trace) there was a need for second inspection (activity `c4` corresponds to the planning of second inspection and `h3` corresponds to the check, `is second inspection done?`).

The activity `e7` corresponds to the determination of a candidate tenant and the activities `g6` and `d4` correspond to registration of lease, and getting the payment and signing of contract respectively. It could also be observed from the alignment that there is an exceptional behavior in one of the cases where we see a manifestation of the activity subsequence `g6d4b1e7g6d4` subsequent to activity `e7` (the activity `e7` appears twice in this trace). This indicates the fact that for this case, there was a need for determining the candidate tenant twice. The determination of the second candidate tenant followed the activity `b1` which corresponds to the termination of provisional lease (the provisional lease was terminated due to nonpayment by the tenant).

Furthermore, the activities `h2b2` corresponding to the `drafting of final note` and `archiving of lease termination` is concurrent in this process (as is evident from the fact that they manifest in different columns across mutually exclusive rows in the alignment). Figure 23 summarizes how trace alignment can assist in answering some of the diagnostic questions. The alignments pertaining to the other clusters also exhibited similar behavior.

7.3. Municipality Log

We have applied trace alignment on another real life log pertaining to one of the Dutch municipalities where the cases correspond to the processing of building permit requests. This process is less structured and is not well understood.

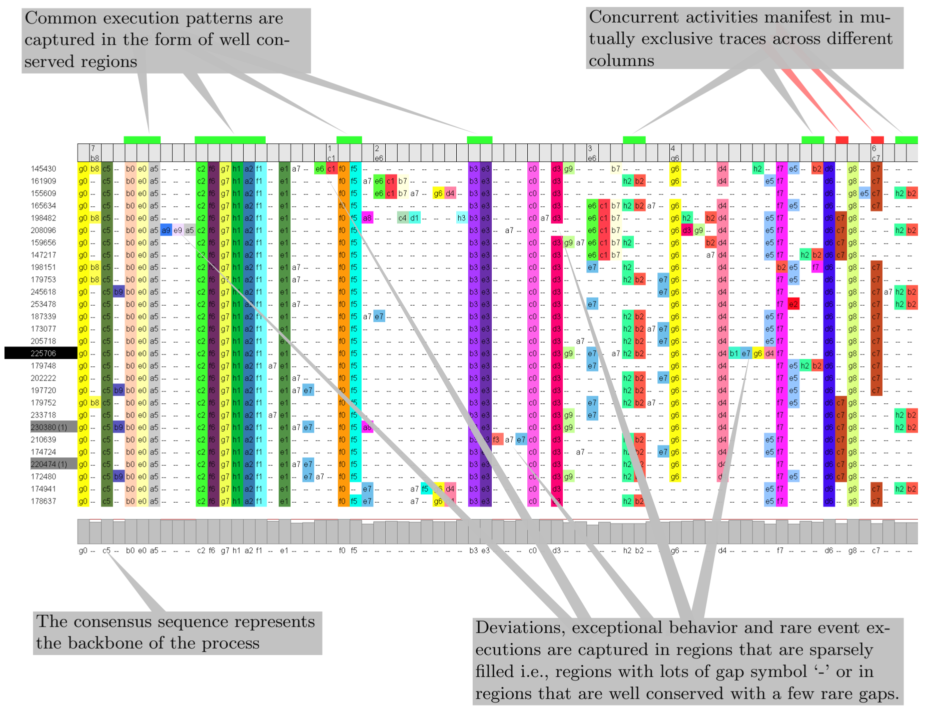

Common execution patterns are captured in the form of well conserved regions
Concurrent activities manifest in mutually exclusive traces across different columns
The consensus sequence represents the backbone of the process
Deviations, exceptional behavior and rare event executions are captured in regions that are sparsely filled i.e., regions with lots of gap symbol '-' or in regions that are well conserved with a few rare gaps.

Figure 23: Aligned traces for one of the clusters of the rental agency log with annotations indicating how trace alignment can assist in answering some of the diagnostic questions.

There are 45 event classes, one event type, 2075 process instances, 11582 events and 71 originators in this event log. As we can see, this is a complex log in terms of the size of the alphabet and the number of cases (Figure A.26(a) depicts the heuristic net and Figure A.26(b) depicts the dotted chart mined using the log. The answers to questions raised in this paper is not explicit using the mined results.). The event log is first encoded into traces where each activity is encoded with a unique character. Though there are 2075 process instances in the event log, when encoded as traces, there are only 451 unique traces.

Figure 24 depicts the alignment for one of the six clusters of this log. This cluster has 51 traces. We can see certain common execution patterns such as `yfj`, `St` and `DCR` among the traces. All the cases start with `y` corresponding to the activity `Case Start`. Each case is then processed by three different departments viz., `L` (fire department), `h` (environment department) and `c` (CCT) to verify for the admissibility of the building permit. It is to be noted that these three activities are concurrent as they manifest across different columns in mutually exclusive rows.

Deviations, exceptional behavior and rare event executions are captured in re-

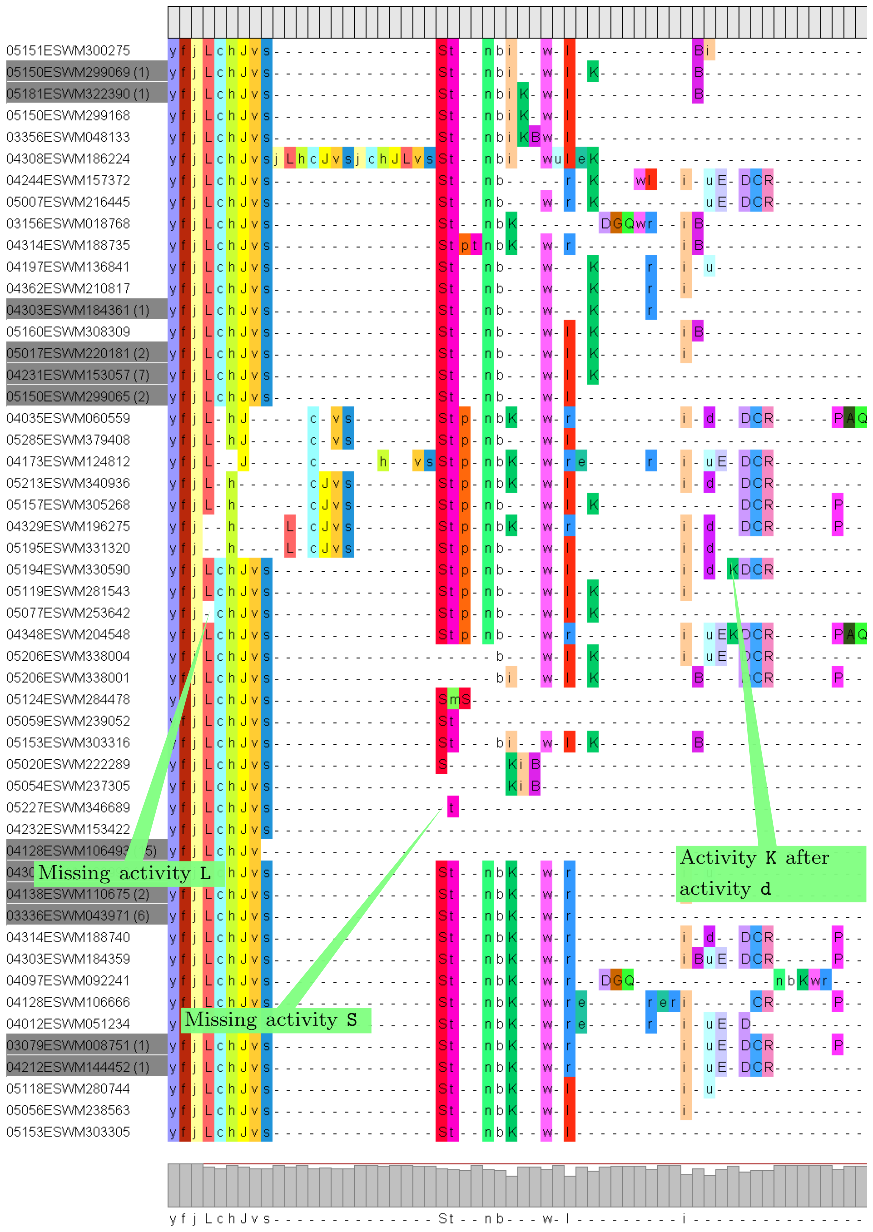


Figure 24: Trace alignment of the traces in one of the clusters of the municipality log

gions that are sparsely filled i.e., regions with lots of gap symbol '-' or in regions that are well conserved with a few rare gaps. We can notice that in one of the traces, the activity L is missing indicating that the case was not handled by the fire department which is a violation of the normal behavior. In another trace (sixth trace from the top), we notice that the case was referred to these departments multiple times (three times). Once the responses from the three departments are obtained, the contents are verified (activity `S`) and a decision is prepared (activity `t`). We see an exception to this behavior in one of the traces where the decision is prepared without the verification (activity `t` manifests without `S`). Furthermore, in some cases, an optional legal decision is also taken (activity `p`).

Activity `B` corresponds to the decision that no construction is permitted for one year whereas activity `d` corresponds to the rejection of the request. We see an exceptional behavior in one of the cases that has been rejected–activity `K` which corresponds to taxes occurs after the rejection `d` whereas in all the other cases where it had been rejected either `K` is not present or if present, `K` always occurs before `d`. In this fashion, trace alignment can assist in uncovering extremely interesting insights.

The diagnostic question, *what are the process instances that share/capture a desired behavior either exactly or approximately?*, can be answered using trace alignment by formulating the desired behavior as an activity sequence and applying trace alignment of this sequence with the traces in the log. Traces that share the desired behavior have a lot of their activities aligned with that of the activities in the desired behavior sequence.

8. Computational Complexity

The major computation steps in the proposed approach pertains to the computation of guide tree and the progressive alignment of traces. As discussed earlier, we use the agglomerative hierarchical clustering algorithm to build the guide tree, which can be informally described as comprising of the following steps:

- Step 1. Determine all inter-object dissimilarities i.e., determine the distance between every pair of traces
- Step 2. Form cluster from two closest traces or clusters
- Step 3. Redefine dissimilarities between new cluster and other traces or clusters (all other inter-object dissimilarities remain unchanged).
- Step 4. Return to Step 2 until all traces are in one cluster.

The hierarchy involves the construction of $n-1$ clusters, and so $n-1$ iterations (Steps 2, 3, and 4). Step 1 above requires $\mathcal{O}(n^2)$ calculations, i.e., $n(n-1)/2)$ inter-trace dissimilarities, where n is the number of traces. The distance between the traces can be computed either by defining some features and transforming each trace into a vector space defined by the features or by using syntactic edit distance approaches. We use the vector-space approach to compute the distances because edit distance approaches are time consuming in that it takes quadratic time (with respect to the length of the traces) to compute the distance between a pair of traces. In the vector-space approach, the number of variables (features) obviously effects the calculation time required, but they are usually considered constant for any set of data. A naive attempt at Step 2 will make the algorithm $\mathcal{O}(n^3)$ complex as the minimum of an n^2 matrix must be found in each of the $n-1$ iterations. However, this can be reduced to $\mathcal{O}(n^2)$ by maintaining a pointer to the minimum value in each row of the matrix. Step 3 can be carried out in $\mathcal{O}(n)$ time using the Lance–Williams combinatorial formula [32, 33]. In summary, the guide tree can be constructed in $\mathcal{O}(n^2)$ time. In all our case studies, we have used the combination of $3-$gram and maximal repeats [25] as the feature set. The choice of this feature set is primarily because they are context-aware and process-centric and is shown to perform better when compared to other approaches [2, 25]. We split the overall time to build the guide tree into the time to (a) extract the features (b) compute the inter-trace distance using the features (Step 1) and (c) form the hierarchy i.e., grouping objects (Steps 2, 3, and 4). Table 1 depicts the computational time (on an i3 Core CPU M350 @ 2.27 GHz with 4GB RAM running a 64-bit Windows 7 OS) required to compute the guide tree for the three case studies. As can be seen, the feature extraction and the computation of inter-trace dissimilarities are the major time consuming portions and the grouping of objects is just a fraction of these. One can slightly reduce the inter-trace dissimilarity computation time by filtering some features that are insignificant (based on frequency etc.).

The time and space complexity of a single iteration of progressive alignment is $\mathcal{O}(l^2 + nl)$ [26] where l is the average length of the traces/profiles considered for alignment and n is the number of traces involved in the alignment. There exists $n-1$ iterations in the progressive alignment thus making the alignment time to be polynomial. However, it is to be noted that during the initial iterations, the value of n is much smaller than the total number of traces. Furthermore, unless the traces to align are heterogeneous the length of the aligned traces tends to be closer to the average trace length of the input traces. Table 1 depicts the computational time required to align the traces for the three case studies. It can be seen that the overall alignment can be performed in manageable times even for large datasets.

9. Outlook

Finding good quality alignments is notoriously complex. In this section, we highlight some of the extensions that can be done to further improve the results of trace alignment. We also mention some of the challenges related to trace alignment and directions for future research.

- *Multi-phase approach:* In order to deal with complex event logs (logs with a large number of activities/event classes etc.), one can try to find align-

Log	No. Unique Traces	No. Features	Guide Tree Using AHC			Progressive Alignment
			Feature Extraction (msecs)	Inter-object distance (msecs)	grouping objects (msecs)	(msecs)
Repair Example	77	167	1158 ± 308	70 ± 9	31 ± 14	352 ± 35
Rental Agency	210	1415	1542 ± 450	252 ± 21	78 ± 0	3865 ± 243
Municipality	451	1234	1209 ± 155	1018 ± 43	284 ± 17	5297 ± 400

Table 1: Computational time required to build the guide tree and align the traces. The notation used for time is $t \pm \delta$ where t is the mean time over 5 different runs and δ is the 95% confidence interval.

ments in a multi-phase approach. The basic idea is to first simplify the log by defining abstractions over the activities and then transforming the log with these abstractions. Sub-logs are created for each abstract activity during this transformation. Trace alignment is then applied on the most abstract log with a provision to zoom into an abstract activity. Upon zooming into an abstract activity, an alignment obtained on the traces defined in the sub-log of the abstract activity is shown. The definition of abstractions and simplification of logs can be enabled by incorporating semantics in the log specification [31, 34] or through (semi-)automated means as in [30, 35].

- *Outlier/noise detection:* Noisy data pose a risk of misleading the alignment procedure and thereby resulting in low quality alignments. There is a need for techniques to identify difficult or even un-alignable sets of traces. For certain perspectives of analysis such as finding common execution patterns, backbone/critical elements of a process, such traces can be safely ignored. However, for certain other perspectives such as finding deviations/non-conforming traces, it is a tricky proposition to filter outliers as non-conforming traces could be treated as outliers. Efficient techniques for noise detection and filtering and identifying coherency in event logs need to be explored and one can take leverage of the techniques in data mining and machine learning community and adapt them (if need be) by providing a process tinge to them.

- *Refinement strategies to improve the quality of an alignment:* In this paper, we have presented three different refinement techniques to deal with misplaced gaps. However, robust quality metrics along with realignment strategies to effectively deal with the variations that might arise due to the manifestation of different process model constructs is highly desirable. Defining and identifying problematic regions in an alignment is a prerequisite to any refinement technique. Automated means of identifying such problematic regions is non-trivial and is in itself an interesting area of research.

- *Advanced alignment techniques:* The progressive alignment approach adopted in this paper is susceptible to converging at local optima. This is due to the fact that when aligning an individual trace with an alignment, the trace is not aligned directly and optimally with every internal trace within the alignment but with the whole alignment taken as an atomic entity. Gaps in the resulting alignment are inserted into the sequence, or as a column into the sub-alignment as a whole, but are not optimally distributed throughout the resultant alignment. As discussed in Section 5, the splitting up of common patterns of execution is an effect of such local optima. Recent advances in multiple sequence alignment such as the one that takes into consideration some user defined constraints [36] while performing an alignment need to be explored. One can specify the conservation of common execution patterns in the final alignment as a constraint. Such common execution patterns can be discovered automatically [30]. Other techniques such as the partial-order alignment [37] and the hybrid approach combining the progressive and partial-order alignments [38] have been shown to produce better alignments.

10. Related Work

Trace alignment is a log visualization technique that assists in uncovering interesting insights in process executions. Dotted chart analysis [11] is one of the most commonly used log visualization technique. Dotted chart analysis, analogous to Gantt charts, presents a "helicopter view" of the event log and assists in analyzing process performance by depicting process events in a graphical way and primarily focuses on the time dimension of events. The dotted chart analysis computes some metrics for performance such as the minimum, maximum and average interval between events. A business analyst needs to *manually* investigate the dotted chart to identify any potential performance issues. For logs with medium to large number of activities (of the order of a few tens to hundreds), the manual inspection and comprehension of the dotted chart becomes cumbersome and often infeasible to identify interesting patterns. Trace alignment alleviates this problem, by finding those patterns automatically and presenting them to the user. In the parlance of dotted chart analysis, trace alignment considers the *logical relative* time perspective of the event log. Furthermore, it would be simple and a natural extension to project the performance metrics proposed in [11] onto the aligned traces.

Stream scope visualization [39] is a trace visualization technique that is based on the event class correlations. Using stream scope visualization, patterns of co-occurring events can be easily recognized by their vicinity. However, stream scope visualization is restricted in that it visualizes each trace separately and does not provide a holistic view of the event log. In contrast, trace alignment enables the visualization of multiple traces at a time and is able to uncover common execution patterns within and across traces.

One of the applications of trace alignment is in uncovering deviations between anomalous and normative traces. Conformance checking aims at detecting inconsistencies/deviations between a process model (that captures the expected behavior) and its corresponding execution log [40]. Conformance checking has inherent limitations in its applicability especially for diagnostic purposes. Firstly, it assumes the existence of a process model. However, in reality, process models are either not present or if present are incorrect or outdated (their quality typically leaves much to be desired). One can argue that process models can be discovered from the event logs and conformance checking be applied on the discovered models. However, this approach is not suitable for the analysis of highly complex and/or flexible processes, the class of models which most of the real-life logs fall into and where the discovered models are “spaghetti-like”. Even in cases where the process models are available , it is difficult to look inside of the processes to identify and locate problems especially with models that are large. Trace alignment is complementary to this approach in that it highlights the deviations by analyzing the raw event traces (avoiding the need for process models).

Trace alignment is largely inspired from Multiple Sequence Alignment (MSA) [5]. However there are challenges in adapting these techniques for trace alignment. Alignment of biological sequences typically happen over homologous sequences and with less variation in length [18]. However, traces in an event log in process mining need not be from a coherent set of cases and can be of different lengths. Variation in lengths can occur due to variation in execution paths of the instances and due to manifestation of process model constructs such as choice/loop constructs. In biological sequence alignment, there are standard scoring matrices for substitution that are derived based on physio-chemical properties of the amino acids. Insertion/deletion operations are primarily considered either with a constant gap-score (or penalty) or as an affine function. In contrast, indel operation in trace alignment is context sensitive (insertion/deletion of an activity given its left activity, `indelRightGivenLeft`). Scoring matrices for trace alignment need to be derived automatically from the event log or provided by the domain experts. Furthermore, biological sequences deal with an alphabet size of either 4 (for four nucleic acids) or 20 (for amino acids). However, the number of distinct activities (event classes) in a typical process mining log can be of the order of a few hundreds. This adds to the complexity of deriving good scoring matrices and aligning traces. We took inspiration from MSA techniques [20, 21, 41] and adapted them for trace alignment.

The quality of the final alignment over a set of traces largely depends on the order in which the progressive alignment is performed. We have used the agglomerative hierarchical clustering to group traces so that homogenous traces are aligned in the early iterations of the progressive alignment. Context-aware approaches to trace clustering are shown to form more coherent clusters [2, 25, 24]. However, the choice of the feature set largely depends on the domain and the context of analysis [24]. Similarly, the resulting clusters also depend on the choice of the distance/similarity metric e.g., if the input data have elliptical distributions, then the Mahalanobis distance [42] is preferred over Euclidean distance. The influence of a distance/similarity metric on the formed clusters and thereby on trace alignment is an interesting topic that needs further research and is beyond the scope of this paper.

11. Conclusions

In this paper, we proposed a novel approach of aligning traces and showed that this approach uncovers interesting patterns and assists in getting better insights on process execution. We have listed some of the interesting questions in process diagnostics and showed how trace alignment can help in diagnostic efforts. This paper extends the work presented in [43]. In this extended paper, we discussed the various scenarios of misalignment and their impact on the resulting alignment. A metric that measures the degree of misalignment and three refinement techniques to cope with misalignments have been proposed. The approach has been implemented in ProM and evaluated using three case studies. This is just a first step in this direction. Due to the computational complexity of multiple trace alignment, automatic generation of high-quality alignments is still challenging and there is much to be done to fully exploit the potential of this approach. Moreover, to further validate the approach we plan to conduct extensive case studies based on real-life event logs.

Acknowledgments R. P. J. C. Bose and W. M. P. van der Aalst are grateful to Philips Healthcare for funding the research in process mining.

References

[1] W. M. P. van der Aalst, A. J. M. M. Weijters, L. Maruster, Workflow Mining: Discovering Process Models from Event Logs, IEEE Transactions on Knowledge and Data Engineering 16 (9) (2004) 1128–1142.

[2] R. P. J. C. Bose, W. M. P. van der Aalst, Context Aware Trace Clustering: Towards Improving Process Mining Results, in: Proceedings of the SIAM International Conference on Data Mining, SDM, 2009, pp. 401–412.

[3] M. S. Waterman, Introduction to Computational Biology: Maps, Sequences and Genomes, Chapman & Hall/CRC, 2000.

[4] S. Chan, A. K. C. Wong, D. Chiu, A Survey of Multiple Sequence Comparison Methods, Bulletin of Mathematical Biology 54 (4) (1992) 563–598.

[5] O. Gotoh, Multiple Sequence Alignment: Algorithms and Applications, Advanced Biophysics 36 (1999) 159–206.

[6] J. D. Thompson, B. Linard, O. Lecompte, O. Poch, A Comprehensive Benchmark Study of Multiple Sequence Alignment Methods: Current Challenges and Future Perspectives, PLoS ONE 6 (3) (2011) e18093.

[7] M. R. Aniba, O. Poch, J. D. Thompson, Issues in Bioinformatics Benchmarking: The Case Study of Multiple Sequence Alignment, Nucleic Acids Research 38 (21) (2010) 7353–7363.

[8] C. Kemena, C. Notredame, Upcoming challenges for multiple sequence alignment methods in the high-throughput era, Bioinformatics 25 (19) (2009) 2455–2465.

[9] J. D. Thompson, O. Poch, New Challenges and Strategies for Multiple Sequence Alignment in the Proteomics Era, in: J. M. Walker (Ed.), The Proteomics Protocols Handbook, Springer, 2005, pp. 475–492.

[10] K. M. Kjer, U. Roshan, J. G. Gillepie, Structural and Evolutionary Considerations for Multiple Sequence Alignment of RNA, and the Challenges for Algorithms That Ignore Them, in: M. S. Rosenberg (Ed.), Sequence Alignment: Methods, Models, Concepts and Strategies, Univeristy of California Press, 2009, pp. 105–150.

[11] M. Song, W. M. P. van der Aalst, Supporting Process Mining by Showing Events at a Glance, in: Proceedings of the 17th Annual Workshop on Information Technologies and Systems (WITS), 2007, pp. 139–145.

[12] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, A. P. Barros, Workflow Patterns, Distributed and Parallel Databases 14 (1) (2003) 5–51.

[13] S. Needleman, C. Wunsch, A General Method Applicable to the Search for Similarities in the Amino Acid Sequences of Two Proteins, Journal of Molecular Biology 48 (1970) 443–453.

[14] H. Carillo, D. Lipman, The Multiple Sequence Alignment Problem in Biology, SIAM Journal of Applied Mathematics 48 (5) (1988) 1073–1082.

[15] D. Bacon, W. Anderson, Multiple Sequence Alignment, Journal of Molecular Biology 191 (1986) 153–161.

[16] R. Durbin, S. Eddy, A. Krogh, G. Mitchison, Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids, Cambridge University Press, 2002.

[17] L. Wang, T. Jiang, On the Complexity of Multiple Sequence Alignment, Journal of Computational Biology 1 (4) (1994) 337–348.

[18] C. Notredame, Recent Progress in Multiple Sequence Alignment: A Survey, Pharmacogenomics 3 (2002) 131–144.

[19] R. C. Edgar, S. Batzoglou, Multiple Sequence Alignment, Current Opinion in Structural Biology 16 (2006) 368–373.

[20] D. Feng, R. Doolittle, Progressive Sequence Alignment as a Prerequisite to Correct Phylogenetic Trees, Journal of Molecular Evolution 25 (1987) 351–360.

[21] D. Feng, R. Doolittle, Progressive Alignment of Amino Acid Sequences and Construction of Phylogenetic Trees from Them, Methods in Enzymology 266 (1996) 368–382.

[22] A. K. Jain, R. C. Dubes, Algorithms for Clustering Data, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[23] J. H. Ward, Hierarchical Grouping to Optimize an Objective Function, Journal of the American Statistical Association 58 (1963) 236–244.

[24] G. Greco, A. Guzzo, L. Pontieri, et al., Discovering expressive process models by clustering log traces, IEEE Transactions on Knowledge and Data Engineering (2006) 1010–1027.

[25] R. P. J. C. Bose, W. M. P. van der Aalst, Trace Clustering Based on Conserved Patterns: Towards Achieving Better Process Models, in: Proceedings of BPM Workshops, Vol. 43 of LNBIP, 2009, pp. 170–181.

[26] R. C. Edgar, MUSCLE: a Multiple Sequence Alignment with Reduced Time and Space Complexity, BMC Bioinformatics 5 (2004) 113. doi:10.1186/1471-2105-5-113.

[27] M. Simonsen, T. Mailund, C. N. S. Pedersen, Rapid Neighbor-Joining, in: Algorithms in Bioinformatics, 2008, pp. 113–122.

[28] L. Ghionna, G. Greco, A. Guzzo, , L. Pontieri, Outlier Detection Techniques for Process Mining Applications, in: Proceedings of the 17th International Symposium on Methodologies for Intelligent Systems (ISMIS), Springer-Verlag, 2008, pp. 150–159.

[29] Mining Usage Scenarios in Business Processes: Outlier-Aware Discovery and Run-Time Prediction, Tech. rep., Universit della Calabria.
URL `http://biblio.cs.icar.cnr.it/tr/scaricaTR.asp?FileID=84`

[30] R. P. J. C. Bose, W. M. P. van der Aalst, Abstractions in Process Mining: A Taxonomy of Patterns, in: Proceedings of the International Conference on Business Process Management (BPM), Vol. 5701 of LNCS, 2009, pp. 159–175.

[31] A. K. A. de Medeiros, W. M. P. van der Aalst, Process Mining Towards Semantics, in: Advances in Web Semantics-I, 2008, pp. 35–80.

[32] F. Murtagh, A Survey of Recent Advances in Hierarchical Clustering Algorithms, The Computer Journal 26 (4) (1983) 354–359.

[33] G. N. Lance, W. T. Williams, A General Theory of Classificatory Sorting Strategies, The Computer Journal 9 (1967) 373–380.

[34] A. K. A. de Medeiros, W. M. P. van der Aalst, C. Pedrinaci, Semantic Process Mining Tools: Core Building Blocks, in: 16th European Conference on Information Systems, 2008, pp. 1953–1964.

[35] J. Li, R. P. J. C. Bose, W. M. P. van der Aalst, Mining Context-Dependent and Interactive Business Process Maps using Execution Patterns, in: M. zur Muehlen, J. Su (Eds.), BPM 2010 Workshops, Vol. 66 of LNBIP, Springer-Verlag, 2011, pp. 109–121.

[36] B. Morgenstern, S. J. Prohaska, D. Pöhler, P. F. Stadler, Multiple Sequence Alignment with User-defined Anchor Points, Algorithms for Molecular Biology 1 (6) (2006) 1–12.

[37] C. Lee, C. Grasso, M. F. Sharlow, Multiple Sequence Alignment Using Partial Order Graphs, Bioinformatics 18 (3) (2002) 452–464.

[38] C. Grasso, C. Lee, Combining Partial Order Alignment and Progressive Multiple Sequence Alignment Increases Alignment Speed and Scalability to Very Large Alignment Problems, Bioinformatics 20 (10) (2004) 1546–1556.

[39] C. W. Günther, Process Mining in Flexible Environments, Ph.D. thesis, Eindhoven University of Technology (2009).

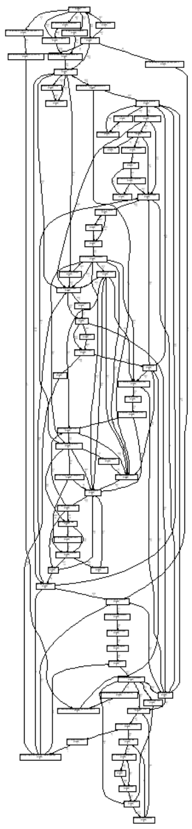
[40] A. Rozinat, W. M. P. van der Aalst, Conformance Checking of Processes Based on Monitoring Real Behavior, Information Systems 33 (1) (2008) 64–95.

[41] C. Daniel, D. Paul, M. Vidhya, O. Marco, H. Eun-Jong, W. Yaoyu, S. Shyamal, C. Brian, P. Shobha, H. Enoch, PFAAT version 2.0: A Tool for Editing, Annotating, and Analyzing Multiple Sequence Alignments, BMC Bioinformatics 8 (1) (2007) 381.

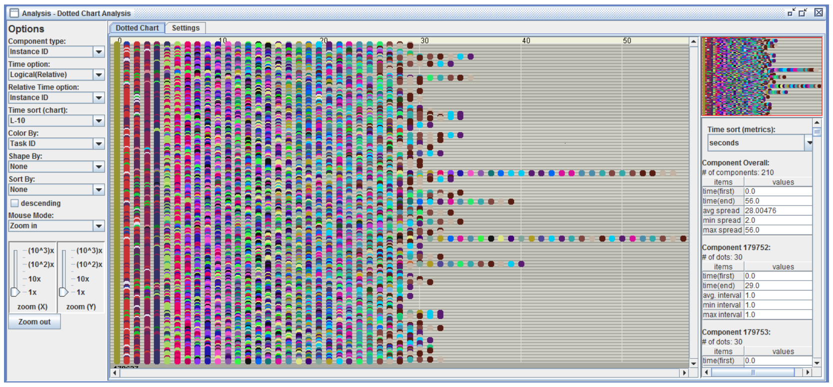
[42] P. C. Mahalanobis, On the Generalized Distance in Statistics, in: Proceedings of the National Institute of Sciences of India, Vol. 2, 1936, pp. 49–55.

[43] R. P. J. C. Bose, W. M. P. van der Aalst, Trace Alignment in Process Mining: Opportunities for Process Diagnostics, in: Proceedings of the International Conference on Business Process Management (BPM), Vol. 6336 of LNCS, 2010, pp. 227–242.

Appendix A. Appendix



(a) Heuristic net



Options

Component type: Instance ID

Time option: Logical(Relative)

Relative Time option: Instance ID

Time sort (chart): L-10

Color By: Task ID

Shape By: None

Sort By: None

☐ descending

Mouse Mode: Zoom in

zoom (X): 1x

zoom (Y): 1x

Time sort (metrics): seconds

Component Overall:
# of components: 210

items	values
time(first)	0.0
time(end)	56.0
avg spread	28.00476
min spread	2.0
max spread	56.0

Component 179752:
# of dots: 30

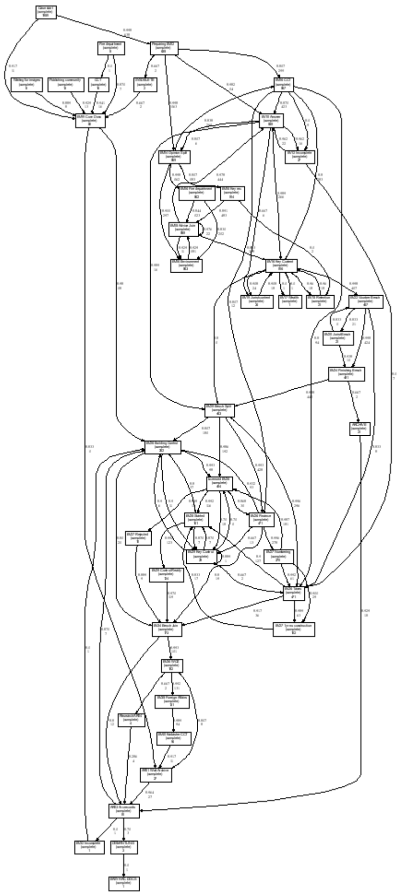
items	values
time(first)	0.0
time(end)	29.0
avg. interval	1.0
min interval	1.0
max interval	1.0

Component 179753:
# of dots: 30

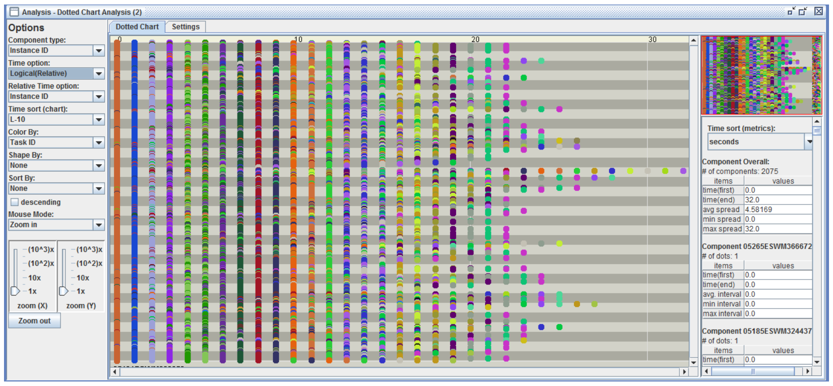
items	values
time(first)	0.0

(b) Dotted chart mined using the logical relative time perspective

Figure A.25: (a) Heuristic net and dotted chart mined using the rental agency event log



(a) Heuristic net



Options

Component type: Instance ID

Time option: Logical(Relative)

Relative Time option: Instance ID

Time sort (chart): L-10

Color By: Task ID

Shape By: None

Sort By: None

descending

Mouse Mode: Zoom in

zoom (X): 1x

zoom (Y): 1x

Time sort (metrics): seconds

Component Overall:
# of components: 2075

items	values
time(first)	0.0
time(end)	32.0
avg spread	4.58169
min spread	0.0
max spread	32.0

Component 05265ESWM366672
# of dots: 1

items	values
time(first)	0.0
time(end)	0.0
avg. interval	0.0
min interval	0.0
max interval	0.0

Component 05185ESWM324437
# of dots: 1

items	values
time(first)	0.0

(b) Dotted chart mined using the logical relative time perspective

Figure A.26: Heuristic net and dotted chart mined using the municipality event log