A Process Mining Approach Based on Trace Alignment Information for Very Large Sequential Processes with Duplicated Activities

Pamela Viale
Laboratoire des Sciences de l'Information et des Systèmes
Marseille, France
STMicroelectronics
Rousset, France
pamela.viale@lsis.org

Claudia Frydman
Laboratoire des Sciences de l'Information et des Systèmes
Marseille, France
CIFASIS, Centro Internacional Franco Argentino de Ciencias de la información y de Sistemas
Rosario, Argentina
claudia.frydman@lsis.org

Jacques Pinaton
STMicroelectronics
Rousset, France
jacques.pinaton@st.com

ABSTRACT

This paper presents an approach to construct models for very large sequential processes with duplicated activities. Very large meaning processes containing hundreds of activities. The algorithm uses traces of the process we are interested in modeling and construct an alignment between them. The alignment obtained is then used to guide the model construction. We present also the application of our approach to STMicroelectronics' manufacturing processes. The constant modifications of semiconductor specifications forces engineers to continuously update processes. For this reason, STMicroelectronics is interested in re-constructing models of its manufacturing processes to being able to control their definitions and quickly react in case a problem is detected.

Keywords

Sequential Processes, Sequence Alignment, Process Mining, Block-structured Models

1. INTRODUCTION

Mining process models from process traces is an alternative to constructing process models from scratch. It also offers a mean to analyze and optimize already existent models. This paper presents an approach for constructing (or reconstructing) process models using process trace data. The idea of this approach is to construct process models using an alignment between different traces of a same process.

The method we propose in this paper could be applied for mining different kind of sequential processes. However, we are specially interested in mining very large sequential processes (i.e. processes containing hundreds of tasks), such as STMicroelectronics' manufacturing processes. This enterprise is interested in applying process mining methods over its trace data for reconstructing its process models. Semiconductor specifications are modified very often (STMicroelectronics has an average of 50 modifications to its manufacturing process per week). Therefore, process definitions are continuously updated. Products cycle time (i.e. period of production) is mostly longer than sixty days. This means that there are objects being manufactured that can follow different process definitions for a same final product. So, reconstructing process models using process mining techniques will help STMicroelectronics' experts to easily detect added/deleted/modified options to processes and will enable to react quicker if an abnormal situation is detected. The final objective is to reduce reaction time to problems detected on process definitions.

The two main characteristics of the processes we are interested in mining are: 1) they contain hundreds of activities; and 2) duplicated activities are allowed. For this reason, we consider that is helpful to differentiate when a repetition of an activity is part of normal execution of the process and when it is due to problems encountered. In fact, a repetition of a an activity can be due to a situation that arrives frequently during the execution of a process. When executing a process some controls are done. If the desired results are not achieved some actions (special actions) has to be done to correct problems. These actions usually consist on performing special activities and/or the repeating already executed activities.

This is the reason why we propose to align the different process traces to identify equivalent activities over the different executions. It is straight forwards to think that when a subsequence of activities is present in all process traces this subsequence is part of normal execution. Subsequences of tasks executions not aligned can be consider to be special subsequences. If we consider STMicroelectronics production traces, these special subsequences could be: 1) tasks executed to correct problems; or 2) tasks representing modifications suffered by the process in the period between the dates of the process traces considered.

We have analyzed several multiple sequence alignment methods considering optimality and time and memory needs. After this analysis, we have decided to work with the T-Coffee (Tree-based Consistency Objective Function For alignmEnt Evaluation) algorithm presented in [10]. The original implementation of T-Coffee accepted only biological sequences as input. We have re-implemented this method for process traces sequences.

We present in this article all the necessary steps for creating process models from an alignment. The alignment is used to identify subsequences of tasks that are contained in all traces. As a consequence, all options of the process are also identified. We could analyze if our method is complete and minimal. Completeness describes that a model preserve all the dependencies between activities that are present in a process trace. Minimality assures that the model should neither introduce paths of execution nor spurious dependencies between activities which are not present in the traces considered. In the article we will explain why our method is complete but not minimal.

We have already tested our ideas using STMicroelectronics' manufacturing process traces obtaining encouraging results.

In the following Section (Section 2), we will present the process mining state of art and we will explain why we did not use neither of the algorithms studied. Section 3, introduces the vocabulary that will be used in the rest of the article. In Section 4 we define trace alignment and briefly introduce the alignment methods. In the same Section we present the method chosen, T-Coffee algorithm, and we expose the reasons for this choice. Afterward, in Section 5 we present our process mining method and propose a small example for its well understanding. In Section 6 we show the application of the method for mining STMicroelectronics' production processes. Finally, in Section 7 we make a conclusion and we talk about future work.

2. PROCESS MINING STATE OF ART

In this section we will present a small analysis of some process mining methods that have been studied during our research.

The first process mining technique was proposed in the context of software engineering process in [3]. In that article, three methods are described for process discovery: one using neural networks, one consisting in a purely algorithmic approach and one Markovian approach. The neural network approach is not well explained in that article. The purely algorithmic approach constructs *finite state machines* where states are fused if their futures (in terms of possible future behavior in the next k steps) are identical. The markovian approach uses a mixture of algorithmic and statistical methods and is able to deal with noise. The authors consider the last two approaches to be the most promising. Notice that all of them deal with sequential processes. This could be useful for us because our work is limited for the moment to sequential processes. The model representation used by these approaches is finite state machines. This representation accepts duplicated activities. However, we decided to avoid using this kind of representation due to the size of the models we are interesting in modeling. There is a consensus to consider that finite state machines are difficult to understand and to validate. The same authors extended their approach to deal with concurrent processes in [4] and [2]. This extension cannot deal with duplicated activities.

Another interesting process mining algorithm is presented in [1]. Authors divide process mining problem into two different sub-problems: graph mining and condition mining. The graph mining problem consist to discover the structure of the process and the condition mining problem consists on finding the conditions associated to the different execution paths. They only treated the first problem in their article, the graph mining. The model representation they used for their processes are *process graphs*. One characteristic of this kind of model is that an activity can appear only once in a model. We find here again the same problem that we found before, the method cannot be used for modeling processes containing duplicated activities.

Other approach has been published in [16]. The method presented in that article can deal with noise and concurrency. The model representation they use are a subset of Petri nets, *WF-nets*. We know that Petri-nets provide a graphical formal language designed for modeling concurrency, what it is appropriate for the kind of processes they want to discover. Their method consists on a series of steps: (i) construction of dependency/frequency table, (ii) induction of a graph from the previous table, (iii) reconstruction of a WF-net from the dependency/frequency table and the graph obtained in the previous steps. Working from the dependency/frequency table forbids the method to deal with duplicated tasks. They transform traces into a dependency/frequency table and they consider that repetitions of an activity in a trace are due to loops. Once again, the proposed approach is not useful for mining processes with duplicated tasks.

[11] introduces a different approach from the others just described. The main difference is that authors of this last article have considered activities' lifespans. All methods presented before consider that activities are executed in an atomic way. Here the approach is different, they consider that execution of task may take some time, so the use two events for identifying the starting and ending times of each activity. In this way they are able to easily discover concurrency between activities. The disadvantage of this method from our optic is the same as with the previous ones. Their model representation is based on process graphs. So, we cannot represent processes containing duplicated activities.

Two articles [13] and [14] present an algorithm for mining exact models for concurrent process. This method consider, as the last presented, that activities are non-atomic. So, two events are considered for each activity, events stating the starting and ending moments of execution of each activity. This method constructs complete and minimal models, complete in the sense that all the traces used for its construction can be obtained from the model and minimal meaning that no spurious trace can be generated from the final model. The model representation used is block-oriented. The building blocks are terms and operators. This kind of model allows to represent processes containing duplicated tasks, because basic terms are pointers to tasks or sub-processes. However, the approach proposed in both articles [13] and [14] does not consider the mining of processes containing duplicated tasks. We will go back to the block-oriented representation in Section 5.1.

We can see from this analysis, that neither of the approaches studied provided a solution to our problem of mining sequential processes with duplicated activities. However, we found an interesting work in [8]. This article proposes to use trace alignment for process diagnostics. We agree with the authors that *"trace alignment uncovers common execution paths patterns and deviations in the log* [1]*"*. Based on this, we propose to use trace alignment not just for process diagnostics but also for mining processes.

3. NOTATIONS

We will introduce here the vocabulary that will be used in subsequent sections. For the sake of simplicity, we will use notations introduced in [8]:

- Let Σ denote the set of activities. $|\Sigma|$ is the number of activities.
- Σ^+ is the set of all non-empty finite sequences of activities from Σ. $T \in \Sigma^+$ is a trace over Σ. $|T|$ denotes the length of a trace T.
- The set of all *n-length* sequences over the alphabet Σ is denoted by Σ^n. A trace of length n is denoted as T^n i.e., $T^n \in \Sigma^n$, and $|T^n| = n$.
- The ordered sequence of activities in T^n is denoted as $T(1)T(2)\ldots T(n)$ where $T(k)$ represents the k^{th} activity in the trace.
- T^{n-1} denotes the $n-1$ length prefix of T^n. In other words $T^n = T^{n-1}T(n)$.
- An event log, $\mathcal{L}$, corresponds to a multi-set (or bag) of traces from Σ^+.

4. TRACE ALIGNMENT

In this section we define formally what a trace alignment is. This definition was first introduced in [8].

Definition 1. *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*

- $|\overline{T_1}| = |\overline{T_2}| = \ldots = |\overline{T_n}| = m$,
- *$\overline{T_i}$ by removing all "$-$" gap symbols is equal to T_i,*
- $\nexists k, 1 \leq k \leq m$ *such that* $\forall_{1\leq i\leq n}, \overline{T_i}(k) = -$

[1]A log is the set of traces considered

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 traces in $\mathbb{T}$.

4.1 Trace Alignment Algorithms

It is important to realize that there are many different possible alignments between a set of traces. The choice that make us consider that one is better that another one is based on a function *Score* that assigns a numerical value to an alignment. There are many propositions in the literature for this function Score, but we will not speak about this here. Refer to [9], [10] or [15] if interested. Function *Score* is calculated based on a matrix that we will present here. The scores for aligning the different activities in Σ are obtained from a matrix $\mathcal{S} = \{s_{x,y}\}(x \in \Sigma, y \in \Sigma)$, called similarity matrix. The value $s_{x,y}$ represents the similarity between activity $x \in \Sigma$ and activity $y \in \Sigma$. There is also a particular value that is considered, the gap penalization value, that is a value associated to the alignment of an activity from Σ with a gap $(-)$. So, the objective of alignment algorithms is to find alignments that maximize the *Score* function. However, finding a similarity matrix is not an easy problem.

In biology, there exists some well-known similarity matrices that are used for aligning, for example, DNA sequences. However, this problem has also been encountered in many other areas, i.e. social science field, where there are no well known similarity matrices pre-calculated. The problem in social science is that there is no equivalent to such a strong theoretical and empirical framework like evolutionary theory. Thus the relationship between the symbols constituting sequences of social events remain at least partly uncertain. Some ideas have been proposed to find a way to construct the similarity matrix $\mathcal{S}$ in the social science context. We consider that it could be interesting in the future to investigate this for constructing special similarity matrices for any kind of sequences, in particular, for process traces sequences, following ideas published in [5].

Pairwise trace alignment methods can be used to find the best alignment between two traces. Alignment methods can be classified into global and local alignment algorithms. The difference between global and local alignments is that global alignments forces the alignment to span the entire length of all query traces whereas local alignment algorithms only identify regions of similarity within all query traces. There is a dynamic programming algorithm proposed in [9] that computes the optimal global alignment between two traces. [15], by contrast, proposes a local alignment method. Sometimes, there are more than one alignment associated to the optimal score. In these situations, an arbitrary choice is made.

Multiple trace alignment is an extension of pairwise trace alignment to align more than two traces at a time. Multiple alignment methods try to align all of the traces in a given query set. Multiple traces alignments are computationally difficult to produce and most formulations of the problem lead to NP-complete combinatorial optimization problems.

The technique of dynamic programming is theoretically applicable to any number of sequences; however, because it is computationally expensive in both time and memory, it is rarely used for more than three or four sequences in its most basic form. Although this technique is computationally expensive, its guarantee of a global optimum solution is useful in cases where only a few sequences need to be aligned accurately.

Progressive, hierarchical, or tree methods generate a multiple alignments by first aligning the most similar traces and then adding successively less related traces or groups to the alignment until the entire query set has been incorporated into the solution. Progressive alignment results are dependent on the choice of most related sequences and thus can be sensitive to inaccuracies in the initial pairwise alignments.

4.2 T-Coffee alignment algorithm

T-Coffee is an algorithm that can be used for multiple trace alignment. This method is broadly based on the popular progressive approach to multiple alignment but avoids the most serious pitfalls caused by the greedy nature of this algorithm. T-Coffee pre-process a data set of all pairwise alignments between the traces. This provides a library of alignment information that can be used to guide the progressive alignment. Intermediate alignments are then based not only on the sequences to be aligned next but also on how all the sequences align with each other.

The T-Coffee algorithm is composed of several steps that we enumerate below:

1. Generating a primary library of alignments;
2. Derivation of the primary library weights;
3. Combination of the libraries;
4. Extending the library;
5. Progressive alignment strategy.

We are not going to describe in detail each step of the approach. Refer to [10] if interested. The first four steps of the approach are concerned in the construction of libraries. T-Coffee makes use of the information in the library to carry out the last step, the progressive alignment, in a manner that allows to consider the alignments between all the pairs while it carry out each step of the progressive multiple alignment. This provides a progressive alignment, with all its advantages of speed and simplicity, but with a far lesser tendency to make errors. We decided to use this algorithm for aligning traces of very large sequential processes because of its advantages over other progressive alignment methods and its complexity.

4.2.1 Complexity Analysis

Original Implementation

Considering N is the number of sequences to align and L is the average length of the considered sequences, the complexity of the whole procedure in its original implementation is given by:

$$O(N^2L^2) + O(N^3L) + O(N^3) + O(NL^2) \qquad (1)$$

where $O(N^2L^2)$ is associated with the computation of the pairwise library, $O(N^3L)$ the extension, $O(N^3) + O(NL2)$ the computation of the progressive alignment.

The CPU time consumption of T-Coffee original implementation was analyzed empirically by the authors. Measurements indicate the apparent complexity of the program is quadratic, both relative to the average sequence lengths and to the number of sequences. This result can be explained by the fact that in the cases analyzed by the authors, $L >> N$. Therefore, the time required for the library and the alignment computation is much larger than the time required for the library extension: $O(N^2L^2) + O(NL^2) >> O(N^3L)$.

The efficiency of this algorithm was the reason why we decided to use it. The original implementation of this algorithm was oriented toward biological sequences and was capable of working with $max|\Sigma| = 23$. In our process traces we sometimes find that $|\Sigma| = 1000$ so a re-implementation has been necessary. We have analyzed the source code of this original implementation and we re-implement it for our traces.

Our Implementation

Now we will consider N as the number of traces to align, L as the average length of the considered traces and $|\Sigma|$ as the number of different activities. We can analize the different steps that compose the T-Coffee method (Steps 1, 2, 3, 4, 5) and calculate the complexity of our implementation. Looking into the implementation done, the complexity of the first three steps (1, 2, 3) is the following:

$$O(N) + O(N^2L) + O(N^2|\Sigma|^2) + O(N^2L^2) + O(L) \qquad (2)$$

Then, the complexity of the library extension (step 4) is the following:

$$O(N^3L^2) \qquad (3)$$

in the worst case. However, this will only occur when all the included pairwise alignments are totally inconsistent. In practice, for the data sets used in our examples the complexity is closer to $O(N^3L)$.

Considering the implementation of the progressive alignment step (step 5), we obtain the following results:

$$O(NL) + O(LN^2) + O(L^2N^3) + O(N^3) + O(N) \quad (4)$$

As for the previous step, the term $O(L^2N^3)$ is in the worst case. For our traces, this term is closer to $O(LN^3)$ so the complexity of this step results :

$$O(NL) + O(LN^2) + O(LN^3) + O(N^3) + O(N) \quad (5)$$

instead.

The CPU time consumption of our T-Coffee implementation was analyzed empirically (following the same idea of the original paper [10]). Our empirical results indicate similar behavior as the one observed by the authors of the original paper [10]. The apparent complexity of the algorithm is quadratic. This result can be explained by the fact that in the cases considered, using STMicroelectronics' traces (see Section 6 for an explanation of these traces), $L >> N$ ($L = 550$, $N \in \{1, \dots, 50\}$). This means that $O(N^2L^2) + O(NL^2) >> O(N^3L) + O(N^3)$.

The complexity of our implementation is not as efficient as the original one, concerning the number of comparisons required. However, the apparent quadratic time complexity is also observed in our version of the algorithm.

5. OUR PROCESS MINING METHOD

5.1 Model Representation

The model representation chosen, i.e., the meta model of the extracted models are based on, is a block-oriented model originally proposed in [13] and [14].

This meta model defines that each model consists of an arbitrary number of nested building blocks. These building blocks are differentiated into operators and terms. Operators combine building blocks and define the control-flow of a workflow. We use the following operators: *Sequence*, *Parallel*, *Alternative*, and *Loop*.

The *Sequence* operator defines that all embedded blocks are performed in a particular sequential order. Let $\mathscr{S}$ denote the *Sequence* operator and let a and b denote two blocks. $\mathscr{S}(a, b)$ is then a model defining that block a is always performed before block b.

The *Parallel* operator defines that all embedded blocks can be performed concurrently, i.e. in parallel or in any order. Let $\mathscr{P}$ denote the *Parallel* operator and let a and b denote two blocks. $\mathscr{P}(a, b)$ is then a model defining that block a and b can be performed concurrently.

The *Alternative* operator defines a choice of exactly one block out of all its embedded blocks. This operator is supplemented by a set of rules determining the choice. Let $\mathscr{A}$ denote the *Alternative* operator and let a and b denote two blocks. $\mathscr{A}(a, b)$ is then a model defining that either block a or block b is performed. Additionally, we define a *Loop* operator $\mathscr{L}$. A *Loop* operator contains only one block that is executed repeatedly while its loop condition holds.

Basic terms are pointers to tasks and sub-workflows, which are embedded in a control-flow. ϵ represents an 'empty process', used for describing empty paths in alternatives.

We call the model that conforms to this meta-model a block-structured model. We can construct a model top-down by setting one operator as the starting point of the model and nesting other operators until we get the desired control-flow structure. At the bottom of this structure, we embed basic terms into operators which terminates the nesting process. Beside the representation of block-structured models as terms, we represent them in the form of diagrams. Such a diagram depicts the model's blocks in the form of nested rectangles. The control-flow is explicated by arrows and operator symbols.

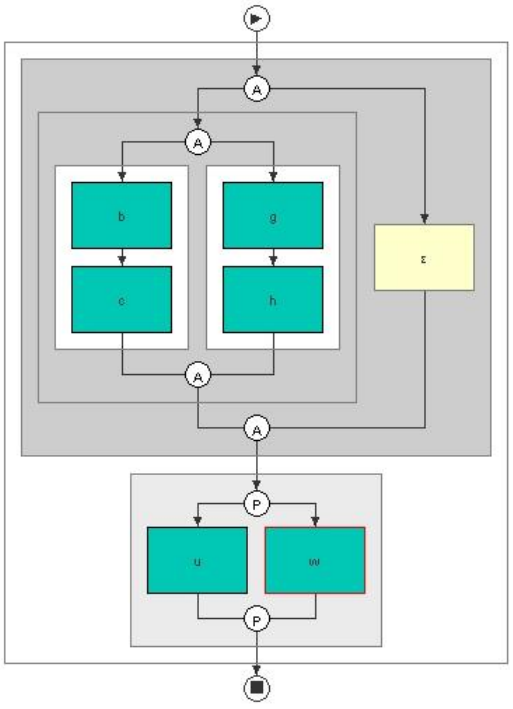

Figure 1: A simple process model

For example, Figure 1[2] [12] shows the process model $\mathscr{S}(\mathscr{A}(\mathscr{A}(\mathscr{S}(b, c), \mathscr{S}(g, h)), \epsilon), \mathscr{P}(u, w))$ in the form of a diagram. The root of this model is a *Sequence* operator block that contains all other blocks in a hierarchical manner, so that the task pointers b, c, g, h, ϵ (empty process), u and w are embedded in a nested control-flow.

For this meta-model there is an algebra that consists of a set of axioms covering distributivity, associativity, commutativity and idempotency. These axioms are the basis of term rewriting systems that can be used in order to transforms models. A set of basic axioms is shown in Table 1. More details about this meta-model can be found in [13] and [14].

Block-structured models have some advantages: they are well formed and always sound. We have decided to construct

[2]Figure created using ProcessMiner tool, downloaded from http://www.processmining.de/

Table 1: Basic Algebra

A$_1$:	$\mathscr{A}_1(x_1,\ldots,x_n) = \ldots = \mathscr{A}_{n!}(x_n,\ldots,x_1)$
A$_2$:	$\mathscr{A}(\ldots,x_1,\ldots,x_n,\ldots) = \mathscr{A}(\ldots,\mathscr{A}(x_1,\ldots,x_n),\ldots)$
A$_3$:	$\mathscr{A}(x,x,\ldots) = x$
A$_4$:	$\mathscr{S}(\mathscr{A}(x_1,\ldots,x_n),y_1,\ldots,y_m) =$
	$\mathscr{A}(\mathscr{S}_1(x_1,y_1,\ldots,y_m),\ldots,\mathscr{S}_n(x_n,y_1,\ldots,y_m))$
A$_5$:	$\mathscr{S}(\ldots,x_1,\ldots,x_n,\ldots) = \mathscr{S}(\ldots,S(x_1,\ldots,x_n),\ldots)$
A$_6$:	$\mathscr{P}_1(x_1,\ldots,x_n) = \mathscr{P}_{n!}(x_n,\ldots,x_1)$
A$_7$:	$\mathscr{P}(\mathscr{A}(x_1,\ldots,x_n),y_1,\ldots,y_m) =$
	$\mathscr{A}(\mathscr{P}_1(x_1,y_1,\ldots,y_m),\ldots,\mathscr{P}_n(x_n,y_1,\ldots,y_m))$
A$_7$:	$\mathscr{P}(\ldots,x_1,\ldots,x_n,\ldots) = \mathscr{P}(\ldots,P(x_1,\ldots,x_n),\ldots)$
A$_8$:	$x = \mathscr{S}(x,\epsilon)$
A$_9$:	$x = \mathscr{S}(\epsilon,x)$

models based on this meta-model because they can be easily converted to many models applicable in different workflow systems. We plan to automate, afterward, the comparison between the workflows constructed from this process mining method to workflow models defined by experts.

5.2 Mining Process Models

In this section we will introduce the process mining algorithm we propose to construct models from process traces.

ALGORITHM 1. *Given a set of traces $\mathbb{T} = \{T_1, T_2, \ldots, T_n\}$ of process P where $\forall 1 \le i \le n$, $T_i \in \Sigma^+$, a provided similarity matrix $\mathcal{S} = \{s_{x,y}\}(x \in \Sigma, y \in \Sigma)$ and a gap penalty d. We propose to construct a process model from this set of traces $\mathbb{T}$ in 3 steps:*

- *Step 1: Construct a trace alignment $\mathcal{A} = \{a_{ij}\}(1 \le i \le n, 1 \le j \le m)$ for $\mathbb{T}$ using T-Coffee alignmnet algorithm considering $\mathcal{S}$ and d;*
- *Step 2: Considering the alignment $\mathcal{A} = \{a_{ij}\}(1 \le i \le n, 1 \le j \le m)$ constructed in the previous step, identify all $j \in \{1,\ldots,m\}$ where $a_{ij} = a_{kj} \in \Sigma, \forall i, \forall k \in \{1,\ldots,n\}$. We call this set SAA (Same Activity Aligned).*
  Set SAA will contain the indexes of columns where only identical activities are aligned (− gap symbol is not permitted).
- *Step 3: Considering the set SAA obtained in the previous step we can construct the process model based on the meta model introduced in subsection 5.1 using the following instructions:*
  - *Starting from index j equal to 1 up to m (this means we have to consider all columns in the alignment $\mathcal{A}$), identity all pairs of indexes $(start, end)$ marking the beginning and the ending of indexes belonging to SAA and to $\overline{SAA}$[3], respectively.*
  - *For all pairs $(start, end)$ where $start, end \in SAA$ use function Construct_Sequence to construct the block model representing the only sequence admitted by the traces.*
    Function Construct_Sequence$(\mathcal{A}, start, end)$ construct a Sequence block model containing the elements of Σ found in $\mathcal{A}$ between indexes start and end.
  - *For all pairs $(start, end)$ where $start, end \in \overline{SAA}$ use function Construct_Alternative_Sequences to construct the block models representing each of the different paths introduced by the traces.*
    Function Construct_Alternative_Sequences$(\mathcal{A}, start, end)$ construct an Alternative block model, containing a block for each alternative path proposed in alignment $\mathcal{A}$ between indexes start and end. If a path contain more than one activity, the corresponding Sequence block model is constructed.
  - *Finally, respecting the order of all the pairs (start, end) found, the final model is a Sequence block model containing all the block models constructed in the previous steps, respecting the order of indexes.*

[3]$\overline{SAA}$ means the complement of SAA

Let us show how it is working by an example.

5.2.1 Example

Consider the set of traces $\mathbb{T} = \{T_1, T_2, T_3, T_4\}$ of process P. Imagine that:

$T_1 = bcdffghijk$

$T_2 = bcdghijk$

$T_3 = bcdffghi$

$T_4 = bcghik$

$\Sigma = \{b, c, d, f, g, h, i\}$ in this example.

We apply our process mining method over this log.

Step 1: We apply T-Coffee alignment algorithm over $\mathbb{T}$. Imagine we obtain the following alignment matrix:

$$A = \begin{pmatrix} b & c & d & f & f & g & h & i & j & k \\ b & c & d & - & - & g & h & i & j & k \\ b & c & d & f & f & g & h & i & - & - \\ b & c & - & - & - & g & h & i & - & k \end{pmatrix}$$

Step 2: Once we have constructed alignment $\mathcal{A}$ we proceed to find the indexes of the columns where in each row appears the same identifier of Σ (and no gaps). We construct set SAA. Looking at $\mathcal{A}$ it is easy to see it is the case of the first two columns and also columns 6, 7 and 8. The resulting set is $SAA = \{1, 2, 6, 7, 8\}$.

Step 3: We must identity all pairs of indexes $(start, end)$ marking the beginning and the ending of indexes belonging to SAA and to $\overline{SAA}$. We obtain: $(1, 2)$ where $1, 2 \in SAA$, $(3, 5)$ where $3, 5 \in \overline{SAA}$, $(6, 8)$ where $6, 8 \in SAA$, $(9, 10)$ where $9, 10 \in \overline{SAA}$.

We use the appropriate function in each case:

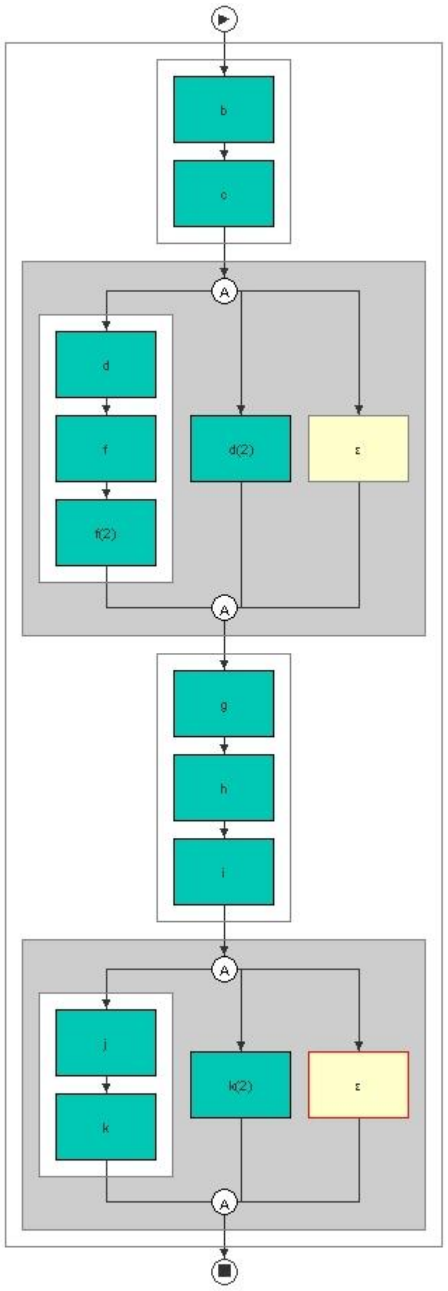

Figure 2: Resulting process model of our example

- *Construct_Sequence*$(\mathcal{A}, 1, 2)$. This function will construct the following model: $\mathscr{S}(b, c)$.
- *Construct_Alternative_Sequences*$(\mathcal{A}, 3, 5)$. This function construct a model that its root is an Alternative operator and each of the alternative blocks are the different paths proposed in the alignment between indexes 3 and 5. There are 3 paths proposed: $\mathscr{S}(d, f, f)$, d and ϵ. So, the result proposed by function *Construct_Alternative_Sequences*$(\mathcal{A}, 3, 5)$ would be: $\mathscr{A}(\mathscr{S}(d, f, f), d, \epsilon)$.
- *Construct_Sequence*$(\mathcal{A}, 6, 8)$. The resulting model is $\mathscr{S}(g, h, i)$.
- *Construct_Alternative_Sequences*$(\mathcal{A}, 9, 10)$ constructs the model: $\mathscr{A}(\mathscr{S}(j, k), \epsilon, k)$.

The final model is a Sequence block model containing the concatenation of the previous blocks just obtained:

$$\mathscr{S}(\mathscr{S}(b, c), \mathscr{A}(\mathscr{S}(d, f, f), d, \epsilon), \mathscr{S}(g, h, i), \mathscr{A}(\mathscr{S}(j, k), \epsilon, k)).$$

A graphical representation of this model is shown in Figure 2.

5.2.2 *Results*

Analyzing the procedure for constructing models proposed we can state that the models we obtain are always complete. The reason is that there is no step that omits a task. All tasks are represented in the resulting model and order between different tasks is also respected. However, the models we obtain are not minimal. We can prove this just looking at the previous example. The resulting model accepts the following sequence of activities: $b \rightarrow c \rightarrow d \rightarrow f \rightarrow f \rightarrow g \rightarrow h \rightarrow i \rightarrow k$ that is not present in the log used for the model construction.

We must also remark that models we construct could sometimes be transformed into simpler ones. Look again at the result obtained in the example. Consider the constructed block:

$$\mathscr{A}(\mathscr{S}(j, k), \epsilon, k).$$

There we can see that in two paths, activity k is referenced. We could simplify this model using the basic algebra proposed in Table 1. We could transform that block in the following way:

$$\begin{gathered}\mathscr{A}(\mathscr{S}(j, k), \epsilon, k) \equiv_{(A9)} \\ \mathscr{A}(\mathscr{S}(j, k), \epsilon, \mathscr{S}(\epsilon, k)) \equiv_{(A1)} \\ \mathscr{A}(\mathscr{S}(j, k), \mathscr{S}(\epsilon, k), \epsilon) \equiv_{(A2)} \\ \mathscr{A}(\mathscr{A}(\mathscr{S}(j, k), \mathscr{S}(\epsilon, k)), \epsilon) \equiv_{(A4)} \\ \mathscr{A}(\mathscr{S}(\mathscr{A}(j, \epsilon), k), \epsilon)\end{gathered}$$

We are thinking already a way to automatize the construction of simpler models. However, the first interest that was finding a model for easily differentiate normal executions paths from optional executions paths for very large sequential processes has been achieved using our process mining method.

6. APPLICATION

We are going to show now the results obtained using our process mining method in a particular case: STMicroelectronics' sequential manufacturing processes. But, before that, we are going to introduce the vocabulary associated to STMicroelectronics' production processes (vocabulary used by its experts).

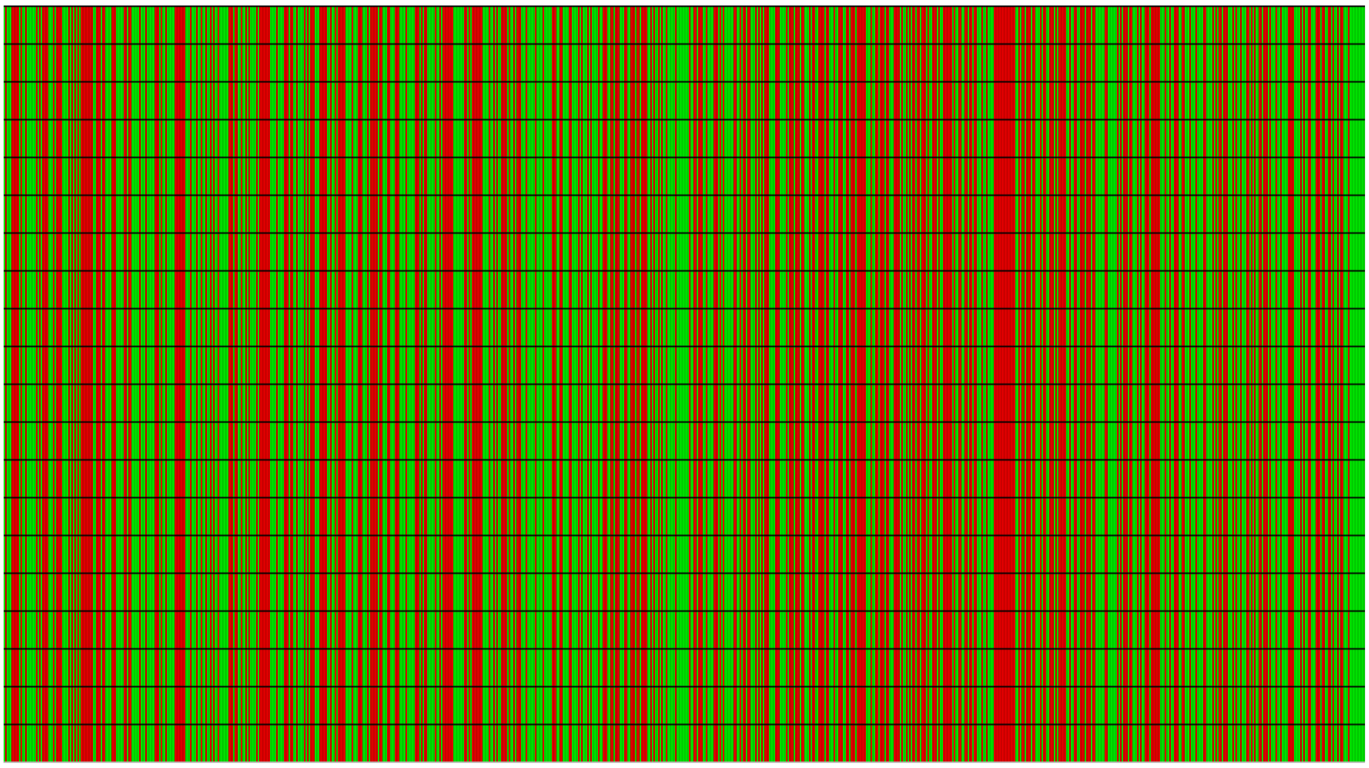

Figure 3: Colored representation of alignment for 20 process traces from STMicroelectronics

6.1 STMicroelectronics Experts' Vocabulary

A manufacturing process is called *'Production Route'* in STMicroelectronics' vocabulary. Each of these production routes is composed of a sequence of *'Operations'*. Each operation has a numerical identifier and inside a route, operations' identifiers are ordered respecting an increasing order. Each operation is, as well, a sequence of *'Steps'* and *'Events'*. In fact, a step is always associated to an event. A step is just a numerical identifier used to order the events in an operation; increasing order is also respected here. An event is a desired treatment over the manufactured object. Associated to each event there are *'Entities'*. An entity is a machine that is qualified, i.e. that has been tested and validated for the desired treatment. Depending on the event to perform and the entity chosen, there is a *'Recipe'* associated. A recipe is a list of machine instructions that are given to the entity chosen to perform the desired treatment. There may be different recipes associated to a same event. The reason is that there exists different machines in STMicroelectronics that are used for performing same treatments. So, the list of machine instructions are not the same if the machines are not equal.

A manufactured object is called a *lot* in STMicroelectronics' vocabulary. In fact, a lot is a box containing, at most, 25 silicon plates (*wafers*). During production, the lot goes from one machine to another to be transformed according to the process definition.

6.2 STMicroelectronics Data Volume

STMicroelectronics plant situated in the south of France, in Rousset, counts actually with more than 1000 active production routes. Each of these production processes is composed of, in average, more than 500 different operations. Each operation is composed of steps, steps number vary from 1 to 4, 5 steps in the longest operations. This means that a lot must go through (in average) 600 different transformations steps before obtaining the final product. Some other interesting numbers can be cited to show the volume of data related to this enterprise processes definitions. There exist approximately 10000 different recipes (list of machines instructions) and there are more than 300 equipments in this plant.

These are the reasons why the alignment method chosen for performing the alignment has been studied in detail, there is a huge data volume to treat. Optimality has been considered. However, time and memory needs were the priority.

6.3 STMicroelectronics Traces

STMicroelectronics counts with a Supervision System of Production that is called *'WorkStream'*. Each time a lot is treated, the corresponding information is saved in WorkStream's databases. Information saved in these databases are lot identifier, production route identifier to which the lot is associated, operation, step, event, recipe being executed, starting and ending times of the treatments, machine used, etc. All this information is saved for all lots in production. We use a subset of all this information for testing our approach.

Each trace in this example will be associated to a lot. For each lot, the trace contain an ordered sequence of the activities that the lot went through from the beginning to the end of its production. Considering the description of STMicroelectronics' processes given in Subsection 6.1, the names of the activities will be consider to be the concatenation between the name of the event and the name of the recipe chosen for execution. We do not consider information such as operations and steps numbers in our traces because they are used by STMicroelectronics' engineers to order the activities. Introducing this information will implicitly introduce information about the process model and we do specially want to avoid that.

6.4 Results

We have tested our approach over STMicroelectronics' real production traces. We extracted a set of 20 traces $\mathbb{T} = \{T_1, \ldots, T_{20}\}$ of a same process. Traces length varied from 520 to 620 recorded activities. A total of 551 different activities have been found in $\mathbb{T}$, this means $|\Sigma| = 551$. We performed an alignment between these traces using our implementation of T-Coffee algorithm, obtaining the alignment matrix $\mathcal{A} = (\overline{T_1}, \ldots, \overline{T_{20}})^T$ as result. We have used a similarity matrix $\mathcal{S}$ such that $s_{x,y} = 1$ when $x = y$ and $s_{x,y} = -1$ when $x \neq y$, where x and $y \in \Sigma$. Aligning any activity identifier with a gap symbol $(-)$ has been penalized using a d value set to -1.

In Figure 3 we can see a colored representation of the alignment matrix $\mathcal{A}$ obtained as result. The dimensions of this matrix are 20 *rows* $\times$ 884 *columns*. The construction of $\mathcal{A}$ corresponds to the *Step 1* of the algorithm proposed in Subsection 5.2. Each trace is represented in one row in the matrix. Due to the huge dimensions of this alignment, we have colored columns for a better understanding of the results. In fact, we do not show exactly the content of this matrix in Figure 3, we just show a colored version of it for the readers' better understanding. Green columns show columns where all tasks identifiers (for all 20 lots) match (set SAA). Identifying these green columns corresponds to the *Step 2* of the same algorithm. We colored in red the rest. Analyzing the alignment contents we can appreciate that even though differences have been found between traces, a common process is identified from the 462 columns colored in green (for the sake of brevity this content is not shown in the article). This means that 462 similar tasks have been identified between the $520 - 620$ that compose the traces.

Doing an analysis of alignment $\mathcal{A}$, we can confirm that differences are due to (in most of the cases) to repetitions of tasks for correcting errors in production, repetition of measures and modifications done to the process definitions between the different dates of the traces that compose the log.

The construction of the model from this colored matrix is straight forwards, *Step 3* in our mining algorithm. However, we are not going to show the model obtained here. In fact, the construction procedure has been already explained for the example of Subsection 5.2.1.

7. CONCLUSION AND FUTURE WORK

We are satisfied by the results obtained in the application of our approach over STMicroelectronics traces. The results obtained encourage us to continue to investigate the use of alignments to differentiate normal process executions and special activities executions inside process traces. Our future work will surely consist in the transformation of the models obtained by our method into simpler models.

This work is part a project that searches to construct generic process models. These generic process models will be obtained merging models obtained from two different sources: activity (process traces) and experts process specification. For being able to merge models from these two different sources, all models will need to be translated to a common formalism. The formalism that we will use are Timed Sequential Machines (TSM) [7] and [6]. This will enable us to simulate if required in the future.

8. REFERENCES

[1] R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. *In Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.

[2] J. E. Cook, Z. Du, C. Liu, and A. Wolf. Discovering models of behavior for concurrent workflows. *Computers in Industry*, 53:297–319, 2004.

[3] J. E. Cook and A. L. Wolf. Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology*, 7:215–249, 1998.

[4] J. E. Cook and A. L. Wolf. Event-based detection of concurrency. In *in: Proceeding of the Sixth ACM SIGSOFT International Symposium on the Fondation of Software Engineering (FSE-6)*, volume 53, pages 35–45, 1998.

[5] J. A. Gauthier, E. D. Widmer, P. Bucher, and C. Notredame. How much does it cost? optimization of costs in sequence analysis of social science data. *Sociological Methods and Research*, 2008.

[6] N. Giambiasi. An introduction to timed sequential machines.

[7] N. Giambiasi. From sequential machines to devs formalism. *International Simulation Multiconference(ISMc'09)*, 2009.

[8] R. Jagadeesh Chandra Bose and W. van der Aalst. Trace alignment in process mining: Opportunities for process diagnostics. 6336:227–242, 2010.

[9] S. B. Needleman and C. D. Wunsch. A general method applicable to the search of similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, 48:443–453, 1970.

[10] C. Notredame, D. G. Higgins, and J. Heringa. T-coffee: A novel method for fast and accurate multiple sequence alignment. *J. Mol. Biol.*, 302:205–217, 2000.

[11] S. Pinter and M. Golani. Discovering workflow models from activities' lifespans. In *Special issue: Process/workflow mining*, volume 53, pages 283–296, 2004.

[12] G. Schimm. Process miner – a tool for mining process schemes from event-based data. In *Proceedings of the European Conference on Logics in Artificial Intelligence*, volume 2424, pages 525–528, 2002.

[13] G. Schimm. Mining most specific workflow models from event-based data. In *Business Process Management'03*, pages 25–40, 2003.

[14] G. Schimm. Mining exact models of concurrent

workflows. *Comput. Ind.*, 53:265–281, April 2004.

[15] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

[16] A. Weijters and W. van der Aalst. Process mining discovering workflow models from event-based data. In *Proceedings of the ECAI Workshop on Knowledge Discovery and Spatial Data*, pages 283–290, 2001.