Empowering Conformance Checking using Big Data through Horizontal Decomposition

Álvaro Valencia-Parra[a], Ángel Jesús Varela-Vaca[a,*], María Teresa Gómez-López[a], Josep Carmona[b], Robin Bergenthum[c]

[a] *Departamento de Lenguajes y Sistemas Informáticos,*
Universidad de Sevilla, Spain
http://www.idea.us.es
[b] *Department of Computer Science,*
Universitat Politècnica de Catalunya, Spain
https://www.cs.upc.edu
[c] *Fakultät für Mathematik und Informatik,*
FernUniversität in Hagen, Germany
https://www.fernuni-hagen.de

Abstract

Conformance checking unleashes the full power of process mining: techniques from this discipline enable the analysis of the quality of a process model discovered from event data, the identification of potential deviations, and the projection of real traces onto process models. This way, the insights gained from the available event data can be transferred to a richer conceptual level, amenable for a human interpretation. Unfortunately, most of the aforementioned functionalities are grounded in a very hard fundamental problem: given an observed trace and a process model, to find the model trace that is most similar to the trace observed. This paper presents an architecture that supports the creation and distribution of alignment subproblems based on a novel horizontal acyclic model decomposition, disengaged of the conformance checking algorithm applied to solve them. This is supported by a Big Data infrastructure that facilitates the customised distribution of a

[*] Corresponding author: Ángel J. Varela-Vaca

Email addresses: avalencia@us.es (Álvaro Valencia-Parra), ajvarela@us.es (Ángel Jesús Varela-Vaca), maytegomez@us.es (María Teresa Gómez-López), jcarmona@cs.upc.edu (Josep Carmona), robin.bergenthum@fernuni-hagen.de (Robin Bergenthum)

great amount of data. Experiments are provided witnessing the enormous potential of the architecture proposed, opening the door to further research in several directions.

Keywords: Conformance Checking, Decompositional Techniques, Big Data, MapReduce

1. Introduction

Organisations tend to define, by means of conceptual models, complex business processes that must be followed to achieve their objectives [1]. Sometimes the corresponding processes are distributed in different systems, and most of the cases include human tasks, enabling the occurrence of unexpected deviations with respect to the (normative) process model. This is aggravated by the appearance of more and more complex processes, where the observations are provided by heterogeneous sources, such as Internet-of-Things (IoT) devices involved in Cyber-physical Systems [2].

Conformance checking [3] techniques provide mechanisms to relate modelled and observed behaviour, so the frictions between the footprints left by process executions, and the process models that formalise the expected behaviour, can be revealed.

One of the major challenges in conformance checking is the *alignment problem*: given an observed trace σ, compute an end-to-end model run that is more similar to σ. Computing alignments is a very hard problem, with a complexity exponential in the size of the model or the trace [4]. Intuitively, computing an alignment requires a search through the model' state space, which implies in some cases a large exploration when the process model is large and/or highly concurrent.

In order to face the challenge of computing alignments, the conformance checking community has proposed very different alternatives. Among these, we highlight decompositional techniques, that break the alignment problem into pieces, whose solutions can be composed to reconstruct the final alignment [5, 6, 7, 8]. All these decompositional approaches have in common their strategy to decompose the problem by means of *vertical cuts* of the process model, and then projecting the traces in the log accordingly in order to derive subtraces that only contain events of the alphabet corresponding to each model fragment. Although in very particular cases (e.g., well-structured process models) the aforementioned decompositional approaches represent a significant alleviation of the alignment problem, they rely on very stringent conditions (e.g., model fragments should agree on the alphabet at the frontiers), and provide weak guarantees (e.g., necessary conditions for deriving an alignment), which hamper them from being applied in general.

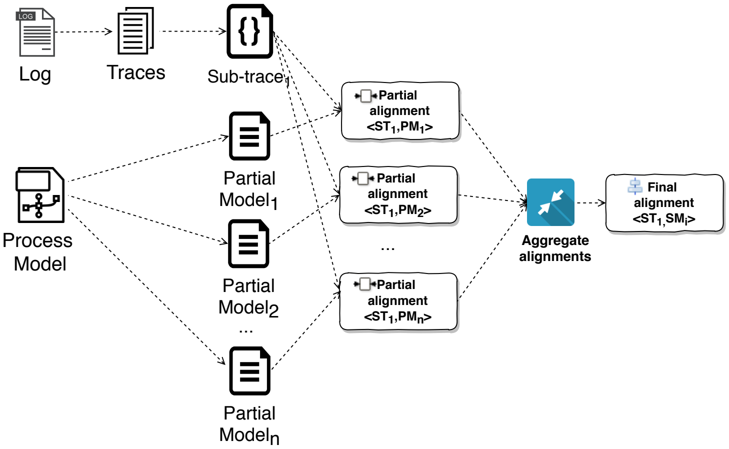

LOG
Log
Traces
{}
Sub-trace1
Process Model
Partial Model1
Partial Model2
...
Partial Modeln
Partial alignment <ST1,PM1>
Partial alignment <ST1,PM2>
...
Partial alignment <ST1,PMn>
Aggregate alignments
Final alignment <ST1,SMi>

Figure 1: Functional description of the Big Data architecture to compute alignments.

In this paper, we step back from the decompositional approach, and focus on working at a more abstract, architectural level. We propose a Big Data infrastructure that can be instantiated to any of the existing decompositional approaches, although our approach is focused on an specific horizontal decomposition, employing the MapReduce paradigm [9] for the decomposition and aggregation. A first sight to our functional strategy (depicted in Figure 1) will not bring any new idea to the landscape of decompositional techniques: the process model is decomposed into a set of partial models, and traces in the log are projected into subtraces. These two types of elements are then distributed and their partial solutions composed to aggregate a final alignment. This distribution of the problem may facilitate the simplification of the problem, splitting the conformance analysis into partial models and subtraces across different nodes (Map), and combining the partial alignments obtained from different algorithms in the nodes (Reduce). The general idea of applying MapReduce for conformance checking is not new, as is analysed in the related work section. However, the Big Data [10] framework proposed in this paper is novel for the following reasons:

- A novel decomposition is proposed, which differs from the aforementioned approaches in one important feature: instead of a vertical cut, it is based on horizontal, end-to-end cuts that can be obtained by what we call *acyclic cover*, which is originated from a partial order representation of the initial process model.

- Tuning the construction, distribution and parallelisation of the computing alignment between different nodes in a Big Data environment, according to the features of the problem and the available requirements. Moreover, the application of heuristics is proposed for optimising the resolution of the subproblems.
- It enables us to choose and customise the conformance checking algorithm, being possible to compute the alignment with different techniques. In this case, we have used the A* algorithm, as a classical solution, and *Constraint Programming Paradigm* [11], as novel one, in order to show how different types of alignment algorithms can be applied in the distributed paradigm.
- The development of a practicable infrastructure based on Big Data, that represents a leap forward in the resolution of more complex conformance checking problems reducing the resource limitations of the current solutions evaluated locally.

The paper is organised as follows: Section 2 analyses the related work. Section 3 includes the necessary foundations to understand the state of the art and the proposal. Section 4 determines how the use of Big Data techniques provides mechanisms for the partitioning and distribution of the computation of the conformance checking analysis. Section 5 describes how the A* algorithm and Constraint Programming can be applied to traces that represent the acyclic horizontal partial models. Section 6 depicts the experiments used to evaluate our proposal, and finally, the paper is concluded.

2. Related Work

The seminal work in [4] proposed the notion of alignment and developed a technique based on A* to compute optimal alignments for a particular class of process models. Improvements of this approach have been presented recently in different papers [12, 13]. The approach represents the state-of-the-art technique for computing alignments, and can be adapted (at the expense of increasing significantly the memory footprint) to provide all optimal alignments. Alternatives to A* have appeared in the last years: in the approach presented in [14], the alignment problem is mapped as an *automated planning* instance. Automata-based techniques have also appeared [15, 16]. The techniques in [15] (recently extended in [17]) rely on state-space exploration and determination of the automata corresponding to both the event log and the process model, whilst the technique in [16] is based on computing several subsets of activities and projecting the alignment instances accordingly.

The work in [18] presented the notion of *approximate* alignment to alleviate the computational demands by proposing a recursive paradigm on the basis of the structural theory of Petri nets. In spite of resource efficiency, the solution is not guaranteed to be executable. Alternatively, the technique in [19] presents a framework to reduce a process model and the event log accordingly, with the goal of alleviating the computation of alignments. The obtained alignment, called *macro-alignment* since some of the positions are high-level elements, is expanded based on the information gathered during the initial reduction. Techniques using local search have recently been also proposed very recently [20].

Against this background, the process mining community has focused on divide-and-conquering the problem of computing alignments, as a valid alternative to this problem with the aim of alleviating its complexity without degrading the quality of the solutions found. We turn now our focus to decompositional approaches to compute alignments, which are more related to the research of this paper.

Decompositional techniques have been presented [5, 6, 7] that, instead of computing optimal alignments, they focus on the *crucial problem* of whether a given trace fits or not a process model. These techniques vertically decompose the process model into pieces satisfying certain conditions (so only *valid* decompositions [5], which satisfy restrictive conditions on the labels and connections forming a decomposition, guarantee the derivation of a real alignment). Later on, the notion of *recomposition* has been proposed on top of decompositional techniques, in order to obtain optimal alignments whenever possible by iterating the decompositional methods when the required conditions do not hold [8]. In contrast to the aforementioned vertical decomposition techniques, our methodology does not require this last consolidation step of partial solutions, and therefore can be a fast alternative to these methods at the expense of loosing the guarantee of optimality.

There has been related work also on the use of partial order representations of process models for computing alignments. In [21], unfoldings were used to capture all possible transition relations of a model so that they can be used for online conformance checking. In contrast, unfoldings were used recently in a series of papers [22, 23] to speed-up significantly the computation of alignments. We believe these approaches, specially the last two, can be easily integrated in our framework.

Also, the work of [17] can also be considered a decompositional approach, since it proposes decomposing the model into sequential elements (*S-components*) so that the state-space explosion of having concurrent activities is significantly alleviated. We believe that this work is quite compatible with the framework suggested in this paper, since the model restrictions assumed in [17] are satisfied by the partial models arising from our horizontal decomposition.

Finally, the MapReduce distributed programming model has already been considered for process mining. For instance, Evermann applies it to process discovery [24], whilst [25] applies it for monitoring declarative business processes.

3. Foundations

We denote, $\perp$ the empty set. Let A be a set of elements, we denote A^* the set of all sequences over elements of A. Let $a, b \in (A \cup \{\perp\})^*$ be two sequences. We denote, $a^{\not\perp}$ the sequence a, but omit all elements $\perp$ from a. We write, $a \mathrel{\widehat{=}} b$ if $a^{\not\perp} = b^{\not\perp}$ holds.

3.1. Process Models

In this paper, we model process models and partial models by means of labelled Petri net.

Definition 1 (Labelled Petri net). A labelled Petri net is a tuple (P, T, F, Σ, ℓ) where P and T are finite disjoint sets of places and transitions, respectively, $F : (P \times T) \cup (T \times P) \to \{0, 1\}$ is the flow-relation, Σ is the alphabet, $\ell : T \to \Sigma \cup \{\perp\}$ is the labelling function.

Figure 2 depicts a labelled Petri net. Places are circles and transitions are rectangles. Every transition has a unique name and a label on top. Places and transitions are connected according to the flow-relation.

In Petri nets, there is the so-called firing rule. Transitions of a Petri net can fire changing the state of the net.

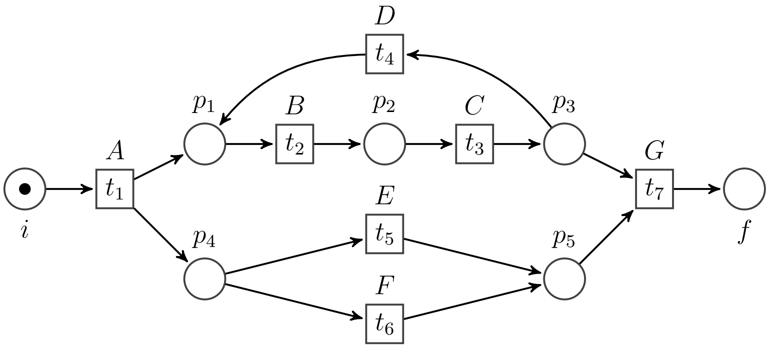


Figure 2: A labelled Petri net.

Definition 2 (Firing Rule). Let $N = (P, T, F, \Sigma, \ell)$ be a labelled Petri net. A function $m : P \to \mathbb{N}_0$ is a marking of N. We define, $\bullet t : P \to \{0, 1\}$ by $\bullet t(p) := F(p, t)$, and $t\bullet : P \to \{0, 1\}$ by $t \bullet (p) := F(t, p)$. A transition $t \in T$ is enabled at marking m if $m \geq \bullet t$ holds. If transition t is enabled, transition t can fire. In this case, we write $m\,[t\rangle$. Firing t changes the marking m to $m' := m - \bullet t + t\bullet$. In this case, we write $m\,[t\rangle\, m'$.

We depict a marking by putting black dots, called tokens, in the places of the marking. For example, Figure 2 depicts the initial state of the labelled Petri net. The initial marking only contains place i once. In this marking, only transition t_1, labelled by A, is enabled. Firing t_1 leads to the marking where i does not carry a token, but both places in the post-set of t_1 carry a token each.

Starting at the initial marking, sequentially enabled sequences of transitions are words of the language of the Petri net. The related traces of labels are the so-called trace-language.

Definition 3 (Language of a Petri net). Let $N = (P, T, F, \Sigma, \ell)$ be a labelled Petri net. A marked Petri net is a tuple (N, m_0, m_f) where m_0 is the initial marking and m_f is the final marking. A sequence $\langle t_1, \ldots, t_n\rangle \in T^*$ is a firing sequence. If there is a sequence of markings $\langle m_1, \ldots, m_{n+1}\rangle$ so that $m_1\,[t_1\rangle\, m_2$, $m_2\,[t_2\rangle\, m_3$, …, $m_n\,[t_n\rangle\, m_{n+1}$ holds, we write $m_1\,[t_1, \ldots, t_n\rangle\, m_{n+1}$.

$$\mathcal{L}(N) := \{\langle t_1, \ldots, t_n\rangle \in T^* \mid m_0\,[t_1, \ldots, t_n\rangle\, m_f\}$$
$$\mathcal{T}(N) := \{\sigma \in \Sigma^* \mid \langle t_1, \ldots, t_n\rangle \in \mathcal{L}(N) \wedge \sigma \mathrel{\widehat{=}} \langle \ell(t_1), \ldots, \ell(t_n)\rangle\}$$

$\mathcal{L}(N)$ is the language of N, $\mathcal{T}(N)$ is the trace-language of N.

In Figure 2, if we assume the final marking where only place f carries one token, for example, $\langle t_1, t_2, t_3, t_6, t_7\rangle$ and $\langle t_1, t_2, t_3, t_5, t_4, t_2, t_3, t_7\rangle$ are words of the language, and $\langle A, B, C, F, G\rangle$ and $\langle A, B, C, E, D, B, C, G\rangle$ are the related traces.

3.2. Conformance checking

Event logs record the behaviour observed from the execution of a business process.

Definition 4 (Trace, Event Log). Let Σ be an alphabet. A sequence $\sigma \in \Sigma^*$ is a trace. A multi-set of traces $L : \Sigma^* \to \mathbb{N}_0$ is an event log.

The classical notion of aligning an event log and a process model was introduced by [4]. An alignment maps a trace of an event log to a firing sequence of the model. An alignment replays the trace and the firing sequence simultaneously, where either the trace, the firing sequence, or both move. Trace and sequence are only allowed to move synchronously if the label of the transition matches the event.

We consider the Petri net depicted in Figure 2 with initial state i and final state f. Aligning the Petri net to, e.g., the trace $\langle A, A, B, C, D, B, C, G\rangle$, we get a lot of possible alignments, where moves of the trace are at the top, and moves of the model are at the bottom of a table.

A	A	B	C	D	B	C	$\perp$	G
t_1	$\perp$	t_2	t_3	t_4	t_2	t_3	t_5	t_7

A	A	B	C	D	B	$\perp$	C	G
t_1	$\perp$	t_2	t_3	t_4	t_2	t_5	t_3	t_7

A	A	B	C	D	B	$\perp$	C	G
$\perp$	t_1	t_2	t_3	t_4	t_2	t_6	t_3	t_7

...

Definition 5 (Alignment). Let $N = (P, T, F, \Sigma, \ell, m_0, m_f)$ be a marked Petri net, σ be a trace of an event log $L : \Sigma^* \to \mathbb{N}_0$, and $\tau \in \mathcal{L}(N)$ be a firing sequence. The set

$$\mathcal{M} := \{(a, t) \in (\Sigma \times T) | \ell(t) = a\} \cup (\Sigma \times \{\perp\}) \cup (\{\perp\} \times T)$$

is the set of legal moves. An element $\langle(a_1, t_1), \ldots, (a_n, t_n)\rangle \in \mathcal{M}^*$ is an alignment of σ and τ iff $\langle a_1, \ldots, a_n\rangle \mathrel{\widehat{=}} \sigma$ and $\langle t_1, \ldots, t_n\rangle \mathrel{\widehat{=}} \tau$ holds.

We define a cost-function to get a cost for every alignment. Every move of an alignment adds to its cost, where asynchronous moves add greater cost than synchronous ones [4].

Definition 6 (Cost-Function). Let $N = (P, T, F, \Sigma, \ell)$ be a labelled Petri net and let $L : \Sigma^* \to \mathbb{N}_0$ be an event log. Let $0 \leq \delta_1 < \delta_2, \delta_3$ hold. We define the cost-function $\lambda_{\delta_1,\delta_2,\delta_3} : \mathcal{M}^* \to \mathbb{N}_0$ as follows: for every alignment $\alpha = \langle(a_1, t_1), \ldots, (a_n, t_n)\rangle \in \mathcal{M}^*$, we define

$$\begin{aligned}\lambda(\alpha)_{\delta_1,\delta_2,\delta_3} := {} & \delta_1 \cdot |\{(a, t) \ \in \alpha \mid a = \ell(t)\}| \\ & + \delta_2 \cdot |\{(a, \perp) \in \alpha\}| \\ & + \delta_3 \cdot |\{(\perp, t) \ \in \alpha\}|\end{aligned}$$

We fix a cost-function to calculate an optimal alignment between a trace of an event log and a process model. In the previous example, if we define the cost of an asynchronous move as 1 and the cost of a synchronous move as 0, the depicted alignments have cost 2.

4. Computing Conformance Checking by Big Data

4.1. Overview of the Approach

The fundamental problem in conformance checking is to align a trace concerning a process model [3]. This problem, known as *the alignment problem*, is a search (which can be very time consuming) to find a model trace similar (according to a cost-function) to the observed trace. Please refer to Section 2 for a complete overview of the current approaches for computing alignments.

Derived from the complexity of the alignment problem, we present a solution based on the creation of simpler problems that can be distributed in a Big Data architecture that aims at facilitating the computation of alignments on the large. In this paper, we assume both process models and logs can be decomposed so that we can take advantage of a Big Data infrastructure, and therefore the fundamental problem of computing an alignment can be distributed over the infrastructure, in a MapReduce fashion [9]. As we will see in Section 5, to instantiate the architecture for a real situation, we build upon our previous work [11] and the case of a partial order decomposition of a process model (cf., Section 4.2). However, while the architecture presented in this section is not tied to any particular conformance checking algorithm, the decomposition technique must be based on the extraction of subtraces and the unfolding of a process model into partial models. Other decompositional approaches available in the literature [5, 6, 7] might be employed, but it leads to changes in the way the *Generate partitions of Problems*, *Map* and *Reduce* activities are implemented. In addition, bear in mind that other decompositional approaches must be capable of forming partitions so that these can be distributed among the nodes of the nodes.

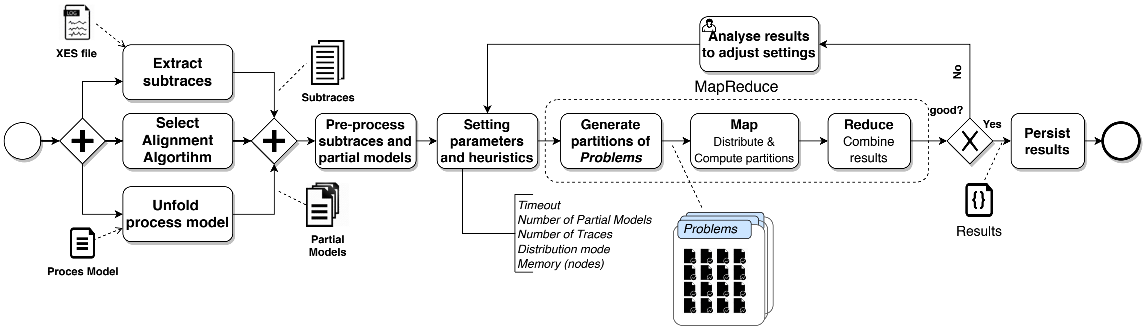


Figure 3: Workflow of the approach.

To determine each of these parameters that describe how the subproblems are created, distributed, solved and combined, Figure 3 summarises the workflow followed in our approach. Since our proposal is not hooked to a concrete alignment algorithm, it has been tested with two very different algorithms to analyse how the type of conformance technique algorithms can affect the Map and Reduce stages. In the first phase, the alignment algorithm is determined as both the subtraces[1] and partial models. Once these aspects are defined, a subtrace and partial model pre-processing are needed (cf., *Pre-process traces and partial models*) to find out certain features used in the heuristics for the later problem distribution. Afterwards, the system is set up (cf., *Setting parameters and heuristics*) in terms of the number of partitions (set of alignment subproblems) to be distributed in each node, the subproblem assignations to each node according to the parameters obtained

[1] As the reader will identify later, in this paper we use subtrace to stress the fact that the methodology proposed is general, although in our particular explanations subtraces will be full traces.

from the previous activity, the thresholds of time used for solving each subproblem, and the threshold of memory in the nodes of each cluster. When the parameters are configured, the MapReduce paradigm can be applied following the three following activities: *Generate partitions of problems*, *Map - Distribution and compute partitions* and *Reduce - Combine results* that will be detailed in Subsection 4.2, 4.3 and 4.4 respectively. The framework follows the idea of MapReduce paradigm as depicted in Figure 4. The input of the problem is the set of alignment problems formed of a combination of a subtrace and a partial model. These alignment problems will be distributed in different divisions solved in each node, where the *Map* function is applied obtaining a map $\langle key, value \rangle$ whose key is the trace and the value the alignment found for the set of traces involved in this subproblem. All the partial solutions represented by maps are then combined.

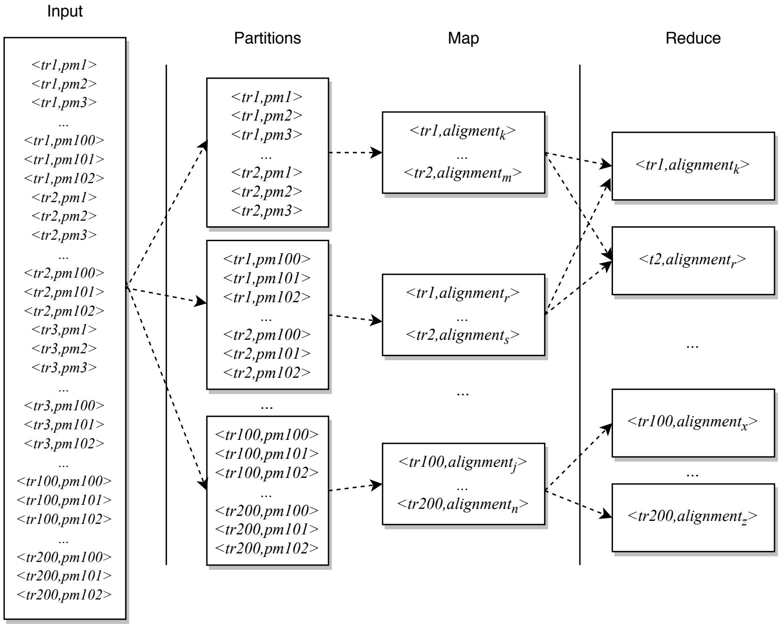


Figure 4: *MapReduce for alignment analysis.*

Our framework provides a mechanism to set up the parameters to perform the alignment analysis in a more efficient way. Therefore, after a solution is found, the parameters (i.e., timeout and number of partitions) can be adjusted to re-execute the alignment analysis reducing the time.

4.2. Generate Partitions of Problems

As indicated in Section 1, we aim at alleviating the complexity of a conformance checking problem by dividing a model into a set of partial models. A partial model covers a part of the trace-language of the original model. Furthermore, a partial model needs to be acyclic and conflict-free.

Definition 7 (Partial Model, Cover). Let $N = (P, T, F, \Sigma, \ell, m_0, m_f)$ and $N' = (P', T', F', \Sigma', \ell', m'_0, m'_f)$ be two marked Petri nets. N' is conflict-free iff $(m\,[t_1\rangle\, m' \wedge m\,[t_2\rangle) \implies m'\,[t_2\rangle$ holds. We call N' a partial model of N iff N' is conflict-free, acyclic, and $\mathcal{T}(N') \subseteq \mathcal{T}(N)$ holds. We call a set of partial models $\{N_1, \ldots, N_n\}$ a cover of N iff $\bigcup_i \mathcal{T}(N_i) = \mathcal{T}(N)$ holds.

Figure 5 depicts a partial model of Figure 2. The depicted marked Petri net is conflict-free, acyclic, and its trace-language is $\{\langle A, B, C, E, G\rangle, \langle A, B, E, C, G\rangle, \langle A, E, B, C, G\rangle\}$. Obviously, this trace-language is a sub-set of the trace-language of Figure 2. Figure 6 depicts another partial model. In this example, transitions t_2 and t_5 carry the label B, transitions t_3 and t_6 carry the label C. Thus, the loop of Figure 2 is unfolded.

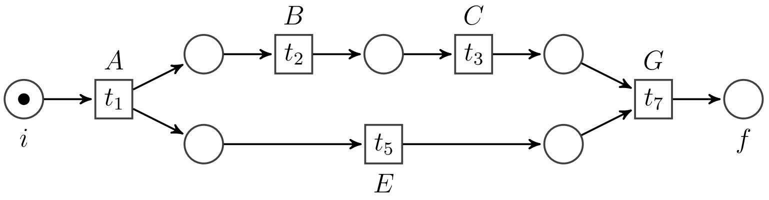


Figure 5: A partial model of Figure 2.

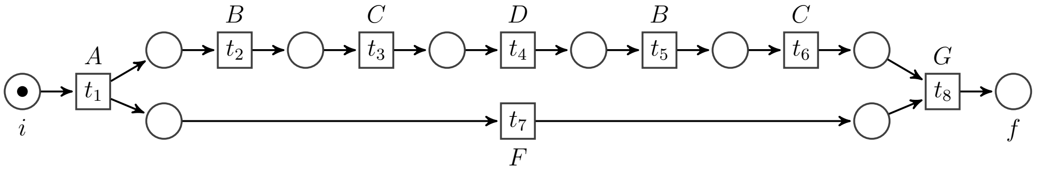


Figure 6: Another partial model of Figure 2.

One straight forward approach to split a Petri net into a cover is to calculate its branching process [26]. It is well-known, that the set of so-called process nets of a branching process is a cover. Remark, the branching process itself can be infinite, but knowing the maximal length of a trace of the log, we can always determine a sufficient depth to calculate an appropriate prefix of a cover for the alignment problem at hand. In the literature, there is a rich body of different approaches calculating finite representations of an infinite branching process in a reasonable time, i.e. [27].

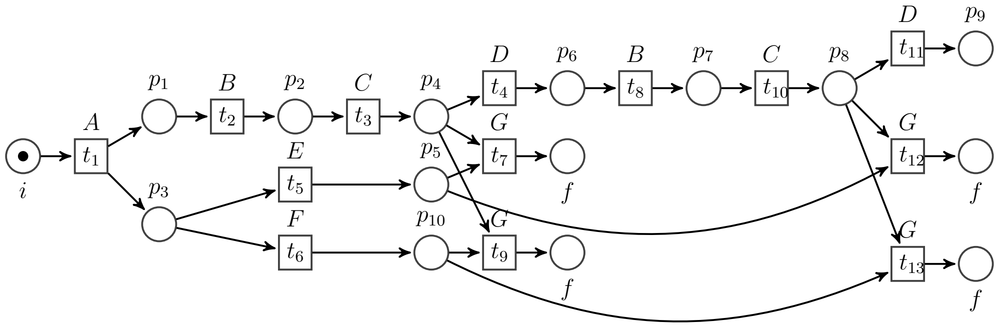


Figure 7: A prefix of the branching process of Figure 2.

Figure 7 depicts a prefix of the branching process of the model of Figure 2. This acyclic labelled net is able to execute all traces up to length seven of the original model. It is a prefix because the looping behaviour of transitions B, C, and D generate infinite behaviour. In a branching process, a model is unfolded so that all places have at most one preceding transition. For instance both places (cf., p_5 and p_{10}) behind transitions labelled by E and F. In Figure 2, this pair is only one place (cf., p_5). The same holds for all places before transitions labelled by B and places behind transitions labelled by D. In the original model, they are just one place (cf., p_1 in Figure 2). Thus, conflicts and cycles are unfolded. Every connected subnet of a branching process, so that all places have at most one subsequent transition, are called occurrence nets. By construction, these occurrence nets are partial models. For example, the set of transitions $\{t_1, t_2, t_3, t_5, t_7\}$ with all connected places form the partial of Figure 5. Transitions $\{t_1, t_2, t_3, t_4, t_6, t_8, t_{10}, t_{13}\}$ are the partial model of Figure 6.

Occurrence nets of branching processes are only one of many possibilities to decompose a model horizontally. Log-based unfolding [28] or token flow-based unfolding [27] can generate similar decompositions. In the more general setting of the paper, we just required the set of partial models is given as a set of acyclic, conflict-free marked Petri nets.

Every trace of a partial model can be replayed by its original model. Thus, for every alignment of the partial model, there is a related alignment in the original model having the same cost. For example, the firing sequence $\langle t_1, t_2, t_3, t_4, t_5, t_7, t_6, t_8 \rangle$ of the partial model depicted in Figure 6 can be replayed by the original model depicted in Figure 2 by $\langle t_1, t_2, t_3, t_4, t_2, t_6, t_3, t_7 \rangle$. Obviously, related (replayed) alignments have the same cost:

$$\begin{array}{|c|c|c|c|c|c|c|c|c|}
A & A & B & C & D & B & \perp & C & G \\
\perp & t_1 & t_1 & t_3 & t_4 & t_5 & t_7 & t_6 & t_8 \\
\uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\
\perp & t_1 & t_2 & t_3 & t_4 & t_2 & t_6 & t_3 & t_7
\end{array}$$

If a set of partial models covers a Petri net, for every alignment of the original model, a partial model covering this alignment, because the set of partial models can replay every trace of the original model. Thus, we can calculate an optimal alignment for the original model by calculating an optimal alignment for the set of partial models.

The division of the problem in smaller partitions is the base of the application of the MapReduce paradigm. Thereby, it is necessary to tackle the problem of the partitioning of an alignment problem (AP) into a set of subproblems by distributing the set of traces of an event log and the set of partial models extracted from a process model. Firstly, and following Figure 3, the process model and log are decomposed into subtraces and partial models that can be analysed independently obtaining the alignment in a more efficient way.

Definition 8. (Decomposition, Alignment Subproblem). Let AP be an alignment problem aligning a set of traces $Tr = \{tr_1, tr_2, \cdots, tr_n\}$ to a model M. Let $Pm = \{pm_1, pm_2, \cdots, pm_m\}$ be a cover of M. We call every element $pr \in (Tr \times Pm)$ an *alignment subproblem.* We write $AP = (Tr \times Pm)$ and call $(Tr \times Pm)$ a *decomposition* of AP into $n \cdot m$ subproblems.

In some ideal scenario with unlimited resources, we can solve each alignment subproblem independently and in parallel. In this case, the total runtime needed to solve AP is the time spent in the most complex subproblem plus the time spent to combine the partial alignments. Here, we would need as many nodes as subproblems to process all subproblems in parallel.

In real-life applications, the number of subproblems is much too high to just generate a node for every problem. Thus, different subproblems need to share nodes. To control the distribution of subproblems to nodes, we partition the set of all possible subproblems into groups of subproblems sharing the same features. Features are the involved trace and partial alignments calculated in other subproblems. How to properly group and distribute subproblems to calculate solutions efficiently is analysed bellow.

Definition 9 (Partitions). Let Tr be a set of traces. We call a set of disjoint sets of traces $\{Tr_1, Tr_2, \ldots, Tr_n\}$ a *partition* of Tr if $Tr = \bigcup_{i=1}^{n} Tr_i$ holds. Let Pm be a set of partial models. We call a set of disjoint sets of partial models $\{Pm_1, Pm_2, \ldots, Pm_m\}$ a *partition* of Pm if $Pm = \bigcup_{i=1}^{m} Pm_i$ holds. Fix a set Tr_i of the partition of Tr and fix a set Pm_j of the partition of Pm. We call $pr_{(i,j)} := (Tr_i \times Pm_j)$ a partition of the alignment subproblems.

Partitions define sets of alignment subproblems. Each set will result in a partial alignment. Figure 8 depicts schematically partitions of Tr and Pm and the resulting sets of alignment subproblems. Remark, every trace is handled by nodes of the same trace-partition (in one row). Section 4.3 will discuss how this speeds-up the computation by taking advantage of other partial alignments.

In the next subsection, we will discuss the distribution of every partition of alignment problems following the MapReduce strategy [9], which is a programming model to support the parallel computing for large collections of data.

4.3. Map - Distribute and compute alignment problem partitions

The map function is based on solving smaller problems, obtaining partial solutions that will be combined later. The algorithm used in the map function is represented in Figure 9. It receives a partition of subproblems and creates a dictionary of *partial solutions* with default values. Then, for each subproblem, it makes a lower bound estimation for the possible alignment that can be taken before it will be solved. This estimation is employed to sort the subproblems to solve in the same node (sequentially solved). The estimation is obtained by comparing model and trace: (1) checking the size of the trace w.r.t. the maximum number events that can be extracted from the submodel (e.g., if the trace has 100 events and the longest trace generated by the submodel is 90, the alignment must be at least 10); (2) the events that

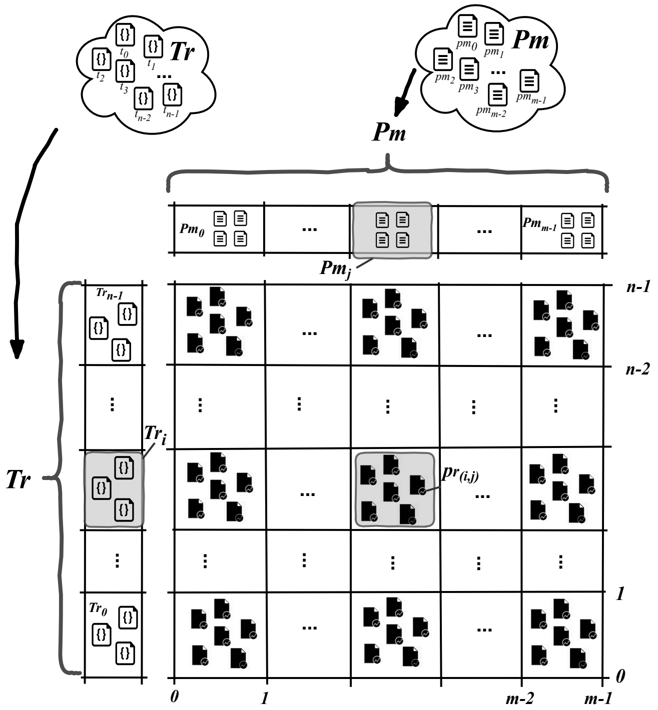
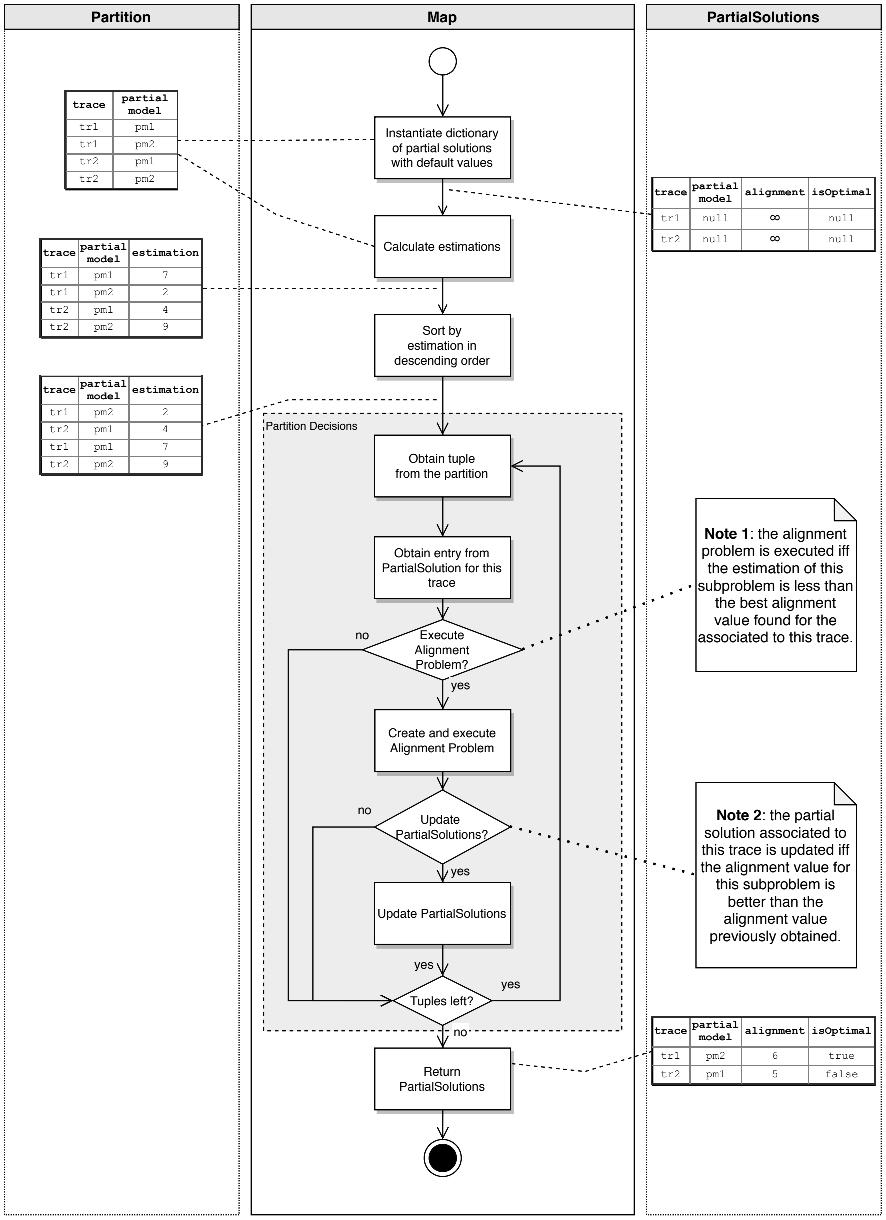


Figure 8: Partitions and sets of subproblems.

occur in the trace but not in the model and vice versa; and (3) considering the number of occurrences of events with regard to the submodel (e.g., the event A is repeated three times in the trace but two in the submodel, the alignment cost must be at least 1). These values are calculated and aggregated to generate an estimation as a lower value which the alignment can take. If a solved model has found out a better alignment than the estimation, it makes no sense to evaluate the rest of subproblems with a worse estimation. The partition of subproblems is then sorted by estimation in ascending order. It is crucial for optimising the execution time since it prevents the alignment process from executing subproblems that would not beat the best alignment found up to that moment (cf., Note 1 in Figure 9). If a new alignment is obtained, then the partial solution associated with that trace is updated iff the new alignment value is better than the previous one (see Note 2 in Figure 9). Note that the partial solutions have an attribute called *isOptimal*. This will be true iff it is possible to guarantee that the solution associated with this trace is the optimal one. If the isOptimal attribute of any of the subproblems


Partition
Map
PartialSolutions
Instantiate dictionary of partial solutions with default values
Calculate estimations
Sort by estimation in descending order
Partition Decisions
Obtain tuple from the partition
Obtain entry from PartialSolution for this trace
Execute Alignment Problem?
no
yes
Create and execute Alignment Problem
Update PartialSolutions?
no
yes
Update PartialSolutions
yes
Tuples left?
yes
no
Return PartialSolutions

trace	partial model
tr1	pm1
tr1	pm2
tr2	pm1
tr2	pm2

trace	partial model	estimation
tr1	pm1	7
tr1	pm2	2
tr2	pm1	4
tr2	pm2	9

trace	partial model	estimation
tr1	pm2	2
tr2	pm1	4
tr1	pm1	7
tr2	pm2	9

trace	partial model	alignment	isOptimal
tr1	null	∞	null
tr2	null	∞	null

Note 1: the alignment problem is executed iff the estimation of this subproblem is less than the best alignment value found for the associated to this trace.

Note 2: the partial solution associated to this trace is updated iff the alignment value for this subproblem is better than the alignment value previously obtained.

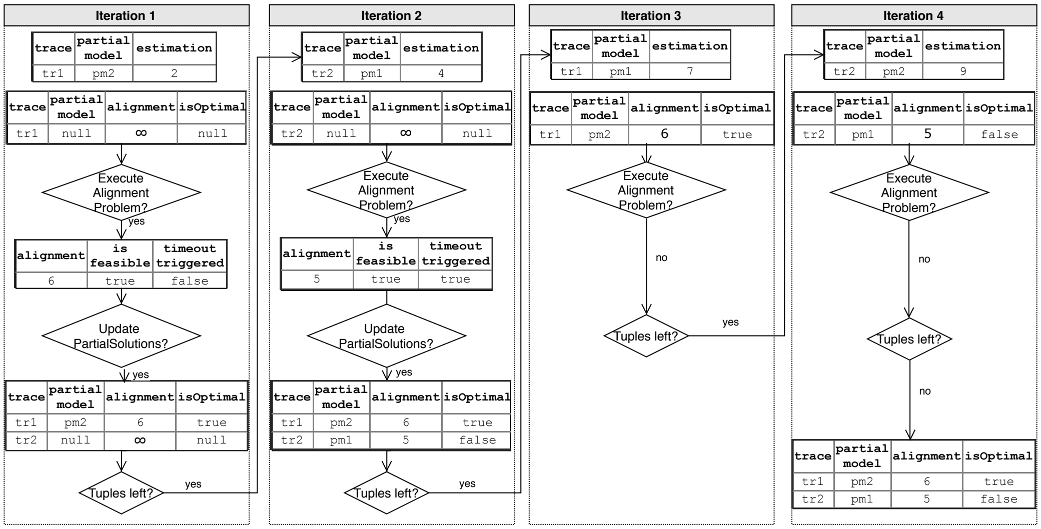
trace	partial model	alignment	isOptimal
tr1	pm2	6	true
tr2	pm1	5	false

Figure 9: Activity diagram to describe the Map algorithm.

that were executed was marked as *false* because of the timeout was reached, we cannot guarantee that the solution to any other subproblem associated to that trace is the optimal one, because any other subproblem with better estimation value previously executed could have returned a better alignment value if the timeout had not been reached.

In order to illustrate the algorithm, Figure 10 presents the iteration of the *partition* presented in Figure 9. Remark that at this point, the partitions are already sorted by estimation. There are four elements to process (hence, there are four iterations). In *Iteration 1* the subproblem $\langle tr1, pm2\rangle$ is processed. Then, the alignment process is executed because the estimation yielded a value of 2, which could improve the partial solution found until the moment (∞). Once computed the alignment value (6), the partial solution for $tr1$ $\langle tr1, pm2, 6, true\rangle$ is stored. In *Iteration 2*, we have a similar situation with $\langle tr2, pm1\rangle$, where the partial solution is updated as well after obtaining an alignment of 5. However, the timeout was fired, so the alignment cannot be guaranteed to be optimal ($\langle tr2, pm1, 5, false\rangle$). However *Iteration 3* does not execute the alignment process nor updates the partial solution previously obtained for $tr1$, since the estimation for the subproblem $\langle tr1, pm1\rangle$ is greater than the best optimal computed alignment.

In *Iteration 4*, we have the same situation, so the partial solution formerly found for $tr2$ is not updated.



trace	partial model	estimation
tr1	pm2	2

trace	partial model	alignment	isOptimal
tr1	null	∞	null

alignment	is feasible	timeout triggered
6	true	false

trace	partial model	alignment	isOptimal
tr1	pm2	6	true
tr2	null	∞	null

trace	partial model	estimation
tr2	pm1	4

trace	partial model	alignment	isOptimal
tr2	null	∞	null

alignment	is feasible	timeout triggered
5	true	true

trace	partial model	alignment	isOptimal
tr1	pm2	6	true
tr2	pm1	5	false

trace	partial model	estimation
tr1	pm1	7

trace	partial model	alignment	isOptimal
tr1	pm2	6	true

trace	partial model	estimation
tr2	pm2	9

trace	partial model	alignment	isOptimal
tr2	pm1	5	false

trace	partial model	alignment	isOptimal
tr1	pm2	6	true
tr2	pm1	5	false

Figure 10: Execution trace of the Map function.

4.4. Reduce - Combining Alignment Problem Result

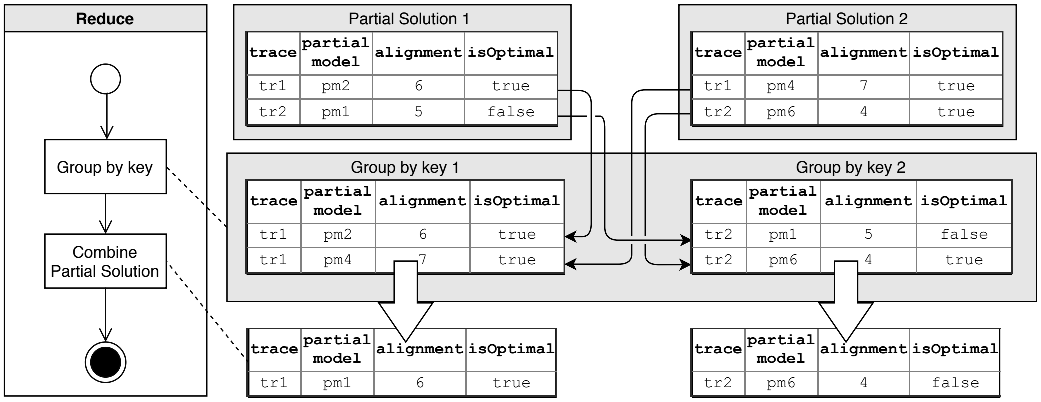
The reduce phase is responsible for combining the partial solutions that are generated during the map phase. Bear in mind that each partition yielded a set of partial solutions, with the following information: trace, partial model, alignment value, and an indicator pointing out whether the solution is optimal or it is impossible to ensure that the obtained alignment is the optimal one. In this phase, all the partial solutions corresponding to the same trace are combined, and the one with the best alignment value is selected as the best solution for such a trace.

Figure 11 depicts the reduce process, including some partial solutions to follow the execution. The proposal is based on the function know as *reduceByKey*, that groups by the key of the data provided by the map function, and then apply a function in order to combine the values associated with each key. For the alignment problem, the reduce phase groups the partial solutions with the same key (i.e., with the same trace), and combines every partial solution in the same group obtaining another partial solution. Following the example of Figure 11, the tuples of *Partial Solution* 1 and 2 are grouped by trace ($tr1$ and $tr2$). The tuples in each of these groups are combined returning a single tuple in each case. The new obtained partial solution follows the form:

- **trace:** the trace which was employed to create the groups and shared by every tuple in the group.
- **partial model:** the partial model whose alignment is minimal for that trace.
- **alignment:** the minimal alignment of every tuple.
- **isOptimal:** the $\wedge$ combination of the *isOptimal* values of every tuple. It means that it is *false* for a tuple, the others related to the same trace will be marked as *false* as well, since it is impossible to ensure that the found alignment is optimal because the problem has not been fully analysed.

5. Interchangeable Solutions for Encoding Alignment

The MapReduce algorithm presented in the previous section can be applied to different types of alignment techniques, subtraces and partial models.



trace	partial model	alignment	isOptimal
tr1	pm2	6	true
tr2	pm1	5	false

trace	partial model	alignment	isOptimal
tr1	pm4	7	true
tr2	pm6	4	true

trace	partial model	alignment	isOptimal
tr1	pm2	6	true
tr1	pm4	7	true

trace	partial model	alignment	isOptimal
tr2	pm1	5	false
tr2	pm6	4	true

trace	partial model	alignment	isOptimal
tr1	pm1	6	true

trace	partial model	alignment	isOptimal
tr2	pm6	4	false

Figure 11: Execution trace of the Reduce function.

Several are the algorithms that have faced the conformance checking problem in the context of business processes (cf., Section 2 for a full description). In this section, we included two, the A* algorithm as an example of a classical algorithm developed by other authors [4] and a novel implemented solution based on the Constraint Programming Paradigm. Both algorithms have the same objective (i.e., finding out the alignment). However, we have included the Constraint Programming Paradigm since it let us incorporate some special features, such as to restrict the domain of the possible value where the alignment can be found, and determining a maximum time of resolution per subproblem returning the best solution found until the timeout.

5.1. Alignment based on the A Algorithm*

One of the most relevant solutions to computing alignments found in literature is the A* algorithm [4]. It has been successfully employed as a feasible approximation to find out the optimal alignment between the process model and traces [3]. Basically, the model and trace are combined into a synchronous product. Figure 12 illustrates the synchronous product, showing the partial model, obtained from a cover (see Definition 7) given in Figure 6, and the log *trace*: $\langle A, B, E, D, C, B, C, F, G \rangle$.

The simplest way to compute alignment is to build the reachability graph (cf., Definition 7, [3]) from the synchronous product, and then to find out the shortest path from an initial marking to final marking. However, the construction of the full reachability graph is not always possible due to the state space explosion problem. To overcome that problem, the reachability graph is built in pieces. The A* algorithm is efficiently used (cf., chapter

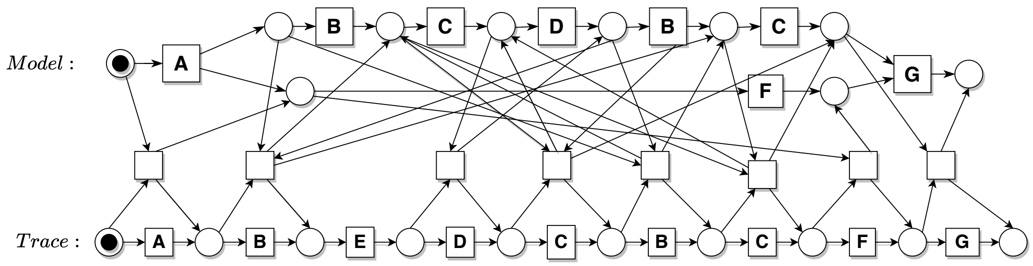


Figure 12: Example of synchronous product of model and trace.

7.3, Procedure 2 [3]) to compute the shortest path. The core of the A* algorithm relies on a heuristic function, $f(m) = g(m) + h(m)$, that guides the search, where $g(m)$ is the cost of the path from the initial marking to m. For instance, for any m reachable state, A* must determine $h(m) \leq h^*(m)$, where $h^*(m)$ is the shortest path from m to the final marking. There are cases in which A* fails to compute the alignments since it is very complex and time-consuming (e.g., in models with a very high level of parallelisms [17]).

Our approach integrates the implementation of the A* algorithm provided by the Python library *PM4Py*[2].

5.2. Alignment based on Constraint Programming

The Constraint Programming paradigm is a general-purpose technique that can be applied to optimise problems. Since the alignment problem is an optimisation problem that can be distributed, we considered relevant the incorporation of this novel solution in our framework as an evolution of a previous proposal [11]. Moreover, since the alignment computation can be modelled as a variable and restrictions whose domain can be bounded, and the breakage of the model in submodels, we consider relevant to analyse how the former resolution of subproblems can be used to tight the possible domain to analyse in further resolutions. The partial model, obtained from a cover (see Definition 7) given in Figure 6, and the log *trace*: $\langle A, B, E, D, C, B, C, F, G\rangle$ are used as a running example to illustrate the encoding based on Constraint Programming. The partial model can contain concurrent paths, that is, there would be and-splits that divide the execution into various branches that can be executed in parallel.

[2] PM4Py: `https://pm4py.fit.fraunhofer.de/`

In our approach, the computation of the alignment problem of a log trace and a partial model is encoded as a Constraint Problem as an improvement of time and resources of the proposal presented in [11]. Thus, the information extracted to the partial model and the trace such as the name of transitions, events, the execution order and their possible positions are translated to variables, constraints and an objective function of a Constraint Optimisation Problem (COP).

5.2.1. Constraint and Optimisation Problems in a nutshell

Constraint programming is a paradigm that permits the declarative description of the constraints that determine a problem [29][30]. Constraint Programming brings together a set of algorithms to find out the solutions of a problem described.

Definition 10. Constraint Satisfaction Problem (CSP) is a defined by a 3-tuple $\langle X, D, C\rangle$, where $X = \{x_1, \ldots, x_n\}$ is a finite set of variables, $D = \{d(x_1), \ldots, d(x_n)\}$ is a set of domains of the values of the variables, and $C = \{C_1, \ldots, C_m\}$ is a set of constraints. Each constraint C_i determines relations R between a subset of the variables $V = \{x_i, x_j, \ldots, x_l\}$.

A constraint $C_i = (V_i, R_i)$ simultaneously specifies the possible values of the variables in V that satisfy R. Let $V_k = \{x_{k_1}, \ldots, x_{k_l}\}$ be a subset of X, and an l-tuple $(x_{k_1}, \ldots, x_{k_l})$ from $d(x_{k_1}), \ldots, d(x_{k_l})$ can therefore be called an *instantiation* of the variables in V_k. An instantiation is a solution iff it satisfies the constraints C. The CSP solvers permit to search for one tuple of instantiation of one, multiple of all these values, according to the requirement of the problem.

An example of its applicability in the alignment context, it is by representing the order relation existing in the models and the traces, as found in Figure 6. By using a set of variables to represent the order of the events, and satisfying the relative constraints of the activities that appear in the partial model, the alignment can be encoded in the following CSP.

```
// Variables
$position_A$, $position_B$, $position_C$ . . . in the domain {0..trace.lenght-1}
// Constraints of the log trace
$position_A == 0$
$position_C == 1$
$position_B == 2$
$position_{AD} == 3$
. . .
$position_{AZ} == 14$
// Constraints of the partial model
$position_A < position_C$
$position_C < position_{AC}$
$position_C < position_{AF}$
$position_{AC} < position_{AD}$
. . .
```

If the model and the event cannot be aligned, this CSP will not be satisfied. However, no more feedback about the level of misalignment is provided by the resolution of the CSP. In this case, a Constraint Optimisation Problem (COP) is able to know the minimal distance between the partial model and the log observed since a COP is a CSP that includes an optimisation function. Only the solution of the CSP that satisfies the optimal function could be the solution of the COP.

Constraint Optimisation Problems (COPs) have already been used to detect the alignment between the expected and the observed behaviour in model-based diagnosis [31, 32], and specifically, when the behaviour is described by means of business process models [33, 34, 35]. These works used the concept of *reified constraints* as a mechanism to assign a Boolean value to the constraints included in the model [33], being possible that a constraint that cannot be satisfied during the CSP resolution can be relaxed. Since the idea is to find out the minimal distance between the model and the log, these constraints relaxed must be the minimum number, defined as the objective of the function to optimise. Following the previous example, the below COP is created where the Ref variables regard to the reified constraints.

```
// Variables
$Ref_A$, $Ref_C$, $Ref_C$ ... in the domain {0..1}
$position_A$, $position_B$, $position_C$ ... in the domain {0..trace.lenght-1}
// Constraints of trace
$position_A == 0$
$position_C == 1$
$position_B == 2$
$position_{AD} == 3$
...
$position_A == 14$
// Constraints of the model
$Ref_A \wedge Ref_C \implies (position_A < position_C)$
$Ref_C \wedge Ref_{AC} \implies (position_C < position_{AC})$
$Ref_C \wedge Ref_{AF} \implies (position_C < position_{AF})$
$Ref_{AC} \wedge Ref_{AD} \implies (position_{AC} < position_{AD})$
...
maximize($Ref_A + Ref_C + \ldots + Ref_{AF}$)
```

Albeit the idea of the COP modelling follows the previous COP, in the following subsection we approach the definition included in Section 3 in relation with a COP to find out the alignment between a partial model and a log trace.

5.2.2. Constraint Optimisation Problem for Solving an Alignment Subproblem

Our proposal builds the COP from the perspective of the placement of the events in a positional order that satisfies both the log trace order and the partial model, but if this is not possible some of the constraints are skipped from the COP firing reified constraints. Then, the structure of the COP is as shown in Figure 13.

As defined above, a COP is composed of a set of variables, a set of constraints and an objective function. It is important to take into account the possibility that an event can appear more than once in a log trace derived, for example, from an unfolding process. In this case, a relabelling of the events is necessary to differentiate the variables that represent one or another, although some constraints must be included to express that they can represent the same transition. In detail a COP is formed of:

- *Variables for the Log Events*: for each event in the log trace, two variables are created:
    - *Position (pos):* Integer variable with a domain between 0 and the number of events, that is, all the different locations of the events. This domain represents all the possible positions with respect to

% Variables - Log Events

$A.dev = Boolean$

$B1.dev = Boolean$

$E.dev = Boolean$

$C1.dev = Boolean$

$D.dev = Boolean$

$B2.dev = Boolean$

$C2.dev = Boolean$

$F.dev = Boolean$

$G.dev = Boolean$

$A.pos = integer(0, 8)$

$B1.pos = integer(0..8)$

$C1.pos = integer(0..8)$

$E.pos = integer(0..8)$

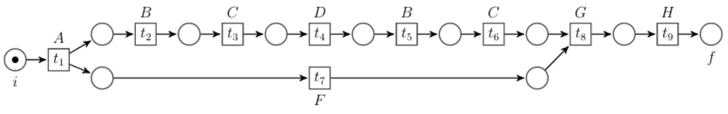
$D.pos = integer(0..8)$

$B2.pos = integer(0..8)$

$C2.pos = integer(0..8)$

$F.pos = integer(0..8)$

$G.pos = integer(0..8)$



$Trace: \quad \langle A, B1, E, D, C1, B2, C2, F, G \rangle$

% Constraints - Enforce Partial Model

$\neg A.dev \wedge \neg B1.dev \implies A.pos < B1.pos$

$\neg B1.dev \wedge \neg C1.dev \implies B1.pos < C1.pos$

$\neg C1.dev \wedge \neg D.dev \implies C1.pos < D.pos$

$\neg D.dev \wedge \neg B2.dev \implies D.pos < B2.pos$

$\neg B2.dev \wedge \neg C2.dev \implies B2.pos < C2.pos$

$\neg C2.dev \wedge \neg G.dev \implies C2.pos < G.pos$

$\neg A.dev \wedge \neg F.dev \implies A.pos < F.pos$

$\neg F.dev \wedge \neg G.Dev \implies F.pos < G.pos$

$B1.pos \neq B2.pos$

$C1.pos \neq C2.pos$

% Constraints - Enforce Log Trace

$\neg A.dev \implies A.pos == 0$

$\neg B1.dev \implies B1.pos == 1 \vee B1.pos == 5$

$\neg C1.dev \implies C1.pos == 4 \vee C1.pos == 6$

$\neg D.dev \implies D.pos == 3$

$\neg B2.dev \implies B2.pos == 1 \vee B2.pos == 5$

$\neg C2.dev \implies C2.pos == 3 \vee C2.pos == 6$

$\neg F.dev \implies F.pos == 7$

$\neg G.dev \implies G.pos == 8$

$E.dev == true$

% Objective Function - Alignment

$$MINIMIZE(A.dev + B1.dev + E.dev + C1.dev + D.dev + B2.dev + C2.dev + F.dev + G.dev)$$

Figure 13: COP for the example of Figure 6.

the partial model. In the running example, all the variables get a domain from 0 to 8 since 9 is the total number of events, although the event E is not in the partial model and the transition H in the partial model is not included in the events.

- *Deviation (dev):* Boolean variable which represents the correct or incorrect order of the event according to the model. Thus, semantically the *false* value indicates that the event is aligned with the partial model, and the *true* value otherwise. These variables are the key to obtain the log and model moves in the alignment calculation as it will be seen in the objective function. These variables are also used to enable/disable to fire the reified constraints of the COP.

- *Constraints to enforce Log Traces*: According to the log-relation of the events in the trace, the events are enforced to take those positions. Thus, a set of reified constraints are build to represent conditions of the position of the events with respect to the log trace. For instance, the event A occurs first:

$$\neg A.dev \implies A.pos == 0 \tag{1}$$

In case the event does not occur in the partial model, it is a deviation, then a constraint is included to force the establishment of a *true* value for the *dev* variable of the event, as occurs with E event:

$$E.dev == true \tag{2}$$

In the case of the repeated events, the COP must evaluate all the possibilities of occurrence, as in the case of *B1* and *B2*. The reified constraint must consider the two possibilities, as follows:

$$\neg B1.dev \implies B1.pos == 1 \vee B1.pos == 5 \tag{3}$$

- *Constraints to Enforce Partial Model Run*: These reified constraints represent conditions of the position (*pos*) of the events with regard to the partial model. The reified constraint describes whether an event can be aligned according to the partial model or not. According to the flow-relation of the partial model, we build reified constraints to represent the related 'later than'-relations between the occurrences of transitions. Take into account that in the partial models used in our proposal, the XORs are eliminated, and every transition of the model participate in any correct event log. Therefore, the next constraint is an example of this type of reified constraints:

  $$\neg A.dev \wedge \neg B1.dev \implies A.pos < B1.pos \tag{4}$$

  The reified constraint means, if events A and *B1* are aligned with the model, the value assigned to *pos* of the event A have to be lower that the values of *pos* of the event *B1*.

  In case of repeated events (e.g., *B1* and *B2*), extra constraints have to be included to avoid the occurrence of them at the same positions:

  $$B1.pos \neq B2.pos \tag{5}$$

  When a transition in the model is not supported by the execution of an event (taking into account that in the partial model supported by the proposal every transition must be involved in a correct trace since only and-branches are included), constraints related to this transitions are not added, albeit a misalignment will be included (a model move). See below for a description of how this is computed.

- *Optimisation function*: The objective function tries to find out a solution that minimise the number of deviations. The Boolean variables are considered as Integer, that is, the *false* is the 0 value and *true* is 1 value. As shown in Figure 13 the objective function is the minimisation of the sum of all deviation variables of our problem. Thus, to find a solution (an assignment) where all the *dev* variables are fixed as *false*, means that every event of the log trace is aligned with the partial model. In case of any *dev* variable is fixed to *true*, the alignment will be, at least, the number of *dev true* values.

This COP allows to detect the possible deviations between the partial models and the events:

- **Log moves:** The log moves are determined by consulting the *false* values fixed in the deviation (*dev*). If the *dev* variable of an event reached a *false* value then this event is not producing a log move. When an event does not occur in the partial model, this situation is a log move, thereby this situation is controlled by forcing the *true* value in the *dev* variable of the event.
- **Model moves:** Model moves occur when there exists a transition in the partial model that does not occur in the log trace. This situation is easy to identify since a partial model is conflict-free (see Definition 7), meaning that all the transitions must occur in a partial model run. Then, we only have to penalise that situation as a model move by adding one to the alignment cost function.

After the COP resolution, the log and model moves are known, therefore the alignment cost function is determined as follows:

$$alignment = \underbrace{\sum_{e_i \in Tr}^{i} e_i.dev}_{log\ moves} + \underbrace{\sum_{e_i \in Pm \wedge e_i \notin Tr}^{i} 1}_{model\ moves} \tag{6}$$

For the example, the COP reached two optimal solutions where the alignment is equal to 3, one value for the *E.dev=true*, other due to the *C1.dev=true* (*D.dev=true* in the another equivalent solution) since it is impossible to locate it according to the log trace, and another because H does not occurs in the log trace.

The inclusion of the *time limit* is crucial in Constraint Programming since the solvers return partial solutions during its execution. If the solver is stopped by the time limits, we could have, at least, the best option found until the moment, although it can or cannot be the global optima since the search space has not been completely solved.

6. Experiments and Evaluation

In order to evaluate our proposal, we have performed different tests to compare the local and distributed version of the A* algorithm and the COP-based approach on computing alignments. The structure of section is the following:

- Design of the architecture and the technology stack to support our framework (see Subsection 6.1).

- Selection of a set of representative datasets (see Section 6.2) that includes examples where: (1) local (standalone) resolutions have a better performance than the distributed ones for the same type of algorithm; (2) the distribution of A* implies a better performance than the local alignment, and (3) the application of the distribution of Constraint Programming problems improves the resolution in comparison with the distributed A*.

- Analysis of the configuration parameters for the subproblem distributions, and it can affect the alignment resolution. To do that, the datasets and various setups for the parameters are tested (see Section 6.3). For the evaluation, we just consider the performance in terms of the time[3] related to the computation of the alignments through the approach presented in this paper.

[3] The *Elapsed Real Time* (*ERT*) as the time from the start of the execution of a program to the end of it.

6.1. Architecture

We propose the use of a three-layer architecture, as shown in Figure 14. Additionally, we include information about the employed technological stack to instantiate this architecture and to perform the experiments.

- *Storage Layer*. The role of this layer is to store the log and process model so that it can be accessed by the rest of the nodes that compose the system. In our particular implementation, it is based on Hadoop HDFS[4], which is a distributed storage system.
- *Persistence Layer*. This layer is intended to store the results of the alignments. Our implementation relies on the NoSQL database MongoDB[5].
- *Computing layer*. It is intended to perform the computing operations related to the generation and distribution of partitions and computing alignments. In our implementation, it is based on Apache Spark[6], which is a distributed computing framework which enables users to implement applications for the distribution of tasks and Big Data processing.

The mechanism for the generation and distribution of partitions explained in Section 4.2 has been implemented in Apache Spark. As aforementioned, we have integrated PM4Py[7] platform for the computation of alignments using the A* algorithm. On the other hand, the COPs have been implemented with ILOG CPLEX[8] although other solvers can be applied. Regarding the architecture of the Apache Spark cluster (i.e., the architecture of the Computing layer), it is composed of five nodes. Each node is configured with 16GB of RAM and 4 CPUs. This cluster is composed of three types of nodes:

- *Cluster manager*. It is responsible for monitoring and assigning resources among the nodes of the cluster. There is one node entirely dedicated to this task.

[4] HDFS: `https://hadoop.apache.org/`

[5] MongoDB: `https://www.mongodb.com/`

[6] Apache Spark: `https://spark.apache.org/`

[7] PM4Py: `https://pm4py.fit.fraunhofer.de/`

[8] IBM-ILOG CPLEX: `https://www.ibm.com/products/ilog-cplex-optimization-studio`

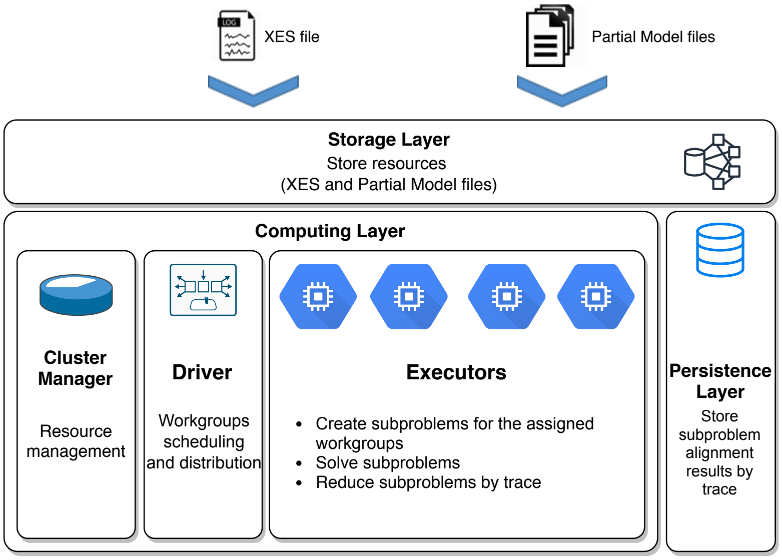

XES file
Partial Model files
Storage Layer
Store resources
(XES and Partial Model files)
Computing Layer
Cluster Manager
Resource management
Driver
Workgroups scheduling and distribution
Executors
• Create subproblems for the assigned workgroups
• Solve subproblems
• Reduce subproblems by trace
Persistence Layer
Store subproblem alignment results by trace

Figure 14: Cluster Architecture.

- *Driver program.* This node is responsible for distributing the tasks among the Executor nodes. Regarding our implementation, it will schedule the partitioning process, assigning the partitions and the tasks related to them to the executor nodes. In our case, the driver program is configured for using 8GB of RAM and 1 CPU by default, and it is run on one of the five nodes of the cluster.

- *Executor nodes.* These nodes execute the tasks assigned by the driver program. They receive the partitions and execute their corresponding tasks. We configured each executor node with 8GB of RAM and 4 CPUs by default. All the four nodes of the cluster host one executor.

Both the source code with the implementation of the framework and the datasets used for the experimentation are available for the community at `http://www.idea.us.es/confcheckingbigdata/`.

6.2. Setting Experiments

Five benchmark datasets have been used for the experiments. These are composed of a set of files in XES format as event logs and a set of partial models in Labelled partial orders (LPO) format [36]. For a better understanding, the LPO format is a simplification but compatible with the

PNML format. An LPO represents a run of a place/transition Petri net if it is enabled w.r.t the net. It makes sense when the events of the LPO modelling transition occurrences can fire in the net respecting the concurrency and dependency relations given by the LPO.

The event logs and Petri nets used to illustrate how our proposal works with different size of problems are extracted from [19, 37, 38], whereas the partial models are of the unfolding of Petri nets. Table 1 summarises the dimensions of the datasets employed for this evaluation in terms of (1) the event logs (number of cases, events, variants and size); (2) Petri net (number of places, transitions, arcs and the Cardoso metric (CFC) [39]), and; (3) partial models (number of unfolded partial models, number of problems to compute to solve the alignment problem and size, regarding the number of problems to tackle).

Once traces and partial models are combined, the total number of subproblems (cf., Num. of Problems of Table 1) is derived from the application of the Cartesian product and the theoretical required storage space of them. The objective of this table is giving an idea about the complexity involved for solving all the problems of each dataset, especially *M5* and *prGm6*, in which more than five and forty millions of subproblems must be solved. Approximately, the approach must manage a total of 96GB and 2.5TB of data volume, as appeared for *M5* and *prGm6* in the table. The distribution of the alignment computation can imply the transferring of a great amount of data among the nodes, producing a negative impact on the performance. For this reason, the access to traces and partial models have been centralised and they are accessed by a unique identifier assigned to each one of them.

Dataset	Event Log				Petri Net				Partial Models		
	Cases	Events	Variants	Size (MB)	Places	Transitions	Arcs	CFC	Num. of Models	Num. of Problems	Size (MB)
$M2$ [19]	500	8,809	500	2.20	34	34	160	36	102	51,000	509.4
$M5$ [19]	500	17,028	500	4.2	35	33	156	35	10,545	5,272,500	96,989
$M8$ [19]	500	8,246	432	2.1	17	15	72	18	4,1590	2,079.5	31,408.733
$CCC20d$ [37]	1,265	28,440	732	13.3	45	44	94	47	26	32,890	346.619
$prGm6$ [38]	1,200	171,685	335	41.8	714	335	1644	383	33,457	40,148,400	2,488,254.81

Table 1: Datasets used for the experimentation.

The main aspect that might impact on the performance of our approach is the setup configuration in terms of the number of partitions for the set of traces (n), and the number of partitions for a set of partial models (m). It is crucial to find out the best setup in terms of the timeout, n and m, albeit they will depend on the type of problem. These parameters are configured as explained below:

- **Grouping the subproblems from the same trace in a partition helps to reduce the number of subproblems to solve**. The approach is optimised for avoiding the execution of subproblems in twofold: *(a)* a subproblem is executed iff its estimation of the alignment is lower than the best alignment value obtained for the subproblem related to the same trace in the same partition; and *(b)* the subproblems are sorted by estimation in ascending order. In consequence, when any subproblem is not executed for the reason explained before, the execution of the remaining subproblems related to the same trace are skipped since the estimation will be always worse. For this reason, a good setup should concentrate all the problems related to the same trace in the fewest number of partitions as possible, but not reducing the parallelisation too much. By setting n equals to the number of cases in the log, we can assure that each partition will contain subproblems related to the same trace. Hence, this parameter is set to: 500 for $M2$, $M5$ and $M8$; $1,265$ for $CCC20d$, and $1,200$ for $prGm6$.

- **Balancing the number of partitions**. Some problems are too complex. It might lead to bottlenecks in the resolution of the whole alignment problem. A proper partitioning might help in avoiding them. For instance, if there is a low number of partitions, it is possible to take advantage of the factor explained above (i.e., avoiding mixing subproblems generated from different traces). The drawback is the fact that there could be one or several partitions with a bunch of complex subproblems increasing disproportionately the workload of some executors, at the same time there are idle nodes. The other extreme is having a high number of partitions, the number of subproblems to be solved will increase, since it would not take advantage of the factor previously explained. In this situation, we assume the rule of thumb as the higher the number of partition is, the more subproblems are solved. But, the lower the number of partition is, the fewer subproblems are solved, but it might cause bottlenecks. Since it is not possible to know what is the best number of partitions for each dataset, we will test the following values for m in the tests: 1, 2, 4, 5, 6, 8, 12, and 16.

In addition, we remark that for each configuration, 10 executions will be performed, so that all the results depicted are the average of those executions.

6.3. Results of the experiments

We have performed the experiments confronting three scenarios: (a) Scenario 1, the models and traces are decomposed and distributed, and the alignment computations are determined by the A* algorithm; (b) Scenario 2, the models and traces are decomposed and distributed, and the alignment computation is done by the COP-based approach, and; (c) Scenario 3, comparing the computation of the alignments performed locally (standalone) by using the A* algorithm and the best results from the Scenarios 1 and 2. In order to measure the performance, we employed the metric *Elapsed Real Time* (*ERT*)[9] for each scenario and dataset. Each *ERT* shown in this section comprises the average of ten executions.

6.3.1. Scenario 1: Distributed Computation of the Alignment with the A Algorithm*

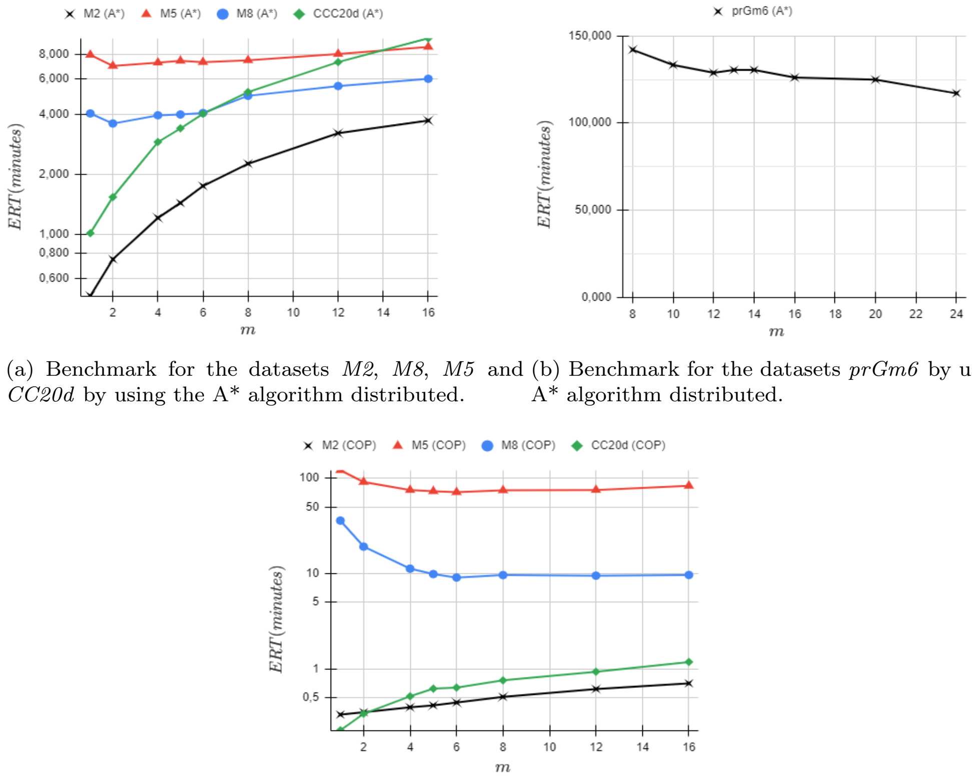
The chart in Figure 15a shows the evolution of the *ERT* increasing the number of partitions of the set of partial models (m) for the datasets $M2$, $M5$, $M8$, and $CCC20d$. We can see in the figure how the larger m is, the smaller partitions are distributed. For these tests, no timeout has been established since the A* algorithm can solve all the subproblems in a reasonable time. Therefore, all the alignments that have been obtained are optimal. Note that the results for $prGm6$ are not in the chart because of the complexity in that particular case, so it is analysed separately.

In details, the best *ERT* for $M5$ and $M8$ have been obtained with $m = 2$. From there, the *ERT* tends to worsen. If we analyse the slope of their trendlines, we find that the one for $M5$ is 0.08, while the one for $M8$ is 0.16. Compared to $M2$ and $CCC20d$, the best *ERT* is obtained with $m = 1$, and the slope of their trendline is 0.22 and 0.57, respectively.

As aforementioned, the results obtained for the dataset $prGm6$ are depicted in Figure 15b. Due to memory issues arising from the size of the partitions, it has been no possible to employ values for m from 1 to 6. For this reason, we have used the following values for m in this benchmark: 8, 10, 12, 13, 14, 16, 20, and 24. The best *ERT* value was obtained for $m = 24$, being the slope of the trendline -1.28.

From these results, we can conclude that the datasets that produce a

[9] The *Elapsed Real Time* (*ERT*) is the time from the start of the execution of a program to the end of it.



(a) Benchmark for the datasets *M2*, *M8*, *M5* and *CC20d* by using the A* algorithm distributed.

(b) Benchmark for the datasets *prGm6* by using the A* algorithm distributed.

(c) Benchmark for the datasets *M2*, *M8*, *M5* and *CC20d* by using the distributed COP-based approach.

Figure 15: Results in terms of ERT for the distributed algorithm (logarithmic scale).

larger number of subproblems are more benefited from a larger distribution. It is especially noticeable in the *prGm6* dataset, as it has a clear tendency to decrease the ERT when the number of partitions increases.

6.3.2. Scenario 2: Distributed Computation of the Alignments with the COP-based approach

Similarly done in the previous scenario, the chart in Figure 15c shows the evolution of the ERT increasing the number of partitions of the set of partial models (m). Once again, the larger m is, the smaller partitions are distributed. For these tests, a timeout of $500ms$ per subproblem has been established, since the COP is not capable to solve all the subproblems in a reasonable time if it is unbounded (note that the datasets which have a large number of subproblems might contain a high number of them producing bottlenecks). Therefore, certain alignments might not achieve the optimal. In the next section, the percentage of traces per dataset for which an optimal alignment value was found will be shown and analysed.

Remark that due to the excessive memory consumption by the COPs, it has been impossible to successfully complete any of the executions for the *prGm*6 dataset hence none results are shown.

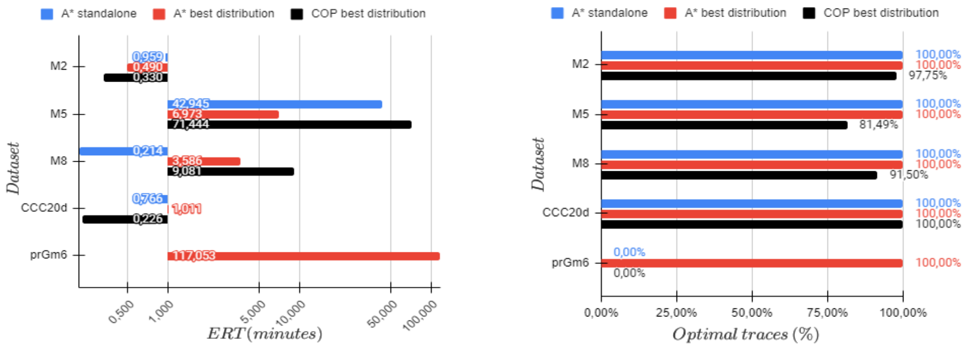
Analysing the results, for $M5$ and $M8$, the best ERT was obtained for $m = 6$. The slopes of the trendline for both of them are -3.13, and -1.44, respectively. On the other hand, for $M2$ and $CCC20d$ the best ERT value was obtained for $m = 1$. The slopes of both datasets are 0.02, and 0.06, respectively.

In this scenario the conclusions are similar to the previous one, we can conclude that the datasets which produce a larger number of subproblems are more benefited from a larger distribution (note that the slopes of the trendlines of $M5$ and $M8$ have a negative tendency). Additionally, we can also conclude that the more complex the algorithm for computing alignments is, the more benefits the execution gets by a larger distribution. It is justified by the fact that the slopes of the trendlines tend to be closer to zero or negative as the time spent by subproblem increases.

6.3.3. Scenario 3: Comparing the A Algorithm in standalone with the distributed approaches*

Figure 16a presents a comparison between the ERT of the A* algorithm in standalone, and the best results for A* and COPs in the Scenarios 1 and 2. Remark that there are no results for the COP-based approach in standalone since the COP implementation proposed in this paper was only conceived to be performed in distributed scenarios. Moreover, there are no results for the *prGm*6 and the A* algorithm in standalone nor the COP in distributed because (i) in the case of the A* algorithm in standalone, the PM4Py execution took more than 24 hours without any results; and (ii) regarding the distributed COP-based approach, some issues due to excessive memory consumption during the execution of these tests made it impossible to successfully finish any execution.

Figure 16b depicts the percentage of optimal alignments found over the set of traces for each dataset. Note that the only dataset for which the COP-based approach was able to find an optimal value for all the traces was $CC20d$.



(a) Comparison of the best configuration for each algorithm. The abscissa has a logarithmic scale.

(b) Percentage of optimal alignments found over the set of traces for each dataset.

Figure 16: Comparison between the standalone A* and the distributed A* and the COP-based approach in terms of ERT and percentage of optimal traces.

In summary, the distributed approach gets better results for $M2$, $M5$ and $prGm6$ with the A* algorithm, and for $CCC20d$ with the COP-based approach. On the other and, the only dataset for which the A* algorithm in standalone gets a better ERT is in the $M8$ dataset. In the light of the results, we can conclude that our approach on decomposing the alignment problem in subproblems and distributing them, in general, achieve better results in terms of ERT in comparison with the standalone approach. Finally, we remark that the complexity of the conformance checking algorithm has a heavy influence both in the ERT and in the number of optimal alignments (e.g., the COP-based approach).

7. Conclusion

In this paper, a Big Data framework is provided for the parallelisation and distribution of the conformance checking analysis disengaged of the algorithm applied. The creation of subproblems that can be solved distributed makes it possible to tackle problems whose complexity could not be approachable with local algorithms. For the decomposition, we have proposed a novel horizontal technique to build subproblems whose resolution is based on a map function, and combined by a *reduceByKey* strategy, with the improvement of an estimation metric that avoids the resolution of unpromising subproblems.

The proposed framework includes the capacity of customising the distribution of models and traces to find out the best configuration for distributing the alignment resolution. To demonstrate the applicability of our proposal, the framework has been tested by two alignment techniques, the classical A* approach and a novel approach based on the Constraint Optimisation paradigm. The analysis of these two options is derived from the interest to compare a classical solution, with others such as Constraint Optimisation Problems that enables to enclose the domain and limiting the amount of time available for finding an optimal alignment value. Five different datasets have been used for testing our framework to compare local (standalone) and distributed solutions, the distributed solution among them, and the effects of the configuration of the distribution in the performance. In summary, the framework provides a high degree of flexibility, facilitating the tuning of the parameters that determine the level of distribution of the subproblems, the application of different alignment algorithms, and the applicability of an estimation of the alignment, before it is computed, to avoid the analysis of unpromising subproblems.

After analysing experiments, it is possible to find examples where a local solver is more efficient, but for other examples, the distribution of the problem is more efficient than the local. Comparing the two algorithms in distributed scenarios, it is possible to find problems where both of them find a better solution. This is why we plan to carry out a deeper analysis of the features of the models and logs to characterise the problems for ascertaining which perform is better or worse before computing alignments.

Acknowledgment

This work has been partially funded by the Ministry of Science and Technology of Spain ECLIPSE (RTI2018-094283-B-C33) project, the European Regional Development Fund (ERDF/FEDER), MINECO (TIN2017-86727-C2-1-R), and by the University of Seville with *VI Plan Propio de Investigación y Transferencia* (VI PPIT-US).

Disclosure

All the authors are responsible for the concept of the paper, the results presented and the writing. All the authors have approved the final content of the manuscript. No potential conflict of interest was reported by the authors.

References

[1] M. Dumas, W. M. P. van der Aalst, A. H. M. ter Hofstede, Process-aware Information Systems: Bridging People and Software through Process Technology, Wiley, 2005. URL: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0471663069.html.

[2] H. Roehm, J. Oehlerking, M. Woehrle, M. Althoff, Model conformance for cyber-physical systems: A survey, TCPS 3 (2019) 30:1–30:26. URL: https://doi.org/10.1145/3306157. doi:10.1145/3306157.

[3] J. Carmona, B. F. van Dongen, A. Solti, M. Weidlich, Conformance Checking - Relating Processes and Models, Springer, 2018. URL: https://doi.org/10.1007/978-3-319-99414-7. doi:10.1007/978-3-319-99414-7.

[4] A. Adriansyah, Aligning observed and modeled behavior, Ph.D. thesis, Technische Universiteit Eindhoven, 2014.

[5] W. M. P. van der Aalst, Decomposing Petri nets for process mining: A generic approach, Distributed and Parallel Databases 31 (2013) 471–507. URL: http://dx.doi.org/10.1007/s10619-013-7127-5. doi:10.1007/s10619-013-7127-5.

[6] J. Munoz-Gama, J. Carmona, W. M. P. van der Aalst, Single-entry single-exit decomposed conformance checking, Inf. Syst. 46 (2014) 102–122. URL: http://dx.doi.org/10.1016/j.is.2014.04.003. doi:10.1016/j.is.2014.04.003.

[7] H. M. W. Verbeek, W. M. P. van der Aalst, Merging alignments for decomposed replay, in: F. Kordon, D. Moldt (Eds.), Application and Theory of Petri Nets and Concurrency: 37th International Conference, PETRI NETS 2016, Toruń, Poland, June 19-24, 2016. Proceedings, Springer International Publishing, Cham, 2016, pp. 219–239. URL: http://dx.doi.org/10.1007/978-3-319-39086-4_14. doi:10.1007/978-3-319-39086-4_14.

[8] W. L. J. Lee, H. M. W. Verbeek, J. Munoz-Gama, W. M. P. van der Aalst, M. Sepúlveda, Recomposing conformance: Closing the circle on decomposed alignment-based conformance checking in process mining,

Inf. Sci. 466 (2018) 55–91. URL: https://doi.org/10.1016/j.ins.2018.07.026. doi:10.1016/j.ins.2018.07.026.

[9] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, in: E. A. Brewer, P. Chen (Eds.), 6th Symposium on Operating System Design and Implementation (OSDI 2004), USENIX Association, San Francisco, 2004, pp. 137–150. URL: https://ai.google/research/pubs/pub62.

[10] S. Sakr, Z. Maamar, A. Awad, B. Benatallah, W. M. P. van der Aalst, Business process analytics and big data systems: A roadmap to bridge the gap, IEEE Access 6 (2018) 77308–77320. URL: https://doi.org/10.1109/ACCESS.2018.2881759. doi:10.1109/ACCESS.2018.2881759.

[11] M. T. Gómez-López, D. Borrego, J. Carmona, R. M. Gasca, Computing alignments with constraint programming: The acyclic case, in: Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2016 Satellite event of the conferences (ATAED) 2016, Torun, Poland, June 20-21, 2016, 2016, pp. 96–110. URL: http://ceur-ws.org/Vol-1592/paper07.pdf.

[12] B. F. van Dongen, J. Carmona, T. Chatain, F. Taymouri, Aligning modeled and observed behavior: A compromise between computation complexity and quality, in: Advanced Information Systems Engineering - 29th International Conference, Essen, Germany, June 12-16, 2017, 2017, pp. 94–109.

[13] B. F. van Dongen, Efficiently computing alignments - using the extended marking equation, in: Business Process Management - 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings, 2018, pp. 197–214.

[14] M. de Leoni, A. Marrella, Aligning real process executions and prescriptive process models through automated planning, Expert Syst. Appl. 82 (2017) 162–183.

[15] D. Reißner, R. Conforti, M. Dumas, M. L. Rosa, A. Armas-Cervantes, Scalable conformance checking of business processes, in: OTM CoopIS, , Rhodes, Greece, 2017, pp. 607–627.

[16] S. J. J. Leemans, D. Fahland, W. M. P. van der Aalst, Scalable process discovery and conformance checking, Softw. Syst. Model. 17 (2018) 599–631. URL: https://doi.org/10.1007/s10270-016-0545-x. doi:10.1007/s10270-016-0545-x.

[17] D. Reißner, A. Armas-Cervantes, R. Conforti, M. Dumas, D. Fahland, M. La Rosa, Scalable alignment of process models and event logs: An approach based on automata and s-components, Information Systems 94 (2020) 101561. URL: http://www.sciencedirect.com/science/article/pii/S0306437920300545. doi:https://doi.org/10.1016/j.is.2020.101561.

[18] F. Taymouri, J. Carmona, A recursive paradigm for aligning observed behavior of large structured process models, in: 14th International Conference of Business Process Management (BPM), Rio de Janeiro, Brazil, September 18 - 22, 2016, pp. 197–214.

[19] F. Taymouri, J. Carmona, Model and event log reductions to boost the computation of alignments, in: P. Ceravolo, C. Guetl, S. Rinderle-Ma (Eds.), Data-Driven Process Discovery and Analysis, Springer International Publishing, 2018, pp. 1–21.

[20] F. Taymouri, J. Carmona, Computing alignments of well-formed process models using local search, ACM Trans. Softw. Eng. Methodol. 29 (2020) 15:1–15:41. URL: https://doi.org/10.1145/3394056. doi:10.1145/3394056.

[21] A. Burattin, S. J. van Zelst, A. Armas-Cervantes, B. F. van Dongen, J. Carmona, Online conformance checking using behavioural patterns, in: Business Process Management - 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings, 2018, pp. 250–267.

[22] F. Taymouri, J. Carmona, Structural computation of alignments of business processes over partial orders, in: 19th International Conference on Application of Concurrency to System Design, ACSD 2019, Aachen, Germany, June 23-28, 2019, 2019, pp. 73–81.

[23] L. Padró, J. Carmona, Approximate computation of alignments of business processes through relaxation labelling, in: Business Process Man-

agement - 17th International Conference, BPM 2019, Vienna, Austria, September 1-6, 2019, Proceedings, 2019, pp. 250–267.

[24] J. Evermann, Scalable process discovery using map-reduce, IEEE Transactions on Services Computing 9 (2016) 469–481. doi:10.1109/TSC.2014.2367525.

[25] F. Chesani, A. Ciampolini, D. Loreti, P. Mello, Map reduce autoscaling over the cloud with process mining monitoring, in: M. Helfert, D. Ferguson, V. Méndez Muñoz, J. Cardoso (Eds.), Cloud Computing and Services Science, Springer International Publishing, Cham, 2017, pp. 109–130.

[26] J. Engelfriet, Branching processes of petri nets, Acta Inf. 28 (1991) 575–591. URL: https://doi.org/10.1007/BF01463946. doi:10.1007/BF01463946.

[27] R. Bergenthum, S. Mauser, R. Lorenz, G. Juhás, Unfolding semantics of petri nets based on token flows, Fundam. Inform. 94 (2009) 331–360. URL: https://doi.org/10.3233/FI-2009-134. doi:10.3233/FI-2009-134.

[28] D. Fahland, W. M. P. van der Aalst, Simplifying discovered process models in a controlled manner, Inf. Syst. 38 (2013) 585–605. URL: https://doi.org/10.1016/j.is.2012.07.004. doi:10.1016/j.is.2012.07.004.

[29] R. Dechter, Constraint Processing (The Morgan Kaufmann Series in Artificial Intelligence), Morgan Kaufmann, 2003.

[30] K. Apt, Principles of Constraint Programming, Cambridge University Press, New York, NY, USA, 2003.

[31] M. T. Gómez-López, R. Ceballos, R. M. Gasca, C. D. Valle, Developing a labelled object-relational constraint database architecture for the projection operator, Data Knowl. Eng. 68 (2009) 146–172. URL: https://doi.org/10.1016/j.datak.2008.09.002. doi:10.1016/j.datak.2008.09.002.

[32] A. J. Varela-Vaca, L. Parody, R. M. Gasca, M. T. Gómez-López, Automatic verification and diagnosis of security risk assessments in business

process models, IEEE Access 7 (2019) 26448–26465. URL: https://doi.org/10.1109/ACCESS.2019.2901408. doi:10.1109/ACCESS.2019.2901408.

[33] M. T. Gómez-López, R. M. Gasca, J. M. Pérez-Álvarez, Compliance validation and diagnosis of business data constraints in business processes at runtime, Inf. Syst. 48 (2015) 26–43. URL: http://dx.doi.org/10.1016/j.is.2014.07.007. doi:10.1016/j.is.2014.07.007.

[34] M. T. Gómez-López, L. Parody, R. M. Gasca, S. Rinderle-Ma, Prognosing the compliance of declarative business processes using event trace robustness, in: On the Move to Meaningful Internet Systems: OTM 2014 Conferences - Confederated International Conferences: CoopIS, and ODBASE 2014, Amantea, Italy, October 27-31, 2014, Proceedings, 2014, pp. 327–344.

[35] D. Borrego, R. Eshuis, M. T. Gómez-López, R. M. Gasca, Diagnosing correctness of semantic workflow models, Data Knowl. Eng. 87 (2013) 167–184.

[36] R. Bergenthum, S. Mauser, Synthesis of petri nets from infinite partial languages with viptool, in: 15th German Workshop on Algorithms and Tools for Petri Nets, Algorithmen und Werkzeuge für Petrinetze, AWPN 2008, Rostock, Germany, September 26-27, 2008. Proceedings, 2008, pp. 81–86. URL: http://ceur-ws.org/Vol-380/paper13.pdf.

[37] J. Buijs, Environmental permit application process ('wabo'), CoSeLoG project, 2014. URL: https://data.4tu.nl/collections/Environmental_permit_application_process_WABO_CoSeLoG_project/5065529. doi:10.4121/UUID:26ABA40D-8B2D-435B-B5AF-6D4BFBD7A270.

[38] J. Munoz-Gama, J. Carmona, W. M. P. van der Aalst, Conformance checking in the large: Partitioning and topology, in: F. Daniel, J. Wang, B. Weber (Eds.), Business Process Management, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 130–145.

[39] J. Cardoso, Business process control-flow complexity: Metric, evaluation, and validation, International Journal of Web Services Research 5 (2008) 49–76. URL: http://www.igi-global.com/bookstore/titledetails.aspx?titleid=1079.