Information Systems 99 (2021) 101731

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

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[1]
[b] Department of Computer Science, Universitat Politècnica de Catalunya, Spain[2]
[c] Fakultät für Mathematik und Informatik, FernUniversität in Hagen, Germany[3]

ARTICLE INFO

Article history:
Received 1 March 2020
Received in revised form 17 November 2020
Accepted 24 January 2021
Available online 18 February 2021
Recommended by Gottfried Vossen

Keywords:
Conformance checking
Decompositional techniques
Big Data
MapReduce

ABSTRACT

Conformance checking unleashes the full power of process mining: techniques from this discipline enable the analysis of the quality of a process model through the discovery of event data, the identification of potential deviations, and the projection of real traces onto process models. In this way, the insights gained from the available event data can be transferred to a richer conceptual level, amenable for human interpretation. Unfortunately, most of the aforementioned functionalities are grounded in an extremely difficult fundamental problem: given an observed trace and a process model, find the model trace that most closely resembles to the trace observed. This paper presents an architecture that supports the creation and distribution of alignment subproblems based on an innovative horizontal acyclic model decomposition, disengaged from the conformance checking algorithm applied for their solution. This is supported by a Big Data infrastructure that facilitates the customised distribution of a gross amount of data. Experiments are provided that testify to the enormous potential of the architecture proposed, thereby opening the door to further research in several directions.

1. Introduction

By means of conceptual models, organisations tend to define complex business processes that must be followed to achieve their objectives [1]. Sometimes the corresponding processes are distributed across various systems, in which the majority of cases include human tasks, thereby inadvertently enabling the occurrence of unexpected deviations with respect to the (normative) process model. This is aggravated by the appearance of increasingly 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 that the deviations 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 more closely resembles σ. Computing alignments is an extremely difficult problem, with a complexity exponential in the size of the model or the trace [4]. Intuitively, computing an alignment requires a search through the state space of the model which, in certain cases implies an extensive 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 widely differing alternatives. Among these, we highlight decompositional techniques, which break the alignment problem into segments, whose solutions can be composed to reconstruct the final alignment [5–8]. All these decompositional approaches feature a common strategy which involves decomposing 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

* Corresponding author.
E-mail addresses: avalencia@us.es (Á. Valencia-Parra), ajvarela@us.es (Á.J. Varela-Vaca), maytegomez@us.es (M.T. Gómez-López), jcarmona@cs.upc.edu (J. Carmona), robin.bergenthum@fernuni-hagen.de (R. Bergenthum).

1 http://www.idea.us.es.
2 https://www.cs.upc.edu.
3 https://www.fernuni-hagen.de.

https://doi.org/10.1016/j.is.2021.101731
0306-4379/

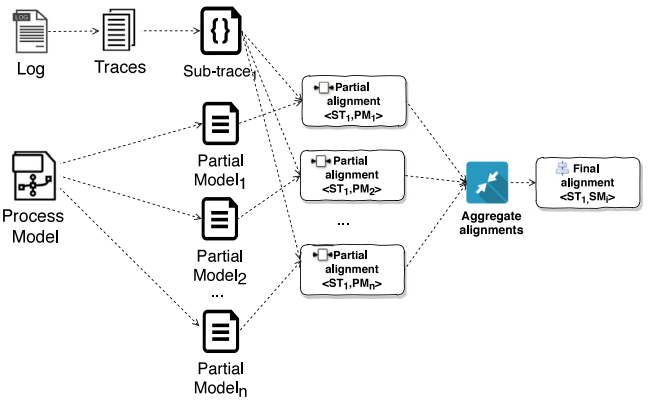


Fig. 1. Functional description of the Big Data architecture to compute alignments.

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.

In this paper, we step back from the decompositional approach, and focus on working at a more abstract, architectural level. As earlier mentioned, normally the complexity of computing the alignment problem has been addressed by means of the horizontal and vertical decomposition techniques [9]. We propose a Big Data infrastructure focused on a specific horizontal decomposition, which involves the unfolding of process models, and employing the MapReduce paradigm [10] for the decomposition and aggregation. At first sight, our functional strategy (depicted in Fig. 1) will not bring any new ideas 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 are composed to aggregate a final alignment. This distribution of the problem may facilitate the simplification of the problem, by splitting the conformance analysis into partial models (with smaller search spaces) and subtraces across several nodes (Map), and combining the partial alignments[4] 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 [11] framework proposed in this paper is innovative for the following reasons:

- A new 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 originates from a partial order representation of the initial process model. The horizontal decomposition of the model limits the search space dividing the model into its possible execution models, and analysing the alignment with every trace. However, in the vertical decomposition, both traces and the model are fragmented, thereby reducing the search space for each part of the model and each trace fragment.
- The framework enables the construction, distribution, and parallelisation of the computing alignment between different nodes in a Big Data environment to be tuned in accordance with the features of the problem and the available requirements. Moreover, the application of heuristics is proposed to optimise the resolution of the subproblems.
- It enables us to choose and customise the conformance checking algorithm, by making it possible to compute the alignment with different techniques. In this case, we have used the A* algorithm as a classic solution, and the *Constraint Programming Paradigm* [12] as a new solution, 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 represents a leap forward in the resolution of conformance checking problems of a more complex nature, and reduces 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 carried out to evaluate our proposal, and finally, Section 7 presents the conclusions.

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 various papers [13,14]. These approaches represent the state-of-the-art technique for computing alignments, and can be adapted (at the expense of a significant increase in the memory footprint) to provide all optimal alignments. Alternatives to A* have appeared in recent years: in the approach presented in [15], the alignment problem is mapped as an instance of *automated planning*. Automata-based techniques have also appeared [16,17]. The techniques in [16] (and recently extended in [18]) rely on state-space exploration and determination of the automata corresponding to both the event log and the process model, whilst the technique in [17] is based on computing several subsets of activities and projecting the alignment instances accordingly. In spite of the significant progress made, the aforementioned techniques still have problems in dealing with large inputs.

The work in [19] presents the notion of *approximate* alignment to alleviate computational demands by proposing a recursive paradigm on the basis of the structural theory of Petri nets. In spite of its resource efficiency, the solution is not guaranteed to be executable. Alternatively, the technique in [20] 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 a local search have also recently been proposed [21]. Although the approximate techniques can provide solutions where exact/optimal techniques fail, they only provide certain guarantees for very restricted classes of models.

Against this background, the process mining community has focused on dividing 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. Our focus now turns to decompositional

[4] *Partial alignment* means the computation of the alignment for a partial model and a subtrace.

approaches towards the computation of alignments, which are more closely related to the research of this paper.

Decompositional techniques have been presented [5–7] which, instead of computing optimal alignments, focus on the *crucial problem* of whether a given trace fits a process model. These techniques vertically decompose the process model into pieces that satisfy certain conditions. Therefore, only *valid* decompositions [5], which satisfy restrictive conditions on the labels and connections forming a decomposition, guarantee the derivation of a real alignment. The notion of *recomposition* has since been proposed on top of decompositional techniques, in order to obtain optimal alignments whenever possible by modifying the decomposition (typically by merging sets) when the required conditions are not met [8]. In contrast to the aforementioned vertical decomposition techniques, our methodology does not require this last modification of partial solutions, and therefore can provide a fast alternative to these methods at the expense of losing the guarantee of optimality.

There has also been related work on the use of partial order representations of process models for the computation of alignments. In [22], unfoldings are employed 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 [23,24] to significantly accelerate the computation of alignments. We believe that these approaches, especially the latter two, can easily be integrated into our framework.

The work of [18] 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. This work is compatible with the framework suggested in this paper since the model restrictions assumed in [18] are satisfied by the partial models arising from our horizontal decomposition.

Finally, the MapReduce distributed programming model has previously been considered for process mining. For instance, Evermann applies it to process discovery [25], whilst [26] applies it for monitoring declarative business processes.

3. Foundations

We denote $\perp$ as the empty set. Let A be a set of elements, and we denote A^* as the set of all sequences over elements of A. Let $a, b \in (A \cup \{\perp\})^*$ be two sequences. We denote $a^{\not\perp}$ as 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 describe process models and partial models by means of labelled Petri nets.

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, and $\ell : T \to \Sigma \cup \{\perp\}$ is the labelling function.

Fig. 2 depicts a labelled Petri net. Places are represented by circles and transitions by rectangles. Every transition has a unique name and a label on top. Places and transitions are connected in accordance with the flow-relation.

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

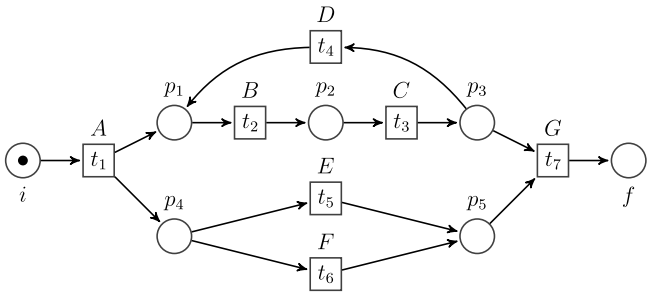


Fig. 2. A labelled Petri net.

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, $\bullet t : P \to \{0, 1\}$ as $\bullet t(p) := F(p, t)$, and $t\bullet : P \to \{0, 1\}$ as $t \bullet (p) := F(t, p)$. A transition $t \in T$ is enabled at marking m if $m \geq \bullet t$ holds. If transition t is enabled, then transition t can be fired. 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, Fig. 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 as A, is enabled. Firing t_1 leads to the marking where i does not carry a token, and both places in the post-set of t_1 each carry a token.

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 $\langle m_1, \ldots, m_{n+1}\rangle$ such 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 can 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 Fig. 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 classic 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 allowed to move synchronously only if the label of the transition matches the event.

We consider the Petri net depicted in Fig. 2 with initial state i and final state f. By aligning the Petri net to the trace $\langle A, A, B, C,$

$D, B, C, G\rangle$ for instance, we obtain numerous 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 attain a cost for every alignment. Every move of an alignment adds to its cost, where asynchronous moves add greater cost than synchronous moves [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 a so-called optimal alignment between a trace of an event log and a process model. An optimal alignment is an alignment with a lowest cost. In the previous example, if we define the cost of an asynchronous move as 1 and the cost of a synchronous move as 0, then the depicted alignments have a cost of 2.

4. Computing conformance checking with big data

4.1. Overview of the approach

The fundamental problem in conformance checking involves aligning a trace concerning a process model [3]. This problem, known as *the alignment problem*, is a search (which can be highly 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 on a Big Data architecture that aims to facilitate the computation of alignments on a grand scale. 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 [10]. As will be observed see in Section 5, to instantiate the architecture for a real situation, we build upon our previous work [12] and the case of a partial order decomposition of a process model (see 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 through horizontal decomposition. Other decompositional approaches available in the literature [5–7] might be employed, but lead to changes in the way in which the *Generate partitions of Problems*, *Map*, and *Reduce* activities are implemented. Furthermore, it should be borne in mind that other decompositional approaches must be capable of forming partitions so that these can be distributed among the nodes of the cluster.

In order to determine each of these parameters that describe how the subproblems are created, distributed, solved, and combined, Fig. 3 summarises the workflow followed in our approach. Since our proposal is not hooked to a specific 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 are the subtraces[5] and partial models. These are obtained through the unfolding process, which applies a horizontal decomposition technique. Once these aspects are defined, a subtrace and partial model pre-processing are needed (see *Pre-process traces and partial models*) to determine certain features used in the heuristics for the subsequent problem distribution. The system is then set up (see *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 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 following three activities: *Generate partitions of problems*, *Map — Distribution and compute partitions*, and *Reduce — Combine results*. These are given in detail in Sections 4.2, 4.3, and 4.4, respectively. The framework follows the idea of the MapReduce paradigm as depicted in Fig. 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 is the alignment found for the set of traces involved in this subproblem. All the partial solutions represented by maps are then combined.

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, thereby reducing the time.

4.2. Generate partitions of problems

As indicated in Section 1, we aim to alleviate 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 behaviour of the original model. Furthermore, a partial model needs to be acyclic and conflict-free. Finally, it should be borne in mind that the approach assumes that partial models are generated through horizontal decomposition.

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

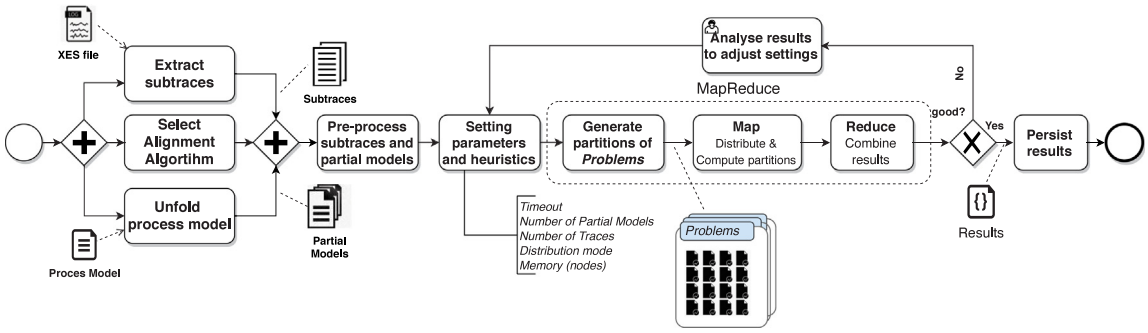


Fig. 3. Workflow of the approach.

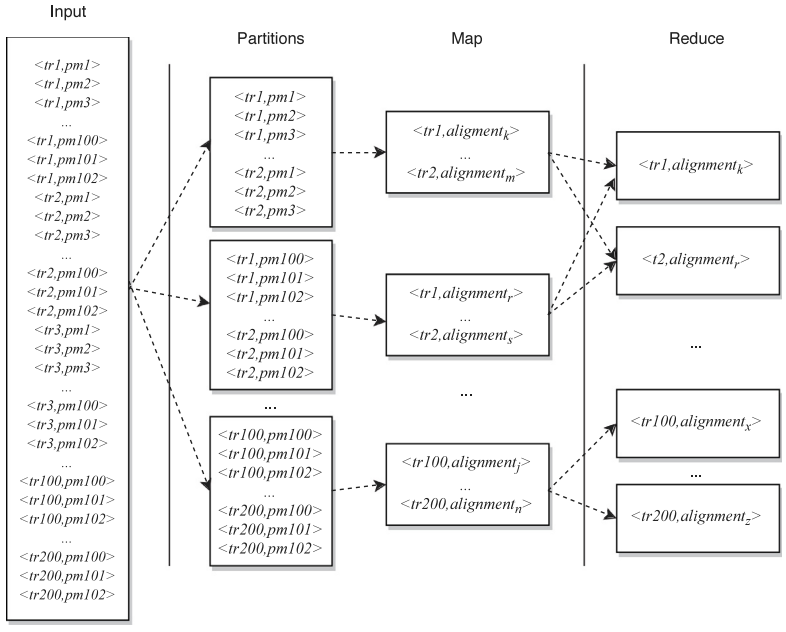


Fig. 4. MapReduce for alignment analysis.

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 $(m\,[t_1\rangle\, m' \wedge m\,[t_2\rangle) \Longrightarrow m'\,[t_2\rangle$ 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.

In this respect, a partial alignment is the computation of the alignment of a partial model.

Fig. 5 depicts a partial model of Fig. 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 Fig. 2. Fig. 6 depicts another partial model. In this example, transitions t_2 and t_5 carry the label B, while transitions t_3 and t_6 carry the label C. Thus, the loop of Fig. 2 is unfolded.

One straightforward approach to splitting a Petri net into a cover is to calculate its branching process [27]. It is well-known that the set of so-called process nets of a branching process is a cover. It should be borne in mind that the branching process itself can be infinite, but given 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 approaches to calculating finite representations of an infinite branching process in a reasonable time [28].

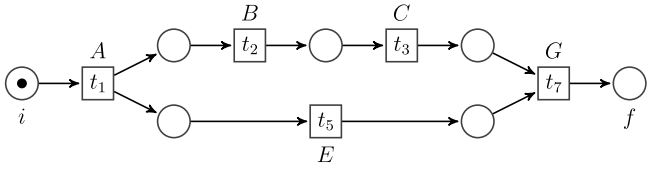


Fig. 5. A partial model of Fig. 2.

Fig. 7 depicts a prefix of the branching process of the model of Fig. 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, the two places (see p_5 and p_{10}) behind transitions labelled by E and F. In Fig. 2, this pair is only one place (see 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 (see p_1 in Fig. 2). Thus, conflicts and cycles are unfolded. Every connected subnet of a branching process, whereby all places have at most one subsequent transition, is called an occurrence net. For example, the set of transitions $\{t_1, t_2, t_3, t_5, t_7\}$, with all connected places, form the partial model of Fig. 5. Transitions $\{t_1, t_2, t_3, t_4, t_6, t_8, t_{10}, t_{13}\}$ are the partial model of Fig. 6.

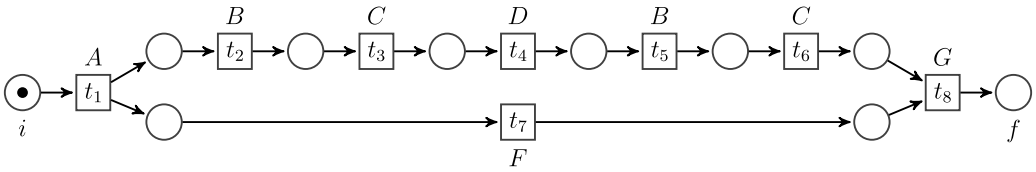


Fig. 6. Another partial model of Fig. 2.

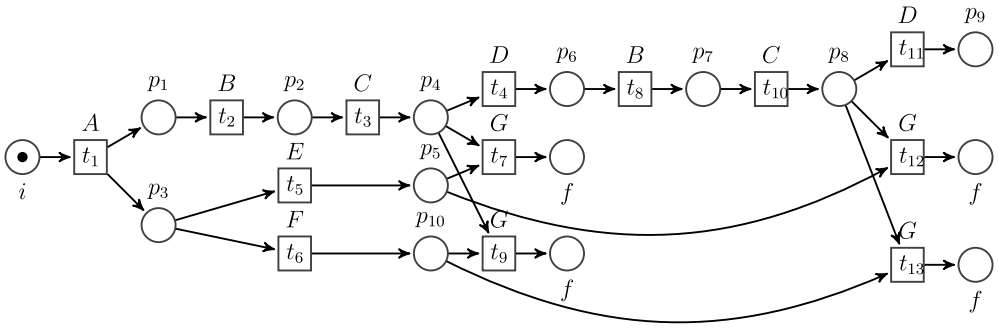


Fig. 7. A prefix of the branching process of Fig. 2.

The use of occurrence nets of branching processes constitutes only one of many possibilities for the horizontal decomposition of a model. Log-based unfolding [29] and token flow-based unfolding [28] can generate similar decompositions. In the more general setting of the paper, the set of partial models is required to be 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 that incurs 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 Fig. 6 can be replayed by the original model depicted in Fig. 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:

A	A	B	C	D	B	$\perp$	C	G
$\perp$	t_1	t_2	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

If a set of partial models covers a Petri net, then, for every alignment of the original model, there exists a partial model covering each alignment. This holds true since the set of partial models can replay every trace of the original model. An optimal alignment can be therefore be calculated for the original model by calculating an optimal alignment for the set of partial models.

The division of the problem into smaller partitions forms the base of the application of the MapReduce paradigm. It is therefore necessary to tackle the problem of partitioning 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 Fig. 3, the process model and log are decomposed into subtraces and partial models that can be analysed independently, thereby obtaining the alignment in a more efficient way.

Definition 8 (*Decomposition, Alignment Subproblem*)**.** Let AP be an alignment problem aligning a set of traces $Tr = \{tr_1, tr_2, \ldots, tr_n\}$ to a model M. Let $Pm = \{pm_1, pm_2, \ldots, pm_m\}$ 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.

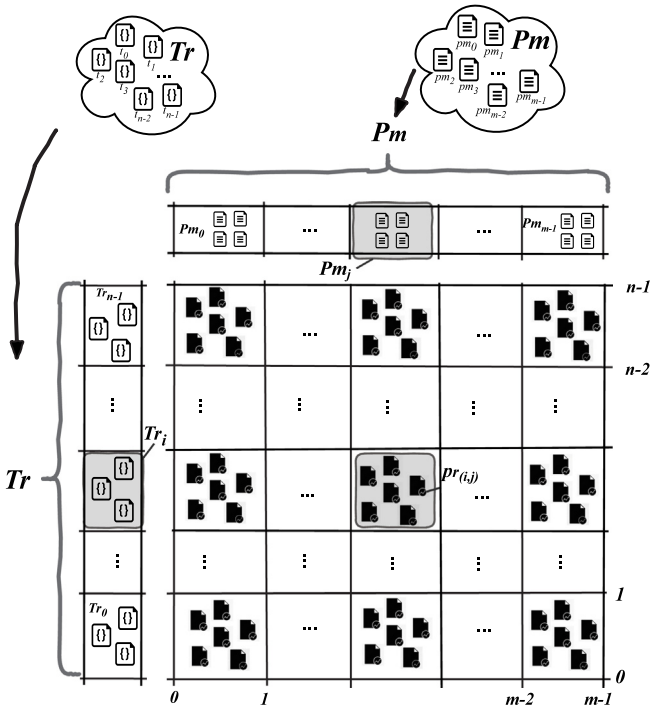


Fig. 8. Partitions and sets of subproblems.

In an ideal scenario with unlimited resources, each alignment subproblem could be solved independently and in parallel. In this case, the total run-time needed to solve AP would be the time spent on the most complex subproblem plus the time spent combining 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 simply generate a node for every problem. Thus, subproblems need to share nodes. To control the distribution of subproblems to nodes, the set of all possible subproblems is partitioned into groups of subproblems that share the same features. Features are made up of the involved trace and partial alignments

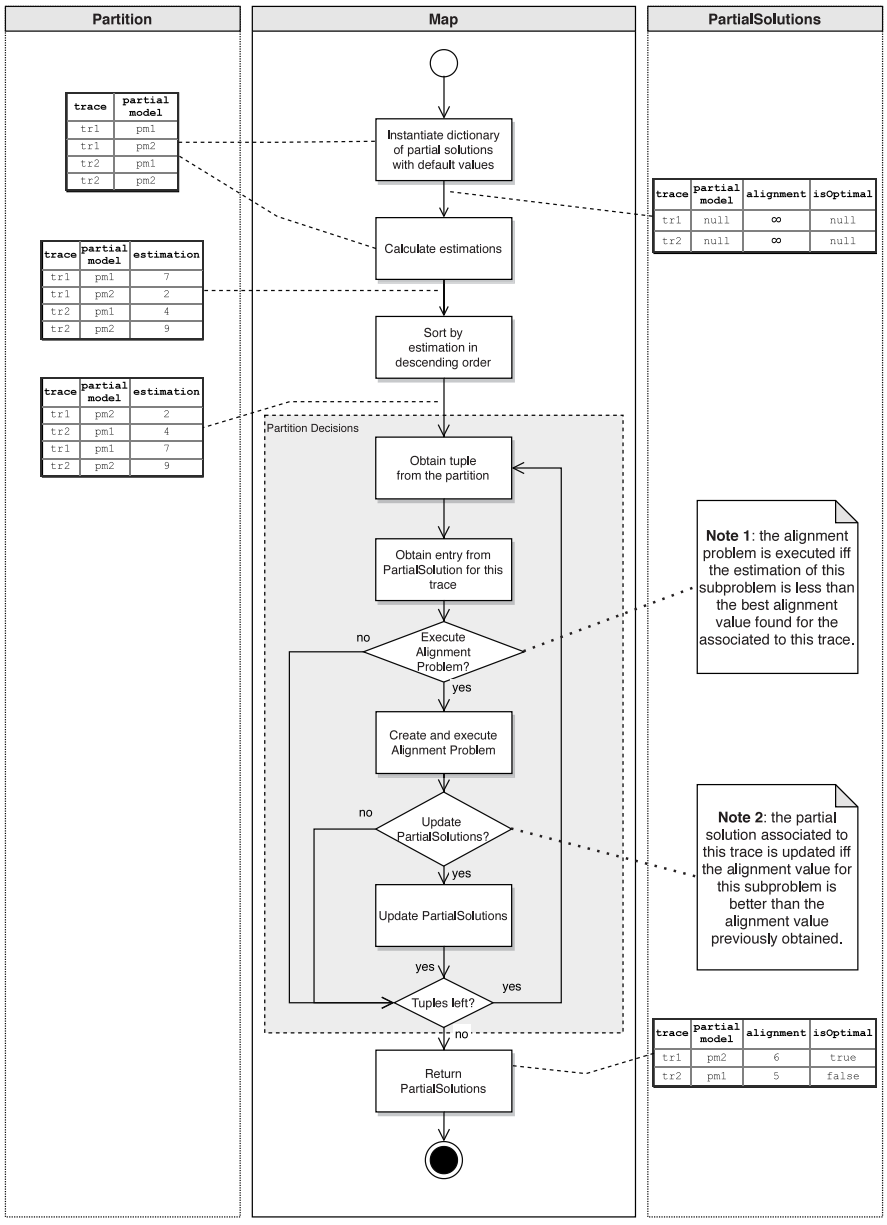


Fig. 9. Activity diagram to describe the Map algorithm.

calculated in other subproblems. How to properly group and distribute subproblems to calculate solutions efficiently is analysed below.

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. Let Tr_i be a set of the partition of Tr and let Pm_j be a set of the partition of Pm. We call $pr_{(i,j)} := (Tr_i \times Pm_j)$ a partition of the alignment subproblems.

Partitions $pr_{(i,j)}$ define sets of alignment subproblems. When each alignment subproblem $(Tr_i \times Pm_j)$ is solved, a partial alignment is obtained. Figure schematically 8 depicts partitions of Tr and Pm and the resulting sets of alignment subproblems.

In the next subsection, we will discuss the distribution of every partition of alignment problems following the MapReduce strategy [10], which is a programming model to support 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, thereby obtaining partial solutions for their subsequent combination. The algorithm used in the map function is represented in Fig. 9. It receives a partition of subproblems and creates a dictionary of *partial solutions* with default values. For each subproblem, it then makes a lower-bound estimation for the possible alignment that can be taken before it is solved. This estimation is employed to sort the subproblems to solve in the same partition (sequentially solved). The estimation is obtained by comparing model and trace: (1) checking the size of the trace w.r.t. the maximum number of 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, then the alignment (see Definitions 5 and 6) must be at least 10); (2) the events that 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., if the event A is repeated three times in the

trace but only twice in the submodel, then the alignment cost must be at least 1). These values are calculated and aggregated to generate an estimation as the lowest value that the alignment can take. If an alignment subproblem discovers better alignment than the estimation for the same trace, then it is illogical to evaluate the remaining subproblems with an inferior estimation. The partition of subproblems is then sorted by estimation in ascending order. Sorting is crucial for the optimisation of the execution time since it prevents the alignment process from executing subproblems that would fail to provide an improvement of the best alignment found up to that moment (see Note 1 in Fig. 9). If a new alignment is obtained, then the partial solution associated with that trace is updated if the new alignment value is better than the previous value (see Note 2 in Fig. 9). Note that the partial solutions have an attribute called *isOptimal*. This will be true when it is possible to guarantee that the solution associated with this trace is the optimal solution (note that the notion of non-optimality is introduced due to the existence of a timeout, which prevents the search space from being searched in its entirety). If the isOptimal attribute of any of the subproblems that were executed is marked as *false* due to the timeout being reached, then we cannot guarantee that the solution to any other subproblem associated to that trace is the optimal solution, because any other subproblem with a better previously executed estimation value could have returned a better alignment value if the timeout had not been reached.

In order to illustrate the algorithm, Fig. 10 presents the iteration of the *partition* presented in Fig. 9. At this point, it should be borne in mind that the partitions are already sorted by estimation. There are four elements to process, and hence, there are four iterations. In *Iteration 1*, the subproblem $\langle tr1, pm2\rangle$ is processed. The alignment process is then executed because the estimation yields a value of 2, which could improve the partial solution found until the moment (∞). Once the alignment value (6) has been computed, 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 also updated after obtaining an alignment of 5. However, the timeout was fired, and therefore the alignment cannot be guaranteed to be optimal ($\langle tr2, pm1, 5, false\rangle$). However *Iteration 3* does not execute the alignment process nor update 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*, the same situation arises, and hence the partial solution formerly found for $tr2$ is not updated.

4.4. Reduce - Combining alignment problem result

The Reduce phase is responsible for combining the partial solutions that are generated during the Map phase. 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 if it is impossible to ensure that the alignment obtained is the optimal solution. In this phase, all the partial solutions corresponding to the same trace are combined, and that with the best alignment value is selected as the best solution for such a trace.

Fig. 11 depicts the Reduce process, and includes some partial solutions to show the execution. The proposal is based on the function known as *reduceByKey*, which groups data in terms of the key of the data provided by the map function, and then applies 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, thereby obtaining another partial solution. Following the example of Fig. 11, the tuples of *Partial Solutions* 1 and 2 are grouped according to their 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 that was employed to create the groups and is 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. This means that if it is *false* for a tuple, then the other tuples related to the same trace will also be marked as *false*, 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 various types of alignment techniques, subtraces, and partial models. Several algorithms that have tackled the conformance checking problem in the context of business processes (see Section 2 for a full description). In this section, two such algorithms are included: the A* algorithm as an example of a classic algorithm developed by other authors [4]; and a new implemented solution based on the Constraint Programming Paradigm. These two algorithms have the same objective (i.e., to discover the alignment). However, we have included the Constraint Programming Paradigm since it enables certain special features to be incorporated, such as restricting the domain of the possible value where the alignment can be found, and determining a maximum time of resolution per subproblem that returns the best solution found up until the timeout.

5.1. Alignment based on the A* algorithm

One of the most relevant solutions to computing alignments found in the literature is the A* algorithm [4]. It has been successfully employed as a feasible approximation to discover the optimal alignment between the process model and traces [3]. Essentially, the model and trace are combined into a synchronous product. Fig. 12 illustrates the synchronous product, and presents the partial model, obtained from a cover (see Definition 7) given in Fig. 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 (see Definition 7, [3]) from the synchronous product, and then to deduce the shortest path from the initial marking to the final marking. However, the construction of the full reachability graph is not always possible due to the state space explosion problem. To overcome this problem, the reachability graph is built in pieces. The A* algorithm is efficiently used (see Chapter 7.3, Procedure 2 [3]) to compute the shortest path. The core of the A* algorithm relies on a heuristic function, $f(m) = g(m) + h(m)$, which guides the search, where $g(m)$ is the cost of the path from the initial marking to m. For instance, for any reachable state m, 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 highly complex and time-consuming (e.g., in models with a very high levels of parallelism [18]).

Our approach integrates the implementation of the A* algorithm provided by the Python library *PM4Py*.[6]

[6] PM4Py: https://pm4py.fit.fraunhofer.de/.

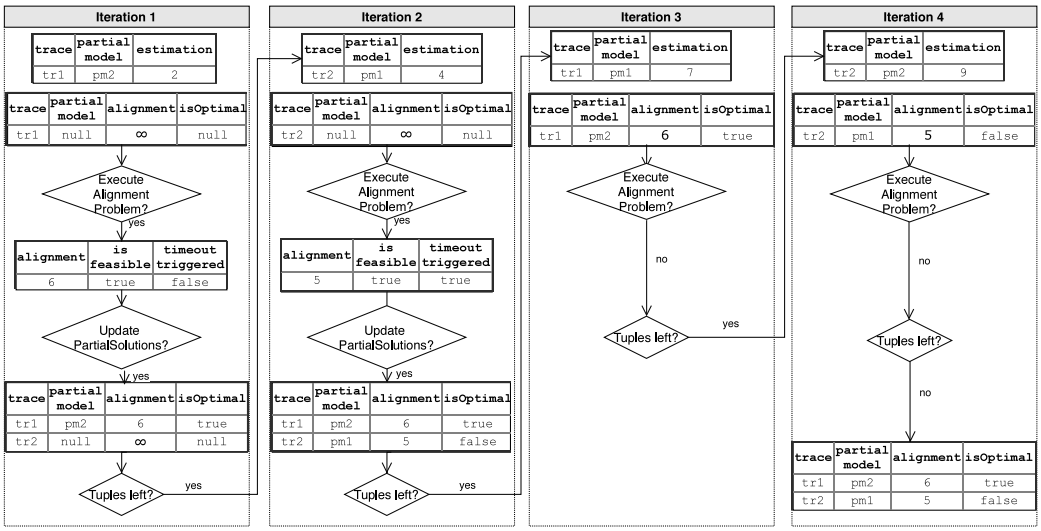


Fig. 10. Execution trace of the Map function.

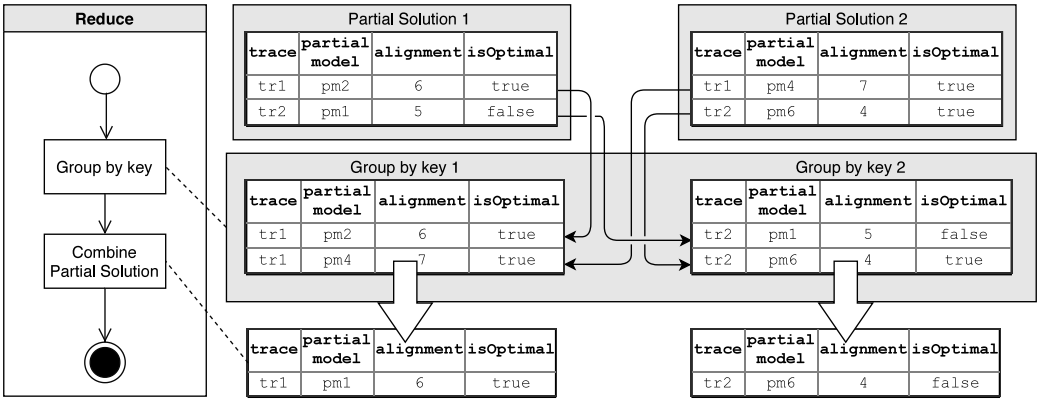


Fig. 11. Execution trace of the Reduce function.

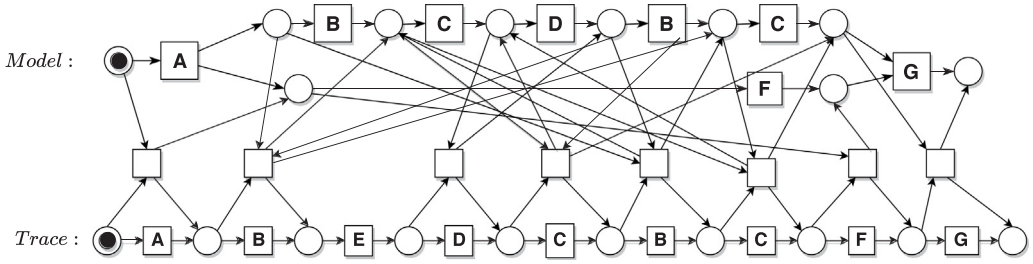


Fig. 12. Example of synchronous product of model and trace.

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, the incorporation of this new solution in our framework is considered to be relevant as an evolution of a previous proposal [12]. Moreover, since the alignment computation can be modelled as a variable and restrictions whose domain can be bounded, and thanks to the decomposition of the model into submodels, we consider it relevant to analyse how the former resolution of subproblems can be used to tighten the possible domain for the analysis in further resolutions. The partial model, obtained from a cover (see Definition 7) given in Fig. 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.

In our approach, the computation of the alignment problem of a log trace and a partial model is encoded as a Constraint Problem for the reduction in running time and resources of the proposal presented in [12]. Thus, the information extracted from the partial model and the trace, such as the name of transitions, events, the execution, order and their possible positions, are translated into variables, constraints, and an objective function of a Constraint Optimisation Problem (COP). The horizontal decomposition can ensure that the resulting partial models contain

or-splits. In fact, this helps reduce the complexity of the COP regarding the number of restrictions and the number of variables and the domain of the variables.

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 [30, 31]. Constraint Programming brings together a set of algorithms to determine the solutions of a problem described.

Definition 10 (*The Constraint Satisfaction Problem*)**.** (CSP) is 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 enable one tuple of instantiation of one, multiple, or all these values to be sought in accordance with the requirement of the problem.

An example of its applicability in the alignment context is given by its representation of the order relation existing in the models and the traces, as found in Fig. 6. By using a set of variables to represent the order of the events, and by 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.length-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 regarding the level of misalignment is provided by the resolution of the CSP. In this case, a Constraint Optimisation Problem (COP) is able to ascertain 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 can be the solution of the COP.

Constraint Optimisation Problems (COPs) have already been employed to detect the alignment between the expected and the observed behaviour in model-based diagnosis [32,33], specifically when the behaviour is described by means of business process models [34–36]. These studies used the concept of *reified constraints* as a mechanism to assign a Boolean value to the constraints included in the model [34], whereby a constraint that cannot be satisfied during the CSP resolution can be relaxed. Since the idea is to determine the minimal distance between the model and the log, these relaxed constraints must be the minimum number, defined as the objective of the function to be optimised.

Following the previous example, the COP below is created where the *Ref* variables relate 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($Ref_A + Ref_C + \ldots + Ref_{AF}$)

Although 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 determine 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 that of the partial model. However if this is not possible, then a number of the constraints are ignored from the COP firing reified constraints. The structure of the COP is as shown in Fig. 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 the partial model. In the running example, all the variables receive 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 indicates otherwise. These variables are the key to obtain the log and model moves in the alignment calculation as will be seen in the objective function. These variables are also used to enable/disable the firing of 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 built to represent conditions of the position of the events

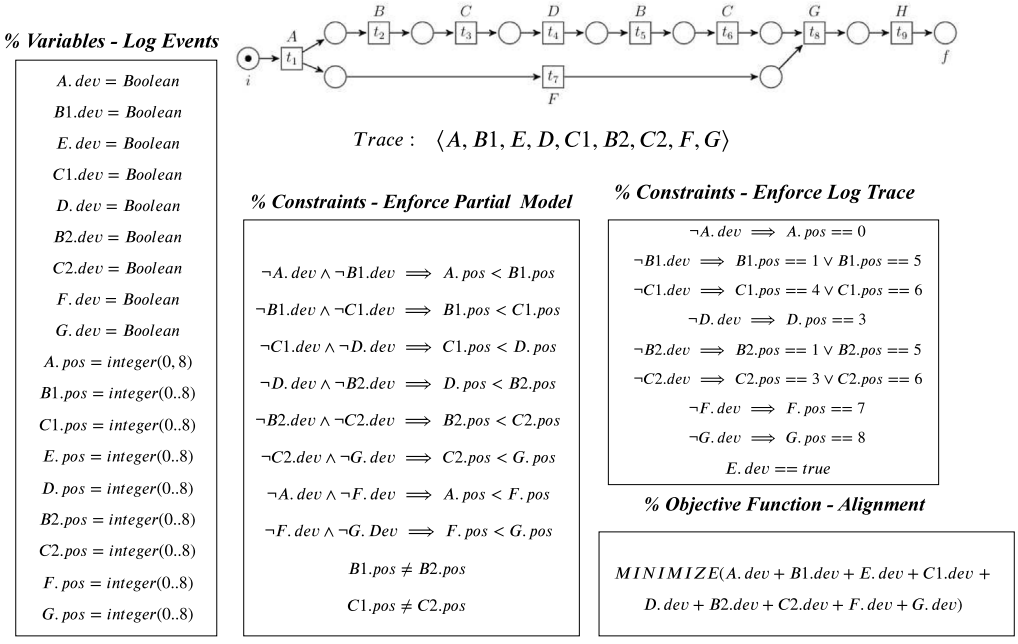


Fig. 13. COP for the example of Fig. 6.

with respect to the log trace. For instance, event *A* occurs first:

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

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

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

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. 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. It should be taken into account that, in the partial models used in our proposal, the XORs are eliminated, and every transition of the model participates in any correct event log. Therefore, the next constraint is an example of this type of reified constraint:

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

The reified constraint means that if events *A* and *B1* are aligned with the model, then the value assigned to *pos* of event *A* has to be lower than the values of *pos* of event *B1*. In the case of repeated events (e.g., *B1* and *B2*), extra constraints have to be included in order to prevent their occurrence at the same position:

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

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 transition are not added, although a misalignment will be included (a model move). See below for a description of how this is computed.

- *Optimisation function*: The objective function strives to find a solution that minimises the number of deviations. The Boolean variables are considered as Integer, that is, *false* is the 0 value and *true* is the 1 value. As shown in Fig. 13, the objective function is the minimisation of the sum of all deviation variables of our problem. Thus, finding 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 the case where any *dev* variable is fixed to *true*, the alignment will be, at least, the number of *dev true* values.

This COP enables the possible deviations between the partial models and the events to be detected:

- **Log moves:** The log moves are determined by consulting the *false* values fixed in the deviation (*dev*). If the *dev* variable of an event reach a *false* value, then this event does not produce a log move. When an event does not occur in the partial model, this situation is a log move, and therefore 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. Hence, this situation is penalised as a model move by adding one to the alignment cost function.

Subsequent to the COP resolution, the log and model moves are known, and therefore the alignment cost function can be 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 $E.dev = true$, and another value due to the $C1.dev = true$ ($D.dev = true$ in the other equivalent solution) since it is impossible to locate it according to the log trace, and another value because H does not occur 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 limit, then we have, at least, the best option found up to that moment, although it may not be the global optimal since the search space has not been completely solved.

6. Experiments and evaluation

In order to evaluate our proposal, we have performed various tests to compare the distributed approach proposed in this paper with the classic standalone A* algorithm for computing alignments. Regarding the distributed approach, two algorithms are employed for encoding alignments: the A* algorithm and the COP-based approach presented in this paper (see Section 5). This section is structured as follows:

- Design of the architecture and the technology stack to support our framework (see Section 6.1).
- Selection of a set of representative datasets (see Section 6.2) that includes examples which work better with the distributed approach proposed in this paper, and examples which work better with the classic standalone approach. The configuration of the parameters is also studied.
- Analysis of the approach proposed in this paper, by solving the alignment subproblems derived from the previously selected datasets. The distributed approach is compared to the classic standalone approach (see Section 6.3). For the evaluation and comparison, the performance is considered in terms of the Elapsed Real Time (*ERT*)[7] related to the computation of the alignments.

6.1. Architecture

We propose the use of a three-layer architecture, as shown in Fig. 14. Additionally, we include information regarding the technological stack that has been employed 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 these can be accessed by the nodes that comprise the system. In our particular implementation, it is based on Hadoop HDFS,[8] 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.[9]
- *Computing layer*. It is intended to perform the computing operations related to the generation and distribution of partitions and computing alignments. Our implementation is based on Apache Spark,[10] which is a distributed computing framework that 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 mentioned earlier, we have integrated the PM4Py[11] platform for the computation of alignments using the A* algorithm. On the other hand, the COPs have been implemented with ILOG CPLEX[12] 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*. This is responsible for monitoring and assigning resources among the nodes of the cluster. There is one node entirely dedicated to this task.
- *Driver program*. This node is responsible for distributing the tasks among the Executor nodes. Regarding our implementation, it will schedule the partitioning process by assigning the partitions and their related tasks to the executor nodes. In our case, the driver program is configured to use 8 GB 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 8 GB of RAM and 4 CPUs by default. Each of the four nodes of the cluster hosts one executor.

Both the source code with the implementation of the framework and the datasets used for the experimentation are available 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 order (LPO) format [37]. For a better understanding, the LPO format is a simplification but remains 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. The events of the LPO modelling transition occurrences can fire in the net in accordance with the concurrency and dependency relations given by the LPO.

The event logs and Petri nets employed to illustrate how our proposal works with problems of different sizes are extracted from [20,38,39], whereas the partial models are obtained from 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) [40]); 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 (see Num. of Problems of Table 1) is derived from the application of the Cartesian product and their required theoretical storage space. The objective of this table involves obtaining an estimate of the complexity involved in the solution of all the problems of each dataset, especially *M5* and *prGm6*, in which more than five and forty million subproblems must be solved, respectively. Approximately, the approach must manage a total of 96 GB and 2.5 TB of data volume, as appear for *M5* and *prGm6* in the table. The distribution of the alignment computation

[7] The *Elapsed Real Time* (*ERT*) is the time from the start of the execution of a program to its completion.

[8] HDFS: https://hadoop.apache.org/.

[9] MongoDB: https://www.mongodb.com/.

[10] Apache Spark: https://spark.apache.org/.

[11] PM4Py: https://pm4py.fit.fraunhofer.de/.

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

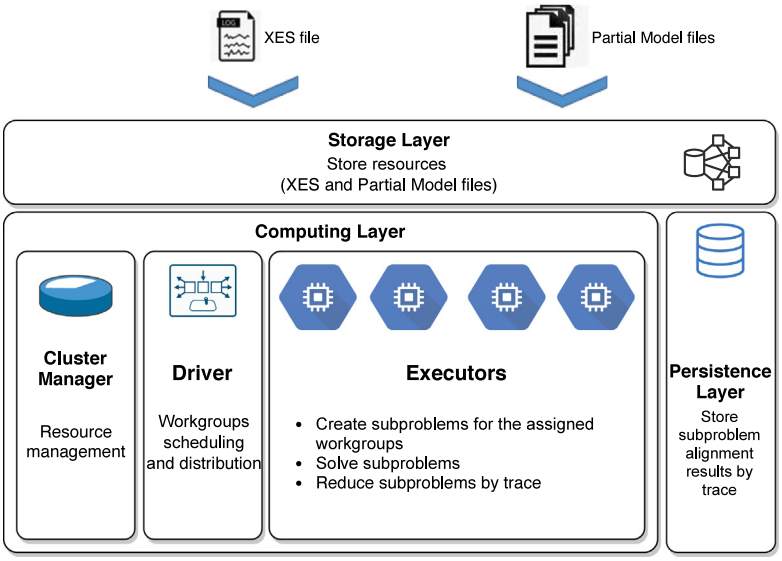


Fig. 14. Cluster architecture.

can imply the transfer of a vast amount of data between the executors, thereby producing a negative impact on the performance. For this reason, the access to traces and partial models has been centralised and they are each accessed by a unique identifier.

The main aspects that might influence the performance of our approach include 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 ascertain out the best setup in terms of the timeout, n, and m, albeit dependent 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 in order to prevent the execution of subproblems for two reasons: *(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 aforementioned reason, the execution of the remaining subproblems related to the same trace are skipped since the estimation will always be 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 without over-reducing the parallelisation. By setting n equal 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 $M8$; 1265 for $CCC20d$, and 1200 for $prGm6$.
- **Balancing the number of partitions of alignment subproblems**. Some alignment subproblems are too complex and may lead to the formation of bottlenecks in the resolution of the whole alignment problem. A proper partitioning might help in preventing such bottlenecks. 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 of a low number of partitions is that there could be one or several partitions with a group of complex subproblems which would disproportionately increase the workload of certain executors, while simultaneously leaving other executors idle. The other extreme involves having a high number of partitions. In such a case, the number of subproblems to be solved would increase, since it would not take advantage of the factor previously explained. In this situation, we assume the rule of thumb to be that the higher the number of partitions is, the more subproblems are solved. However, the lower the number of partitions is, the fewer subproblems are solved, although this situation may trigger the creation of 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, for each configuration, 10 executions will be performed, and all the results depicted are the average of those executions.

6.3. Results of the experiments

This section is organised as follows. First, Section 6.3.1 highlights the results obtained from the distributed approach proposed in Section 4. Two algorithms are compared with this approach: the A* algorithm, and the COP-based approach. Secondly, Section 6.3.2 compares those results with the classic standalone approach with the A* algorithm.

The metric we employ in order to measure and compare each approach is the *Elapsed Real Time* (*ERT*)[13] for each scenario and dataset. Each *ERT* shown in this section comprises the average of ten executions.

The evaluation is performed by means of an analytic study. This study includes: (i) graphics which depict how the *ERT* evolves as the number of partitions increases; and (ii) an analysis of the trendline of each dataset. By analysing the slope of each trendline, we can quantitatively observe how the *ERT* scales as the number of partitions increases. For each dataset, if the slope is positive, it means that the creation of more partitions (i.e., a better distribution) is not beneficial for that dataset, and the greater the slope is, the worse it scales. If the slope is negative, it means that the approach does scale well for that dataset (and the less steep the slope is, the better it scales). The slope also enables a comparison between several datasets with the same configuration.

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

Table 1
Datasets used for the experimentation.

Dataset	Event log				Petri net				Partial models		
	Cases	Events	Variants	Size (MB)	Places	Transitions	Arcs	CFC	Num. of models	Num. of problems	Size (MB)
*M*2 [20]	500	8,809	500	2.20	34	34	160	36	102	51,000	509.4
*M*5 [20]	500	17,028	500	4.2	35	33	156	35	10,545	5,272,500	96,989
*M*8 [20]	500	8,246	432	2.1	17	15	72	18	4,1590	2,079.5	31,408.733
*CCC*20*d* [38]	1265	28,440	732	13.3	45	44	94	47	26	32,890	346.619
*prGm*6 [39]	1200	171,685	335	41.8	714	335	1644	383	33,457	40,148,400	2,488,254.81

6.3.1. Results obtained from the distributed approach

The chart in Fig. 15a shows the evolution of the *ERT* increasing the number of partitions of the set of partial models (m) for the datasets *M*2, *M*5, *M*8, and *CCC*20*d*. 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 *prGm*6 are not in the chart because of the complexity in that particular case, and it is therefore analysed separately.

In detail, the best *ERTs* for *M*5 and *M*8 have been obtained with $m = 2$. From there, the *ERTs* tend to worsen. If the slopes of their trendlines are analysed, that for *M*5 is 0.08, while that for *M*8 is 0.16. For *M*2 and *CCC*20*d*, the best *ERT* is obtained with $m = 1$, and the slopes of their trendlines are 0.22 and 0.57, respectively. This means that *M*5 benefits more from the distribution than the other three datasets.

As aforementioned, the results obtained for the dataset *prGm*6 are depicted in Fig. 15b. Due to memory issues arising from the size of the partitions, it has not been 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 is obtained for $m = 24$, with the slope of the trendline at -1.28. It means that this dataset benefits from a better distribution, since the *ERT* tends to decrease as the number of partitions increases.

From these results, we can conclude that the datasets that produce a larger number of subproblems more benefit are from a larger distribution. It is especially noticeable in the *prGm*6 dataset, as it has a clear tendency to decrease the *ERT* when the number of partitions increases. It should be noted that *M*5, which also produces a large number of alignment subproblems, has a slope of 0.08, which shows better scalability than *M*8, *M*2, and *CCC*20*d*.

Next, we present the results obtained from the distributed approach with the COP-based approach. Fig. 15c shows the evolution of the *ERT* increasing the number of partitions of the set of partial models (m). Once again, the larger m becomes, the smaller are the distributed partitions. For these tests, a timeout of 500 ms per subproblem has been established, since the COP is not capable of solving 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 thereof 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 is found will be shown and analysed. 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 no results are shown.

Analysing the results for *M*5 and *M*8, the best *ERT* is obtained for $m = 6$. The slopes of the trendline for the two datasets are -3.13 and -1.44, respectively. This shows that the distribution of the alignment subproblems improve the resolution in terms of *ERT*. This is observed when *ERT* is compared to the previous results with the A* algorithm, where the slopes for *M*5 and *M*8 are 0.08 and 0.16, respectively. However, in absolute terms, the *ERT* tends to be higher with the COP solver.

On the other hand, for *M*2 and *CCC*20*d*, the best *ERT* value is obtained for $m = 1$. The slopes of both datasets are 0.02, and 0.06, respectively, which also shows a better scaling than in the previous case with the A* algorithm. For these datasets, the *ERTs* are similar to those obtained with the A* algorithm.

6.3.2. Comparing the A algorithm in standalone with the distributed approaches*

Fig. 16a presents a comparison between the *ERT* of the A* algorithm in standalone, and the best results obtained from the distributed approach.[14,15]

Fig. 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 is able to find an optimal value for all the traces is *CCC*20*d*.

In summary, the distributed approach attains better results for *M*2, *M*5, and *prGm*6 with the A* algorithm, and for *CCC*20*d* with the COP-based approach. The datasets which produce a larger number of subproblems, in general, gain greater benefits from a larger distribution (note that the trendlines of *M*5 and *M*8 have a negative slope when solved with the COP-based approach, and a slightly positive tendency when solved with A*). Note also that datasets with extremely complex models might be solved in a reasonable time with the distributed approach, as can be seen with *prGm*6. However, in certain cases, the application of decomposition and the distribution of the subproblems fails to produce the best results. For example, the best results of *M*8 were obtained from the standalone A* algorithm. This is due to the characteristics of the model. An in-depth study into the factors which make a decomposition worthwhile in terms of *ERT* could be performed in the future.

By comparing the distributed approach with A* and the COP-based approach, we can conclude that the more complex the algorithm for computing alignments is, the more benefits the execution attains from a larger distribution. This is justified by the fact that the slopes of the trendlines tend to be closer to zero or negative as the time spent by the subproblems increases. It is especially noticeable in the case of the COP-based approach, which takes more time per subproblem than does the A* algorithm.

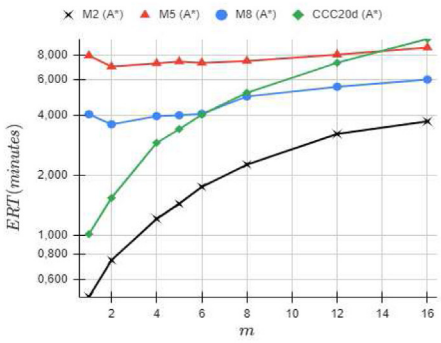
In the light of the results, we can conclude that our approach to decomposing the alignment problem into subproblems and their subsequent distribution, in general, achieves better results in terms of *ERT* in comparison with the standalone approach. Finally, we remark that the complexity of the conformance checking algorithm exerts a heavy influence both on the *ERT* and on the number of optimal alignments (e.g., the COP-based approach).

7. Conclusion

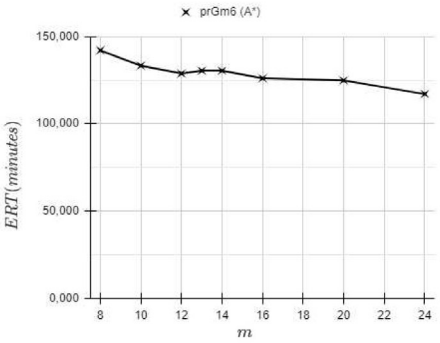
In this paper, a Big Data framework is provided for the parallelisation and distribution of the conformance checking analysis disengaged from the algorithm applied. The creation of subproblems that can be solved distributed makes it possible to tackle

[14] 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.

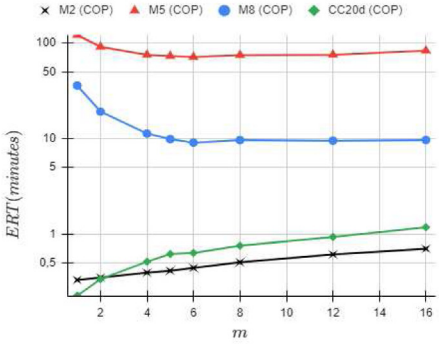
[15] There are no results for the *prGm*6 and the A* algorithm in standalone because the PM4Py execution took more than 24 h without yielding any result.



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

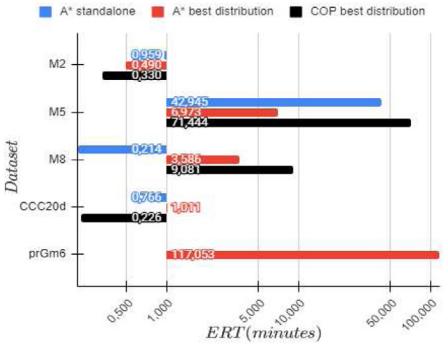


(b) Benchmark for the datasets *prGm6* by using the A* algorithm distributed.

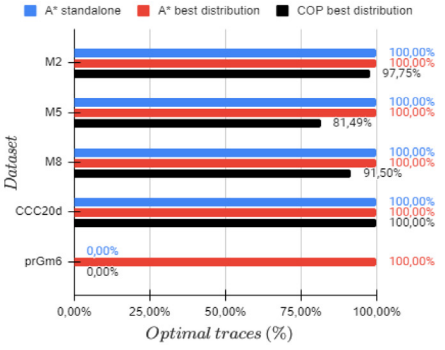


(c) Benchmark for the datasets *M2*, *M8*, *M5* and *CC20d* by using the distributed COP-based approach.

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



(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.

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

problems whose complexity could not be approached with local algorithms. For the decomposition, we have proposed an innovative 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 prevents the resolution of unpromising subproblems.

The proposed framework includes the capacity of customising the distribution of models and traces to determine the best configuration for the distribution of the alignment resolution. To demonstrate the applicability of our proposal, the framework has been tested by two alignment techniques: the classic A* approach and a new approach based on the Constraint Optimisation paradigm. The analysis of these two options is derived from the interest in comparing a classic solution with others, such as Constraint Optimisation Problems, which enables the domain to be enclosed and the amount of time available for finding an optimal alignment value to be limited. 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 on the performance. In summary, the framework provides a high degree of flexibility, since it facilitates 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, in order to prevent the analysis of unpromising subproblems.

From this analysis of the 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. By comparing the two algorithms in distributed scenarios, it is possible to pinpoint the problem areas where each algorithm can find a better solution or take a shorter time. This is why we plan to carry out a more in-depth analysis of the features of the models and logs in order to characterise the problems in terms of ascertaining which performs better before computing the alignments.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

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, Spain (TIN2017-86727-C2-1-R), and by the University of Seville, Spain 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, http://dx.doi.org/10.1145/3306157.

[3] J. Carmona, B.F. van Dongen, A. Solti, M. Weidlich, Conformance Checking - Relating Processes and Models, Springer, 2018, http://dx.doi.org/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, Distrib. Parallel Databases 31 (2013) 471–507, http://dx.doi.org/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, http://dx.doi.org/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, http://dx.doi.org/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, Inform. Sci. 466 (2018) 55–91, http://dx.doi.org/10.1016/j.ins.2018.07.026.

[9] W.M.P. van der Aalst, Distributed process discovery and conformance checking, in: J. de Lara, A. Zisman (Eds.), Fundamental Approaches to Software Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 1–25.

[10] 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.

[11] 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, http://dx.doi.org/10.1109/ACCESS.2018.2881759.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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, http://dx.doi.org/10.1007/s10270-016-0545-x.

[18] 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, Inf. Syst. 94 (2020) 101561, http://dx.doi.org/10.1016/j.is.2020.101561, URL: http://www.sciencedirect.com/science/article/pii/S0306437920300545.

[19] 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.

[20] 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.

[21] 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, http://dx.doi.org/10.1145/3394056.

[22] 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.

[23] 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.

[24] L. Padró, J. Carmona, Approximate computation of alignments of business processes through relaxation labelling, in: Business Process Management - 17th International Conference, BPM 2019, Vienna, Austria, September 1-6, 2019, Proceedings, 2019, pp. 250–267.

[25] J. Evermann, Scalable process discovery using map-reduce, IEEE Trans. Serv. Comput. 9 (2016) 469–481, http://dx.doi.org/10.1109/TSC.2014.2367525.

[26] 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.

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

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

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

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

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

[32] 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, http://dx.doi.org/10.1016/j.datak.2008.09.002.

[33] 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, http://dx.doi.org/10.1109/ACCESS.2019.2901408.

[34] 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, http://dx.doi.org/10.1016/j.is.2014.07.007.

[35] 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.

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

[37] 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.

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

[39] 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.

[40] J. Cardoso, Business process control-flow complexity: Metric, evaluation, and validation, Int. J. Web Serv. Res. 5 (2008) 49–76, URL: http://www.igi-global.com/bookstore/titledetails.aspx?titleid=1079.