Noname manuscript No.
(will be inserted by the editor)

Optimized SAT Encoding of Conformance Checking Artefacts

Mathilde Boltenhagen · Thomas Chatain · Josep Carmona

the date of receipt and acceptance should be inserted later

Abstract Conformance checking is a growing discipline that aims at assisting organizations in monitoring their processes. On its core, conformance checking relies on the computation of particular artefacts which enable reasoning on the relation between observed and modeled behavior. It is widely acknowledge that the computation of these artifacts is the lion's share of conformance checking techniques. This paper shows how important conformance artefacts like *alignments*, *anti-alignments* or *multi-alignments*, defined over the Levenshtein edit distance, can be efficiently computed by encoding the problem as an optimized SAT instance. From a general perspective, the work advocates for a unified family of techniques that can compute conformance artefacts in the same way. The implementation of the techniques presented in this paper show capabilities for dealing with both synthetic and real-life instances, which may open the door for a fresh way of applying conformance checking in the near future.

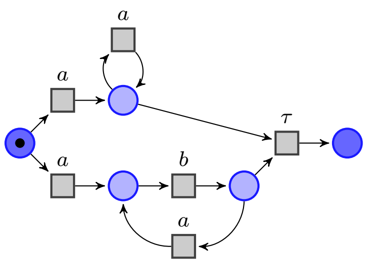
1 Introduction

Organizations are struggling to adapt to the demands arising from the digital transformation. Thus, the use of all kinds of digital data related to an organization's daily routines is a mandatory practice: one cannot think of a successful organization that disregards the data generated by its clients, or the data generated by its processes. In terms of business processes, modern organizations are extremely complex and evolving entities, representing a real challenge for current technology. The fields of Business Process Management, and Process Mining, propose strategies to tackle this challenge.

M. Boltenhagen and T. Chatain
ENS Paris-Saclay

J. Carmona
Universitat Politècnica de Catalunya

This is a post-peer-review, pre-copyedit version of an article published in *Computing*.
The final authenticated version is available online at: http://dx.doi.org/10.1007/s00607-020-00831-8



(a) Model with two types of runs : $\langle a, a, \ldots, a\rangle$ and $\langle a, b, a, b, \ldots, a, b\rangle$

Log trace : $\langle b, a, b, a, b\rangle$	Hamming distance	Edit distance
run (size = 6) $\langle a, a, a, a, a, a\rangle$	3	6
run (size = 6) $\langle a, b, a, b, a, b\rangle$	6	2

(b) Comparision of Hamming and edit distances

Fig. 1: How Hamming distance penalizes alignment and anti-alignment's quality: Trace $\langle b, a, b, a, b, a\rangle$ is closer to $\langle a, a, a, a, a, a\rangle$ than $\langle a, b, a, b, a, b\rangle$ according to Hamming distance. However there is only a shift of letter a and b between $\langle b, a, b, a, b, a\rangle$ and $\langle a, b, a, b, a, b\rangle$. The model run $\langle a, b, a, b, a, b\rangle$ is a better anti-alignment for trace $\langle b, a, b, a, b, a\rangle$, revealed by the edit distance.

In particular, conformance checking is a growing subdiscipline of process mining that focuses on providing algorithmic means to support process models in organizations, with a clear emphasis on the relation of processes and process models. On its core, conformance checking relies on the computation of artefacts that link observed and modeled behavior, which are used with different purposes: spotting deviations, evaluating quality metrics of a process model, extending a process model with evidence-based information, among others [10].

Conformance checking is expected to be the fastest growing segment within Process Mining in the years to come[1]. Still, the field is facing several challenges. Among them, we highlight two important ones: techniques for a sound replay of event data on top of process models, and the advent of faithful metrics for evaluating process models with respect to observed behavior.

The former challenge is strongly related to the notion of *alignment* artefact: given a trace and a process model, find a run in the process model that is as close as possible (in edit distance terms) to the observed trace. The seminal work in [1] describes an algorithm for computing alignments based on A^*. Alternatives to this algorithm have appeared recently (see Section 2 for a detailed description of the current approaches). In this paper, we provide an alternative to the aforementioned techniques, that is based on encoding the computation of an alignment as a SAT instance. We also show how to encode in SAT *multi-alignments* [13], which are artefacts that generalize the notion of alignments, so that not one but several traces are considered when computing the closest process model run. The latter challenge is mainly concerned with the proposal of sound and meaningful metrics for *precision* and *generalization*, nowadays acknowledged as the quality dimensions with less convincing estimations (e.g., [26]). *Anti-alignments*, presented in [12], are an effective conformance artefact to foresee those model runs that deviate most (again, in terms of edit distance) with

[1] `https://www.marketsandmarkets.com/Market-Reports/process-analytics-market-254139591.html`

respect to a trace or a complete log. In [32] it is shown how anti-alignments can be used to provide a more consistent estimation to both precision and generalization, using the Hamming distance. The use of Hamming distance for anti-alignment shows important drawbacks, highlighted in Fig. 1.

Overall, the current paper adopts SAT as a reasoning engine to compute conformance checking artefacts like alignments, multi-alignments and anti-alignments, over the edit distance. Furthermore, we show for the first time how the baseline encoding for Levenshtein edit distance, reported in recent works [7,11], can be significantly optimized to process larger instances. Formal proofs are provided of these optimizations, and a novel adaptation of the SAT objective function to improve the quality of the obtained artefacts is proposed in this paper. Finally, all the techniques proposed in this paper are implemented in a new tool, which is evaluated over synthetic and real-life benchmarks from the literature.

The paper is organized as follows: next section provides related techniques for the tasks considered in this paper. Then in Section 3 we provide the background for understanding the paper. Section 4 shows how to encode the computation of a run at a given edit distance of a trace. Then Section 5 shows how to adapt this encoding to particular conformance checking artefacts, and Section 6 presents optimizations which refine these encodings. Section 7 reports experiments and Section 8 discusses theoretical and experimental complexity of our problems, whilst Section 9 concludes the paper.

2 Related work.

The seminal work in [1] introduced the notion of alignment. The approach uses a depth-first search, alongside with the A^* method over the state space corresponding to the synchronous product between the model and the trace. A recent publication shows under which circumstances the A^* can be improved by extending the heuristic pruning the search space [30].

Alternatives to the A^* have appeared recently: in the approach presented in [15], the alignment problem is mapped as an *automated planning* instance. Unlike the A^*, the aforementioned work is only able to produce one optimal alignment (not all optimal), but it is expected to consume considerably less memory. Automata-based techniques have also appeared [25,20]. In particular, the technique in [25] can compute all optimal alignments. The technique in [25] relies on state space exploration and determinization of automata, whilst the technique in [20] is based on computing several subsets of activities and projecting the alignment instances accordingly. A different perspective that uses *event structures* focusing in *behavioral alignments* is presented in [16]. Finally, the work in [6] uses binary decision diagrams to compute alignments. These alternative techniques are competitive for certain inputs, but at the same time are more sensitive to crucial aspects as problem size or degree of concurrency.

To tackle the computational challenge of computing an alignment, [28] proposed an approach grounded on the resolution of *Integer Linear Programming* (ILP), alongside with partitioning the input observed trace σ. This technique can provide *approximate alignments*, a novel class of alignments where deviations can be explained between sets of transitions, instead of singleton pairs as in [1]. Although this approach is efficient both in time and memory, it is not complete, i.e., it cannot guarantee the computation of a real solution in general. A similar approach which can always guarantee a solution and heavily uses the resolution of ILP and marking equation in combination with a bounded backtracking is presented in [32].

Decompositional and projection approaches have also appeared in the last years to fight the complexity of computing alignments. The technique in [27], presents a framework to reduce a process model and the event log accordingly, with the goal to alleviate the computation of the alignments. The computed alignment, which is called *macro-alignment*, will be expanded based on the gathered information during the reduction. A different line of work is by opting for a decomposition perspective [29, 23]. The proposed approach decomposes a given model to smaller parts and projects the observed trace to each corresponding part, and then computes the alignment for each part independently. This technique is very efficient, but the result is decisional (a yes/no answer on the fitness of the trace) and cannot provide a global alignment. Recently, [33, 19] proposed a set of conditions for providing a global alignment from the decomposed alignments.

Recent studies focus on SAT implementation of Data Mining algorithms, in order to satisfy all the constrains and get optima [22, 14]. By introducing a SAT implementation of alignments in this paper, we hope to push a new family of algorithmic methods for conformance checking in the line of [12, 8]. However, these works mostly consider Hamming distance between log traces and process models, which is usually considered less appropriate than edit distance (c.f. Fig. 1).

The SAT encoding of the edit distance between words has already been introduced in previous works [3, 17]. The work in [3] studies the computation of edit distance for its interest in complexity theory.

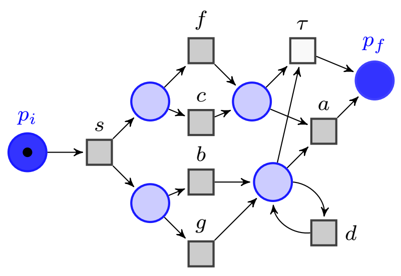
3 Preliminaries

We use labeled Petri nets as process models.

Definition 1 (Process Model (Labeled Petri Net System) [24]) A Process Model defined by a *labeled Petri net system* (or simply *Petri net*) is a tuple $N = \langle P, T, F, m_0, m_f, \Sigma, \Lambda \rangle$, where P is the set of places, T is the set of transitions (with $P \cap T = \emptyset$), $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, m_0 is the initial marking, m_f is the final marking, Σ is an alphabet of actions and $\Lambda : T \to \Sigma \cup \{\tau\}$ labels every transition by an action or as silent.

A *marking* is the set of places that contain tokens for a given instant. A transition x can *fire* if all the places before x, noted ${}^\bullet x \overset{\text{def}}{=} \{y \in P \mid (y, x) \in F\}$, are marked. When a transition fires, all the tokens in ${}^\bullet x$ are removed and all the places in $x^\bullet \overset{\text{def}}{=} \{y \in P \mid (y, x) \in F\}$ become marked. A marking m' is *reachable* from m if there is a sequence of firings $\langle t_1 \dots t_n \rangle$ that transforms m into m', denoted by $m[t_1 \dots t_n\rangle m'$.

Definition 2 (Full runs) A *full run* of a model N is a firing sequence $\langle t_1 \dots t_n \rangle$ of transitions that can transform the initial marking m_0 of N to the final marking m_f of N. We note $Runs(N)$ the full runs of N.



Log traces (some non-fitting):
$\langle s, f, b, a\rangle$
$\langle s, g, c\rangle$
$\langle s, c, b, a\rangle$
$\langle s, g, c, d, d\rangle$
$\langle s, a, a\rangle$

Fig. 2: Example of process model of the behaviors of users rating an app. `s` represents the start activity, `f` and `c` indicate if the user sent a file or wrote a comment. Transitions `g` and `b` separate good and bad marks. Bad ratings get apologies noted by activity `a`. Finally, the `d` loop is enabled when a user donates to the developer of the app.

The run $\langle s, f, b, a\rangle$ is a full run of the model of Fig. 2.

Now we formalize logs:

Definition 3 (Log) A *log* L over an alphabet Σ is a finite set of words, i.e., $L \subseteq \Sigma^*$. An element $\sigma \in L$ is called *log trace*.

3.1 Alignments

Aligning log traces to model traces has been identified as a central problem in Process Mining [1]. The problem is to find a run in the process model that is as close as possible to the observed trace. This closeness is usually defined in terms of the edit distance:

Definition 4 (Edit distance) The *edit distance* $dist(u, v)$ between two words u and $v \in \Sigma^*$ is the minimal number of edits needed to transform u to v. In our case, edits can be deletions or additions of a letter in words.

Example 1 Considering words $u = \langle s, g, c\rangle$ and $v = \langle s, b, c, a\rangle$ the number of edits to transform u to v is indeed 3. The letter g has to be removed and the letters b and a inserted. Then the two words are at distance 3.

Formally, we define alignments in a way that not only achieves the optimal edit distance between the log trace and a model trace, but also makes explicit where they match and mismatch.

Definition 5 (Alignment, optimal alignment) An alignment of a log trace $s = \langle s_1, \ldots, s_m\rangle \in L$ to a run $u = \langle u_1, \ldots, u_n\rangle$ of process model $N = \langle P, T, F, m_0, m_f, \Sigma, \Lambda\rangle$ is a sequence of *moves* $\langle (s'_1, u'_1), \ldots, (s'_p, u'_p)\rangle$ with $p \leq m + n$ such that

– each move (s'_i, u'_i) is either:

  - (e_i, t_i) with $e_i = \Lambda(t_i)$ for a synchronous move
  - $(e_i, \gg)$ for a log move ($\gg$ is a skip symbol which indicates the mismatch), i.e. e_i is deleted in the trace
  - $(\gg, t_i)$ for a model move, i.e. $\Lambda(t_i)$ is inserted in the trace;
- projection of $\langle s'_1, \ldots, s'_p \rangle$ to Σ^* (which drops the occurrences of $\gg$), yields s
- projection of $\langle u'_1, \ldots, u'_p \rangle$ to T^* (which drops the occurrences of $\gg$), yields u

A move $(\gg, \tau)$ is called a silent move, and is considered to not influence in the cost of an alignment. Therefore, an alignment between $u \in \mathit{Runs}(N)$ and s is *optimal* if it minimizes the number of occurrences of $\gg$ in non-silent moves. This minimal number of mismatches corresponds to the edit distance $dist(u, s)$ between u and s.

Alignments can be represented in a two-row matrix. Next figure shows an alignment between the second log trace ($\langle s, g, c \rangle$) and the run $\langle s, b, c, a \rangle$ of the model of Fig. 2.

trace	s	g	$\gg$	c	$\gg$
run	s	$\gg$	b	c	a

In general, aligning a trace to a process model often means searching the minimal number of moves, i.e. *the optimal alignment*. Different cost functions are used but the most common one applies the weights 0 for a synchronous move, and 1 for a log move or a model move. This function is know as the *standard cost function*. Next figure shows an alignment for the trace $\langle s, g, c \rangle$ and the run $\langle s, g, c, \tau \rangle$ of the model of Fig. 2. Since τ moves inccur into no cost, the alignment has cost 0, and hence it is optimal.

trace	s	g	c	$\gg$
run	s	g	c	τ

4 SAT Encoding of the Edit Distance

We first introduce our SAT encoding of the edit distance between two words, which will serve as a building block for computing alignments, multi-alignments and anti-alignments.

The Boolean satisfiability (or SAT) problem, is the problem of determining, for a given Boolean formula, if there exists a combination of assignments to the variables that satisfies it. In the case of alignment, a SAT formula would encode the following question: *Does an alignment exist between the trace σ and the model N for a cost d?* In other words, we are looking for a formula that encodes *the edit distance d between a trace and a run of the model*. Our encoding is based on the same relations that are used by the classical dynamic programming recursive algorithm for computing the edit distance between two words $u = \langle u_1, \ldots, u_n \rangle$ and $v = \langle v_1, \ldots, v_m \rangle$:

$$\begin{cases} dist(\langle u_1, \ldots, u_i \rangle, \epsilon) = i \\ dist(\epsilon, \langle v_1, \ldots, v_j \rangle) = j \\ dist(\langle u_1, \ldots, u_{i+1} \rangle, \langle v_1, \ldots, v_{j+1} \rangle) = \\ \qquad \begin{cases} dist(\langle u_1, \ldots, u_i \rangle, \langle v_1, \ldots, v_j \rangle) & \text{if } u_{i+1} = v_{j+1} \\ 1 + \min(dist(\langle u_1, \ldots, u_{i+1} \rangle, \langle v_1, \ldots, v_j \rangle), \\ \qquad\qquad dist(\langle u_1, \ldots, u_i \rangle, \langle v_1, \ldots, v_{j+1} \rangle)) & \text{if } u_{i+1} \neq v_{j+1} \end{cases} \end{cases}$$

where ϵ denotes the empty word.

We encode this computation in a SAT formula ϕ over variables $\boldsymbol{\delta_{i,j,d}}$, for $i = 0, \ldots, n$, $j = 0, \ldots, m$ and $d = 0, \ldots, n + m$. The formula ϕ will have exactly one solution, in which each variable $\boldsymbol{\delta_{i,j,d}}$ is `true` iff $dist(\langle u_1 \ldots u_i \rangle, \langle v_1 \ldots v_j \rangle) \geq d$.

In order to test equality between the u_i and v_j, we use variables $\boldsymbol{\lambda_{i,a}}$ and $\boldsymbol{\lambda'_{j,a}}$, for $i = 0, \ldots, n$, $j = 0, \ldots, m$ and $a \in \Sigma$, and we set their value such that $\lambda_{i,a}$ is `true` iff $u_i = a$, and $\lambda'_{j,a}$ is `true` iff $v_j = a$. Hence, the test $u_{i+1} = v_{j+1}$ becomes in our formulas: $\bigvee_{a \in \Sigma} (\lambda_{i+1,a} \wedge \lambda'_{j+1,a})$. For readability of the formulas, we refer to this coding by $[u_{i+1} = v_{j+1}]$. We also write similarly $[u_{i+1} \neq v_{j+1}]$.

In the following, we describe the different clauses of the formula ϕ of our SAT encoding of the edit distance[2].

$$\delta_{0,0,0} \quad \wedge \quad \bigwedge_{d>0} \neg \delta_{0,0,d} \tag{1}$$

$$\bigwedge_d \bigwedge_{i=0}^{n} \quad (\delta_{i+1,0,d+1} \Leftrightarrow \delta_{i,0,d}) \tag{2}$$

$$\bigwedge_d \bigwedge_{j=0}^{n} \quad (\delta_{0,j+1,d+1} \Leftrightarrow \delta_{0,j,d}) \tag{3}$$

$$\bigwedge_d \bigwedge_{i=0}^{n} \bigwedge_{j=0}^{n} \quad [u_{i+1} = v_{j+1}] \Rightarrow (\delta_{i+1,j+1,d} \Leftrightarrow \delta_{i,j,d}) \tag{4}$$

$$\bigwedge_d \bigwedge_{i=0}^{n} \bigwedge_{j=0}^{n} \quad [u_{i+1} \neq v_{j+1}] \Rightarrow (\delta_{i+1,j+1,d+1} \Leftrightarrow (\delta_{i+1,j,d} \wedge \delta_{i,j+1,d})) \tag{5}$$

Example 2 At instants $i = 1$ and $j = 1$ of words $u = \langle s, g, c \rangle$ and $v = \langle s, b, c, a \rangle$, the letters are the same, then, by (4), the distance is only higher or equal to 0 : $(u_1 = v_1) \Rightarrow (\delta_{1,1,0} \Leftrightarrow \delta_{0,0,0})$.

However at instants $i = 2$ and $j = 2$, the letters u_2 and v_2 are different. A step before, $\delta_{1,2,1}$ and $\delta_{2,1,1}$ are `true` because of the length of the subwords. Then, by (5), the distance at instants $i = 2$ and $j = 2$ is higher or equal to 2 : $\delta_{2,2,2}$. The result is understandable because the edit distance costs the deletion of g and the addition of b to transform u into v.

5 SAT Encoding of Conformance Checking Artefacts

The distance between log traces and a process model is not only a distance between words, since the process model describe a (possibly infinite) language. In this section, we recall the SAT encoding of full runs of Petri nets [12], and combine it with the encoding of the edit distance to obtain SAT encondings for alignments, multi-alignments and anti-alignments.

[2] We use the logic operator $A \Leftrightarrow B$ as a shortcut to $A \Rightarrow B \wedge B \Rightarrow A$.

5.1 SAT implementation of process models.

For a Petri net $N = \langle P, T, F, m_0, m_f, \Sigma, \Lambda\rangle$ and n the size of the full runs, the variables $\boldsymbol{m}_{i,p}$, with $i \in \{0..n\}$ and $p \in P$, represent the marking at instant i. The variables $\boldsymbol{\mu}_{i,t}$ encode a firing transition $t \in T$ at instant $i \in \{0..n\}$. The following constraints encode the semantics of the Petri net.

– Initial marking:
$$(\bigwedge_{p\in m_0} m_{0,p}) \wedge (\bigwedge_{p\in P\setminus m_0} \neg m_{0,p}) \tag{6}$$
– Final marking:
$$(\bigwedge_{p\in m_f} M_{n,p}) \wedge (\bigwedge_{p\in P\setminus m_f} \neg m_{n,p}) \tag{7}$$
– One and only one t_i for each i:
$$\bigwedge_{i=1}^{n} \bigvee_{t\in T}(\mu_{i,t} \wedge \bigwedge_{t'\in T\setminus\{t\}} \neg \mu_{i,t'}) \tag{8}$$
– The transitions are enabled when they fire:
$$\bigwedge_{i=1}^{n} \bigwedge_{t\in T}(\mu_{i,t} \implies \bigwedge_{p\in {}^\bullet t} m_{i-1,p}) \tag{9}$$
– Token game (for safe Petri nets):
$$\bigwedge_{i=1}^{n} \bigwedge_{t\in T} \bigwedge_{p\in t^\bullet}(\mu_{i,t} \implies m_{i,p}) \tag{10}$$
$$\bigwedge_{i=1}^{n} \bigwedge_{t\in T} \bigwedge_{p\in {}^\bullet t\setminus t^\bullet}(\mu_{i,t} \implies \neg m_{i,p}) \tag{11}$$
$$\bigwedge_{i=1}^{n} \bigwedge_{t\in T} \bigwedge_{p\in P, p\notin {}^\bullet t, p\notin t^\bullet}(\mu_{i,t} \implies (m_{i,p} \iff m_{i-1,p})) \tag{12}$$

Example 3 Consider the model of Fig. 2. At the initialization, place p_i has a token. This information is encoded by $m_{0,p_i} = \texttt{true}$. All the other places do not have a token at this instant, this is described by axiom (6). At the first instant, only transition s can fire. We then have $\mu_{1,s} = \texttt{true}$. All the others transitions do not fire at this instant. For instance $\mu_{1,f}$ is false, i.e., transition f does not fire at instant 1. This behavior is defined by axiom (8). The last axioms defined the token game.

Henceforth, a process model' semantics can be encoded as a SAT formula, and as we will see below, can be combined with log traces for obtaining different conformance artefacts.

5.2 SAT Edit Distance for Alignments

Log traces are sequences of activities that can be considered as words, and hence can be implemented as described in Section 4. SAT encoding of process models has been recalled in previous section. All the above clauses are considered in the SAT implementation of alignments. A last series of constraints is needed to be appended, to relate the fired transitions, represented by the $\boldsymbol{\mu}_{i,t}$, with the actions in the corresponding model trace, represented by variables $\boldsymbol{\lambda}_{i,a}$ from the encoding of Section 4:

$$\bigwedge_{i=1}^{n} \bigwedge_{a\in\Sigma} \left(\lambda_{i,a} \iff \bigvee_{t\in T, \Lambda(t)=a} \mu_{i,t}\right) \tag{13}$$

Trace	Alignment	Distance
$\langle s, f, b, a\rangle$	$\langle s, f, b, a\rangle$	0
$\langle s, g, c\rangle$	$\langle s, g, c, \tau\rangle$	0
$\langle s, c, b, a\rangle$	$\langle s, c, b, a\rangle$	0
$\langle s, g, c, d, d\rangle$	$\langle s, g, c, d, d, \tau\rangle$	0
$\langle s, a, a\rangle$	$\langle s, b, c, a\rangle$	3

Fig. 3: Alignment of each trace and the model of Fig. 2

Example 4 All the full runs of the process model of Fig. 2 contain an s at the first position. So the variable $\mu_{1,s}$ is $\texttt{true}$. If the log trace is $\sigma = \langle s, f, g\rangle$ then, $\lambda_{1,s}$ is $\texttt{true}$ which implies $\delta_{1,1,0}$ by (4).

5.2.1 Minimization of the edit distance.

The conjunction of the previous clauses for the full runs of the model and their edit distance to a given log trace σ, gives a formula which has one solution per full run of the model. With each solution, the values of the $\delta_{n,m,d}$ determine the edit distance between the corresponding model trace and σ. Our goal for optimal alignments is to minimize this distance, which corresponds to the number of variables assigned to $\texttt{true}$ among the $\delta_{n,m,d}$. Pseudo-Boolean solvers like MINISAT+ deal with minimization objectives [3] under the form of a weighted sum of variables; in our case: $\sum_d 1 \times \delta_{n,m,d}$.

5.2.2 How to deal with runs of different length.

In order to consider different sizes of traces and different sizes of runs, we added a loop on a *wait* activity on the final marking of the model. The SAT encoding of the edit distance is adjusted so that skipping a *wait* activity does not increment the distance between words.

Example 5 Fig. 3 shows optimal alignments of every trace of the log of Fig. 2. The deviating trace $\langle s, a, a\rangle$ is then highlighted by the distance to its alignment.

5.3 SAT Implementation for Multi-alignments

Multi-alignments were introduced in [13] as a generalization of alignments. They were recently extended in the contex of model-based trace clustering [8]. Instead of aligning a trace to a run of a process model, multi-alignments consider aligning a set of log traces (typically from the same cluster) to a common run of the model.

Definition 6 (Multi-alignment) Given a finite collection C of log traces and a model N, an (optimal) *multi-alignment* of C to N is a full run $u \in \mathit{Runs}(N)$ which minimizes the maximal distance to the traces : $\max_{\sigma\in C} \mathit{dist}(\sigma, u)$.

[3] SAT problems with minimization objectives and weigthed variables are also called MaxSAT problems

Multi-Alignment	Trace	Distance
$\langle s,g,c,a\rangle$	$\langle s,f,b,a\rangle$	4
	$\langle s,g,c\rangle$	1
	$\langle s,c,b,a\rangle$	2
	$\langle s,g,c,d,d\rangle$	3
	$\langle s,a,a\rangle$	3

Fig. 4: A multi-alignment of the log traces and the model of Fig. 2

Anti-Alignment	Trace	Distance
$\langle s,b,f,d,d,d,d,\tau\rangle$	$\langle s,f,b,a\rangle$	7
	$\langle s,g,c\rangle$	8
	$\langle s,c,b,a\rangle$	7
	$\langle s,g,c,d,d\rangle$	6
	$\langle s,a,a\rangle$	8

Fig. 5: An anti-alignment of the log traces and the model of Fig. 2

The SAT implementation of multi-alignment requires us to duplicate the variables $\lambda^{\sigma}_{i,a}$ that represent actions in the log traces $\sigma \in L$ and the variables $\delta^{\sigma}_{i,j,d}$ that measure the edit distance to the model trace. Section 5.5 details the weighted variables to get the minimization.

Example 6 We computed a multi-alignment of the model and the full log of Fig. 2. An optimal multi-alignment is the full run $\langle s,g,c,a\rangle$ which is at distance $d \leq 4$ to all the log traces. Fig. 4 shows the distance between the multi-alignment and the log traces. Notice that other runs of the model are also at maximal distance 4 to the log traces. There are then many optimal multi-alignments.

5.4 SAT Implementation for Anti-alignments

Anti-alignment was introduced in [12]. Contrary to multi-alignments, the aim of anti-alignments is to get, for a given log, the run of a model which differs as much as possible to all the traces in the log. The notion of anti-alignment is used in some quality metrics like precision and generalization [31].

Definition 7 (Anti-alignment) Given a finite collection L of log traces and a model N, an *anti-alignment* is a run $u \in \mathit{Runs}(N)$ which maximizes the minimal distance $\min_{\sigma\in L} \mathit{dist}(\sigma,u)$ to the log.

The encoding is then very similar to the multi-alignment version. Instead of minimizing the maximal distance to the set of log traces, we maximize the minimal distance using the opposite minimization objective.

Example 7 We computed an anti-alignment of model and the set of log traces of Fig. 2. Limited by a maximum size of run to 8, an optimal anti-alignment found by our tool is $\langle s,b,f,d,d,d,d,\tau\rangle$ (see Fig. 5). The minimal distance between each trace and this full run is 6. We compared our result with the module *Anti-Alignment* of

ProM [4], which computes anti-alignment over Hamming distance. For the same size of run, the algorithm returned the sequence $\langle s, b, d, d, d, c, a, d\rangle$ that is indeed linearly far from the log traces. However as either letters c or a are present in every trace, the run looks more similar to the log than the one found with edit distance.

5.5 Minimization of Minimal Distance versus Sum of Distances

In our previous work [7], we presented a variant of multi-alignment and anti-alignment definitions that aims at optimizing the *sum* of the distances. Although in a way, optimizing the sum of distances tends to obtain in general reasonable witness of multi- and anti-alignments, in some situations they can fail in representing a good solution. In this section we propose a variation of this optimization, which instead focuses on minimizing the minimal distance. We detail the two kinds of MaxSAT problems, and explain their differences.

Minimizing Minimal Distance. As presented in this paper, multi-alignment considers the run of the model that minimizes its maximal distance to the log traces. To produce the general distance of the run and all the traces, we introduce novel variables Δ_d and the following axiom:

$$\bigwedge_d \left(\bigvee_j \delta_{j,n,|\sigma|,d} \Leftrightarrow \Delta_d \right) \tag{14}$$

where n is the size of the run. The Δ_d variables define the distances d for which all the traces verify this distance to the run of the model. The minimization objective for multi-alignment is then : $\sum_d 1 \times \Delta_d$.

Conversely, an anti-alignment is a run of the model which maximizes its minimal distance to the log traces; hence, the SAT objective is this opposite : $\sum_d -1 \times \Delta_d$, with the Δ_d now satisfying:

$$\bigwedge_d \left(\bigwedge_j \delta_{j,n,|\sigma|,d} \Leftrightarrow \Delta_d \right) \tag{15}$$

Minimizing Sum of Distances. A variant of anti-alignment and multi-alignment has been presented in [7]. Instead of minimizing (or maximizing) a distance to the traces of the log, this version of the conformance artefacts minimizes (respectively maximizes) the sum of the distances. The SAT formula does not require variables Δ_d introduced in the previous minimization. Similarly to Section 5.2, the optimal multi-alignment is found by minimizing objective: $\sum_d \sum_{\sigma\in L} 1 \times \delta^{\sigma}_{n,|\sigma|,d}$. The minimization objective for anti-alignment is then: $\sum_d \sum_{\sigma\in L} -1 \times \delta^{\sigma}_{n,|\sigma|,d}$.

The different minimizations presented in this section give different definitions, that may not be used in the same context. To convince readers, we propose the following example:

[4] Anti-alignment Precision/Generalization of package AntiAlignments of ProM software version 6.8, http://www.promtools.org/

Method	Anti-Alignment	$P_{aa}(M,L)$
Minimizing Sum of Distances	$\langle s,f,b,a\rangle$	1
Maximizing Minimal Distance	$\langle s,c,b,a\rangle$	0.75

Table 1: Comparison of minimization method used for Anti-Alignment of model of Fig. 2 and log $L_{aa} = \{\langle s,f,b,a\rangle^1, \langle s,c,g\rangle^{10}\}$

Example 8 A recent metric for precision that uses anti-alignments is the following [11]:

$$P_{aa}(N,L) \overset{\text{def}}{=} 1 - \sup_{\gamma \in \mathcal{L}(N)} dist(\gamma, L)\,.$$

Let's define $L_{aa} = \{\langle s,f,b,a\rangle^1, \langle s,c,g\rangle^{10}\}$ a log of 11 traces that fit in model M of Fig. 2. When one computes precision $P_{aa}(M,L)$, one uses anti-alignments that maximize the minimal distance to any trace of the log. Table 1 highlights a great difference in the use of minimizing the sum of distance, which is not desired in this case. Anti-alignment $\langle s,f,b,a\rangle$ obtains a sum of edit distance equal to 50 but is at minimal distance to any trace to 0 as this trace exists in the log. Anti-alignment $\langle s,c,b,a\rangle$ is at minimal distance to any trace of the log equal to 2 and precision is then not optimal which is much more realistic. The use of the different minimizations is then dependent to the need.

As a concluding remark, the interested reader may have noticed that in contrast to (multi-) alignments, the selection of a particular length for the case of anti-alignment may seem a bit arbitrary at first glance. The previous example illustrates that, when using anti-alignments in practice (e.g., to estimate precision), focusing on a particular length makes sense. The anti-alignments described in Table 1 are both of lengh 4, which is actually the maximal length observed in the log L_{aa}. Since the model of Fig. 2 is fitting, using 4 as the maximal anti-alignment length will not affect considerably deviations that may be less related to the observed patterns in the log. In general, and to be more resilient to fitness and other issues, one may allow certain variations on the maximal observed lenth (e.g., twice the maximal length of a fitting trace observed in the log).

6 Optimizations

For multi-alignment and anti-alignment, the SAT edit distance formula presented in section 4 is duplicated by the number of log traces, which generates thousands of variables (see graphs of Fig. 6). In this section we present a reduction of the formula per artefact. The main idea is to keep only one direction of the double implications. As double implications create clauses in the SAT formula, we improve the size of the formula and its resolution.

6.1 Formula Reduction for Multi-Alignment

As seen in Section 4, the edit distance between two words corresponds to the number of variables assigned to `true` among the $\delta_{n,m,d}$ in the (unique) solution s of the formula ϕ. Let us denote this value $val(s)$. Now, when searching for an alignment to a log trace (consider only alignment to one trace for simplicity), we combine the formula ϕ with a formula ψ which encodes a set of runs of the model, and look for the solution s of the combined formula $\Phi \equiv \phi \wedge \psi$ which minimizes $val(s)$. This minimization amounts to filter the set of solutions of Φ. Here, we show that, relying on this filtering by the minimization, we can reduce the formula Φ (that we now denote $\Phi_{\Leftrightarrow}$) constructed from the formula ϕ of Section 4 (now denoted $\phi_{\Leftrightarrow}$) into a simpler formula $\Phi_{\Leftarrow}$ constructed from a reduced version of $\phi_{\Leftrightarrow}$, denoted $\phi_{\Leftarrow}$ and defined as:

$$\delta_{0,0,0} \quad \wedge \quad \textstyle\bigwedge_{d>0} \neg\delta_{0,0,d} \tag{16}$$

$$\textstyle\bigwedge_d \bigwedge_{i=0}^{n} \quad (\delta_{i+1,0,d+1} \Leftarrow \delta_{i,0,d}) \tag{17}$$

$$\textstyle\bigwedge_d \bigwedge_{j=0}^{n} \quad (\delta_{0,j+1,d+1} \Leftarrow \delta_{0,j,d}) \tag{18}$$

$$\bigwedge_{d} \bigwedge_{i=0}^{n} \bigwedge_{j=0}^{n} \quad [u_{i+1} = v_{j+1}] \Rightarrow (\delta_{i+1,j+1,d} \Leftarrow \delta_{i,j,d}) \tag{19}$$

$$\bigwedge_{d} \bigwedge_{i=0}^{n} \bigwedge_{j=0}^{n} \quad [u_{i+1} \neq v_{j+1}] \Rightarrow (\delta_{i+1,j+1,d+1} \Leftarrow (\delta_{i+1,j,d} \wedge \delta_{i,j+1,d})) \tag{20}$$

Lemma 1 *The minimal value obtained by minimizing $val(s)$ over the solutions of $\Phi_{\Leftarrow}$ is equal to the minimal multi-alignment distance obtained using $\Phi_{\Leftrightarrow}$. Formally, denote $sol(\Phi_{\Leftrightarrow})$ (respectively $sol(\Phi_{\Leftarrow})$) the set of solutions of $\Phi_{\Leftrightarrow}$ (respectively $\Phi_{\Leftarrow}$):*

$$\min_{s \in sol(\Phi_{\Leftarrow})} val(s) = \min_{s' \in sol(\Phi_{\Leftrightarrow})} val(s') \ .$$

Proof We represent every solution of a SAT formula as an application $s : Vars \rightarrow \{\texttt{true}, \texttt{false}\}$ where $Vars$ is the set of variables of the formula, so that $s(v)$ denotes the value assigned to variable v in s.

1. $\min_{s \in sol(\Phi_{\Leftarrow})} val(s) \geq \min_{s' \in sol(\Phi_{\Leftrightarrow})} val(s')$: As $\Phi_{\Leftarrow}$ is defined like $\Phi_{\Leftrightarrow}$ with less constrains, we have $\Phi_{\Leftrightarrow} \Rightarrow \Phi_{\Leftarrow}$, then $sol(\Phi_{\Leftrightarrow}) \subseteq sol(\Phi_{\Leftarrow})$ which implies $\min_{s \in sol(\Phi_{\Leftarrow})} val(s) \geq \min_{s' \in sol(\Phi_{\Leftrightarrow})} val(s')$.
2. $\min_{s \in sol(\Phi_{\Leftarrow})} val(s) \leq \min_{s' \in sol(\Phi_{\Leftrightarrow})} val(s')$: Let $s \in sol(\Phi_{\Leftarrow})$. We will show how to create $s' \in sol(\Phi_{\Leftrightarrow})$ such that $val(s') \geq val(s)$. We define s' as follows :
   - $\forall_{i \in \{0...n\},\, p \in P} \quad s'(m_{i,p}) := s(m_{i,p})$
   - $\forall_{i \in \{1...n\},\, a \in \Sigma} \quad s'(\mu_{i,a}) := s(\mu_{i,a})$
   - $\forall_{\sigma \in L,\, i \in \{1...|\sigma|\},\, a \in \Sigma} \quad s'(\lambda^{\sigma}_{i,a}) := s(\lambda^{\sigma}_{i,a})$

   and
   - $\forall_{\sigma \in L,\, i \in \{1...n\}, j \in \{1...|\sigma|\},\, d \in \{0...(n+|\sigma|)\}}$ $s'(\delta^{\sigma}_{i,j,d}) := (dist(\langle u_1,..,u_i\rangle, \langle \sigma_1,..,\sigma_j\rangle) \geq d)$ where $dist$ is the edit distance defined in section 3. I.e., s' assigns the values for the variables $\delta^{\sigma}_{i,j,d}$ according to the exact edit distance while solution s represents under-approximation of the distances.

We then demonstrate that s' is indeed a solution of $\Phi_{\Leftrightarrow}$:

- variables $m_{i,p},\ \mu_{i,a}$ and $\lambda^{\sigma}_{i,a}$ are assigned like in s which is a solution of $\Phi_{\Leftarrow}$ and those variables are not affected by the reduction.
- variables $\delta^{\sigma}_{i,j,d}$ are defined using the edit distance which verifies axioms (2) to (5) of $\Phi_{\Leftrightarrow}$.

Finally, we show that $val(s) \leq val(s')$: As, for multi-alignment, we minimize the distances, let's demonstrate that $s'(\delta^{\sigma}_{i,j,d}) \Rightarrow s(\delta^{\sigma}_{i,j,d})$ for $\sigma \in L$, $i \in \{0,..,n\}$, $j \in \{0,..,|\sigma|\}$ and $d \in \{0,..,(n+|\sigma|)\}$.
Let $\sigma \in L$. We prove by induction on n that:

$$\forall_{i,j,i+j\leq n} \quad \forall_d \quad (dist(\langle u_1,\dots,u_i\rangle,\langle\sigma_1,\dots,\sigma_j\rangle) \geq d) \Rightarrow s(\delta^{\sigma}_{i,j,d}).$$

- Initialization : for $n=0$, which implies $i=0$ and $j=0$, we have:
  - $d=0 \quad dist(\epsilon,\epsilon)=0$ and $\delta^{\sigma}_{0,0,d}=$ `true`.
    Then it verifies $(dist(\epsilon,\epsilon)\geq 0) \Rightarrow s(\delta^{\sigma}_{0,0,0})$
  - $\forall_{d>0}, \quad dist(\epsilon,\epsilon) < d$ and $\delta^{\sigma}_{0,0,d}=$ `false`.
    Then it verifies $(dist(\epsilon,\epsilon)\geq d) \Rightarrow s(\delta^{\sigma}_{0,0,d})$.
- Induction step: Assuming that $(dist(\langle u_1,\dots,u_i\rangle,\langle\sigma_1,\dots,\sigma_j\rangle) \geq d) \Rightarrow s(\delta^{\sigma}_{i,j,d})$ holds for all i,j,d such that $i+j\leq n$, we show that the implication still holds when $i+j=n+1$:
  - for $i = 0$ (i.e. $u = \epsilon$): assume $(dist(\epsilon,\langle\sigma_1,\dots,\sigma_j\rangle) \geq d)$ is true. By definition of the edit distance, $dist(\epsilon,\langle\sigma_1,\dots,\sigma_j\rangle) = 1 + dist(\epsilon,\langle\sigma_1,\dots,\sigma_{j-1}\rangle$, which implies $dist(\epsilon,\langle\sigma_1,\dots,\sigma_{j-1}\rangle) \geq d-1$. By the induction we know that $(dist(\epsilon,\langle\sigma_1,\dots,\sigma_{j-1}\rangle) \geq d-1) \Rightarrow s(\delta^{\sigma}_{0,j-1,d-1})$. And finally, since s satisfies axiom (17), we have $s(\delta^{\sigma}_{0,j,d})$.
  - for $j = 0$ (i.e. $\sigma = \epsilon$): the proof is similar to the previous case (with axiom (18).
  - for $u_i = \sigma_j$: assume $dist(\langle u_1,\dots,u_i\rangle,\langle\sigma_1,\dots,\sigma_j\rangle) \geq d$. By the induction hypothesis, $(dist(\langle u_1,\dots,u_{i-1}\rangle,\langle\sigma_1,\dots,\sigma_{j-1}\rangle) \geq d) \Rightarrow s(\delta^{\sigma}_{i-1,j-1,d})$. Here, $dist(\langle u_1,\dots,u_{i-1}\rangle,\langle\sigma_1,\dots,\sigma_{j-1}\rangle) = dist(\langle u_1,\dots,u_i\rangle,\langle\sigma_1,\dots,\sigma_j\rangle)$ by definition of the edit distance. Then $s(\delta^{\sigma}_{i-1,j-1,d})$ holds, and, since s satisfies axiom (20), we have $s(\delta^{\sigma}_{i,j,d})$.
  - for $u_i \neq \sigma_j$: if $(dist(\langle u_1,\dots,u_i\rangle,\langle\sigma_1,\dots,\sigma_j\rangle) \geq d)$ is true, then, by definition of the edit distance, $(dist(\langle u_1,\dots,u_{i-1}\rangle,\langle\sigma_1,\dots,\sigma_j\rangle) \geq d-1)$ and $(dist(\langle u_1,\dots,u_i\rangle,\langle\sigma_1,\dots,\sigma_{j-1}\rangle) \geq d-1)$. As $i-1+j=n$ and $i+j-1=n$, we use the induction hypothesis to get $s(\delta^{\sigma}_{i-1,j,d}) =$ `true` and $s(\delta^{\sigma}_{i,j-1,d}) =$ `true`. From axiom (20), we obtain $s(\delta^{\sigma}_{i1,j,d}) =$ `true`.

6.2 Formula Reduction for Anti-Alignment

In the previous section, we have shown how, relying on the minimization of the $\delta_{n,m,d}$ variables, we can reduce $\Phi_{\Leftrightarrow}$ to $\Phi_{\Leftarrow}$. Conversely, for anti-alignment, we found that $\Phi_{\Leftrightarrow}$ can be simplified relying on the maximization. We demonstrate how $\Phi_{\Leftrightarrow}$ can be reduced into $\Phi_{\Rightarrow}$, described below, when $\delta_{n,m,d}$ variables are maximized.

$$\delta_{0,0,0} \quad \wedge \quad \bigwedge_{d>0} \neg\delta_{0,0,d} \tag{21}$$

$$\bigwedge_d \bigwedge_{i=0}^{n} \quad (\delta_{i+1,0,d+1} \Rightarrow \delta_{i,0,d}) \tag{22}$$

$$\bigwedge_d \bigwedge_{j=0}^{n} \quad (\delta_{0,j+1,d+1} \Rightarrow \delta_{0,j,d}) \tag{23}$$

$$\bigwedge_d \bigwedge_{i=0}^{n} \bigwedge_{j=0}^{n} \quad [u_{i+1} = v_{j+1}] \Rightarrow (\delta_{i+1,j+1,d} \Rightarrow \delta_{i,j,d}) \tag{24}$$

$$\bigwedge_d \bigwedge_{i=0}^{n} \bigwedge_{j=0}^{n} \quad [u_{i+1} \neq v_{j+1}] \Rightarrow (\delta_{i+1,j+1,d+1} \Rightarrow (\delta_{i+1,j,d} \wedge \delta_{i,j+1,d})) \tag{25}$$

Lemma 2 *The maximal value obtained by maximizing $val(s)$ over the solutions of $\Phi_{\Rightarrow}$ is equal to the maximal anti-alignment distance obtained using $\Phi_{\Leftrightarrow}$. Formally:*

$$\max_{s \in sol(\Phi_{\Rightarrow})} val(s) = \max_{s' \in sol(\Phi_{\Leftrightarrow})} val(s') \; .$$

Proof Similarly to the reduction to $\Phi_{\Leftarrow}$ in Lemma 1, we show that $\Phi_{\Leftrightarrow}$ and $\Phi_{\Rightarrow}$ define the same anti-alignment distance when we maximize $val(s)$:

1. $\max_{s \in sol(\Phi_{\Rightarrow})} val(s) \geq \max_{s' \in sol(\Phi_{\Leftrightarrow})} val(s')$: The proof is exactly the same as for multi-alignments (see Lemma 1).
2. $\max_{s \in sol(\Phi_{\Rightarrow})} val(s) \leq \max_{s' \in sol(\Phi_{\Leftrightarrow})} val(s')$: The idea of the proof is similar to $\Phi_{\Leftarrow}$ reduction. We create $s' \in sol(\Phi_{\Leftrightarrow})$ such as, for $s \in sol(\Phi_{\Rightarrow})$, $\quad val(s) \leq val(s')$. This is proved by induction with : $\forall_{i,j,i+j \leq n} \quad \forall_d \quad s(\delta^{\sigma}_{i,j,d}) \Rightarrow (dist(\langle u_1, \ldots, u_i \rangle, \langle \sigma_1, \ldots, \sigma_j \rangle) \geq d)$.

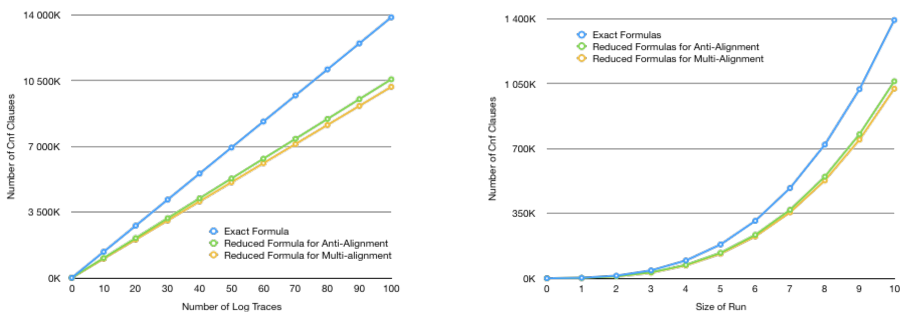
7 Experiments

In this section, we show different experiments of the SAT encoding artefacts over edit distance. We first present our new implementation, `da4py`. Then in section 7.2 we report on the impact of optimizing the formulas as described in Section 6. In section 7.3, we compare previous running times to the new optimized implementation. Finally, in 7.4 we present new experiments on real-life logs.

7.1 da4py: Python Librairy for Conformance Artefacts

In [7], we presented an Ocaml version of the SAT encoding called DARKSIDER. In this paper, we present `da4py`, a novel Python library that implements the three edit distance SAT encodings presented above[5]. With this new library, we improve the efficiency of artefacts computation in different ways. First, the reduced formulas optimize the number of clauses and variables, which accelerates the formula construction and resolution. Due to the use of the SAT library `PySAT` [18], `da4py` runs RC2, a MaxSAT algorithm that finds solutions faster by using heuristics. Finally, one can

[5] The `da4py` library and examples is available at `https://github.com/BoltMaud/da4py`.



(a) Number of clauses per formula for a size of run to 10 by increasing number of traces

(b) Number of clauses per formula for a log of 10 traces by increasing size of run

Fig. 6: Comparison of Number of CNF Clauses of Produced Formulas for a model of 10 transitions.

easily choose its SAT solver in a set of state-of-the-art solvers with `PySAT`. `da4py` is compatible with the library `pm4py` [5], and uses the same data objects.

Like DARKSIDER, `da4py` proposes two heuristics to deal with large logs. To alleviate the complexity, one can compute a prefix of artefacts instead of requiring a full run (parameter PRE). Furthermore, another simplification is to add a threshold on the number of editions between the run and the traces (parameter LIM). This solution computes a lower-bound for the anti-alignment, instead of the complete anti-alignment.

7.2 Optimization Analysis

In the Section 6, we show two optimizations of the SAT encoding. To convince readers, we present Fig. 6 that reports the size of the formulas in term of *Conjunctive normal form* (CNF) clauses. The formulas have been created for a proces model consisting of 12 transitions. In Fig. 6a, for 0 traces, the number of clauses of the formula represents the number of clauses needed to create the SAT encoding of the model, described in Section 5.1. This number is not significant when compared to the size of the edit distance formulas, as the number of traces increases. In Fig. 6b, we show the number of clauses when increasing the size of the run. As expected, the exact formula creates much more clauses than the optimized variants. We see that the reduced versions do not define the same number of clauses, which is also expected because we do not keep the same side of the implications in the two reductions.

7.3 Artificial Log - Improvement Results

In this section, we compare the previous work [7] to the optimized one presented in this paper with artificial models and logs. To illustrate the differences, we decided to show the same table than the one in [7] (see Table 2), with a new column for `da4py` results. Notice that while for alignments we show average numbers over one hundred traces, for multi- and anti-alingments (where only a run is computed for the whole log), total numbers are provided. The first two columns (Model and $|L|$) describe the model and the log sizes, respectively. Column "Size of run" shows the maximal size allowed for the run in the model (which will be an alignment in (a), a multi-alignment in (b), and an anti-alignment in (c)). For memory issues, we sometimes had to limit the length of the run as explained in section 7.1. Then, if only prefix of runs are computed, we specify it with PRE notation. Moreover, the fourth column reports the maximal number of edits allowed (see section 7.1). When LIM is indicated, the distance between the model and the trace is larger than what was tested. The last columns show computation times.

For simple alignment, our SAT encoding still do not beat PROM implementation time scores, but our new optimized encoding improves significantly the baseline SAT encoding of our previous works. Clearly, the incorporation of several engineering efforts like caching intermediate solutions, will make our solution at the same implementation level than the one in PROM, so that the comparison is more fair. For multi-aligment and anti-alignment, `da4py` gives the best computation times of the state-of-the-art.

7.4 Real-life Logs

By reducing the size of the SAT encoding with the optimizations proposed in this paper, our new implementation can handle now real-life problems that were intractable with the baseline encoding of previous work. The following set of experiments has been run in a virtual machine with 12 CPU Intel Xeon 2.67GHz and 50GB RAM.

We launch experiments over the twelve real-life logs available at the 4DTU Data Center[6]. For each log, we compute anti-alignments and multi-alignments on sublogs and models discovered with both the Inductive Miner [21], and the Split Miner [2]. Table 3 shows average execution times of the twelve logs and corresponding models. As sizes of models vary from 8 to 150 transitions, we also give minimal and maximal execution times. The results show very similar execution times for anti-alignment and multi-alignments, which is expected as the formulas are very similar. The library is able to handle sublogs of 200 traces and execution times are reasonable. However, because of memory space, we had to limit the size of run and the number of edits. Even for 100 traces, we see that, for a prefix of 10, parameters are bounded, i.e., the maximal number of edits is lower than the real possible number of edits for a size of run to 10. The size of the run and maximal number of edits increase a lot the formulas size, and it is still too complex to get full runs without these limitations in the search. Still, this is the first time a tool is able to compute multi-alignment and anti-alignment on real-life logs of this nature.

[6] `https://data.4tu.nl/repository/collection:event_logs_real`

Model			$\lvert L\rvert$	Size of run	Maximal number of edits	Darksider execution time (sec)	da4py execution time (sec)	ProM execution time (sec)
Reference	$\lvert T\rvert$	$\lvert P\rvert$						
Fig. 2	8	7	100	7	5	0.349	0.264	0.002
M8 of [27]	15	17	100	PRE: 20	LIM:10	15.530	6.781	0.001
M1 of [27]	40	39	100	PRE: 7	LIM:10	7.16	2.845	0.005
Loan [9]	15	16	100	PRE: 19	LIM: 10	20.915	6.169	0.002

(a) Alignments (showing averages).

Fig. 2	8	7	10	8	7	15.362	4.406	/
			100	8	7	200.569	34.607	
M8 of [27]	15	17	10	18	LIM: 6	414.174	29.953	/
			100	PRE: 15	LIM: 6	741.162	196.412	
M1 of [27]	40	39	10	PRE: 13	LIM: 10	172.500	59.362	/
			100	PRE: 13	LIM: 5	1066.94	266.670	
Loan [9]	15	16	10	PRE: 19	15	373.683	81.854	/
			100	PRE: 9	LIM:10	508.542	120.352	

(b) Multi-alignments.

Fig. 2	8	7	10	8	LIM: 10	21.502	6.127	/
			100	8	LIM: 10	243.842	51.563	
M8 of [27]	15	17	10	18	LIM: 10	148.271	47.086	/
			100	PRE: 10	LIM: 10	496.733	146.478	
M1 of [27]	40	39	10	39	LIM: 10	2069.505	477.036	/
			100	PRE: 13	LIM: 5	995.361	265.132	
Loan [9]	15	16	10	PRE: 19	LIM: 10	203.257	56.173	/
			100	PRE: 19	LIM: 10	2185.785	121.114	

(c) Anti-alignments.

Table 2: Experimental results for the computation of optimum and approximations of alignments and anti-alignments over Edit Distance with our Python library `da4py`, obtained on a virtual machine with 12 CPU Intel Xeon 2.67GHz and 50GB RAM.

$\lvert L\rvert$	Size of run	Maximal number of edits	Method	Inductive Miner			Split Miner		
				avg	min	max	avg	min	max
100	5	10	MA	150.008	25.162	305.978	127.379	37.201	234.195
			AA	151.253	25.553	309.400	129.829	37.241	237.013
200	5	10	MA	295.830	45.377	605.277	252.437	67.537	466.752
			AA	298.425	46.598	611.877	252.502	68.048	465.374
100	10	10	MA	614.136	95.212	1245.584	500.603	143.315	926.757
			AA	600.697	101.354	1243.523	501.265	145.366	923.399

Table 3: Multi-Alignments (MA) and Anti-Alignments (AA) Execution Times (in seconds) by `da4py` using the twelve real-life logs.

8 Theoretical and Experimental Complexity

Deciding the existence of alignments, anti-alignments or multi-alignments (of size bounded by an integer n given as input in unary) are NP-complete problems [12,8]. In this paper we propose to encode them as SAT instances and rely on efficient SAT solvers to compute the artefacts. The dominating factor in the time complexity of our technique is to solve the formulas, i.e. the call to the SAT solver dominates the complexity. The size of our formulas (and the computation time to construct them) are polynomial in the input.

More precisely, the formula that encodes the edit distance between two words of lengths n and m, has size $O((n+m) \times n \times m)$. The formula for runs of length n of a model $N = \langle P, T, F, m_0, m_f, \Sigma, \Lambda \rangle$ has size $O(|P| + n \times (|T|^2 + |F|))$. The $n \times |T|^2$ comes from the SAT formula (8) which enumerates pairs of transitions; it is immediate to encode the same constraint as a pseudo-SAT formula of size $O(n \times |T|)$ using the ability of pseudo-SAT to express directly constraints on the number of variables assigned to `true` among a set of variables. Hence, the encoding of the model runs has size $O(n \times |F|)$.

For anti-alignments or multi-alignments, we need to repeat the encoding of the edit distance for each log trace. The size of the final formula sums to $O((n+m) \times n \times m \times |L| + n \times |F|)$, where m is the maximum length of log traces. The $(n+m)$ factor can be reduced significantly by setting a limit threshold to the value of d in the computation of edit distances (parameter LIM of Section 7.1) when the distances of interest are expected to be sufficiently small. With this threshold, our formulas are essentially:

- linear in the size of the model
- linear in the size of the log
- quadratic in the length n of the considered anti- or multi-alignments. Actually, going further with the heuristic using threshold LIM for edit distance, one could eliminate all the variables $\delta_{i,j,d}$ with $|j-i| \geq$ LIM and then make our formulas linear also in n. We have not implemented this optimization.

The optimizations presented in Section 6 have a very significant impact in practice but do not change the theoretical complexity.

In practice, what limits our approach is mainly memory used by the solver. While in theory, solving a SAT formula requires only linear space, in practice, solvers tend to store information in order to improve their time complexity. On the other hand, the time required to solve our formulas in practice, does not grow as fast as the exponential than one could expect from the theoretical complexity.

9 Conclusion

This paper has shown a unified approach to compute important conformance artefacts over SAT formulas. Thanks to its high versatility, the encoding as SAT formula allows one to compute exact solutions to various problems (here alignments, anti-alignments and multi-alignments), under various optimality criteria. For multi-alignment and anti-alignments, two minimizations, respectively maximizations, of the distances between the traces and the models are proposed. Moreover, a new optimized version of the baseline SAT encodings is presented, for which formal proofs of correctness are provided. These optimization are crucial to widen the applicability of the family of the alignment-based techniques grounded in SAT to realistic, large conformance checking instances.

For future work, several directions will be considered. First, we aim at exploring new optimizations that can allow a further reduction of the SAT formulas arising from encoding conformance checking artefacts. Second, we will consider how to incorporate projection mechanisms into the SAT encoding techniques, so that only a fraction of the process model is considered. Third, we will explore the combination of the techniques of this paper with log sampling, following the interesting line conducted in [4]. Finally, we aim at combining the current work with model repair.

Acknowledgments.

This work has been supported by Farman institute at ENS Paris-Saclay and by MINECO and FEDER funds under grant TIN2017-86727-C2-1-R.

References

1. A. Adriansyah. *Aligning observed and modeled behavior*. PhD thesis, Department of Mathematics and Computer Science, 2014.
2. Adriano Augusto, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, and Artem Polyvyanyy. Split miner: automated discovery of accurate and simple business process models from event logs. *Knowl. Inf. Syst.*, 59(2):251–284, 2019.
3. Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 51–58. ACM, 2015.
4. Martin Bauer, Han van der Aa, and Matthias Weidlich. Estimating process conformance by trace sampling and result approximation. In *Business Process Management - 17th International Conference, BPM 2019, Vienna, Austria, September 1-6, 2019, Proceedings*, pages 179–197, 2019.
5. Alessandro Berti, Sebastiaan J van Zelst, and Wil van der Aalst. Process Mining for Python (PM4Py): Bridging the Gap Between Process-and Data Science. pages 13–16, 2019.
6. Vincent Bloemen, Jaco van de Pol, and Wil M. P. van der Aalst. Symbolically aligning observed and modelled behaviour. In *18th International Conference on Application of Concurrency to System Design, ACSD, Bratislava, Slovakia, June 25-29*, pages 50–59, 2018.
7. Mathilde Boltenhagen, Thomas Chatain, and Josep Carmona. Encoding conformance checking artefacts in SAT. In *Business Process Management Workshops*. Springer, 2019.
8. Mathilde Boltenhagen, Thomas Chatain, and Josep Carmona. Generalized alignment-based trace clustering of process behavior. In *Proceedings of the 40th International Conference on Applications and Theory of Petri Nets (ICATPN'19)*, number 11522 in Lecture Notes in Computer Science. Springer, 2019.
9. J.C.A.M. Buijs. Loan application example. 4TU. Centre for Research Data. Dataset. doi.org/10.4121, 2013.
10. Josep Carmona, Boudewijn F. van Dongen, Andreas Solti, and Matthias Weidlich. *Conformance Checking - Relating Processes and Models*. Springer, 2018.
11. Thomas Chatain, Mathilde Boltenhagen, and Josep Carmona. Anti-Alignments – Measuring The Precision of Process Models and Event Logs. Report hal-02383546: https://hal.inria.fr/hal-02383546, November 2019.

12. Thomas Chatain and Josep Carmona. Anti-alignments in conformance checking–the dark side of process models. In *International Conference on Application and Theory of Petri Nets and Concurrency*, pages 240–258. Springer, 2016.
13. Thomas Chatain, Josep Carmona, and Boudewijn Van Dongen. Alignment-based trace clustering. In *International Conference on Conceptual Modeling*, pages 295–308. Springer, 2017.
14. Ian Davidson, SS Ravi, and Leonid Shamis. A sat-based framework for efficient constrained clustering. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 94–105. SIAM, 2010.
15. Massimiliano de Leoni and Andrea Marrella. Aligning real process executions and prescriptive process models through automated planning. *Expert Syst. Appl.*, 82:162–183, 2017.
16. Luciano García-Bañuelos, Nick R.T.P. van Beest, Marlon Dumas, Marcello La Rosa, and Willem Mertens. Complete and interpretable conformance checking of business processes. *IEEE Transactions on Software Engineering*, 44(3):262–290, March 2018.
17. Alex Groce, Sagar Chaki, Daniel Kroening, and Ofer Strichman. Error explanation with distance metrics. *International Journal on Software Tools for Technology Transfer*, 8(3):229–247, 2006.
18. Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018.
19. Wai Lam Jonathan Lee, H. M. W. Verbeek, Jorge Munoz-Gama, Wil M. P. van der Aalst, and Marcos Sepúlveda. Recomposing conformance: Closing the circle on decomposed alignment-based conformance checking in process mining. *Inf. Sci.*, 466:55–91, 2018.
20. Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Scalable process discovery and conformance checking. *Software and System Modeling*, 17(2):599–631, 2018.
21. Sander JJ Leemans, Dirk Fahland, and Wil MP van der Aalst. Discovering block-structured process models from event logs containing infrequent behaviour. In *International conference on business process management*, pages 66–78. Springer, 2013.
22. Jean-Philippe Métivier, Patrice Boizumault, Bruno Crémilleux, Mehdi Khiari, and Samir Loudni. Constrained clustering using sat. In *International Symposium on Intelligent Data Analysis*, pages 207–218. Springer, 2012.
23. Jorge Munoz-Gama, Josep Carmona, and Wil M. P. Van Der Aalst. Single-entry single-exit decomposed conformance checking. *Inf. Syst.*, 46:102–122, December 2014.
24. T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–574, April 1989.
25. Daniel Reißner, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, and Abel Armas-Cervantes. Scalable conformance checking of business processes. In *OTM CoopIS, , Rhodes, Greece*, pages 607–627, 2017.
26. Niek Tax, Xixi Lu, Natalia Sidorova, Dirk Fahland, and Wil M. P. van der Aalst. The imprecisions of precision measures in process mining. *Inf. Process. Lett.*, 135:1–8, 2018.
27. Farbod Taymouri and Josep Carmona. Model and event log reductions to boost the computation of alignments. In *Proceedings of the 6th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2016), Graz, Austria, December 15-16, 2016.*, pages 50–62, 2016.
28. Farbod Taymouri and Josep 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.
29. Wil M. P. van der Aalst. Decomposing petri nets for process mining: A generic approach. *Distributed and Parallel Databases*, 31(4):471–507, 2013.
30. Boudewijn 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*, pages 197–214, 2018.
31. Boudewijn F. van Dongen, Josep Carmona, and Thomas Chatain. A unified approach for measuring precision and generalization based on anti-alignments. In *Business Process Management - 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016. Proceedings*, pages 39–56, 2016.
32. Boudewijn F. van Dongen, Josep Carmona, Thomas Chatain, and Farbod Taymouri. Aligning modeled and observed behavior: A compromise between computation complexity and quality. In *Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings*, pages 94–109, 2017.
33. H. M. W. Verbeek and W. M. P. van der Aalst. *Merging Alignments for Decomposed Replay*, pages 219–239. Springer International Publishing, Cham, 2016.