A Discounted Cost Function for Fast Alignments of Business Processes

Mathilde Boltenhagen[1(✉)], Thomas Chatain[1(✉)], and Josep Carmona[2(✉)]

[1] Université Paris-Saclay, CNRS, ENS Paris-Saclay, Inria, LMF, Gif-sur-Yvette, France
{boltenhagen,chatain}@lsv.fr
[2] Universitat Politècnica de Catalunya, Barcelona, Spain
jcarmona@cs.upc.edu

Abstract. Alignments are a central notion in conformance checking. They establish the best possible connection between an observed trace and a process model, exhibiting the closest model run to the trace. Computing these alignments for huge amounts of traces, coming from big logs, is a computational bottleneck. We show that, for a slightly modified version of the distance function between traces and model runs, we significantly improve the execution time of an A*-based search algorithm. We show experimentally that the alignments found with our modified distance approximate very nicely the optimal alignments for the classical distance.

1 Introduction

Conformance checking techniques establish relations between modeled and observed behavior [6]. The techniques on this field are grounded on solving a very particular problem, known as *alignment* [1]: given a process model that describes a certain process, and a trace which contains a potential observation of this process, to decide if the trace is in the language of the model, and if not, to pinpoint where it deviates. Computing alignments is not necessarily the ultimate goal of an analysis, but instead can be used to further enhance a process model with the evidences found in the data, e.g. depicting explicitly in the model the bottlenecks of the underlying process [23].

The current process mining field is living an interesting paradox: whilst it is widely accepted that the capabilities of discovering huge process models exist, when it comes to analysing these discovered models through conformance checking techniques, only approximate techniques for deriving alignments are used in practice. In the next section we provide a complete overview of current alternatives for alignment computation.

In this paper, however, we somehow go back to the roots, and adopt the seminal work from Arya Adriansyah's PhD thesis [1] as main alignment algorithm. It consists of an A^* graph search algorithm over the state space of a *synchronous product net* made out of the initial process model and a *trace net* corresponding to the input trace. The cost function that governs the A^* search is typically a *standard* cost function which assigns unitary costs to all the possible types of deviations.

A. Polyvyanyy et al. (Eds.): BPM 2021, LNCS 12875, pp. 252–269, 2021.
https://doi.org/10.1007/978-3-030-85469-0_17

We consider a rather simple, yet powerful idea that is motivated from the following use case: for certain processes, the costs associated to deviations at early stages of the process' execution are more important than the ones at the end. For instance, consider a loan application process that has two decisions: one at the beginning, assessing the type of customer (gold, silver, normal), and one at the end, determining whereas the loan was received in a labour day or not. It is normal that the stage in which these decisions are made in any possible execution of the process reveal their importance. For instance, if for the company it is very important to know the type of the customer because further information needs to be gathered depending on the customer's type, then it is likely that the corresponding process has the type of customer decision close to the start of any possible execution. On the contrary, if the day when the loan was received is not so important, then it is likely that the corresponding events will be pushed to the end of the traces.

The aforementioned situation holds for *knock-out processes* [24], representing processes where two outcomes are possible: OK or NOK. In these processes, ordered checks are usually observed, because it allows faster process executions. Indeed, as many tasks of those processes aim at determining the final output, the knock-out decisions should be taken at the beginning of the process, in growing order [14].

Having this in mind, one can instantiate the A^* algorithm to make the cost function exponentially biased to this use case: giving more importance (higher cost) to the deviations that occur in early stages of the alignments, and exponentially reducing the cost as the search algorithm progresses. Importantly, this *discounted* cost function has a huge impact on the size of the search space required for the A^* search, since the cost asymmetry makes the search space rapidly shrink after the first alignment steps are made. For processes which follow the aforementioned use case, this cost function puts the search focus in the right place, deriving alignments that aim at synchronizing modeled and observed behavior in the important decisions that are made at the beginning. Interestingly, this idea can also be used for processes that are not following this trend, since although putting the focus at the beginning may not be the most likely explanation, the computational alleviation can make the problem tractable, where other techniques fail.

In this paper we formalize this simple idea, and show the great impact in performance with respect to several variations of the A^* search proposed in the last years. Interestingly, this improvement causes only a very minor loss in quality: as we will see in the experiments, for well-known and accepted benchmarks, the proposed techniques are often able to produce alignments very close to the optimal ones.

This paper is organized as follows: in the next section we provide a detailed overview of the different techniques to compute alignments. Then in Sect. 3 we provide the necessary definitions to understand the technique of this paper. In Sects. 4 and 5 we provide the formal definition and corresponding algorithmic adaptations for the discounted cost function presented in this paper. Then in Sect. 6 an evaluation of the proposed technique is reported, and Sect. 7 concludes the paper.

2 Related Work

The seminal work in [1] proposed the notion of alignment and developed a technique based on A* to compute optimal alignments for a particular class of process models. The approach represents the state-of-the-art technique for computing alignments, and can be adapted (at the expense of increasing significantly the memory footprint) to provide all optimal alignments. Alternatives to A* have appeared in recent years: in the approach presented in [7], the alignment problem is mapped as an *automated planning* instance. Automata-based techniques have also appeared [10,15]. The techniques in [15] (recently extended in [16]) rely on state-space exploration and determination of the automata corresponding to both the event log and the process model, whilst the technique in [10] is based on computing several subsets of activities and projecting the alignment instances accordingly.

The work in [19] presented the notion of *approximate* alignment to alleviate the computational demands by proposing a recursive paradigm on the basis of the structural theory of Petri nets. In spite of resource efficiency, the solution is not guaranteed to be executable. Alternatively, the technique in [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 local search have been also proposed very recently [21].

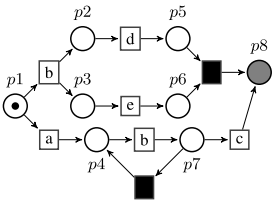
Decompositional techniques have been presented [12,22,27] that, instead of computing optimal alignments, focus on the *crucial problem* of whether a given trace fits or not a process model. These techniques vertically decompose the process model into pieces satisfying certain conditions (so only *valid* decompositions [22], which satisfy restrictive conditions on the labels and connections forming a decomposition, guarantee the derivation of a real alignment). Later on, the notion of *recomposition* has been proposed on top of decompositional techniques, in order to obtain optimal alignments whenever possible by iterating the decompositional methods when the required conditions do not hold [9]. In contrast to the aforementioned vertical decomposition techniques, our methodology does not require this last consolidation step of partial solutions, and therefore can be a fast alternative to these methods at the expense of loosing the guarantee of optimality.

We believe our work has similarities and synergies with two recent works. In [5], a symbolic algorithm to maximize the number of synchronous moves in the alignment is proposed, by changing the cost function to only penalize log moves, thus allowing an arbitrary number of model moves if this contributes to maximizing synchronous moves. We believe the discounted cost function of this paper may be used in the context of [5], to balance better the solutions found. Recently, in [25], an online alignment technique with a window-based backwards exploration is proposed. Again, by discounting this window-based exploration, a speedup of the online technique can be obtained so that it can be applied on a larger problem instances.

3 Preliminaries

We represent event data as log traces and process models as labeled Petri nets.

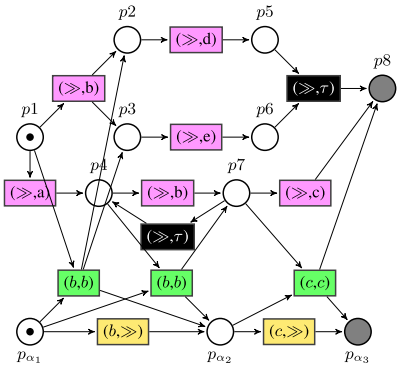
Definition 1 (Log Traces). *Let Σ be a set of activities. We define a* log *L as a finite multiset of sequences $\sigma \in \Sigma^*$, which we refer to as* log traces.



(a) Process Model N

$\langle b, c \rangle$

(b) Log trace σ



(c) Synchronous Product between N and σ. Transitions in purple correspond to the model moves, green transitions are the synchronous moves and yellow transitions the log moves. Black transitions are silent.

Fig. 1. Synchronous product for alignments between N and σ that has the marking reachability problem.

Definition 2 (Process Model (Labeled Petri Net) [13]). *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 activity or as silent.*

Semantics. The semantics of Petri nets is given in term of *firing sequences*. Given a node $x \in P \cup T$, we define its pre-set ${}^\bullet x \stackrel{\text{def}}{=} \{y \in P \cup T \mid (y, x) \in F\}$ and its post-set $x^\bullet \stackrel{\text{def}}{=} \{y \in P \cup T \mid (x, y) \in F\}$. A marking is an assignment of a non-negative integer to each place. A transition t is *enabled* in a marking m when all places in ${}^\bullet t$ are marked. When a transition t is enabled, it can *fire* by removing a token from each place in ${}^\bullet t$ and putting a token to each place in $t^\bullet$. 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 $m[t_1 \dots t_n \rangle m'$. The set of reachable markings from m_0 is denoted by $[m_0\rangle$. A Petri net is *k-bounded* if no marking in $[m_0\rangle$ assigns more than k tokens to any place. A Petri net is *safe* if it is 1-bounded. A *full run* of a Petri net N is a firing sequence $m_0[t_1 \dots t_n \rangle m_f$ from the initial

marking m_0 to the final marking m_f. A Petri net is *easy sound* [2] if it has at least one full run, i.e. m_f is reachable from m_0.

In this paper we assume safe and easy sound Petri nets.

Definition 3 (Alignments). *Given a log trace $\sigma = \langle \sigma_1, \ldots, \sigma_m \rangle \in L$ of alphabet Σ, and a process model $N = \langle P, T, F, m_0, m_f, \Sigma, \lambda \rangle$, an alignment of σ with N is a finite sequence $\varphi = \langle (\sigma'_1, u'_1), \ldots, (\sigma'_p, u'_p) \rangle$ of* moves *such that:*

- *each move is either: a synchronous move $(a, t) \in \Sigma \times T$ with $a = \lambda(t)$, a log move $(a, \gg)$ (where $\gg$ is a special 'skip' symbol), or a model move $(\gg, t)$,*
- *dropping the $\gg$ symbols from the left projection $\langle \sigma'_1, \ldots, \sigma'_p \rangle$ of φ, yields σ;*
- *dropping the $\gg$ symbols from the right projection $\langle u'_1, \ldots, u'_p \rangle$ of φ, yields a full run u of N.*

Example 1. Figure 1 shows a process model (Fig. 1a) and a log trace (Fig. 1b). An alignment of N and σ is $\varphi = \langle (\gg, a), (b, b), (c, c) \rangle$.

In order to compare the quality of alignment quality, we define a cost function which penalizes log moves and model moves.

Definition 4 (Classical Alignment Cost Function, Optimal Alignments). *For every alignment φ between a model N and a log trace σ, the* Classical Alignment Cost Function $\mathcal{C}$ *assigns a cost* 0 *to synchronous moves and a cost* 1 *to log moves or model moves. The cost of an alignment is the sum of the costs of its moves. An alignment between a model N and a log trace σ is* optimal *if it minimizes the cost.*

Example 2. The alignment $\varphi = \langle (\gg, a), (b, b), (c, c) \rangle$ between N of Fig. 1a and σ of Fig. 1b costs 1 and is optimal. We recognize the Petri net transformation of σ which are the bottom places and the yellow transitions. The synchronous moves are drawn in green. From the initial state, possible moves are $(\gg, b)$, $(\gg, a)$, (b, b) or $(b, \gg)$.

The optimal alignment cost given with the classical alignment cost function $\mathcal{C}$ gives the Levenshtein edit distance between a run of M and the trace σ.

Definition 5 (Levenshtein Edit distance). *The* Levenshtein 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 insertions of a letter in words,*

$$\begin{cases} \mathcal{L}(\langle\rangle, \langle\rangle) = 0 \\ \mathcal{L}(u, \langle\rangle) = |u| \\ \mathcal{L}(\langle\rangle, v) = |v| \\ \mathcal{L}(a.u', b.v') = \mathcal{L}(u', v') & \text{if } (a = b) \\ \mathcal{L}(a.u', b.v') = \min \begin{cases} \mathcal{L}(a.u', v) + 1, \\ \mathcal{L}(u, b.v') + 1 \end{cases} & \text{otherwise.} \end{cases}$$

The main methods of the literature to compute optimal alignments are Dijkstra-based algorithms which often implies the construction of the *Synchronous Product* between the given process model and a sequential Petri net representing the log trace [1].

Definition 6 (Synchronous Product for Alignments). *For a process model* $N = \langle P, T, F, m_0, m_f, \Sigma, \lambda \rangle$ *and a log trace* $\sigma = \langle \sigma_1, \ldots, \sigma_m \rangle \in \Sigma^*$, *the Synchronous Product used for computing alignments is the Petri net* $SN = \langle P_{SN}, T_{SN}, F_{SN}, m_{SN_0}, m_{SN_f}, (\Sigma \cup \{\gg\})^2, \lambda_{SN} \rangle$ *defined as:*

- $N_\sigma = \langle P_\sigma, T_\sigma, F_\sigma, m_{\sigma 0}, m_{\sigma f}, \Sigma, \lambda_\sigma \rangle$ *is a translation of* σ *to a sequential Petri net with:* $P_\sigma = \{P_{\sigma_0}, \ldots P_{\sigma_m}\}$, $T_\sigma = \{t_{\sigma_i} = \lambda_\sigma(\sigma_i) \mid i \in \{1, \ldots, m\}\}$, $F_\sigma = \{(P_{\sigma_{i-1}}, t_{\sigma_i}), (t_{\sigma_i}, P_{\sigma_i}) \mid i \in \{1, \ldots, m\}\}$, $m_{\sigma_0} = \{P_{\sigma 0} : 1\}$, $m_{\sigma f} = \{P_{\sigma_m} : 1\}$,
- $P_{SN} = P \cup P_\sigma$
- $T_{SN} = T^{\gg} \cup T_\sigma^{\gg} \cup T_S$, *where* $T^{\gg} = \{(\gg, t) \mid t \in T\}$ *represents the model moves,* $T_\sigma^{\gg} = \{(t, \gg) \mid t \in T_\sigma\}$ *represents the log moves,* $T_S = \{(t_1, t_2) \mid t_1 \in T, t_2 \in T_\sigma$ *and* $\lambda(t_1) = \lambda_\sigma(t_2)\}$ *represents the synchronous moves,*
- $F_{SN} = F \cup F\sigma \cup \{(P_i, t_i) \mid t_i = (t_1, t_2) \in T_{SN},\ t_1 \neq \gg,\ t_2 \neq \gg,\ P_i \in {}^\bullet t_1 \cap {}^\bullet t_2\} \cup \{(t_i, P_i) \mid t_i = (t_1, t_2) \in T_{SN},\ t_1 \neq \gg,\ t_2 \neq \gg,\ P_i \in t_1{}^\bullet \cap t_2{}^\bullet\}$
- $m_{SN_0} = m_0 \cup m_{\sigma_0}$,
- $m_{SN_f} = m_f \cup m_{\sigma_f}$,
- λ_{SN} *maps every* $t \in T_{SN}$ *to its move.*

Example 3. Figure 1 shows the synchronous product for alignments of the process model N given in Fig. 1a and the log trace σ of Fig. 1b.

The Dijkstra-based algorithm for finding optimal alignments, explores the reachability graph of the synchronous product of Definition 6. Weights are given by the transitions fired to reach the markings, according to the type of move that they represent. The best firing sequences found for reaching a marking is the less costly one. The algorithm that we present in Sect. 5 is an adaptation of this classical Dijkstra-based algorithm for alignments. As we are using easy-sound Petri nets as process models, the Synchronous Products for Alignments are easy-sound which implies termination of the Dijkstra algorithm with the condition to reach the final marking m_{SN_f} [28].

4 Discounted Cost Function and Properties

The classical alignment cost function corresponds to Levenshtein edit distance between a run of a process model and a log trace, where additions and deletions represent model and log moves. In this section, we introduce the *Discounted Edit Distance* and its impact when using it as alignment cost function.

The idea of this Discounted Edit Distance is to penalize more insertions and deletions when they occur at the beginning of the strings, and less when they occur later.

Definition 7 (Discounted Edit Distance). *We define the* Discounted Edit Distance *between two strings u and v (of length $|u|$ and $|v|$ respectively) with discount parameter $\theta \geq 1$ by $\mathcal{D}_\theta(u,v) \stackrel{\text{def}}{=} \mathcal{D}^0_\theta(u,v)$ where:*

$$\begin{cases} \mathcal{D}^k_\theta(\langle\rangle,\langle\rangle) = 0 \\ \mathcal{D}^k_\theta(\langle\rangle, b.v') = \mathcal{D}^{k+1}_\theta(\langle\rangle, v') + \theta^{-k} \\ \mathcal{D}^k_\theta(a.u',\langle\rangle) = \mathcal{D}^{k+1}_\theta(u',\langle\rangle) + \theta^{-k} \\ \mathcal{D}^k_\theta(a.u', b.v') = \mathcal{D}^{k+2}_\theta(u', v') & \text{if } (a = b) \\ \mathcal{D}^k_\theta(a.u', b.v') = \min \begin{cases} \mathcal{D}^{k+1}_\theta(u', v) + \theta^{-k} \\ \mathcal{D}^{k+1}_\theta(u, v') + \theta^{-k} \end{cases} & \text{otherwise.} \end{cases}$$

thus allowing equation $((u_i = v_j))$ for free and insertions and deletions at cost θ^{-k} where k refers to the position where the edit occurs.

Lemma 1. *For $\theta = 1$, the Discounted Edit Distance corresponds to the Levenshtein distance.*

Proof. With $\theta = 1$, we have $\theta^{-k} = 1$ for any k and we obtain Definition 5 from Definition 7. □

In practice, relevant values for the discount parameter θ are slightly larger than 1. For $\theta \geq 2$, the discount is already very severe since an edit at position k costs more than the sum of all the following edits.

Lemma 2. *With the Discounted Edit Distance, for $\theta \geq 2$, an edit at position k costs more than the sum of all the following edits.*

Proof. For u and v, two words, let k be the position of a non-free cost in $\mathcal{D}_\theta(u,v)$. We note its cost $c(k) = \theta^{-k}$.

The next edits can occur at positions $j \in \{k+1, \ldots, n\}$ where, in the worst case, $n = |u| + |v|$. We write $S(j,n)$ the sum of costs. The maximal value of this sum appears when only non-free edits are used by the discounted edit distance:

$$S(k,n) = \sum_{j=k+1}^{n} c(j) = \theta^{-(k+1)} + \theta^{-(k+2)} + \cdots + \theta^{-n} = \frac{\theta^{-k} - \theta^{-n}}{\theta - 1}$$

Hence, $c(k) = \theta^{-k} > S(k,n)$ for $\theta \in [2, \infty[$. Otherwise, in the best case, there is no edit after position k and the cost of the edit at position k is higher than a null sum. □

Example 4. Let $u = \langle x, a, b\rangle$ and $v = \langle a, y, b\rangle$. The discounted edit distance between u and v is $\mathcal{D}_\theta(u,v) = \theta^{-1} + \theta^{-3}$. If $\theta = 1$, the distance equals to 2 and is the Levenshtein edit distance where deleting x costs 1 and adding y costs 1.

Similarly to the Levenshtein edit distance, the Discounted Edit Distance can be applied to alignments.

Definition 8 (Discounted Cost Function for Alignments). *For an alignment* $\varphi = \langle (\sigma'_1, u'_1), \ldots, (\sigma'_p, u'_p) \rangle$ *between a process model* N *and a log trace* σ*, the* Discounted Cost Function for Alignments *assigns a cost 0 to every synchronous move and* θ^{-k} *to every pair* (σ'_k, u'_k) *that is either a log move or a model move, where* $k \in \{1, \ldots, p\}$*,* $p \in \mathbb{N}$ *is the length of the alignment, and* $\theta \geq 1$ *is the discount parameter.*

For $\theta = 1$, the Discounted Cost Function for Alignment is equivalent to the Classical Alignment Cost Function. However when $\theta > 1$, the costs of moves are dynamic and depend on the number of previous moves of the alignment.

Example 5. For the alignment $\varphi = \langle (\gg, a), (b, b), (c, c) \rangle$, presented in the first example, the cost of $(\gg, a)$ is θ^{-1} because it is the first move of the alignment. For $\varphi' = \langle (b, b), (\gg, d), (\gg, e), (\gg, \tau) \rangle$ which is certainly not an optimal alignment but still an alignment of $\sigma = \langle b, c \rangle$ and the Petri net N of Fig. 1a, the cost of $(\gg, e)$ is θ^{-3}.

Lemma 3. *For* $\theta > 1$*, a non-free move* t *of position* j*, any move of position* $k > j$ *costs less than* t*.*

Proof. Any function $f : k \rightarrow \theta^{-k}$ where $\theta > 1$ is strictly decreasing. Then for $j < k$, we have $\theta^{-j} > \theta^{-k}$. □

As a consequence, algorithms for computing optimal discounted alignments will tend to align in priority the prefixes of the log traces. Suffixes are less costly. From Lemma 2, when the discount parameter is $\theta = 2$, a non-free move of position j is more costly than the sum of all the next costs.

Example 6. In Example 2, we saw that the optimal alignment by using the classical alignment cost function between $\sigma = \langle b, c \rangle$ and the model N of Fig. 1a is $\varphi = \langle (\gg, a), (b, b), (c, c) \rangle$ of cost 1. However, by using the discounted cost function with $\theta = 2$, optimal alignments are $\varphi' = \langle (b, b), (\gg, d), (\gg, e), (\gg, \tau) \rangle$ and $\varphi'' = \langle (b, b), (\gg, e), (\gg, d), (\gg, \tau) \rangle$ of cost $2^{-2} + 2^{-3}$, where $(\gg, \tau)$ is a free model move. This is due to the discounted cost function which penalizes the model move $(\gg, a)$ at first position.

5 Using the Discounted Cost Function in an A*-Based Algorithm for Discounted Alignments

To compute alignments by using the discounted cost function, we present an A*-based algorithm which assigns weights to the explored states according to the discounted cost function for alignment. Let θ be the discount parameter. Then, to a state reached by a move t occurring in position i, will be assigned the weight of its predecessor, increased by the cost

$$h(t, i, \theta) \stackrel{\text{def}}{=} (0 \text{ if } t \text{ is a synchronous move}, \theta^{-i} \text{ otherwise}).$$

As a result of Lemma 3, this heuristic aims at aligning prefix first more than suffixes.

The function h, based on the discounted cost function, is easily incorporated in the state-of-the-art algorithms for computing alignments. We present two versions of the incorporation, one by using the synchronous product of the process model and the log trace, and another one that avoids the computation of the product by exploring the process model along with the trace.

5.1 Algorithm for Computing Optimal Discounted Alignments

Our algorithm Algorithm 1, noted $A^*SP_{\mathcal{D}=\theta}$, is inspired from [6]. It proceeds by exploring the state space of the synchronous product of the process model with a sequential Petri net representing the log trace as defined in Definition 6. An optimal alignment corresponds to the shortest path between the initial marking to the final marking of the synchronous product. For this purpose, our A* algorithm maintains a priority queue Q of prefixes of runs, implemented as a heap of tuples $\langle \gamma, m, d \rangle$, for a prefix γ reaching marking m at cost d, such that the tuple with minimal cost d pops first. Line 1 initializes the heap with the empty prefix reaching the initial marking at cost 0, i.e. $\langle \langle\rangle, m_0, 0 \rangle$.

Algorithm 1: Computation of Optimal Alignments by using the Discounted Edit Distance ($A^*SP_{\mathcal{D}=\theta}$)

Input : $SP = ((P, T, F, m_0, m_f, (\Sigma \cup \{\gg\})^2, \lambda))$: synchronous product, θ: discount parameter

1 $Q \leftarrow \{\langle \langle\rangle, m_0, 0 \rangle\}$ ▷ Heap of open states ordered by distance
2 $A \leftarrow \emptyset$ ▷ Initialize closed set
3 **while** $Q \neq \emptyset$ ▷ While not all reachable states visited
4 **do**
5 $\quad \langle \gamma, m, d \rangle \leftarrow Q.pop()$ ▷ Get next state minimizing d
6 $\quad$ **if** $m = m_f$ **then**
$\quad\quad$ **Return**: $\langle d, \gamma \rangle$
7 $\quad A \leftarrow A \cup \{\langle m, |\gamma| \rangle\}$ ▷ Add state to closed set
8 $\quad$ **for** $t \in T$ *with* $m[t\rangle m'$ **do**
9 $\quad\quad \gamma' \leftarrow \gamma \bullet t$ ▷ Get new prefix
10 $\quad\quad$ **if** $\langle m', |\gamma'| \rangle \notin A$ ▷ Reaching a not yet visited state
11 $\quad\quad$ **then**
12 $\quad\quad\quad d' \leftarrow d + h(t, |\gamma'|, \theta)$ ▷ Compute cost of γ'
13 $\quad\quad\quad Q \leftarrow Q.insert(\langle \gamma', m', d' \rangle)$ ▷ Insert new prefix in heap
$\quad$ **Raise** : m_f is not reachable

Line 3 starts a *while* loop that ends only when the final marking is reached (line 6) or when the priority queue is empty (line 3). Line 9 gets the next firing transitions of the synchronous product. Some transitions correspond to the log and model moves and are costly. The other transitions are the synchronous moves and are free, like in the original algorithm.

Our discounted cost function h appears on line 12 and determines the cost of the new prefix. Line 13 adds the new discovered state in the priority queue with its prefix γ' and it cost d' for reaching m'.

When the algorithm reaches the final marking, line 6, the while loop is broken and the algorithm returns the sequence of firing transitions to reach the final marking. In fact, this sequence of transitions gives the sequence of moves of the alignment.

Role of Length in States. In the classical version of alignment computation, the closed set contains the markings only. However, the length of the current alignment plays an important role in the discounted cost function (line 8). Indeed, the first visit of a marking might not be the optimal one, as it is the case in the classical version of alignments. A same marking m can be reached with different firing sequences of moves of different lengths. The first path that gives the first visit of the marking m is the shortest one. Let's call this short path γ_{short}, and γ_{long} a longer path from the initial marking to this marking m. We have $|\gamma_{long}| > |\gamma_{short}|$. However, γ_{short} might contain future costly moves to reach the final marking. If those moves are of position lower than $|\gamma_{long}|$, it questions the optimality of γ_{short}. We give an example of this situation below.

Example 7. Let's suppose that silent transition labelled by τ costs for this example. To reach marking $m = \{p4 : 1, p_{\alpha_2} : 1\}$, the algorithm can play $\gamma = \langle(\gg, a), (\gg, b), (\gg, \tau), (b, b)\rangle$ whose cost is $\theta^{-1} + \theta^{-2} + \theta^{-3}$. This firing sequence costs as much as $\gamma' = \langle(\gg, a), (\gg, b), (b, \gg)\rangle$ which reaches the same marking m. However we notice that γ has a synchronisation at position 4 but we don't know yet what appears at position 4 for γ'. Then both paths should be kept.

Note that we tackled the problem of optimality of the alignment with the discounted cost function. For $\theta > 1$, this optimality does not correspond to the optimal classical alignment.

Comparison to Classical Alignments. Due to the discount parameter θ in the discounted cost function, our heuristic prioritizes the alignment of the beginning of the log trace. In the algorithm, this difference with the classical alignment algorithm appears in line 11 of Algorithm 1, where the markings that minimize the cost are much more different with the discounted cost function than by using the classical cost function for alignments. Indeed, when costs are all equivalent, many paths compete in the search for the optimal alignment. However, with very different costs, the number of paths with similar costs is low, thus reducing the search space.

Example 8. For the example of Fig. 1, there is a first choice between a and b. For large θ, the decision is quickly given thus disabling testing the depth of the other paths. For instance, with $\theta = 2$, the log sequences of type $\langle a, b, c\rangle, \langle a, b, \tau, b, c\rangle, \langle a, b, \tau, b, \tau, b, c\rangle$ won't be explored because they cannot have a better discounted cost.

5.2 A Heuristic for Reducing the Search Space of the Algorithm

The search space of $A^*SP_{\mathcal{D}=\theta}$ is large and even larger than the Dijsktra-based algorithm for alignments due to the incorporation of the lengths of the runs that reach the same marking. To reduce it, we come back to the classical closed set that contain the markings only. Every $\langle m, |\gamma| \rangle$ of the closed set A is reduced to m (like in [6]).

With this reduction, only the first paths that reaches the marking are used. When several concurrent firing sequences of equal cost exist, line 5 picks one as the optimal path and line 8 classifies the marking in A. Then the other firing sequences of equal cost for this marking are not considered anymore (line 10).

Example 9. For the marking $m = \{p4 : 1, p_{\alpha_2} : 1\}$ of the synchronous product given in Fig. 1c, two firing sequences compete for the minimization of the cost (in case that silent transition costs). Indeed both $\gamma = \langle (\gg, a), (\gg, b), (\gg, \tau), (b, b) \rangle$ and $\gamma' = \langle (\gg, a), (\gg, b), (b, \gg) \rangle$ have a cost of $\theta^{-1} + \theta^{-2} + \theta^{-3}$ and reach the marking m. Hence, when using markings only in the states, the algorithm picks one of the sequence as the optimal one and adds m in the closed set. Later, it does not consider the other firing sequence. However, we saw in Example 7 that γ' is better than γ by using the discounted cost function but it can go to the hatch in the reduced version.

With this reduction of the search space, the modified algorithm is not guaranteed any more to return the optimal discounted alignments, but the gain in runtime is extremely significant. Moreover, in practice, the loss of quality is very limited: we observed that the alignments found by the modified algorithm have very similar discounted cost than the optimal discounted alignments.

Process Model Exploration Along with Trace Exploration (Noted $A^*PT_{\mathcal{D}=\theta}$). In order to speed up the exploration, the alignment algorithm can simulate the synchronous product without explicitly constructing it. The synchronous product allows to easily play the moves of alignment. However, those moves can be found by exploring the process model and the trace separately. By comparing the next activity of the process model, given by the semantic of the net, and the next activity of the trace, we obtain the type of move. For instance, at the initialization, one possible next activity of N of Fig. 1a is b and the first activity in σ is also b. Then, we can move forward with a synchronous move, like in the synchronous product but without constructing the corresponding transition of the move. Then the m in the algorithm (for the marking of the synchronous product) is replaced by a pair $\langle m, p \rangle$ where m is the marking of the process model and p the position in the trace. Any marking of $A^*SP_{\mathcal{D}=\theta}$ can be given into a couple $\langle m, p \rangle$ for $A^*PT_{\mathcal{D}=\theta}$. For instance, marking $\{p4 : 1, p_{\alpha_2} : 1\}$ of the synchronous product given in Fig. 1c corresponds to $\langle \{p4 : 1\}, 1 \rangle$ where $\{p4 : 1\}$ is the marking in N and 1 the position in σ. The final marking becomes $\langle m_f, |\sigma| \rangle$ where the trace has been read and the process model reaches its final places.

6 Experiments: Discounted Alignments as a Heuristic for Approximating Classical Alignments

The algorithms presented in the previous section for computing alignment have been implemented in a branch of pm4py[1]. In this section we present general comparisons of quality and runtimes between them and existing methods, where the runtime reflects the search space. We also stress the impact of the discounted parameter by zooming on particular cases.

6.1 Comparison with Respect to Baselines

Inputs. We played the algorithms for both artificial and real-life logs and the corresponding models. Artificial set is taken from [18] and contains large models. For real-life logs, we used data given in the Business Process Intelligence Challenges from 2012 to 2020. We mined the process models of those logs with methods of the literature[2]. First, we applied a preprocessing method introduced by [8] to extract good prototypes of the logs[3]. This prepossessing step allows us to obtain not perfectly fitting models when using miners, interesting for alignments comparison. In fact, the method aims at finding more precise models. Then, we launched two different discovery algorithms on the found prototypes: the inductive miner [11] and the split miner 2.0 [3]. As the latter tool gives BPMN models, we use ProM plugins to transform them into Petri nets.

We computed the alignments on *variants* only, i.e., unique sequences of activities. This choice of using variants only allows to correctly compare the method's runtimes and prevents the situation where one method reduces the log to variants and not another one. Indeed alignment of log sequences of the same variant

Log	#variants	$\lvert\Sigma\rvert$	$\max_{\sigma \in Log} len(\sigma)$	Model	$\lvert T\rvert$	$\lvert P\rvert$	$\lvert F\rvert$
L1	453	36	37	M1	39	40	92
L2	500	32	52	M2	34	34	80
L3	462	109	217	M3	123	108	276
L4	496	44	176	M4	52	36	106
L5	500	32	71	M5	33	35	78

(a) Artificial Logs and Models

Log	#variants	$\lvert\Sigma\rvert$	$\max_{\sigma \in Log} len(\sigma)$	Model Miner	$\lvert T\rvert$	$\lvert P\rvert$	$\lvert F\rvert$
BPI2012	4366	24	175	IM	34	24	68
				SM	30	23	60
$BPI2018_{pa}$	3832	24	100	IM	22	24	60
				SM	20	15	40
BPI2019	11973	42	990	IM	18	13	38
				SM	13	10	26
$BPI2020_{dd}$	99	17	24	IM	15	11	32
				SM	14	9	28
$BPI2020_{rp}$	89	19	20	IM	31	26	74
				SM	23	12	46

(b) Real-life Logs and Models

Fig. 2. Input Description, where Σ is the alphabet of activities in the log, T the set of transitions of the model, P the set of places of the model and F the flow relations between the nodes in the model.

[1] Currently available at https://github.com/BoltMaud/pm4py-core.

[2] Available at https://github.com/BoltMaud/A-Discounted-Cost-Function-for-Fast-Alignments-of-Business-Processes-Sources.

[3] Prototype Selection plugin of ProM software with default settings.

is equivalent but this optimization can be used for any algorithm. Figure 2 gives an overview of the inputs.

Comparison. We compare our alignment results to the four current methods of the state-of-the-art implemented in pm4py which are: – the Dijkstra search on the Synchronous Product without heuristic (DSP) [1], – a Disjkstra that consumes less memory by using a similar idea of our second algorithm (DLM), – an A*-based algorithm on the state-space of the synchronous product that incorporates an heuristic on reaching the final marking (A^*SP_{mf}) [26] and – its less-memory version (A^*LM_{mf}). To compare the runtimes, we exclude log-based implementations and used the trace-based version to avoid the use of the parallelism between variants that can be added to any version at the case level. We recall notation of our methods $A^*SP_{\mathcal{D}=\theta}$, for the version that uses the synchronous product, and $A^*PT_{\mathcal{D}=\theta}$, for the second one that explores only the process model and the trace.

Results. The *quality* of an alignment found by a heuristic method, is defined as the ratio (in %) between the classical cost (number of model or log moves) of the optimal alignment (given by the DSP method) and the classical cost of the alignment found by the method. In Fig. 3a we give the quality of each method.

Similarly, in Fig. 3b, each line shows the sum of the runtimes of alignment computations by a method, expressed in percentage of the runtime of the DSP method. The runtime reflects the space of search. The box charts have wide range because they summarize the results of all the alignments which are very different (depending on both the model and the log involved).

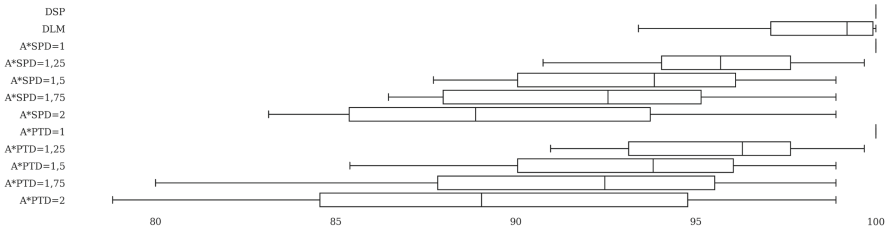
We see in Fig. 3b that the runtime of the DLM algorithm is 20% of the runtime of the DSP method. For our heuristic $A^*PTD = 2$, the average runtime is around 10% of the reference method DSP (which corresponds to a gain of 90% of runtime, the result of a large reduction of the search space), for an average quality between 90 and 85% of the optimal alignments.

We did not represent in the charts the runtimes for methods A^*SP_{mf} and A^*LM_{mf} (implemented in pm4py) since they are much higher than the others: A^*SP_{mf} ran up to 30 times longer than the DSP and A^*LM_{mf} up to 7 times longer. We invite the reader to find the results and scripts on github[4].

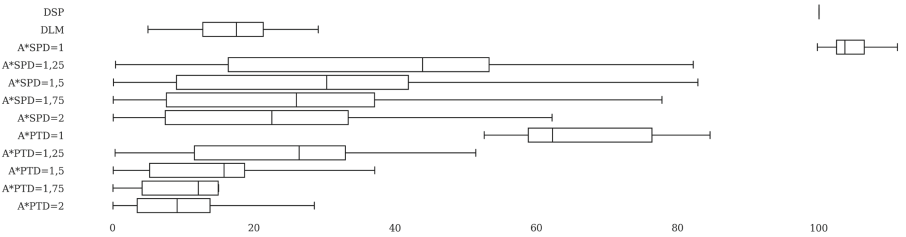
6.2 Influence of the Discount Parameter θ on the Quality and Runtime

Figure 3a shows that the quality decreases when the parameter θ of the discounted cost function raises. However the gain in term of search space is high when $\theta > 1$ (depicted in practice with a gain in runtime). The output of those experiments is the correlation between the parameter θ and the compromise

[4] Available at https://github.com/BoltMaud/A-Discounted-Cost-Function-for-Fast-Alignments-of-Business-Processes-Sources.



(a) Quality of the alignments obtained by different methods. Quality is defined as the ratio (in %) between the classical cost (number of model or log moves) of the optimal alignment and the classical cost of the alignment found by the method. The first line is the baseline and present the optimal approach which has no lost in quality, i.e., we observe 100% of quality. In general, one can see that the loss of quality with our heuristics is rather limited.



(b) Runtime (in % of the runtime of the DSP Algorithm). A percentage lower than 100 corresponds to a gain of runtime compared to the DSP method.

Fig. 3. Comparison of quality and runtimes of different methods.

between quality and runtime of alignments. For $\theta = 1$, one gets exact alignments but runtimes are slow because the search space is large. For higher θ one can extract very fast alignments but the quality is reduced. In practice, we recommend to try θ slightly higher than 1 and not larger than 2 (which is already a very severe discount factor). Values around 1.1–1.5 should give good results.

On another hand, we want to raise awareness on the method $A^*PTD = 1$ which corresponds to optimal alignment without the construction of the synchronous product. The method gives exact alignments for reduced runtimes because it disables the construction of the synchronous product. Method DLM also does not construct the synchronous product but we see in Fig. 3a that there is a lost of quality.

Additive Comparisons. The omission of ProM and other tool results in the previous section is due to the differences between the output formats which made difficult the comparison of quality and runtimes. However, in this section we zoom in particular cases, i.e., by running a log sequence only, thus making human interpretation possible. We add PNR the results given by the PNetReplayer package of [2] in ProM and REC_{ilp} the results given by [19].

Method	DSP	DLM	A^*SP_{mf}	A^*LM_{mf}	PNR	REC_{ilp}	A^*SPD					A^*PTD				
							1	1.25	1.5	1.75	2	1	1.25	1.5	1.75	2
Number of Asynchronous Moves	14	14	14	14	14	–	14	14	14	15	15	14	14	14	15	15
Runtime (s)	0.14	0.62	0.73	0.02	0.01	–	0.10	0.10	0.04	0.04	0.02	0.13	0.12	0.06	0.04	0.04

(a) BPI2020$_{rp}$

Method	DSP	DLM	A^*SP_{mf}	A^*LM_{mf}	PNR	REC_{ilp}	A^*SPD					A^*PTD				
							1	1.25	1.5	1.75	2	1	1.25	1.5	1.75	2
Number of Asynchronous Moves	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
Runtime (s)	47.78	628.75	3385.59	1.55	0.06	1.38	13.28	0.02	0.01	0.01	0.01	43.41	0.03	0.02	0.02	0.01

(b) M5

Method	DSP	DLM	A^*SP_{mf}	A^*LM_{mf}	PNR	REC_{ilp}	A^*SPD					A^*PTD				
							1	1.25	1.5	1.75	2	1	1.25	1.5	1.75	2
Number of Asynchronous Moves	36	38	36	36	36	–	36	886	2137	1898	4265	36	161	146	330	338
Runtime (s)	6.19	2121.98	16011.60	0.76	1.94	–	4.38	0.07	0.14	0.13	0.45	8.24	0.10	0.12	0.07	0.12

(c) M3

Fig. 4. Particular alignments that draws advantages and disadvantages of our methods.

Specific Inputs. We choose 3 models and traces that have particular characteristics. First, we run the alignment between the first trace of BPI2020$_{rp}$ log and model IM because this couple (BPI2020$_{rp}$, IM) gives the least differences between the methods (Fig. 4a). The model has only 2 parallelism patterns and no loop. Then, we run the alignment of the first trace of $L5$ and model $M5$ where our method specifically works well (Fig. 4b). This model contains a concurrent pattern including 28 transitions and one loop. Finally we present an alignment of the fifth trace of L3 which is very long (215 activities) and its model which has many loops and many parallelism patterns (Fig. 4c). The latter aims at showing the weakness of our method.

Results. From the three tables of Fig. 4, we observe that the methods usually find exact alignments. This is not true for the last experiment given in Fig. 4c which highlights our weakness. This due to the size of the trace (215) that, for the high base of logarithm θ, the algorithms face a situation where all the markings have the same cost (θ^n where n is very large borders zero). At this point, we already advise the user to check the length of the traces to set the discounted parameter θ (or to tackle very small differences between costs with an implementation where more decimal are allowed).

Observe now that for Fig. 4a and b using high value of θ brings very fast result for nearly optimal alignment in Fig. 4a and optimal alignment in Fig. 4b. Moreover, this latter result even beats all the other methods including the ProM implementation (noted PNR). The particularity of model M5 is the large concurrency pattern that creates many paths of different behaviors. Most methods have to explore the different combinations created by the concurrency pattern. Our discounted function favors only the path that align at the beginning of this pattern and does not consider the other combinations of the activities.

The versions using less memory seem to be much less efficient sometimes even for less quality (see DLM for $M3$). The method REC_{ilp} worked only for the second model. $M3$ is too large for the Gurobi open source version and format of model 2020_{rp} is not accepted by the tool.

Comparison with the Token Replay Approach. Last but not least we give a comparison of runtimes between our algorithms for computing alignment and the token replay approach given in [4] because it also computes an approximation of conformance (more precisely fitness) of process models for a given log. For those experiments, we set our method with $\theta = 2$. We observe in Table 1 that our second algorithm gives faster results in most times. The token replay is however much faster for $\text{BPI2018}_p a$. We plan to compare fitness approximation in future works.

Table 1. Runtime Comparison (in seconds) between the Algorithms for computing Discounted Alignments and the Token Replay Method given in [4] run on a un on a MacBook air 2017 model with a 1.8 GHz Intel ® Core™ i5 CPU and 8 GB RAM.

Method	BPI2012		BPI2018_{pa}		BPI2019		BPI2020_{dd}		BPI2020_{rp}	
	IM	SM	IM	SM	IM	SM	IM	SM	IM	SM
A^*SP	78.18	69.39	493.96	43.92	143.25	103.99	0.43	0.23	1.08	0.24
A^*PT	25.03	19.71	419.37	11.15	42.14	26.61	0.14	0.09	0.70	0.09
Token replay	35.41	36.86	36.11	31.01	45.99	49.40	0.20	0.19	0.22	0.18

7 Conclusion

In this paper, we present a novel cost function for alignments. By using the position of the moves, our discounted cost function penalizes deviations of business processes that appear at early stage of the process execution. While the first aim is to align prefixes first, we nicely see that the proposed discounted cost function gives a heuristic for classical alignments. We implemented two versions of an A*-based algorithm that incorporate this heuristic and we experimented with artificial and real-life logs. The outputs of the experiments clearly show that the lost of quality, in term of log and model moves, is correlated to the gain of runtime, the result of the reduction of the search space. This is due to the parameter θ of our discounted cost function that forces prefix-first alignments. The compromise between quality and runtime can easily be set by using this parameter.

As future work, we suggest to combine the discounted cost function with other heuristics used for alignments. Also, the idea of using a discounted cost for alignments may be more or less relevant depending on the application that one is targeting. Among the multiple applications of alignments in conformance checking, some may be more or less resilient to the little loss of quality that we

accept when using heuristics. In some settings, even, the alignments obtained for our discounted cost function may be more relevant than classical alignments, typically when the application justifies to penalize early deviations more than late ones.

Another interesting research line would be to use a machine learning approach (like it was done in [17] for the case of predicting the best process discovery technique), for learning the best parameter setting (mainly, the θ value used for discounting) as a previous step to our alignment technique.

References

1. Adriansyah, A.: Aligning observed and modeled behavior, Ph.D. thesis. Technische Universiteit Eindhoven (2014)
2. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Memory-efficient alignment of observed and modeled behavior. BPM Center Report, vol. 3, pp. 1–44 (2013)
3. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Polyvyanyy, A.: Split miner: automated discovery of accurate and simple business process models from event logs. Knowl. Inf. Syst. **59**(2), 251–284 (2019)
4. Berti, A., van der Aalst, W.M.P.: Reviving token-based replay: increasing speed while improving diagnostics. In: ATAED@ Petri Nets/ACSD, pp. 87–103 (2019)
5. Bloemen, V., van Zelst, S.J., van der Aalst, W.M.P., van Dongen, B.F., van de Pol, J.: Maximizing synchronization for aligning observed and modelled behaviour. In: Proceedings of the 16th International Conference on Business Process Management, BPM 2018, Sydney, NSW, Australia, 9–14 September 2018, pp. 233–249 (2018)
6. Carmona, J., van Dongen, B.F., Solti, A., Weidlich, M.: Conformance Checking - Relating Processes and Models. Springer, Switzerland (2018). https://doi.org/10.1007/978-3-319-99414-7
7. de Leoni, M., Marrella, A.: Aligning real process executions and prescriptive process models through automated planning. Exp. Syst. Appl. **82**, 162–183 (2017)
8. Sani, M.F., Boltenhagen, M., van der Aalst, W.: Prototype selection based on clustering and conformance metrics for model discovery. arXiv (2019)
9. Lee, W.L.J., Verbeek, H.M.W., Munoz-Gama, J., van der Aalst, W.M.P., Sepúlveda, M.: Recomposing conformance: closing the circle on decomposed alignment-based conformance checking in process mining. Inf. Sci. **466**, 55–91 (2018)
10. Leemans, S.J.J., Fahland, D., Wil, M.P.: van der Aalst. Scalable process discovery and conformance checking. Softw. Syst. Model. **17**(2), 599–631 (2018)
11. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Lohmann, N., Song, M., Wohed, P. (eds.) BPM 2013. LNBIP, vol. 171, pp. 66–78. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06257-0_6
12. Munoz-Gama, J., Carmona, J., van der Aalst, W.M.P.: Single-entry single-exit decomposed conformance checking. Inf. Syst. **46**, 102–122 (2014)
13. Murata, T.: Petri Nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–574 (1989)

14. Reijers, H.A., Liman Mansar, S.: Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics. Omega **33**(4), 283–306 (2005)
15. Reißner, D., Conforti, R., Dumas, M., Rosa, M.L., Armas-Cervantes, A.: Scalable conformance checking of business processes. In: OTM CoopIS, Rhodes, Greece, pp. 607–627 (2017)
16. Reißner, D., Armas-Cervantes, A., Conforti, R., Dumas, M., Fahland, D., La Rosa, M.: Scalable alignment of process models and event logs: an approach based on automata and s-componentsd. Inf. Syst. **94**, 101561 (2020)
17. Ribeiro, J., Carmona, J., Mısır, M., Sebag, M.: A recommender system for process discovery. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 67–83. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10172-9_5
18. Taymouri, F., Carmona, J.: Model and event log reductions to boost the computation of alignments. In: Ceravolo, P., Guetl, C., Rinderle-Ma, S. (eds.) SIMPDA 2016. LNBIP, vol. 307, pp. 1–21. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74161-1_1
19. Taymouri, F., Carmona, J.: 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, 18–22 September, pp. 197–214 (2016)
20. Taymouri, F., Carmona, J.: Model and event log reductions to boost the computation of alignments. In: Ceravolo, P., Guetl, C., Rinderle-Ma, S. (eds.) SIMPDA 2016. LNBIP, vol. 307, pp. 1–21. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74161-1_1
21. Taymouri, F., Carmona, J.: Computing alignments of well-formed process models using local search. ACM Trans. Softw. Eng. Methodol. **29**(3), 15:1–15:41 (2020)
22. van der Aalst, W.M.P.: Decomposing Petri Nets for process mining: a generic approach. Distrib. Parallel Databases **31**(4), 471–507 (2013)
23. van der Aalst, W.M.P.: Process Mining - Data Science in Action, 2nd edn. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
24. van der Aalst, W.M.P.: Re-engineering knock-out processes. Decis. Support Syst. **30**(4), 451–468 (2001)
25. van Zelst, S.J., Bolt, A., Hassani, M., van Dongen, B.F., Wil, M.P.: van der Aalst. Online conformance checking: relating event streams to process models using prefix-alignments. Int. J. Data Sci. Anal. **8**(3), 269–284 (2019)
26. van Zelst, S.J., Bolt, A., van Dongen, B.F.: Tuning alignment computation: an experimental evaluation. In: ATAED@ Petri Nets/ACSD, pp. 6–20 (2017)
27. Verbeek, H.M.W., van der Aalst, W.M.P.: Merging alignments for decomposed replay. In: Kordon, F., Moldt, D. (eds.) PETRI NETS 2016. LNCS, vol. 9698, pp. 219–239. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39086-4_14
28. Winskel, G.: Petri Nets, algebras, morphisms, and compositionality. Inf. Comput. **72**(3), 197–238 (1987)