Cache Enhanced Split-Point-Based Alignment Calculation

Tian Li
Chair of Process and Data Science
RWTH Aachen University
Aachen, Germany
tian.li@rwth-aachen.de

Sebastiaan J. van Zelst
Fraunhofer Institute for Applied Information Technology FIT, Germany
RWTH Aachen University
Sankt Augustin, Germany
sebastiaan.van.zelst@fit.fraunhofer.de

***Abstract*—The execution of (business) processes often deviates from their behavioral specification (e.g., captured in a BPMN model). Conformance checking techniques evaluate whether event logs, i.e., data records capturing process behavior, and process models conform to each other. As such, conformance checking techniques provide insights into the correctness of the process execution. Alignments are conformance checking artifacts used to compute conformance metrics and, particularly, diagnostics. Several alignment algorithms exist, yet most existing methods solve an underlying search problem in which one typically calculates a heuristic to guide the search. Recently, a promising novel search approach was presented that reduces the overall number of heuristics required to solve the alignment problem. This paper extends this approach by proposing a caching strategy that improves the overall search speed and efficiency. We conducted a large set of experiments, confirming that the overall search efficiency increases significantly due to our contribution.**

***Index Terms*—Process Mining, Conformance Checking, Alignments**

I. Introduction

Conformance checking [1] has received increasing attention in the field of *Process Mining* [2]. Conformance checking techniques measure the deviations between the executions of a process (recorded in an event log) and its normative behavior described by a behavioral specification (e.g., in BPMN [3]). For instance, an event log can be "replayed" on a given process model to uncover potential problems in the process execution. As such, the deviations exposed between the model and the process execution can, for example, facilitate better working instructions. Also, business process managers benefit by determining whether the process is executed as planned.

Alignments [4] are fundamental conformance checking artifacts, as they allow one to quantify how the event log can be replayed on the process model. In case an observed process execution is not described by the reference model, alignments quantify the problems of the execution in great detail. For example, alignments indicate whether the execution of an observed activity was obsolete or missing. The de facto strategy to compute alignments is *solving the shortest path problem*. In the search, typically, the A^* algorithm [5] is adopted, i.e., a search strategy that requires a heuristic to guide the search. The heuristic is computed by solving an (Integer) Linear Program [6] ((I)LP) for every state visited.

In [7], a new search approach is proposed that reduces the number of (I)LPs computed during the search. Although the algorithm outperforms other approaches, it frequently restarts from scratch, redundantly revisiting states in the state space multiple times. Hence, in this paper, we extend the approach presented in [7] by adding a caching strategy that allows us to reuse previously visited states in the search. We additionally devise a restart strategy, which regularly clears the cache.

We conducted a large set of experiments with several different real event logs and various process models. The results of our experiments show that our proposed caching strategy consistently improves the overall search efficiency by reducing the number of states visited during the search.

The remainder of the paper is structured as follows: In Section II and Section III, we discuss related work and introduce preliminary information, respectively. In Section IV, we present our main contribution. In Section V, we evaluate the approach with various data sets and compare the results with existing techniques. Finally, in Section VI, we conclude our work and suggest possible directions for future work.

II. Related work

In this section, we present related work in conformance checking. A general overview of the process mining field is out of this paper's scope. Hence, we refer to [2].

Token-based replay [8] is one of the early algorithms which laid the foundation for conformance checking. The approach replays traces in the event log on a given Petri net, and fitness is calculated based on missing and remaining tokens, together with generated and consumed tokens. However, the approach does not exploit a path in the model to map the recorded behavior to modeled behavior. As such, token-based replay techniques are not helpful for diagnostic purposes.

The concept of alignment-based conformance checking is introduced in [9], which provides a complete path for every trace in the process model and explicitly shows where deviations take place. The technique not only maps observed behavior in the trace to modeled behavior in the model but also finds a realizable behavioral execution sequence in the model. In [10], the use of a *synchronous product net* based on a given Petri net and a trace from an event log is proposed to compute alignments. The authors show that computing an alignment is

equivalent to solving the shortest path problem on the state space of the synchronous product net. As a solution method, in [10], the A^* algorithm [5] for shortest paths is proposed, combined with a heuristic based on the marking equation of Petri nets. To improve the scalability of this method, the authors in [4] propose different tuning and parametrization techniques, which show that efficiency gains can be achieved by not scheduling certain types of states during the search.

In [7], van Dongen proposed an alternative search strategy. Initially, an extended version of the marking equation is solved (using (I)LP). The solution *may correspond to a realizable path in the state space of the synchronous product net*. The algorithm traverses the state space by only considering states that are *described by the solution*. Using this strategy, the *search can get stuck*. When the search gets stuck, the algorithm finds the maximum event index of the trace (encapsulated in the synchronous product net) that has been explored and uses it to further refine the extended marking equation. In [11], the authors propose to derive structural or behavioral rules from the process model so that the initial version of the extended equation is more refined.

In this paper, we extend [7] by proposing a caching strategy that enhances the efficiency of the underlying search. Our approach can be combined with [11].

III. Background

In this section, we present basic concepts that support the general readability of this paper, including mathematical notation, event data, Petri nets and alignments.

A. Mathematical Concepts and Notation

We let $\mathbb{N}$ denote the set of natural numbers (including 0). A sequence $\sigma{=}\langle\sigma(1),\ldots,\sigma(n)\rangle$ of length $n{\in}\mathbb{N}$ over set X is a function $\sigma\colon\{1,\ldots,n\}{\to}X$. The set of all sequences over some set X is denoted as X^*. Given arbitrary $\sigma{\in}X^*$, $|\sigma|$ denotes the length of σ and we let $\#_\sigma(x){=}|\{i{\in}\{1,\ldots,|\sigma||\sigma(i){=}x\}|$. Further, given $\sigma{\in}X^*$ and $X'{\subseteq}X$, we let $\sigma|_{X'}$ be the restriction of σ to X', e.g., $\langle a,b,c,b,d\rangle|_{\{a,b\}}{=}\langle a,b,b\rangle$. The Parikh vector of $\sigma{\in}X^*$, where $X{=}\{x_1,x_2,\ldots,x_k\}$, is a column vector capturing the number of occurrences of a certain element in σ, i.e., $\vec{\sigma}{=}(\#_\sigma(x_1),\#_\sigma(x_2),\ldots,\#_\sigma(x_k))^\intercal$. Given a tuple $\mathbf{x}{=}(x_1,x_2,\ldots,x_n){\in}X_1{\times}X_2{\times}\ldots{\times}X_n$ of size $n{\in}\mathbb{N}$, we let $\pi_i(\mathbf{x}){=}x_i$ for $1{\leq}i{\leq}n$. For sequences of tuples, i.e., for $\sigma = \langle(x_1^1,x_2^1,\ldots x_n^1),(x_1^2,x_2^2,\ldots x_n^2),\ldots,(x_1^m,x_2^m,\ldots x_n^m)\rangle{\in}(X_1{\times}X_2{\times}\ldots{\times}X_n)^*$ (observe $|\sigma|{=}m$), we let $\pi_i^*{=}\langle x_i^1,x_i^2,\ldots x_i^m\rangle$ for $1{\leq}i{\leq}n$. A multiset is a collection of objects that allows for multiple occurrences of its elements. For instance, $m{=}[a^2,b]$ contains two a elements, one b element and zero c elements. The set of all possible multisets over X is written as $\mathcal{M}(X)$. Given $m{\in}\mathcal{M}(X)$, we let $\overline{m}{=}\{x{\in}X|m(x){>}0\}$.

B. Event Data

The information systems used in organizations, e.g., ERP and CRM systems, track the execution of (business) processes.

TABLE I: Example fragment of an event log.

Case-id	Activity	Time-stamp
...	...	...
484	receive return(a)	09/10/2021 10:27
485	receive return(a)	10/10/2021 10:13
485	get replacement(b)	11/10/2021 11:21
485	register return(d)	11/10/2021 11:21
485	inform progress to customers(e)	12/10/2021 10:51
484	inform progress to customers(e)	12/10/2021 11:14
485	send replacement(h)	12/10/2021 15:01
484	send repair(g)	12/10/2021 15:22
...	...	...

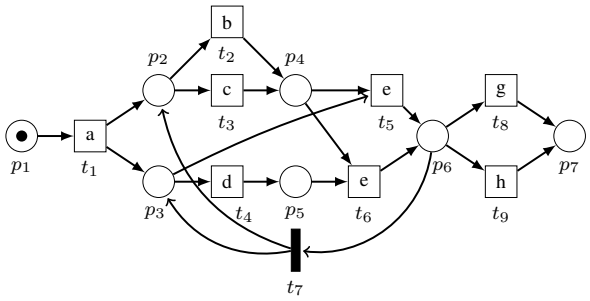


Fig. 1: Petri net N_1: A simplified product repair/replace process

As such, *event logs* can be extracted from these systems. Consider Table I, depicting a (simplified) example event log. Each row records the execution of an *activity*, e.g., the first row of the table records the execution of the activity *receive return* (short-hand notation a). Multiple activities are recorded for the same instance of the process. The *Case-id* column tracks this relationship, e.g., the 2nd and 3rd row in the table refer to the same process instance, identified by id 484. In this paper, we only use the *control-flow perspective* of processes, i.e., the sequential scheduling of the process activities in the context of an instance of the process. We define event logs as follows.

Definition 1 (Event Log)**.** *Let* Σ *denote the* universe of process activities*. An* event log L *is a multiset of sequences of activities, i.e.,* $L{\in}\mathcal{M}(\Sigma)$*. A sequence* $\sigma{\in}\overline{L}$ *is a* trace*.*

C. Petri Nets

Different *process modeling languages* exist, e.g., Business Process Model and Notation (BPMN) [3], allowing one to create a conceptual representation of a business process. Most of these modeling languages can be translated into *Petri nets* [12], i.e., a well-defined mathematical notation for concurrent systems. Hence, we adopt Petri nets as a process modeling formalism in this paper.

A Petri net is a bipartite graph consisting of *places* (visualized as circles) and *transitions* (visualized as boxes). Places are only connected to transitions and vice versa. Consider Fig. 1 which exemplifies a Petri net, based on the (short-hand) activity labels presented in Table I. To represent the state of a Petri net, we assign *tokens* to the places of the net. For example, in Fig. 1, there is one token assigned to place p_1. A precondition to *fire* a transition is that all of the places connected to it by means of an incoming arc should contain a

token, e.g., in Fig. 1, transition t_1 is the only transition that is enabled, since place p_1 contains a token. When we fire t_1, the token from p_1 is removed and a token is produced in places p_2 and p_3. We assume that firing transition t_1 emits activity a. Some transitions are used for *token-routing* and do not emit an activity, e.g., transition t_7 in Fig. 1. We formally define (labeled) Petri nets as follows.

Definition 2 (Labeled Petri Nets)**.** *Let Σ denote the universe of process activities and let $\tau \notin \Sigma$. Let P denote a finite set of* places *and let T denote a finite set of* transitions *s.t. $P \cap T = \emptyset$. Let $F \subseteq (P \times T) \cup (T \times P)$ be a set of directed arcs representing the* flow relation*. Let $\ell : T \to \Sigma \cup \{\tau\}$ be a label function. A labeled Petri net N is a quadruple $N = (P, T, F, \ell)$.*

Given net $N=(P,T,F,\ell)$ and $x \in P \cup T$, we write $\bullet x = \{y \in P \cup T \mid (y,x) \in F\}$ and $x \bullet = \{y \in P \cup T \mid (x,y) \in F\}$. We refer to the aforementioned distribution of tokens over places as a *marking*, which is a *multiset of places*, e.g., the marking of N_1 (Fig. 1) is $[p_1]$. Given a net $N=(P,T,F,\ell)$ and marking $m \in \mathcal{M}(P)$, pair (N,m) is a *marked net*. A transition $t \in T$ is enabled in marking m, written $(N,m)[t\rangle$, iff for each $p \in \bullet t$, $m(p) > 0$. If we fire t in marking m, we obtain marking $m' = (m - \bullet t) \uplus t\bullet$. We write this as $(N,m) \xrightarrow{t} (N,m')$. A sequence of transitions $\sigma \in T^*$ of length n is a *firing sequence* of a marked net (N,m), if there exist markings $m_1, m_2, \ldots, m_n$, s.t., $(N,m) \xrightarrow{\sigma(1)} (N,m_1) \xrightarrow{\sigma(2)} (N,m_2) \ldots (N,m_{n-1}) \xrightarrow{\sigma(n)} (N,m_n)$. For this, we write $(N,m) \xrightarrow{\sigma} (N,m_n)$. We let $\mathcal{L}(N,m,m') = \{\sigma \in T^* \mid (N,m) \xrightarrow{\sigma} (N,m')\}$ and $\mathcal{R}(N,m) = \{m' \in \mathcal{M}(P) \mid \exists \sigma \in T^* \left((N,m) \xrightarrow{\sigma} (N,m')\right)\}$. The *incidence matrix* $\mathbf{N}$ of net $N=(P,T,F,\ell)$ is a $|P| \times |T|$ matrix s.t.:

$$\mathbf{N}(p,t) = \begin{cases} -1 & \text{if } (p,t) \in F \text{ and } (t,p) \notin F \\ 1 & \text{if } (t,p) \in F \text{ and } (p,t) \notin F \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

We let $\mathbf{N}^-$ denote the *consumption matrix* of N, where:

$$\mathbf{N}^-(p,t) = \begin{cases} -1 & \text{if } (p,t) \in F \text{ and } (t,p) \notin F \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Let $m, m' \in \mathcal{M}(P)$, let $\vec{m}, \vec{m'}$ denote two corresponding *column vectors* representing markings m and m', and let $\sigma \in T^*$ s.t. $\sigma \in \mathcal{L}(N,m,m')$. The *marking equation* [12] states that $\vec{m} + \mathbf{N} \cdot \vec{\sigma} = \vec{m'}$.[1] Further, let $\sigma_1, \sigma_2 \in T^*$ be two firing sequences and let m, m', m'' s.t. $(N,m) \xrightarrow{\sigma_1} (N,m') \xrightarrow{\sigma_2} (N,m'')$ (i.e., $\sigma_1 \cdot \sigma_2 \in \mathcal{L}(N,m,m'')$). The *extended marking equation* [7] states that (i) $m + \mathbf{N} \cdot (\vec{\sigma_1} + \vec{\sigma_2}) = \vec{m''}$ and (ii) $m + \mathbf{N} \cdot \vec{\sigma_1} + \mathbf{N}^- \cdot \vec{1}_{\sigma_2(1)} = \vec{m''}$.[2] Hence, the extended marking equation additionally states (w.r.t. the conventional marking equation) that the first element of σ_2 needs to be enabled after firing σ_1. Both equations generate a body of constraints when replacing σ (σ_1 and σ_2, respectively) by a vector of variables $\vec{x}$ ($\vec{x}_1$ and $\vec{x}_2$, respectively) of size $|T|$. Many solutions to such a constraint body may exist. In case one defines a cost function over the elements of T, techniques such as (I)LP [6] can be used to find an *optimal solution*. However, it is not guaranteed that a solution found (optimal or not), i.e., a variable assignment $\vec{x}$, corresponds to the Parikh vector of any $\sigma \in \mathcal{L}(N,m,m')$. In the remainder, we refer to a solution that does correspond to the Parikh vector of some $\sigma \in \mathcal{L}(N,m,m')$ as a *realizable solution*.

[1]We assume the indexes of the elements in the vectors coincide with $\mathbf{N}$.

[2]Note that $\vec{1}_x$ represents the unit-vector for element x.

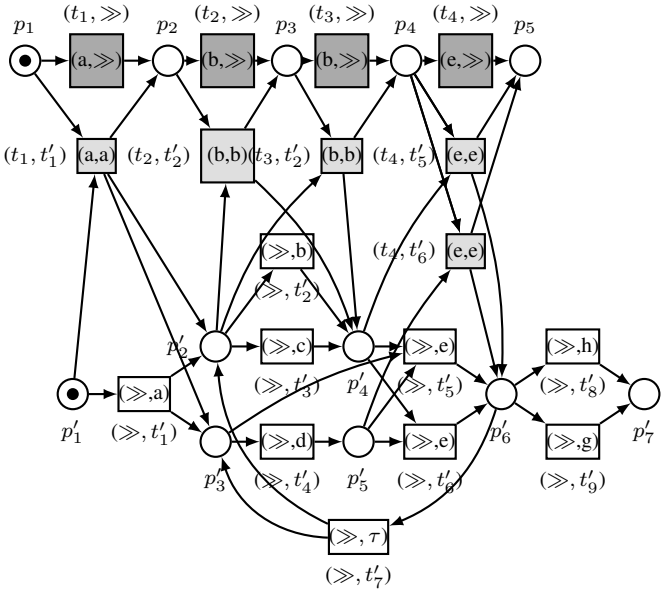


Fig. 2: Example Synchronous product net of Petri net N_1 (Fig. 1) and the Petri net depicted in Fig. 4. Note that all elements $x \in P \cup T$ of N_1 have been relabeled to x', e.g., p'_1, t'_1, etc.

Finally, we introduce the *synchronous product net*, which combines two separate nets into one net. For example, consider Fig. 2, in which we depict an example synchronous product net of Petri net N_1 (Fig. 1) and the simple Petri net depicted in Fig. 4. Note that, in Fig. 2, all elements $x \in P \cup T$ of N_1 have been relabeled to x', e.g., p'_1, t'_1, etc. We formally define the notion of a synchronous product net as follows.

Definition 3 (**Synchronous Product Nets**)**.**
Let $N=(P,T,F,\ell)$ and $N'=(P',T',F',\ell')$ be two labeled Petri nets with $P \cap P' = \emptyset$ and $T \cap T' = \emptyset$. The synchronous product net of N and N' is $N \otimes N' = (P^\otimes, T^\otimes, F^\otimes, \ell^\otimes)$ where:
- $P^\otimes = P \cup P'$
- $T^\otimes = (T \times \{\gg\}) \cup \{\gg\} \times T') \cup \{(t,t') \in T \times T' \mid \ell(t) = \ell(t')\}$
- $F^\otimes = \{(p,(t,t')) \in P^\otimes \times T^\otimes \mid (p,t) \in F \vee (p,t') \in F'\} \cup \{((t,t'),p) \in T^\otimes \times P^\otimes \mid (t,p) \in F \vee (t',p) \in F'\}$
- $\ell^\otimes : T^\otimes \to (A \cup \{\tau\} \cup \{\gg\}) \times (A \cup \{\tau\} \cup \{\gg\})$ *with* $\ell^\otimes(t, \gg) = (\ell(t), \gg)$ *for* $t \in T$, $\ell^\otimes(\gg, t') = (\gg, \ell'(t'))$ *for* $t' \in T'$ *and* $\ell^\otimes(t,t') = (\ell(t), \ell'(t'))$ *for* $t \in T$ *and* $t' \in T'$.

D. Alignments

Alignments relate a sequence of activities (i.e., a trace) to a firing sequence in a Petri net. An example alignment

σ	a	b	b	e	$\gg$
T	t_1	t_2	$\gg$	t_5	t_8

Fig. 3: An alignment of N_1 and trace $\langle a, b, b, e\rangle$; The 2nd observed activity b is obsolete and an activity g (described by t_8) is missing.

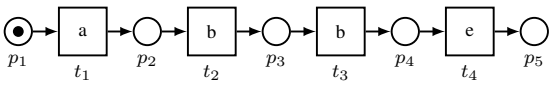


Fig. 4: The *trace net* of trace $\langle a, b, b, e\rangle$

is depicted in Fig. 3. The alignment quantifies the trace $\langle a, b, b, e\rangle$ in terms of Petri net N_1 (Fig. 1), i.e., the trace is mapped to a firing sequence in the language of the net. In the example, the trace is mapped to firing sequence $\langle t_1, t_2, t_5, t_8\rangle$. The individual elements of an alignment are *moves*, e.g., (a, t_1) in Fig. 3. Moves of the form (a, t) for $a{\in}\Sigma$ and $t{\in}T$ (*synchronous moves*) indicate that observed behavior in the trace can be mapped to the behavior described by the model. Moves of the form $(a, \gg)$ for $a{\in}\Sigma$ (*log moves*) indicate that the observed behavior cannot be mapped to the behavior described by the model. Finally, moves of the form $(\gg, t)$ for $t{\in}T$ (*model moves*) indicate that behavior is missing, according to the model. We formally define alignments as follows.

Definition 4 (Alignment)**.** *Let Σ denote the universe of activities, let $N{=}(P, T, F, \ell)$ be a labeled Petri net (where $\ell\colon T{\to}\Sigma{\cup}\{\tau\}$ and $\tau{\notin}\Sigma$) and let $m, m'{\in}\mathcal{M}(P)$. Further, let $\gg{\notin}\Sigma{\cup}T$. An* alignment *γ of $\sigma{\in}\Sigma^*$ and N, conditional to m and m', is a sequence of moves, i.e., $\gamma{\in}((\Sigma{\cup}\{\gg\}){\times}(T{\cup}\{\gg\}))^*$, s.t.,: (i) $\pi_1^*(\gamma)|_\Sigma{=}\sigma$, (ii) $\pi_2^*(\gamma)|_T{\in}\mathcal{L}(N, m, m')$, and (iii) if for $1{\leq}i{\leq}|\gamma|$, $\gamma(i){=}(a, t)$ with $a{\in}\Sigma$ and $t{\in}T$, then $a{=}\ell(t)$. We let $\Gamma(\sigma, N, m, m')$ denote the set of all possible alignments of σ and N, conditional to m and m'.*

For a trace, model, and markings, multiple alignments exist, e.g., replacing $(\gg, t_8)$ by $(\gg, t_9)$ in Fig. 3 also yields an alignment. We prefer alignments that describe the observed trace using a firing sequence closest to the trace. As such, we associate a cost function $c\colon (\Sigma{\cup}\{\gg\}){\times}(T{\cup}\{\gg\}){\to}\mathbb{N}$ to the moves and aim to find any $\gamma{\in}\arg\min_{\gamma'\in\Gamma(\sigma,N,m,m')} \sum_{i\leftarrow 1}^{|\gamma'|} c(\gamma'(i))$, i.e., *an optimal alignment under cost function c.* The *standard cost function* assigns cost 0 to synchronous moves (and moves $(\gg, t)$ with $\ell(t){=}\tau$), 1 to log and model moves and ∞ to $(\gg, \gg)$. Computing an optimal alignment is equivalent to solving the shortest path problem (using cost functino c as a distance function) on the state space of the synchronous product net of a *trace net* of the trace (e.g., see Fig. 4 for trace $\langle a, b, b, e\rangle$) and the Petri net modeling the process [10].

IV. Cache Enhanced Split-Point-Based Alignment Calculation

In this section, we present our main contribution. We briefly describe split-point-based alignment calculation in Section IV-A. Subsequently, we present our proposed caching strategy in Section IV-B. Finally, we present parameterization and pruning techniques for our approach in Section IV-C.

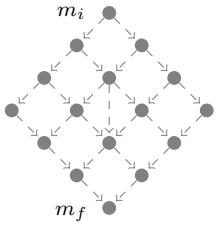
A. Split-Point-Based Alignment Calculation

This section presents the conceptual basics of the split-point-based alignment computation [7]. For the sake of brevity, we provide an informal algorithmic sketch.

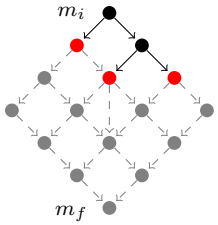
1) *Compute Heuristic*; An extension of the *extended marking equation* is solved for the synchronous product net SN [7, Definition 8]. The extension generalizes the number of possible subsequences used in the equation to compute a *possibly realizable* transition vector $\vec{x}$.
2) *State-Space Traversal*; The algorithm starts in the initial marking m_i of the synchronous product SN and traverses its state-space $\mathcal{S}(SN)$. In general, a marking m in $\mathcal{S}(SN)$ is considered by the search algorithm if it, during the search, is reached by a sequence of transitions $\sigma{\in}T^*$ (i.e., $(SN, m_i)\overset{\sigma}{\twoheadrightarrow}(SN, m)$) s.t. $\vec{x}{-}\vec{\sigma}{\geq}\vec{0}$. Markings eligible for consideration are referred to as *feasible markings*, and all other markings are referred to as *infeasible*. Prioritization of feasible markings is based on the underlying alignment cost function, i.e., the marking corresponding to the cheapest currently known path is selected first.
   There are two possible outcomes of the traversal (with an associated *resolution strategy*):
   a) During the search, we reach the final marking m_f. We reconstruct the path $\gamma{\in}T^*$ s.t. $(SN, m_i)\overset{\gamma}{\twoheadrightarrow}(SN, m_f)$, i.e., the *optimal alignment*. In this case, $\vec{x}$ is *realizable* and is the Parikh vector of the optimal alignment γ.
   b) The search gets stuck, i.e., $\vec{x}$ is *not realizable*. In this case, two possible *resolution strategies* apply:
      i) If we can further *split-up* the extended marking equation, we do so and restart the complete procedure from step 1.
      ii) If no additional *split* can be defined, the algorithm resorts back to the regular A^*-search, using the currently assessed state space as a starting point.[3]

Consider Fig. 5, in which we present a schematic visualization of the algorithm. Observe that each iteration starts from the unexplored state space, i.e., depicted in Fig. 5a. Given the initial solution vector $\vec{x}$, i.e., based on the initial solution to the extended marking equation, the algorithm explores the state space, starting from m_i. In the first search iteration (Fig. 5b), the left-most child (visualized in red) is infeasible, i.e., it is not described by $\vec{x}$. The right-most child of m_i is feasible. However, after subsequent exploration of said marking, the algorithm gets stuck. After revising the extended marking equation, the 2-nd iteration of the algorithm gets stuck

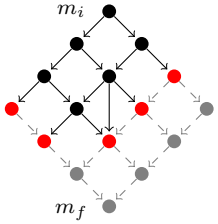
[3]The algorithm maintains an *Open* and *Closed* set, which can directly be adopted by conventional A^*.



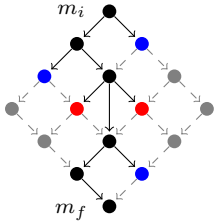
(a) Unexplored state space



(b) 1-st search iteration gets stuck



(c) 2-nd search iteration gets stuck



(d) 3-rd iteration reaches final marking

Fig. 5: Schematic visualization of the state space traversal of the conventional split-point-based alignment algorithm. In-between each step, the algorithm restarts, i.e., any previous knowledge is ignored.Black states are previously explored feasible states, blue states are unexplored feasible states, red states are infeasible states.

again (Fig. 5c). Finally, the 3-rd iteration of the search does allow us to reach the final marking.

The computation of the initial heuristic and corresponding solution vector is defined in [7, Definition 8]. Conceptually, the algorithm solves an (I)LP program based on the extended marking equation (Equation 2). The (I)LP allows for using multiple subsequences, i.e., k fragments, rather than just 2 as presented in Equation 2. The objective function minimizes the cost function associated with the synchronous product net (i.e., equivalent to the alignment cost function). If the search gets stuck, the algorithm investigates the maximally explained event of the trace in the current iteration of the search. For example, if the algorithm applied on the synchronous product net of Fig. 2 never considered any marking that contains place p_4, the maximum explained event is the third event of the trace (the 2-nd b activity, represented by $(t_3, \gg)$). In the next iteration, the algorithm adds an additional firing sequence fragment to its body of constraints and enforces that the heuristic computed either uses $(t_4, \gg)$, (t_4, t'_5) or (t_4, t'_5). The addition of the additional constraint forces the next heuristic to be different from the previous solution. Suppose the maximally explained event was already added to a heuristic of a previous iteration. In that case, the algorithm keeps adding *split-points* to other events that occur before the maximally explained event. If no split-point can be added, the algorithm resorts to classical A^*.

B. Caching Infeasible Markings

Observe that the baseline algorithm, i.e., as described in Section IV-A, redundantly revisits several states in the state space. In the example depicted in Fig. 5, the three states

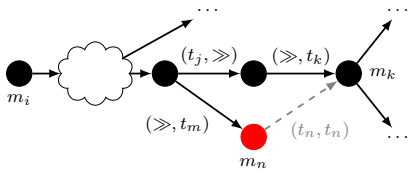


Fig. 6: Unexplored cheaper sequences may lead to a marking in closed set.

connected to m_i are visited in every search. Therefore, we propose a *caching* strategy that stores infeasible states. When the search gets stuck, we revise the extended marking equation similarly to [7]. However, we directly assess whether any infeasible states of the previous iteration become feasible using the heuristic computed in the current iteration. In the remainder, we denote $\vec{x}_i$ to represent the possibly realizable solution vector computed in iteration i of the algorithm (note that $i \leq |\sigma|$, i.e., we add at most $|\sigma|$ split-points when aligning σ). The initial solution computed is referred to as $\vec{x}_0$.

Main Search Procedure: Our algorithm maintains three sets, i.e., O, C, and X, in which markings of the state space of SN are stored. Set O is the *open set*, containing all markings that still need to be explored (visualized in blue in all upcoming visualizations in this paper). Set C is the *closed set*, containing all markings that have been visited (visualized in black in all upcoming visualizations in this paper). Finally, set X is a cache set containing infeasible markings, i.e., infeasible in the context of the current search iteration (visualized in red). For every marking m, we additionally keep track of the Parikh vectors of all possible firing sequences σ of the form $(SN, m_i) \xrightarrow{\sigma} (SN, m)$ that we have observed so far during the search. Given $m \in \mathcal{M}(P)$, we refer to these vectors as $\mathrm{p}(m) \in \mathcal{P}(\mathbb{N}^{|T|})$. When we add a marking to the closed set, we store the current iteration i with it.

In the initial state of the algorithm, we have $O = \{m_i\}$ and $X = C = \emptyset$. As long as the open set is not empty, in any iteration i, we perform the following procedure.

1) Obtain the marking m from O that represents the currently known shortest path among all members of O (typically, O is implemented as a priority queue).
2) For every m' s.t. $\exists t \in T((SN, m) \xrightarrow{t} (SN, m'))$ and $(m', i) \notin C$, we add $\{\vec{y} - \vec{1}_t | \vec{y} \in \mathrm{p}(m)\}$ to $\mathrm{p}(m')$.
3) If any $\vec{y} \in \mathrm{p}(m')$ exists s.t. $\vec{x}_i - \vec{y} \geq \vec{0}$, we add m' to O, else, we add it to X.
4) We add (m, i) to C.

Our main procedure re-assesses markings that have been added to the closed set in a previous iteration of the search (i.e., step 2 implies assessing any (m, j) with $j < i$). This is required since the current path assessed may be cheaper than the previously known path. Consider Fig. 6, in which we present a schematic example of this scenario. The shortest path to reach m_k from m_i is via m_n, yet, in the current iteration, m_n is infeasible. Consequently, an alternative and more expensive path is currently assessed for m_k.

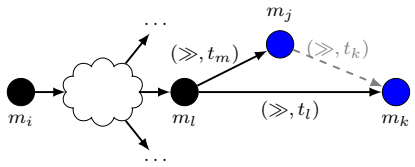


Fig. 7: Schematic visualization of ignoring more expensive paths. The move $(\gg, t_k)$ does not need to be considered as it yields a more expensive path to m_k

Reviving Cached States: Similar to the original algorithm, our search procedure may get stuck, i.e., if solution $\vec{x}_i$ is not realizable. In this case, we perform the same splitting procedure as the original algorithm, i.e., we compute a new solution $\vec{x}_{i+1}$ using an enriched version of the extended marking equation. However, rather than restarting the search, we assess the markings in the cache X for feasibility under the newly computed solution $\vec{x}_{i+1}$. We move any marking $m{\in}X$ back to O iff $\exists \vec{y}{\in}\mathrm{p}(m)(\vec{x}_{i+1}{-}\vec{y}{\geq}\vec{0})$, i.e., any marking that is on a possible realizable firing sequence is reconsidered. After assessing all markings in the cache set, the aforementioned *Main Search Procedure* is triggered again.

Fall-Back to A^*: Finally, like the original algorithm, our approach falls back to regular A^* when the open set is empty, and no further extension of the extended marking equation is possible. However, in our search, i.e., as opposed to the regular A^* (and [7]), whenever a state is visited that is already the closed set, the state should be reconsidered.

C. Parameterization and Pruning

In this section, we present parameterization and pruning.

Clearing the Cache: Whereas our approach avoids redundantly revisiting states during the alignment search, in practical settings, the initially computed heuristic may guide the search "in the wrong direction". To this end, we define the use of a *reset-counter* $r{\in}\mathbb{N}$. Given iteration i of the algorithm, whenever $i \bmod r = 0$, we clear the closed, open, and cache set and restart the algorithm with $O{=}\{m_i\}$. Observe that for $r{=}1$, the algorithm is equal to the original algorithm as presented in [7]. Similarly, no restarts are executed for $r{=}\infty$.

Ignoring Expensive Paths: Our basic scheme considers any path leading to a reachable marking m' (from some current marking m). However, from marking m' (i.e., a marking of the SN), we are free to continue in the state space in any possible way, i.e., this does not depend on the path that leads to m'. Only the paths that yield the same costs to the currently known best costs for m' need to be considered. Consequently, all previously stored paths are removed if a cheaper path to m' is discovered. Consider Fig. 7, in which we schematically visualize an example of such a situation. The path to m_k via m_j needs not to be considered as it is more expensive than the path via $(\gg, t_l)$. The same applies to adding states that are already in the closed set, i.e., states of the form $(m', j){\in}C$ with $j{<}i$. Only when the newly observed path is cheaper, we reinsert the marking to O.

TABLE II: Experimental Setup

Parameter	Value
Event log	Hospital Bill [16] Conformance Checking Challenge 2019 [17] BPI Challenge 2020: International Declarations [18]
Models	Inductive Miner & Noise threshold=0.05 Inductive Miner & Noise threshold=0.20 Inductive Miner & Noise threshold=0.80
Alignment Algorithm	Regular A* search (ras) [10] Split-point based search (sps) [7] Split-point & naive caching search (spncs) Split-point & modified caching search (spsmc) Split-point & reset counter=1 search (spsc1) ... Split-point & reset counter=5 search (spsc5)

Ignoring Closed States: In the basic scheme presented in Section IV-B, we only ignore a visited state if $(m, i){\in}C$, i.e., the marking is already closed and visited in the current iteration i (possibly, only if it is also cheaper). If we ignore any state present in C, i.e., any tuple of the form (m', j) with $j{\leq}i$, we further enhance the overall search efficiency. However, optimality of the alignments computed cannot be guaranteed (cf. Fig. 6).

V. Evaluation

In this section, we present the evaluation of our cache-enhanced split-point based algorithm, which is implemented in `python`, i.e., extending the process mining framework `pm4py` [13].[4] We re-implemented the regular A^* and split-point-based approach to achieve a fair comparison.

A. Experimental Setup

We conducted experiments on ten different event logs to analyze the performance of different algorithms. The main difference between the selected logs is the average length of the traces and the number of trace variants. Due to space limitations, we report on a subset of the results.[5] For each log, we mined four models with Inductive Miner (Infrequent) algorithm [14], implemented in ProM [15]. Specifically, the noise threshold of the Inductive Miner is set to 0.05, 0.20, 0.40, and 0.80. Increasing the noise threshold filters out more infrequent behaviors, resulting in models with fewer transitions.

We summarize the experimental setup of the results presented in Table II. In the *spncs* variant of the algorithm, closed states are never reconsidered (as described in Section IV-C). Similarly, *spsmc* refers to revisiting closed states when a shorter path is found. In all versions with restart, i.e., *spsc1*, ..., *spsc5*, states in the closed set are re-openend when required. We uploaded the code, event logs and models to an elastic cloud server, and conducted all experiments in single-threaded mode.[6].

[4] https://github.com/brucelit/State_space_traversal

[5] See https://drive.google.com/drive/folders/1l-y09o-DuataYtJCLsYZxnzSDSgO83YO?usp=sharing for all results.

[6] `3.5 GHz Intel Xeon Platinum 8369HC CPU`, `4 GB RAM` and `40 GB Disk Space`

TABLE III: Results of total computation time and state space traversed for Hospital Bill [16]

Algorithm	Time			State space explored	
	total	heuristic	queue	states	arcs
Model 0.05					
ras	1.4E+04	1.2E+04	4.8E+02	9.1E+06	4.9E+07
sps	1.8E+04	1.0E+04	5.7E+03	9.2E+06	5.2E+07
spncs	1.2E+04	8.1E+03	3.1E+03	6.4E+06	2.8E+07
spmcs	1.3E+04	8.3E+03	3.2E+03	6.5E+06	2.8E+07
spsc1	1.7E+04	9.9E+03	4.9E+03	9.4E+06	5.2E+07
spsc2	1.5E+04	9.1E+03	3.7E+03	8.4E+06	3.8E+07
spsc3	1.4E+04	9.0E+03	3.5E+03	7.8E+06	3.5E+07
spsc4	1.3E+04	8.4E+03	3.2E+03	7.0E+06	3.0E+07
spsc5	1.3E+04	8.3E+03	3.1E+03	6.7E+06	2.9E+07
Model 0.20					
ras	3.5E+03	3.1E+03	1.8E+02	3.5E+06	2.1E+07
sps	9.0E+02	3.6E+02	3.3E+02	1.7E+06	8.3E+06
spncs	6.9E+02	3.2E+02	1.5E+02	1.6E+06	6.7E+06
spmcs	7.3E+02	3.2E+02	1.5E+02	1.6E+06	6.7E+06
spsc1	8.3E+02	3.4E+02	2.0E+02	1.7E+06	8.5E+06
spsc2	7.5E+02	3.3E+02	1.6E+02	1.7E+06	7.0E+06
spsc3	7.5E+02	3.3E+02	1.5E+02	1.6E+06	6.8E+06
spsc4	7.4E+02	3.3E+02	1.5E+02	1.6E+06	6.8E+06
spsc5	7.3E+02	3.3E+02	1.5E+02	1.6E+06	6.8E+06
Model 0.80					
ras	3.4E+02	2.9E+02	1.3E+02	7.1E+05	2.5E+06
sps	3.7E+02	2.3E+02	7.0E+01	8.1E+05	2.8E+06
spncs	2.9E+02	2.1E+02	5.3E+00	7.5E+05	2.4E+06
spmcs	3.0E+02	2.1E+02	5.5E+00	7.5E+05	2.4E+06
spsc1	3.1E+02	2.0E+02	1.4E+01	7.5E+05	2.7E+06
spsc2	3.0E+02	2.1E+02	7.2E+00	7.6E+05	2.4E+06
spsc3	3.0E+02	2.1E+02	5.7E+00	7.5E+05	2.4E+06
spsc4	3.0E+02	2.1E+02	5.4E+00	7.5E+05	2.4E+06
spsc5	3.0E+02	2.1E+02	5.4E+00	7.5E+05	2.4E+06

TABLE IV: Results of total computation time and state space traversed for BPIC2020-International Declarations [18]

Algorithm	Time			State space explored	
	total	heuristic	queue	states	arcs
Model 0.05					
ras	1.6E+02	1.4E+02	1.3E+01	2.8E+05	1.4E+06
sps	7.1E+01	2.3E+01	3.1E+01		7.6E+05
spncs	5.3E+01	2.3E+01	9.4E+00	1.5E+05	6.8E+05
spmcs	5.6E+01	2.3E+01	9.4E+00	1.5E+05	6.8E+05
spsc1	6.1E+01	2.3E+01	1.3E+01	1.6E+05	7.3E+05
spsc2	5.8E+01	2.3E+01	1.1E+01	1.6E+05	7.1E+05
spsc3	5.6E+01	2.3E+01	9.4E+00	1.5E+05	6.9E+05
spsc4	5.6E+01	2.3E+01	9.3E+00	1.5E+05	6.9E+05
spsc5	5.6E+01	2.3E+01	9.2E+00	1.5E+05	6.9E+05
Model 0.20					
ras	6.0E+03	5.6E+03	8.9E+01	1.9E+06	1.4E+07
sps	4.7E+02	7.7E+01	2.5E+02	8.5E+05	5.6E+06
spncs	3.0E+02	1.1E+02	8.6E+01	6.3E+05	2.9E+06
spmcs	3.1E+02	8.6E+01	7.5E+01	6.1E+05	3.9E+06
spsc1	3.9E+02	8.0E+01	1.4E+02	8.7E+05	4.3E+06
spsc2	3.0E+02	8.0E+01	9.2E+01	7.1E+05	3.3E+06
spsc3	3.1E+02	8.1E+01	7.5E+01	6.2E+05	4.0E+06
spsc4	3.1E+02	8.8E+01	7.4E+01	6.1E+05	3.9E+06
spsc5	3.2E+02	9.0E+01	7.5E+01	6.1E+05	4.0E+06
Model 0.80					
ras	1.7E+02	1.6E+02	1.2E+01	1.2E+05	4.2E+05
sps	1.0E+02	7.0E+01	1.7E+01	1.4E+05	5.2E+05
spncs	8.5E+01	7.0E+01	3.5E+00	1.2E+05	3.1E+05
spmcs	9.0E+01	7.1E+01	4.2E+00	1.1E+05	4.0E+05
spsc1	9.5E+01	7.1E+01	8.9E+00	1.6E+05	4.0E+05
spsc2	9.0E+01	7.0E+01	5.6E+00	1.3E+05	3.4E+05
spsc3	9.3E+01	7.2E+01	5.5E+00	1.2E+05	4.3E+05
spsc4	9.1E+01	7.1E+01	4.4E+00	1.2E+05	4.1E+05
spsc5	9.2E+01	7.2E+01	4.4E+00	1.2E+05	4.1E+05

B. Results

In this section, we present the results of the experiments performed. We first assess the total computation time to align all traces in the log, after which we focus on the state space traversed regarding the number of states and arcs visited. Due to space limitations, we only show the results of three event logs (out of the ten considered) in Tables III- V. All obtained results show similar trends to the results presented here. We apply a blue-red color-coding scheme for each column to visualize the respective magnitude of a particular trend within a column. A blue color represents a value that is below average, a red color indicates a value that is higher than average.

Time Consumption: The total running time is significantly reduced for *spncs*. As indicated, the computed alignments are not always optimal. However, we hardly observe non-optimal alignments in the experiments. For example, for the event log Hospital Bill (Table III), there are 4 traces yielding non-optimal alignments, representing 0.5% of traces in the log. For *spmcs*, the overall running time is slightly higher compared to *spncs*. We also observe an increasing trend in performance when increasing the reset counter value. Hence, we observe that using a cache set without resets leads to the best time performance.

Search Efficiency: Another critical factor we measured is the state-space traversal efficiency, which is represented by the number of states and arcs traversed during the search. Due to the frequent restart of *sps*, the state space is repeatedly explored. Its state-space traversal efficiency is usually the worst among all experiments. In line with the results obtained for the time performance, we observe that *spncs* performs slightly better than *spmcs*.

Memory Efficiency: As our caching strategy requires additional memory, i.e., storing Parikh vectors per marking in the search space and keeping markings in the cache set after recomputing the initial heuristic, we additionally measured memory performance. In general, the memory used during the search increases up to 20%, depending on the log and model. The reason is that caching strategy forbids restart, and more memory is allocated to store all known information of the state space explored, i.e., all states visited and corresponding Parikh vectors. For example, in Conformance Checking Challenge 2019 with model 0.05, the average memory for the split-point-based approach is 186.75 MB, while the naive and modified caching strategy consumed 201.7 MB and 223.55 MB, respectively.

VI. Conclusion

In this paper, we presented an extension of the existing split-point-based alignment algorithm. Our approach introduces a cache in which markings, rendered infeasible in the current iteration of the search, are temporarily stored. The addition of the cache reduces the number of redundantly revisiting the same states in the state space. We conducted various

TABLE V: Results of total computation time and state space traversed for Conformance Checking Challenge 2019 [17]

Algorithm	Time			State space explored	
	total	heuristic	queue	states	arcs
Model 0.05					
ras	4.20E+02	3.92E+02	3.70E+00	6.4E+04	7.9E+05
sps	1.84E+03	1.27E+02	1.56E+03	1.4E+05	1.8E+06
spncs	4.94E+02	6.87E+01	2.90E+02	5.8E+04	7.0E+05
spmcs	4.97E+02	6.66E+01	3.00E+02	5.7E+04	7.0E+05
spsc1	9.18E+02	7.27E+01	6.98E+02	1.4E+05	1.8E+06
spsc2	4.89E+02	7.34E+01	3.07E+02	1.1E+05	1.4E+06
spsc3	5.47E+02	7.02E+01	3.60E+02	8.5E+04	1.0E+06
spsc4	5.93E+02	6.95E+01	3.86E+02	7.7E+04	9.5E+05
spsc5	5.47E+02	7.49E+01	3.44E+02	7.2E+04	8.8E+05
Model 0.20					
ras	5.55E+01	5.33E+01	5.00E-01	1.2E+04	6.3E+04
sps	1.75E+02	1.14E+02	4.85E+01	3.9E+04	2.2E+05
spncs	4.03E+01	3.20E+01	5.50E+00	7.4E+03	3.7E+04
spmcs	3.90E+01	2.96E+01	5.60E+00	8.3E+03	4.4E+04
spsc1	2.41E+02	1.00E+02	8.07E+01	1.2E+05	6.0E+05
spsc2	1.55E+02	6.12E+01	4.71E+01	9.0E+04	4.4E+05
spsc3	1.26E+02	5.88E+01	3.40E+01	6.6E+04	3.3E+05
spsc4	7.92E+01	4.06E+01	1.96E+01	4.0E+04	2.0E+05
spsc5	5.76E+01	4.02E+01	1.05E+01	1.8E+04	1.0E+05
Model 0.80					
ras	3.40E+00	3.20E+00	1.00E-01	1.9E+03	5.0E+03
sps	2.50E+00	1.80E+00	4.00E-01	2.2E+03	6.2E+03
spncs	2.20E+00	2.10E+00	0.00E+00	1.1E+03	2.2E+03
spmcs	1.30E+00	1.10E+00	0.00E+00	1.1E+03	3.2E+03
spsc1	1.60E+00	1.20E+00	2.00E-01	2.2E+03	6.4E+03
spsc2	1.40E+00	1.10E+00	1.00E-01	1.7E+03	4.9E+03
spsc3	1.50E+00	1.20E+00	1.00E-01	1.5E+03	4.3E+03
spsc4	1.30E+00	1.10E+00	1.00E-01	1.4E+03	4.0E+03
spsc5	1.30E+00	1.10E+00	1.00E-01	1.3E+03	3.6E+03

experiments with real-life event logs and evaluated the results with existing techniques. With a compromise on memory efficiency to store all visited states without restart, our proposed caching strategy aids the split-point-based algorithm to traverse state space more efficiently. Due to this efficiency gain, the overall time efficiency of alignment calculation is significantly improved.

Future Work: We aim to extend our current work as follows. During the search, some states remain in the cache during various subsequent iterations. As such, we aim to develop dynamic caching to reduce the memory consumption of the cache. We further aim to apply *bidirectional search* based on the extended marking equation.

References

[1] J. Carmona, B. F. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking - Relating Processes and Models*. Springer, 2018.

[2] W. M. P. van der Aalst, *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.

[3] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in BPMN," *Inf. Softw. Technol.*, vol. 50, no. 12, pp. 1281–1294, 2008.

[4] S. J. van Zelst, A. Bolt, and B. F. van Dongen, "Computing alignments of event data and process models," *Trans. Petri Nets Other Model. Concurr.*, vol. 13, pp. 1–26, 2018.

[5] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[6] A. Schrijver, *Theory of linear and integer programming*, ser. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

[7] B. F. van Dongen, "Efficiently computing alignments - using the extended marking equation," in *BPM 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings*, M. Weske, M. Montali, I. Weber, and J. vom Brocke, Eds., vol. 11080. Springer, 2018, pp. 197–214.

[8] A. Rozinat and W. M. P. van der Aalst, "Conformance checking of processes based on monitoring real behavior," *Inf. Syst.*, vol. 33, no. 1, pp. 64–95, 2008.

[9] R. P. J. C. Bose and W. M. P. van der Aalst, "Trace alignment in process mining: Opportunities for process diagnostics," in *BPM 2010, Hoboken, NJ, USA, September 13-16, 2010. Proceedings*, R. Hull, J. Mendling, and S. Tai, Eds., vol. 6336. Springer, 2010, pp. 227–242.

[10] A. Adriansyah, "Aligning observed and modeled behavior (Ph. D. thesis)," *Eindhoven University of Technology*, 2014.

[11] A. Syamsiyah and B. F. van Dongen, "Improving alignment computation using model-based preprocessing," in *ICPM 2019, Aachen, Germany, June 24-26, 2019*. IEEE, 2019, pp. 73–80.

[12] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[13] A. Berti, S. J. van Zelst, and W. M. P. van der Aalst, "Process mining for python (pm4py): Bridging the gap between process- and data science," *CoRR*, vol. abs/1905.06169, 2019.

[14] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from event logs containing infrequent behaviour," in *BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers*, ser. Lecture Notes in Business Information Processing, N. Lohmann, M. Song, and P. Wohed, Eds., vol. 171. Springer, 2013, pp. 66–78.

[15] E. Verbeek, J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst, "Prom 6: The process mining toolkit," in *Proceedings of the Business Process Management 2010 Demonstration Track, Hoboken, NJ, USA, September 14-16, 2010*, M. L. Rosa, Ed., vol. 615. CEUR-WS.org, 2010.

[16] F. Mannhardt, "Hospital billing - event log," Aug. 2017.

[17] J. Munoz-Gama, R. de la Fuente R., M. Sepúlveda, and R. Fuentes, "Conformance checking challenge 2019 (ccc19)," Feb. 2019.

[18] B. F. van Dongen, "BPI Challenge 2020: International Declarations," 3 2020.