Computing Alignments of Event Data and Process Models

Sebastiaan J. van Zelst, Alfredo Bolt, and Boudewijn F. van Dongen

Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{s.j.v.zelst, a.bolt, b.f.v.dongen}@tue.nl

Abstract. The aim of conformance checking is to assess whether a process model and event data, recorded in an event log, conform to each other. In recent years, alignments have proven extremely useful for calculating conformance statistics. Computing optimal alignments is equivalent to solving a shortest path problem on the state space of the synchronous product net of a process model and event data. State-of-the-art alignment based conformance checking implementations exploit the A^*-algorithm, a heuristic search method for shortest path problems, and include a wide range of parameters that likely influence their performance. In previous work, we presented a preliminary and exploratory analysis of the effect of these parameters. This paper extends the aforementioned work by means of large-scale statistically-sound experiments that describe the effects and trends of these parameters for different populations of process models. Our results show that, indeed, there exist parameter configurations that have a significant positive impact on alignment computation efficiency.

Keywords: Process Mining, Conformance Checking, Alignments

1 Introduction

Most organizations, in a variety of fields such as banking, insurance and healthcare, execute several different (business) processes. Modern information systems allow us to track, store and retrieve data related to the execution of such processes, in the form of *event logs*. Often, an organization has a global idea, or even a formal specification, of how the process is supposed to be executed. In other cases, laws and legislations dictate explicitly in what way a process is ought to be executed. Hence, it is in the company's interest to assess to what degree the execution of their processes is in line with the corresponding specification.

Conformance checking techniques, originating from the field of *process mining* [3], aim at solving the aforementioned problem. Conformance checking techniques allow us to quantify to what degree the actual execution of a process, as recorded in an event log, conforms to a corresponding process specification. Recently, *alignments* were introduced [4,6], which rapidly developed into the de-facto standard in conformance checking. The major advantage of alignments w.r.t. alternative conformance checking techniques, is the fact that deviations and/or mismatches are quantified in an exact, unambiguous manner.

When computing alignments, we convert a given process model, together with the behaviour recorded in an event log, into a *synchronous product net* and subsequently solve a shortest path problem on its state space. Typically, the well known A^* algorithm [8] is used as an underlying solution to the shortest path problem. However, several (in some cases alignment-specific) parametrization options are defined and applied on top of the basic A^* solution.

In previous work [19] we presented a preliminary and exploratory analysis of the effect of several parameters on the conventional alignment algorithm. In this paper, we extend the aforementioned work further, by assessing a significantly larger population of process models. We specifically focus on those parametrizations of the basic approach that, in our previous work, have shown to have a positive impact on the algorithm's overall performance. Moreover, we present a concise algorithmic description of alignment calculation which explicitly includes these parameters. Our experiments confirm that, indeed, the parameters studied enable us to increase the overall efficiency of computing alignments.

The remainder of this paper is organized as follows. In Section 2, we present preliminaries. In Section 3, we present the basic A^*-based alignment algorithm. In Section 4, we evaluate the proposed parametrization. In Section 5, we discuss related work. Section 6 concludes the paper.

2 Preliminaries

In this section we present preliminary concepts needed for a basic understanding of the paper. We assume the reader to be reasonably familiar with concepts such as functions, sets, bags, sequences and Petri nets.

2.1 Sets, Tuples, Sequences and Matrices

We denote the set of all possible multisets over set X as $\mathcal{B}(X)$. We denote the set of all possible sequences over set X as X^*. The empty sequence is denoted $\langle\rangle$. Concatenation of sequences σ_1 and σ_2 is denoted as $\sigma_1 \cdot \sigma_2$. Given tuple $\overline{x} = (x_1, x_2, ..., x_n)$ of Cartesian product $X_1 \times X_2 \times ... \times X_n$, we define $\pi_i(\overline{x}) = x_i$ for all $i \in \{1, 2, ..., n\}$. In case we have a tuple $\overline{t} \in X \times Y \times Z$, we have $\pi_1(\overline{t}) \in X$, $\pi_2(\overline{t}) \in Y$ and $\pi_3(\overline{t}) \in Z$. We overload notation and extend projection to sequences, i.e. given sequence $\overline{\sigma} \in (X_1 \times X_2 \times ... \times X_n)^*$ of length k where $\overline{\sigma} = \langle(x_1^1, x_2^1, ..., x_n^1), (x_1^2, x_2^2, ..., x_n^2), ..., (x_1^k, x_2^k, ..., x_n^k)\rangle$, we have $\pi_i(\overline{\sigma}) = \langle x_i^1, x_i^2, ..., x_i^k\rangle \in X_i^*$, for all $i \in \{1, 2, ..., n\}$. Given a sequence $\sigma = \langle x_1, x_2, ..., x_k\rangle \in X^*$ and a function $f : X \to Y$, we define $\pi^f \colon X^* \to Y^*$ with $\pi^f(\sigma) = \langle f(x_1), f(x_2), ..., f(x_k)\rangle$. Given $Y \subseteq X$ we define $\downarrow_Y \colon X^* \to Y^*$ recursively with $\downarrow_Y (\langle\rangle) = \langle\rangle$ and $\downarrow_Y (\langle x\rangle \cdot \sigma) = \langle x\rangle \cdot \downarrow_Y (\sigma)$ if $x \in Y$ and $\downarrow_Y (\langle x\rangle \cdot \sigma) = \downarrow_Y (\sigma)$ if $x \notin Y$. We write $\sigma_{\downarrow_Y}$ for $\downarrow_Y (\sigma)$. Given an $m \times n$ matrix $\mathbf{A}$, i.e. $\mathbf{A}$ has m rows and n columns, $\mathbf{A}_{i,j}$ represents the element on row i and

Table 1: Example event log fragment.

Event-id	**Case-id**	**Activity**	**Resource**	**Time-stamp**
...	...	...	...	...
12474	*1215*	*test (e)*	*John*	*2017-11-14 14:45*
12475	*1216*	*register defect (a)*	*Abdul*	*2017-11-14 15:12*
12476	*1215*	*order replacement (h)*	*Maggy*	*2017-11-14 15:14*
12477	*1216*	*repair (b)*	*Maggy*	*2017-11-14 15:31*
12478	*1216*	*inform client (d)*	*Harry*	*2017-11-14 15:40*
12479	*1216*	*test (e)*	*Maggy*	*2017-11-14 14:49*
12480	*1216*	*return to client (g)*	*Maggy*	*2017-11-14 16:01*
12481	*1217*	*register defect (a)*	*John*	*2017-11-14 16:03*
...	...	...	...	...

column j $(1 \leq i \leq m, 1 \leq j \leq n)$. $\mathbf{A}^\intercal$ represents the transpose of $\mathbf{A}$. $\vec{x} \in \mathbb{R}^n$ denotes a column vector of length n, whereas $\vec{x}^\intercal$ represents a row vector.

2.2 Event Logs and Petri Nets

The execution of business processes within a company generates traces of event data in its supporting information systems. We are able to extract such data from these information systems, describing, for specific instances of the process, e.g. an insurance claim, what sequence of activities has been performed over time. We refer to a collection of such event data as an *event log*. A sequence of executed process activities, related to a process instance, is referred to as a *trace*. Consider Table 1, which depicts a simplified view of an event log.

The event log describes the execution of activities related to a phone repair process. For example, consider all events related to case 1216, i.e. a new *defect is registered* by *Abdul*, *Maggy* subsequently *repairs* the phone, *Harry informs* the corresponding *client*, etc. When we consider all events executed for case *1216*, ordered by time-stamp, we observe that it generates the sequence of activities $\langle a, b, d, e, g \rangle$ (note that we use short-hand activity names for simplicity). Such projection, i.e. merely focussing on the sequential ordering of activities, is also referred to as the *control-flow perspective*. In the remainder of this paper we assume this perspective, which we formalize in Definition 1.

Definition 1 (Event Log). *Let $\mathcal{A}$ denote the universe of activities. An event log L is a multiset of sequences over activities in $\mathcal{A}$, i.e. $L \in \mathcal{B}(\mathcal{A}^*)$.*

Each sequence $\sigma \in L$ describes a *trace*, which potentially occurs multiple times in an event log.

Event logs describe the actual execution of business processes, i.e. as recorded within a company's information system. Process models on the other hand allow us to describe the intended behaviour of a process. In this paper we use Petri nets [15] as a process modelling notation. An example Petri net is depicted in Fig. 1. The Petri net, like the event log, describes a process related to phone repair. It dictates that first a *register defect* activity needs to be performed. After this, a *repair* needs to be performed. Such repair is alternatively *outsourced*. In

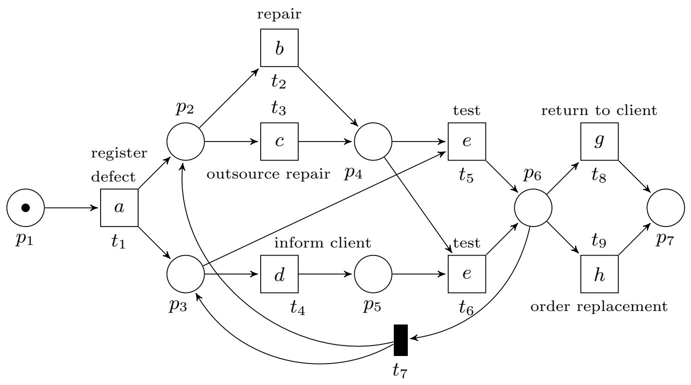


Fig. 1: Example labelled Petri net $N_1 = (P_1, T_1, F_1, \lambda_1)$ describing a (simplified) phone-repair process.

parallel with the repair, the client is optionally *informed* about the status of the repair. In any case, after the repair is completed, the repaired phone is *tested*. If the test succeeds the phone is *returned to the client*. If the test fails, either a new repair is performed, or, a *replacement* is ordered.

A Petri net is simply a bipartite graph with a set of vertices called *places* and a set of vertices called *transitions*. Places, depicted as circles, represent the state of the process. Transitions, depicted as boxes, represent executable actions, i.e. activities. A place can be *marked* by one ore more *tokens* which are graphically represented by black dots, depicted inside of the place, e.g. place p_1 is marked with one token in Fig. 1. If all places connected to a transition, by means of an *ingoing arc into the transition*, contain a token, we are able to fire the transition, i.e. equivalent with executing the activity represented by the transition. In such case, the transition consumes a token for each incoming arc, and produces a token for each of its outgoing arcs, e.g. transition t_1 is enabled in Fig. 1. After firing transition t_1, the token in p_1 is removed and both place p_2 and place p_3 contain a token. In this paper we assume that each transition has an associated (possibly unobservable) label which represents the corresponding activity, e.g. the label of transition t_1 is *register defect* (or simply a in short-hand notation) whereas transition t_7 is unobservable.

Definition 2 (Labelled Petri net). *Let P denote a set of places, let T denote a set of transitions and let $F \subseteq (P \times T) \cup (T \times P)$ denote the flow relation. Let Σ denote the universe of labels, let $\tau \notin \Sigma$ denote the unobservable label and let $\lambda \colon T \to \Sigma \cup \{\tau\}$. A labelled Petri net is a quadruple $N = (P, T, F, \lambda)$.*

Observe that N_1 in Fig. 1, in terms of Definition 2, is described as $N_1 = (P_1 = \{p_1, ..., p_7\}, T_1 = \{t_1, t_2, ..., t_9\}, F_1 = \{(p_1, t_1), ..., (t_9, p_7)\}, \lambda_1 = \{\lambda_1(t_1) = a, \lambda_1(t_2) = b, ..., \lambda_1(t_9) = h\})$. Additionally observe that $\lambda_1(t_7) = \tau$, which is graphically visualized by the black solid fill of transition t_7.

Given an element $x \in P \cup T$, we define $\bullet x = \{y \in P \cup T \mid (y, x) \in F\}$ and $x \bullet = \{y \in P \cup T \mid (x, y) \in F\}$. A marking m of Petri net $N = (P, T, F, \lambda)$ is a multiset of P, i.e. $m \in \mathcal{B}(P)$. Given such marking and a Petri net, we write (N, m), which we refer to as a *marked net*. The initial marking of N is denoted as m_i, and thus, (N, m_i) represents the *initially marked net*. When a transition t is enabled in a marking m, i.e. $\forall p \in \bullet t(m(p) > 0)$, we write $(N, m)[t\rangle$, e.g. $(N_1, [p_1])[t_1\rangle$. Firing an enabled transition t in marking m, yielding $m' = (m - \bullet t) \uplus t\bullet$, is written as $(N, m) \xrightarrow{t} (N, m')$. If firing a sequence of transitions $\sigma = \langle t_1, t_2, ..., t_n \rangle \in T^*$, starts in marking m and yields marking m', i.e. $(N, m) \xrightarrow{t_1} (N, m_1) \xrightarrow{t_2} \dots (N, m_{n-1}) \xrightarrow{t_n} (N, m')$, we write $(N, m) \xrightarrow{\sigma} (N, m')$. The set of all reachable markings from marking m is denoted $\mathcal{R}(N, m) = \{m' \subseteq \mathcal{B}(P) \mid \exists \sigma \in T^*((N, m) \xrightarrow{\sigma} (N, m'))\}$. Given a designated marking m and target marking m', we let $\mathcal{L}(N, m, m') = \{\sigma \in T^* \mid (N, m) \xrightarrow{\sigma} (N, m')\}$. For example, $\langle t_1, t_2, t_5, t_8 \rangle \in \mathcal{L}(N_1, [p_1], [p_7])$. In case we are interested in the sequence of activities described by a firing sequence σ we apply $(\pi^{\lambda}(\sigma))_{\downarrow \Sigma}$, e.g. $(\pi^{\lambda_1}(\langle t_1, t_2, t_5, t_7, t_3, t_4, t_6, t_8 \rangle))_{\downarrow \Sigma} = \langle a, b, e, c, d, e, g \rangle$.

In the remainder, we let $\mathbf{A}$ denote the incidence matrix of a (labelled) Petri net $N = (P, T, F, \lambda)$. $\mathbf{A}$ is an $|T| \times |P|$ matrix where $\mathbf{A}_{i,j} = 1$ if $p_j \in t_i \bullet \setminus \bullet\, t_i$, $\mathbf{A}_{i,j} = -1$ if $p_j \in \bullet t_i \setminus t_i \bullet$ and $\mathbf{A}_{i,j} = 0$ otherwise. Further more, given some marking $m \in \mathcal{B}(P)$, we write $\vec{m}$ to denote a $|P|$-sized column vector with $\vec{m}(i) = m(p_i)$ for $1 \leq i \leq |P|$.

2.3 Alignments

The example event log and Petri net, presented in Table 1 and Fig. 1 respectively, are both related to a simplified phone repair process. In case of our example trace related to case *1216*, i.e. $\langle a, b, d, e, g \rangle$, it is easy to see that there exists a $\sigma \in T_1^*$ s.t. $(\pi^{\lambda_1}(\sigma))_{\downarrow \Sigma} = \langle a, b, d, e, g \rangle$, i.e. $\sigma = \langle t_1, t_2, t_4, t_6, t_8 \rangle$. In practice however, such direct mapping between observed activities and transition firings in a Petri net is often not possible. In some cases, activities are not executed whereas the model specifies they are supposed to. Similarly, in some cases we observe activities that according to the model are not possible, at least at that specific point within the trace. Such mismatches are for example caused by employees deviating from the process as specified, e.g. activities are executed twice or mandatory activities are skipped. Moreover, in many cases the process specification is not exactly in line with the actual execution of the process, i.e. some aspects of the process are overlooked when designing the process specification.

Alignments allow us to compare the behaviour recorded in an event log with the behaviour as described by a Petri net. Conceptually, an alignment represents a mapping between the activities observed in a trace $\sigma \in L$ and the execution of transitions in the Petri net. As an example, consider trace $\langle a, d, d, e, g \rangle$ and reconsider Petri net N_1 in Fig. 1. The trace does not fit the behaviour described by N_1. In Fig. 2 we present three different alignments of $\langle a, d, d, e, g \rangle$ and N_1. The first alignment, i.e. γ_1, specifies that the execution of the second d-activity is abundant, and, that an activity described by transition t_2 is missing. Similarly,

γ_1 :

a	d	d	$\gg$	e	g
t_1	t_4	$\gg$	t_2	t_6	t_8

γ_2 :

a	$\gg$	d	d	e	g
t_1	t_3	$\gg$	t_4	t_6	t_8

γ_3 :

a	$\gg$	d	$\gg$	$\gg$	$\gg$	d	e	g
t_1	t_2	t_4	t_6	t_7	t_3	t_4	t_6	t_8

Fig. 2: Example alignments for $\langle a, d, d, e, g \rangle$ and N_1.

the second alignment, i.e. γ_2, specifies that an activity described by transition t_3 is missing and that the first execution of the d-activity is abundant. Alignment γ_3 specifies that we are able to map each activity observed in the trace to a transition in the model, however, in such case, we at least miss activities described by transitions t_2, t_3 and t_6. Note that we do not miss an activity related to the execution of transition t_7, as this is an invisible transition.

When ignoring the $\gg$-symbols, the top row of each alignment equals the given trace, i.e. $\langle a, d, d, e, g \rangle$. The bottom row of each alignment, again when ignoring the $\gg$-symbols, represents a firing sequence in the language of N_1, i.e. $\sigma \in \mathcal{L}(N_1, [p_1], [p_7])$. Each individual column is referred to as a *move*. A move of the form $|\frac{a}{\gg}|$ is called a *log move* and represents behaviour observed in the trace that is not mapped onto the model. A move of the form $|\frac{\gg}{t}|$ is called a *model move* and represents behaviour that according to the model should have happened, yet was not observed at that position in the trace. A move of the form $|\frac{a}{t}|$ is called a *synchronous move* and represents an observed activity that is also described by the model at that position in the trace.

Definition 3 (Alignment). *Let $\sigma \in \mathcal{A}^*$ be a trace. Let $N = (P, T, F, \lambda)$ be a labelled Petri net and let $m_i, m_f \in \mathcal{B}(P)$ denote N's initial and final marking. Let $\gg \notin \mathcal{A} \cup T$. A sequence $\gamma \in ((\mathcal{A} \cup \{\gg\}) \times (T \cup \{\gg\}))^*$ is an alignment if:*

1. *$(\pi_1(\gamma))_{\downarrow_{\mathcal{A}}} = \sigma$; event part equals σ.*
2. *$(N, m_i) \xrightarrow{(\pi_2(\gamma))_{\downarrow_T}} (N, m_f)$; transition part is in the Petri net's language.*
3. *$\forall (a, t) \in \gamma (a \neq \gg \vee t \neq \gg)$; $(\gg, \gg)$ is not valid in an alignment.*

We let $\Gamma(N, \sigma, m_i, m_f)$ denote the set of all possible alignments of Petri net N and trace σ given markings m_i and m_f.

As exemplified by the three alignments of $\langle a, d, d, e, g \rangle$ and N_1, a multitude of alignments exists for a given trace and model. Hence, we need a means to be able to rank and compare alignments in such way that we are able to express our preference of an alignment w.r.t. other alignments. For example, in Fig. 2, we prefer γ_1 and γ_2 over γ_3, as we need less $\gg$-symbols to explain the observed behaviour in terms of the model. Computing such preference is performed by means of minimizing a cost function defined over the possible moves of an alignment. We present a general definition of such cost function in Definition 4, after which we provide a commonly used corresponding instantiation.

Definition 4 (Alignment Cost). *Let $\sigma \in \mathcal{A}^*$, let $N = (P, T, F, \lambda)$ be a labelled Petri net with $m_i, m_f \in \mathcal{B}(P)$, let $\gg \notin \mathcal{A} \cup T$ and let $c \colon (\mathcal{A} \cup \{\gg\}) \times (T \cup \{\gg\}) \to \mathbb{R}_{\geq 0}$. Given alignment $\gamma \in \Gamma(N, \sigma, m_i, m_f)$, the costs κ^c of γ, given* *move cost function c, is defined as $\kappa^c(\gamma) = \sum_{i=1}^{|\gamma|} c(\gamma(i))$. Finally, we let $\gamma_c^* \in$* $\textbf{arg min}_{\gamma \in \Gamma(N,\sigma,m_i,m_f)} \kappa^c(\gamma)$.

The cost of an alignment is defined as the sum of the cost of each move within the alignment, as specified by cost function c. If it is clear from context what cost function c is used, we omit it from the cost related notation, i.e. we write κ, γ^* etc. Note that γ^* is an alignment that has minimum costs amongst all alignments of a given model and trace, i.e. an *optimal alignment.* In general one can opt to use an arbitrary instantiation of c, however, a cost function that is used quite often is the following *unit-cost function*:

1. $c(a,t) = 0 \Leftrightarrow a \in \mathcal{A}, t \in T$ and $\lambda(t) = a$ or $a = \gg$ and $\lambda(t) = \tau$.[1]
2. $c(a,t) = \infty \Leftrightarrow a \in \mathcal{A}, t \in T$ and $\lambda(t) \neq a$
3. $c(a,t) = 1$ otherwise

Using the unit-cost function, γ_1 and γ_2 of Fig. 2 are both optimal for $\langle a,d,d,e,g \rangle$ and N_1, i.e. both alignments have cost 2. This shows that *optimality is not guaranteed to be unique for alignments.*

3 Computing Optimal Alignments

In this section we present the basic alignment computation algorithm. The algorithm, in essence, is a modification of the A^* algorithm [8], i.e. a general purpose shortest path algorithm. We do however incorporate alignment-specific optimizations within the algorithm that have shown to be beneficial for the overall performance of the approach, i.e. in terms of search efficiency and memory usage [19]. The algorithm applies a shortest path search on the state-space of the *synchronous product net* of the given trace and Petri net. As such, we first present how such synchronous product net is constructed, after which we present the alignment algorithm.

3.1 Constructing the Synchronous Product Net

To find an optimal alignment, i.e. an alignment that minimizes the cost function of choice, we solve a shortest path problem defined on the state space of the synchronous product net of the given trace and model. Such synchronous product net encodes the trace as a sequential Petri net and integrates it with the original model. As such, each transition in the synchronous product net represents a move within the resulting alignment. Executing such transition corresponds to putting the corresponding move in the alignment. Consider Fig. 3, which depicts the synchronous product net of trace $\langle a,d,d,e,g \rangle$ and example Petri net N_1.

The sequence of black transitions, depicted on the top of the synchronous product net represents the input trace, i.e. $\langle a,d,d,e,g \rangle$. The labels of these

[1] In some cases, if the absence of *token-generators* is not guaranteed, we use $c(a,t) = \epsilon$, where ϵ is a positive real number smaller than 1 and close to 0

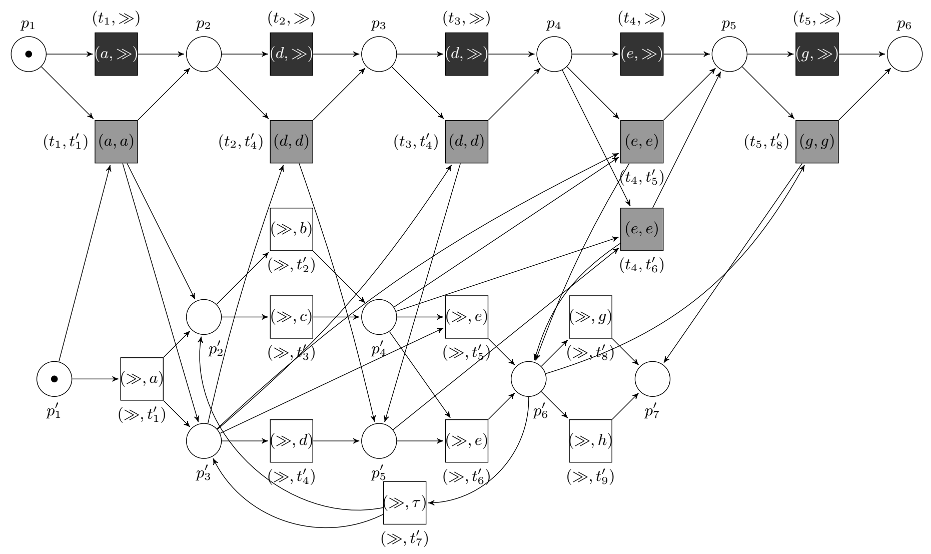


Fig. 3: Synchronous product net N_1^S of trace $\langle a, d, d, e, g \rangle$ and example Petri net N_1. Note that we have renamed elements of N_1 using a $'$-symbol, i.e. p'_1, t'_1 etc.

transitions represent log moves, e.g. transition $(t_1, \gg)$ has label $(a, \gg)$. Observe that, we are able to, from the initial marking $[p_1, p'_1]$, generate a firing sequence $\langle (a, \gg), (d, \gg), ..., (g, \gg) \rangle$ (projected onto labels) marking $[p'_1, p_6]$. Such firing sequence corresponds to a sequence of log moves which describe the given trace. The lower part of the synchronous product net represents Petri net N_1, however, the transition names represent model moves, e.g. transition $(\gg, t'_1)$ directly relates to a model move on t'_1. Observe that, using these transitions we are able to generate firing sequences of model moves that correspond to firing sequences that are in N_1's language. Finally, the middle (grey) transitions manipulate both the marking of the top part of the synchronous product net as the bottom part. Each of these transitions represents a synchronous move, e.g. consider transition (t_1, t'_1) representing a synchronous move of the first event of $\langle a, d, d, e, g \rangle$, i.e. representing activity a, and transition t'_1 (which in Fig. 1 is identified as t_1).

We formally define a synchronous product net as the product of a Petri net that represents the input trace (i.e., a trace net), together with the given process model. As such, we first define a *trace net*, after which we provide a general definition of the product of two Petri nets.

Definition 5 (Trace net). *Let $\sigma \in \mathcal{A}^*$ be a trace. We define the trace net of σ as a labelled Petri net $N = (P, T, F, \lambda)$, where:*

- $P = \{p_i \mid 1 \leq i \leq |\sigma| + 1\}$.
- $T = \{t_i \mid 1 \leq i \leq |\sigma|\}$.
- $F = \{(p_i, t_i) \mid 1 \leq i \leq |\sigma| \wedge p_i \in P \wedge t_i \in T\} \cup \{(t_i, p_{i+1}) \mid 1 \leq i \leq |\sigma| \wedge p_{i+1} \in P \wedge t_i \in T\}$.

– *$\lambda(t_i) = \sigma(i)$, for $1 \leq i \leq |\sigma|$.*

Given a trace σ we write N^σ to refer to the trace net of σ. We subsequently define the product of two arbitrary labelled Petri nets.

Definition 6 (Petri net Product). *Let $N = (P, T, F, \lambda)$ and $N' = (P', T', F', \lambda')$ be two Petri nets (where $P \cap P' = \emptyset$ and $T \cap T' = \emptyset$). The product of N and N', i.e. Petri net $N \otimes N' = (P^\otimes, T^\otimes, F^\otimes, \lambda^\otimes)$ where:*

– *$P^\otimes = P \cup P'$.*
– *$T^\otimes = (T \times \{\gg\}) \cup (\{\gg\} \times T') \cup \{(t, t') \in T \times T' \mid \lambda(t) = \lambda'(t')\}$*
– *$F^\otimes = \{(p,(t,t')) \in P^\otimes \times T^\otimes \mid (p,t) \in F \vee (p,t') \in F')\} \cup \{((t,t'),p) \in T^\otimes \times P^\otimes \mid (t,p) \in F \vee (t',p) \in F'\}$.*
– *$\lambda^\otimes \colon T^\otimes \to (\Sigma \cup \{\tau\} \cup \{\gg\}) \times (\Sigma \cup \{\tau\} \cup \{\gg\})$ (assuming $\gg \notin \Sigma \cup \{\tau\}$) where:*
  $\lambda^\otimes(t, \gg) = (\lambda(t), \gg)$ for $t \in T$
  $\lambda^\otimes(\gg, t') = (\gg, \lambda'(t'))$ for $t' \in T'$
  $\lambda^\otimes(t, t') = (\lambda(t), \lambda'(t'))$ for $t \in T, t' \in T'$.

A *synchronous product net* is defined as the product of a trace net N^σ and an arbitrary Petri net N, i.e. $N^\sigma \otimes N$. Assume we construct such synchronous product net $N^S = (P^S, T^S, F^S, \lambda^S)$ based on a trace net $N^\sigma = (P^\sigma, T^\sigma, F^\sigma, \lambda^\sigma)$ of trace σ and Petri net $N = (P, T, F, \lambda)$. Moreover, let $p_i \in P^\sigma$ with $\bullet p_i = \emptyset$, $p_f \in P^\sigma$ with $p_f \bullet = \emptyset$, and, let m_i, m_f denote a designated initial and final marking of N. Furthermore, let $m_i^S = m_i \uplus [p_i]$ and $m_f^S = m_f \uplus [p_f]$. Any firing sequence $\sigma' \in (T^S)^*$, s.t. $m_i^S \xrightarrow{\sigma'} m_f^S$ corresponds to an alignment of σ and N [6]. To be able to compute an alignment based on a synchronous product net, such firing sequence needs to exist. The problem of determining whether such sequence exists is known as the *reachability problem*, which is shown to be decidable [10,13]. However, within conformance checking, we assume that a reference model of a process is designed by a human business process analyst/designer. We therefore assume that a process model has a certain level of quality, e.g. the Petri net is a *sound workflow net* [1, Definition 7]. In context of alignment computation we therefore simply assume that a Petri net N, given initial marking m_i and final marking m_f is *easy sound*, i.e. $\mathcal{L}(N, m_i, m_f) \neq \emptyset$. A synchronous product net of a trace net and an easy sound Petri net is, by construction, easy sound, and thus guarantees reachability of its final marking.

To derive the actual move related to each transition in some firing sequence $\sigma' \in \mathcal{L}(N^S, m_i^S, m_f^S)$, we utilize the label function of the synchronous product net. In case we observe a transition of the form $(\gg, t)$, i.e. with $t \in T^\sigma$, we know it relates to a log move, which is obtained by applying $\lambda^S((t, \gg))$, e.g. $\lambda((t_1, \gg)) = (a, \gg)$ in Fig. 3. In case we observe a transition of the form $(\gg, t)$ $t \in T$, we know it relates to a model move, which is reflected by the transition name, i.e. we do not need to fetch the transition's label, e.g. $(\gg, t_1')$ in Fig. 3. A transition of the form $(t, t') \in T^S$, i.e. with $t \neq \gg$ and $t' \neq \gg$ corresponds to a synchronous move, which we translate into such move by applying $(\lambda^\sigma(t), t')$, e.g. $(\lambda^\sigma(t_1), t_1') = (a, t_1')$ in Fig. 3. Since we are able to map each transition in the synchronous product net onto a corresponding move, we are also able to deduce the move costs corresponding to any such transition present in the synchronous product net. *Therefore, in the remainder, given a synchronous product net $N^S = (P^S, T^S, F^S, \lambda^S)$, we assume the existence of a corresponding transition-based move cost function $c^S : T^S \to \mathbb{R}_{\geq 0}$ that maps each transition in the synchronous product net to the costs of the underlying move it represents.*

3.2 Searching for Optimal Alignments

In this section, we present the state-of-the art algorithm for optimal alignment computation. We first present an informal overview of the A^*-algorithm. Subsequently we describe how to exploit the marking equation for the purpose of heuristic estimation, after which we show how to limit the number of states enqueued during the search. Finally we present a concise corresponding algorithmic description.

Applying A^* Each transition in the synchronous product net corresponds to a move in an alignment, and moreover, to an arc in the state space of the synchronous product. Since each move/transition has an associated cost, we are able to assign the weight of each arc in the net's state space with the cost of the associated move. For example, observe Fig. 4, in which we depict a (small) part of the state-space of N_1^S (Fig. 3). The initial state of the state space, i.e. $[p_1, p_1']$, is depicted on the top-left. We are able to fire $(t_1, \gg)$, corresponding to a log-move $|\frac{a}{\gg}|$, yielding marking $[p_2, p_1']$. Similarly, in $[p_1, p_1']$, we are able to fire $(\gg, t_1')$, corresponding to model move $|\frac{\gg}{t_1}|$, yielding marking $[p_1, p_2', p_3']$. The order of firing these two transitions is irrelevant, i.e. $[p_1, p_1'] \xrightarrow{\langle (t_1,\gg),(\gg,t_1') \rangle} [p_2, p_2', p_3']$, and, $[p_1, p_1'] \xrightarrow{\langle (\gg,t_1'),(t_1,\gg) \rangle} [p_2, p_2', p_3']$.[2] Observe that we are also able to mark $[p_2, p_2', p_3']$ by firing (t_1, t_1') in marking $[p_1, p_1']$, corresponding to synchronous move $|\frac{a}{t_1'}|$. From $[p_2, p_2', p_3']$ we are able to fire $(t_2, \gg)$, (t_2, t_4'), $(\gg, t_2')$, $(\gg, t_3')$, and $(\gg, t_4')$ (not all of these transitions are explicitly visualized for the ease of readability/simplicity).

As indicated, each transition corresponds to a move, which, according to the corresponding cost function c^S has an associated cost. As such, the goal of finding an optimal alignment is equivalent to solving a shortest path problem on the state space of the synchronous product net [6]. Within the given shortest path problem, the initial marking of the given Petri net combined with the first place of the trace net defines the initial state (i.e. m_i^S). Similarly the target state is a combination of the given final marking of the model combined with the last place of the trace net (i.e. m_f^S).

Many algorithms exist that solve a shortest path problem on a weighted graph with a unique start vertex and a set of end vertices. In this paper we

[2] The label $[p_2, p_2', p_3']$ is not shown in Fig. 4, it corresponds to the state on the second row and second column.

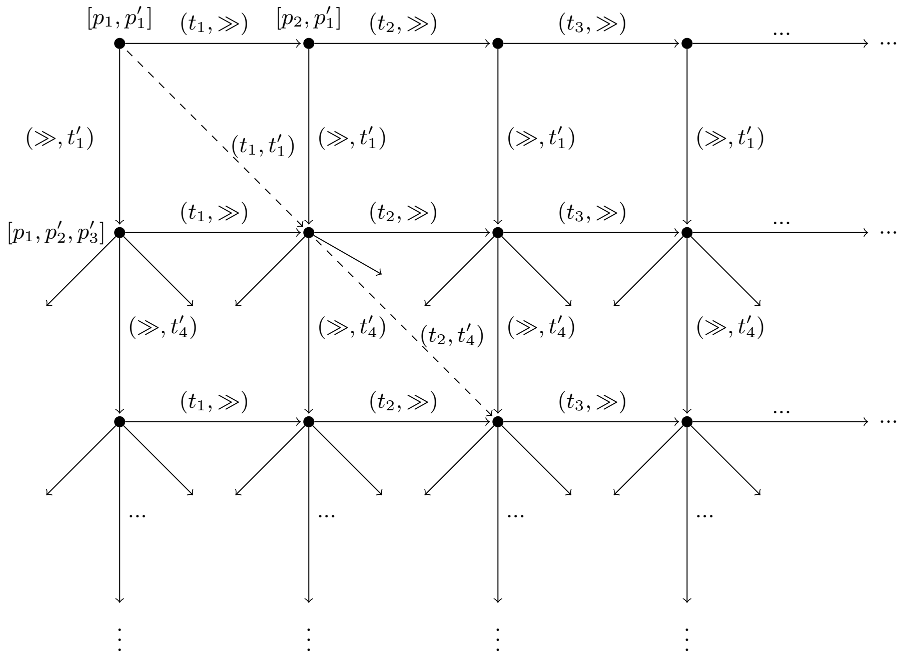


Fig. 4: Part of the state-space of the synchronous product net N_1^S shown in Fig. 3. Observe that in some markings, more transitions are enabled than we explicitly show here, e.g. in marking $[p_1, p_2', p_3']$, transitions $(\gg, t_2')$ and $(\gg, t_3')$are additionally enabled.

predominantly focus on the A^* algorithm [8]. The A^* algorithm is an *informed search algorithm*, i.e. it tries to incorporate specific knowledge of the graph within the search. In particular, it uses a *heuristic function* that approximates, for each vertex in the given graph, the expected remaining distance to the closest end vertex. The A^* algorithm is *admissible*, i.e. it guarantees to find a shortest path, if the heuristic always underestimates the actual distance to the/any final state. In case of computing optimal alignments based on the state space of the synchronous product net, markings of the synchronous product net represent vertices. Hence, we formally define a heuristic function on the basis of arbitrary labelled Petri nets, after which we provide an instantiation tied to synchronous product nets.

Definition 7 (Petri net based heuristic function). *Let $N = (P, T, F, \lambda)$ be a Petri net. A heuristic function h^N is a function $h^N : \mathcal{B}(P) \times \mathcal{B}(P) \to \mathbb{R}_{\geq 0}$.*

Using the previously defined heuristic, we are able to, given an initial marking m_i^S and final marking m_f^S, of a synchronous product net $N^S = (P^S, T^S, F^S, \lambda^S)$, apply the default A^* approach, which roughly performs the following steps.

1. Inspect marking m that minimizes $f(m) = g(m) + h^{N^S}(m, m_f^S)$, where $g(m)$ is the *actual distance* from m_i^S to m (note that $g(m_i^S) = 0$).
2. For each adjacent marking m', i.e. $\exists t \in T^S(m \xrightarrow{t} m')$, compute $h^{N^S}(m', m_f^S)$. Furthermore, $\forall t \in T^S(m \xrightarrow{t} m')$, we apply $g(m') \leftarrow \mathbf{min}(g(m'), g(m)+c^S(t))$ (initially $g(m) = \infty, \forall m \in \mathcal{R}(N, m_i^S) \setminus m_i^S$).

Initially, marking m_i^S is the only known marking with a g-value unequal to ∞, i.e. $g(m_i^S) = 0$. Thus, starting with m_i^S, we repeat the two aforementioned steps until either we end up at m_f^S, or, no more markings are to be assessed. Due to the easy-soundness assumption we are guaranteed to always arrive, at some point, at m_f^S. Moreover, admissibility implies that the first time we assess marking m_f^S, $g(m_f^S)$ represents the shortest path from m_i^S to m_f^S in terms of move costs, and thus corresponding alignment costs. In general, it is possible to visit a marking m multiple times in *step 2*, potentially leading to a lower $g(m)$-value. However, if a heuristic function is *consistent*, i.e. for markings m, m', m'' and transition $t \in T^S$ s.t. $(N^S, m) \xrightarrow{t} (N^S, m')$, we have $h^{N^S}(m, m'') \leq h^{N^S}(m', m'') + c^S(t)$, we are guaranteed that once we reach a vertex during the A^* search, we are not able to reach it using an alternative path with lower costs than the current path. Hence, in case the heuristic function used is consistent, we know that once we inspect a marking m in *step 1*, $g(m)$ is minimal. As a consequence, whenever we reach it again in *step 2*, we are allowed to ignore it.

Exploiting the State Equation In this paper we provide an instantiation of the heuristic function, i.e. h^{N^S}, that exploits the *state equation* of Petri nets, i.e. an algebraic expression of marking changes in a Petri net. Let $\vec{x}$ denote at $|T|$-sized column vector of integers, let m and m' denote two markings and let $\sigma \in T^*$ s.t. $(N, m) \xrightarrow{\sigma} (N, m')$ and let $\vec{m}$ and $\vec{m}'$ denote the corresponding $|P|$-sized marking column vectors. The state equation states that when we instantiate $\vec{x}$ as the *Parikh vector* of σ, i.e. if transition t_i occurs k times in σ then $\vec{x}(i) = k$, $\vec{x}$ is a solution to $\vec{m}' = \vec{m} + \mathbf{A}^\intercal \vec{x}$. The reverse does however not hold, i.e. if we find a solution to $\vec{m}' = \vec{m} + \mathbf{A}^\intercal \vec{x}$, such solution $\vec{x}$ is not necessarily a Parikh representation of a $\sigma' \in T^*$ s.t. $(N, m) \xrightarrow{\sigma'} (N, m')$.

Nonetheless, we utilize the state equation for the purpose of calculating a Petri net based heuristic function. Given a marking m and target marking m' within the synchronous product net, we try to find a solution to $\vec{m}' = \vec{m} + \mathbf{A}^\intercal \vec{x}$, where $\mathbf{A}$ and $\vec{x}$ are defined in terms of the synchronous product net. Moreover, such solution needs to minimize the corresponding alignment cost, i.e. recall that for $1 \leq i \leq |T^S|$, $\vec{x}(i)$ refers to a transition $t_i \in T^S$ which has an associated cost as defined by the transition-based move cost function $c^S(t_i)$.

Definition 8 (State equation based heuristic). *Let $\sigma \in \mathcal{A}^*$ be a trace and let $N = (P, T, F, \lambda)$ be a Petri net. Let $N^S = N^\sigma \otimes N = (P^S, T^S, F^S, \lambda^S)$ be the synchronous product net of σ and N. Let $\mathbf{A}$ denote the incidence matrix of N^S, let $m, m' \in \mathcal{B}(P^S)$, and let $\vec{m}, \vec{m}'$ be the corresponding $|P^S|$-sized vectors Let $c^S \colon T^S \to \mathbb{R}_{\geq 0}$ be the transition-based move cost function and let $\vec{c}$ denote* *a corresponding $|T^S|$-sized vector with $\vec{c}(i) = c^S(t_i)$ (for $t_i \in T^S$). Let $\vec{x} \in \mathbb{R}_{\geq 0}^{|T^S|}$ be a $|T^S|$-sized vector. We instantiate h^{N^S} (Definition 7) with $h^{N^S}(m, m') = \infty$ if no solution exists to $\vec{m}' = \vec{m} + \mathbf{A}^\intercal \vec{x}$, and otherwise:*

$$\boldsymbol{min}(\vec{c}^\intercal \vec{x} \mid \vec{m}' = \vec{m} + \mathbf{A}^\intercal \vec{x})$$

Observe that we define $\vec{x}$ as a vector containing non-negative real valued numbers ($\mathbb{R}_{\geq 0}$) rather than naturals ($\mathbb{N}$). Note that this potentially leads to fractional values in $\vec{x}$. However, since we aim at underestimating the true distance to the final marking this is acceptable, i.e. the vector does not need to correspond to an actual firing sequence. We are thus able to compute the state equation based heuristic by formulating and subsequently solving it as either a Linear Programming- (LP) or an Integer Linear Programming (ILP) problem [17].[3] Observe that, in case no solution to the (I)LP exists, we simply assign a value of ∞ to the heuristic. Since an (I)LP solution is always smaller or equal to the true costs of reaching target marking m' from marking m, the state equation based heuristic is admissible. As shown in [6], the heuristic is also *consistent*.

In step 2 of the A^* approach, we compute the heuristic $h^{N^S}(m', m_f^S)$ as defined in Definition 8 by solving an (Integer) Linear Programming problem. Observe that, as exemplified earlier, there are often multiple ways to arrive at a certain marking within the state space of the synchronous product net. To avoid solving the same (I)LP multiple times, once we have computed $h^{N^S}(m', m_f^S)$ we are able to store the solution value for m' in a temporary cache, and, remove it when we fetch m' in step 1. However, specifically in case of solving an Integer Linear Programming problem, computing the $h^{N^S}(m', m_f^S)$ is potentially time consuming. As it turns out, in some cases, the solution vector $\vec{x}$ of the (I)LP solved for marking m allows us to derive, for an adjacent marking m', i.e. $\exists t \in T^{N^S}(m \xrightarrow{t} m')$, an exact value for $h^{N^S}(m', m_f^S)$.

Conceptually, given that we assess some marking m, this works as follows. When we compute $h^{N^S}(m, m_f^S)$, given that it is not equal to ∞, we obtain an associated solution vector $\vec{x}$. Such vector essentially describes the number of times a transition is ought to be fired to reach m_f from m, even though there does not necessarily exists a corresponding firing sequence containing the exact number of transition firings as described by $\vec{x}$. Assume that from m we are able to traverse an edge related to firing a transition t_i, for which $\vec{x}(i) \geq 1$, yielding marking m'. In such case, we are guaranteed, as we show in Proposition 1, that the solution value for $h^{N^S}(m', m_f^S)$ equals $h^{N^S}(m, m_f^S) - c^S(t_i)$, i.e. we are able to subtract the cost of the move represented by t_i from $h^{N^S}(m, m_f^S)$. Even in the case that $\vec{x}(i) < 1$, we are able to devise a lower bound for the value of $h^{N^S}(m', m_f^S)$, as we show in Proposition 2.

Proposition 1 (State based heuristic provides exact solution). *Let $\mathbf{A}$ denote the incidence matrix of a synchronous product net $N^S = (P^S, T^S, F^S, \lambda^S)$*

[3] In case we solve an ILP, we enforce $\vec{x} \in \mathbb{N}^{|T^S|}$.

*and let $m, m_f \in \mathcal{B}(P^S)$. Let $c^S \colon T^S \to \mathbb{R}_{\geq 0}$ be the transition-based move cost function and let $\vec{c} \in \mathbb{R}_{\geq 0}^{|T^S|}$ with $\vec{c}(i) = c^S(t_i)$ for $t_i \in T^S$. Let $\vec{x}^* \in$ **arg min**$_{\vec{x} \in \mathbb{R}_{\geq 0}^{|T^S|}}(\vec{c}^\intercal \vec{x} \mid \vec{m_f} = \vec{m} + \mathbf{A}^\intercal \vec{x})$. Let $m' \in \mathcal{B}(P^S)$ and let $t_i \in T^S$ s.t. $(N^S, m) \xrightarrow{t_i} (N^S, m')$. If $\vec{x}^*(i) \geq 1$, then $\vec{c}^\intercal(\vec{x}^* - \vec{1}_{t_i}) = \boldsymbol{min}_{\vec{x} \in \mathbb{R}_{\geq 0}^{|T^S|}}(\vec{c}^\intercal \vec{x} \mid \vec{m_f} = \vec{m'} + \mathbf{A}^\intercal \vec{x})$.*

Proof. *Observe that, according to the state equation, $\vec{m'} = \vec{m} + \mathbf{A}^\intercal \vec{1}_{t_i}$, and thus, $\vec{m} = \vec{m'} - \mathbf{A}^\intercal \vec{1}_{t_i}$. From this, we deduce $\vec{m_f} = \vec{m} + \mathbf{A}^\intercal \vec{x}^* = \vec{m'} - \mathbf{A}^\intercal \vec{1}_{t_i} + \mathbf{A}^\intercal \vec{x}^* = \vec{m'} + \mathbf{A}^\intercal(\vec{x}^* - \vec{1}_{t_i})$, i.e. $\vec{x}^* - \vec{1}_{t_i}$ is a solution to $\vec{m_f} = \vec{m'} + \mathbf{A}^\intercal \vec{x}$.*

*Assume $\vec{c}^\intercal(\vec{x}^* - \vec{1}_{t_i}) > \boldsymbol{min}_{\vec{x} \in \mathbb{R}_{\geq 0}^{|T^S|}}(\vec{c}^\intercal \vec{x} \mid \vec{m_f} = \vec{m'} + \mathbf{A}^\intercal \vec{x})$, which implies that there exists an alternative minimal solution for $\vec{m_f} = \vec{m'} + \mathbf{A}^\intercal \vec{x}$, i.e. $\exists \vec{y} \in$ **arg min**$_{\vec{x} \in \mathbb{R}_{\geq 0}^{|T^S|}}(\vec{c}^\intercal \vec{x} \mid \vec{m_f} = \vec{m'} + \mathbf{A}^\intercal \vec{x})$ with $\vec{c}^\intercal \vec{y} < \vec{c}^\intercal(\vec{x}^* - \vec{1}_{t_i})$.*

Again by using the fact that $\vec{m'} = \vec{m} + \mathbf{A}^\intercal \vec{1}_{t_i}$, we observe that since $\vec{y}$ is a solution to $\vec{m_f} = \vec{m'} + \mathbf{A}^\intercal \vec{x}$, also $(\vec{y} + \vec{1}_{t_i})$ is a solution to $\vec{m_f} = \vec{m} + \mathbf{A}^\intercal \vec{x}$. This however contradicts minimality of $\vec{x}^*$ since $\vec{c}^\intercal \vec{y} < \vec{c}^\intercal(\vec{x}^* - \vec{1}_{t_i}) \implies \vec{c}^\intercal(\vec{y} + \vec{1}_{t_i}) < \vec{c}^\intercal \vec{x}^*$. □

Proposition 1 shows that if we compute a heuristic value for $h^{N^S}(m, m_f^S)$ formed by underlying variable assignment $\vec{x}^*$, then in case there exists some $t_i \in T^S$ with $\vec{x}^*(i) \geq 1$ and $m \xrightarrow{t_i} m'$, we are guaranteed that $h^{N^S}(m', m_f^S) = h^{N^S}(m, m_f^S) - \vec{c}(i) = h^{N^S}(m, m_f^S) - c^S(t_i)$. This effectively allows us to reduce the number of (I)LP's we need to solve. It is however also possible that there is some $t_j \in T^S$ with $\vec{x}^*(j) < 1$. In such case $\vec{x}^* - \vec{1}_{t_j}$ is not a solution to $\vec{m_f} = \vec{m'} + \mathbf{A}^\intercal \vec{x}$, it does however provide a lower bound on the actual heuristic value of m'.

Proposition 2 (State based heuristic provides an upper bound). *Let $\mathbf{A}$ denote the incidence matrix of a synchronous product net $N^S = (P^S, T^S, F^S, \lambda^S)$ and let $m, m_f \in \mathcal{B}(P^S)$. Let $c^S \colon T^S \to \mathbb{R}_{\geq 0}$ be the transition-based move cost function and let $\vec{c} \in \mathbb{R}_{\geq 0}^{|T^S|}$ with $\vec{c}(i) = c^S(t_i)$ for $t_i \in T^S$. Let $\vec{x}^* \in$ **arg min**$_{\vec{x} \in \mathbb{R}_{\geq 0}^{|T^S|}}(\vec{c}^\intercal \vec{x} \mid \vec{m_f} = \vec{m} + \mathbf{A}^\intercal \vec{x})$. Let $m' \in \mathcal{B}(P^S)$ and let $t_i \in T^S$ s.t. $(N^S, m) \xrightarrow{t_i} (N^S, m')$. If $\vec{x}^*(i) < 1$, then $\vec{c}^\intercal(\vec{x}^* - \vec{1}_{t_i}) \leq \boldsymbol{min}_{\vec{x} \in \mathbb{R}_{\geq 0}^{|T^S|}}(\vec{c}^\intercal \vec{x} \mid \vec{m_f} = \vec{m'} + \mathbf{A}^\intercal \vec{x})$.*

Proof. *Assume there is a minimal solution $\vec{y}$ to $\vec{m_f} = \vec{m'} + \mathbf{A}^\intercal \vec{x}$, i.e. $\exists \vec{y} \in$ **arg min**$_{\vec{x} \in \mathbb{R}_{\geq 0}^{|T^S|}}(\vec{c}^\intercal \vec{x} \mid \vec{m_f} = \vec{m'} + \mathbf{A}^\intercal \vec{x})$ s.t. $\vec{c}^\intercal \vec{y} < \vec{c}^\intercal(\vec{x}^* - \vec{1}_{t_i})$. Since $(\vec{y} + \vec{1}_{t_i})$ is a solution to $\vec{m_f} = \vec{m} + \mathbf{A}^\intercal \vec{x}$ (c.f. Proposition 1), this contradicts $\vec{x}^* \in$ **arg min**$_{\vec{x} \in \mathbb{R}_{\geq 0}^{|T^S|}}(\vec{c}^\intercal \vec{x} \mid \vec{m_f} = \vec{m} + \mathbf{A}^\intercal \vec{x})$ since $\vec{c}^\intercal(\vec{y} + \vec{1}_{t_i}) < \vec{c}^\intercal \vec{x}^*$.* □

If Proposition 2 applies, we know that $h^{N^S}(m', m_f^S) \geq \vec{c}^\intercal(\vec{x}^* - \vec{1}_{t_i})$. Thus, $\vec{c}^\intercal(\vec{x}^* - \vec{1}_{t_i})$ underestimates the true value of $h^{N^S}(m', m_f^S)$ and we write $\hat{h}^{N^S}(m', m_f^S) = \vec{c}^\intercal(\vec{x}^* - \vec{1}_{t_i})$. Whenever we derive $\hat{h}^{N^S}(m', m_f^S)$, we know $g(m')$, i.e. we compute $\hat{h}^{N^S}(m', m_f^S)$ in step 2 of the basic A^* approach. Thus, we are also able to derive an underestimating $f(m')$ value, i.e. $\hat{f}(m') = g(m') + \hat{h}^{N^S}(m', m_f^S)$. In practice this implies that instead of solving an (I)LP when we investigate a new marking, we just deduce the f-value, which is potentially an underestimate. In case it is an underestimate, we keep track of this, and whenever we, in step 1, inspect an element with a minimal underestimated f-value, we try to find an exact solution by solving an (I)LP. In such case it is possible that $f(m) > \hat{f}(m)$, in which we need to select a new marking in step 1 that minimizes the f-value.

Limiting Transition Ordering Reconsider the synchronous product net shown in Fig. 3 with initial marking $[p_1, p'_1]$. Recall that there are three firing sequences in the net to achieve marking $[p_2, p'_2, p'_3]$, i.e. $\langle(\gg, t'_1), (t_1, \gg)\rangle$, $\langle(t_1, \gg), (\gg, t'_1)\rangle$ and $\langle(t_1, t'_1)\rangle$. The cost associated with $\langle(t_1, t'_1)\rangle$ is 0 whereas the cost for $\langle(\gg, t'_1), (t_1, \gg)\rangle$ and $\langle(t_1, \gg), (\gg, t'_1)\rangle$ is 2. We observe that both possible permutations of the sequence containing $(t_1, \gg)$ and $(\gg, t'_1)$ have the same cost and are both part of a (sub-optimal) alignment.

In general, assume we have an alignment $\gamma \cdot \langle x, y\rangle \cdot \gamma' \in \Gamma(N, \sigma, m_i, m_f)$ s.t. x is a log move and y is a model move. Moreover, let t_x denote the transition in the underlying synchronous product related to x, and let t_y denote the transition related to move y. Since, by construction, $\bullet t_x \cap \bullet t_y = \emptyset$, $\bullet t_x \cap t_y \bullet = \emptyset$, $t_x \bullet \cap \bullet t_y = \emptyset$ and $t_x \bullet \cap t_y \bullet = \emptyset$, we trivially deduce that also $\gamma \cdot \langle y, x\rangle \cdot \gamma' \in \Gamma(N, \sigma, m_i, m_f)$. Additionally, we have $\kappa(\gamma \cdot \langle x, y\rangle \cdot \gamma') = \kappa(\gamma \cdot \langle x, y\rangle \cdot \gamma')$, i.e. alignment costs are order independent.

Hence, to find an optimal alignment we only need to traverse/inspect one specific permutation of such log/model move combinations, rather than all possible permutations. In step 2 of the basic A^* scheme, each enabled transition in marking m is investigated. However, we are able to limit this number of transitions by exploiting the previously mentioned property, i.e.:

- *Log move restriction*; If the transition leading to the current marking relates to a *model move* we only consider those transitions t that relate to a *model* or *synchronous move.*
- *Model move restriction*; If the transition leading to the current marking relates to a *log move* we only consider those transitions t that relate to a *log* or *synchronous move.*

In the first option we are not able to schedule a log move after a model move. The other way around is however possible, i.e. we are allowed to schedule a model move after a log move. The second option behaves exactly opposite, i.e. we are not allowed to schedule a model move after a log move. Note that during the search we either only apply log move restriction, or, model move restriction, i.e. these techniques cannot be mixed.

Algorithmic Description In Algorithm 1, we present the basic algorithm for optimal alignment computation using A^*, which additionally incorporates Proposition 1 and Proposition 2 together with model move restriction. The algorithm takes a trace net and a sequence as an input, and for both nets, expects an

Algorithm 1: A* (Alignments)

```
input  : $N^\sigma = (P^\sigma, T^\sigma, F^\sigma, \lambda^\sigma), m_i^\sigma, m_f^\sigma \in \mathcal{B}(P^\sigma), N = (P, T, F, \lambda), m_i, m_f \in \mathcal{B}(P)$
output: optimal alignment $\gamma^* \in \Gamma(N, \sigma, m'_i, m'_f)$
begin
1      $N^S = (P^S, T^S, F^S, \lambda^S) = N^\sigma \otimes N$;                 // create synchronous product
2      $m_i^S \leftarrow m_i^\sigma \uplus m_i^S$; $m_f^S \leftarrow m_f^\sigma \uplus m'_f$;        // create initial/final marking
3      $C \leftarrow \emptyset$;                                                 // initialize closed set
4      $X \leftarrow \{m_i^S\}$;                                                // initialize open set
5      $Y \leftarrow \emptyset$;                                         // initialize estimated heuristics
6      $p(m_i^S) = (\varnothing, \varnothing)$;                               // initialize predecessor function
7      $\forall m \in \mathcal{R}(N^S, m_i^S)\ g(m) \leftarrow \infty$;               // initialize cost so far function $g$
8      $g(m_i^S) \leftarrow 0$;                                  // initialize distance for initial marking
9      $f(m_i^S) \leftarrow h^{N^S}(m_i^S, m_f^S)$;                     // compute estimate for initial marking
10     while $|X| > 0$ do
11         $m \leftarrow$ arg min$_{m \in X} f(m)$;
12         if $m = m_f^S$ then
13             return alignment derived from $\langle t_1, \ldots, t_n \rangle$ where $t_n = \pi_1(p(m_f^S))$,
                 $t_{n-1} = \pi_1(p(\pi_2(p(m_f^S))))$, etc. until the initial marking is reached recursively;
14         if $m \in Y$ then
15             $Y \leftarrow Y \setminus \{m\}$ ;                                // remove estimated heuristic
16             if $h^{N^S}(m, m_f^S) > \hat{h}^{N^S}(m, m_f^S)$ then
17                 $f(m) \leftarrow g(m) + h^{N^S}(m, m_f^S)$;
18                 continue while;         // $m$ is not nessecarily minimizing $f$ any more
19         $C \leftarrow C \cup \{m\}$;                                        // add $m$ to the closed set
20         $X \leftarrow X \setminus \{m\}$;                                   // remove $m$ from the open set
21         $T' \leftarrow T^S$;
22         if $\pi_1(p(m)) = (t, \gg)$, where $t \in T^\sigma$ then
23             $T' \leftarrow T' \setminus (\{\gg\} \times T)$;              // model moves not allowed after log moves
24         forall $t \in T'$ s.t. $(N^S, m) \xrightarrow{t} (N^S, m')$ do
25             if $m' \notin C$ then
26                 if $g(m) + c^S(t) < g(m')$ then
27                     $g(m') \leftarrow g(m) + c^S(t)$;                    // update cost so far function
28                     $\hat{h}^{N^S}(m', m_f^S) \leftarrow h^{N^S}(m, m_f^S) - c^S(t)$;         // estimate heuristic
29                     $f(m') \leftarrow g(m') + \hat{h}^{N^S}(m', m_f^S)$;
30                     if $\hat{h}^{N^S}(m', m_f^S)$ is not exact then
31                         $Y \leftarrow Y \cup \{m'\}$;       // add $m'$ to the estimated heuristics set
32                     $X \leftarrow X \cup \{m'\}$;                               // add $m'$ to the open set
33                     $p(m') \leftarrow (t, m)$;                        // update predecessor function
34     return failure;
```

initial and final marking. In line 1 and line 2 we construct the synchronous product net and corresponding initial- and final marking. Subsequently, in lines 3-5, we initialize the closed set C, open set X, and estimated heuristic set Y. Since the heuristic is consistent, whenever we investigate a marking, we know that the f-value for such marking no longer changes. Hence, the closed set C contains all markings that we have already visited, i.e. for which we have a corresponding final f-value. Within X we maintain all markings inspected at some point, i.e. their g-value is known, and their h-value is either exact or estimated. In Y we keep track of all markings with an underestimating heuristic. In line 6 we initialize pointer function p, which allows us to reconstruct the actual alignment once we reach m_f^S. As long as X contains markings, we select one of the markings having a minimal f-value (line 11). In case the new marking equals m_f^S we construct, using pointer-structure $p(m_f^S)$, the alignment. If the marking is not equal to m_f^S, we, in line 14, check if the corresponding f-value is exact or not. In case it is not, and, the exact $h^{N^S}(m, m_f^S)$ value is exceeding the estimate, we recalculate the marking's f-measure and go back to line 11. In any other case, we proceed by storing marking m in the closed set C and by removing it from X. In lines 21-23 we apply model move restriction. Note that it is trivial to alter the code in these lines in order to apply log move restriction. Finally, we fire each transition $t \in T'$ s.t. $(N^S, m) \xrightarrow{t} (N^S, m')$. If the newly reached marking m' is not in the closed set C, we add it to X and check whether we found a shorter path to reach it. If so we update its g-value and derive its h-value.[4]. If we actually compute an underestimate, i.e. an $\hat{h}$-value, we register this by adding m' to Y. Finally, we update the pointer-structure p for m'.

It is important to note that set X of Algorithm 1, is typically implemented as a queue. In its basic form, fetching the top element of the queue, i.e. as represented by $m \leftarrow \mathbf{arg\ min}_{m \in X} f(m)$ in line 11, yields any marking that minimizes the (potentially estimated) f-value. In general, a multitude of such markings exists. As observed in [19], minimizing the individual h^{N^S}-value (or $\hat{h}^{N^S}$) as a *second-order criterion*, enhances alignment computation efficiency significantly. We call such second-order criterion `DFS`, as it effectively reduces the estimated distance to a final marking. Within this paper we assume `DFS` is always applied as a second-order sorting criterion.

4 Evaluation

In this section, we evaluate the effect of different parameters in the search for an optimal alignment for different populations of Petri nets, measured in terms of *search efficiency* and *memory usage*. We measure the efficiency of the search using two metrics: the number of *visited states* and the number of *traversed arcs*. We measure the memory usage of the search using a single metric: the maximum number of *queued states*. Due to the scale of the experiments we have used multiple machines which does not allow us to compare computation time/memory usage directly in all cases. We do however provide such results in case they are comparable. In the remainder of this section we describe the experimental set-up and present a discussion of the obtained results.

[4] In practice, we cache h-values, thus we only derive a new h-value if we did not compute an exact h-value in an earlier stage

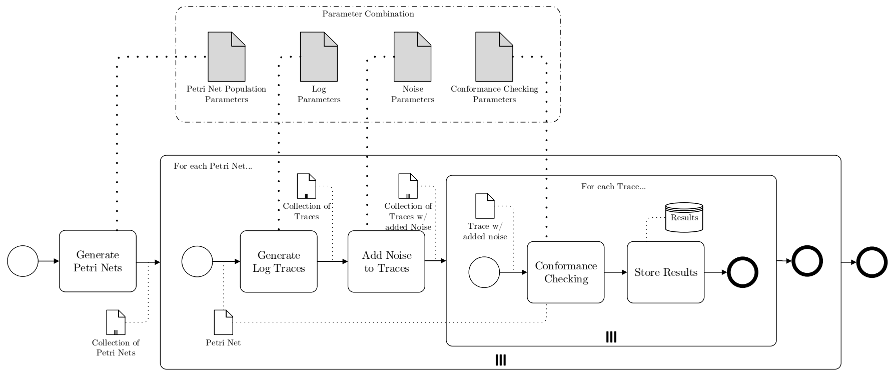


Fig. 5: Overview of the Experiment Design (modeled in BPMN notation).

4.1 Experimental Setup

The global workflow used in the experiment is illustrated in Fig. 5. For each combination of parameters, the following analysis steps are executed:

1. Generate a sample of (block-structured) Petri nets from a given population (defined in the "Petri net Population Parameters" object).
2. For each Petri net, generate a sample of traces that fit it (defined in the "Log Parameters" object).
3. For each generated trace, add an amount of noise (defined in the "Noise Parameters" object).
4. For each trace with added noise, check the conformance of the trace with respect to the Petri net (using the parameters defined in the "Conformance Checking Parameters" object) and store the results.

Note that the *blocks*, i.e. analysis steps, included in this high-level workflow are not necessarily bound to a concrete implementation. For example, one can use the approach presented in [9] to generate *process trees* that are translated into block-structured Petri nets, and also to generate event logs from them. Alternatively, any other approach that can generate Petri nets from defined populations can be used instead.

For the experiment, this high-level workflow was implemented as a scientific process mining workflow [7] in `RapidMiner` using building-blocks from the process mining extension `RapidProM` [5].

The generic workflow presented above enables many types of analysis that allow us to answer a wide variety of research questions related to the efficiency and memory usage of alignments. The experiment performed in this paper focuses on alignment parameters through the following research questions:

Q1 What is the global effect of approximation of the heuristic on the search efficiency and memory usage of alignments?

Q2 What is the effect of incorporating exact derived heuristics within the second-order queuing criterion on the search efficiency and memory usage of alignments?

Q3 What is the effect of transition restriction on the search efficiency and memory usage of alignments?

In order to be able to generalize the results, these effects are studied for different types of Petri nets with different levels of noise added to traces. Within the experiment, we considered 768 value combinations of seven parameters. The alignment related parameters and their values are described in Table 2, whereas the model related parameters are described in Table 3. For each parameter value combination, a collection of 64 Petri nets is generated, and 10 traces are generated from each Petri net. Then, after adding noise, each trace is aligned with the Petri net. In total, this resulted in computing roughly $500,000$ alignments within the experiment.

It is important to note that the different parameter combinations (e.g., heuristics, second-order queueing criterion) were not tested using the same set of Petri nets and traces. To the contrary, they were tested using independent samples of Petri nets and traces randomly obtained from the same populations of processes. In this way we avoid selection bias. Therefore, we consider the absolute differences and trends described in Section 4.2 as mere indications. For a proper analysis, in Section 4.3 we evaluated the differences in terms of statistical tests and not in terms of absolute differences.

4.2 Results

The results of the experiment are scoped in order to provide a straight-forward answer to the research questions proposed earlier.

Fig. 6 shows the results that relate to **Q1** (i.e., the effect of the *Heuristic* parameter). Fig. 6a shows the average number of traversed arcs (related to search efficiency) over increasing levels of loops for two different values of the *Heuristic* parameter: `LP with lower-bound estimation` and `LP without lower-bound estimation`. Here, using LP with lower-bound estimation refers to always deriving (a potentially approximate) heuristic based on a previously computed heuristic, i.e. applying both Proposition 1 and Proposition 2. Using LP without estimation refers to only deriving a heuristic when we are guaranteed that it is exact, i.e. only when Proposition 1 holds. When no exact heuristic can be derived an LP is solved immediately. We observe that for increasing levels of loops, the number of traversed arcs is relatively equal. This is as expected as the lower-bound does not affect the search efficiency directly, it merely allows us to, potentially postpone or even prohibit needless solve calls to the underlying LP-solver. Fig. 6b shows the average number of queued states (related to memory usage) over increasing levels of parallelism for the same two values of the *Heuristic* parameter: `LP with lower-bound estimation` and `LP without lower-bound estimation`. Again we observe that there is no clear setting that outperforms the other. Also in this case this is expected as using lower-bound

Table 2: Alignment parameters used in the experiment.

Parameter	***Type***	***Values***

Heuristic (h)	Categorical	`LP without lower-bound estimation`
		`LP with lower-bound estimation`

Second-order Queueing Criterion	Categorical	`DFS` (sort on minimized `h`-value) `DFS with certainty priority` (sort on minimized `h`-value)
Transition Restriction	Categorical	`MODEL` `LOG`

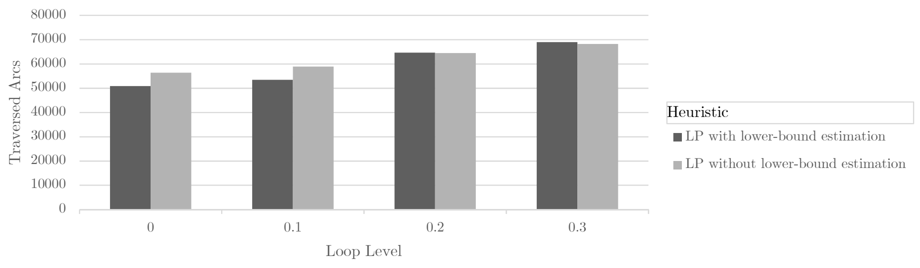
Table 3: Petri net (Pn) and Log generation parameters used in the experiment.

Parameter	***Type***	***Values***
Number of activities (Pn)	Numerical	25 50 75
Control-flow Charactersitic (Pn)	Categorical	`Parallelism` `Loops`
C-f Characteristic Level (Pn)	Numerical	0% 10% 20% 30%
Added Noise (Log)	Numerical	0% 20% 40% 60%

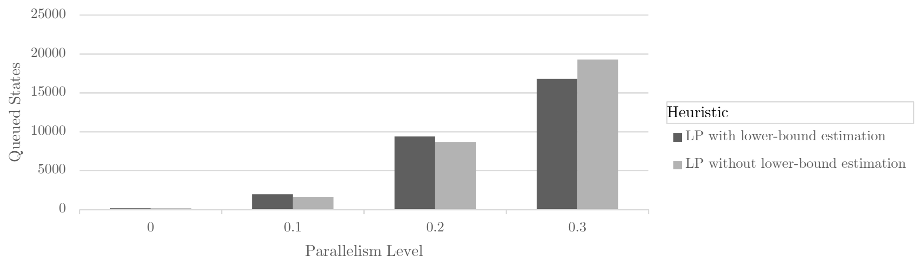
estimation does not affect the number of states put in the queue. In Fig. 6c we present computation time.[5] For these results, we expect using lower-bound estimation is beneficial in terms of computation time, i.e. we potentially solve less LP's. We do however not observe this. This is most likely explained by the fact that within the search, markings with an estimated heuristic end up in the top of the queue, and, LP's have to be solved anyway. Moreover, in such case, a marking is potentially reinserted on a lower position in the priority queue. Hence, we do not observe a clear impact on the global efficiency of the search by applying heuristic estimation.

The previous results seem to indicate that the effect of heuristic approximation is negligible. Observe however, that apart from using a `DFS`-based second order criterion, we arbitrarily select any marking on top of queue X. Fig. 7 shows the results that relate to **Q2** (i.e., the effect of heuristic approximation on the *Second-order Queueing Criterion* parameter). In particular, we compare arbitrary top-of-queue selection versus prioritizing markings with an exact heuristic w.r.t. estimated heuristics. Thus, if two markings m and m' have the same f and h value, yet the h value for m is exact whereas that of m' is not, we prioritize m over m'.

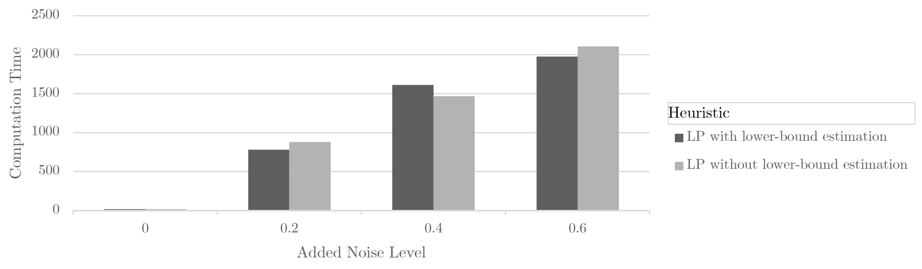
[5] Both experiments ran on the same machine in this instance.



(a) Effect on Traversed Arcs over increasing levels of loops.



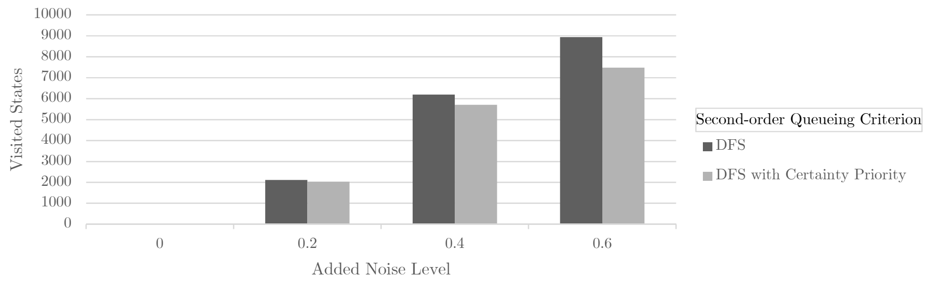
(b) Effect on Queued States over increasing levels of parallelism.



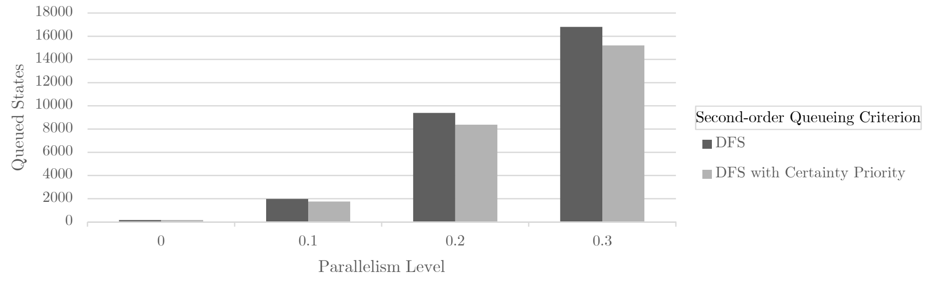
(c) Effect on computation time over increasing levels of noise.

Fig. 6: The effects of the *Heuristic* parameter

Fig. 7a shows the average number of visited states (related to search efficiency) over increasing levels of added noise for two different values of the *Second-order Queueing Criterion* parameter: `DFS` and `DFS with certainty priority`. We observe that `DFS with certainty priority` outperforms default `DFS`. This is most likely explained by the fact that in case a solution to the state equation, at some point, actually corresponds to a firing sequence, the search progresses extremely efficiently. In contract, in such case, using default `DFS` leads to unnecessary exploration of markings that do not lead to a final state. Fig. 6b shows the average number of queued states (related to memory usage) over increasing levels of parallelism for the same two values of the *Second-order Queueing Criterion* parameter: `DFS` and `DFS with certainty priority`. As expected we observe similar results to Fig. 7a.



(a) Effect on Visited States over increasing levels of added noise.



(b) Effect on Queued States over increasing levels of parallelism.

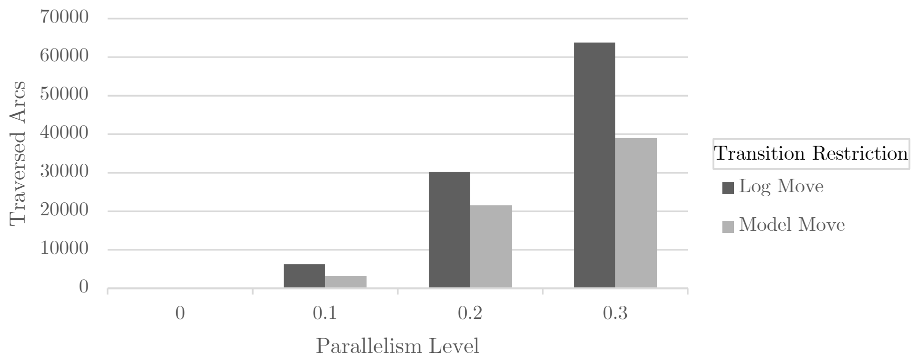
Fig. 7: The effects of the *Second-order Queueing Criterion* parameter

Finally, Fig. 8 shows the results that relate to **Q3** (i.e., the effect of transition restriction on the search efficiency). In Fig. 8a we present the effect of the different types of transition restriction w.r.t. the traversed arcs, for increasing levels of parallelism. Similarly, in Fig. 8b we present the effects on the number of queued states, for increasing levels of noise. Interestingly, except for noise level of 0.6, using model move restriction outperforms log move restriction. This is as expected since the model part of the synchronous product entails the most variety in terms of behaviour. Hence, limiting model move scheduling is expected to have a positive impact on the search performance.

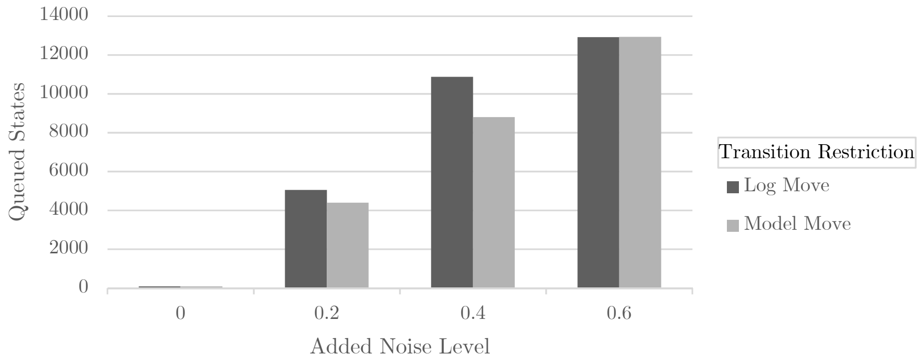
4.3 Statistical Analysis

In this section we analyse the statistical significance of the differences in terms of search efficiency and memory usage of several values of the three alignment parameters. We do not assume normal distributions of values, hence, the Kruskal-Wallis non-parametric test [11] is used. This a one-way significance rank-based test for multiple samples, designed to determine whether samples originate from the same population by observing their average ranks. This test does not assume that the samples are normally distributed.

Regarding research question 1 (**Q1**) we performed a Kruskal-Wallis test to assess the significance of the effect of the `Heuristic` parameter in the num-



(a) Effect on traversed arcs over increasing levels of parallelism.



(b) Effect on Queued States over increasing levels of noise.

Fig. 8: The effects of the *Transition Restriction* parameter

ber of traversed arcs, visited states and queued states with an alpha of 0.05. The test indicated that the effect of the `Heuristic` parameter is not statistically significant in any of the search efficiency or memory usage measurements ($p - values \approx 0.5$). The same applies for computation time. Regarding research question 2 (**Q2**) we performed a Kruskal-Wallis test to assess the significance of the effect of the `Second-order Queueing Criterion` parameter on the number of traversed arcs, visited states and queued states with an alpha of 0.05. The test indicated that the effect of the parameter on the tree measurements is statistically significant ($p - value < 0.001$). Regarding research question 3 (**Q3**) we performed a Kruskal-Wallis test to assess the significance of the effect of the `Transition Restriction` parameter in the number of traversed arcs, visited states and queued states with an alpha of 0.05. The test indicated that the effect of the parameter on the number of traversed arcs and queued states is statistically significant ($p - value < 0.001$ and $p - value = 0.008$ respectively) but the effect on the number of visited states is not statistically significant ($p - value = 0.1139$)

5 Related Work

A complete overview of process mining is outside the scope of this paper, hence we refer to [3]. Here, we primarily focus on related work in conformance checking.

Early work in conformance checking uses token-based replay [16]. The techniques try to replay a given trace in a model and add missing tokens if a transition is not able to fire. After replaying the full trace, remaining tokens are counted and a conformance statistic is computed based on missing and remaining tokens.

Alignments are introduced in [6]. The work proposes to transform a given Petri net and a trace from an event log into a synchronous product net, and, subsequently solve the shortest path problem on the corresponding state space. Its implementation in ProM may be regarded as the state-of-the-art technique in alignment computation and serves as a basis for this paper.

In [2,14] decomposition techniques are proposed together with computing alignments. The input model is split into smaller, transition-bordered, submodels for which local alignments are computed. Using decomposition techniques greatly enhances computation time. The downside of the techniques is the fact that they are capable to decide whether a trace fits the model or not, rather than quantifying to what degree a trace fits.

Recently approximation schemes for alignments, i.e. computation of near-optimal alignments, have been proposed in [18]. The techniques use a recursive partitioning scheme, based on the input traces, and solve multiple Integer Linear Programming problems. The techniques identify deviations between sets of transitions, rather than deviations between singletons (which is the case in [6]). Finally, alignments have also been defined as a planning problem [12] and have been recently studied in online settings [20].

6 Conclusion

In this paper we have presented and formalized an adapted version of the A^* search algorithm used in alignment computation. Within the algorithm we have integrated a number of parameters that, in previous work [19], have shown to be most promising in terms of algorithm efficiency. Based on large-scale experiments, we have assessed the impact of these parameters w.r.t. the algorithm's efficiency. Our results show that restricting the scheduling of model-move based transitions of the synchronous product net most prominently affects search efficiency. Moreover, the explicit prioritization of exactly derived heuristics seems to have a positive, yet less prominent, effect as well.

Future Work Within this work we have assessed, using large-scale experiments, the impact of several parameters on the efficiency of computing optimal alignments. However, several approximation schemes exist for A^*, e.g. using a scaling function within the heuristic. We plan to assess the impact of these approximation schemes on alignment computation as well. We also plan to examine the use of alternative informed search methods, e.g. *Iterative Deepening* A^*.

References

1. van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. Journal of Circuits, Systems, and Computers 8(1), 21–66 (1998)
2. van der Aalst, W.M.P.: Decomposing Petri Nets for Process Mining: A Generic Approach. Distributed and Parallel Databases 31(4), 471–507 (2013)
3. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)
4. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying History on Process Models for Conformance Checking and Performance Analysis. Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery 2(2), 182–192 (2012)
5. van der Aalst, W.M.P., Bolt, A., van Zelst, S.J.: RapidProM: Mine Your Processes and Not Just Your Data. CoRR abs/1703.03740 (2017)
6. Adriansyah, A.: Aligning Observed and Modeled Behavior. Ph.D. thesis, Eindhoven University of Technology, Dept. of Mathematics and Computer Science (Jul 2014)
7. Bolt, A., de Leoni, M., van der Aalst, W.M.P.: Scientific Workflows for Process Mining: Building Blocks, Scenarios, and Implementation. STTT 18(6), 607–628 (2016)
8. Hart, P.E., Nilsson, N.J., Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE SSC 4(2), 100–107 (1968)
9. Jouck, T. and Depaire, B.: PTandLogGenerator: A Generator for Artificial Event Data. In: BPM Demos. vol. 1789, pp. 23–27. CEUR-WS.org (2016)
10. Kosaraju, S.R.: Decidability of Reachability in Vector Addition Systems (Preliminary Version). In: ACM Theory of Computing. pp. 267–281. ACM (1982)
11. Kruskal, W.H., Wallis, W.A.: Use of Ranks in One-Criterion Variance Analysis. Journal of the American Statistical Association 47(260), 583–621 (1952)
12. de Leoni, M., Marrella, A.: Aligning Real Process Executions and Prescriptive Process Models through Automated Planning. Expert Syst. Appl. 82, 162–183 (2017)
13. Mayr, Ernst W.: An Algorithm for the General Petri Net Reachability Problem. SIAM J. Comput. 13(3), 441–460 (1984)
14. Munoz-Gama, J., Carmona, J., van der Aalst, W.M.P.: Single-Entry Single-Exit Decomposed Conformance Checking. Inf. Syst. 46, 102–122 (2014)
15. Murata, T.: Petri nets: Properties, Analysis and Applications. Proceedings of the IEEE 77(4), 541–580 (Apr 1989)
16. Rozinat, A., van der Aalst, W.M.P.: Conformance Checking of Processes Based on Monitoring Real Behavior. Inf. Syst. 33(1), 64–95 (2008)
17. Schrijver, A.: Theory of Linear and Integer Programming. Wiley-Interscience series in discrete mathematics and optimization, Wiley (1999)
18. Taymouri, F., Carmona, J.: A Recursive Paradigm for Aligning Observed Behavior of Large Structured Process Models. In: BPM 2016. LNCS, vol. 9850, pp. 197–214. Springer (2016)
19. van Zelst, S.J., Bolt, A., van Dongen, B.F.: Tuning Alingment Computation: An Experimental Evaluation. In: Proceedings of ATAED 2017. pp. 1–15 (2017)
20. van Zelst, S.J., Bolt, A., Hassani, M., van Dongen, B.F., van der Aalst, W.M.P.: Online Conformance Checking: Relating Event Streams to Process Models using Prefix-Alignments. IJDSA (Oct 2017)