Efficiently Computing Alignments Using the Extended Marking Equation

Boudewijn F. van Dongen(✉)

Eindhoven University of Technology, Eindhoven, The Netherlands
B.F.v.Dongen@tue.nl

Abstract. Conformance checking is considered to be anything where observed behaviour needs to be related to already modelled behaviour. Fundamental to conformance checking are alignments which provide a precise relation between a sequence of activities observed in an event log and a execution sequence of a model. However, computing alignments is a complex task, both in time and memory, especially when models contain large amounts of parallelism.

When computing alignments for Petri nets, (Integer) Linear Programming problems based on the marking equation are typically used to guide the search. Solving such problems is the main driver for the time complexity of alignments. In this paper, we adopt existing work in such a way that (a) the extended marking equation is used rather than the marking equation and (b) the number of linear problems that is solved is kept at a minimum.

To do so, we exploit fundamental properties of the Petri nets and we show that we are able to compute optimal alignments for models for which this was previously infeasible. Furthermore, using a large collection of benchmark models, we empirically show that we improve on the state-of-the-art in terms of time and memory complexity.

Keywords: Alignments · Conformance checking · Process Mining

1 Introduction

Conformance checking is considered to be anything where observed behaviour needs to be related to already modelled behaviour. Conformance checking is embedded in the larger contexts of Business Process Management and Process Mining [2], where conformance checking is typically used to compute metrics such as fitness, precision and generalization to quantify the relation between a log and a model.

Fundamental to conformance checking are alignments [3,4]. Alignments provide a precise relation between a sequence of activities observed in an event log and a execution sequence of a model. For each trace, this precise relation is expressed as a sequence of "moves". Such a move is either a "synchronous move" referring to the fact that the observed event in the trace corresponds directly to the execution of a transition in the model, a "log move" referring to the fact that the observed event has no corresponding transition in the model, or a "model move" referring to the fact that a transition occurred which was not observed in the trace.

Computing alignments is a complex task, both in time and memory, especially when models contain large amounts of parallelism and traces contain swapped events. Consider the example in Fig. 1. In this example, the model requires the process to finish with transitions A and B, while the trace shows them in the wrong order. The technique of [3,4] will, when reaching the indicated marking, have to investigate all interleavings of the parallel parts of the model inside the upper cloud before reaching the conclusion that there is a swap in the end.

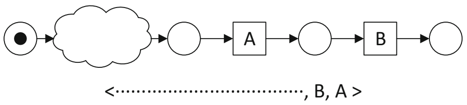

Fig. 1. A model ending in A, B and an example trace ending in B, A.

In this paper, we present a technique to efficiently compute alignments using the extended marking equation which, in the example above, would recognize the swapped activities. We present related work in Sect. 2 followed by some preliminaries in Sect. 3. In Sect. 4, we present alignments as well as our incremental technique for computing them. In Sect. 5 we compare our technique to existing approaches and discuss time and memory use before concluding the paper in Sect. 6.

2 Related Work

In [13], Rozinat et al. laid the foundation for conformance checking. They approached the problem by firing transitions in the model, regardless of available tokens and they kept track of missing and remaining tokens after completion of an observed trace. However, these techniques could not handle duplicate labels (identically labelled transition occurring at different locations in the model) or "invisible" transitions, i.e. τ-labelled transitions in the model purely for routing purposes.

As an improvement to token replay, alignments were introduced in [4]. The work proposes to transform a given Petri net and a trace from an event log into a synchronous product net, and solve the shortest path problem using $A^\star$ [6] on the its reachability graph. This graph may be considerable in size as it is worst-case exponential in the size of the synchronous product, in some cases, the proposed algorithm has to investigate the entire graph despite of tweaks identified in [16].

To mitigate these complexity issues, [1,10] proposes decomposition techniques for computing alignments. While this leads to shorter processing times, these techniques cannot always guarantee optimality of the alignment returned. Instead, they focus on giving bounds on the cost of the alignment. In [7], the authors provide techniques to identify misconformance based on sets of labels. The result is not an alignment.

Recently approximation schemes for alignments, i.e. computation of near-optimal alignments, have been proposed in [14]. The techniques use a recursive partitioning scheme, based on the input traces, and solve multiple Integer Linear Programming problems. In this paper, we build on the ideas outlined in [14] and we combine them with the state-of-the-art to compute optimal alignments for large process models.

Several techniques exist to compute alignments. Planner-based techniques [5] are available for safe Petri nets. For safe, acyclic models, constraint satisfaction [8] can be used. When not using a model, but its reachability graph as input, automata matching [12] can be applied. The models in this paper are safe. However, they are cyclic and, due to parallelism, the reachability graphs cannot be computed in reasonable time.

Finally, in [15] the authors provide a pre-processing mechanism to reduce the complexity of the alignment problem for structured input. This can also be applied in the context of this paper.

3 Preliminaries

In the context of conformance checking, we generally assume that there is a global set of activities A. An event log consists of a set of traces where each event in a trace refers to a specific activity execution. For sake of simplicity, we assume a log to be a set of sequences of activities and we also assume all sets to be totally ordered so that, when translating sets to vectors and vice versa, the translation respects the ordering.

Event Logs. An event log $L \subseteq A*$ is a set of sequences over A and $\sigma \in L$ a trace. We use $\sigma = \langle a, b, c \rangle$ to denote the fact that σ is a sequence of three activities $a, b, c \in A$. With $\sigma(i)$ we denote the i^{th} element of σ. **Petri nets.** Petri nets provide a modelling language for modelling business processes. Petri nets consist of places and transitions and exact semantics are provided to describe the behaviour of a Petri net [11]. In this paper, transitions can be labelled with activities or be silent, i.e. their execution would not be recorded in an event log.

We define $PN = (P, T, F, \lambda)$ as a labelled Petri net, where T is a set of transitions, P is a set of places, $F \subseteq P \times T \cup T \times P$ the flow relation and $\lambda : T \rightarrow A \cup \{\tau\}$ the labelling function, labelling each transition with an activity or τ indicating there is no label associated to the transition.

For $t \in T$ we use $\bullet t = \{p \in P \mid (p, t) \in F\}$ to denote the preset of t, i.e. the set of places serving as input of a transition. The postset of t is defined as $t\bullet = \{p \in P \mid (t, p) \in F\}$.

A marking m is a multiset of places, i.e., $m \in \mathcal{B}(P)$. A transition $t \in T$ is *enabled* in a marking m if and only if $\bullet t \leq m$. *Firing* transition t in m results in a new marking $m' = m - \bullet t + t\bullet$, i.e., tokens are removed from $\bullet t$ and added to $t\bullet$. A marking m' is *reachable* from m if there is a sequence of firings $\theta = \langle t_1, t_2 \ldots t_n \rangle$ that transforms m into m', denoted by $m[\theta\rangle m'$. We call θ a firing sequence. For marking m, we use $\vec{m}$ to represent them as column vector, where for each $p \in P$ holds that $\vec{m}(p) = m(p)$, i.e. the number of tokens in place p in marking m. Similarly, for firing sequence θ, we use $\vec{\theta}$ to denote the parikh vector of θ, i.e. a column vector over T with $\vec{\theta}(t) = \#_{0 \leq i < |\theta|}\theta(i) = t$, i.e. a vector counting the occurrences of each transition in θ. The set of all reachable markings of a Petri net given an initial marking is called the statespace.

Marking Equation. The structure of a Petri net can be translated into a so-called incidence matrix.

For Petri net $PN = (P, T, F, \lambda)$, The incidence matrix $\mathbf{C}$ is a matrix with $|P|$ rows and $|T|$ columns, such that for all $t \in T$ and $p \in P$ holds that

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

The incidence matrix can be used to mathematically relate markings through the so-called marking equation in the following way.

For a firing sequence $\theta \in T^*$ between two markings m_1 and m_2, i.e. $m_1[\theta\rangle m_2$, the marking equation states that: $\vec{m_1} + \mathbf{C} \cdot \vec{\theta} = \vec{m_2}$.

The marking equation implies that *if* there is a firing sequence θ from marking m_1 to marking m_2, *then* this equation holds. The inverse is not necessarily true, i.e. if $\vec{\theta}$ provides a solution to this equation, then it is not guaranteed that there exists a firing sequence θ corresponding to $\vec{\theta}$.

Where applicable, we mix sets and multisets by assuming a set is a multiset containing each element only once. As Petri nets in this paper are used to model processes, they have a clear initial and final marking.

$PM = (PN, m_i, m_f)$ is a process model consisting of a Petri net (P, T, F, λ) with initial marking m_i and final marking m_f. We assume there exists $\sigma \in T^*$ such that $m_i[\sigma\rangle m_f$, i.e. the final marking is reachable from the initial marking.

4 An Incremental Technique for Computing Alignments

Given a Petri net and its initial and final marking, the behaviour of the model can be seen as the set of possible firing sequences leading from the initial to the final marking. This set is, in many cases, infinite. An event log captures sequences of activity executions observed in practice and an *alignment* is a concept able to express the precise relation between this sequence and the model.

To explain the concept of an alignment, we use a so-called synchronous product net. These synchronous produces are specific types of nets, constructed from the combination of a model and a trace, where the trace is first converted into a trace model.

Definition 1 (Trace model). *Let $\sigma \in A^*$ be a trace. $TN = ((P, T, F, \lambda), m_i, m_f)$ is a process model called the trace model of σ if and only if:*

- $P = \{p_0 \dots p_{|\sigma|}\}$,
- $T = \{t_1 \dots t_{|\sigma|}\}$,
- $F = \{(p_i, t_{i+1}) \in P \times T \mid 0 \leq i < |\sigma|\} \cup \{(t_i, p_i) \in T \times P \mid 0 < i \leq |\sigma|\}$,

- *for all* $0 < i \leq |\sigma|$ *holds* $\lambda(t_i) = \sigma(i)$,
- $m_i = [p_0]$, *and*
- $m_f = [p_{|\sigma|}]$.

A trace model is a straightforward translation of a sequence of activities into a linear Petri net model. Using a process model and a trace model, we define the synchronous product as follows:

Definition 2 (Synchronous Product). *Let* $PN = (P^m, T^m, F^m, \lambda^m)$ *and* $PM = (PN, m_i^m, m_f^m)$ *a process model. Furthermore, let* $\sigma \in A^*$ *be a trace with* $TN = ((P^l, T^l, F^l, \lambda^l), m_i^l, m_f^l)$ *its trace model.*

The synchronous product $SN = ((P, T, F, \lambda), m_i, m_f)$ *is a Process Model, with:*

- $P = P^m \cup P^l$ *is the combined set of places,*
- $T = \{(t^m, t^l) \in (T^m \cup \{\gg\}) \times (T^l \cup \{\gg\}) \mid t^m \neq \gg \vee t^l \neq \gg \vee \lambda^m(t^m) = \lambda^l(t^l)\}$ *is the set of original transitions merged with synchronous ones,*
- $F = \{((t^m, t^l), p) \in T \times P \mid (t^m, p) \in F^m \vee (t^l, p) \in F^l\} \cup \{(p, (t^m, t^l)) \in P \times T \mid (p, t^m) \in F^m \vee (p, t^l) \in F^l\}$, *is the set of edges,*
- *for all* $(t^m, t^l) \in T$ *holds that* $\lambda((t^m, t^l)) = \begin{cases} \lambda^m(t^m) & \text{if } t^m \neq \gg \\ \lambda^l(t^l) & \text{if } t^l \neq \gg \end{cases}$, *i.e.* λ *respects the original labelling,*
- $m_i = m_i^m \uplus m_i^l$ *is the combined initial marking, and*
- $m_f = m_f^m \uplus m_f^l$ *is the combined final marking.*

A synchronous product net is a combination of a process model and a trace model in such a way that for each pair of transitions of which the labels agree, a synchronous transition is added. Any complete firing sequence of the synchronous product represents an alignment in which each transition firing represents either a model move (transitions of the type $(t^m, \gg)$), a log move (transitions of the type $(\gg, t^l)$ or a synchronous move (transitions of the form (t^m, t^l)).

An optimal alignment is an alignment for which a cost function, associating costs to the firing of each transition, is minimized.

Definition 3 (Optimal alignment). *Let* $SN = ((P, T, F, \lambda), m_i, m_f)$ *be a synchronous product model and let* $c : T \to \mathbb{R}_{\geq 0}$ *be a cost function associating costs to each transition in the synchronous product. An optimal alignment* $\gamma \in T^*$ *is a firing sequence of* SN*, such that* $m_i[\gamma\rangle m_f$ *and there is no* $\theta \in T^*$ *with* $m_i[\theta\rangle m_f$ *and* $\sum_{t \in \theta} c(t) < \sum_{t \in \gamma} c(t)$. *We use* $c(\theta)$ *as shorthand for* $\sum_{t \in \theta} c(t)$.

Note that there may be many optimal alignments in the general case, i.e. alignments with the same minimal costs. Typically, the cost function c is chosen in such a way that 0 costs are only associated to synchronous transitions, i.e. transitions $(t^m, t^L) \in T$ for which $t_M \neq \gg$ and $t_L \neq \gg$. Routing transitions of the form $(\tau, \gg)$ are typically given a small cost $\epsilon > 0$, while transitions of the form $(t, \gg)$ or $(\gg, t)$ with $t \neq \tau$ receive cost 1. For the work in this paper the cost function should be chosen such that there are no infinite sets of markings reachable with the same costs, for which the above cost function provides a sufficient condition. For models that do not contain infinite sets of markings reachable through routing transitions only, $\epsilon = 0$ is a valid assignment, which is what we use in this paper.

4.1 Underestimation Using the Marking Equation

To compute alignments, we build on a technique introduced in [4] with the parameter settings optimized according to [16]. This technique is based on $A^\star$, which is a shortest path search algorithm that can be guided towards the destination using a function that *underestimates* the remaining costs of the optimal alignment.

Definition 4 (Underestimation Function). *Let $PN = ((P,T,F,\lambda), m_i, m_f)$ be a Process model and let $c : T \to \mathbb{R}_{\geq 0}$ be a cost function. We define $h : \mathcal{B}(P) \to \mathbb{R}_{\geq 0}$ to be an underestimation function if and only if for all $m \in \mathcal{B}(P)$ and $\sigma \in T^*$ with $m[\sigma\rangle m_f$ holds that $h(m) \leq c(\sigma)$.*

Several underestimation functions exist in literature. The most trivial one is the function that always returns 0. This function leads to $A^\star$ behaving like Dijkstra's shortest path algorithm, essentially doing a breadth-first search through the reachability graph of the synchronous product.

Using the marking equation of the synchronous product, several other underestimation functions can be defined.

Definition 5 (ILP-based Underestimation Function). *Let $PN = ((P,T,F,\lambda), m_i, m_f)$ be a process model with incidence matrix $\mathbf{C}$ and let $c : T \to \mathbb{R}_{\geq 0}$ be a cost function. We use $\vec{c}$ to denote a column vector of the cost values for each transition, i.e. $\vec{c}(t) = c(t)$.*

We define $h^{ILP} : \mathcal{B}(P) \to \mathbb{R}_{\geq 0}$ as an ILP based underestimation, such that for each marking $m \in \mathcal{B}(P)$ holds that $h^{ILP}(m) = \vec{c}^{\intercal} \cdot \vec{x}$ where $\vec{x}$ is the solution to:

$$\begin{array}{ll} \textit{minimize} & \vec{c}^{\intercal} \cdot \vec{x} \\ \textit{subject to} & \vec{m} + \mathbf{C} \cdot \vec{x} = \vec{m_f} \\ & \forall_{t \in T} \vec{x}(t) \in \mathbb{N} \end{array}$$

If no solution $\vec{x}$ to the linear equation system exists, then $h^{ILP}(m) = +\infty$

The ILP-based estimation function uses the marking equation to underestimate the remaining cost to the final marking from any marking reached. Recall that for any firing sequence, the marking equation has a solution, but not the other way around. As a consequence, if the marking equation has no solution, there is no firing sequence to the final marking. Furthermore, by minimizing, we guarantee to provide a lower bound on the remaining distance.

Theorem 1 (h^{ILP} Provides a lower bound on the costs [4]). *Let $PN = ((P,T,F,\lambda), m_i, m_f)$ be a process model with incidence matrix $\mathbf{C}$ and let $c : T \to \mathbb{R}_{\geq 0}$ be a cost function. Let $m \in \mathcal{B}(P)$ be a marking and $\gamma \in T^*$ an optimal firing sequence such that $m[\gamma\rangle m_f$, i.e. no firing sequence $\theta \in T^*$ with $m[\theta\rangle m_f$ exists such that $c(\theta) < c(\gamma)$. We prove that $h^{ILP}(m) \leq c(\gamma)$.*

Proof. Since γ is a firing sequence, we know that $\vec{m} + \mathbf{C}\vec{\gamma} = \vec{m_f}$ (marking equation). Furthermore, for all $t \in T$ $\vec{\gamma}(t) = \#_{0 \leq i < |\gamma|}\theta(i) = t \in \mathbb{N}$ and $\vec{c}^{\intercal} \cdot \vec{\gamma} = c(\gamma)$. Assume $\vec{x}$ minimizes the ILP shown in Definition 5 for marking m. It is trivial that either $\vec{c}^{\intercal} \cdot \vec{x} \leq \vec{c}^{\intercal} \cdot \vec{\gamma}$, otherwise $\vec{x}$ is not minimizing. Hence $h^{ILP}(m) \leq c(\gamma)$.

The estimation function h^{ILP} can be used to underestimate the remaining cost of reaching the final marking in the synchronous product from any marking reached and is therefore suitable to be used as a heuristic in $A^{\star}$. Unfortunately, solving the integer linear program for every marking in the model is a complex task. It is well-known that solving integer linear programs is exponential in the rank of the matrix, i.e. in the number of transitions and places in the synchronous product.

Therefore, we relax the constraints a bit and to use a non-integer linear program to underestimate the remaining cost, i.e. we replace the last line of Definition 5 by $\forall_{t \in T}\vec{x}(t) \in \mathbb{R}_{\geq 0}$ to obtain $h^{LP} : \mathcal{B}(P) \rightarrow \mathbb{R}_{\geq 0}$.

Since any integer solution is also a real valued solution, for any marking m holds that $h^{LP}(m) \leq h^{ILP}(m)$. Therefore, h^{LP} also provided an underestimate, but the computational complexity is polynomial in the rank of the matrix.

In practice, as h^{LP} provides a worse underestimate than h^{ILP}, more markings need to be expanded when computing alignments, i.e. with a worse underestimation function, $A^{\star}$ investigates a larger part of the search space. However, due to the fact that most LP solvers use Simplex as a solving technique, it is unlikely that the actual solutions returned are non-integer for real-life examples.

An important property of both functions is the fact that the vectors $\vec{x}$ that provide the minimum can be used to derive a solution in the next marking of the search space.

Theorem 2 (Minimizing solutions can be reused). *Let $PN = ((P, T, F, \lambda), m_i, m_f)$ be a process model with incidence matrix $\mathbf{C}$ and let $c : T \rightarrow \mathbb{R}_{\geq 0}$ be a cost function. Let $m, m' \in \mathcal{B}(P)$ be two markings and $\vec{x}$ a vector minimizing the linear program of Definition 5. Furthermore, let $t \in T$ be such that $m[t\rangle m'$ with $\vec{x} \geq 1$. We show that $h^{ILP}(m') = h^{ILP}(m) - c(t)$ with solution vector $\vec{x'} = \vec{x} - \vec{1}_t$ (where $\vec{1}_t$ is a vector with 0 on all rows except the row corresponding to t, which has value 1).*

Proof. We prove by contradiction. Let $\vec{y}$ be a vector satisfying the constraints of Definition 5 such that $h^{ILP}(m') = \vec{c}^{\intercal} \cdot \vec{y} < h^{ILP}(m) - c(t)$, i.e. $\vec{y}$ provides a better solution than $\vec{x'}$.

Since $m[t\rangle m'$, we know that $\vec{m} + \mathbf{C} \cdot \vec{1}_t = \vec{m'}$ (marking equation). Therefore $\vec{m} + \mathbf{C} \cdot \vec{1}_t + C\vec{y} = \vec{m_f}$, i.e. $\vec{m} + \mathbf{C} \cdot (\vec{1}_t + \vec{y}) = \vec{m_f}$ and since this provides an integer solution for the linear program, we know $h^{ILP}(m) \leq \vec{c}^{\intercal} \cdot (\vec{1}_t + \vec{y})$. However, $\vec{c}^{\intercal} \cdot (\vec{1}_t + \vec{y}) = \vec{c}^{\intercal} \cdot \vec{1}_t + \vec{c}^{\intercal} \cdot \vec{y} = c(t) + \vec{c}^{\intercal} \cdot \vec{y}$. In other words: $h^{ILP}(m) - c(t) \leq \vec{c}^{\intercal} \cdot \vec{y}$ which contradicts the fact that y provides a better solution than $\vec{x'}$.

It is trivial to see that $\vec{x'}$ is indeed an integer solution to the linear equation system since $\vec{m} = \vec{m'} - \mathbf{C} \cdot \vec{1}_t$, and therefore $\vec{m'} + \mathbf{C} \cdot \vec{x} - \mathbf{C} \cdot \vec{1}_t = \vec{m_f}$, i.e.

$\vec{m'} + \mathbf{C} \cdot (\vec{x} - \vec{1}_t) = \vec{m_f}$ and thus $\vec{m'} + \mathbf{C} \cdot \vec{x'} = \vec{m_f}$. Since $\vec{x}(t) \geq 1, \vec{x}(t) - 1 = \vec{x'}(t) \geq 0$ and hence $\vec{x'}$ provides a solution to the linear equation system for $h^{ILP}(m')$.

Note that the proof is analogous for the h^{LP} case. Also there, a new solution can be derived from a previous one when firing transition t, as long as $\vec{x}(t) \geq 1$. By guiding $A^\star$ to favour markings for which the solution can be reused, the search time can be improved considerably.

Unfortunately, even when re-using solutions, there are scenarios that are common in practical cases where $A^\star$ performs poorly (almost at worst-case, i.e. exploring the full reachability graph of the synchronous product net), for example in the case of Fig. 2.

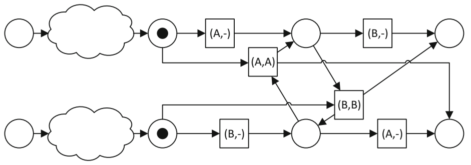

Fig. 2. Synchronous product for the example of Fig. 1.

This is due to the fact that in all reachable markings, the solution to the linear equation system suggests that the two synchronous transitions (A, A) and (B, B) can be executed with cost 0, while in fact, only two sequences are possible with minimal cost of 2, namely $\langle(A, -), (B, B), (-, A)\rangle$ or $\langle(B, -), (A, A), (-, B)\rangle$.

The root cause for the poor performance in this case is the use of the marking equation in the heuristic function. In particular, the fact that the vector $\vec{x}$ minimizing the (integer) linear program does not necessarily correspond to a realizable firing sequence. In the next subsection, we show how the marking equation can be extended to improve the solution vectors of the linear program.

4.2 Underestimation Using the Extended Marking Equation

We first consider the marking equation again, but this time we also include the consumption matrix.

Definition 6 (Consumption Matrix). *Let $PN = (P, T, F, \lambda)$ be a Petri net. The consumption matrix $\mathbf{C}^-$ is a matrix with $|P|$ rows and $|T|$ columns, such that for all $t \in T$ and $p \in P$ holds that* $\mathbf{C}^-(p, t) = \begin{cases} -1 & \textit{if } (p,t) \in F \\ 0 & \textit{othwerwise} \end{cases}$

The consumption matrix can be used to guarantee that before the firing of a specific transition, the necessary tokens are present in a Petri net by extending the marking equation slightly.

Definition 7 (Extended Marking Equation). *Let $PN = (P, T, F, \lambda)$ be a Petri net, $\mathbf{C}$ its incidence matrix and $\mathbf{C}^-$ its consumption matrix. Let $\theta_1, \theta_2 \in T^*$ be two firing sequences and m_1, m_2, m_3 three markings, such that $m_1[\theta_1\rangle m_2$ and $m_2[\theta_2\rangle m_3$. Furthermore, assume $\theta_2 = \langle t, \ldots\rangle$, i.e. θ_2 starts with transition t. The extended marking equation states that:* $\begin{array}{l} \vec{m_1} + \mathbf{C} \cdot (\vec{\theta_1} + \vec{\theta_2}) = \vec{m_3}, \textit{and} \\ \vec{m_1} + \mathbf{C} \cdot \vec{\theta_1} + \mathbf{C}^- \cdot \vec{1}_t \geq \vec{0} \end{array}$.

The second part of the extended marking equation is essentially a translation of the enabling condition for Petri nets. If a transition is enabled, we know that each input place contains sufficiently many tokens. The marking reached after firing any sequence θ_1 in the net is determined by the first part of the original marking equation. The consumption matrix is then used to express that after *consumption* of all tokens required by transition t (but before production of its output tokens), the net does not have any place with a negative number of tokens.

As with the original marking equation, the reverse does not hold, i.e. not all solutions for $\vec{\theta_1}$ and $\vec{\theta_2}$ that satisfy these conditions correspond to realizable firing sequences. However, as these conditions are more restrictive, we can use them in the estimation function for $A^\star$ if we can identify the point where we should split between θ_1 and θ_2. For this, we reason over the original trace.

Theorem 3 (An optimal alignment is at least as long as the trace). *Let $SN = ((P,T,F,\lambda), m_i, m_f)$ be a synchronous product with $PM = ((P^m, T^m, F^m, \lambda^m),\ m_i^m,\ m_f^m)$ the process model and $TN = ((P^l, T^l, F^l, \lambda^l), m_i^l, m_f^l)$ the trace model for trace $\sigma \in A^*$ it is constructed from.*

Let $\gamma \in T^*$ be an optimal alignment for some cost function $C : T \rightarrow \mathbb{R}_{\geq 0}$. We show that $|\gamma| \geq |\sigma| = |T^l|$, i.e. the length of the optimal alignment is longer than the length of the trace σ.

Proof. This follows trivially from the structure of the synchronous product and the fact that the token from the place initially marked in the trace model needs to be "transported" to the final place of the trace model in any trace reaching this final marking.

We use this property of the optimal alignment to provide a better estimation function. Simply put, we cut the optimal alignment into a number of k predefined pieces (where k is less or equal to the length of the sequence) and we guarantee that after consumption of each transition at the start of each subtrace, the marking is non-negative.

Definition 8 (Underestimation with k subtraces in the initial marking). *Let $SN = ((P,T,F,\lambda), m_i, m_f)$ be a synchronous product model for trace σ. SN has incidence matrix $\mathbf{C}$ and consumption matrix $\mathbf{C}^-$. Let $c : T \rightarrow \mathbb{R}_{\geq 0}$ be a cost function and let $\sigma = \sigma_1 \circ \ldots \circ \sigma_k$ be a division of the trace into k non-empty subtraces.*

We define $h_{m_i} = \vec{c}^\intercal \cdot \sum_{0 \leq a \leq k} \vec{x_a} + \vec{c}^\intercal \cdot \sum_{0 \leq a < k} \vec{y_a}$ where $\vec{x_a}$ and $\vec{y_a}$ provide the solution to:

$$
\begin{array}{lllr}
\textit{minimize} & & \vec{c}^\intercal \cdot \sum_{0 \leq i \leq k} \vec{x_i} + \vec{c}^\intercal \cdot \sum_{1 \leq i \leq k} \vec{y_i} & \\
\textit{subject to} & & \vec{m_i} + \mathbf{C} \cdot \vec{x_0} + \mathbf{C} \cdot \sum_{1 \leq a \leq k} (\vec{x_a} + \vec{y_a}) = \vec{m_f} & (1) \\
\forall_{1 \leq a \leq k} & & \vec{m_i} + \mathbf{C} \cdot \vec{x_0} + \mathbf{C} \cdot \sum_{1 \leq b < a} (\vec{x_b} + \vec{y_b}) + \mathbf{C}^- \cdot \vec{y_a} \geq \vec{0} & (2) \\
\forall_{0 \leq a \leq k}\ \forall_{t \in T} & & \vec{x_a}(t) \in \mathbb{N} & (3) \\
\forall_{1 \leq a \leq k}\ \forall_{t \in T} & & \vec{y_a}(t) \in \{0,1\} & (4) \\
\forall_{1 \leq a \leq k}\ \forall_{(t^m,t^l) \in T} & \textit{with } t^l \neq t_{(1+\sum_{1 \leq b < a} |\sigma_b|)} & \vec{y_a}((t^m,t^l)) = 0 & (5) \\
\forall_{1 \leq a \leq k} & & \vec{1}^\intercal \cdot \vec{y_a} = 1 & (6)
\end{array}
$$

Note that the assumption is that there is at least one alignment, hence this equation system is guaranteed to have a solution.

The equation system in Definition 8 appears complicated, but is a fairly straightforward translation of the extended marking equation. Consider an optimal alignment $\gamma \in T^*$ for a trace σ of length $|\sigma|$. As we know, we can split this optimal alignment into $k \leq |\sigma|$ subtraces, each of which starts with one transition corresponding to an event from the trace, i.e. $\gamma = \gamma_0 \circ \ldots \circ \gamma_k$, where for $0 < i \leq k$ holds that $\gamma_i = \langle (t, t^l), \ldots \rangle$ with $t^l \in T^l$, i.e. γ_i starts with one transition that moves the token in the original trace model. γ_0 is a (possibly empty) prefix of transitions of the form $(t^m, \gg)$.

In Definition 8, variables $\vec{y_a}$ refer to the first transition in each γ_i, i.e. these vectors encode the firing of a single transition at the start of each subtrace. Variables $\vec{x_a}$ (bound by rule 3) correspond to any other transitions firing (in any order). These vectors may be empty. Rule 4 guarantees that every element of $\vec{y_a}$ is 0 or 1 and rule 5 and 6 ensure that only one element of $\vec{y_a}$ equals 1 and that that element corresponds to a transition which is a transition of the synchronous product corresponding to the start of σ_a.

Rule 1 is a translation of the original marking equation. It simply states that when combining the firing of all transitions in all $\vec{x_a}$ and $\vec{y_a}$, the final marking is reached from the initial marking. Finally, rule 2 uses the extended marking equation to guarantee that after firing a prefix of transitions γ_0 through γ_{a-1}, sufficient tokens are available to fire the first transition in γ_a (expressed by $\vec{y_a}$).

Definition 8 provides again an underestimate for the total cost of an optimal alignment in the initial marking. It can be generalized to any arbitrary reachable marking by assuming that we know how many events of the original trace remain to be explained, i.e. by adjusting both m_i and σ. This information is generally available by considering the marking of the places in the trace model. Furthermore, like before, we can relax constraints 3 and 4 from integers to real valued numbers to reduce the complexity of minimizing the linear inequation system.

Definition 9 (Underestimation with k subtraces). *Let $SN = ((P, T, F, \lambda), m_i, m_f)$ be a synchronous product model for trace σ. Let $m \in \mathcal{B}(P)$ be a marking in which l events of trace σ are explained and let $\sigma = \sigma_0 \circ \sigma_1 \circ \ldots \circ \sigma_k$ be a division of the trace into $k+1$ non-empty subtraces, such that the first l events are in σ_0, i.e. $k \leq |\sigma| - l$.*

We define $h^{ILP,k} : \mathcal{B}(P) \to \mathbb{R}_{\geq 0}$ as an underestimation function, such that $h^{ILP,k}(m) = h_m$ following Definition 8.

Like before, we prove that $h^{ILP,k}$ indeed provides an underestimation function, provided that we know, for each marking, how many events have been explained by the path leading up to this marking. Recall that this information is trivially derived from the location of the token in the places corresponding to the trace model.

Theorem 4 ($h^{ILP,k}$ is an underestimation function). *Let $SN = ((P, T, F, \lambda), m_i,\ m_f)$ be a synchronous product model for trace σ and let $\sigma = \sigma_0 \circ \sigma_1 \circ \ldots \circ \sigma_k$ be a division of the trace into $k+1$ non-empty subtraces, such that the first l events are in σ_0.*

Let $m \in \mathcal{B}(P)$ be a marking in which l events of trace σ are explained. We prove that for each $\theta \in T^*$ with $m[\theta\rangle m_f$ holds that $h^{ILP,k}(m) \leq c(\theta)$.

Proof. Let $\gamma \in T^*$ be a firing sequence with $m[\gamma\rangle m_f$, such that for each $\theta \in T^*$ with $m[\theta\rangle m_f$ holds that $c(\gamma) \leq c(\theta)$, i.e. γ is optimal.

We know that $\gamma = \gamma_0 \circ \ldots \circ \gamma_k$ such that for $1 \leq a \leq k$ holds that $\gamma_a = \langle t_a, \ldots \rangle$ with $t_a = (t^m, t^l)$ and $t^l = t_{(1+\sum_{0 \leq b < a} |\sigma_b|)} \in T^l$, i.e. γ can be split into $k+1$ subtraces such that all but the first subtrace start with a transition related to an event in the original trace. We show that γ provides a solution for the inequation system.

Let $\vec{x_0} = \vec{\gamma_0}$ and for all $1 \leq a \leq k$, let $\vec{y_a} = \vec{1_{t_a}}$ and let $\vec{x_a} = \vec{\gamma_a} - \vec{y_a}$, i.e. we translate the subtraces of γ into Parikh vectors, where we separate the first transitions into vectors $\vec{y}$ and the remainder into vectors $\vec{x}$.

It is trivial to see that by definition, conditions 3, 4, and 6 are met by this translation. Condition 5 is also met, since the only element with value 1 is the element that satisfies the given condition. Furthermore, Condition 1 is met, since $\vec{m} + \mathbf{C} \cdot \vec{x_0} + \mathbf{C} \cdot \sum_{1 \leq a \leq k} (\vec{x_a} + \vec{y_a}) = \vec{m} + \mathbf{C} \cdot (\vec{x_0} + \sum_{1 \leq a \leq k} (\vec{x_a} + \vec{y_a})) = \vec{m} + \mathbf{C} \cdot \vec{\gamma} = \vec{m_f}$. This is again the marking equation which holds for any sequence.

Condition 2 is more complicated. Let $1 \leq a \leq k$ and let $m' \in \mathcal{B}(P)$ be the marking, such that $m[\gamma_0, \ldots, \gamma_{a-1}\rangle m'$, i.e. m' is the marking reached after executing the first a subtraces. We know that $\vec{m} + \mathbf{C} \cdot \vec{x_0} + \mathbf{C} \cdot (\sum_{1 \leq b < a} (\vec{x_b} + \vec{y_b}) = \vec{m'}$ (again from the marking equation). Furthermore, we know that firing transition t_a is possible in marking m' as this is the first transition in γ_a. Hence $\vec{m'} + \mathbf{C}^- \cdot \vec{y_a} \geq \vec{0}$ which follows from the extended marking equation. Combining the two yields $\vec{m} + \mathbf{C} \cdot \vec{x_0} + \mathbf{C} \cdot (\sum_{1 \leq b < a} (\vec{x_b} + \vec{y_b}) \geq -\mathbf{C}^- \cdot \vec{y_a}$. Therefore $\vec{m} + \mathbf{C} \cdot \vec{x_0} + \mathbf{C} \cdot (\sum_{1 \leq b < a} (\vec{x_b} + \vec{y_b}) + \mathbf{C}^- \cdot \vec{y_a} \geq \vec{0}$.

Since γ provides a solution to the inequation system, we know that any minimal solution has less or equal costs. Hence $h^{ILP,k}$ is an underestimation function.

It is fairly easy to see that any solution in terms of $\vec{x}$ and $\vec{y}$ to the inequation system of Definition 8 can be translated into a solution to the equation systems of Definition 5 in terms of $\vec{z}$ by defining $\vec{z} = \vec{x_0} + \sum_{1 \leq a \leq k} (\vec{x_a} + \vec{y_a})$. Hence for any marking m holds that $h^{LP}(m) \leq h^{ILP}(m) \leq h^{ILP,k}(m)$. Furthermore, the k-variant can also easily be extended to the real domain in which case $h^{LP}(m) \leq h^{LP,k}(m) \leq h^{ILP,k}(m)$.

This implies that $h^{ILP,k}$ provides a better underestimation function than the previous ones, since it guarantees that *if* the first vector $\vec{x_0}$ corresponds to a realizable firing sequence, *then* it reaches a marking that enables the transition indicated by $\vec{y_1}$. So for each splitpoint, the extended marking equations comes with an additional "guarantee" on a transition being enabled.

However, this comes at a cost of computational complexity, since the inequation system has more variables (and constraints). Where Definition 5 has $|T|$ variables, Definition 8 has $(2 * k + 1) * |T|$ variables (of which many are bound to 0 by constraint 5).

The observation that there are many more variables involved in the k variants of the underestimation function leads to the question which value for k should be chosen. Furthermore, the re-use of solutions of the inequation system to derive new solutions is not as trivial as it is for the traditional underestimation functions. Therefore, we propose an incremental search technique to find optimal alignments which combines Theorem 2 with Definition 9.

4.3 Incrementally Extending the Heuristic Function

Using the k-based underestimation function, we propose a special version of the $A^\star$ search algorithm which incrementally increases the value of k when needed and maximizes the reuse of previously computes solution vectors. The basic principle of the search is simple:

We start in the initial marking m_i by computing an underestimate for $k = 1$. After solving the inequation system, we remember the solution vector $\vec{z_{m_i}} = \vec{x_0} + \sum_{1 \leq a \leq k}(\vec{x_a} + \vec{y_a})$ as well as the estimate h.

Then, we follow the classical $A^\star$ algorithm to find the optimal alignment. For each marking m' reached by firing transition t in marking m, the new underestimate value is computed either exactly if for marking m holds that $\vec{z_m}(t) \geq 1$, in which case $h(m') = h(m) - c(t)$ and $\vec{z_{m'}} = \vec{z_m} - \vec{1_t}$, (cf. Theorem 2). If $\vec{z_m}(t) < 1$, then $h(m') = \max(0, h(m) - c(t))$, but $\vec{z_{m'}} = \perp$, i.e. the underestimate is decreased, but the solution vector is unknown.

During the $A^\star$ search, we prioritize markings for which the solution vector is known and since we monotonously decrease the underestimate with the actual costs, we know that if we ever have to expand a marking for which the solution vector is unknown, there are no markings with known solution vectors in the open set of the search algorithm.

If we do visit a marking with an unknown solution vector, we have reached a point where the original estimate computed in the initial marking corresponds to a firing sequence that is guaranteed not to be realizable. In that case, we try to improve the underestimate function by increasing k and choosing a new way to split the trace. Here, we consider the maximum number of events already explained by any marking reached in the search space so-far and we split there, i.e. if a marking explains a events for $k = 1$, we restart the procedure from scratch with $k = 2$ and $\sigma = \sigma_1 \circ \sigma_2$ with $|\sigma_1| = a$.

Suppose that in the next iteration, we encounter an unknown solution vector again, while we have found a marking explaining b events with $b > a$, we then restart with $k = 3$ and $\sigma = \sigma_1 \circ \sigma_2 \circ \sigma_3$ with $|\sigma_1| = a$ and $|\sigma_2| = b - a$. If $b < a$, then we restart with $k = 3$ and $\sigma = \sigma_1 \circ \sigma_2 \circ \sigma_3$ with $|\sigma_1| = b$ and $|\sigma_2| = a - b$.

If we encounter again an unknown solution vector with $b = a$ events explained, we follow the normal steps in the search, i.e. we compute an exact solution for the state and we requeue the state if the new estimate is higher than the previous estimate.

This procedure incrementally improves the underestimation function used in the $A^\star$. Worst case, this procedure leads to the entire alignment problem being encoded as a linear program and especially for larger process models, this implies that the computation time for the initial marking might increase substantially. It is therefore advised not to use the ILP variant, but the LP variant. In practical examples, k does not grow so big and in the experiment section, we show that this procedure can considerably reduce the computation time of finding alignments.

5 Experiments

The procedure outlined in this paper was implemented in ProM as part of the normal alignment plugins. It requires the user to install the "alignment" package in which case the incremental algorithm shows up as a advanced variant in the conformance checking plugin. The input objects as well as the returned objects are identical to the classical conformance checking, i.e. optimal alignments are returned.

In this section, we compare the performance of the algorithm with two existing alignment techniques. First, the classical $A^\star$ using optimal parameter settings [16] and second the planner-based approach [5]. Note that, for the former, we use a completely new codebase which is optimized for memory use and also part of the aforementioned alignment package in ProM. For the latter, we use the ProM package "PlanningBasedAlignment"[1].

We report the CPU times of all three techniques. For the planner, we show both the time reported by the software, as well as the wallclock time. The latter is considerably larger due to the overhead of reading and writing files and instantiating the external planner processed from Python. As the planner can only be run in a single thread, we conducted all experiments in single-threaded mode on a 2.8 GHz Intel Xeon W3530 CPU. The tools were given maximum 10 GB of RAM and for the planner, file IO was done on a 2 GB RAM disk in physical RAM, eliminating slow disk access. The $A^\star$ variants were bound to 60 s per trace. The planner does not support a time limit. It was given as much time as needed and was only terminated after running out of memory (10 GB). The $A^\star$ approaches never needed more than 10 MB of memory to store the internal data structures for a single trace (including the memory used by the LP Solver LpSolve). The Java VM however needed about 1 GB to store the event logs and Petri net objects in memory. The total wallclock time to conduct all experiments (not including the logs for which the planner ran out of memory) was 9 h for the planner, 5 h for $A^\star$ and 35 min for incremental $A^\star$. The remaining logs took 32 h in $A^\star$ (however, 45% of the traces timed out after 60 s) and 2 h using our incremental version.

[1] Due to the new codebase, the time performance of $A^\star$ cannot directly be compared to the results presented in [5]. Furthermore, the CPU times reported in [5] were obtained using a proprietary implementation which is not available for download.

Table 1. Time and memory to align benchmark event logs.

Log	Cases	Classical $A^\star$ [16]			Incremental $A^\star$		Planner [5]			cost
		time (s)	(timeout)	solved LPs	time (s)	solved LPs	pre (s)	src (s)	clock (s)	
[18] road fines	10,247	0.2		458	**0.2**	176	4.3	1.6	2,362.4	7,284
[17] bpi12	12,136	7.9		29,089	**1.3**	759	10.8	43.4	3,111.0	57,727
[19] sepsis	1,051	120.0		440,825	**10.7**	5,553	1.8	59.6	323.4	2,448
[9] Fitting logs	4,004	4.6		2,935	**4.5**	2,935	15.2	13.7	1,123.9	53
[9] Noisy logs	16,016	52.6		100,529	**21.7**	14,155	55.3	237.8	4,671.8	12,111
[20] Fitting log (prBm6)	1,201	5.0		1,126	**4.3**	1,126	72.6	24.5	679.4	14
[20] Noisy logs	6,506	132,824.2	2,182	24,016,461	**8,190.7**	47,270	1,465.0	Out of Mem		98,708
[21] Fitting logs	38,019	53.2		25,620	**51.5**	25,620	326.2	185.1	12,022.5	253
[21] Noisy logs	28,014	15,503.4		11,756,515	**488.5**	76,857	288.1	575.5	9,618.0	91,603
[20] prAm6	1,201	**55.0**		29,496	99.2	2,752	64.3	798.3	1,475.1	4,969
[20] prCm6	501	1,780.4		804,518	1,302.7	7,683	31.7	166.1	**445.1**	12,552
[20] prEm6	1,201	**29.1**		7,200	60.8	4,800	148.5	Out of Mem		4,880
[20] prFm6	1,201	30,847.8	514	2,702,736	**1,354.8**	7,878	396.1	Out of Mem		21,425
[20] prGm6	1,201	71,941.9	1,200	18,643,107	**1,526.6**	13,794	253.4	Out of Mem		26,411
[21] pr1151_l4	2,001	2,467.9		1,457,592	**68.9**	7,486	26.9	61.4	715.8	9,044
[21] pr191_l4	2,001	3,978.8		2,702,250	**126.5**	11,707	30.5	94.1	763.7	16,044

Table 1 shows the results for three real life logs, three collections divided in fitting and non-fitting logs[2] and selected individual model/log combinations from these collections. We show the number of cases and per algorithm, the time needed to align all traces, the number of traces that reached the timeout and the number of linear programs solved. The time is the total time needed to align all traces in the log (including the timed-out ones). If the planner ran out of memory, we show the pre-processing time (excluding IO) for translating the traces to PDDL files. For each row, the fastest technique (wallclock time) is highlighted.

The table shows that for the top three real-life cases, the incremental $A^\star$ outperforms the others, mainly due to the significant reduction in number of LP's that are solved. For the entire collections, the totals are shown over the full collection, divided into fitting and noisy event logs. For fitting event logs, the performance of both $A^\star$ variants is equal. For noisy logs, the incremental version is orders of magnitude faster. The planner's preprocessing time is already higher than the computation time for both $A^\star$ variants.

The models from [20] present the worst-case scenario for incremental $A^\star$. These logs are created using swapping of events in various parts of the model. The incremental $A^\star$ needs to reach the swapped event, after which a splitpoint will be introduced. For prAm6 and prEm6, we see that $A^\star$ is faster despite the reduction in number of LPs solved, i.e. the CPU can investigate enough states per second to eliminate the need for incremental $A^\star$. This is due to the location of the swaps in these logs. For prEm6 the planner runs out of memory, but the pre-processing time (excluding IO) is already higher than the $A^\star$ time.

For prCm6, a model which does not have a large amount of parallelism, the planner is fastest in the reported time, but not in wallclock time.

[2] Note that the cost are never 0, due to the empty trace that is always included in the computation. Furthermore, for the largest of these logs, no optimal alignments could be computed before.

For prFm6 and prGm6, regular $A^\star$ runs out of time for many (if not all) of the traces. This is due to the fact that these models contain vast amounts of parallelism and swapped events towards the end of the traces. The full reachability graph is therefore expanded by $A^\star$. Our incremental version however correctly identifies the swaps and returns optimal alignments, whereas the planner runs out of memory, probably also due to the parallelism.

Figure 3 shows the relation between the cost of the alignment, the time to compute the alignment and the memory use for the two $A^\star$ variants for model prCm6. The planner is not included here as it does not report the time per trace, but only the total time for the entire event log. We see that the times per trace are comparable for the two algorithms (time is plotted on the lefthand axis, with logarithmic scale). As the costs of the alignments increase, so does the computation time. The time complexity for this model shows a polynomial trend in the cost of the alignment, as does the memory use. For regular $A^\star$ the memory use is rather low, as markings are not explicitly stored (they are computed when needed by following the firing sequence back to the root). For incremental $A^\star$,

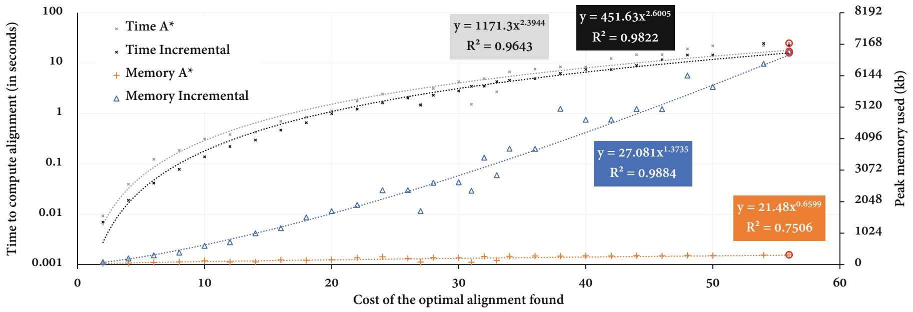

Fig. 3. Time and memory use of both approaches vs. the cost of the optimal alignment found for prCm6 of [20]. The trendlines show polynomial time and memory complexity.

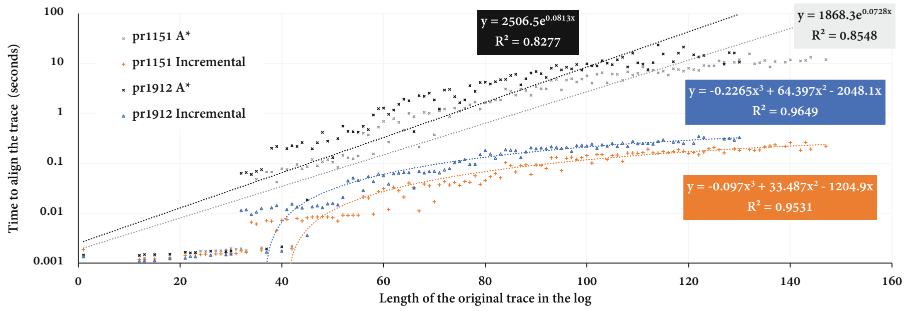

Fig. 4. Time to compute optimal alignments for the two most complex models in [21]. Trendlines show exponential time complexity for $A^\star$ and cubic complexity for the incremental approach.

the memory is dominated by the size of the linear program. For example, the maximum number of linear programs solved was 31 for model prCm6 with trace "instance_124" (highlighted). About 6 MB of memory is needed there, mainly to store the non-zero coefficients of a linear program of 11,688 rows and 10,262 columns in the solver. The total time to compute the optimal alignment was 22 s for incremental $A^\star$, of which 16 s was spent in the LP solver (for all 31 LPs together) and the remaining 6 s on investigating 7,575 markings. For the same trace, the classical $A^\star$ searched 9,792 markings and solved 8,706 LPs in 25 s.

In Fig. 4, we show the computation time per trace for the two most complex event logs in Table 1. For each trace, we plot the time needed to align it in the model using classic $A^\star$ (on a logarithmic axis) as well as our incremental version (again the planner is omitted as it does not report times per trace). These models contain considerably more parallelism which results in an exponential time trendline for $A^\star$. More interestingly though, the time complexity for the incremental version shows a cubic trend in both cases. This difference is explained by the parallelism in the model, which leads to considerable differences in the percentage of the time that is spent in the LP solver, as well as the number of solved LPs. The incremental version spends 72% of the computation time in the solver for pr1151 (70% for pr1912), vs. 99% (99%) for the classical one. Per trace however, the incremental version solves only 7.0 (4.9) LPs per trace rather than 953 (1612). The time per solve call is roughly 4.5 times higher for the incremental version as the linear programs are larger.

The main difference between the various datasets is the length of the traces compared to the size of the model. As our technique explicitly exploits the trace to identify split-points, it benefits from long traces in relatively small models. In [20], the models do not contain loops, so the traces are relatively short (up to 271 events) compared to the model size (up to 429 transitions). In the other datasets, the trace lengths are up to three times the number of transitions in the models.

6 Conclusion

Computing optimal alignments is a time consuming task which is essential in the context of conformance checking. Traditional algorithms for computing optimal alignments use the marking equation borrowed from Petri net theory as an underestimating heuristic function in the context of an $A^\star$ search. Unfortunately, this heuristic is proven to perform poorly in certain cases which cause the $A^\star$ to expand nearly the full reachability graph of the Petri net, leading to excessive computation times.

In this paper, we reconsider the heuristic function by exploiting knowledge of the traces being aligned. Essentially, we use the original trace to guarantee progress in the depth of the $A^\star$ search. We do this by splitting the marking equation into a number of sub-problems which together provide a more accurate under estimation of the remaining cost. Rather than starting the search with the fully split marking equation, we use $A^\star$ itself to decide when to split the marking equation and we show this leads to a considerable reduction in number of computed linear programs.

The work is implemented in ProM and we use publicly available benchmark datasets to compare our work to existing techniques, showing significant improvements in computation time. For future work, we develop techniques select appropriate splitpoints in advance.

References

1. van der Aalst, W.M.P.: Decomposing Petri nets for process mining: a generic approach. Distrib. Parallel Databases **31**(4), 471–507 (2013)
2. van der Aalst, W.M.P.: Process Mining - Data Science in Action, 2nd edn. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
3. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdiscip. Rev.: Data Min. Knowl. Discov. **2**(2), 182–192 (2012)
4. Adriansyah, A.: Aligning observed and modeled behavior. Ph.D. thesis, Department of Mathematics and Computer Science. Eindhoven University of Technology, July 2014
5. de Leoni, M., Marrella, A.: Aligning real process executions and prescriptive process models through automated planning. Expert Syst. Appl. **82**, 162–183 (2017)
6. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968)
7. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Scalable process discovery and conformance checking. Softw. Syst. Model. **17**(2), 599–631 (2018)
8. López, M.T.G., Borrego, D., Carmona, J., Gasca, R.M.: Computing alignments with constraint programming: the acyclic case. In: Proceedings of ATAED, Torun, Poland, CEUR Workshop Proceedings, vol. 1592, pp. 96–110. CEUR-WS.org (2016)
9. Măruşter, L., Weijters, A.J., van der Aalst, W.M., van den Bosch, A.: A rule-based approach for process discovery: dealing with noise and imbalance in process logs. Data Min. Knowl. Discov. **13**(1), 67–87 (2006)
10. Munoz-Gama, J., Carmona, J., van der Aalst, W.M.P.: Single-entry single-exit decomposed conformance checking. Inf. Syst. **46**, 102–122 (2014)
11. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
12. Reißner, D., Conforti, R., Dumas, M., La Rosa, M., Armas-Cervantes, A.: Scalable conformance checking of business processes. In: Panetto, H., et al. (eds.) OTM 2017. LNCS, vol. 10573, pp. 607–627. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69462-7_38
13. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1), 64–95 (2008)
14. Taymouri, F., Carmona, J.: A recursive paradigm for aligning observed behavior of large structured process models. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 197–214. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45348-4_12
15. Taymouri, F., Carmona, J.: Model and event log reductions to boost the computation of alignments. In: Ceravolo, P., Guetl, C., Rinderle-Ma, S. (eds.) SIMPDA 2016. LNBIP, vol. 307, pp. 1–21. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74161-1_1

16. van Zelst, S.J., Bolt, A., van Dongen, B.F.: Tuning alingment computation: an experimental evaluation. In: Proceedings of ATAED, 25–30 June 2017, Zaragoza, Spain, pp. 1–15 (2017)
17. van Dongen, B.F.: BPI challenge dataset, 2012, in 4TU Center for Research Data. https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
18. de Leoni, M., Mannhardt, F.: Road Fines dataset, 2015, in 4TU Center for Research Data. https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5
19. Mannhardt, F.: Sepsis dataset, 2016, in 4TU Center for Research Data. https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460
20. Munoz-Gama, J.: Synthetic dataset, 2013, in 4TU Center for Research Data. https://doi.org/10.4121/uuid:44c32783-15d0-4dbd-af8a-78b97be3de49
21. Munoz-Gama, J.: Synthetic dataset, 2014, in 4TU Center for Research Data. https://doi.org/10.4121/uuid:b8c59ccb-6e14-4fab-976d-dd76707bcb8a