The Complexity of Alignments on (Un-)Labeled and (Un-)Bounded Petri Nets

Patrizia Schalk[1,*], Robert Lorenz[1], Jakub Kovář[2] and Robin Bergenthum[2]

[1]University of Augsburg, Universitätsstraße 6a, Augsburg, 86135, Germany
[2]FernUniversität in Hagen, Universitätsstraße 11, Hagen, 58097, Germany

Abstract

Petri nets are one of the standard modeling languages for visualizing and analyzing processes. However, automatically discovered models for such processes only show a snapshot of all possible system behavior. Thus, a common task is to decide whether an observed sequence of activities is part of a Petri net's language. If the answer is *no*, the next question is how far the sequence deviates from the model's language. Optimal alignments answer this question by finding the minimum number of insert and delete operations to transform the sequence into a word of the language. Alignments are used in several metrics that evaluate a model's conformance to recorded data. However, algorithms that compute optimal alignments have an exponential runtime. Only recently has research found that finding optimal alignments is computationally hard for sound free-choice workflow nets. This paper continues this line of research. It investigates the effects of the labeling and boundedness of a Petri net on the computational complexity of finding optimal alignments. Unlabeled and unbounded Petri nets have attracted increasing research interest in disciplines such as object-centric process mining. The results of this paper show that the problem of finding optimal alignments is PSPACE-complete for unlabeled nets, PSPACE-complete for labeled and bounded nets, and ACKERMANN-complete for labeled and unbounded nets.

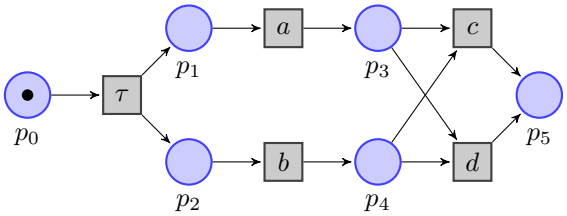
Keywords

Conformance Checking, Alignments, Computational Complexity, Petri Nets

1. Introduction

Petri nets are a popular modeling language for the analysis of processes [1, 2]. Their intuitive formal semantics make them easily accessible, while their mathematical foundations unlock a large toolbox of formal analysis techniques. To ensure high model quality, it is common practice to check whether it describes some previously recorded data of the process. As such, the model's language should contain as much of the recorded behavior as possible (i.e., have high *fitness*), but not much beyond that (i.e., have high *precision*). There are several metrics that compute the fitness and precision of a model with respect to the recorded data [3]. The most frequently used metrics are based on *alignments* [4, 5]: For some sequence σ, alignments show which insert and delete operations are necessary to transform σ into a word of the model's language. For example, consider the Petri net N in Fig. 1 and the sequence acb.

Model N:



Alignments:

γ_1:

Word:	$\gg$	a	$\gg$	c	b
Model:	τ	a	b	c	$\gg$

γ_2:

Word:	$\gg$	a	$\gg$	c	b	$\gg$
Model:	τ	a	b	$\gg$	$\gg$	d

Figure 1: A Petri net N with transitions τ, a, b, c, d, and two of its alignments, γ_1 and γ_2, for the word acb.

ATAED'26: Algorithms & Theories for the Analysis of Event Data 2026, June 23, 2026, Hamburg, Germany
*Corresponding author.
patrizia.schalk@uni-a.de (P. Schalk); robert.lorenz@uni-a.de (R. Lorenz); jakub.kovar@fernuni-hagen.de (J. Kovář); robin.bergenthum@fernuni-hagen.de (R. Bergenthum)
0009-0001-7757-6628 (P. Schalk); 0009-0005-0906-745X (R. Lorenz); 0000-0002-7775-3698 (J. Kovář); 0000-0003-0464-8843 (R. Bergenthum)

Fig. 1 shows two alignments, γ_1 and γ_2, for acb with respect to the model N. Both alignments show how to transform acb into a word in the language of N: γ_1 proposes to insert τ before the first a, to insert b directly after the first a, and to delete the last b of the sequence acb. γ_2, on the other hand, proposes not only to delete the last b of acb, but also the last c, and to insert a d at the end of the sequence. Thus, γ_1 transforms acb into τabc, while γ_2 transforms acb into the word τabd. We use the notation for alignments in [5], denoting silent transitions by τ and a "skip" symbol by $\gg$. That is, for every insertion, we add $\gg$ to the log sequence, and for every deletion, we add $\gg$ to the model sequence.

Similar to the Levenshtein distance [6], we determine how close a sequence σ is to the language $L(N)$ of a Petri net N by asking for the minimum number of operations that transform σ into a word of $L(N)$. Note that inserting or deleting some symbols might be more problematic than inserting or deleting others. In the model of Fig. 1, the leftmost transition τ has the special role of initiating the parallel execution of a and b. This might just be a design choice in the model that is not reflected in the recorded data. If this is the case, the insertion of τ in an alignment should not infer any penalty. To incorporate this use case, alignments allow one to assign individual costs to each possible insert and delete operation. With the so-called *standard costs* for alignments, each insertion of control flow transitions such as τ has costs 0, while inserting or deleting any other symbol has costs 1.

The cost of an alignment is the sum of the costs for each performed operation. With standard costs, the total cost of γ_1 is 2, while the total cost of γ_2 is 4. The most popular fitness and precision measures calculate *optimal alignments*, i.e., alignments with the minimum possible cost. To calculate optimal alignments, the literature provides an algorithm that transforms the alignment problem into a single-source shortest path problem on the reachability graph of a special Petri net [3]. However, since the original net is a subnet of this special Petri net, the size of the reachability graph can be exponential in the size of the original net. Consequently, the runtime of this algorithm is exponential. For various reasons, researchers believe that this runtime cannot be asymptotically improved [3].

Related Work. If this conjecture is true, calculating optimal alignments for large Petri nets may not be feasible in practice. However, since alignments form the basis for many popular fitness and precision measures, there is active research on strategies to speed up this calculation. Van Dongen [7] uses knowledge about the sequence to be aligned and defines an extended marking equation. The resulting algorithm shows significant runtime improvements over the traditional method when considering difficult instances. Boltenhagen et al. [8] encode the task of finding optimal alignments as an optimized SAT instance. Since the satisfiability of formulas is a well-researched NP-complete problem, this enables them to use some of the fastest known approximation algorithms to calculate optimal alignments. However, as the name suggests, approximation algorithms do not return exact results. Fani Sani et al. [9] therefore propose a framework based on proxy sets to determine bounds for the approximation error. Their framework for mitigating the approximation error allows approaches such as those of Boltenhagen et al. [8] to become efficient for calculating fitness scores. However, the exact complexity class of the problem of calculating optimal alignments remained an open question for a long time.

Only recently, Schwanen et al. [5] found an answer to this question for the most commonly used Petri net classes in process modeling. They showed that the problem of finding optimal alignments is NP-complete for *live*, *bounded*, *free-choice* systems (short LBFC systems). A net is live iff any transition t can eventually fire, regardless of which transitions were fired beforehand. It is bounded iff there is a bound b such that in every reachable marking of the model, the number of tokens per place does not exceed b. Finally, a net is free-choice iff two distinct transitions either have no common places from which they take tokens or share the same set of such places. For example, the Petri net in Fig. 1 is not live, since every transition can fire at most once, but it is bounded by $b = 1$ and free-choice. Schwanen et al. [5] also found that the problem of calculating alignments for 1-bounded, free-choice *workflow nets* is NP-complete as well. Workflow nets are a subclass of Petri nets with a distinct source and sink place, where every place and transition lie on a directed walk from the source to the sink. For non-free-choice 1-bounded systems, Schwanen et al. [5] found that calculating optimal alignments is PSPACE-complete. Their results allow one to estimate when computing alignments is feasible in practice.

Contribution. This paper continues the investigation started by Schwanen et al. [5]. While the Petri net classes these authors analyzed are the most common to model the control structure of a process, recent trends in research show a shift towards model types that model dependencies and interactions between the control flow of different objects as well. Fahland [10] uses so-called proclets to describe the life cycle of objects in a process. These proclets synchronize over the same transition labels to show how two objects interact with each other. Understanding each proclet as an independent system, it is likely to contain unique transition labels. Thus, we investigate the effect of a Petri net's labeling on the complexity class of the alignment problem.

Van Hee et al. [11] expand Petri nets with token identifiers, so each token represents an object. In this case, 1-bounded nets cover trivial cases where there is exactly one object present. Thus, we investigate the more general case where a net is b-bounded for some natural number b. Furthermore, van Hee et al. [12] and van der Werf et al. [13] describe nets that contain arbitrarily many objects of one type. As tokens represent objects, they describe situations where the underlying system is unbounded.

There is active and ongoing research on how to define alignments for the previously mentioned model types [14, 15, 16]. Thus, investigating the complexity class of the alignment problem for the cases described above holds value for this line of research. This paper therefore investigates the computational complexity of calculating optimal alignments for the following Petri net classes:

- Unlabeled and bounded place/transition nets;
- Unlabeled and unbounded place/transition nets;
- Labeled and bounded place/transition nets; and
- Labeled and unbounded place/transition nets.

With this contribution, we expand what is known about the computational complexity of the alignment problem for Petri nets to the results shown in Table 1.

Table 1
An overview of results on the computational complexity of computing optimal alignments for Petri nets.

Petri net class	Result	Reference
(acyclic) LBFC systems	NP-complete	[5]
(acyclic) sound free-choice workflow nets	NP-complete	[5]
1-bounded systems	Pspace-complete	[5]
1-bounded and sound workflow nets	Pspace-complete	[5]
unlabeled, bounded place/transition nets	Pspace-complete	Corollary 1
labeled, bounded place/transition nets	Pspace-complete	Corollary 2
unlabeled, unbounded place/transition nets	Pspace-complete	Corollary 3
labeled, unbounded place/transition nets	Ackermann-complete	Theorem 8

Note that all Petri nets considered by Schwanen et al. [5] are place/transition nets. That is, the nets contain at most one arc between each place and transition, and they contain no special arcs (such as reset or inhibitor arcs). Furthermore, all nets they consider are labeled.

Schwanen et al. also found results on the computational complexity of the alignment problem for process trees [17]. Process trees are a special modeling language that can be translated to Petri nets. The authors show that the alignment problem on process trees can be described as solutions of mixed integer linear programs. If the model does not contain parallel executions, they show that the alignment problem can even be solved efficiently. However, since these results do not directly concern the alignment problem with respect to Petri net classes, we did not add them to the overview in Table 1.

The remainder of this paper is organized as follows: Section 2 introduces definitions for the investigated Petri net classes, alignments, and the alignment problem. It also briefly describes the relevant complexity classes. Section 3 and Section 4 investigate the computational complexity of the alignment decision problem for bounded and unbounded place/transition nets, respectively. Both sections distinguish between cases where the net is labeled and cases where the net is unlabeled. Finally, Section 5 concludes the paper and offers further directions for future research.

2. Basic Definitions

We denote the set of natural numbers by $\mathbb{N} := \{1, 2, 3, \dots\}$ and define $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. Furthermore, by $\mathbb{R}$ we denote the set of real numbers, and by $\mathbb{R}_0^+ := \{r \in \mathbb{R} \mid r \geq 0\}$ the set of non-negative real numbers. Next, we give the definition of the modeling language investigated in this paper, that is, place/transition nets.

Definition 1. ***(Place/Transition Net).*** *A* place/transition net *(short:* P/T net*) is a tuple $N = (P, T, F)$, where P is a finite set of places, T is a finite set of transitions, $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation. For any place $p \in P$, we call ${}^\bullet p = \{t \in T \mid (t, p) \in F\}$ the* preset *of p and $p^\bullet = \{t \in T \mid (p, t) \in F\}$ the* postset *of p. Similarly, we define ${}^\bullet t = \{p \in P \mid (p, t) \in F\}$ and $t^\bullet = \{p \in P \mid (t, p) \in F\}$ as the pre- and postset of a transition $t \in T$.*

The model N of Fig. 1 shows a place/transition net with six places, drawn as circles, and five transitions, drawn as rectangles. The flow relation F defines the directed edges in this place/transition net. There can only be edges from places to transitions or from transitions to places. The preset (postset) of a place p is the set of transitions that have an edge to (from) p. For example, ${}^\bullet p_5 = \{c, d\}$.

In some literature, the notions of place/transition nets and Petri nets are used interchangeably. However, sometimes the term Petri nets is used specifically for models with weighted arcs or with arcs featuring special semantics. To avoid confusion, in this paper, we use the term *place/transition net*, or short *P/T net*, for a class of Petri nets without arc weights or special arcs. To define the semantics and thus the behavior modeled by a P/T net, we first define the notion of *markings*.

Definition 2. ***(Multisets, Markings).*** *Let D be a set. A multiset is a mapping $m : D \to \mathbb{N}_0$ that assigns to each element $d \in D$ how often it occurs in the multiset. We denote the number of occurrences of an element $d \in D$ in the multiset m by $m(d)$. We denote finite multisets by $M = [d_1^{m(d_1)}, \dots, d_n^{m(d_n)}]$, where $d_1, \dots d_n \in D$. We omit all entries $d^{m(d)}$ with $m(d) = 0$ and omit the exponent if it is equal to 1. For two multisets m and m' over a domain set D, we define $(m + m')$ as a multiset over D where $(m + m')(d) = m(d) + m'(d)$ for each $d \in D$. Similarly, we define $(m - m')$ as a multiset over D where $(m - m')(d) = \max\{m(d) - m'(d), 0\}$ for each element $d \in D$.*

A marking M of a P/T net $N = (P, T, F)$ is a multiset over the set of its places, that is, $M : P \to \mathbb{N}_0$.

Markings define the state of a P/T net. We visualize a marking M by drawing $M(p)$ black bullets inside each place $p \in P$. We call these black bullets *tokens*. For example, in Fig. 1, the place p_0 contains exactly one token, while all other places contain no tokens. Thus, the marking of this example net is $M = [p_0^1, p_1^0, p_2^0, p_3^0, p_4^0, p_5^0]$, which we write as $M = [p_0]$.

The tokens in a P/T net determine which of the transitions can fire. A transition can *fire* if all places in its preset contain at least one token. If it fires, it takes one token from each of the places in its preset and puts one token in each place of its postset. In this way, firing a transition can enable other transitions that could not fire before. The language of a P/T net is the set of all sequences of transition firings. Let us now define these ideas formally.

Definition 3. ***(Firing Rule, Language of a P/T Net).*** *Let $N = (P, T, F)$ be a P/T net and $M : P \to \mathbb{N}_0$ a marking of N. We call a transition $t \in T$* enabled *if $M(p) \geq 1$ for all places $p \in {}^\bullet t$ and write $(N, M)[t\rangle$ in this case. If some $t \in T$ is enabled in M, it can* fire*. Firing t leads to the marking $M' = M - {}^\bullet t + t^\bullet$. We denote this fact by $(N, M)[t\rangle M'$ or, if N is clear from the context, by $M \xrightarrow{t} M'$.*

A sequence of transitions $\sigma = t_1 \dots t_n$ with $t_1, \dots, t_n \in T$ is called a firing sequence *in M if there are markings $M_1, \dots, M_n$ with $(N, M)[t_1\rangle M_1$ and $\forall i \in \{2, \dots, n\} : (N, M_{i-1})[t_i\rangle M_i$. In this case, we write $(N, M)[\sigma\rangle$ and call the sequence σ* enabled*. Furthermore, we write $(N, M)[\sigma\rangle M_n$ and say firing σ leads to M_n. Again, if N is clear from the context, we also write $M \xrightarrow{\sigma} M_n$.*

The language *of a P/T net N with respect to some initial marking M_0 is the set of all its enabled and finite firing sequences, that is, $L(N, M_0) = \{t_1 \dots t_n \mid t_1, \dots, t_n \in T \wedge (N, M_0)[t_1 \dots t_n\rangle\}$. This set is a subset of all possible finite sequences of transitions, $T^* = \{t_1 \dots t_n \mid t_1, \dots, t_n \in T\}$.*

In the example of Fig. 1, the initial marking $[p_0]$ enables only the transition τ. Firing this transition leads to the marking $[p_1, p_2]$, where both the transitions a and b are enabled. Since they are enabled simultaneously and have disjoint presets, we can fire them both. We can choose whether to fire a first, b first, or even both transitions at the same time. However, since we defined the language of a P/T net as a set of sequences, we need to choose one of the transitions to appear first. Firing a and b in any order leads to the marking $[p_3, p_4]$, where the transitions c and d are enabled. These transitions have the same preset and thus compete for the tokens in the respective places, so here we have to choose whether to fire c or d. Regardless of our choice, firing the transition will lead to the marking $[p_5]$, where no transition is enabled. The language of N with respect to the marking $[p_0]$ is $L(N, [p_0]) = \{\varepsilon, \tau, \tau a, \tau b, \tau ab, \tau ba, \tau abc, \tau abd, \tau bac, \tau bad\}$, where ε denotes the empty sequence.

In the context of process modeling, we are typically interested in firing sequences that lead to a special marking that we call the *final marking*. This is because if we record data of a process, we do not store all prefixes of sequences of events but instead only those that end in some predefined state. We call P/T nets with final states *accepting systems*.

Definition 4. ***(Accepting System).*** *Let $N = (P, T, F)$ be a P/T net and $M_0 : P \to \mathbb{N}_0$ a marking. We call the tuple (N, M_0) a* system *and M_0 the* initial marking*. In addition, let $M_f : P \to \mathbb{N}_0$ be a marking. Then, we call the triple (N, M_0, M_f) an* accepting system *and M_f its* final marking*. The language of an accepting system is $L(N, M_0, M_f) = \{\sigma \in T^* \mid (N, M_0)[\sigma\rangle M_f\}$.*

For example, by setting the final marking of N in Fig. 1 to $[p_5]$, we construct an accepting system with the language $L(N, [p_0], [p_5]) = \{\tau abc, \tau abd, \tau bac, \tau bad\}$. Setting the final marking to $[p_2, p_3]$ would lead to an accepting system with the language $L(N, [p_0], [p_2, p_3]) = \{\tau a\}$.

Next, we increase the expressive power of P/T nets by giving *labels* to transitions. Furthermore, we introduce transitions without any externally visible semantics. These transitions allow us to move around tokens inside the P/T net but are not registered as a label in the word built by a firing sequence.

Definition 5. ***(Labeled P/T nets).*** *Let Σ be an alphabet with $\tau \notin \Sigma$. A* labeled P/T net *over Σ is a 5-tuple $N = (P, T, F, \Sigma, \lambda)$, where (P, T, F) is a P/T net, and $\lambda : T \to (\Sigma \cup \{\tau\})$ is a labeling function that assigns a label to each transition $t \in T$. Transitions $t \in T$ with $\lambda(t) = \tau$ are called* silent transitions*.*

A labeled P/T net N together with an initial marking M_0 is called a labeled system*. Its language is $L(N, M_0) = \{\rho_\lambda(\sigma) \mid \sigma \in L((P, T, F), M_0)\}$, where*

- *$\rho_\lambda(\varepsilon) = \varepsilon$ and*
- *$\rho_\lambda(t \cdot \sigma) = \begin{cases} \rho_\lambda(\sigma) & \text{if } \lambda(t) = \tau \\ \lambda(t) \cdot \rho_\lambda(\sigma) & \text{otherwise.} \end{cases}$*

Similarly, a labeled P/T net N together with an initial marking M_0 and a final marking M_f is called a labeled accepting system*. Its language is $L(N, M_0, M_f) = \{\rho_\lambda(\sigma) \mid \sigma \in L((P, T, F), M_0, M_f)\}$.*

For example, if we interpret the model N in Fig. 1 as a labeled system, its language becomes $L(N, [p_0]) = \{\varepsilon, a, b, ab, ba, abc, abd, bac, bad\}$, since we ignore all the occurrences of the silent transition τ when constructing the language. Interpreted as an accepting system with the final marking $[p_5]$, the language of this model becomes $L(N, [p_0], [p_5]) = \{abc, abd, bac, bad\}$.

Note that two transitions t_1, t_2 with $t_1 \neq t_2$ can receive the same transition label from λ. This makes it easier to create P/T nets with rather involved languages but also makes it more difficult to find whether a word is part of a model's language. Later, we investigate how this affects the complexity of the alignment problem. For now, we define one last important property of P/T nets.

Definition 6. ***(Reachable Markings, Boundedness).*** *Let (N, M_0) be a system, where $N = (P, T, F)$ is a P/T net. The* set of reachable markings *of (N, M_0) is the set of markings that can be reached by firing any firing sequence, that is, $[N, M_0\rangle = \{M : P \to \mathbb{N}_0 \mid \exists \sigma : (N, M_0)[\sigma\rangle M\}$.*

We call a system (N, M_0) bounded *if there is a place-bound $b \in \mathbb{N}_0$, such that every reachable marking contains at most b tokens, that is, $\forall M \in [N, M_0\rangle : \forall p \in P : M(p) \leq b$.*

As discussed earlier, the set of reachable markings for the system $(N, [p_0])$ shown in Fig. 1 is $[N, [p_0]\rangle = \{[p_0], [p_1, p_2], [p_1, p_4], [p_2, p_3], [p_3, p_4], [p_5]\}$. In all of these markings, every place contains 0 or 1 tokens. Thus, $(N, [p_0])$ is a 1-bounded system. It is also 2-bounded, 3-bounded, and so on, since no place can ever contain more than 2, 3, or more tokens.

Alignments. With the syntax and semantics of important Petri net classes defined, we now draw our attention to *alignments*. As described in Section 1, alignments visualize and evaluate how closely some word σ resembles an element of an accepting system's language. They do this by using four types of *moves*. *Synchronous moves*, where the model and the word agree; *log moves*, where we ignore the next symbol of the word; *model moves*, where we allow the model to fire a transition that is not the same as the next symbol of the word; and *silent moves*, where the model fires a silent transition.

Definition 7. ***(Alignments).*** *Let Σ be an alphabet with $\Sigma \cap \{\tau, \gg\} = \emptyset$. Furthermore, let (N, M_0, M_f) be an accepting system, where $N = (P, T, F, \Sigma, \lambda)$. Let $\sigma_L = \sigma_1 \dots \sigma_j$ be a sequence of elements $\sigma_1, \dots, \sigma_j \in \Sigma$ (i.e., a word of Σ^*). An* alignment *between σ_L and a firing sequence of (N, M_0, M_f) is an n-tuple $((\alpha_1, \beta_1), \dots, (\alpha_n, \beta_n))$ with $n \geq \max\{j, k\}$, where $\alpha_1, \dots, \alpha_n, \beta_1, \dots, \beta_n \in (\Sigma \cup \{\tau, \gg\})$, and where for any $i \in \{1, \dots, n\}$, one of the following holds:*

- *$\alpha_i = \beta_i$ (then (α_i, β_i) is called a* synchronous move*),*
- *$\alpha_i \in \Sigma$ and $\beta_i = \gg$ (then (α_i, β_i) is called an* (asynchronous) log move*),*
- *$\alpha_i = \gg$ and $\beta_i \in \Sigma$ (then (α_i, β_i) is called an* (asynchronous) model move*), or*
- *$\alpha_i = \gg$ and $\beta_i = \tau$ (then (α_i, β_i) is called a* silent move*).*

Furthermore, deleting all occurrences of the symbol $\gg$ in the sequence $\alpha_1 \dots \alpha_n$ results in σ_L, and deleting all occurrences of the symbol $\gg$ in the sequence $\beta_1 \dots \beta_n$ results in some $\sigma_N = \lambda(t_1) \dots \lambda(t_k)$, where $t_1 \dots t_k \in L((P, T, F), M_0, M_f)$ is a firing sequence of the accepting system (including silent transitions).

On its right side, Fig. 1 shows two alignments, γ_1 and γ_2. If we write these alignments as tuples, we get $\gamma_1 = ((\gg, \tau), (a, a), (\gg, b), (c, c), (b, \gg))$ and $\gamma_2 = ((\gg, \tau), (a, a), (\gg, b), (c, \gg), (b, \gg), (\gg, d))$. The alignment γ_1 contains two synchronous moves, the alignment γ_2 only one. Since γ_2 is longer than γ_1, γ_2 is a worse alignment than γ_1: It contains more log- and model moves than necessary.

We quantify this observation by defining a *cost function* for moves in an alignment and for whole alignments. A cost function assigns a positive real-valued score to each move. We get a cost for the overall alignment by summing the costs of all tuples that the alignment consists of.

Definition 8. ***(Cost Function).*** *Let Σ be an alphabet with $\Sigma \cap \{\tau, \gg\} = \emptyset$. A* cost function *for alignment moves is a function $cost : (\Sigma \cup \{\gg\}) \times (\Sigma \cup \{\tau, \gg\}) \to \mathbb{R}_0^+$ that assigns non-negative costs to each of the possible tuples in an alignment. For an alignment $\gamma = ((\alpha_1, \beta_1), \dots, (\alpha_n, \beta_n))$ of length $n \in \mathbb{N}_0$, we define $cost(\gamma) = \sum_{i=1}^{n} cost(\alpha_i, \beta_i)$.*

To find how close a word σ is to the accepting system, we find the smallest possible cost of an alignment between σ and a word of the accepting system's language. An alignment with the smallest possible cost is called an *optimal alignment*. Optimal alignments are important for calculating fitness and precision scores to relate process models with the data recorded from real processes. To facilitate their use, these measures use a *standard cost function*. In the standard cost function, all asynchronous moves have a cost of 1, while synchronous and silent moves have a cost of 0.

Definition 9. ***(Standard Cost Function).*** *Let Σ be an alphabet with $\Sigma \cap \{\tau, \gg\} = \emptyset$. The* standard cost function *for alignments is $cost_{std} : (\Sigma \cup \{\gg\}) \times (\Sigma \cup \{\tau, \gg\}) \to \{0, 1, \infty\}$ where*

$$cost_{std}(\alpha_i, \beta_i) = \begin{cases} 0 & \text{if } \alpha_i = \beta_i \text{ or } (\alpha_i = \gg \wedge\ \beta_i = \tau), \\ 1 & \text{if } (\alpha_i \in \Sigma \wedge \beta_i = \gg) \text{ or } (\alpha_i = \gg \wedge\ \beta_i \in \Sigma), \\ \infty & \text{otherwise.} \end{cases}$$

Since most algorithms that rely on alignments use the standard cost function, this paper investigates the alignment problem with respect to $cost_{std}$. In particular, we want to find the complexity class for calculating optimal alignments w.r.t. $cost_{std}$. However, complexity classes are only defined for decision problems, so we first reformulate the problem of finding an optimal alignment into a decision problem.

Decision Problems and Complexity Classes. To define the alignment problem, we first reiterate which Petri net classes we are interested in. In this paper, we investigate (un-)labeled, (un-)bounded P/T nets. However, for a first general introduction to the alignment decision problem, let us suppose that we have some predefined Petri net class $\mathcal{C}$. Then, the alignment problem is the following:

(ALIGN$_\mathcal{C}$) Alignment Decision Problem for Petri Nets of Class $\mathcal{C}$

Input:	An alphabet Σ, a finite sequence $\sigma \in \Sigma^*$, an accepting system (N, M_0, M_f), where N is a Petri net of class $\mathcal{C}$, and a natural number $k \in \mathbb{N}_0$.
To decide:	Is there an alignment γ between the sequence σ and a firing sequence of (N, M_0, M_f) that leads to the marking M_f and has $cost_{std}(\gamma) \leq k$?

Instead of asking for an optimal alignment or its cost, we fix some cost $k \in \mathbb{N}_0$ and ask whether there is an alignment with standard cost at most k. This results in a question that can be answered by *"yes"* or *"no"*, so ALIGN$_\mathcal{C}$ is in fact a decision problem.

Decision problems can be solved under different time and space constraints. Some decision problems require more time or space to be solved than others. We say that the former problems are more difficult to solve than the latter. However, to be able to compare the time and space requirements of two decision problems, we need to ensure that they are solved on the same type of machine. Due to their expressive power, the literature agrees to use *Turing machines* for this task, which essentially consist of a state machine and an infinite tape of storage space.

To ensure comparability of the space requirements with other decision problems, we need to encode the Petri nets in a series of 0s and 1s. Special dividing symbols, in our case #, and a blank symbol ␣ for empty cells are also allowed. We represent a P/T net (P, T, F) with two binary matrices. The first matrix, $A_{p,t}$ has $|P|$ rows (one for each place) and $|T|$ columns (one for each transition). An entry (i, j) in this matrix is 1 if for the respective pair of place and transition $(p_i, t_j) \in F$, and 0 otherwise. The second matrix, $A_{t,p}$ has $|T|$ rows and $|P|$ columns. Similarly, an entry (i, j) in this matrix is 1 if $(t_i, p_j) \in F$, 0 otherwise. For the net N in Fig. 1, we get

$$A_{p,t} = \begin{array}{c} \\ p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{array} \begin{array}{c} \begin{array}{ccccc} \tau & a & b & c & d \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \qquad A_{t,p} = \begin{array}{c} \\ \tau \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccccc} p_0 & p_1 & p_2 & p_3 & p_4 & p_5 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Using the same idea, we can encode a marking M of a P/T net as a vector. For each place p we write the binary representation of $M(p)$ on the tape to store M. Fig. 2 shows how we store the example net N of Fig. 1 and the marking $M = [p_1^3, p_4^2]$ for this net on the tape of a Turing machine. As this figure shows, storing the matrix $A_{p,t}$ takes $(|P|+1)\cdot|T|+3$ cells of the Turing machine's tape. Storing the matrix $A_{t,p}$, on the other hand, takes $(|T|+1)\cdot|P|+3$ cells. Thus, to store a P/T net (P, T, F), we need a total of $\mathcal{O}(|P|\cdot|T|)$ cells. For a marking M, we use $\mathcal{O}(|P|\cdot\log_2(b_M))$ cells, where $b_M = \max\{M(p) \mid p \in P\}$ is the highest amount of tokens a place holds in M.

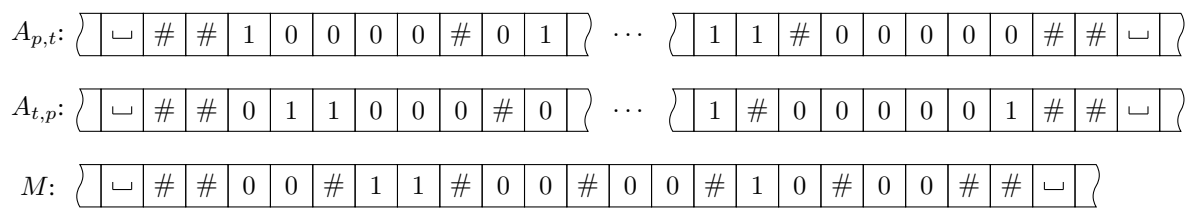


Figure 2: How to store binary encoded P/T nets and their markings on the tape of a Turing machine.

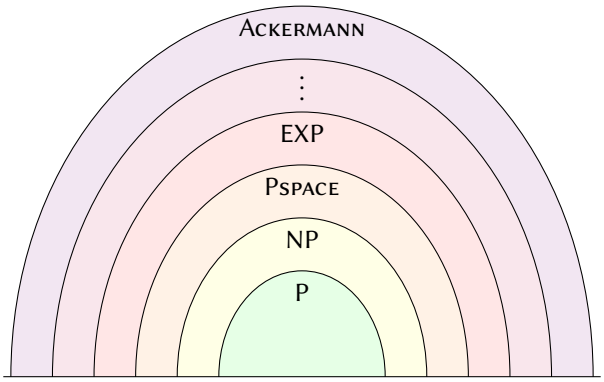


Figure 3: A part of the hierarchy of the complexity classes (see [19, p. 499]) relevant to this paper.

To store the alphabet Σ is even simpler: Since all elements in Σ are unique, we count the number of elements in Σ. To write $|\Sigma|$ on the tape, we need $\mathcal{O}(\log_2(|\Sigma|))$ cells. Then, we can write a sequence σ onto the tape using $\mathcal{O}(|\sigma| \cdot \log_2(|\Sigma|))$ cells, where $|\sigma|$ denotes the length of the sequence σ. Thus, to store the input of the decision problem $\text{ALIGN}_{\mathcal{C}}$, we need $\mathcal{O}(|P| \cdot (|T| + \log_2(b_M)) + |\sigma| \cdot \log_2(|\Sigma|) + \log_2(k))$ cells on the tape of a Turing machine. We call this the *input size* of the problem $\text{ALIGN}_{\mathcal{C}}$.

We are interested in finding how much time or space relative to the input size we need to solve the decision problem. We say that a problem is in the class P if it can be solved in polynomial time by a deterministic Turing Machine. If the Turing machine needs nondeterminism to solve the problem in polynomial time, we say that the problem is in the class NP.

If we wish to ignore how much time we need to solve a decision problem, we can instead restrict the space requirements. Problems that can be solved with polynomial space by a deterministic or non-deterministic Turing machine form the class Pspace. According to Esparza [18], most interesting decision problems concerning Petri nets are at least in the class Pspace. Beyond the classes P, NP, and Pspace, we can allow the Turing machine to use an exponential amount of time or storage space to solve a decision problem, leading to classes EXP, EXPspace, and so on. If the time or space requirements grow as fast as the Ackermann function with respect to the input size, we enter the class Ackermann.

Fig. 3 shows the hierarchy of the complexity classes described above. There are many more complexity classes that we do not investigate. To analyze which complexity class the four versions of the alignment problem belong to, we first assume that the input P/T net is bounded. Recall that boundedness means that in all reachable markings of the P/T net, places contain at most b tokens for some $b \in \mathbb{N}_0$.

3. The Alignment Problem on Bounded P/T Nets

In this section, we analyze accepting systems that are bounded by some $b \in \mathbb{N}_0$. We first assume that the input accepting system is unlabeled. We address the case where it is labeled later in this section. Thus, we investigate the following decision problem for unlabeled, bounded accepting systems.

($\text{ALIGN}_b^{\neg\lambda}$) Alignment Decision Problem for Unlabeled, Bounded P/T Nets

Input:	An alphabet Σ, a sequence $\sigma \in \Sigma^*$, an accepting system (N, M_0, M_f), where N is an unlabeled, b-bounded P/T net, and a natural number $k \in \mathbb{N}_0$.
To decide:	Is there an alignment γ between the sequence σ and a firing sequence of (N, M_0, M_f) that leads to the marking M_f and has $cost_{std}(\gamma) \leq k$?

To solve this decision problem, we first need to answer whether M_f is reachable in the accepting system (N, M_0, M_f). If the system is bounded, its set of reachable markings $[N, M_0\rangle$ is finite. Thus, a possible approach lists all reachable markings and checks whether M_f is one of them. However, this procedure is not runtime efficient. Consider for example the net N_{bin} shown in Fig. 4. This net simulates

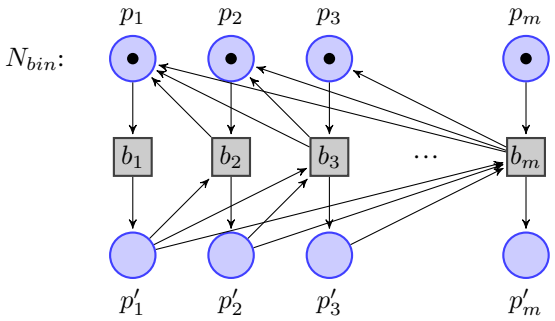


Figure 4: A 1-bounded P/T net with $3m$ nodes and 2^m reachable markings.

a binary counter. To enable some transition b_i with $i \in \{1, \ldots, m\}$, we first have to fire all transitions b_j with $1 \leq j < i$. Only then do all places $p'_1, \ldots, p'_{i-1} \in {}^\bullet b_i$ contain a token. If we fire b_i, it consumes all of these tokens and produces a token in p'_i. The places $p'_1, \ldots, p'_m$ then represent the current state of the counter in binary form. Thus, the number of reachable markings in N_{bin} is the same as the number of values that a binary number can represent when using m bits, that is, $|[N, M_0\rangle| = 2^m$. N_{bin} contains $3m$ nodes, so this number of reachable markings is exponential in the size of the net. Since we will use this property later on, Lemma 1 shows that this is a general rule: The number of reachable markings in any bounded P/T net is at most exponential in the size of the net and the bound b.

Lemma 1. *Let $b \in \mathbb{N}_0$ and (N, M_0) be a b-bounded system with $N = (P, T, F)$. Then (N, M_0) has at most $(b+1)^{|P|}$ reachable markings, i.e., $|[N, M_0\rangle| \leq (b+1)^{|P|}$.*

Proof. We prove by induction. Suppose $|P| = 1$. Then N contains only one place p. Since (N, M_0) is b-bounded, $M(p) \in \{0, 1, \ldots, b\}$ for any reachable marking $M \in [N, M_0\rangle$. Thus, in this case, there are at most $b + 1$ reachable markings. Now, consider the case $|P| > 1$. By induction, the subnet with $|P| - 1$ places has at most $(b+1)^{|P|-1}$ reachable markings. With one more place p, we keep all these possible markings, but add all possibilities for the number of tokens in p. Since p can be marked in $b+1$ different ways, the total number of reachable markings is $(b+1)^{|P|-1} \cdot (b+1) = (b+1)^{|P|}$. □

In turn, an algorithm that inspects all reachable markings to find if M_f is reachable uses exponential time with respect to the input size in the worst case. This raises the question whether this problem can be solved more efficiently. The literature calls this problem the *reachability problem* and shows that it is unlikely that an efficient solution for it exists.

($\text{REACH}_1^{\neg\lambda}$) Reachability Problem for Unlabeled, 1-Bounded P/T Nets

Input:	A 1-bounded, unlabeled system (N, M_0), with N a P/T net, and a marking M.
To decide:	Is there a firing sequence σ with $(N, M_0)[\sigma\rangle M$?

Cheng et al. [20] found that the decision problem $\text{REACH}_1^{\neg\lambda}$ is PSPACE-complete. Completeness means that a more efficient algorithm for this problem implies that all PSPACE-complete problems can be solved more efficiently. This would make the classes NP and PSPACE merge into one, which the literature considers unlikely to be true [19].

Next, we show that the decision problem $\text{ALIGN}_b^{\neg\lambda}$ is at least as difficult to solve as the problem $\text{REACH}_1^{\neg\lambda}$. We then say that $\text{ALIGN}_b^{\neg\lambda}$ is PSPACE-hard. As is standard, we *reduce* the problem $\text{REACH}_1^{\neg\lambda}$ to $\text{ALIGN}_b^{\neg\lambda}$: We assume that there is an efficient algorithm for $\text{ALIGN}_b^{\neg\lambda}$. Then, we show how to use this algorithm to construct one that solves $\text{REACH}_1^{\neg\lambda}$ efficiently. Since the literature considers the existence of such an algorithm to be unlikely, we then conclude that our assumption was false and there is no efficient algorithm for $\text{ALIGN}_b^{\neg\lambda}$. Consequently, the decision problem $\text{ALIGN}_b^{\neg\lambda}$ then is at least as difficult to solve as the problem $\text{REACH}_1^{\neg\lambda}$.

Thus, we need to transform a problem instance of $\text{REACH}_1^{\neg\lambda}$ into an instance of $\text{ALIGN}_b^{\neg\lambda}$. Such a transformation should take at most linear or pseudolinear time and space. Otherwise, if the transformation takes more time, using the efficient algorithm for $\text{ALIGN}_b^{\neg\lambda}$ would require one to first execute the inefficient transformation. This would lead to the overall algorithm for $\text{REACH}_1^{\neg\lambda}$ also being inefficient.

Consider an instance of $\text{REACH}_1^{\neg\lambda}$ where (N, M_0) is a 1-bounded, unlabeled system and M is a marking. The idea is to transform the system (N, M_0) into an accepting system (N, M_0, M) by setting M as the final marking. We then set $\sigma = \varepsilon$ to the empty sequence. In this way, an alignment between σ and a firing sequence of (N, M_0, M) contains only model moves and exists iff M is reachable from M_0. Since the system is unlabeled, each model move has cost 1. Now, we need to find an upper bound for the total cost k of an alignment. This bound must be high enough to allow an algorithm for $\text{ALIGN}_b^{\neg\lambda}$ to find a firing sequence that leads to M if reachable. Lemma 2 shows that such a bound exists.

Lemma 2. *Let $b \in \mathbb{N}_0$, (N, M_0) be a b-bounded system with $N = (P, T, F)$, and $M \in [N, M_0\rangle$ be a reachable marking of (N, M_0). Then, the length of a shortest firing sequence σ from M_0 to M is bounded by $|\sigma| \leq (b+1)^{|P|} - 1$.*

Proof. By contradiction. Since (N, M_0) is b-bounded, we know by Lemma 1 that $|[N, M_0\rangle| \leq (b+1)^{|P|}$. Let $\sigma = t_1 \dots t_n$ be a shortest firing sequence from M_0 to M. Then, by definition, there are markings $M_1, \dots, M_n$ such that $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots \xrightarrow{t_{n-1}} M_{n-1} \xrightarrow{t_n} M_n$ and $M_n = M$. Suppose that $|\sigma| = n > (b+1)^{|P|}$. Since there are only $(b+1)^{|P|}$ different reachable markings, there must be indices $i, j \in \{0, \dots, n\}$ with $i < j$ and $i \neq j$, such that $M_i = M_j$. But then t_{j+1} is enabled in M_i, i.e., $M_i[t_{j+1}\rangle$, and we can shorten σ to $t_1 \dots t_i t_{j+1} \dots t_n$. Thus, σ could not have been the shortest firing sequence from M_0 to M, a contradiction. Thus, our assumption was incorrect, and $n \leq (b+1)^{|P|}$. □

By Lemma 2, we need to fire no more than $(b+1)^{|P|}$ transitions in a b-bounded system $((P, T, F), M_0)$ to reach some arbitrary but fixed marking M. Thus, to reduce $\text{REACH}_1^{\neg\lambda}$ to $\text{ALIGN}_b^{\neg\lambda}$, we set the maximum alignment costs k to $2^{|P|}$, as the input system for $\text{REACH}_1^{\neg\lambda}$ is 1-bounded. Thus, we can now prove our first result for the alignment problem.

Theorem 1. *$\text{ALIGN}_b^{\neg\lambda}$ is PSPACE-hard.*

Proof. By reduction from $\text{REACH}_1^{\neg\lambda}$ to $\text{ALIGN}_b^{\neg\lambda}$. Suppose that we receive a 1-bounded, unlabeled system (N, M_0) with $N = (P, T, F)$ a P/T net and M a marking as input. We want to return whether M is reachable in (N, M_0). To transform this problem into a problem of $\text{ALIGN}_b^{\neg\lambda}$, we first set $\Sigma = T$ as the alphabet, since N is unlabeled. It takes $\mathcal{O}(|P| \cdot |T|)$ time to find the value $|T|$ and $\mathcal{O}(\log_2(|T|))$ bits to store it: We simply count the number of transitions and store this number in binary. Furthermore, we set $\sigma = \varepsilon$, which takes a constant amount of time and space. We also set $M_f = M$, which does not need any extra time or space, since M is already part of the input. Finally, we store the number $k = 2^{|P|}$. We require $\mathcal{O}(|P| \cdot |T|)$ time to find the value $|P|$ and $\mathcal{O}(|P|)$ bits to store k. Thus, we have transformed the problem for $\text{REACH}_1^{\neg\lambda}$ into a problem for $\text{ALIGN}_b^{\neg\lambda}$ using linear time and space.

We now solve $\text{REACH}_1^{\neg\lambda}$ by finding a solution for this new instance of $\text{ALIGN}_b^{\neg\lambda}$. If an algorithm for the decision problem $\text{ALIGN}_b^{\neg\lambda}$ returns *"yes"* as an answer, then there is an alignment γ between $\sigma = \varepsilon$ and a firing sequence of (N, M_0, M) with $cost_{std}(\gamma) \leq k = 2^{|P|}$. Now:

There is an alignment γ for ε and a firing sequence of (N, M_0, M) with $cost_{std}(\gamma) \leq 2^{|P|}$
$\Leftrightarrow$ There is a firing sequence $t_1 \dots t_n$ with $n \leq 2^{|P|}$ and $(N, M_0)[t_1 \dots t_n\rangle M$ (N is unlabeled)
$\Leftrightarrow$ There is a firing sequence $t_1 \dots t_n$ with $(N, M_0)[t_1 \dots t_n\rangle M$ (because of Lemma 2)
$\Leftrightarrow$ The marking M is reachable in the system (N, M_0) (according to Definition 6)

Thus, the algorithm for $\text{ALIGN}_b^{\neg\lambda}$ returns *"yes"* if M is reachable in (N, M_0) and *"no"* otherwise. We have thus shown how to solve an instance of $\text{REACH}_1^{\neg\lambda}$ by using an algorithm for $\text{ALIGN}_b^{\neg\lambda}$. In turn, the decision problem $\text{ALIGN}_b^{\neg\lambda}$ is at least as difficult as the decision problem $\text{REACH}_1^{\neg\lambda}$. Thus, it is PSPACE-hard. □

By Theorem 1, an algorithm that solves $\text{ALIGN}_b^{\neg\lambda}$ needs at least polynomial space. We will now investigate whether polynomial space is enough to solve $\text{ALIGN}_b^{\neg\lambda}$. We use a well-known theorem that allows us to describe a nondeterministic algorithm to show this.

Lemma 3. ***(Savitch's theorem [21])*** *Fix a decision problem with input size n and let $f \in \Omega(\log(n))$ be a space-constructable function. If a nondeterministic Turing machine can solve the problem using $f(n)$ space, then there is a deterministic Turing machine that can solve the same problem using $f(n^2)$ space.*

Due to Lemma 3, every nondeterministic algorithm that uses polynomial space can be transformed into a deterministic algorithm that also uses polynomial space. This theorem is the reason why we never mentioned a class NPSPACE of decision problems that can be solved by a nondeterministic Turing machine with polynomial space: As it turns out, NPSPACE = PSPACE.

We will use Lemma 3 to prove that $\text{ALIGN}_b^{\neg\lambda}$ is part of the class PSPACE by describing a nondeterministic algorithm for it. Recall that the input of $\text{ALIGN}_b^{\neg\lambda}$ consists of an alphabet Σ, a sequence σ, a system (N, M_0) with $N = (P, T, F)$ and a cost-bound k. The algorithm first constructs a new P/T net that can only fire the sequence σ. We call this net a *sequence net*.

Definition 10. ***(Sequence Net).*** *Let Σ be an alphabet and $\sigma = a_1 \dots a_n$ be a sequence of symbols $a_1, \dots, a_n \in \Sigma$. The* sequence net *for σ is a labeled system (N, M_0) with $N = (P, T, F, \Sigma, \lambda)$ where $P = \{q_1, \dots, q_{n+1}\}$, $T = \{t_1, \dots, t_n\}$, $F = \{(q_i, t_i), (t_i, q_{i+1}) \mid i \in \{1, \dots, n\}\}$, $\lambda(t_i) = a_i$ for all $i \in \{1, \dots, n\}$, and $M_0 = [q_1]$.*

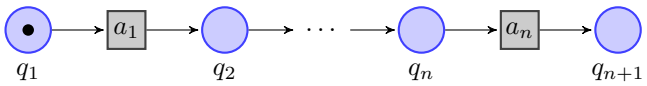


Figure 5: The sequence net (also known as *trace net*) of a sequence $\sigma = a_1 \dots a_n$ that can only fire σ.

Fig. 5 shows the sequence net of some sequence $\sigma = a_1 \dots a_n$. This net has only one maximal firing sequence, $[q_1] \xrightarrow{a_1} [q_2] \xrightarrow{a_2} \dots \xrightarrow{a_n} [q_{n+1}]$. Storing it takes $\mathcal{O}(|\sigma|^2)$ bits, which is a polynomial in the input size. An algorithm to find whether there is an alignment with cost at most k can play the token game on the input system and the sequence net simultaneously. When it fires a transition from one of these nets, it increments a counter for the cost. However, if it fires two transitions with the same label from both nets at the same time, it simulates a synchronous move and we do not increment this counter.

Since we want to know whether an alignment with standard cost at most k exists, we can terminate early if the counter reaches a value greater than k. Thus, we need to reserve $\lceil \log_2(k) \rceil + 1$ bits to store the counter, which is polynomial in the input size. Furthermore, we need to store the current marking for each net to deduce which transitions can fire next. As discussed earlier, storing a marking for a b-bounded net N takes $|P| \cdot \lceil \log_2(b+1) \rceil$ bits. Since the sequence net is 1-bounded and consists of $|\sigma| + 1$ places by construction, we need $|\sigma| + 1$ bits to store a marking for it. Thus, we can store all additional information in a space whose size is polynomial to the input size.

Theorem 2. *$\text{ALIGN}_b^{\neg\lambda}$ is in the class PSPACE.*

Proof. First, we transform the input accepting system into a labeled accepting system. This can be done by choosing the labeling function λ as the identity function, that is, $\lambda(t) = t$ for all transitions $t \in T$. Storing this function takes $\mathcal{O}(|T| \cdot \log_2(|T|))$ bits. We then solve the decision problem as shown in Algorithm 1. Since this algorithm is nondeterministic, it returns True if *any* possible computation returns True. As discussed earlier and in the comments of Algorithm 1, we need an amount of additional bits that is polynomial in the input size. Therefore, $\text{ALIGN}_b^{\neg\lambda}$ can be solved with polynomial space and is thus in the class PSPACE. □

Since $\text{ALIGN}_b^{\neg\lambda}$ is PSPACE-hard and is in the class PSPACE, we know that it is PSPACE-complete. This means if there was a more efficient algorithm to solve a problem instance of $\text{ALIGN}_b^{\neg\lambda}$, all decision problems in the class PSPACE could be solved more efficiently.

Corollary 1. *$\text{ALIGN}_b^{\neg\lambda}$ is PSPACE-complete.*

Algorithm 1: An algorithm to solve an instance of $\text{ALIGN}_b^{\neg\lambda}$ or ALIGN_b^{λ} with polynomial space.

Data: An alphabet Σ, a b-bounded labeled accepting system (N, M_0, M_f), a trace $\sigma = a_1 \dots a_n$ of length $n \in \mathbb{N}_0$, a bound k for the maximum standard cost of an alignment.
Result: True if there is an alignment with $cost_{std}(\gamma) \leq k$, False otherwise.

$N_\sigma \leftarrow (P_\sigma, T_\sigma, F_\sigma, \Sigma, \lambda_\sigma)$ the sequence net for σ with $P_\sigma = \{q_1, \dots, q_{n+1}\}$;
$M \leftarrow M_0$; /* reserve $|P| \cdot \lceil \log_2(b+1) \rceil$ bits for M */
$M_\sigma \leftarrow [q_1]$; /* reserve $|\sigma|$ bits for M_σ */
$c \leftarrow 0$; /* reserve $\lceil \log_2(k) \rceil + 1$ bits for c */
while True **do**
  if $c > k$ **then**
    return False; /* the costs grew higher than k */
  end
  if $M = M_f$ *and* $M_\sigma = [q_{n+1}]$ **then**
    return True; /* both nets reached their final markings */
  end
  $T^{[\rangle} \leftarrow \{t \in T \mid (N, M)[t\rangle\}$; /* takes $|T| \cdot \lceil \log_2(|T|) \rceil$ bits */
  $T_\sigma^{[\rangle]} \leftarrow \{t \in T_\sigma \mid (N_\sigma, M_\sigma)[t\rangle\}$; /* takes $\lceil \log_2(|\Sigma|) \rceil$ bits */
  nondeterministically choose a pair $(t_1, t_2) \in (T^{[\rangle} \cup \{\bot\}) \times (T_\sigma^{[\rangle} \cup \{\bot\})$;
  if $t_1 = \bot$ *and* $t_2 = \bot$ **then**
    return False; /* there are no enabled transitions */
  end
  if $t_1 = \bot$ **then**
    $M_\sigma \leftarrow M_\sigma - {}^\bullet t_2 + t_2^\bullet$;
    $c \leftarrow c + 1$; /* simulated a log move */
  end
  if $t_2 = \bot$ **then**
    $M \leftarrow M - {}^\bullet t_2 + t_2^\bullet$;
    $c \leftarrow c + 1$; /* simulated a model move */
  end
  if $\lambda(t_1) = \lambda(t_2)$ **then**
    $M \leftarrow M - {}^\bullet t_1 + t_1^\bullet$;
    $M_\sigma \leftarrow M_\sigma - {}^\bullet t_2 + t_2^\bullet$;
  end
end

So far, we assumed that the input accepting system is unlabeled. However, the proofs of Theorem 1 and Theorem 2 work in a similar way if we assume labeled accepting systems as input. Thus, we will now quickly investigate the complexity class of the following alignment decision problem.

(ALIGN_b^{λ}) Alignment Decision Problem for Labeled, Bounded P/T Nets

Input:	An alphabet Σ, a sequence $\sigma \in \Sigma^*$, an accepting system (N, M_0, M_f), where N is a labeled, bounded P/T net, and a natural number $k \in \mathbb{N}_0$.
To decide:	Is there an alignment γ between the sequence σ and a firing sequence of (N, M_0, M_f) that leads to the marking M_f and has $cost_{std}(\gamma) \leq k$?

Theorem 3. *ALIGN_b^{λ} is PSPACE-hard.*

Proof. By reduction from $\text{ALIGN}_b^{\neg\lambda}$ to ALIGN_b^{λ}. Both decision problems have the same input, but ALIGN_b^{λ} expects a labeled accepting system. Thus, we use the identity function $\lambda(t) = t$ for every

transition t. Storing this function takes $\mathcal{O}(|P| \cdot |T|)$ time to find the value $|T|$ and $\mathcal{O}(|T| \cdot \log_2(|T|))$ bits to store it. This is pseudolinear in the input size. Then, for the input we received, the result of an algorithm for ALIGN_b^λ is obviously a solution for $\text{ALIGN}_b^{\neg\lambda}$. Thus, ALIGN_b^λ is PSPACE-hard. □

To prove that we can solve the problem with polynomial space, we can reuse Algorithm 1. In the proof of Theorem 2, we first had to transform the accepting system into a labeled accepting system, so we could compare the labels of the two nets. Obviously, we can skip this part if the input net is already labeled and can instead directly apply Algorithm 1 to solve the decision problem.

Theorem 4. *ALIGN_b^λ is in the class PSPACE.*

Proof. Use Algorithm 1 to solve the problem nondeterministically. If there is a run through this program that returns True, the nondeterministic algorithm will return True. Otherwise, it will return False. Therefore, since we did not use any additional space, and ALIGN_b^λ is in the class PSPACE. □

With this, we have proved that the decision problem ALIGN_b^λ is also PSPACE-complete.

Corollary 2. *ALIGN_b^λ is PSPACE-complete.*

4. The Alignment Problem on Unbounded P/T Nets

In this section, we investigate the alignment problem for unbounded P/T nets. If a P/T net is unbounded, a place can hold arbitrarily many tokens. In turn, the reachability graph of the net becomes infinite. For the previous proofs, this means that we cannot store a marking in $|P| \cdot \lceil \log_2(b+1) \rceil$ bits anymore because we do not have an upper bound b for the maximum number of tokens per place. We thus need to find another way to restrict the number of tokens in a marking. As in the previous section, we will first assume that the input net is unlabeled.

($\text{ALIGN}_{\neg b}^{\neg\lambda}$) Alignment Decision Problem for Unlabeled, Unbounded P/T Nets

Input:	An alphabet Σ, a sequence $\sigma \in \Sigma^*$, an accepting system (N, M_0, M_f), where N is an unlabeled, unbounded P/T net, and a natural number $k \in \mathbb{N}_0$.
To decide:	Is there an alignment γ between the sequence σ and a firing sequence of (N, M_0, M_f) that leads to the marking M_f and has $cost_{std}(\gamma) \leq k$?

First, let us investigate whether $\text{ALIGN}_{\neg b}^{\neg\lambda}$ is PSPACE-hard. We can use the same reduction as in Theorem 1. However, we have to turn the input accepting system into an unbounded one in case an algorithm for $\text{ALIGN}_{\neg b}^{\neg\lambda}$ relies on this property. This can be done by introducing a new transition and a new place that are disconnected from the rest of the net, and where the new transition can produce arbitrarily many tokens into the new place.

Theorem 5. *$\text{ALIGN}_{\neg b}^{\neg\lambda}$ is PSPACE-hard.*

Proof. By reduction from $\text{REACH}_1^{\neg\lambda}$ to $\text{ALIGN}_{\neg b}^{\neg\lambda}$. Suppose that we receive a 1-bounded, unlabeled system (N, M_0) with $N = (P, T, F)$ a P/T net and M a marking as input. We transform this system into an unbounded accepting system by adding a new transition t and a new place p to the net, where we add only (t, p) to the new flow relation. Thus, the new transition t can fire indefinitely and store any amount of tokens in the new place p. For this addition to the input net, we need $2(|T| + |P| + 1)$ additional bits of space, which is linear in the input size. To perform this change in the input net, we need $\mathcal{O}(|P| \cdot |T|)$ time, which is also linear to the input size. Furthermore, we expand the initial marking M_0 by $M_0(p) = 0$ and the marking M by $M(p) = 0$, which takes 2 additional bits. The rest of the reduction is the same as in the proof of Theorem 1. Thus, $\text{ALIGN}_{\neg b}^{\neg\lambda}$ is PSPACE-hard. □

To show that the problem is in the class PSPACE, we reuse the ideas of Algorithm 1. However, we cannot reserve the amount of bits indicated in this algorithm for the marking. Instead, we use a property of P/T nets to find a different bound on a marking's space requirements.

Lemma 4. *Let (N, M_0) be a system with $N = (P, T, F)$ a P/T net, σ a firing sequence of (N, M_0), and M a marking with $(N, M_0)[\sigma\rangle M$. Then, for any $p \in P$, $M(p) \leq M_0(p) + |\sigma|$.*

Proof. Suppose $\sigma = t_1 \dots t_n$ for some transitions $t_1, \dots, t_n \in T$. Fix a place $p \in P$. Initially, p contains $M_0(p)$ tokens. For any firing of a transition t_i, $1 \leq i \leq n$, this amount can increase by at most 1, since a place can occur at most once in the postset of a transition. Thus, after firing $t_1 \dots t_n$, the place p can contain at most $M_0(p) + n$ tokens. In turn, $M(p) \leq M_0(p) + |\sigma|$ for any place $p \in P$. □

We use Lemma 4 to define an upper bound for a marking's entries during the execution of Algorithm 1. Since we terminate as soon as the counter runs beyond k, we will never simulate more than $k+1$ model moves. Thus, by model moves alone, a place p can never get more than $M_0(p)+k+1$ tokens during the execution of Algorithm 1. However, if we simulate a synchronous move, we do not increase the counter. Since there can be at most $|\sigma|$ synchronous moves, a place p cannot hold more than $M_0(p)+|\sigma|+k+1$ tokens during the execution of Algorithm 1. With $b_{M_0} = \max\{M_0(p) \mid p \in P\}$, we thus need $|P| \cdot \lceil \log_2(b_{M_0} + |\sigma| + k + 2) \rceil$ bits to store such a marking. Since $\log_2(n+m) \leq \log_2(n) + \log_2(m)$ for any $n, m \in \{2, 3, \dots\}$, we need at most $\mathcal{O}(|P| \cdot (\log_2(b_{M_0}) + \log_2(|\sigma|) + \log_2(k)))$ bits to store the current marking, which is a polynomial in the input size.

Theorem 6. *$\text{ALIGN}^{\neg\lambda}_{\neg b}$ is in the class Pspace.*

Proof. Turn the input accepting system into a labeled accepting system by defining λ as the identity function for transitions, i.e., $\lambda(t) = t$ for any $t \in T$. Use Algorithm 1 to solve the decision problem, but reserve $|P| \cdot \lceil \log_2(b_{M_0} + |\sigma| + k + 2) \rceil$ bits to store the marking M. Since we need additional space that is polynomial in the input size, we get that $\text{ALIGN}^{\neg\lambda}_{\neg b}$ is in the class Pspace. □

As before, since $\text{ALIGN}^{\neg\lambda}_{\neg b}$ is in the class Pspace and it is Pspace-hard, it is Pspace-complete.

Corollary 3. *$\text{ALIGN}^{\neg\lambda}_{\neg b}$ is Pspace-complete.*

Finally, we investigate labeled and unbounded accepting systems. As it turns out, the resulting alignment decision problem is much more difficult to solve than the previous ones.

($\text{ALIGN}^{\lambda}_{\neg b}$) Alignment Decision Problem for Labeled, Unbounded P/T Nets

Input:	An alphabet Σ, a sequence $\sigma \in \Sigma^*$, an accepting system (N, M_0, M_f), where N is a labeled, unbounded P/T net, and a natural number $k \in \mathbb{N}_0$.
To decide:	Is there an alignment γ between the sequence σ and a firing sequence of (N, M_0, M_f) that leads to the marking M_f and has $cost_{std}(\gamma) \leq k$?

The main difficulty is that we now allow silent moves, which infer a cost of 0. By setting all transition labels to τ, $\text{ALIGN}^{\lambda}_{\neg b}$ becomes the problem of whether the final marking M_f is reachable in the system.

(REACH) Reachability Problem for Unbounded P/T Nets

Input:	A system (N, M_0) with $N = (P, T, F)$ an unbounded P/T net, a marking M.
To decide:	Is there a firing sequence $\sigma \in T^*$ with $(N, M_0)[\sigma\rangle M$?

Leroux et al. [22] found that REACH is Ackermann-hard. Later, Leroux [23] found that it is even Ackermann-complete. This means that the runtime complexity of an algorithm solving this decision problem grows as fast as the Ackermann function with respect to the input size. In turn, even for very small instances, solving the decision problem REACH is infeasible. As we find in the next theorem, this property translates to the decision problem $\text{ALIGN}^{\lambda}_{\neg b}$.

Theorem 7. *$\text{ALIGN}^{\lambda}_{\neg b}$ is Ackermann-hard.*

Proof. By reduction from REACH to $\text{ALIGN}^{\lambda}_{\neg b}$. Suppose that we get a system (N, M_0) with an unbounded P/T net $N = (P, T, F)$, as well as a marking M as input. We want to find whether the marking M is reachable in (N, M_0). First, we set $\Sigma = T$, which takes $\mathcal{O}(|P| \cdot |T|)$ time and $\mathcal{O}(\log_2(|T|))$ space. Furthermore, we set $\sigma = \varepsilon$, $M_f = M$, and $k = 0$, which takes constant amount of time and space. Finally, we set $\lambda(t) = \tau$ for every $t \in T$, which takes $\mathcal{O}(|T|)$ time and space. Then, we use an algorithm for $\text{ALIGN}^{\lambda}_{\neg b}$ to find whether there is an alignment γ between $\sigma = \varepsilon$ and the accepting system (N, M_0, M_f) with $cost_{std}(\gamma) \leq k = 0$. Then:

There is an alignment γ between ε and a firing sequence in (N, M_0) with $cost_{std}(\gamma) \leq 0$
$\Leftrightarrow$ There is a firing sequence $t_1 \dots t_n \in T^*$ with $(N, M_0)[t_1 \dots t_n\rangle M_f$ and $\lambda(t_1) = \dots = \lambda(t_n) = \tau$
$\Leftrightarrow$ There is a firing sequence $t_1 \dots t_n \in T^*$ with $(N, M_0)[t_1 \dots t_n\rangle M$
$\Leftrightarrow$ The marking M is reachable in the system (N, M_0).

Thus, the result of an algorithm that solves $\text{ALIGN}^{\lambda}_{\neg b}$ is the same as the result we expect for REACH. In turn, we can use an algorithm for $\text{ALIGN}^{\lambda}_{\neg b}$ to solve REACH, so $\text{ALIGN}^{\lambda}_{\neg b}$ is at least as difficult to solve as REACH, and thus Ackermann-hard. □

We have thus shown that the problem $\text{ALIGN}^{\lambda}_{\neg b}$ is at least as difficult to solve as the problem REACH. By reducing the problem $\text{ALIGN}^{\lambda}_{\neg b}$ to REACH, we can show the converse: Solving a problem instance of REACH is at least as difficult as solving a problem instance of $\text{ALIGN}^{\lambda}_{\neg b}$. In this way, we can show that $\text{ALIGN}^{\lambda}_{\neg b}$ and REACH are equally difficult to solve. Since we know REACH to be Ackermann-complete, we can thus show that the decision problem $\text{ALIGN}^{\lambda}_{\neg b}$ is also Ackermann-complete.

Theorem 8. *$\text{ALIGN}^{\lambda}_{\neg b}$ is Ackermann-complete.*

Proof. Suppose that we get an accepting system (N, M_0, M_f) with a labeled, unbounded P/T net $N = (P, T, F, \Sigma, \lambda)$, a sequence $\sigma \in \Sigma^*$ and a natural number $k \in \mathbb{N}_0$ as input. The goal is to describe an efficient algorithm that decides $\text{ALIGN}^{\lambda}_{\neg b}$ for this input. Suppose there is an efficient algorithm $\mathcal{A}$ that decides the reachability problem REACH. Then, we can use $\mathcal{A}$ to solve $\text{ALIGN}^{\lambda}_{\neg b}$ efficiently as well.

We transform the input for $\text{ALIGN}^{\lambda}_{\neg b}$ into an input for REACH by building the extended *synchronous product net* [24] of the accepting system and the sequence net for σ. Let (N_σ, M_0^σ) be the sequence net for the word σ, where $N_\sigma = (P_\sigma, T_\sigma, F_\sigma, \Sigma, \lambda_\sigma)$. Furthermore, let q_f be the unique place in N_σ with $q_f^\bullet = \emptyset$. Without loss of generality, let $P \cap P_\sigma = \emptyset$ and $T \cap T_\sigma = \emptyset$ (otherwise we rename the places and transitions in one of the nets). Additionally, let $p_c \notin P \cup P_\sigma$ be a fresh place and $t_c \notin T \cup T_\sigma$ be a fresh transition. For the synchronous product net, we first define $T_\otimes = \{(t, t_\sigma) \in T \times T_\sigma \mid \lambda(t) = \lambda_\sigma(t_\sigma)\}$ as all transition pairs from N and N_σ that share the same label. Then, the extended synchronous product net of (N, M_0) and (N_σ, M_0^σ) is the system (N', M_0') where $N' = (P', T', F', \Sigma, \lambda')$ and

- $P' = P \cup P_\sigma \cup \{p_c\}$,
- $T' = T \cup T_\sigma \cup \{t_c\} \cup T_\otimes$,
- $F' = F \cup F_\sigma \cup \{(t_c, p_c)\}$
  $\cup \{(p, (t, t_\sigma)) \in ((P \cup P_\sigma) \times T_\otimes) \mid p \in {}^\bullet t \vee p \in {}^\bullet t_\sigma\}$
  $\cup \{((t, t_\sigma), p) \in (T_\otimes \times (P \cup P_\sigma)) \mid p \in t^\bullet \vee p \in t_\sigma^\bullet\}$
  $\cup (T_\sigma \times \{p_c\}) \cup \{(t, p_c) \in (T \times \{p_c\}) \mid \lambda(t) \neq \tau\}$, and
- $M_0' = M_0 + M_0^\sigma$.

Fig. 6 shows this construction on the example net of Fig. 1 and the sequence acb. The idea is that each firing sequence in the synchronous product net that leads to a marking covering $M_f + [q_f]$ corresponds to an alignment and vice versa [24]. The place p_c counts how often a transition representing an asynchronous move fired. Thus, it tracks the standard cost of the simulated alignment.

With this construction, we introduced $\mathcal{O}(|\sigma|)$ new places and $\mathcal{O}(|\sigma|)$ new transitions. To store them, we need $\mathcal{O}(|P| \cdot |\sigma| + |T| \cdot |\sigma| + |\sigma|^2)$ additional storage space, which is a polynomial of the input size. Similarly, we need $\mathcal{O}(|\sigma|)$ additional bits for the new initial marking $M_0 + M_0^\sigma$. Furthermore, we store a new marking $M = M_f + [q_f] + [p_c^k]$, which needs $\mathcal{O}((|P| + |\sigma|) \cdot \max(\{M_f(p) \mid p \in P\} \cup \{1, \log_2(k)\}))$ bits. Since M_f and k are part of the input, this is also a polynomial in the input size.

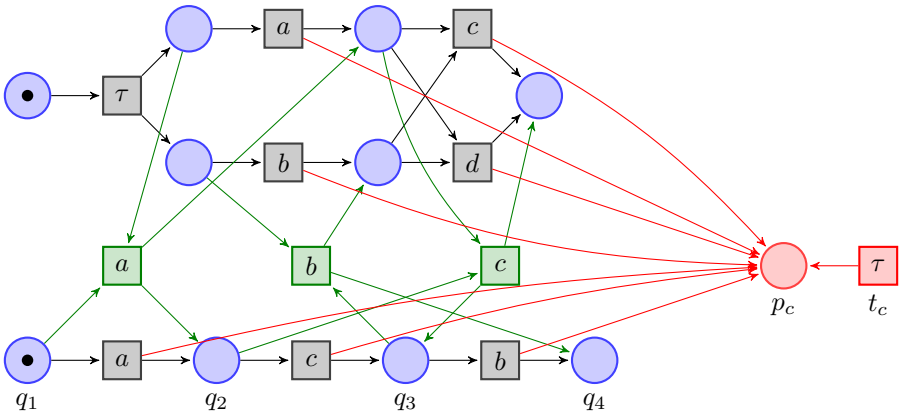


Figure 6: The synchronous net construction for the model of Fig. 1 and the sequence net for acb, together with a counter place p_c for asynchronous moves [24].

Now we can answer whether there is an alignment for σ with standard cost at most k by using the algorithm $\mathcal{A}$ to find whether the marking $M = M_f + [q_f] + [p_c^k]$ is reachable.

The marking M is reachable in (N', M_0')
$\Leftrightarrow$ There is a firing sequence $t_1 \dots t_n \in T'^*$ with $(N', M_0')[t_1 \dots t_n\rangle M'$
$\Leftrightarrow$ There is a firing sequence $t_1' \dots t_m' \in (T \cup T_\sigma \cup T_\otimes)^*$ with $m \leq n$
and a number $k' \in \mathbb{N}$, $k' \leq k$ with $(N', M_0')[t_1' \dots t_m'\rangle(M_f + [q_f] + [p_c^{k'}])$
$\Leftrightarrow$ There is an alignment γ between σ and a firing sequence in (N, M_0, M_f)
with $cost_{std}(\gamma) = k' \leq k$.

Thus, an efficient algorithm $\mathcal{A}$ for the problem REACH can be used to solve the problem $\text{ALIGN}^\lambda_{\neg b}$ efficiently as well. Consequently, the problem $\text{ALIGN}^\lambda_{\neg b}$ is at most as difficult to solve as the problem REACH, placing it in the complexity class Ackermann [23]. Together with the result from Theorem 7, we conclude that $\text{ALIGN}^\lambda_{\neg b}$ is Ackermann-complete. □

5. Conclusion

This paper investigated the decision problem asking whether there is an alignment with standard costs at most k between an activity sequence and a Petri net. Regardless of the labeling, it found that this problem is Pspace-complete for bounded Petri nets. For unbounded Petri nets, the problem is Pspace-complete if the net is unlabeled and Ackermann-complete if it is labeled. These results show that the computation of alignments for labeled, unbounded Petri nets should be avoided in practice.

The main reason why the problem is Ackermann-complete for labeled, unbounded Petri nets is the existence of asynchronous moves that infer zero-costs to an alignment. In the case where other non-negative cost functions than the standard costs are of interest, the proofs of this paper can easily be adapted to receive similar results. As such, for unbounded nets, the alignment problem is Ackermann-complete if the cost function assigns zero-costs to any asynchronous moves and Pspace-complete otherwise. For bounded nets, the problem is Pspace-complete for all non-negative cost functions.

Future work concerns analyzing the computational complexity of the alignment problem for other Petri net classes. This deepens the understanding of the problem and helps practitioners estimate the runtime complexity of their algorithms. If the complexity of the problem they want to solve is too high, they can use techniques such as approximate alignments to get approximate results.

Acknowledgments

The authors thank the reviewers for their constructive comments, and especially Reviewer 3 for their valuable input regarding the proof of Ackermann-completeness.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] W. Van Der Aalst, Process Mining, Springer Berlin Heidelberg, 2016. URL: http://link.springer.com/10.1007/978-3-662-49851-4. doi:10.1007/978-3-662-49851-4.

[2] C. Girault, R. Valk, Petri Nets for Systems Engineering, Springer Berlin Heidelberg, 2003. URL: https://link.springer.com/book/10.1007/978-3-662-05324-9. doi:10.1007/978-3-662-05324-9.

[3] J. Carmona, B. Van Dongen, A. Solti, M. Weidlich, Conformance Checking: Relating Processes and Models, Springer International Publishing, 2018. URL: http://link.springer.com/10.1007/978-3-319-99414-7. doi:10.1007/978-3-319-99414-7.

[4] W. Van Der Aalst, A. Adriansyah, B. Van Dongen, Replaying history on process models for conformance checking and performance analysis, WIREs Data Min & Knowl 2 (2012) 182–192. URL: https://wires.onlinelibrary.wiley.com/doi/10.1002/widm.1045. doi:10.1002/widm.1045.

[5] C. T. Schwanen, W. Pakusa, W. M. P. van der Aalst, Complexity of alignments on sound free-choice workflow nets, in: E. Amparore, Ł. Mikulski (Eds.), Application and Theory of Petri Nets and Concurrency, Springer Nature Switzerland, Cham, 2025, pp. 388–410. doi:10.1007/978-3-031-94634-9_19.

[6] V. I. Levenshtein, et al., Binary codes capable of correcting deletions, insertions, and reversals, in: Soviet physics doklady, volume 10, Soviet Union, 1966, pp. 707–710.

[7] B. F. van Dongen, Efficiently computing alignments, in: M. Weske, M. Montali, I. Weber, J. vom Brocke (Eds.), Business Process Management, Springer International Publishing, Cham, 2018, pp. 197–214. doi:10.1007/978-3-319-98648-7_12.

[8] M. Boltenhagen, T. Chatain, J. Carmona, Optimized SAT encoding of conformance checking artefacts, Computing 103 (2020) 29–50. URL: https://link.springer.com/10.1007/s00607-020-00831-8. doi:10.1007/s00607-020-00831-8.

[9] M. Fani Sani, M. Kabierski, S. J. van Zelst, W. M. P. van der Aalst, Model-independent error bound estimation for conformance checking approximation, in: J. De Weerdt, L. Pufahl (Eds.), Business Process Management Workshops, Springer Nature Switzerland, Cham, 2024, pp. 369–382. doi:10.1007/978-3-031-50974-2_28.

[10] D. Fahland, Describing behavior of processes with many-to-many interactions, in: S. Donatelli, S. Haar (Eds.), Application and Theory of Petri Nets and Concurrency, Springer International Publishing, Cham, 2019, pp. 3–24. doi:10.1007/978-3-030-21571-2_1.

[11] K. van Hee, N. Sidorova, M. Voorhoeve, Soundness and separability of workflow nets in the stepwise refinement approach, in: W. M. P. van der Aalst, E. Best (Eds.), Applications and Theory of Petri Nets 2003, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 337–356. doi:10.1007/3-540-44919-1_22.

[12] K. Van Hee, N. Sidorova, M. Voorhoeve, Resource-constrained workflow nets, Fundamenta Informaticae Vol. 71 (2006). doi:10.3233/FUN-2006-712-306.

[13] J. M. E. M. van der Werf, A. Rivkin, A. Polyvyanyy, M. Montali, Data and process resonance, in: L. Bernardinello, L. Petrucci (Eds.), Application and Theory of Petri Nets and Concurrency, Springer International Publishing, Cham, 2022, pp. 369–392. doi:10.1007/978-3-031-06653-5_19.

[14] D. Sommers, N. Sidorova, B. van Dongen, Aligning event logs to resource-constrained ν-petri nets, in: L. Bernardinello, L. Petrucci (Eds.), Application and Theory of Petri Nets and Concurrency, Springer International Publishing, Cham, 2022, pp. 325–345. doi:10.1007/978-3-031-06653-5_17.

[15] D. Sommers, N. Sidorova, B. van Dongen, Exact and approximated log alignments for processes with inter-case dependencies, in: L. Gomes, R. Lorenz (Eds.), Application and Theory of Petri

Nets and Concurrency, Springer Nature Switzerland, Cham, 2023, pp. 99–119. doi:10.1007/978-3-031-33620-1_6.

[16] D. Sommers, N. Sidorova, B. van Dongen, Conformance checking with model projections, in: L. M. Kristensen, J. M. van der Werf (Eds.), Application and Theory of Petri Nets and Concurrency, Springer Nature Switzerland, Cham, 2024, pp. 61–82. doi:10.1007/978-3-031-61433-0_4.

[17] C. T. Schwanen, W. Pakusa, W. M. P. van der Aalst, Process tree alignments, in: J. Borbinha, T. P. Sales, M. M. da Silva, H. A. Proper, M. Schnellmann (Eds.), Enterprise Design, Operations, and Computing - 28th International Conference, EDOC 2024, Vienna, Austria, September 10-13, 2024, Revised Selected Papers, Lecture Notes in Computer Science, Springer, 2024, pp. 300–317. URL: https://doi.org/10.1007/978-3-031-78338-8_16. doi:10.1007/978-3-031-78338-8_16.

[18] J. Esparza, Decidability and complexity of petri net problems — an introduction, in: W. Reisig, G. Rozenberg (Eds.), Lectures on Petri Nets I: Basic Models: Advances in Petri Nets, Springer Berlin Heidelberg, Berlin, Heidelberg, 1998, pp. 374–428. URL: https://doi.org/10.1007/3-540-65306-6_20. doi:10.1007/3-540-65306-6_20.

[19] C. H. Papadimitriou, Computational Complexity, Addison-Wesley Publishing Compani, Inc., 1994.

[20] A. Cheng, J. Esparza, J. Palsberg, Complexity results for 1-safe nets, Theoretical Computer Science 147 (1995) 117–136. URL: https://www.sciencedirect.com/science/article/pii/0304397594002317. doi:https://doi.org/10.1016/0304-3975(94)00231-7.

[21] W. J. Savitch, Relationships between nondeterministic and deterministic tape complexities, Journal of Computer and System Sciences 4 (1970) 177–192. URL: https://linkinghub.elsevier.com/retrieve/pii/S002200007080006X. doi:10.1016/S0022-0000(70)80006-X.

[22] J. Leroux, S. Schmitz, Reachability in vector addition systems is primitive-recursive in fixed dimension, in: Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '19, IEEE Press, 2021, pp. 1–13.

[23] J. Leroux, The reachability problem for petri nets is not primitive recursive, in: 2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS), 2022, pp. 1241–1252. doi:10.1109/FOCS52979.2021.00121.

[24] A. Adriansyah, N. Sidorova, B. F. van Dongen, Cost-based fitness in conformance checking, in: B. Caillaud, J. Carmona, K. Hiraishi (Eds.), 11th International Conference on Application of Concurrency to System Design, ACSD 2011, Newcastle Upon Tyne, UK, 20-24 June, 2011, IEEE Computer Society, 2011, pp. 57–66. URL: https://doi.org/10.1109/ACSD.2011.19. doi:10.1109/ACSD.2011.19.