Removing Implicit Places Using Regions for Process Discovery

Lisa L. Mannel(✉)[1], Robin Bergenthum[2], and Wil M. P. van der Aalst[1]

[1] PADS Group, RWTH Aachen University
{mannel, wvdaalst}@pads.rwth-aachen.de
[2] FernUniversität in Hagen
robin.bergenthum@fernuni-hagen.de

Abstract. Process discovery aims to derive a process model from an event log. Most discovery techniques use Petri nets or produce results that can be converted to Petri nets. Such discovered Petri nets may contain implicit places, i.e., places that can be removed without changing the behavior of the model. However, implicit places have a negative effect on both the readability and the runtime of analysis techniques. Algorithms that remove implicit places often focus on the structure of a Petri net and need to solve complex optimization problems (e.g., using integer linear programming). Applying a technique adopted from the area of region theory, we show that by replaying an event log on a discovered Petri net, we are able to identify and remove implicit places. The presented approach can be used in conjunction with a variety of discovery algorithms. In this paper, we combine the approach with the eST-Miner which greatly benefits from deleting implicit places during runtime. We present an implementation and show first experimental results.

Keywords: Process Discovery, Petri Nets, Region Theory, Implicit Places

1 Introduction

Nowadays, most information systems record events of the processes they support. These recordings can be stored in the form of an event log. *Process discovery* aims to discover the underlying process of a given event log, extracting order, concurrency, and dependencies from the recorded sequence of events. A broad range of discovery algorithms has been proposed. On one end of this spectrum, there is classical Petri net synthesis [1], which can return very precise models with desirable guarantees but usually lacks integrated noise handling capability and is quite time and space consuming. On the other side, we have specialized discovery algorithms [15] which run faster and can handle noise, but usually perform less well on precision and may not provide guarantees.

Petri nets are often used to represent process models. These Petri nets are either used as input for other algorithms, for example, in the context of conformance checking, or they serve as a graphical process representation to be interpreted by human beings. In the first case, the computational complexity of

the applied algorithms often scales with the size of the given Petri net, while in the second case simplicity and readability are of importance, which are strongly influenced by the size of the net as well. Therefore, we usually aim to provide the smallest Petri net (in the number of places and transitions) that characterizes the desired behavior well.

Unfortunately, process discovery algorithms often produce nets where some of the places do not restrict the behavior, i.e., these places can be removed without changing the language of the net. Such places are called *implicit places* and have been well-studied in literature. Existing approaches aiming to remove such places from a given Petri net, are based on solving time- and space-intensive optimizations problems (Integer Linear Programming is NP-complete) based on the structure of the Petri net [8,14]. Such approaches have been implemented in several tools, for example, in VipTool and ProM [6,17].

Most synthesis approaches avoid returning implicit places by discovering a minimal representation of all feasible places [2,3,9]. In region theory, these are the places corresponding to so-called *minimal regions* [3,4,7], which were originally defined on a transition system that defines the input language. In this paper, we adopt ideas and concepts of these minimal regions to the context of process mining: rather than identifying implicit places using the structure of the Petri net itself or the detour over transition systems, we propose an approach that heavily relies on the input event log as a source of information.

As a proof of concept, we combine the introduced approach with the eST-Miner [10,11]. The eST-Miner enumerates candidate places for a given set of transitions in some clever way. For each place, the miner replays the event log to decide, whether the place is discarded or added to the final result. Here, we present an implementation of the eST-Miner in ProM adapting the new ideas of removing implicit places at runtime. We show the first very promising experimental results of this implementation.

In Section 2 we introduce basic definitions and notation, before presenting and proving the theoretical foundations of our approach. As a first case-study, in Section 3 we describe the combination with the discovery algorithm eST-Miner, and discuss first experimental results. Finally, we conclude the paper in Section 4.

2 Petri nets and Theory of Regions

In this section, we recall the notion of Petri nets and the theory of regions. We adapt ideas of so-called minimal regions to identify places that can be replaced, thus creating a nicer process model. Therefore, we introduce the following notions. Let f be a function and B be a subset of the domain of f. We write $f|_B$ to denote the restriction of f to B. We call a function $m\colon A \to \mathbb{N}_0$ a multiset and write $m = \sum_{a \in A} m(a) \cdot a$ to denote multiplicities of elements in m. We extend some common operators to functions over $\mathbb{Z}$ as follows: for two functions f_1, f_2 and an operator $\circ \in \{=, \leq\}$ we have that $f_1 \circ f_2 \Leftrightarrow \forall x \in X\colon f_1(x) \circ f_2(x)$. We define $f_1 \neq f_2 \Leftrightarrow \exists x \in X\colon f_1(x) \neq f_2(x)$. Based on this we write $f_1 < f_2$ if

$f_1 \leq f_2$ and $f_1 \neq f_2$ hold. Finally, for $\circ \in \{+, -\}$ we have that $f_3 = f_1 \circ f_2 \Leftrightarrow \forall x \in X\colon f_3(x) = f_1(x) \circ f_2(x)$. We denote A^* the set of all finite sequences over elements in A. Let $\sigma_1 = \langle a_1, a_2, \ldots, a_n \rangle$ and $\sigma_2 = \langle b_1, b_2, \ldots, b_m \rangle$ be two sequences, the concatenation of $\sigma_1 \cdot \sigma_2$ is defined by $\langle a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m \rangle$.

Petri nets have formal semantics, an intuitive graphical representation, and are able to express concurrency among the occurrence of activities.

Definition 1 (Petri Nets and Firing Rule). *A* Petri net *is a triple (P, T, W) where P and T are finite disjoint sets of places and transitions, respectively, and $W\colon (P \times T) \cup (T \times P) \to \mathbb{N}_0$ is a multiset of arcs. A multiset $m_0\colon P \to \mathbb{N}_0$ is a* marking *of (P, T, W). We call $N = (P, T, W, m_0)$ a* marked Petri net*, and call m_0 the* initial marking *of N. Let $t \in T$ be a transition. We denote,*

$$\bullet t \coloneqq \sum_{p \in P} W(p, t) \cdot p \text{ the preset of } t \text{ and } t\bullet \coloneqq \sum_{p \in P} W(t, p) \cdot p \text{ the postset of } t.$$

We define the firing rule *as follows: A transition t is* enabled *in marking m_0 if $m_0 \geq \bullet t$ holds. If transition t is enabled, transition t can* fire*. Firing t changes the marking m_0 to*

$$m' \coloneqq m_0 - \bullet t + t\bullet$$

In this case, we write $m_0 \xrightarrow{t} m'$.

Firing a transition changes the marking of a Petri net. Starting at the initial marking, sequentially enabled sequences of transitions define its language.

Definition 2 (Language of a Petri Net). *Let $N = (P, T, W, m_0)$ be a marked Petri net. A sequence of transitions $\sigma = \langle t_1, t_2, \ldots, t_n \rangle \in T^*$ is a firing sequence. The firing sequence σ is enabled in (P, T, W, m_0) if there is a sequence of markings $\langle m_1, m_2, \ldots, m_n \rangle$ such that $m_0 \xrightarrow{t_1} m_1$, $m_1 \xrightarrow{t_2} m_2$, …, $m_{n-1} \xrightarrow{t_n} m_n$ holds. In this case, we write $m_0 \xrightarrow{\sigma} m_n$. Note, that the empty sequence $\langle\rangle \in T^*$ is always enabled, i.e. $\forall m\colon m \xrightarrow{\langle\rangle} m$. We denote $\mathcal{L}(N) \coloneqq \{\sigma \in T^* \mid m\colon P \to \mathbb{N}_0, m_0 \xrightarrow{\sigma} m\}$ the prefix closed language of N.*

In graphical representations, markings of a Petri net are depicted by drawing black dots, called tokens, in places. For example, if the multiplicity of a place in a marking is three, we just show three tokens in that place. Obviously, the number of tokens in places changes during the execution of a sequence according to the firing rule. For every place, the token-function describes the number of tokens based on a given firing sequence.

Definition 3 (Token-function). *Let $N = (P, T, W, m_0)$ be a marked Petri net and $p \in P$ a place. We define the* token-function *$x_p\colon T^* \to \mathbb{Z}$ of p by*

$$x_p(\langle t_1, t_2, \ldots, t_n \rangle) \coloneqq m_0(p) + \sum_{i=1}^{n} (W(t_i, p) - W(p, t_i)),$$

the number of tokens in p after firing $\langle t_1, t_2, \ldots, t_n \rangle$.

Looking at the token-function of a place, we are able to make two simple observations. First, if a net is able to execute a sequence σ, then by Definition 1, for all places, the resulting values of the token-functions are non-negative. Second, by definition of the token-function, every transition adds (or removes) a constant value of tokens to every place. Obviously, these values are defined by the multiset of arcs.

Observation 1 *Let $N = (P, T, W, m_0)$ be a marked Petri net, $\langle t_1, t_2, \ldots, t_n \rangle \in T^*$ a sequence of transitions and $p \in P$ a place with token-function x_p.*

(i) $\langle t_1, t_2, \ldots, t_n \rangle \in \mathcal{L}(N) \Rightarrow \forall i \leq n \colon x_p(\langle t_1, t_2, \ldots, t_{i-1}, t_i \rangle) \geq 0.$

(ii) $x_p(\langle t_1, t_2, \ldots, t_n \rangle) - x_p(\langle t_1, t_2, \ldots, t_{n-1} \rangle) = W(t_n, p) - W(p, t_n).$

Although Observation 1 seems to be straightforward, note that opposite direction of (i) does not hold in general. Consider a place with three tokens and a transition taking four tokens from this place, before putting five tokens back. Even though this transition is not enabled (see Definition 1), by looking at the token-function it seems that the transition is increasing the token-count from three to four.

In the next proposition, we consider Petri nets without so-called self-loops, i.e., for a certain place, every transition can either consume or produce tokens but not both. For such Petri nets, we can obtain an even stronger result.

Proposition 1. *Let $N = (P, T, W, m_0)$ be a marked Petri net without short-loops (i.e. $\forall t \in T, \forall p \in P \colon (W(t, p) = 0 \vee W(p, t) = 0)$), $\sigma = \langle t_1, t_2, \ldots, t_n \rangle \in T^*$ a transition sequence, and $p \in P$ a place with the token-function x_p. We define the one-place net N_p by $N_p \coloneqq (\{p\}, T, W|_{(\{p\} \times T) \cup (T \times \{p\})}, m_0|_{\{p\}})$. Then we have*

$$\forall i \leq n \colon x_p(\langle t_1, t_2, \ldots, t_i \rangle) \geq 0 \Leftrightarrow \sigma \in \mathcal{L}(N_p)$$

Proof (Proposition 1). From Observation 1 we already have that $\sigma \in \mathcal{L}(N_p) \Rightarrow \forall i \leq n \colon x_p(\langle t_1, t_2, \ldots, t_i \rangle) \geq 0$.

It remains to be shown that $\forall i \leq n \colon x_p(\langle t_1, t_2, \ldots, t_i \rangle) \geq 0 \Rightarrow \sigma \in \mathcal{L}(N_p)$. To this end, assume there is a transition t and sequence $\sigma \in \mathcal{L}(N_p)$, such that $\sigma \cdot \langle t \rangle \notin \mathcal{L}(N_p)$. Then $m_0|_{\{p\}} \xrightarrow{\sigma} m|_{\{p\}}$ and t is not enabled in $m|_{\{p\}}$. By definition of the firing rule (Definition 1) and Observation 1 we have that $0 \leq x_p(\sigma) = m|_{\{p\}} < W(p, t)$. Due to not having self-loops this implies $W(t, p) = 0$. Recalling the definition of the token-function (Definition 3) we get $x_p(\sigma \cdot \langle t \rangle) = m|_{\{p\}} - W(p, t) < 0$. □

Adapting the main ideas of the theory of regions, we define a condition to determine whenever a random function on a set of firing sequences is a token-function of some (yet unknown) place.

Definition 4 (Region). *Let $N = (P, T, W, m_0)$ be a marked Petri net without self-loops and let $r \colon \mathcal{L}(N) \to \mathbb{N}_0$ be a function. This function r is a* region *in N, if the following conditions hold. For every $t \in T$ and $\sigma \cdot \langle t \rangle, \sigma' \cdot \langle t \rangle \in \mathcal{L}(N)$:*

$$r(\sigma \cdot \langle t \rangle) - r(\sigma) = r(\sigma' \cdot \langle t \rangle) - r(\sigma')$$

In Definition 4 we consider Petri nets without self-loops, implying that the difference of the token-function of a place before and after executing a transition directly determines the number (and direction) of arcs connecting the place to that transition. For example, if firing a transition changes the token-function of a place by adding three tokens, there are three arcs leading from the transition to that place (i.e., an arc with weight 3).

Definition 5 (Place of a Region). *Let $N = (P, T, W, m_0)$ be a marked Petri net without self-loops, let r be a region in N, and let $p \notin P$ be a place. Assume that for every $t \in T$ there is a sequence $\langle t_1, \ldots, t_n, t \rangle \in \mathcal{L}(N)$.*

Then we can define a weight-function $w \colon T \to \mathbb{Z}$ by $w \coloneqq r(\langle t_1, \ldots, t_n, t \rangle) - r(\langle t_1, \ldots, t_n \rangle)$. Using this well-defined weight-function, we define a multiset of arcs $W' \colon (\{p\} \times T) \cup (T \times \{p\}) \to \mathbb{N}_0$, connecting p to the set of transitions T:

$$W' \coloneqq \sum_{\{t \mid w(t) \geq 0\}} w(t) \cdot (t, p) + \sum_{\{t \mid w(t) < 0\}} -w(t) \cdot (p, t).$$

We call p (connected via W') the place of r.

Observation 2 *Let $N = (P, T, W, m_0)$ be a marked Petri net without self-loops, such that for all $t \in T$ there is at least one trace $\langle t_1, \ldots, t_n, t \rangle \in \mathcal{L}(N)$. Let r be a region in N, and let $p \notin P$ be the place of r (Definition 5). Considering the marked Petri net $N_p = (\{p\}, T, W', r(\langle\rangle))$, by construction, r is the token-function of p. Since r is a region, it is non-negative for all $\sigma \in \mathcal{L}(N)$. According to Proposition 1, $\mathcal{L}(N) \subseteq \mathcal{L}(N_p)$ holds. Thus, adding p to N via W' does not change the language of N, i.e. $\mathcal{L}(N) = \mathcal{L}((P \cup \{p\}, T, W \cup W', m_0 + r(\langle\rangle)))$.*

In this paper, our goal is to identify places that can be deleted or replaced to improve the model using regions. We improve the model by reducing the number of tokens in places during the execution of the event log. This idea is adapted from region theory as well. The related concept is called minimal regions. However, in this paper, we want to identify pairs of places that define a region leading to a place carrying fewer tokens according to its token-function.

Proposition 2 (Decreased Region). *Let $N = (P, T, W, m_0)$ be a marked Petri net without self-loops, such that for all $t \in T$ there is at least one trace $\langle t_1, \ldots, t_n, t \rangle \in \mathcal{L}(N)$. Let $p_1, p_2 \in P$ be two places, and x_{p_1}, x_{p_2} be the corresponding token-functions. If $x_{p_1}|_{\mathcal{L}(N)} > x_{p_2}|_{\mathcal{L}(N)}$ then $(x_{p_1} - x_{p_2})|_{\mathcal{L}(N)}$ is a region.*

Let p_3 be the place of $(x_{p_1} - x_{p_2})|_{\mathcal{L}(N)}$ (connected via its weight-function). We can replace p_1 by p_3 without changing the language of N.

Proof (Proposition 2). If $x_{p_1}|_{\mathcal{L}(N)} > x_{p_2}|_{\mathcal{L}(N)}$, then $(x_{p_1} - x_{p_2})|_{\mathcal{L}(N)}$ is non-negative. Let $t \in T$ be a transition and $\sigma{\cdot}\langle t\rangle \in \mathcal{L}(N)$ a firing sequence. Then, by definition of the token-function (Definition 3), we have

$$\begin{aligned}
&(x_{p_1} - x_{p_2})|_{\mathcal{L}(N)}(\sigma{\cdot}\langle t\rangle) - (x_{p_1} - x_{p_2})|_{\mathcal{L}(N)}(\sigma)\\
=&\big(x_{p_1}|_{\mathcal{L}(N)}(\sigma{\cdot}\langle t\rangle) - x_{p_2}|_{\mathcal{L}(N)}(\sigma{\cdot}\langle t\rangle)\big) - \big(x_{p_1}|_{\mathcal{L}(N)}(\sigma) - x_{p_2}|_{\mathcal{L}(N)}(\sigma)\big)\\
=&x_{p_1}|_{\mathcal{L}(N)}(\sigma) + W(t,p_1) - W(p_1,t) - x_{p_2}|_{\mathcal{L}(N)}(\sigma) - W(t,p_2) + W(p_2,t)\\
&- x_{p_1}|_{\mathcal{L}(N)}(\sigma) + x_{p_2}|_{\mathcal{L}(N)}(\sigma)\\
=&W(t,p_1) - W(p_1,t) - W(t,p_2) + W(p_2,t)
\end{aligned}$$

This shows that σ, $(x_{p_1}-x_{p_2})|_{\mathcal{L}(N)}(\sigma{\cdot}\langle t\rangle)-(x_{p_1}-x_{p_2})|_{\mathcal{L}(N)}(\sigma)$ is a fixed value for every t (independent of σ), and thus, $(x_{p_1} - x_{p_2})|_{\mathcal{L}(N)}$ is a region (Definition 4).

Recall, that a region is the token-function of its place by construction (Observation 2). Therefore, $x_{p_3} = x_{p_1} - x_{p_2}$ is the token-function of the place p_3. According to Observation 2, we can add p_3 (via its weight-function) to N without changing the language of N.

It remains to be shown, that removing p_1 from the Petri net N with p_3 added, does not change the language of the net. Let $\sigma \in \mathcal{L}(N)$ be a transition sequence. Since $x_{p_1}(\sigma) = x_{p_2}(\sigma) + x_{p_3}(\sigma)$ holds, we have $x_{p_1}(\sigma) < 0 \Rightarrow (x_{p_2}(\sigma) < 0) \vee (x_{p_3}(\sigma) < 0)$. Looking at Proposition 1 we can remove p_1 as long as we keep p_2 and p_3 without changing the language of N. □

3 Case Study Using eST-Miner

There are some conceptual differences between Petri net synthesis in general and process discovery in particular. In process discovery, we are usually looking for the control flow of an end-to-end process, which is described by a given event log. An event log is a multiset of *traces*, and a trace is a sequence of *activities*. When replaying an event log on a Petri net, executing an activity in the log corresponds to firing a transition with the corresponding label. In this paper, we assume a bijective mapping between activities and transitions.

In contrast to classical Petri net synthesis, process mining usually focuses on unweighted Petri nets, meaning that every arc is either contained in the net or not contained in the net. Additionally, many Petri nets modeling a process contain self-loops. To model the proper completion of a process instance, we focus on a subset of the language of the Petri net, that contains only the sequences ending in a given final marking, i.e., the language is not prefix-closed.

Models discovered in process mining are usually interpreted by human beings, or used as input for further computations, e.g. in the context of conformance checking. Implicit places are undesirable, because they make interpretation of the model and further computations unnecessarily complex. Fortunately, we can adapt the approach summarized in Proposition 2 in the previous section to identify and remove such places.

The basic idea of applying the approach to process mining is quite straightforward. As input, we expect an event log and the corresponding model. For a

pair of places p_1, p_2 we replay the log and compare the markings of the places after each fired transition. If $x_{p_1}|_L > x_{p_2}|_L$, we compute the place p_3 of the region $(x_{p_1} - x_{p_2})|_L$. If p_3 is present in the model, we identify p_1 to be implicit.

In the following case study, we investigate the combination of Proposition 2 with the discovery algorithm eST-Miner.

Combination with eST-Miner: The discovery algorithm eST-Miner obtains a minimal overapproximation of the log by enumerating and evaluating all possible places, and then returning the subset consisting of all fitting places. In this context, a place is considered to be fitting if it is

- *feasible* with respect to the log,
- connected by *unweighted* arcs, and
- *empty* in the beginning and at the end of replay.

Additionally, places discovered may contain *self-loops*.

Certain properties make eST-Miner particularly well suited to be combined with our approach. The set of places discovered by eST-Miner is *guaranteed to contain all fitting places*, that is, a few desirable places next to a vast number of implicit places. Due to the many superfluous places, traditional approaches of implicit place identification take a lot of time and space, and thus this algorithm can strongly benefit from our approach. On the other hand, our approach of identifying implicit places can exploit the fact that all fitting places are guaranteed to be discovered, to avoid some computations and checks and thus significantly decrease computation time. Details will be provided in the description of our implementation.

Note, that eST-Miner adds artificial start and end activities to the log, and thus the discovered model. However, we apply our approach after this modification, and therefore do not require any special treatment of these transitions, i.e. we don't care about modifications of the log done before starting the discovery phase of the eST-Miner.

The notion of fitting places, as described above, calls for some modifications to allow for the application of the results presented in Proposition 2. In particular, we need to take a closer look at the requirement of places being empty at the beginning and end of replay, the restriction to unweighted arcs, and self-loops.

Emptiness at beginning and end: Recall, that places are considered to be fitting only if they are empty at the end and beginning of replaying a trace. Given two fitting places p_1, p_2 with $x_{p_1}|_L > x_{p_2}|_L$, we can guarantee by construction that the corresponding place p_3 is empty at the beginning and end of the replay as well: since we have $x_{p_3} = x_{p_1} - x_{p_2}$, $x_{p_1} = x_{p_2} = 0$ implies $x_{p_3} = 0$. This guarantee is needed to ensure that the place p_3 is indeed fitting and therefore discovered by eST-Miner at some point.

Handling unweighted arcs: When comparing the two places p_1 and p_2, a problem arises due to the restriction to unweighted arcs: even though all places discovered by eST-Miner, including p_1 and p_2, are unweighted (i.e. arc-weight

1), the place p_3 based on the region $(x_{p_1} - x_{p_2})|_L$ might be connected by arcs with weight up to 2 by construction. See Figure 1 for an example. Therefore, if we have $x_{p_1}|_L > x_{p_2}|_L$, we need to verify that the corresponding place p_3 is connected by arcs that do not need weights. Only then can we declare p_1 implicit. Fortunately, this check does not require significant computation time.

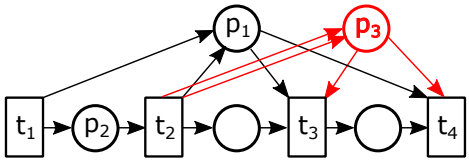

Fig. 1. Consider the event log $L = 10 \cdot \langle t_1, t_2, t_3, t_4 \rangle$ and the given Petri net. We have that $x_{p_1}|_L > x_{p_2}|_L$ and the corresponding place p_3 (red) requires an arc with weight 2.

Handling self-loops: The places returned by eST-Miner often do contain self-loops, not allowing us to directly apply Proposition 2. To avoid this problem, we can easily transform the Petri net by replacing each transition t by two new transitions t_1, t_2 and a place p_t, such that $\bullet t = \bullet t_1, t_1 \bullet = p_t, \bullet t_2 = p_t$ and $t \bullet = t_2 \bullet$ (see Figure 2). The resulting net is free of self-loops. For two original places without self-loops, this has no impact on the relation between their markings. For places with self-loops, this transformation allows to represent the removal of the token when executing the self-loop (which is stored in the added place). This results in a decrease in some reachable markings compared to the original net, but never increases them.

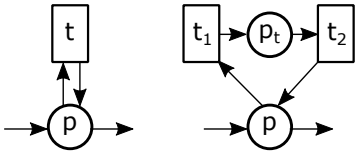

Fig. 2. Transforming the transition t in a Petri net with self-loops (left) to the transitions t_1, t_2 in the corresponding Petri net without self-loops (right).

In our implementation, rather than transforming the Petri net, we simply split the execution of a transition into a consuming event and a producing event. In other words, we replay the log as if each transition was split to identify implicit places.

While this strategy ensures, that we do not incorrectly identify a place as implicit due to self-loops, there is a special case in which it prevents us from correctly classifying a place p_1 as implicit: if p_1 connects the entry transition of

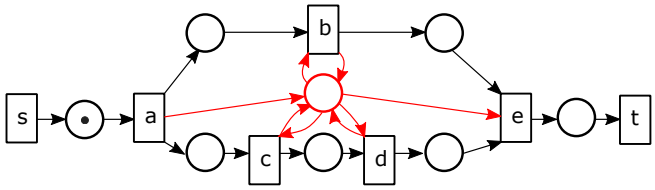

Fig. 3. Consider the Petri net discovered from the log $L = 2 \cdot \langle s, a, b, c, d, e, t \rangle + 1 \cdot \langle s, a, c, b, d, e, t \rangle + 3 \cdot \langle s, a, c, d, b, e, t \rangle$. Marked in red is an implicit place remaining due to self-loops in a parallel construct.

a parallel construct to its exit transition and has the largest set of self-loops of all such places, then we cannot detect it as implicit. Assuming the log reflects the parallel behaviour, there are traces containing the respective transitions in varying order. Thus, for every place p_2 that we can compare to p_1, there is an execution order given by a trace such that p_2 retains all its tokens, while a self-loop on p_1 briefly consumes a token before reproducing it, preventing us from detecting $p_1|_L > p_2|_L$. For an example see Figure 3. Note, that all places p_1', that connect entry and exit transitions of the parallel construct but have only a subset of the self-loops of p_1, are removed because we have $p_1'|_L > p_1|_L$.

Implementation: We implemented two variants of combining our approach of implicit place removal with eST-Miner, both of which incorporate the modifications described above. Moreover, we can exploit some of eST-Miner's properties, to make the overall approach more time- and, in particular, space-efficient.

We call the more straight-forward variant *Final Place Removal (FPR)*. It waits for the eST-Miner to return the final set of fitting places, and then compares pairs of places p_1, p_2 in random order. If $x_{p_1}|_L > x_{p2}|_L$, consider the corresponding place p_3. Since p_1 and p_2 are feasible and empty in the beginning and end of replay, we know that p_3 is feasible (Proposition 2) and empty in the beginning and end of replay (see above). We verify that the arcs connecting it are unweighted to ensure it is a fitting place. Then we can safely identify p_1 to be implicit: we do not need to check whether p_3 is actually contained in the set of places, since eST-Miner guarantees to find all fitting places, in particular p_3. Note, that the relation $<$ is transitive, and therefore we will end up with the same set of minimal places, regardless of the order of place removal.

The second variant integrates the test for implicitness directly into the search phase of eST-Miner, rather than waiting for a fully discovered Petri net. We call this variant *Concurrent Place Removal (CPR)*. Recall, that eST-Miner enumerates and evaluates all possible places sequentially. For every place that is identified as fitting, we run a comparison with all previously discovered places. If for two such places p_1, p_2 we have $x_{p_1}|_L > x_{p_2}|_L$, we compute the corresponding place p_3 and verify its arc weights. If the check is successful, we know p_3 is fitting and will therefore be discovered by eST-Miner, meaning that we can safely remove p_1. The set of places we keep in memory is kept as small as possible, since we retain only the currently minimal places. Due to transitivity of $<$ and

the eST-Miner's guarantee to discover all fitting places, we will end up with the same final set of places as FPR.

For both variants, we can further reduce the number of comparisons, and thereby running time, by focusing on a particularly interesting subset of places, that is, places which share an input transition with the place p_1. If p_1 is implicit, there will be such a place allowing us to identify this: the marking of p_1 constitutes the sum of the markings of the corresponding places p_2 and p_3, and thus, whenever the number of tokens in p_1 is increased by a transition, this transition must increase the tokens of p_2 or p_3 as well. Note that we could focus on shared output transitions instead, but one of them is sufficient.

With respect to time complexity, in the worst case we have to replay the whole log L for each pair of places in $N = (P, T, W)$. Denoting the number of events in L by $|E(L)| = \sum_{\sigma \in L} |\sigma|$, this results in a theoretical time complexity of $\mathcal{O}(|P| \cdot |P| \cdot |E(L)|)$. The eST-Miner may return all possible places as fitting places in the worst case, leading to $\mathcal{O}(2^{|T|} \cdot 2^{|T|})$ places as input to our implicit place removal approach. Thus, the overall worst-case time complexity of the approach combined with eST-Miner is $\mathcal{O}(2^{|T|} \cdot |E(L)|)$. However, on real life event logs we expect a much better performance, since we can utilize the properties of the given log and Petri net to skip many unnecessary computational steps. The experimental results presented below confirm these expectations.

Our implementation is available as a plugin in ProM ([17]). The first experimental results are given in the following and seem very promising.

Experimental results: We tested the approach on various logs, as listed in Table 1. Implicit places are reliably removed as expected. Remaining are very few implicit places due to self-loops and parallel constructs as discussed earlier in this section. Fortunately, our experiments indicate, that these places are a very minor problem: for the tested logs, out of thousands of implicit places, about one such place may remain in the final model. The resulting nets are small enough to easily apply the traditional ILP-based approaches to identify the remaining implicit places. Alternatively, one could apply simple heuristics to remove places with too many self-loops.

Besides the correct removal of implicit places, other important quality criteria are the time and space efficiency of the approach. We summarize some statistical results in Figures 4 and 5.

With respect to space-efficiency, the eST-Miner has the great advantage of having minimal space requirements next to the log and final result, since only one place is stored and evaluated at the same time. The same holds for both variants of our implicit place removal approach. However, when combining eST-Miner with our FPR variant, the intermediate set of fitting places can take significantly more space than the final Petri net. Our experiments show, that the CPR variant effectively avoids such an explosion of stored places. Figure 4 illustrates the

[1] This log has not yet been published. The 2017 version [13] is much smaller.

[2] This log was generated to test the discovery of complex non-free choice structures and self-loops.

Table 1. List of logs used for evaluation. The upper part lists real-life logs while the lower part shows artificial logs. Logs are referred to by their abbreviations. The `Sepsis` log and the `HP2018` log have been reduced by removing all traces that occur only once.

Log Name	Abbreviation	Activities	Trace Variants
Sepsis-sto [12]	`Sepsis-sto`	12	62
HelpDesk2018SiavAnon-sto [1]	`HD18-sto`	11	595
Road Traffic Fine Management [5]	`RTFM`	11	231
Teleclaims [16]	`Teleclaims`	11	12
ILP-Milestone [2]	`ILP-Milestone`	6	4
Loop-Version2 [2]	`LoopV2`	5	28

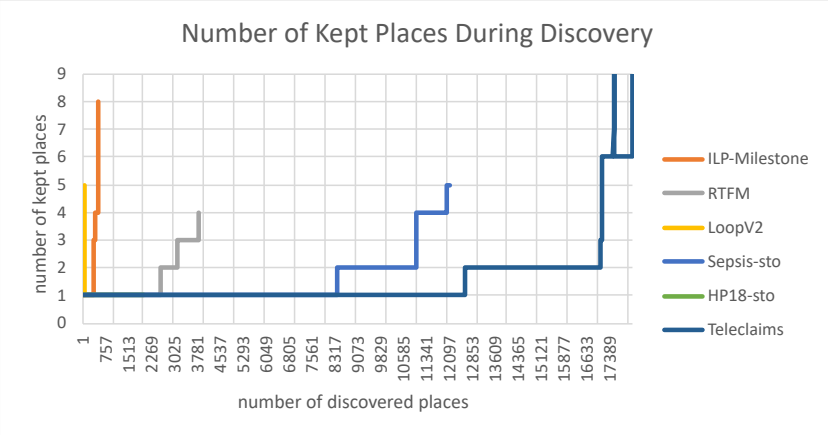

Fig. 4. Number of places in model, that are stored when combining eST-Miner with CPR for various logs.

relation between the number of discovered places versus the number of stored places for this approach on various logs.

For the time efficiency, the number of performed place comparisons is of upmost importance. In Figure 5, we compare the variants FPR and CPR using several measures. For each log investigated, we show the number of fitting places and the number of final places remaining after removing the implicit ones. For each variant, the time needed for implicit place removal and the number of comparisons between pairs of places is given, as well as the ratio between them. Interestingly, the CPR variant results in fewer comparisons made and significantly less time is needed for all our investigated logs. In fact, the CPR variant proves to be up to 86 times faster than the FPR variant.

For the ratio between the time needed and the number of place comparisons on the other hand, no such clear relation could be obtained. One possible explanation for this might be, that even though the final result is independent from the order of places removed, the computational overhead can be reduced

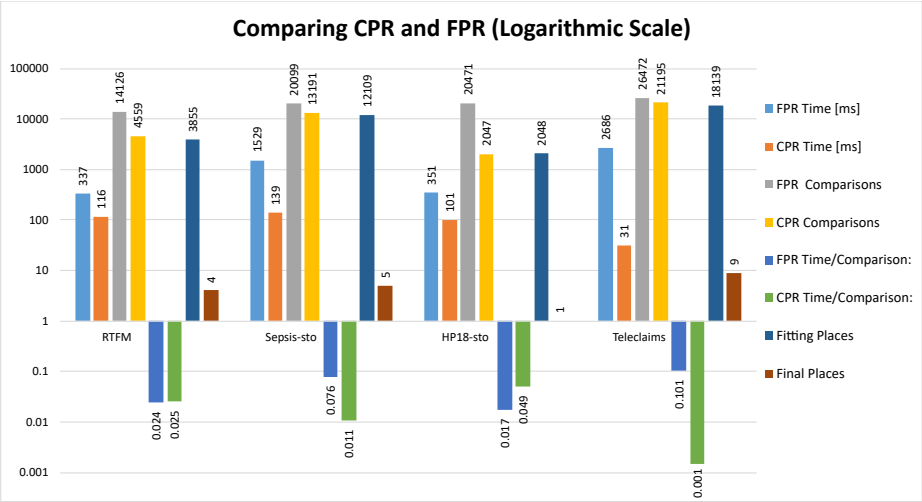

Fig. 5. Comparing CPR and FPR for various logs. CPR performs fewer comparisons and proves to be *up to* 86 *times faster* than FPR. For the ration between time and number of comparisons there is no such clear relation. The number of discovered fitting and remaining final places is the same for both variants, as to be expected.

by choosing this order in a smart way. Additional experiments are needed to verify this theory.

4 Conclusion

The results presented above strongly support the applicability of our approach in process mining. For future research, we have identified two main topics:

First, we are planning further testing and improvement of the combination with eST-Miner. In particular, we would like to explore a possible adaption to the noise handling abilities of this miner, as well as a potential use of implicitness results to improve the search phase by skipping sets of places that we can deduce to be implicit. Also, the concurrent approach (CPR) could be interrupted any time to return the current result, which improves the longer we allow the algorithm to run. This could be particularly interesting when using a variant of eST-Miner that returns places ordered by selected features of interest. Another purpose of investigating certain place orderings is the possible decrease of place comparisons needed to identify implicit places, and thus a decrease in computation time.

Second, we will investigate the more general case of applying our approach to identify implicit places to a given log and a Petri net originating from another source, e.g. another discovery algorithm or user input. Here we are especially interested in possible guarantees that we can provide based on properties of the given log and net: if the event log is complete with respect to the language

defined by the Petri net, we can guarantee correct and complete implicit place identification, however, the effect of weaker requirements on the input are certainly interesting. In this context, we will also investigate the potential of our approach to not only remove implicit places, but to add places that improve a given net to define a behaviour more similar to the given event log.

Acknowledgments: We thank the Alexander von Humboldt (AvH) Stiftung for supporting our research.

References

1. E. Badouel, L. Bernardinello, and P. Darondeau. *Petri Net Synthesis*. Text in Theoretical Computer Science, an EATCS Series. Springer, November 2015.
2. R. Bergenthum. Prime miner - process discovery using prime event structures. In *International Conference on Process Mining, ICPM 2019, Aachen, Germany, June 24-26, 2019*, pages 41–48. IEEE, 2019.
3. J. Carmona, J. Cortadella, and M. Kishinevsky. A region-based algorithm for discovering Petri nets from event logs. volume 5240, pages 358–373, 09 2008.
4. J. Carmona, J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. A symbolic algorithm for the synthesis of bounded Petri nets. pages 92–111, 06 2008.
5. De Leoni, M. and Mannhardt, F. Road traffic fine management process, 2015.
6. J. Desel, G. Juhás, R. Lorenz, and C. Neumair. Modelling and validation with viptool. In W. M. P. van der Aalst and M. Weske, editors, *Business Process Management*, pages 380–389. Springer, 2003.
7. A. Ehrenfeucht and G. Rozenberg. Partial (set) 2-structures. *Acta Informatica*, 27(4):343–368, Mar 1990.
8. F. Garcia-Valles and J.M. Colom. Implicit places in net systems. *Proceedings 8th International Workshop on Petri Nets and Performance Models*, pages 104–113, 1999.
9. R. Lorenz, R. Bergenthum, J. Desel, and S. Mauser. Synthesis of Petri nets from finite partial languages. volume 88, pages 157 – 166, 08 2007.
10. L.L. Mannel and W.M.P. van der Aalst. Finding complex process-structures by exploiting the token-game. In *Application and Theory of Petri Nets and Concurrency*. Springer, 2019.
11. L.L. Mannel and W.M.P. van der Aalst. Finding uniwired Petri nets using eST-miner. In *Business Process Intelligence Workshop 2019*. Springer, to be published.
12. Mannhardt, F. Sepsis cases - event log, 2016.
13. Polato, M. Dataset belonging to the help desk log of an Italian company, 2017.
14. M. Silva, E. Terue, and J. M. Colom. *Linear algebraic and linear programming techniques for the analysis of place/transition net systems*, pages 309–373. Springer, 1998.
15. W.M.P. van der Aalst. *Process Mining: Data Science in Action*. Springer, 2 edition, 2016.
16. van der Aalst, W.M.P. Event logs and models used in Process Mining: Data Science in Action, 2016.
17. B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A new era in process mining tool support. In *Applications and Theory of Petri Nets 2005*, pages 444–454. Springer, 2005.