Removing Implicit Places Using Regions for Process Discovery

Seminar - Selected Topics in Process Mining

Faizan Zafar

Chair of Process and Data Science
RWTH Aachen University
Germany
faizan.zafar@rwth-aachen.de

Abstract. Several processes executed in organizations are handled by information systems which record event data in event logs. One of the ideas behind process discovery is to discover process models using various approaches tailored to different types of models. A Petri net is one form of a process model. We aim to build simple Petri nets that are able to reproduce the behaviour observed in the event log. An implicit place occurrence within a Petri net would be such that its removal does not affect the behaviour executed by the Petri net. To provide fine models, we aim to search for simple places while avoiding adding implicit places. In this paper, we describe a technique to identify and remove implicit places that, in contrast to existing methods that are based solely on the structure of the Petri net, takes into consideration the information contained in the event log. The technique takes inspiration from the theory of regions which transforms a transition system or a formal language into a Petri net. Additionally, we discuss the application of this technique to improve the eST-Miner which focuses on finding fitting places in process models. Finally, we highlight statistical results relevant to the implementation outcomes of the proposed technique.

1 Introduction

The pervasive use of information systems in organizations has driven their expansion into the management of the organizations' business processes. These information systems are well equipped to store or log sequential event data pertinent to the business processes. The discipline of *process mining* proves to be particularly useful for using this event data and gaining insights into the operational workflow of the organizations. In particular, the goal of *process discovery* is to construct a process model from an event log such that it describes the behaviour defined by the given log. An event log can be regarded as a multiset of sequences of activities or events. Each sequence corresponds to one example run of the process. Based on this multiset of example runs, we aim to discover a process model that describes the underlying process.

Processes can be represented in various ways but we focus on modelling them using Petri nets. A Petri net is a theoretical model composed of places and transitions. In particular, we focus on Petri nets where each activity in the log corresponds to exactly one transition. Hence, process discovery is concerned with finding the set of *best* places connecting the transitions.

Petri nets may accommodate *implicit* places which harbour the property that their presence or absence in a Petri net does not alter the language of the Petri net. Implicit places prove to be disadvantageous for process mining purposes since they do not contribute meaningful behaviour to the Petri net. They do not restrict the behaviour of the model more than the minimal set of existing places already does. Implicit places basically clutter the model, make it unreadable for human interpretation, decrease model simplicity, and also have a negative impact on future tasks such as conformance checking. Hence, removal of implicit places can benefit the process discovery phase (and subsequent phases) in process mining.

In this paper, we examine the development of an approach for identifying and removing implicit places as proposed in [10]. Additionally, we describe a practical application of this approach to a known process discovery algorithm - namely, the eST-Miner. The eST-Miner discovers Petri nets starting from a place-less Petri net with one transition for each activity in the event log. Then, the maximal set of *fitting* places is inserted into the Petri net [11]. We discuss how the eST-Miner could immensely benefit from this technique.

The remaining paper is structured as follows: Section 2 presents the preliminaries of Petri nets followed by Section 3 which outlines a primer on region theory along with indicating the motivation behind the proposed technique. Section 4 gives the theoretical and practical details pertinent to the proposed algorithm, along with its running example. Section 5 considers the application of the proposed technique to the eST-Miner ensued by Section 6 which discusses the evaluation benchmarks of the implementation of the technique. Section 7 briefly discusses the related work relevant to the technique. Finally, Section 8 concludes the discussed work as well as possible extensions for future work and ideas for improvement.

2 Preliminaries

In this section, we introduce the preliminaries associated with the discussed work.

2.1 Petri net Theory

Activity, Trace, Event Log Let $\mathcal{A}$ denote the universe of all possible activities or events, denoted by ϵ, a, b, c and d (with ϵ denoting the empty sequence) in Figure 1.

A trace is a sequence containing a single start activity as the first element, several intermediate activities, and a single end activity as the last element. This is denoted by $\langle a, b, c, d \rangle$ in Figure 1.

An event log is a multiset of traces, denoted by L in Figure 1. Note that the trace $\langle a, b, c, d\rangle$ occurs 4 times in the log as indicated.

Petri net Petri nets have a rigorous mathematical foundation and a substantial body of process mining theory has been dedicated for their formal analysis in literature [1].

A *Petri net* is a tuple (P, T, F) where P is a finite collection of places, T is a finite collection of transitions such that there are no common elements between P and T (i.e, $P \cap T = \emptyset$), and F is the flow relation as a multiset of arcs (i.e, $F \subseteq (P \times T) \cup (T \times P) \to \mathbb{N}_0$).

As shown in Figure 1, a Petri net consists of four modelling aspects - namely, places, transitions, arcs and tokens. In a graphical representation, transitions are illustrated as boxes and places as circles. Transitions represent activities contained in the event log while places represent the constraints connecting these transitions. Places and transitions are connected via directed arcs. We denote $\bullet t$ as the set of input places to a transition and $t\bullet$ as the set of output places of a transition. The concept of tokens is deployed to symbolize the occurrences during the execution of a process trace onto a Petri net. The presence of a token inside a place is illustrated by a black dot and indicates the fulfilment of a certain condition. Each place may contain one or several tokens. A *marking* is described as the distribution of token(s) over the place(s). A multiset $m_0 \to \mathbb{N}_0$ is a marking of the Petri net (N, m_0). (N, m_0) is classified as a *marked* Petri net with m_0 as the *initial marking* of N.

A transition is *enabled* if each of its input places contains at least one token i.e. $m_0 \geq \bullet t$. A transition is *dead* at a marking m if it is not enabled at any marking reachable from m. We consider the firing rule of marked Petri nets where the execution of an enabled transition results in one token being consumed from each of its input places and one token being produced in each of its output places. Mathematically, for trace t, this rule is defined as: $(P; T; F; m_0) \overset{\mathrm{t}}{\Rightarrow} (P; T; F; m_0 - \bullet t + t\bullet)$.

Language of a Petri net For a marked Petri net (N, m_0), a sequence of transitions $\sigma = \langle t_1, ..., t_n\rangle$ can possibly be a firing sequence within a Petri net. This firing sequence can be regarded as enabled if there is a sequence of consequent markings $\langle t_1, ..., t_n\rangle$ such that executing this sequence upon the markings leads to a complete workflow from the starting place to the ending place i.e. $m_{n-1} \xrightarrow{t_n} m_n$. Moreover, the empty sequence $\langle\rangle$ always remains enabled (i.e, $m \xrightarrow{\langle\rangle} m$).

$\mathcal{L}(N) := \{\sigma \in T^* | m : P \to \ N_0, m_0 \xrightarrow{\sigma} m\}$ is defined as the prefix-closed language of N describing all the prefixes of its possible traces, as shown in Table 1.

Prefixes
$\langle a \rangle$
$\langle a, b \rangle$
$\langle a, b, c \rangle$
$\langle a, b, c, d \rangle$

Table 1: $\mathcal{L}(N)$ as the prefix-closed language of N

Implicit Place Implicit places are places possessing the distinctive property that their inclusion to or elimination from a Petri net does not change the executing behaviour that is allowed by the Petri net. An implicit place can also be referred to as a redundant place [7]. Considering Petri net N in Figure 1, it is observed that its language L consists of a single unique trace. With the removal of place p_1 from N, it is still possible to execute this trace, hence p_1 is implicit.

Token Count Function The number of tokens in different places at any point in time during the execution of a sequence varies depending upon the conditions of the firing rule in Section 2.1. The token count function calculates the total number of tokens present in a place p after firing a sequence $\langle t_1, ..., t_n \rangle$ in a marked Petri net (P, T, F, m_0) by:

$$x_p(\langle t_1, ..., t_n \rangle) := m_0(p) + \sum_{i=1}^{n} (F(t_i, p) - F(p, t_i))$$

This function adds to the initial marking and also considers the empty trace.

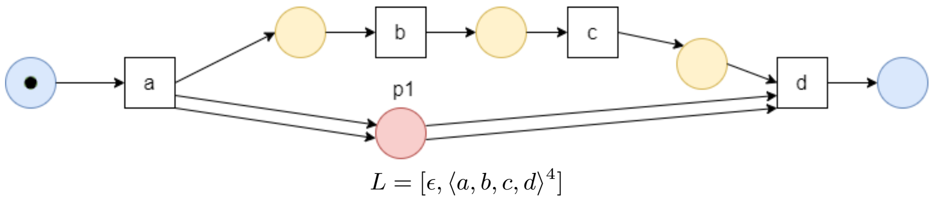

Fig. 1: Petri net N

3 Theory of Regions

A conventional group of process discovery algorithms is based on region theory. The basic idea of region theory is to identify *regions* in the event log or in a derived representation of that log. This derived representation can, for example, be a *transition system* or a *prefix-closed language*. The input can also be a specification that is not derived from a log. Each region corresponds to a *feasible* place in the resulting Petri net. Region theory always aims to construct a Petri net model with maximum fitness i.e, maximizing the ability to explain observed behaviour. This is contrary to other process mining algorithms such as the genetic miner [12], the fuzzy miner [8] and the heuristic miner [14] which primarily focus on model simplicity and human readability.

A region is a mathematical structure in the form of a multiset of states or language-prefixes satisfying certain conditions with respect to the event log. Each region can be transformed into a place connecting the transitions in such a way that the log can still be replayed on that place, and, thus the Petri net. The two main types of region theory are state-based region theory and language-based region theory elaborated in subsequent sections.

3.1 State-based region theory

In state-based region theory, the event log is transformed into a *transition system*, and then this transition system is mutated into a Petri net. Formally, a transition system (or TS) is defined as follows:

$TS = (S, E, T)$ defines a labelled transition system where S is the set of states, E is the set of transition labels i.e, $E = A \cup \tau$ where A is the set of activities recorded in the log and τ is the set of activities not recorded in the log, and lastly T is the transition relation such that $T \subseteq S \times E \times S$.

The transition system and Petri net have the ability to simulate each other i.e, they are bisimilar. This confirms the notion that they are are behaviourally equivalent and if the transition system exhibits concurrency, the Petri net may be much smaller than the transition system [13].

3.2 Language-based region theory

In language-based theory of regions, we consider a language over finite alphabets. From this language, it is possible to develop a Petri net with minimal behaviour such that the vocabulary defined by the language contains possible firing sequences in the language of the Petri net. The Petri net is the active part whose allowed words should be in the language. The concept of a region is then defined as one of the feasible solutions of the linear inequation system, constructed for a given language.

Please note that the event log needs to be represented as a prefix-closed language as observed in Table 1. In the case of language-based regions, finding a solution for each linear inequation in the system is of exponential worst-case time complexity [3].

3.3 Minimal Regions

In region theory, there are different approaches for finding the subset of regions that correspond to the desired places. For example, minimal regions, wrong continuations and base representation constitute among such approaches [10]. Minimal regions lead to places with maximum expressiveness and avoid adding implicit places to a model [5]. For defining minimal regions, r and r' are considered as regions inside a transition system. r' is a sub-region of r if r' is a subset of r. r is a minimal region if and only if there is no other region r' which is a sub-region of r. For each minimal region r, a place p is produced.

For the purposes of the proposed technique, we do not aim to take the detour via region theory, and instead we aim to transition directly from the event log to the set of *best* places and, thus, the resulting model.

Basically, we have a process discovery algorithm which provides us with a set of places based on the event log. From this set of places, we wish to keep the feasible ones and remove all the implicit places. However, since computing the regions is skipped, the existing filtering techniques cannot be applied directly. Hence, we present an approach to identify and remove implicit places in the discovered set of places. This approach is inspired by minimal regions and hence decreases the number of implicit places in a model.

4 Removing Implicit Places

In this section, the main idea of the proposed algorithm is presented along with its running example.

4.1 Assumptions

The Petri nets under consideration are marked Petri nets that do not allow for self-loops and dead transitions but possibly allow for arc weights (or multiple arcs). The language of the Petri net is equal as that of the prefix-closure defined by the event log.

4.2 Main Idea

Given an event log and the corresponding Petri net, the idea behind the proposed algorithm is to compare a pair of places by replaying the event log on them while comparing the number of tokens they contain via the token count function. Considering places p_1 and p_2 from the set P of places for a generic Petri net $N = (P, T, F, m_0)$, the following conditions are ascertained:

Places p_1 and p_2 are compared and it is observed that for all sequences in the language of the Petri net, the token count of p_1 after executing a sequence is always greater than or equal to the token count of p_2 after executing the same sequence. Mathematically, this is described as follows:

$$\forall \sigma \in L(N) : x_{p_1}(\sigma) \geq x_{p_2}(\sigma) \tag{1}$$

There exists at least one sequence in this language where the token count of p_1 is strictly larger than the token count of p_2. Mathematically, this is explained as follows:

$$\exists \sigma \in L(N) : x_{p_1}(\sigma) > x_{p_2}(\sigma) \tag{2}$$

With the fulfillment of conditions (1) and (2), a novel and feasible place p_3 can be computed with the criteria that its token count is always the difference between the token counts of places p_1 and p_2 respectively.

$$x_{p_3} = x_{p_1} - x_{p_2} \tag{3}$$

With the existence of places p_2 and p_3 in the Petri net, it is safe to regard p_1 as implicit and remove it from the Petri net.

The theory behind this is supported by Proposition 2 in [10]. We again consider Petri net $N = (P, T, F, m_0)$ without self-loops and a trace t which belongs to the prefix-closed language of N. For places p_1 and p_2 and their respective token count functions x_1 and x_2, if $x_{p_1} > x_{p_2}$ for the prefix-closed language of N, then $(x_{p_1} - x_{p_2})$ is a region. Successively, we define p_3 as the place of this region. Accordingly, we can replace p_1 by p_3 without changing the language of N.

4.3 Running Example

We consider the event log $L = [\epsilon, \langle a, c, e, g\rangle^2, \langle a, e, c, g\rangle^3, \langle b, d, f, g\rangle^2, \langle b, f, d, g\rangle^4]$ with the discovered Petri net N as shown in Figure 2. The Petri net N is able to generate the behaviour observed in L; however, it is unnecessarily complex. The input places p_5, p_3 and p_2 (marked in red) of transition g are implicit and can be removed from the net without allowing for more traces. The non-implicit places are marked in yellow and the start/end places are marked in blue.

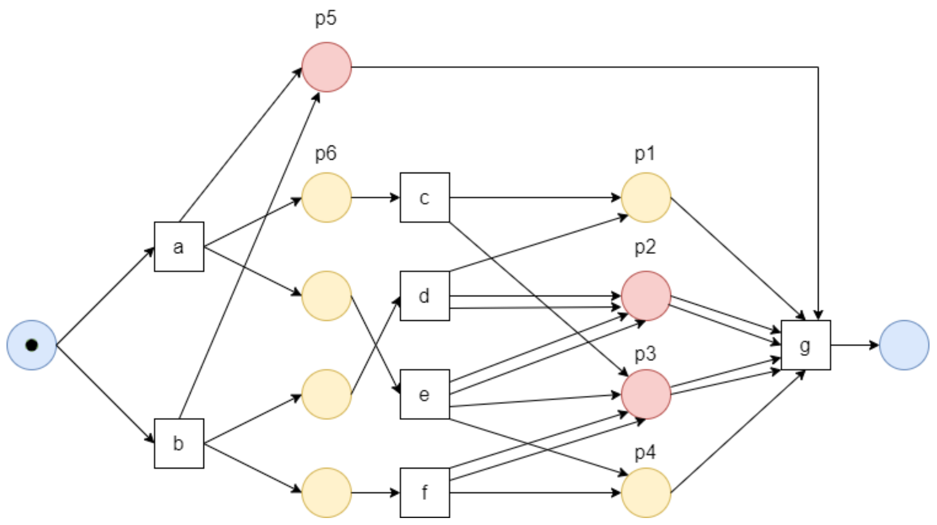

Fig. 2: Initial Petri net N derived from L

We will now proceed with applying the proposed implicit place removal technique in three different scenarios. Please note that we apply the implicit place removal routine to the entire prefix-closed log in all cases.

Fundamental Case We consider the places p_5 and p_6. Initially, we replay the trace $\langle a, c, e, g\rangle$ on the model. In the beginning of the replay, all places are empty and both p_5 and p_6 have no tokens. Then, we fire transition a and this produces 1 token each in places p_5 and p_6. Then, we fire transition c which results in 1 token in place p_5 and 0 tokens in place p_6. Firing transition e subsequently does not change the token count of p_5 and p_6. Finally, after firing transition g, all tokens are consumed and all places are empty at the end of replay. Next, the trace $\langle a, e, c, g\rangle$ is replayed on the model followed by the trace $\langle b, d, f, g\rangle$. Lastly, the trace $\langle b, f, d, g\rangle$ is replayed on the model.

As observed in Table 2, the token counts are kept track of at each point during replay. If we compare those token counts, we can see that place p_5 has at least as many tokens as place p_6 at every point during replay. If the token counts for places p_5 and p_6 would have been exactly equal, then we would realize that both the places are exactly equal, hence one of them is implicit and it would be safe to remove it without adding a new place.

In our case, there are nine examples where the token count of place p_5 is strictly larger than that of place p_6. This means that if the difference between places p_5 and p_6 is computed, we get token counts that are non-negative (i.e, $x_{p_5} > x_{p_6}$). Based on this difference, a novel place p_7 can be constructed (as shown in Figure 3), such that the number of tokens in this new place always corresponds to the difference computed in Table 2. This also implies p_5 as an extraneous place.

	ε	**a**	**c**	**e**	**g**
x_{p_5}	0	1	1	1	0
x_{p_6}	0	1	0	0	0
x_{p_7}	0	0	1	1	0

	ε	**a**	**e**	**c**	**g**
x_{p_5}	0	1	1	1	0
x_{p_6}	0	1	1	0	0
x_{p_7}	0	0	0	1	0

	ε	**b**	**d**	**f**	**g**
x_{p_5}	0	1	1	1	0
x_{p_6}	0	0	0	0	0
x_{p_7}	0	1	1	1	0

	ε	**b**	**f**	**d**	**g**
x_{p_5}	0	1	1	1	0
x_{p_6}	0	0	0	0	0
x_{p_7}	0	1	1	1	0

Table 2: Calculation of novel place p_7 as the difference of places p_5 and p_6

For the construction of place p_7, we search where the token count of the new place is changing in Table 2 and add the corresponding arcs as shown in Figure 3. Considering the first trace $\langle a, c, e, g \rangle$ in Table 2, we witness that we have a token count changing from 0 to 1 from a to c for p_7. This means that firing transition c should add one token to place p_7, and we therefore add an incoming arc with weight 1 from transition c. The token count remains unchanged from c to e so no arcs are added. Lastly, we have a token count changing from 1 to 0 from e to g. This occurs when we fire transition g, thus, transition g must be an outgoing transition of the newly constructed place p_7. This method of observing token count differences and adding corresponding arcs is applied for all the traces in L.

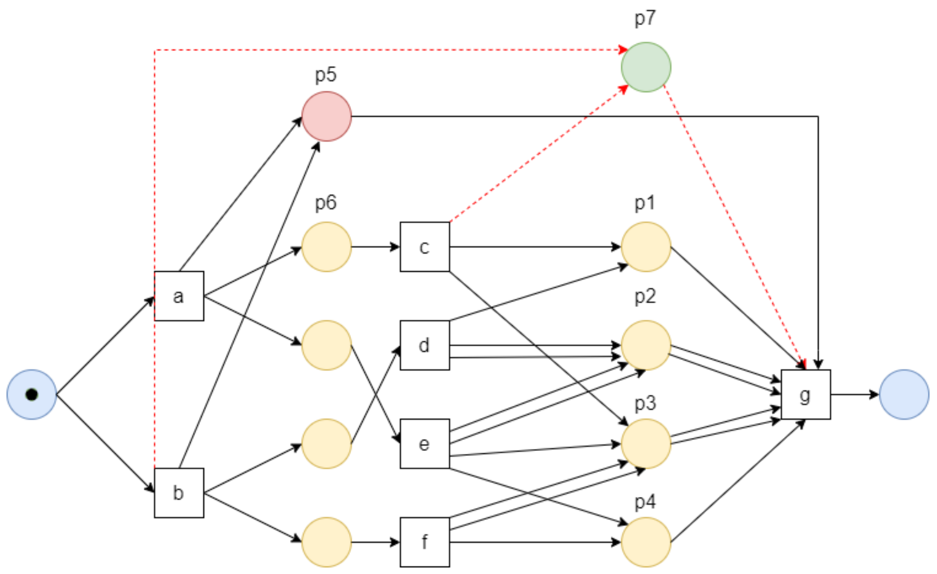

Fig. 3: Intermediate step of introducing novel place p_7

Place p_5 turns out to be an implicit place and can be removed from the Petri net while the newly introduced place p_7 is always feasible with respect to the model, hence it can be added in the final Petri net, as shown in Figure 4.

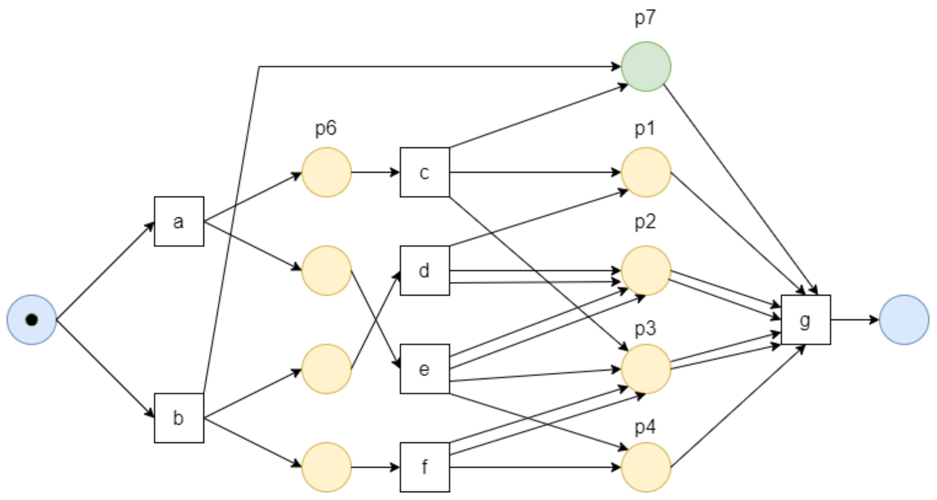

Fig. 4: Final Petri net N' obtained

Extended Case In the second case, multiple arcs from transitions to places are considered and places p_3 and p_4 are inspected as shown in Figure 2. As for the simple case in Section 4.3, the language L is replayed on the Petri net model. The progress of token counts for p_3 and p_4 while executing L is kept track of in Table 3. Here, place p_3 turns out to be an implicit place and, additionally, a novel place p_8 is constructed (as shown in Figure 5), such that the number of tokens in this new place always corresponds to the difference computed in Table 3.

	ε	**a**	**c**	**e**	**g**
x_{p_3}	0	0	1	2	0
x_{p_4}	0	0	0	1	0
x_{p_8}	0	0	1	1	0

	ε	**a**	**e**	**c**	**g**
x_{p_3}	0	0	1	2	0
x_{p_4}	0	0	1	1	0
x_{p_8}	0	0	0	1	0

	ε	**b**	**d**	**f**	**g**
x_{p_3}	0	0	0	2	0
x_{p_4}	0	0	0	1	0
x_{p_8}	0	0	0	1	0

	ε	**b**	**f**	**d**	**g**
x_{p_3}	0	0	2	2	0
x_{p_4}	0	0	1	1	0
x_{p_8}	0	0	1	1	0

Table 3: Calculation of novel place p_8 as the difference of places p_3 and p_4

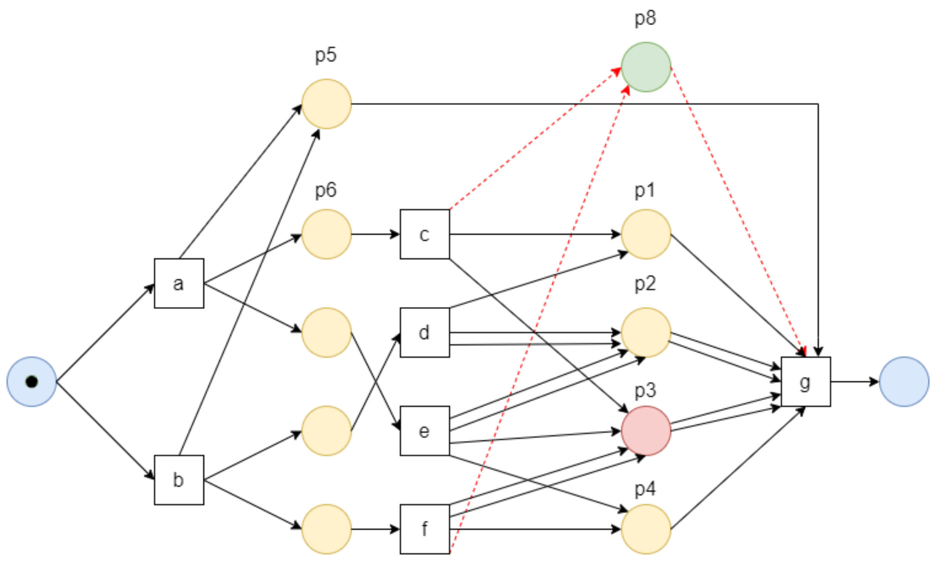

Fig. 5: Intermediate step of introducing novel place p_8

Place p_3 turns out to be an implicit place and can be removed from the Petri net while the newly introduced place p_8 is always feasible with respect to the model, hence it can be added in the final Petri net, as shown in Figure 6.

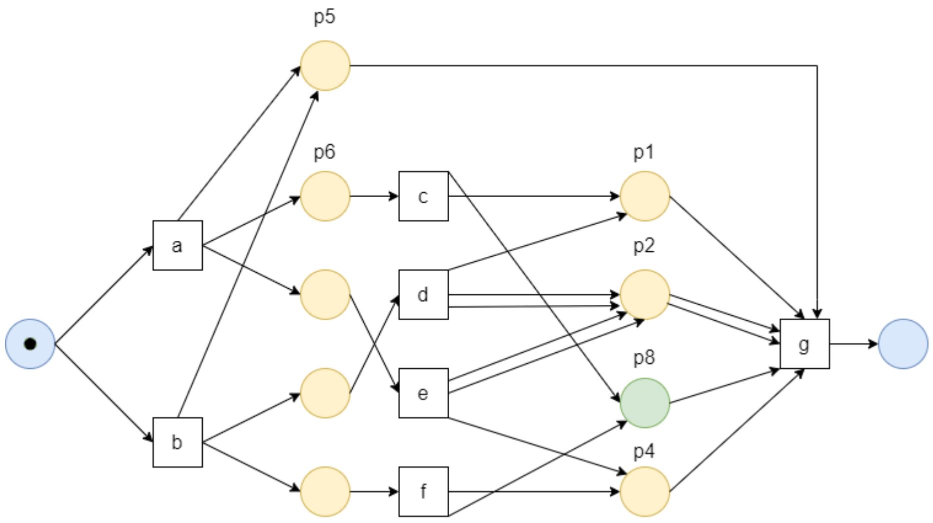

Fig. 6: Final Petri net N' obtained

Case of undiscoverable implicit place In the third case, places p_2 and p_1 are considered where, by construction, p_2 is an implicit place as shown in Figure 2. Interestingly, p_2 does not turn out to be an implicit place according to the proposed technique. The reason being that there exists such a sequence where the token count of place p_2 is not strictly larger than that of place p_1. This means that the difference of token counts between places p_2 and p_1 is negative for some instance. The mentioned phenomenon can be observed for trace $\langle a, c, e, g\rangle$ (marked in red) in Table 4. This case can be regarded as a limitation of the proposed technique since $x_{p_2} > x_{p_1}$ does not hold for some instance of the model replay. This means that a minimal region is not formed from which a feasible place p_9 can be constructed.

	ε	**a**	**c**	**e**	**g**
x_{p_2}	0	0	0	2	0
x_{p_1}	0	0	1	1	0
x_{p_9}	0	0	-1	1	0

	ε	**a**	**e**	**c**	**g**
x_{p_2}	0	0	2	2	0
x_{p_1}	0	0	0	1	0
x_{p_9}	0	0	2	1	0

	ε	**b**	**d**	**f**	**g**
x_{p_2}	0	0	2	2	0
x_{p_1}	0	0	1	1	0
x_{p_9}	0	0	1	1	0

	ε	**b**	**f**	**d**	**g**
x_{p_2}	0	0	0	2	0
x_{p_1}	0	0	0	1	0
x_{p_9}	0	0	0	1	0

Table 4: Calculation of novel place p_9 as the difference of places p_2 and p_1 is not possible

4.4 Correctness and Completion

Process mining is always based on some notion of completeness [2]. This means that we can never realistically assume to have witnessed all possibilities in a given event log. In reality, there exists a remarkable difference between the set of possible traces found in the log and the set of possible traces in the Petri net. For example, there might be additional trace(s) possible in the net but not occurring in the log.

With respect to the correctness and completion aspects, the proposed technique guarantees to never delete non-implicit places. For completeness, nearly all implicit places are identified, except for the special case of places with self-loops within parallel constructs, as discussed in Section 5.3. Both of these propositions hold only if the event log is complete with respect to the language defined by the Petri net.

5 Application to the eST-Miner

In this section, we explore the combination of the proposed technique with the eST-Miner process discovery algorithm.

5.1 Motivation

The eST-Miner process discovery algorithm takes inspiration from language-based region theory and possesses the ability to find complex process structures, improve model simplicity, and handle infrequently occurring model behaviour [11]. The algorithm takes an event log as input and returns a set of places connecting the various transitions. The underlying concept is to traverse all candidate places and evaluate them using token replay, meanwhile omitting uninteresting segments of the search space. This algorithm considers the notion of *fitting* places where every place that can replay the event log on it and is empty (i.e, without tokens) at the beginning and end of replay is considered fitting and can be added to the final Petri net. The problem arises when the returned Petri net does not only contain desirable places but also a significant number of redundant places. Here, the proposed technique proves to be useful. These implicit places can be removed in the post-processing step of the eST-Miner algorithm.

5.2 Removing Implicit Places in eST-Miner

The proposed approach benefits from some of the unique features of the eST-Miner. Since the eST-Miner only keeps the event log, the final result and the place being currently evaluated in memory, it is quite space efficient compared to most other discovery algorithms mentioned in Section 3. The same holds for the implicit place removal strategy under discussion since it also only keeps in memory the event log, the final result, and the two places being compared currently. Hence, combining these two aspects leads to an overall space-efficient algorithm.

The eST-Miner guarantees the discovery of a set of all fitting places [11]. This means that if there are two places p_1 and p_2 under comparison and we need to introduce a place p_3 to prove that p_1 is implicit, then we do not have to check whether p_3 actually exists in the Petri net. The existence of p_3 is already guaranteed by the eST-Miner.

Moreover, the fact that all fitting places are discovered can be used to skip unnecessary place comparisons. For example, if there is a place p_1 and we identify it to be implicit due to places p_2 and p_3, then either place p_2 or p_3 is sufficient to identify place p_1 being implicit. This is possible since it is not needed to check the existence of places as mentioned earlier. Additionally, we know that either place p_2 or p_3 has a shared in-going transition with place p_1. So when comparing places, we can focus on places that share in-going transitions and this results in significantly fewer comparisons. Therefore, the returned Petri net defines the minimal behaviour containing the behaviour defined by the event log. This makes the language of the Petri net complete for the entire prefix closure defined by the log, which makes our theoretical results applicable.

Finally, since the eST-Miner involves the traversal of all candidate places and evaluating them one after the other, two variants of the place comparison strategy are implemented as follows:

Final Place Removal (FPR) The FPR variant is implemented in [10] where the final set of places is computed and then we compare and subsequently remove implicit places.

Concurrent Place Removal (CPR) Additionally, the CPR variant is implemented in [10] where every discovered place is directly compared to existing places and removed if found to be implicit.

5.3 Challenges

The integration of our technique with the eST-Miner comes with its set of challenges. The eST-Miner allows for Petri nets with self-loops which contradicts our theoretical assumptions. This problem arises during model replay. If the token count during replay is simply compared, self-loops might not be detected. As an example, places p_1 and p_2 are considered in the Petri net in Figure 7. Alongside, the token counts while executing trace $\langle b \rangle$ are kept track of in Table 5. Place p_1 has a self-loop on transition b. Both p_1 and p_2 have a token in the beginning and transition b is executed. Nothing happens in place p_2, so the token count there is 1. Similarly, the token in place p_1 is consumed and produced again so the token count remains 1. Hence, this proves that we are unable detect the token count difference.

Eventually, we do not aim to delete place p_1 because it is more restrictive and its self-loop might have some purpose in the Petri net. Hence, a method is devised to deal with self-loops as explained from Table 5 and Figure 7. The consumption of tokens is detached from the production of tokens during replay; so, whenever transition b is fired, tokens are consumed, leaving the place p_1 without a token while place p_2 has 1 token. This is observed in the x place of the x/y notation in Table 5. Then tokens are produced which is demonstrated in the y place of this notation. When the token counts of the places during replay are compared, there is one instance where the token count of place p_1 is smaller than that of place p_2, which prevents us from deleting place p_1 as implicit.

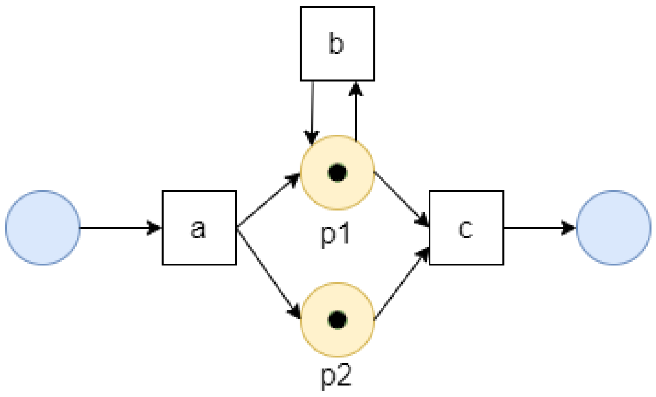

Fig. 7: Petri net with self-loops

		b
x_{p_1}	1	1
x_{p_2}	1	1

		b
x_{p_1}	1	0/1
x_{p_2}	1	1/1

Table 5: Token count on Petri net with self-loops

Finally, the special case of places with self-loops within parallel constructs using the Petri net is considered in Figure 8. As an example, the places p_1 and p_2 are considered. If we choose to execute transition b before executing transitions c and d, then the token count of place p_2 will be larger than that of place p_1, and therefore p_2 could possibly be removed. However, for traces that execute transitions c and d before executing transition b, the token count of place p_2 turns out to be smaller than that of p_1 for some transitions because p_2 consumes tokens due to self-loops. This phenomenon holds for every possible place within the parallel construct that we can compare place p_2 to, hence it cannot be identified as implicit.

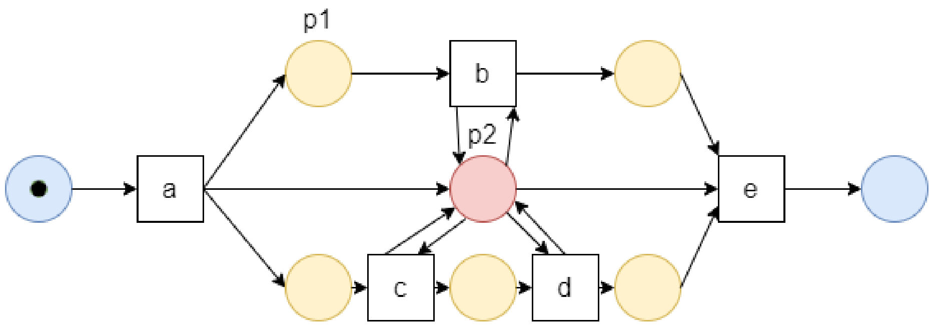

Fig. 8: Self-loops

Please note that the aspects above are not central to the main algorithm but rather rare in real-time experiments and do not impose a substantial problem in application.

6 Evaluation

In this section, we highlight the statistical results relevant to the evaluation characteristics of the proposed algorithm.

6.1 Statistical Results

To gain an understanding of the evaluation criteria of the algorithm under discussion, the statistical results comparing the CPR variant with the FPR variant in the ProM [6] implementation onto various event logs are reviewed. Please note that we calculate the respective logarithmic scales for better intuitive interpretation.

First, the impact of removing implicit places in Figure 9 is considered. A large difference in the number of places that would have been found without removing any implicit places in contrast to the remaining final places is observed. Please note that, both, the CPR and FPR, variants result in the same number of final and fitting places.

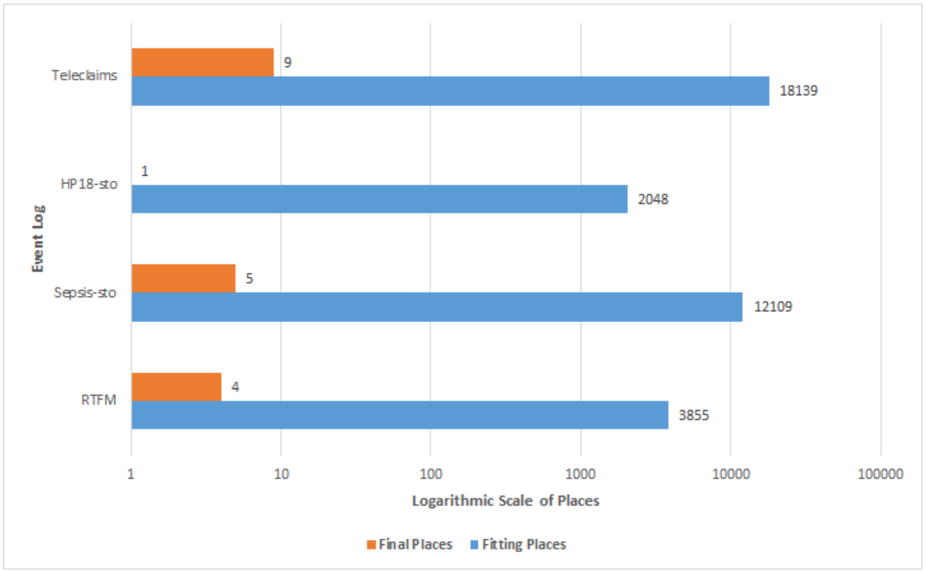

Fig. 9: Final Places vs. Fitting Places

Secondly, the runtime of FPR and CPR in Figure 10 are compared. It turns out that CPR is always significantly faster than FPR. The reason being that CPR performs significantly less comparisons, thus requiring substantially less computation time. The CPR variant removes every place that it can identify as implicit as soon as possible, and thus this place is never going to be used for later comparisons. Meanwhile, the FPR variant has all the possible places available for making comparisons and thus often compares a place to another implicit place.

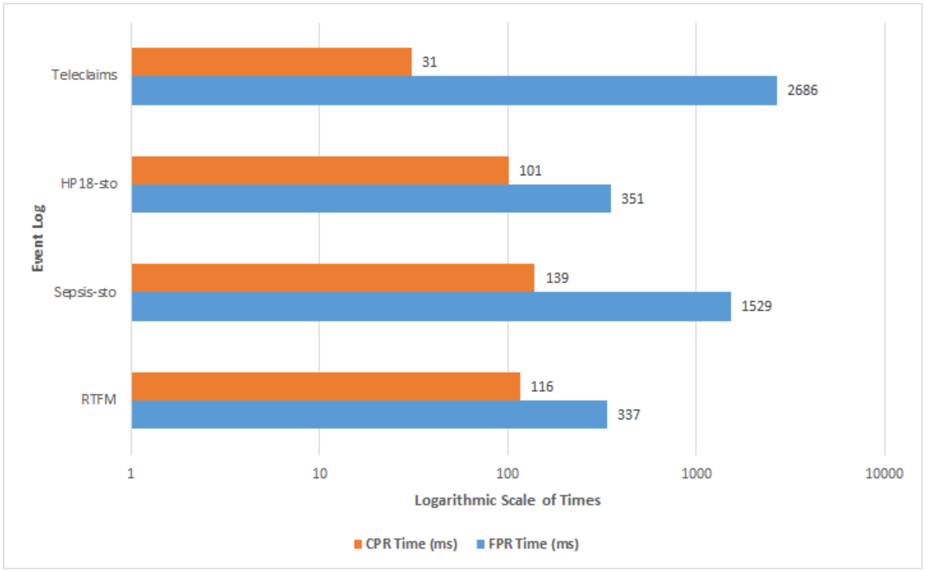

Fig. 10: CPR runtime vs. FPR runtime

7 Related Work

In this section, previous literature in accordance with the proposed technique is discussed. The related work considers the identification of implicit places as an Integer Linear Programming (ILP) problem, such as in [7].

7.1 Counting Petri net markings from reduction equations

The work in [4] discusses an approach for characterizing the state space of a Petri net with the help of systems of linear equations. This method analyzes implicitness properties and identifies implicit places based on mathematical constraints and definitions. It focuses on developing a reduced Petri net architecture with three categories of reduction rules which form the Petri net reduction system. Removal of implicit places is one of these categories.

The linear inequation system identifies implicit places from the Petri net allowing us to compute the number of reachable markings of the original Petri net from the reduced Petri net and the reduction history. The method is particularly useful for highlighting the state space of a Petri net. Moreover, it is appropriate for checking whether a given reachable marking satisfies a given set of linear constraints.

The work in [4] has been suggested as part of the post-processing step of the eST-Miner in existing literature [11]. After the evaluation of potentially fitting candidate places with respect to the given log, several places can be removed without changing the behaviour defined by the net since such extraneous places are undesirable. As part of the post-processing step, researchers suggest to identify and remove implicit places from the discovered set of fitting places by solving the ILP problem. The resulting minimal set of places is finally inserted into the Petri net that forms the output of the eST-Miner.

8 Conclusion

In this paper, we outlined an approach to identify and remove implicit places in marked Petri nets. This approach takes into account the information contained in the event log and is inspired from minimal regions in region theory. We presented the sequential and concurrent application schemes of the algorithm in combination with the eST-Miner process discovery algorithm. In particular, the CPR variant minimizes the returned set of places by immediately discarding implicit places. Finally, we realized that the technique proves to be quite time and space efficient.

8.1 Future Work

For future work, the combination of the proposed technique with various aspects of the eST-Miner could be investigated further.

Firstly, the noise handling threshold of the eST-Miner could be adapted for the log-based place comparison steps. Secondly, the efficiency of the candidate traversal of the eST-Miner could be increased by omitting sets of uninteresting places that are known to be unfitting. The CPR variant might allow us to use the produced results to skip additional candidates and thus increase efficiency. Lastly, in combination with the CPR variant, the eST-Miner could be adjusted to return intermediate results or return results after a certain running time.

The order of place comparisons for calculating the token count differences has an impact on the number of comparisons being performed and, thus, on the overall runtime. Therefore, an idea is to develop strategies on how to choose the order of comparisons which can potentially speed up the algorithm.

Devising a viable solution to the problem of self-loop places in parallel constructs in Section 5.3 would make the proposed approach more polished from a theoretical perspective.

Finally, we could consider the application of the technique to other process discovery algorithms. A viable example would be the Inductive-Miner algorithm since it guarantees sound models [9]. The important point of investigation would be the relation between the event log and the Petri net. In the special cases where the event log differs significantly from the behaviour defined by the Petri net, the fact that places can be added might be used to reduce this difference and, thus, enhance the language defined by the Petri net.

References

1. W. Aalst and A. Ter. Verification of workflow task structures: A petri-net-baset approach. *Information Systems*, 25:43–69, 03 2000.
2. W. V. Aalst, V. Rubin, B. V. Dongen, E. Kindler, and C. Günther. Process mining : A two-step approach using transition systems and regions. 2006.
3. R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process mining based on regions of languages. In G. Alonso, P. Dadam, and M. Rosemann, editors, *Business Process Management*, pages 375–383, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
4. B. Berthomieu, D. Botlan, and S. Dal Zilio. Counting petri net markings from reduction equations. *International Journal on Software Tools for Technology Transfer*, 22, 04 2020.
5. J. Desel and W. Reisig. The synthesis problem of petri nets. In P. Enjalbert, A. Finkel, and K. W. Wagner, editors, *STACS 93*, pages 120–129, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.
6. B. Dongen, A. Medeiros, H. Verbeek, A. Weijters, and W. Aalst. The prom framework: A new era in process mining tool support. volume 3536, pages 444–454, 06 2005.
7. F. Garcia-Valles and J. M. Colom. Implicit places in net systems. In *Proceedings of the The 8th International Workshop on Petri Nets and Performance Models*, PNPM '99, page 104, USA, 1999. IEEE Computer Society.
8. C. W. Günther and W. M. P. van der Aalst. Fuzzy mining – adaptive process simplification based on multi-perspective metrics. In G. Alonso, P. Dadam, and M. Rosemann, editors, *Business Process Management*, pages 328–343, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
9. S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst. Discovering block-structured process models from event logs - a constructive approach. In J.-M. Colom and J. Desel, editors, *Application and Theory of Petri Nets and Concurrency*, pages 311–329, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
10. L. Mannel. *Removing Implicit Places Using Regions for Process Discovery.* 06 2020.
11. L. L. Mannel and W. M. P. van der Aalst. Finding complex process-structures by exploiting the token-game. In S. Donatelli and S. Haar, editors, *Application and Theory of Petri Nets and Concurrency*, pages 258–278, Cham, 2019. Springer International Publishing.
12. A. Medeiros, A. Weijters, and W. Aalst. Genetic process mining: An experimental evaluation. *Data Mining and Knowledge Discovery*, 14:245–304, 04 2007.
13. W. M. P. van der Aalst and B. F. van Dongen. Discovering petri nets from event logs. In K. Jensen, W. M. P. van der Aalst, G. Balbo, M. Koutny, and K. Wolf, editors, *Transactions on Petri Nets and Other Models of Concurrency VII*, pages 372–422, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
14. A. Weijters and W. Aalst. Rediscovering workflow models from event-based data using little thumb. *Integrated Computer-Aided Engineering*, 10:151–162, 07 2003.