Finding Uniwired Petri Nets Using eST-Miner

Lisa L. Mannel(✉) and Wil M. P. van der Aalst

Process and Data Science (PADS), RWTH Aachen University, Aachen, Germany,
mannel@pads.rwth-aachen.de; wvdaalst@pads.rwth-aachen.de

Abstract. In process discovery, the goal is to find, for a given event log, the model describing the underlying process. While process models can be represented in a variety of ways, in this paper we focus on a subclass of Petri nets. In particular, we describe a new class of Petri nets called *Uniwired Petri Nets* and first results on their expressiveness. They provide a balance between simple and readable process models on the one hand, and the ability to model complex dependencies on the other hand. We then present an adaptation of our eST-Miner aiming to find such Petri Nets efficiently. Constraining ourselves to uniwired Petri nets allows for a massive decrease in computation time compared to the original algorithm, while still discovering complex control-flow structures such as long-term-dependencies. Finally, we evaluate and illustrate the performance of our approach by various experiments.

Keywords: Process Discovery, Petri Nets, Language-based Regions

1 Introduction

More and more processes executed in companies are supported by information systems which store each event executed in the context of a so-called *event log*. Each event in such an event log has a name identifying the executed activity (activity name), identification specifying the respective execution instance of the process (case id), a time when the event was observed (time stamp), and often other data related to the activity and/or process instance. In the context of process mining, many algorithms and software tools have been developed to utilize the data contained in event logs: in *conformance checking*, the goal is to determine whether the behaviors given by a process model and event log comply. In *process enhancement*, existing models are improved. Finally, in *process discovery*, a process model is constructed aiming to reflect the behavior defined by the given event log: the observed events are put into relation to each other, preconditions, choices, concurrency, etc. are discovered, and brought together in a process model.

Process discovery is non-trivial for a variety of reasons. The behavior recorded in an event log cannot be assumed to be complete, since behavior allowed by the process specification might simply not have happened yet. Additionally, real-life event logs often contain noise, and finding a balance between filtering this out and at the same time keeping all desired information is often a non-trivial task. Ideally, a discovered model should be able to produce the behavior contained within the event log, not allow for unobserved behavior, represent all dependencies between the events and at the same time be simple enough to be understood by a human interpreter. It is rarely possible to fulfill all these requirements simultaneously. Based on the capabilities and focus of the used algorithm, the discovered models can vary greatly, and different trade-offs are possible.

To decrease computation time and the complexity of the returned process model, many existing discovery algorithms abstract from the full information given in a log or resort to heuristic approaches. Even though the resulting model often cannot fully represent the language defined by the log, they can be very valuable in practical applications. Examples are the Alpha Miner variants ([1]), the Inductive Mining family ([2]), genetic algorithms or Heuristic Miner. On the downside, due to the commonly used abstractions, these miners are not able to (reliably) discover complex model structures, most prominently non-free choice constructs. Algorithms based on region theory ([3–6]) discover models whose behavior is the minimal behavior representing the log. However, they often lead to complex, over-fitting process models that are hard to understand. These approaches are also known to be rather time-consuming and expose severe issues with respect to low-frequent behavior often contained in real-life event logs.

In our previous work [7] we introduce the discovery algorithm eST-Miner, which focuses on mining process models formally represented by Petri nets. This approach aims to combine the capability of finding complex control-flow structures like longterm-dependencies with an inherent ability to handle low-frequent behavior while exploiting the token-game to increase efficiency. Similar to region-based algorithms, the basic idea is to evaluate all possible places to discover a set of fitting ones. Efficiency is significantly increased by skipping uninteresting sections of the search space. This may decrease computation time immensely compared to the brute-force approach evaluating every single candidate place, while still providing guarantees with regard to fitness and precision.

Though inspired by language-based regions, the approach displays a fundamental difference with respect to the evaluation of candidate places: while region-theory traditionally focuses on a globalistic perspective on finding a set of feasible places, our algorithm evaluates each place separately, that is from a local perspective. In contrast to the poor noise-handling abilities of traditional region-based discovery algorithms, this allows us to effectively filter infrequent behavior place-wise. Additionally, we are able to easily enforce all kinds of constraints definable on the place level, e.g. constraints on the number or type of arcs, token throughput or similar.

However, the original eST-Miner has several drawbacks that we aim to tackle in this paper: first, the set of fitting places takes too long to be computed, despite our significant improvement over the brute force approach. Second, the discovered set of places typically contains a huge number of implicit places, that need to be removed in a time-consuming post-processing step. Third, the algorithm finds very complex process structures, which increase precision but at the same time decrease simplicity. In this work, we introduce the new class of *uniwired Petri nets*, which constitutes a well-balanced trade-off between the ability of modeling complex control-flows, such as long-term-dependencies, with an inherent simplicity that allows for human readability and efficient computation. We present a corresponding variant of our eST-Miner, that aims to discover such Petri nets.

In Sec. 2 we provide basic notation and definitions. Afterwards a brief introduction to our original algorithm is given in Sec. 3. We then present the class of *uniwired Petri* *nets* in Sec. 4, together with first results on their expressiveness and relevance for process mining. In Sec. 5 we describe an adaption of our algorithm that aims to efficiently compute models of this sub-class by massively increasing the amount of skipped candidate places. An extensive evaluation follows in Sec. 6. Finally, we conclude the paper with a summary and suggestion of future work in Sec. 7.

2 Basic Notations, Event Logs, and Process Models

A set, e.g. $\{a, b, c\}$, does not contain any element more than once, while a multiset, e.g. $[a, a, b, a] = [a^3, b]$, may contain multiples of the same element. By $\mathbb{P}(X)$ we refer to the power set of the set X, and $\mathbb{M}(X)$ is the set of all multisets over this set. In contrast to sets and multisets, where the order of elements is irrelevant, in sequences the elements are given in a certain order, e.g. $\langle a, b, a, b\rangle \neq \langle a, a, b, b\rangle$. We refer to the i'th element of a sequence σ by $\sigma(i)$. The size of a set, multiset or sequence X, that is $|X|$, is defined to be the number of elements in X. We define activities, traces, and logs as usual, except that we require each trace to begin with a designated start activity (▶) and end with a designated end activity (■). Note, that this is a reasonable assumption in the context of processes, and that any log can easily be transformed accordingly.

Definition 1 (Activity, Trace, Log). *Let $\mathcal{A}$ be the universe of all possible* activities *(e.g., actions or operations), let ▶ $\in \mathcal{A}$ be a designated start activity and let ■ $\in \mathcal{A}$ be a designated end activity. A* trace *is a sequence containing ▶ as the first element, ■ as the last element and in-between elements of $\mathcal{A} \setminus \{$▶, ■$\}$. Let $\mathcal{T}$ be the set of all such traces. A* log *$L \subseteq \mathbb{M}(\mathcal{T})$ is a multiset of traces.*

In this paper, we use an alternative definition for Petri nets. We only allow for places connecting activities that are initially empty (without tokens), because we allow only for traces starting with ▶ and ending with ■. These places are uniquely identified by the set of input activities I and output activities O. Each activity corresponds to exactly one transition, therefore this paper we refer to transitions as activities.

Definition 2 (Petri nets). *A Petri net is a pair $N = (A, \mathcal{P})$, where $A \subseteq \mathcal{A}$ is the set of activities including start and end ($\{$▶, ■$\} \subseteq A$) and $\mathcal{P} \subseteq \{(I|O) \mid I \subseteq A \wedge I \neq \emptyset \wedge O \subseteq A \wedge O \neq \emptyset\}$ is the set of places. We call I the set of* ingoing *activities of a place and O the set of* outgoing *activities.*

Given an activity $a \in A$, $\bullet a = \{(I|O) \in \mathcal{P} \mid a \in O\}$ and $a\bullet = \{(I|O) \in \mathcal{P} \mid a \in I\}$ denote the sets of input and output places. Given a place $p = (I|O) \in \mathcal{P}$, $\bullet p = I$ and $p\bullet = O$ denote the sets of input and output activities.

Definition 3 (Fitting/Unfitting Places). *Let $N = (A, \mathcal{P})$ be a Petri net, let $p = (I|O) \in \mathcal{P}$ be a place, and let $\sigma \in \mathcal{T}$ be a trace. With respect to the given trace σ, p is called*

- unfitting*, denoted by $\boxtimes_\sigma(p)$, if and only if at least one of the following holds:*
  - $\exists k \in \{1, 2, ..., |\sigma|\}$ *such that* $|\{i \mid i \in \{1, 2, ...k-1\} \wedge \sigma(i) \in I\}| < |\{i \mid i \in \{1, 2, ...k\} \wedge \sigma(i) \in O\}|$
  - $|\{i \mid i \in \{1, 2, ...|\sigma|\} \wedge \sigma(i) \in I\}| > |\{i \mid i \in \{1, 2, ...|\sigma|\} \wedge \sigma(i) \in O\}|$,
- fitting*, denoted by $\square_\sigma(p)$, if and only if not $\boxtimes_\sigma(p)$.*

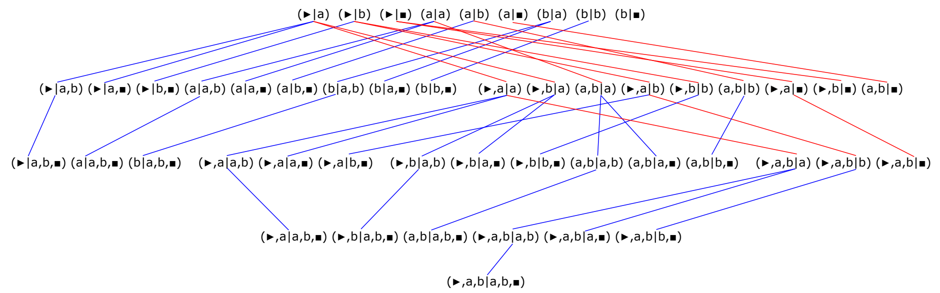
(▶|a) (▶|b) (▶|■) (a|a) (a|b) (a|■) (b|a) (b|b) (b|■)
(▶|a,b) (▶|a,■) (▶|b,■) (a|a,b) (a|a,■) (a|b,■) (b|a,b) (b|a,■) (b|b,■) (▶,a|a) (▶,b|a) (a,b|a) (▶,a|b) (▶,b|b) (a,b|b) (▶,a|■) (▶,b|■) (a,b|■)
(▶|a,b,■) (a|a,b,■) (b|a,b,■) (▶,a|a,b) (▶,a|a,■) (▶,a|b,■) (▶,b|a,b) (▶,b|a,■) (▶,b|b,■) (a,b|a,b) (a,b|a,■) (a,b|b,■) (▶,a,b|a) (▶,a,b|b) (▶,a,b|■)
(▶,a|a,b,■) (▶,b|a,b,■) (a,b|a,b,■) (▶,a,b|a,b) (▶,a,b|a,■) (▶,a,b|b,■)
(▶,a,b|a,b,■)

Fig. 1. Example of a tree-structured candidate space for activities $\{\blacktriangleright, A, B, \blacksquare\}$.

Definition 4 (Behavior of a Petri net). *We define the* behavior *of the Petri net* $(A, \mathcal{P})$ *to be the set of all fitting traces, that is* $\{\sigma \in \mathcal{T} \mid \forall p \in \mathcal{P} \colon \square_{\sigma}(p)\}$.

Note that we only allow for behaviors of the form $\langle \blacktriangleright, a_1, a_2, \dots a_n, \blacksquare \rangle$ such that places are empty at the end of the trace and never have a negative number of tokens.

3 Introducing the Algorithm

We briefly repeat our discovery approach as presented in [7]. For details we refer the reader to the original paper. As input, the algorithm takes a log L and a parameter $\tau \in [0, 1]$, and returns a Petri net as output. A place is considered fitting, if at the fraction τ of traces in the event log is fitting. A place is perfectly fitting when all traces are fitting. Inspired by language-based regions, the basic strategy of the approach is to begin with a Petri net, whose transitions correspond exactly to the activities used in the given log. From the finite set of unmarked, intermediate places a subset of places is inserted, such that the language defined by the resulting net defines the minimal language containing the input language, while, for human readability, using only a minimal number of places to do so.

The algorithm uses token-based replay to discover all fitting places $\mathcal{P}_{\text{fit}}$ out of the set of possible candidate places. To avoid replaying the log on the exponential number of candidates, it organizes the potential places as a set of trees, such that certain properties hold. When traversing the trees using a depth-first-strategy, these properties allow to cut off subtrees, and thus candidates, based on the replay result of their parent. This greatly increases efficiency, while still guaranteeing that all fitting places are found. An example of such a tree-structured candidate space is shown in Figure 1. Note the incremental structure of the trees, i.e. the increase in distance from the base roots corresponds to the increase of input and output activities. To significantly increase readability, implicit places are removed in a post-processing step.

The running time of the original eST-Miner as summarized in this section, strongly depends on the number of candidate places skipped during the search for fitting places. The approach uses monotonicity results [8] to skip sets of places that are known to be unfitting. For example, if 80% of the cases cannot be replayed because the place is empty and does not enable the next activity in the trace, then at least 80% will not allow for a place with even more output transitions. While this results in a significant decrease of computation time compared to the brute force approach, there are still to many candidates to be evaluated by replaying the log. Moreover, typically, the set of all fitting places contains a great number of implicit places, resulting in very slow post-processing. In the following we explore uniwired Petri nets as a solution to these issues.

4 Introducing Uniwired Petri Nets

In this paper, we aim to discover *uniwired* Petri nets. In uniwired Petri nets all pairs of activities are connected by at most one place. *Biwired* nets are all other Petri nets where at least one pair of activities is connected by at least two places.

Definition 5 (Biwired/Uniwired Petri Nets). *Let $N = (A, \mathcal{P})$ be a Petri net. N is* biwired *if there is a pair of activities $a_1, a_2 \in A$, such that $|a_1 \bullet \cap \bullet a_2| \geq 2$. N is* uniwired *if such a pair does not exist. $wired(\mathcal{P}) = \{(a_1, a_2) \mid \exists_{(I|O) \in \mathcal{P}}\, a_1 \in I \wedge a_2 \in O\}$ is the set of wired activities.*

As far as we can tell, the subclass of uniwired Petri nets has not been investigated systematically (like e.g., free-choice nets). However, the class of uniwired nets includes the class of Structured Workflow Nets (SWF-nets) known in the context of process mining. For example, the α-algorithm was shown to be able to rediscover this class of nets [9]. Since we are using an alternative Petri net representation (Definition 2), we define SWF-nets as follows.

Definition 6 (SWF-Nets (Structured Workflow Nets) [9]). *A Petri net $N = (A, \mathcal{P})$, is an* SWF-net*, if the following requirements hold:*

- *N has no implicit places (i.e., removing any of the places changes the behavior of the Petri net),*
- *for any $p \in \mathcal{P}$ and $a \in A$ such that $p \in \bullet a$: (1) $|p\bullet| > 1 \implies |\bullet a| = 1$ (i.e., choice and synchronization are separated), and (2) $|\bullet a| > 1 \implies |\bullet p| = 1$ (i.e., only synchronize a fixed set of activities).*

Lemma 1 (Uniwired Petri nets and SWF-nets). *The class of uniwired Petri nets is a strict superset of the class of SWF-nets.*

Proof. Let $N = (A, \mathcal{P})$ be an SWF-net. We show that the assumption that N is biwired leads to a contradiction. If N is biwired, then there are two activities $a_1, a_2 \in A$ and two different places $p_1, p_2 \in \mathcal{P}$ such that $\{p_1, p_2\} \subseteq a_1\bullet$ and $\{p_1, p_2\} \subseteq \bullet a_2$. Since $p_1 \neq p_2$, Definition 6 implies $|\bullet p_1| = |\bullet p_2| = 1$. Hence, $\bullet p_1 = \bullet p_2 = \{a_1\}$. If $a' \in p_1\bullet$ and $a' \neq a_2$, then $|p_1\bullet| > 1$. Hence, Definition 6 implies $|\bullet a_2| = 1$ leading to a contradiction (there are two places). The same applies to p_2. Hence, $p_1\bullet = p_2\bullet = \{a_2\}$. Combining $\bullet p_1 = \bullet p_2 = \{a_1\}$ and $p_1\bullet = p_2\bullet = \{a_2\}$, implies that p_1 and p_2 must have exactly the same connections, making one of these places implicit. Since there are no implicit places in SWF-nets, we conclude that an SWF-net cannot be biwired (i.e., is uniwired).

Fig. 2 shows that the class of uniwired Petri nets is a strict superset of the class of SWF-nets. The uniwired Petri net N_1 is not an SWF-net. For example the place $p = (\{e\}|\{f, g\})$ and activity g clearly violate the definition of SWF-nets. □

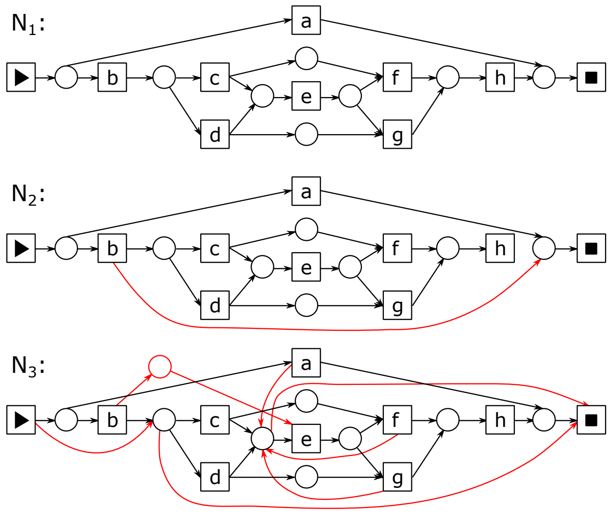

Fig. 2. N_1 shows a uniwired Petri net with long-term dependencies, that is not an SWF-net. The models N_2 and N_3 illustrate the problems resulting from conflicting places of varying (N_3) and similar (N_2) complexity, detailed in Sec. 5, before adding self-loops.

The α-algorithm is able to rediscover SWF-nets, but has well-known limitations with respect to discovering complex control-flow structures, for example long-term dependencies [1]. The class of uniwired Petri nets is clearly more expressive than SWF-nets, as illustrated for example by N_1 in Fig. 2, showing its capability of modeling advanced control-flow structures.

The models included in this class seem to constitute a well-balanced trade-off between human-readable, simple process structures on the one hand, and complex, hard-to-find control-flow structures on the other hand. This makes them a very interesting class of models in the context of process discovery. In Sec. 5, we will show that such models naturally lend themselves to be discovered efficiently by an adaption of our eST-Miner.

5 Discovering Uniwired Petri Nets

The original eST-Miner discovers all non-implicit places that meet a preset quality threshold, and is capable of finding even the most complex control-flow structures. However, the resulting Petri nets can be difficult to interpret by human readers, and might often be more complex then users require. This is indicated for example by the heavy use of Inductive Miner variants in business applications, despite its strong limitations on the discoverable control-flow structures.

In the following, we present an adaption of our eST-Mining algorithm, that aims to discover uniwired Petri nets as introduced in Sec. 4. This approach yields several advantages: the representational bias provided by uniwired Petri nets ensures non-complex and understandable models are discovered, while at the same time allowing for traditionally hard-to-mine control-flow structures such as long-term dependencies. In contrast to many other discovery algorithms, e.g. Inductive Miner, our algorithm is able to discover such structures.

Another advantage of searching for uniwired Petri nets is that incorporating their restrictions in our search for fitting places naturally leads to a massive increase in skipped candidate places and thus efficiency of our discovery algorithm. Additionally, this approach greatly decreases the amount of implicit places found, and thus the complexity of the post-processing step. We will provide details on this variant of our algorithm in the remainder of this section.

Adaption to Uniwired Discovery: The efficiency of our eST-mining algorithm (see Sec. 3) strongly depends on the amount of candidate places skipped by cutting off subtrees in the search space. In the following we optimize this strategy towards the discovery of uniwired Petri nets.

The idea is based on the simple observation, that in a uniwired Petri net there can be only one place connecting two activities. With this in mind, consider our tree-like organized search space. As shown in [7], for every candidate place $p = (I|O)$ it holds that for each of its descendants in the tree $p' = (I'|O')'$ we have that $I \subseteq I'$ and $O \subseteq O'$. In particular, every pair of activities wired by p will also be wired by any descendant p'. Thus, if we include p in our set of fitting places $\mathcal{P}_{fit}$, the whole subtree rooted in p becomes uninteresting and can be skipped.

Additionally, the same two activities will be wired by candidates located in completely independent subtrees. Again, once we have chosen such a place, all these candidates and their children become uninteresting and can be cut off. To keep track of the activities that have already been wired within other subtrees, we globally update the set $wired(\mathcal{P}_{fit})$ (see Definition 5).

Lemma 2 (Bound on Fitting Places). *The set of places discovered by the uniwired variant of our algorithm can contain at most one place for each tree in the search space, that is at most* $|A-1|^2$.

Proof. Our trees are structured in an incremental way: for a root candidate $p_1 = (I_1|O_1)$, for every two descendants $p_2 = (I_2|O_2)$ and $p_3 = (I_3|O_3)$ of p_1 we have that $I_1 \subseteq I_2, O_1 \subseteq O_2$ and $I_1 \subseteq I_3, O_1 \subseteq O_3$. Since we do not allow for candidates with empty activity sets, this implies that $I_2 \cap I_3 \neq \emptyset, O_2 \cap O_3 \neq \emptyset$, and thus a Petri net containing both places would be biwired. We conclude, that for each base root only one descendant (including the root itself) can be part of the discovered set of places. □

While this basic approach is very simple and intuitive, and also results in an enormous decrease in computation time as well as found implicit places, several complications arise. The basic problem is the competition between different fitting candidates, that are wiring the same activities and therefore excluding each other. Our discovery algorithm has to decide which of these conflicting places to include in the final model. Several manifestations of this problem are discussed and addressed in the following, resulting in an incremental improvement of the naive approach.

Included in some of these strategies, we use a heuristic approach to determine how interesting a candidate is. We define a score that is based on the strength of directly-follows-relations expressed by a place $(I|O)$:

$$S((I|O)) = \sum_{x \in I, y \in O} \frac{\#(x,y)}{|I| \cdot |O|},$$

where $\#(x,y)$ denotes the number of times x is directly followed by y in the input log. Applying this heuristic will result in prioritizing places that have a higher token throughput and are therefore expected to better reflect the behavior defined by the log.

Self-looping places: The first conflict we discuss is related to self-looping places. A place $p = (I|O)$ is *self-looping*, if there is at least one activity a with $a \in I, a \in O$. Self-loops can be important to model process behavior, and should not be ignored. However, our naive approach displays a significant drawback with respect to self-looping places: after discovering this place p, the looping activity a is wired, and thus no other place with this self-loop and additional interesting control-flows can be found, since these places are always contained in the skipped subtrees. For example, a place $p' = (I \cup \{a_1\}|O \cup \{a_2\})$ cannot be discovered, which includes places with more than one self-loop.

Fortunately, we can easily fix these problems related to self-loops. Before starting our search for fitting places, we set $wired(\mathcal{P}_{fit}) = \{(a,a) \mid a \in A\}$. Thus all subtrees containing self-loops are skipped, resulting in a discovered set of fitting places without any self-loops. As an additional post-processing step we replay the log on each fitting place extended by each possible self-loop, resulting in at most $|P_{fit}| \cdot |A|$ additional replays. We then insert a copy of each fitting place with its maximal number of self-loop activities added. Superfluous places created this way are deleted during the final implicit-places-removal. Note, that each activity may be wired at most once, i.e. may be involved in at most one self-loop. This results in conflicts between fitting places that can loop on the same activity. We resolve these conflicts by favoring more interesting, i.e. higher scoring places.

Conflicting places of varying complexity: Consider two fitting places $p_1 = (a|b)$ and $p_2 = (c|d)$, implying the existence of another fitting place $p_3 = (a,c|b,d)$. Obviously, p_1 and p_2 cannot coexist with p_3 in a uniwired Petri net. In this scenario, p_1 and p_2 are also preferable to p_3, since they are much more readable, simpler and constraining. However, the depth-first search of our original eST-Miner is likely to encounter p_3 first, update the set $wired(\mathcal{P}_{fit})$ accordingly, and thus prevent p_1 and p_2 from being found. This can result in overly complex and unintuitive process models, because simple connections end up being modeled by a collection of very complicated places, as illustrated in N_3 of Fig. 2. In comparison to the desired Petri net N_1, this model includes for example the complex places $(a,c,d,f,g|e,\blacksquare)$ and $(\blacktriangleright,b|c,d,\blacksquare)$, as well as an additional place $(b|e)$. Clearly, the simpler places $(c,d|e)$ and $(b|c,d)$ resulting in the place $(b|e)$ being implicit, would be preferable.

This conflict between complex places and more simple places constituting a similar control flow can be avoided by prioritizing simple places in our search. By adopting a traversal pattern more similar to breadth-first-search, we ensure that candidate places with less activities, which are closer to the base roots, are evaluated first. Only after considering all candidates with a certain distance, we proceed further down the trees, thus ensuring that for a pair of activities we always choose the simplest place to wire them. With respect to the introductory example, we would find p_1 and p_2, thus skipping the subtree that contains p_3.

We adapt our original algorithm to this new traversal strategy by transforming the list of base roots to a queue. After removing and evaluating the first candidate in this queue, which in the beginning are the base roots, we add any potentially interesting child candidate to the end of the queue. Rather than going into the depth of the tree, we proceed with evaluating the new first element. We continue to cut off subtrees by simply not adding uninteresting children to the queue. When evaluating a candidate, before replay we check the set $wired(\mathcal{P}_{fit})$, which might have been updated since the candidate was added.

Conflicting places of similar complexity: While the base root level contains only candidate places with non-overlapping pairs of input and output activities, this is not the case for deeper levels. Thus, when evaluating the places of a level it is possible that two places are fitting but cannot coexist in a uniwired Petri net. Simply choosing the first such place can lead to the skipping of very interesting places in favor of places that do not carry as much information. An example is the Petri net N_2 shown in Fig. 2. Here, the place $(a, b|\blacksquare)$ is inserted first, resulting in the much more interesting place $(a, h|\blacksquare)$ of same complexity being skipped.

To circumvent this problem we first find all fitting candidates from the same level (i.e. the same distance from the roots) and then check for conflicts. For each set of conflicting places we chose the one scoring best with respect to the directly-follows-based heuristics described previously. This way we can give preference to candidates with high token-throughput. With respect to the running example in Fig. 2 we rediscover the desired net N_1.

The uniwired variant of eST-Miner described in this section to some degree guarantees the discovery of desired places:

Lemma 3 (Maximal set of discovered places). *Let $\mathcal{P}$ be the set of places discovered by the algorithm. No non-implicit, fitting place can be added to $\mathcal{P}$ without making the Petri net biwired.*

Proof. Assume there would be a non-implicit, fitting place p, that could be added to $\mathcal{P}$ without making the net biwired. There are two possibilities for a fitting candidate not to be discovered: either it is conflicting with another place that was chosen instead or it is located in a subtree that was cut-off.

First assume p was discarded in favor of a conflicting place. This implies that there is a pair of transitions wired by both places. Since the other place was discovered, p cannot be added without making the net biwired.

Now assume p was located in a subtree that was cut-off. Since p is fitting, our original eST-Miner does not cut-off this subtree. Thus the subtree was cut-off because the places it contained were already wired. This implies that p wires a set of transitions that is also wired by another, previously discovered place. Thus adding p would make the net biwired.

We conclude, that no place can be added to $\mathcal{P}$ without making the resulting Petri net biwired. □

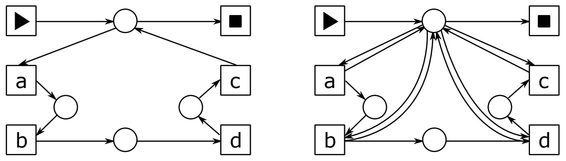

Fig. 3. The left model contains the place $(\blacktriangleright, c|a, \blacksquare)$, that cannot be discovered by our variant. Instead, we discover the model shown on the right, where the missing place is replaced by one having several self-loops.

Table 1. List of logs used for evaluation. The upper part lists real-life logs, the lower part shows artificial logs. Logs are referred to by their abbreviations. The log `HP2018` has not yet been published. The much smaller 2017 version can be found in [10].

Log Name	**Abbreviation**	**Activities**	**Trace Variants**	**Reference**
BPI13-incidents	`BPI13`	15	2278	[11]
HelpDesk2018SiavAnon	`HD2018`	12	2179	(see caption)
Sepsis	`Sepsis`	16	846	[12]
Road Traffic Fine Management	`RTFM`	11	231	[13]
Reviewing	`Reviewing`	16	96	[14]
repairexample	`Repair`	12	77	[14]
Teleclaims	`Teleclaims`	11	12	[14]

Lemma 3 guarantees that we find a maximal number of fitting places, i. e. no place can be added to the resulting net without making it biwired. However, discovered places might still be extended by adding more ingoing and outgoing activities. The presented optimizations aim to find the most expressive of all possible maximal sets of places.

6 Testing Results and Evaluation

The uniwired Petri nets, that can be discovered using our new variant of eST-Miner, may express advanced behaviors. In particular, we are capable of finding complex structures like long-term dependencies, as illustrated by N_1 in Fig. 2.

A category of conflicting places for which an efficient solution has yet to be found are certain places within the same subtree: assume there is a fitting place $p_1 = (I_1|O_1)$ and a conflicting fitting place $p = (I_1 \cup I_2|O_1 \cup O_2)$ within the subtree rooted in p_1. Then our algorithm will choose the simpler place p_1 and skip p. This is fine in the case described in Sec. 5, with the fitting and non-conflicting place $p_2 = (I_2|O_2)$ being part of a different subtree, but problematic if p_2 is unfitting: the relation between I_2 and O_2 exists, but will not be expressed by the discovered model. This is illustrated in Fig. 3.

For the evaluation of efficiency, we use similar logs as in our original paper ([7]) as specified in Table 1. The eST-Miner algorithm increases efficiency by skipping places that are guaranteed to be unfitting. In addition to the places cut off by the original algorithm, the uniwiring variant skips all subtrees that contain places that have been wired already. As illustrated in Fig. 4, this immensely increases the fraction of cut-off candidates: for all our test-logs the percentage of skipped candidates surpasses 99%. The

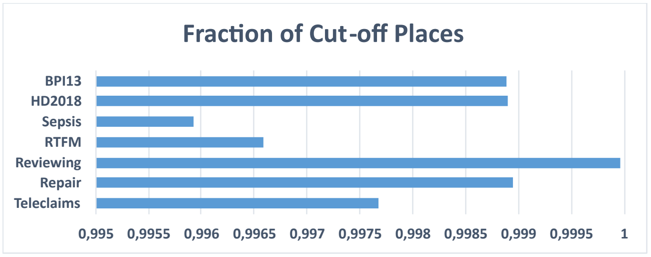

Fig. 4. In comparison to the original algorithm, the uniwired variant is able to increase the fraction of cut-off places dramatically.

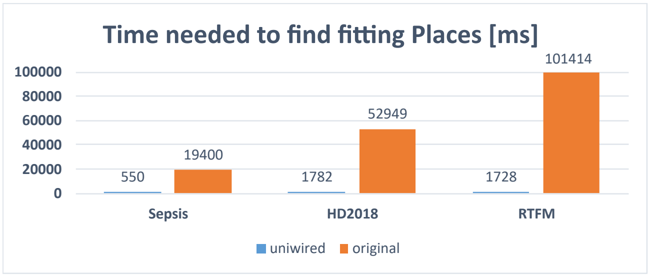

Fig. 5. Comparison of computation time needed by the original eST-Miner and its uniwired variant for computing the set of fitting places. For comparability slightly modified versions of the logs `Sepsis` and `HD2018` were used as detailed in [7].

corresponding increase in efficiency in comparison to the original eST-Miner is presented in Fig. 5. For the tested logs, the uniwired variant proves to be up to 60 times faster than the original algorithm and up to 600 times faster than the brute force approach evaluating all candidates [7]. The results of our evaluation show the potential of uniwired Petri nets in combination with our eST-Miner variant: interesting process models that can represent complex structures such as long-term dependencies can be discovered, while immensely increasing efficiency by skipping nearly all of the uninteresting candidate places.

7 Conclusion

While the original eST-Mining algorithm is capable of discovering traditionally hard-to-mine, complex control-flow structures much faster than the brute force approach, and provides an interesting new approach to process discovery, it still displays some weaknesses. In particular, the computation time for finding the set of fitting places as well as removing implicit places is to high and the resulting models are very precise but hard to read for human users, due to lack of simplicity.

In this paper, we present a new class of Petri nets, the *uniwired Petri nets*. We have shown that they are quite expressive, since they contain the class of SWF-nets, but are strictly larger. In particular, they can model non-free choice constructs. By providing a well-balanced trade-off between simplicity and expressiveness, they seem to introduce a very interesting representational bias to process discovery. We describe a variant of eST-Miner, that aims to discover uniwired Petri nets. While still being able to discover non-free choice constructs, utilizing the uniwiredness-requirement allows us to skip an astonishingly large part of the search space, leading to a massive increase in efficiency when searching for the set of fitting places. At the same time the number of found implicit places is drastically decreased.

For future work, we would like to extend the results on expressiveness and relevance of uniwired Petri nets. Our corresponding discovery algorithm could also be improved in particular with respect to conflicting places within the same subtree, and by refining our scoring system. The influence of the order of candidates within the tree structure should be investigated as well. Additional strategies for skipping even more sets of places, as well as adequate abstractions of the log, can be particularly interesting when analyzing larger logs.

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

References

1. Wen, L., van der Aalst, W., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Mining and Knowledge Discovery **15**(2) (2007)
2. Leemans, S., Fahland, D., van der Aalst, W.: Discovering block-structured process models from event logs - a constructive approach. Application and Theory of Petri Nets and Concurrency (2013)
3. Badouel, E., Bernardinello, L., Darondeau, P.: Petri Net Synthesis. Text in Theoretical Computer Science, an EATCS Series. Springer (2015)
4. Lorenz, R., Mauser, S., Juhás, G.: How to synthesize nets from languages: A survey. In: Proceedings of the 39th Conference on Winter Simulation: 40 Years! The Best is Yet to Come, WSC '07. IEEE Press (2007)
5. van der Werf, J.M., van Dongen, B., Hurkens, C., Serebrenik, A.: Process discovery using integer linear programming. In: Applications and Theory of Petri Nets. Springer (2008)
6. van Zelst, S., van Dongen, B., van der Aalst, W.: Avoiding over-fitting in ILP-based process discovery. In: Business Process Management. Springer International Publishing (2015)
7. Mannel, L., van der Aalst, W.: Finding complex process-structures by exploiting the token-game. In: Application and Theory of Petri Nets and Concurrency. Springer Nature Switzerland AG (2019 (to be published))
8. van der Aalst, W.: Discovering the "glue" connecting activities - exploiting monotonicity to learn places faster. In: It's All About Coordination - Essays to Celebrate the Lifelong Scientific Achievements of Farhad Arbab (2018)
9. Aalst, W., Weijters, A., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering **16**(9)
10. Polato, M.: Dataset belonging to the help desk log of an Italian company (2017)
11. Steeman, W.: BPI challenge 2013, incidents (2013). DOI 10.4121/UUID:500573E6-ACCC-4B0C-9576-AA5468B10CEE
12. Mannhardt, F.: Sepsis cases - event log (2016)
13. De Leoni, M., Mannhardt, F.: Road traffic fine management process (2015)
14. van der Aalst, W.M.P.: Event logs and models used in Process Mining: Data Science in Action (2016). URL http://www.processmining.org/event_logs_and_models_used_in_book