Improving the State-Space Traversal of the eST-Miner by Exploiting Underlying Log Structures

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

[1] PADS Group, RWTH Aachen University
{mannel, wvdaalst}@pads.rwth-aachen.de
[2] RWTH Aachen University
yannick.epstein@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, Petri nets form a theoretically well-explored description language. In this paper, we present an extension of the process discovery algorithm eST-Miner. This approach computes the maximal set of non-redundant places, that are considered to be fitting with respect to a user-definable fraction of the behavior described by the given event log, by evaluating all possible candidate places using token-based replay. The number of candidate places is exponential in the number of activities, and thus evaluating all of them by replay is very time-consuming. To increase efficiency, the eST-miner organizes these candidates in a special search structure, that allows to skip large chunks of the search space, while still returning all the fitting places. While this greatly increases its efficiency compared to the brute force approach evaluating all the candidates, the miner is still very slow compared to other approaches. In this paper, we explore two approaches to increase the fraction of skipped candidates and thus the efficiency of the eST-Miner. The impact of the presented concepts is evaluated by various experiments using both real and artificial event logs.

Keywords: Process Discovery, Petri Nets, eST-Miner

1 Introduction and Related Work

Most corporations and organizations support their processes using information systems, while recording behavior that can be extracted in the form of *event logs*. Each event in such a log has a name identifying the executed activity (activity name), an identification mapping the event to some execution instance (case id), a time stamp showing when the event was observed, and often extended meta-data of the activity or process instance. In the field of *process discovery*, we utilize the event log to identify relations between the activities (e.g. pre-conditions, choices, concurrency), which are then expressed within a process model. This is non-trivial for various reasons. We cannot assume that the given event log is complete, as some possible behavior might be yet unobserved. Also, real life event logs often contain noise, which we would like to filter out. Correctly classifying behavior as noise can be hard to impossible. An ideal process model can reproduce all behavior contained in an event log, while not allowing for unobserved behavior. It should represent all dependencies between events and at the same time be simple enough to be understandable by a human interpreter. Computation should be fast

and robust to noise. Usually, it is impossible to fulfill all these requirements at the same time. Thus, different algorithms focus on different quality criteria, while neglecting others. As a result, the models returned for a given event log can differ significantly.

Many existing discovery algorithms abstract from the full information given in a log and/or generate places heuristically, in order to decrease computation time and complexity of the returned process models. While this is convenient in many applied settings, the resulting models often are underfitting and sometimes even unsound. Examples are the Alpha Miner variants ([1]), the Inductive Mining family ([2]), Split-Miner ([3]), genetic algorithms or Heuristic Miner. In contrast to these approaches, which are not able to (reliably) discover complex model structures, algorithms based on region theory ([4–7]) discover models whose behavior is the minimal behavior representing the log. On the downside, these approaches are known to be rather time-consuming, cannot handle low-frequent behavior, and tend to produce complex, overfitting models which can be hard to interpret.

In [8] we introduce the discovery algorithm eST-Miner. 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. 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.

While traditional region-theory uses a global perspective to find a set of feasible places, the eST-Miner evaluates each place separately, that is from a local perspective. 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.

The most severe limitation of the eST-Miner is its high computation time, even on small event logs. This is due to the extensive search of whole candidate space, as well as the even more time-consuming removal of the many so-called *implicit* places during a post-processing. These implicit places are fitting with respect to the log, but do not restrict behavior of the Petri net, and thus they unnecessarily clutter the model. To tackle these performance problems, we present two approaches, one of which does not change the result, while the other one does. We aim to decrease computation time by further reducing the searched fraction of the candidate space, and already discarding a large number of fitting but uninteresting places during the search, thus speeding up both the search and the post-processing phase.

In Sec. 2 we provide basic notation and definitions. Afterwards, we briefly review the basics of the standard eST-Miner (Sec. 3). Our new concepts are introduced in Sections 4 and 5, and their experimental evaluation is presented in Section 6. Finally, Section 7 concludes this work by summarizing our work and findings and suggesting possibilities for future work.

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 ($\blacktriangleright$) and end with a designated end activity ($\blacksquare$). 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 $\blacktriangleright \in \mathcal{A}$ be a designated start activity and let $\blacksquare \in \mathcal{A}$ be a designated end activity. A* trace *is a sequence containing $\blacktriangleright$ as the first element, $\blacksquare$ as the last element and in-between elements of $\mathcal{A} \setminus \{\blacktriangleright, \blacksquare\}$. 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 $\blacktriangleright$ and ending with $\blacksquare$. These places are uniquely identified by the set of input activities I and output activities O. Each activity corresponds to exactly one activity, therefore, this paper refers 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 ($\{\blacktriangleright, \blacksquare\} \subseteq A$) 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 (Overfed/Underfed/Fitting Places, see [9]). *Let $N = (A, \mathcal{P})$ be a Petri net, let $p = (I|O) \in \mathcal{P}$ be a place, and let σ be a trace. With respect to the given trace σ, p is called*

- underfed*, denoted by $\bigtriangledown_\sigma(p)$, if and only if $\exists k \in \{1, 2, ..., |\sigma|\}$ such that $|\{i \mid i \in \{1, 2, ...k-1\} \wedge \sigma(i) \in I\}| < |\{i \mid i \in \{1, 2, ...k\} \wedge \sigma(i) \in O\}|$,*
- overfed*, denoted by $\bigtriangleup_\sigma(p)$, if and only if $|\{i \mid i \in \{1, 2, ...|\sigma|\} \wedge \sigma(i) \in I\}| > |\{i \mid i \in \{1, 2, ...|\sigma|\} \wedge \sigma(i) \in O\}|$,*
- fitting*, denoted by $\square_\sigma(p)$, if and only if not $\bigtriangledown_\sigma(p)$ and not $\bigtriangleup_\sigma(p)$.*

We extend these notions to the log whole log using the noise parameter: with respect to a log L and parameter $\tau \in [0, 1]$, p is called

- underfed*, denoted by $\bigtriangledown^\tau_L(p)$, if and only if $|\{\sigma \in L \mid \bigtriangledown_\sigma(p)\}|\backslash|L| > 1 - \tau$,*
- overfed*, denoted by $\bigtriangleup^\tau_L(p)$, if and only if $|\{\sigma \in L \mid \bigtriangleup_\sigma(p)\}|\backslash|L| > 1 - \tau$,*
- fitting*, denoted by $\square^\tau_L(p)$, if and only if $|\{\sigma \in L \mid \square_\sigma(p)\}|\backslash|L| \geq \tau$.*

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 $\langle \blacktriangleright, a_1, a_2, \ldots a_n, \blacksquare\rangle$ (due to Def. 1) such that places are empty at the end of the trace and never have a negative number of tokens.

3 Introducing the eST-Miner

We briefly introduce the original eST-Miner first presented in [8]. 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 a fraction τ of traces in the event log is 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, the subset of all fitting places is computed and inserted. To facilitate further computations and human readability, all unneeded, i.e., implicit places are removed from this intermediate result in a post-processing step.

The algorithm uses token-based replay to discover all fitting places out of the set of possible candidate places. To avoid replaying the log on the exponential number of candidates (i.e. all pairs of subsets of activities, $(2^{|A|} - 1)^2$, 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 Fig. 1. Note the incremental structure of the trees, i.e., the increase in distance from the roots corresponds to the increase of input (red edges) and output (blue edges) activities. However, the organization of candidates within the same depth and their connections to other candidates is not fixed, but defined by the order of ingoing activities ($>_i$) and outgoing activities ($>_o$). Additionally, note that blue edges are always part of a purely blue subtree, while red edges may connect subtrees that contain blue edges as well.

Definition 5 (Complete Candidate Tree). *Let A be a set of activities and let $>_i, >_o$ be two total orderings on this set of activities. A* complete candidate tree *is a pair $CT = (N, F)$ with $N = \{(I|O) \mid I \subseteq A \backslash \{\blacksquare\} \wedge O \subseteq A \backslash \{\blacktriangleright\} \wedge I \neq \emptyset \wedge O \neq \emptyset\}$. We have that $F = F_{red} \cup F_{blue}$, with*

$$
\begin{aligned}
F_{red} = &\{((I_1|O_1),(I_2|O_2)) \in N \times N \mid |O_2| = 1 \wedge O_1 = O_2 \\
&\wedge \exists a \in I_1 \colon \big(I_2 \cup \{a\} = I_1 \wedge \forall a' \in I_2 \colon a >_i a'\big)\} \textbf{ (red edges)} \\
F_{blue} = &\{((I_1|O_1),(I_2|O_2)) \in N \times N \mid I_1 = I_2 \\
&\wedge \exists a \in O_1 \colon \big(O_2 \cup \{a\} = O_1 \wedge \forall a' \in O_2 \colon a >_o a'\big)\} \textbf{ (blue edges)}.
\end{aligned}
$$

If $((I_1|O_1),(I_2|O_2)) \in F$, we call the candidate $(I_1|O_1)$ the child *of its* parent $(I_2|O_2)$.

The runtime of the original eST-Miner strongly depends on the number of candidate places skipped during the search for fitting places. The approach uses results of evaluating a place p to skip subtrees of that place, that are known to be unfitting. For example, if 80% of the traces cannot be replayed because p is empty and does not enable the next activity in the trace, i.e., $\bigtriangledown_L^{0.8}(p)$, then at least 80% will not allow for a place p' with even more output activities, i.e. we know that $\bigtriangledown_L^{0.8}(p')$. With respect to the tree-structured candidate traversal, this indicates that all purely blue (outgoing activity is added) subtrees of p can be cut off. If p was overfed, we could cut off all purely red (ingoing activity added) subtrees, respectively.

While this results in a significant decrease in computation time compared to the brute force approach, the algorithm is still slow compared to most other discovery ap-

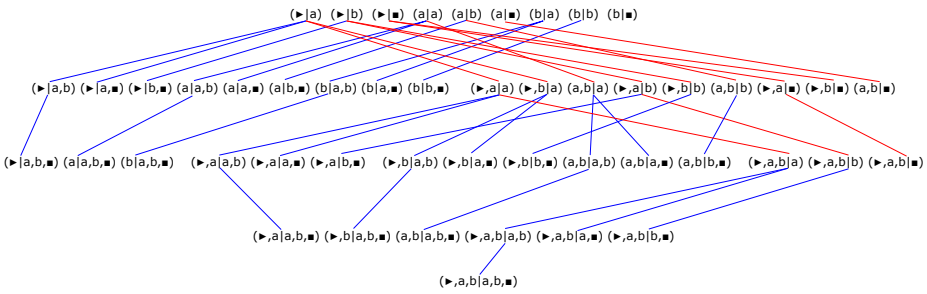

Fig. 1: Example of a tree-structured candidate space for the set of activities $\{\blacktriangleright, A, B, \blacksquare\}$, with orderings $\blacksquare >_i B >_i A >_i \blacktriangleright$ and $\blacksquare >_o B >_o A >_o \blacktriangleright$.

proaches. In this paper, we seek to maximize the number of skipped candidates and thus decrease runtime. We introduce two different heuristic strategies aiming to improve the discovery phase of the eST-Miner. The first strategy is based on organizing the candidates within the tree structure in such a way that the amount of skipped candidate places is maximized. By skipping more candidates, we need to evaluate fewer places and thus terminate faster, without compromising the result, i.e., the discovered set of fitting places remains the same. The second strategy adds additional cut-off criteria to the tree traversal. It heuristically determines certain subtrees to be uninteresting and thus skippable. This way we do not only speed up the search phase, but also significantly reduce the number of implicit places discovered, leading to less time needed for post-processing. However, the returned Petri nets may differ from the original variant.

4 Optimizing the Tree Structure

The positioning of candidate places within the tree-like search structure CT (Def. 5) is directly defined by the two orderings $>_i$ and $>_o$ on the set of all activities A. Consider a place $p = (I|O)$. A red child place of p is a place p_{red} with $p_{\text{red}}\bullet = p\bullet$ and $\bullet p_{\text{red}} = \bullet p \cup \{a\}$, such that $a \in A$ and $\forall b \in \bullet p\colon a >_i b$ (also, $|p\bullet| = 1$, but we focus on the order here). A blue child place of p is a place p_{blue} with $\bullet p_{\text{blue}} = \bullet p$ and $p_{\text{blue}}\bullet = p\bullet \cup \{a\}$, such that $a \in A$ and $\forall b \in p\bullet\colon a >_o b$. Note that the number of children of p is directly defined by the two orderings as well: if $a \in O$ (respectively $a \in I$) is the maximal activity with respect to $>_o$ (respectively $>_i$) of the outgoing (respectively ingoing) activities of p, then the number of blue (respectively red) children of p equals $|\{b \in A \mid b >_o a\}|$ (respectively $|\{b \in A \mid b >_o a\}|$).

We can skip all purely blue subtrees of places which are underfed, and thus we want to maximize the number of blue descendants of such places. Similarly, we want to maximize the number of skippable red descendants of overfed places. Experimental results ([8]) have shown that varying the orderings has a significant effect on the number of places that are cut off and thus the runtime, without changing the final result.

Computing an optimal traversal ordering is unfeasible, and thus we investigate easily computable approximations. In the following, we present heuristic strategies for choosing the orderings $>_i$ and $>_o$ aiming to maximize the number of cut off places. Consider the event log $L = \left[\langle \blacktriangleright, a, b, b, \blacksquare\rangle, \langle \blacktriangleright, c, b, b, b, \blacksquare\rangle^3\right]$ as a motivational example. We observe that the activity b occurs comparatively often in each trace. Thus, places which have b as an outgoing activity are likely to be underfed for each trace. To

increase the number of places which are cut off, we want to maximize the number of blue children for such places. Similarly, places with b as incoming activity are likely to be overfed, and thus we would like to maximize their number of red children.

The intuitive idea illustrated by the example leads to a variety of metrics definable on the activities of the event log, aiming to quantify this intuition. The *Absolute Activity Frequency* counts the number of occurrences of an activity accumulated over all traces in the log. The *Absolute Trace Frequency* counts the number of traces in which an activity occurs. The *Average Trace Occurrence* is defined by the average number of occurrences of an activity in a trace of the log. Finally, by the *Average First Occurence Index* of an activity, we refer to the first index at which an activity occurs in a trace, averaged over the whole log.

Definition 6 (Metrics on Log Properties). *Let L be an event log and $\mathcal{A} \subseteq \mathbb{A}$ be the set of activities, which occur in the log. We assign numerical values to these activities using the following functions:* [1]

- Absolute Activity Frequency:
  $\mathtt{absAF}\colon \mathcal{A} \to \mathbb{N}, \mathtt{absAF}(a) = \sum_{\sigma \in L} |\{i \in \{1, ..., |\sigma|\} \mid \sigma(i) = a\}|$
- Absolute Trace Frequency: $\mathtt{absTF}\colon \mathcal{A} \to \mathbb{N}, \mathtt{absTF}(a) = |[\sigma \in L \mid a \in \sigma]|$
- Average Trace Occurrence: $\mathtt{avgTO}\colon \mathcal{A} \to \mathbb{Q}, \mathtt{avgTO}(a) = \frac{\sum_{\sigma \in L} |\{i \in \{1,...,|\sigma|\} | \sigma(i) = a\}| / |\sigma|}{|L|}$
- Average First Occurence Index:
  $\mathtt{avgFOI}\colon \mathcal{A} \to \mathbb{Q}, \mathtt{avgFOI}(a) = \frac{\sum_{\sigma \in L} min\{i \in \{1,2,...,|\sigma|\} | \sigma(i) = a\}}{\mathtt{absTF}(a)}$

If $\mathtt{absAF}(a)$ is high, we expect many tokens to be produced (consumed) for places that have a as an ingoing (outgoing) activity during replay of the log, and thus such places are more likely to be underfed (overfed). The same holds for high $\mathtt{absTF}(a)$ and $\mathtt{avgTO}(a)$. If $\mathtt{avgFOI}(a)$ is low, we can expect the activity a to generate or consume tokens early on during the replay of a trace. Places which have outgoing activities with low average first occurrence index are more likely to be underfed, as their output activities may require tokens early on during replay, where none might be available.

Definition 7 (Orderings Based on Metrics). *Let L be an event log and $\mathcal{A} \subseteq \mathbb{A}$ be the set of activities, which occur in the log. Based on the metrics given in Def. 6 we propose the following orderings:*

- $\mathtt{absAF}(a) > \mathtt{absAF}(b) \Leftrightarrow a <_i^{\mathtt{absAF}} b \Leftrightarrow a <_o^{\mathtt{absAF}} b$ *(high frequencies first)*
- $\mathtt{absTF}(a) > \mathtt{absTF}(b) \Leftrightarrow a <_i^{\mathtt{absTF}} b \Leftrightarrow a <_o^{\mathtt{absTF}} b$ *(high frequencies first)*
- $\mathtt{avgTO}(a) > \mathtt{avgTO}(b) \Leftrightarrow a <_i^{\mathtt{avgTO}} b \Leftrightarrow a <_o^{\mathtt{avgTO}} b$ *(high occurrences first)*
- $\mathtt{avgFOI}(a) > \mathtt{avgFOI}(b) \Leftrightarrow a <_i^{\mathtt{avgFOI}} b \Leftrightarrow a >_o^{\mathtt{avgFOI}} b$
  (early activities first for ingoing, last for outgoing activities)

Experimental results investigating and comparing the impact of the presented orderings are presented in Section 6.

[1] Note that $\sum_{\sigma \in L} f(\sigma)$ and $[\sigma \in L | f(\sigma)]$ operate on multisets, i.e., if the same trace σ appears multiple times in L, this is taken into account.

5 Pruning Uninteresting Subtrees

Our second strategy adds an additional, heuristic criterion to identify and skip uninteresting candidate subtrees. We notice, that fitting places returned by the eST-Miner often have no evidence, and sometimes even have counter-evidence, in the event log. For example, consider the event log $L = [\langle \blacktriangleright, a, c, d, e, \blacksquare \rangle, \langle \blacktriangleright, b, c, d, f, \blacksquare \rangle]$. The place $p_1 = (a, b|e, f)$ is perfectly fitting with respect to this log. However, it describes dependencies of f on a and e on b, which have no evidence in the event log. The places $p_2 = (a|e)$ and $p_3 = (b|f)$, which are fitting and thus discovered by eST-Miner as well, describe the dependencies much better and make the place p_1 superfluous. We aim to skip p_1 and its whole subtree, since all contained candidates describe the unsupported dependencies and could be replaced by better places contained in different subtrees.

In the following, we introduce a heuristic approach assigning an *interest score* to each place based on the *eventually-follows relations* between its ingoing and outgoing activities. This score is defined in such a way, that it can only decrease with increasing depth, i.e., every place is at most as interesting as its parent. This property allows us to skip whole subtrees based on the score assigned to the root of this subtree.

Definition 8 (Eventually-Follows Relation). *Let $a, b \in \mathcal{A}$ be two (possibly equal) activities. We say that b* eventually follows *a in a trace σ, if a a occurs in σ and later b occurs in σ. Formally, $a \rightsquigarrow_\sigma b := \exists i, j \in \{1, ..., |\sigma|\} : \big(i < j \wedge \sigma(i) = a \wedge \sigma(j) = b\big)$.*

The interest score based on the eventually-follows relation is based on the intuition, that a place is interesting only if all pairs of ingoing and outgoing activities have more evidence than counter-evidence in the event log. The relation between evidence and acceptable counter-evidence is defined by the parameter λ.

Definition 9 (Interest Score). *Let L be an event log and $A \subseteq \mathcal{A}$ the set of activities which occur in L. For a pair of activities $(a, b) \in A \times A$ (possibly $a = b$), we define the* interest score *as*

$$\rightsquigarrow_L\big((a, b)\big) := \frac{|[\sigma \in L \mid a \rightsquigarrow_\sigma b]|}{max(1, |[\sigma \in L \mid a \in \sigma \wedge b \in \sigma]|)}.$$

We define the interest score of a place $p = (I|O)$ as

$$\mathtt{is}_L(p) = min\big(\{\rightsquigarrow_L\big((a, b)\big) \mid a \in I \wedge b \in O\}\big).$$

The place p is called λ-interesting*, if for a user-definable parameter $\lambda \in [0, 1]$ we have that $\mathtt{is}_L(p) \geq \lambda$.*

The interest score can be directly integrated into the eST-Miners search phase to skip additional subtrees. In particular, whenever we encounter a place p that is not λ-interesting, we conclude that none of its descendants is λ-interesting, and thus we can skip the whole subtree rooted in p (Prop. 1). This follows directly from the incremental construction of the tree structure (Def. 5) and the fact that the minimum is taken when computing $\mathtt{is}(p)$ (Def. 9).

Proposition 1 (Interest Score Cannot Increase). *Let L be an event log and $\mathcal{A} \subseteq \mathbb{A}$ the set of activities, which occur in L. Let CT be the search structure of the eST-Miner. Let $p = (I|O)$ be a place with $I \subseteq \mathcal{A}$ and $O \subseteq \mathcal{A}$. Let $\lambda \in [0, 1]$ be a parameter. If p is not a λ-interesting place, then none of its descendants in CT is λ-interesting.*

Table 1: Overview of the event logs used in our experiments. `SepsisA` was obtained from the original log by removing the 9 least frequent activities. `TeleclaimsT` contains the 85% most frequent traces of the original log.

Log Type	Log Name	Abbreviation	Activities	Trace Variants	Source
Real-Life	Sepsis-activity filtered	`SepsisA`	9	642	[10]
	Road Traffic Fines Management	`RTFM`	11	231	[11]
Artificial	Teleclaims-trace filtered	`TeleclaimsT`	10	12	[12]
	repairexample	`Repair`	12	77	[12]

Table 2: Runtimes of the search phase with different heuristics applied, represented as percentage of the original eST-Miner's (lexicographical ordering for $>_i$ and $>_o$) search phase runtime, with minimal and maximal times given as absolute values. All values are averaged over values for $\tau \in \{1.0, 0.9, 0.8, 0.7, 0.6, 0.5\}$. We can see that the impact of the different activity orderings (Sec. 4) on the time performance is much lower than for the λ-interesting pruning strategy (Sec. 5).

	`Repair`			`RTFM`			`SepsisA`		
	mean	min [s]	max [s]	mean	min [s]	max [s]	mean	min [s]	max [s]
`absAF`	57.84%	34	63	74.69%	4075	11196	77.13%	48	140
`absTF`	76.13%	50	77	72.95%	3994	10983	77.99%	56	139
`avgTO`	63.05%	35	74	75.15%	3972	11671	74.37%	54	131
`avgFOI`	57.67%	33	56	64.94%	3272	10384	69.29%	46	120
$\texttt{is}_L(p) \geq 1.0$	4.94%	1.56	2.69	0.13%	12.91	17.06	0.55%	0.43	0.55

The impact of applying this pruning approach to eST-Miner is evaluated in Section 6.

6 Experimental Evaluation

We implemented the eST-Miner and our proposed extensions within the Python PM4Py framework for process mining (Python version 3.7.1). Our experiments are all executed on an Intel Core i5 (2×2.6GHz) with 8GB of RAM, running MacOS Mojawave (10.14.5). We evaluated real and artificial event logs as presented in Tab. 1.

6.1 Evaluation of Optimizing Orderings

We investigate the impact of the different choices for $>_i$ and $>_o$ as presented in Def. 7 on the time needed for the eST-Miner search phase. We compare the resulting times to a base case, given by the performance on lexicographic (i.e. random) orderings. An overview of our results for different logs is given in Tab. 2. The time needed for the search phase when applying the different orderings is averaged over different values of τ and presented as fractions of the runtime needed on the lexicographical ordering.

The table shows that when applying our proposed orderings, searching the candidate space requires at most 78 % of the timed needed with the lexicographical ordering for all tested event logs. In many cases we achieve a runtime of less than two thirds of this base case. Based on the presented results, we can derive that the ordering based on *average first occurrence index* clearly leads to the shortest runtime for all tested logs. The other strategies, *absolute activity frequency, absolute trace frequency* and *average trace occurrence* all lead to a significant improvement over the lexicographical ordering, but none performs consistently better than the other on all logs.

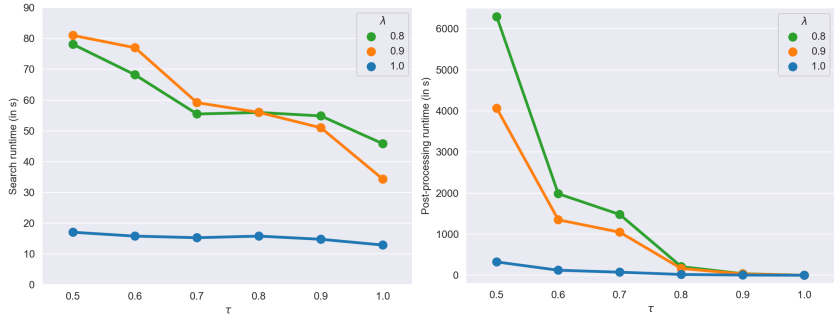

Fig. 2: Investigating runtimes for search (left) and post-processing phase (right) of the ip-eST variant on the RTFM log, for different values of τ and λ.

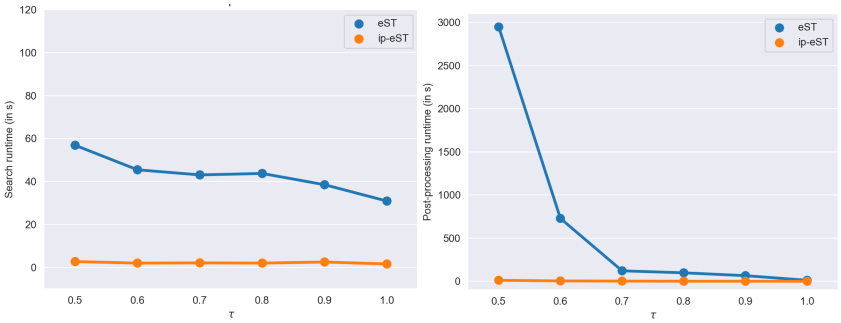

Fig. 3: Comparing runtimes of the original eST-Miner and the ip-eST variant for search (left) and post-processing phase (right) on the Repair event log, for $\lambda = 1.0$ and different values of τ.

We conclude that by choosing the orderings $>_i$ and $>_o$ in a sophisticated way, we are able to perform the search phase in about 60% to 80% of the original runtime, while returning the same set of fitting places.

6.2 Evaluation of Pruning Uninteresting Subtrees

We investigate the potential of the interesting places heuristic as presented in Section 5 to prune uninteresting subtrees and thus speed up the original eST-Miner. This variant will be referred to as *ip-eST* in the following. Our experiments evaluate the runtime of the algorithm as well as the quality of discovered models. First, we explore the impact of choosing different values for λ using the RTFM log as a representative for various experiments. The results are summarized in Fig. 2. We conclude that the improved performance we achieve using a value of $\lambda = 1.0$ rapidly deteriorates for lower values of λ. Since our goal is improved time performance, we focus on high λ values in our other experiments, showing that even for $\lambda = 1.0$ model quality remains acceptable. Tab. 2 summarizes the drastically increased performance for $\lambda = 1.0$, averaged over different values for τ, for various logs as a percentage of the time needed by the original eST-Miner's search phase.

We choose the Repair log to represent our results on time performance. The huge impact of applying the ip-eST variant rather than the original eST-Miner is clearly vis-

Table 3: Time needed for the post-processing step when applying the ip-eST variant with $\lambda = 1.0$, represented as the percentage of time needed by the original eST-Miner, averaged over values for $\tau \in \{1.0, 0.9, 0.8, 0.7, 0.6, 0.5\}$.

	Repair	SepsisA	TeleclaimsT
postprocessing	0.47%	0.17%	0.03%

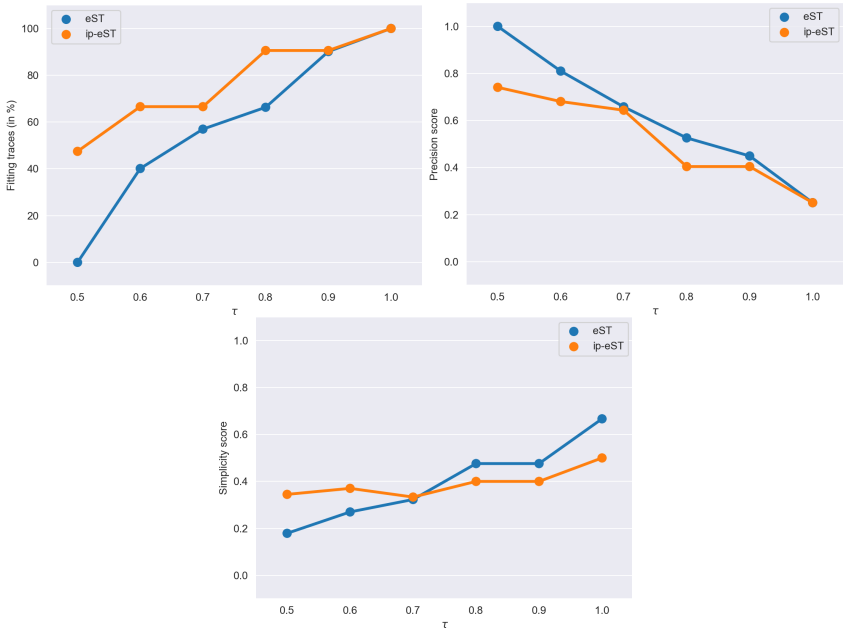

Fig. 4: Comparing quality results of the original eST-Miner and the ip-eST variant on the Repair event log, for $\lambda = 1.0$ and different values of τ.

ible in Fig. 3. For all τ, the search time of the ip-eST variant is only a very small fraction of the standard eST-Miner's search time. The difference becomes larger for smaller values for τ, since the time needed by the original miner increases while the ip-eST variant's search phase runtime remains low. For $\tau = 0.5$, the search phase of the ip-eST-Miner is more than 100 times faster. Since the ip-eST variant returns significantly fewer fitting places than the original eST-Miner, in particular for lower values of τ, the runtime of the post-processing step is greatly decreased. An overview is given in Tab. 3. The difference between those variants increases as τ decreases, peaking in the ip-eST variants post-processing being 4500 times faster than the standard eST-Miner's post-processing at $\tau = 0.5$.

Definition 10 (Simplicity). *We define the simplicity of a Petri net $N = (A, P)$ based on the fraction of nodes being activities:* $\mathtt{simp}(N) = \frac{|A|}{|P|+|A|}$.

In Fig. 4, we investigate the fitness (token-based replay fitness [12]), precision ([13]), and simplicity (Def. 10) of the models returned by the standard eST-Miner and the ip-

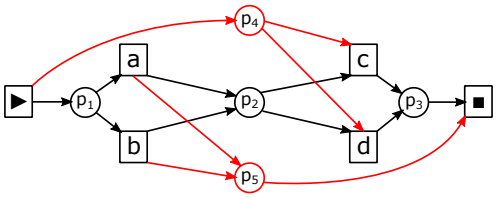

Fig. 5: For the log $[\langle \blacktriangleright, a, c, \blacksquare \rangle^5, \langle \blacktriangleright, b, c, \blacksquare \rangle^5, \langle \blacktriangleright, b, d, \blacksquare \rangle^5]$, the eST-Miner generates the places shown in black, in particular p_2. The ip-eST variant prunes p_2 due to the uninteresting dependency (a, d). Instead, it generates places like p_4 and p_5.

eST variant for different values of τ. To represent the results of our experiments on the various logs, we choose the `Repair` event log. For other logs, the approach produces similar results. Note, that for the models discovered by the original eST-Miner and the ip-eST variant, the performance is very similar for all quality metrics and all values of τ. The models returned by the standard eST-Miner do have a slightly higher precision and slightly lower fitness, which is to be expected since non-implicit fitting places may be skipped during the search phase. It is worth noticing that the ip-eST variant is still capable of discovering long-term dependencies, one of the main features of the eST-Miner.

Recall, that a high number of places results in a low simplicity score. Thus, we expect the ip-eST variant to return models that score at least as high as models discovered by the standard eST-Miner, since it discovers less fitting places. Our experiments confirm this expectation for small values of τ, where the eST-Miner discovers a lot more fitting places than the ip-eST variant using its high λ-value for aggressive pruning. For high τ values, we get reversed results. This can be explained by the few fitting places skipped by the ip-eST variant resulting in significantly less places being removed during post-processing. This phenomenon is illustrated in Fig. 5.

In summary, our experiments have shown a strong boost to the time performance compared to the standard eST-Miner, while we observe only small differences in the quality of discovered models. We conclude that the ip-eST variant seems to restrict the search space in an effective and adequate way and can reliably discover models of similar quality as returned by the eST-Miner in a small fraction of the time.

7 Conclusion

In this paper, we introduced two approaches to improve the time performance of the eST-Miner. The first strategy is based on arranging the candidate places in the tree-like search structure in such a way, that the amount of skipped unfitting candidates is approximately maximized. This decreases the time needed for the search phase to $60 - 80\%$ of the time needed with random ordering, while returning exactly the same set of fitting places. The second strategy is based on heuristically classifying subtrees as uninteresting and thus skippable based on easily computable log properties. This thus not only greatly decreases runtime of the search phase, but also of the post-processing step, since significantly less fitting places are discovered. Our experiments show that most of the skipped fitting places seem to be implicit, since the quality of the models remains comparable. The computation time of the search phase is pushed below 5%

of the time needed by the original algorithm. The post-processing step takes less than 0.5%. Moreover, both approaches can be combined.

Compared with many other discovery algorithms, the eST-Miner returns models with high scores in fitness and particular precision. On the downside, computation times are high and the models often complex. The introduced strategies offer a significant decrease in computation time and make the eST-Miner a competitive discovery approach on event logs with a small number of activities.

The presented approaches are clearly very promising with respect to discovering high-quality models faster. We see potential to further improve time performance as well as model quality by investigating additional variants of both our heuristics. Also, discovering reasonable dependencies between the parameters λ and τ would be interesting. A combination of these heuristics with other variants of eST-Miner, e.g. the uniwired variant ([14]), is clearly possible. In addition to improved computation speed, we may be able to use our approaches to simplify the returned models without loosing important structures.

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

References

1. Wen, L., van der Aalst, W.M.P., 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.M.P.: Discovering block-structured process models from event logs - a constructive approach. Application and Theory of Petri Nets and Concurrency (2013)
3. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Polyvyanyy, A.: Split miner: automated discovery of accurate and simple business process models from event logs. Knowledge and Information Systems (2018)
4. Badouel, E., Bernardinello, L., Darondeau, P.: Petri Net Synthesis. Text in Theoretical Computer Science, an EATCS Series. Springer (2015)
5. 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)
6. 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)
7. Carmona, J., Cortadella, J., Kishinevsky, M.: A Region-Based Algorithm for Discovering Petri Nets from Event Logs. In: Business Process Management (BPM 2008)
8. Mannel, L.L., van der Aalst, W.M.P.: Finding complex process-structures by exploiting the token-game. In: S. Donatelli, S. Haar (eds.) Application and Theory of Petri Nets and Concurrency, pp. 258–278. Springer International Publishing (2019)
9. van der Aalst, W.M.P.: 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)
10. Mannhardt, F.: Sepsis cases - event log (2016)
11. De Leoni, M., Mannhardt, F.: Road traffic fine management process (2015)
12. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer (2016)
13. Munoz-Gama, J., Carmona, J.: A fresh look at precision in process conformance. In: BPM (2010)
14. Mannel, L.L., van der Aalst, W.M.P.: Finding uniwired Petri nets using eST-Miner. In: C. Di Francescomarino, R. Dijkman, U. Zdun (eds.) Business Process Management Workshops, pp. 224–237. Springer International Publishing (2019)