Improving Process Discovery Using Translucent Activity Relationships [⋆]

Harry H. Beyel[0000−0002−6541−3848] and
Wil M.P. van der Aalst[0000−0002−0955−6940]

Chair of Process and Data Science, RWTH Aachen University, Aachen, Germany
{beyel, wvdaalst}@pads.rwth-aachen.de

Abstract. In task mining, data captured by recording user interactions in a desktop environment are stored in a user interaction log. A user interaction log can be transformed into an event log, suitable for process-mining techniques. If the desktop is captured during the recording process, information about *enabled* activities is available (next to the executed activity). This information can be extracted and added to an event log, resulting in a so-called *translucent event log*. A translucent event log can also be extracted from running information systems or created by applying domain knowledge to existing non-translucent event logs. Information about enabled activities is valuable for multiple reasons. For example, one may use this information to discover process models that capture the underlying behavior better. However, until now, only limited work has been done on exploiting the information on enabled activities for process discovery. In this work, we introduce translucent activity relationships derived from translucent event logs. These relationships can be embedded in various discovery algorithms. In this work, we focus on the Inductive Miner. In this process, we derive a translucent directly-follows graph. Based on this graph, we extend the Inductive Miner to use translucent activity relationships. We create three different variants and evaluate them on multiple translucent event logs by comparing them with the Inductive Miner. Based on the evaluation, we show that considering these relationships, fewer recordings are needed to discover a similar, if not even a superior, process model.

Keywords: Process Discovery · Translucent Event Log · Translucent Directly-Follows Graph · Translucent Inductive Miner.

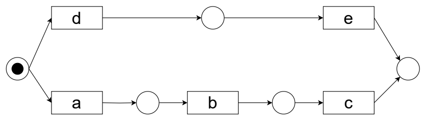
1 Introduction

Robotic Process Automation (RPA) aims to automate repetitive tasks in a desktop application environment by using software bots [31]. Traditionally, cost- and time-intensive methods, such as interviews, are conducted to discover routines that can be automatized. To lower boundaries for identifying routines, *Robotic Process Mining* (RPM) [22] can be used, which is a subfield of *task mining* [16].

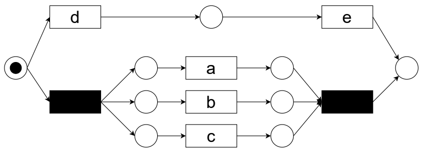
[⋆] We thank the Alexander von Humboldt (AvH) Stiftung for supporting our research.

Table 1: Example translucent event log.

Event	Case	Activity	Enabled Activities	Timestamp
e_1	1	a	a, b, c, d	2023-06-20 13:37:37
e_2	1	b	b, c	2023-06-20 13:37:38
e_3	1	c	c	2023-06-20 13:37:39
e_4	2	d	a, b, c, d	2023-06-20 13:37:40
e_5	2	e	e	2023-06-20 13:37:41



(a) Petri net discovered by the IM [19].



(b) Petri net discovered by the IMto.

Fig. 1: Petri nets discovered from the translucent event log shown in Table 1.

RPM performs its analysis on a *user interaction log.* These logs are created by capturing a user's desktop environment. Each entry in such a log represents an event, for instance, clicking a button. Such user interaction logs can be converted into *event logs*, used as input for *process mining* [1]. An event log is a collection of events, and each event consists of at least three attributes: a case identifier, an activity, and a timestamp. If screenshots linked to events are available, information on enabled activities, besides the executed activity, can be extracted from interaction logs, as shown in [13]. Nonetheless, adding this information using domain knowledge or extracting it from running information systems is also possible. If the information on enabled activities is included in an event log, we call it a *translucent event log.* The information on enabled activities is valuable. Dedicated process-mining techniques can be applied to these event logs [2,14].

To illustrate the value of enabled activities in process discovery, consider the translucent event log shown in Table 1. By considering only the information on enabled activities, we observe that there are two cases $\langle \mathrm{a}, \mathrm{b}, \mathrm{c} \rangle$ and $\langle \mathrm{d}, \mathrm{e} \rangle$. The first executed activity determines which trace is taken, and the behavior is sequential. Applying the Inductive Miner (IM) [19] on the translucent event log results in the Petri net shown in Figure 1a. However, when we consider the information on enabled activities, we denote that activities a, b, and c are initially enabled and are still enabled after executing each after another. This indicates that these activities are parallel to each other. As we show in the remainder of this work, we can utilize the information on enabled activities, leading to the Petri net depicted in Figure 1b. Normally, more variants are needed to discover the same model using the IM [19]. However, this example shows that less information is needed to discover a well-describing process model using our methods. This is beneficial when analyzing behavior in desktop environments to create models that are used to create bots for RPA.

So far, only one process-discovery algorithm has been described that deals with information on enabled activities [2]. This method is based on state-based-region techniques and is not applicable to larger real-life datasets. To overcome the limitations of the current translucent process-discovery technique, we introduce *translucent activity relationships*. The relationships can be used in existing process-discovery algorithms, thus allowing more information to be considered to create better process models even if not all variants of a process are recorded. To demonstrate the value of translucent activity relationships in process discovery and to evaluate our approach, we implemented translucent relationships in the IM [19] provided in PM4Py [12]. We created three different variants based on the original IM [19] that use translucent relationships at different points in the discovery algorithm. Our evaluation covers artificial and real-life scenarios. Using the IM [19], we demonstrate how translucent relationships enhance existing process-discovery techniques. Furthermore, we show that we need fewer variants of a process to discover better process models. This showcases the high value of information on enabled activities and stresses their importance in creating bots based on process models for RPA.

The remainder of this work is structured as follows. In Section 2, we present related work. Subsequently, we introduce preliminaries of our work in Section 3. In Section 4, we define and explain translucent activity relationships. To show the value of the relationships, we embed them at various points in the IM as we show in Section 5. We evaluate our translucent discovery algorithms inspired by the IM in Section 6 and compare the results with the results of the original IM. Finally, we provide a conclusion on our work in Section 7.

2 Related Work

Process discovery deals with discovering a comprehensive process model that represents the underlying behavior in a given event log [1]. Multiple process-discovery techniques do this, but all consider the relationships between the activities that are recorded in a given event log. The Alpha algorithm [6] is the pioneering method for concurrent process model discovery, utilizing the event log's footprint matrix. Similar to the Alpha algorithm, we introduce activity relationships that are beneficial for process discovery and enable translucent process discovery in a great manner. Another process-discovery technique is the IM and its different variants [19,20,21]. A core concept of the IM is the usage of directly-follows graphs (DFGs). The IM applies different cuts on a provided DFG, thus creating multiple partitions, and based on these, sublogs are created. For each partition, a DFG is created on which the method is recursively applied. A different technique that considers a DFG as input is the so-called Split Miner [10,11]. The Split Miner consists of six steps: First, a DFG is constructed, and self-loops and short-loops are identified. Second, concurrency relations are discovered using the DFG. Third, the DFG is filtered by trying to balance fitness and precision of the final process model. In the fourth and fifth steps, split and join gateways are discovered. Finally, OR-joins may be turned into either an XOR-gateway or AND-gateways. We refer to [1,3,9] for more information on various process-discovery techniques.

Nevertheless, all the previously mentioned techniques only consider information based on executed activities; no information on enabled activities is used. To our knowledge, only one process-discovery technique uses information on enabled activities. The indirect process-discovery technique is descibed in [2] and is related to region theory [17,18] and state-based regions [5,15,28,29,30]. In general, state-based region process-discovery techniques are a two-step approach. First, a transition system is created. Second, minimal regions are extracted from the created transition system. Each minimal region corresponds to a place of a Petri net. In the only existing translucent process-discovery technique, each set of enabled activities represents its own state. Arcs connecting different states are labeled with the executed activity that leads from one set to the other, as denoted in the given translucent event log. As the last step, such a system is transformed into a Petri net. However, since this approach is based on state-based regions and, therefore, region-based process discovery, it may have similar limitations, such as the computation time or a too-large model. We propose a more practical approach by defining translucent activity relationships that can be implemented in existing process-discovery techniques. Furthermore, we show how translucent activity relationships can be used in DFG-based approaches. As a result, we enhance widely known and embedded techniques to work with more information.

3 Preliminaries

Given a set X, a sequence $\sigma \in X^*$ of length $n \in \mathbb{N}$ assigns an enumeration to elements of the set, i.e., $\sigma : \{1, ..., n\} \to X$. We denote this with $\sigma = \langle \sigma_1, \dots, \sigma_n \rangle$. $\langle\rangle$ is the empty sequence. $\sigma\restriction_{X'}$ is the projection of σ onto a set X', e.g., $\langle a, b, c, d, a, b\rangle\restriction_{\{a,b\}} = \langle a, b, a, b\rangle$. In the remainder, we refer with σ_i to the sequence's i-th element. Given a sequence σ, $|\sigma|$ denotes the length of a sequence. Given two sequences $\sigma = \langle \sigma_1, \dots, \sigma_{|\sigma|}\rangle$ and $\sigma' = \langle \sigma'_1, \dots, \sigma'_{|\sigma'|}\rangle$, a conjunction is denoted as $\sigma \cdot \sigma' = \langle \sigma_1, \dots, \sigma_{|\sigma|}, \sigma'_1, \dots, \sigma'_{|\sigma'|}\rangle$. We denote the Cartesian product of two sets X and Y as $X \times Y = \{(x, y) \mid x \in X \ \wedge \ y \in Y\}$.

Our work focuses on developing process-discovery techniques that consider translucent event logs, i.e., for each event, information on enabled activities besides the executed activity is available. Based on the characteristics of enabled activities, the executed activity must also be enabled in the corresponding event. $\mathcal{U}_{case}$ is the universe of case identifiers, $\mathcal{U}_{act}$ is the universe of activity names, and $\mathcal{U}_{time}$ is the universe of timestamps.

Definition 1 (Translucent Event Log). *$\mathcal{U}_{ev}$ is the universe of events. $e \in \mathcal{U}_{ev}$ is an event, $\pi_{case}(e) \in \mathcal{U}_{case}$ is the case of e, $\pi_{time}(e) \in \mathcal{U}_{time}$ is the time of e, $\pi_{en}(e) \subseteq \mathcal{U}_{act}$, and $\pi_{act}(e) \in \pi_{en}(e)$ is the activity of e. $\mathcal{U}_{trace} = \mathcal{U}_{ev}^*$ is the universe of traces. $L \subseteq \mathcal{U}_{trace}$ is an event log s.t. for all $\langle e_1, \dots, e_n\rangle \in L$, $\pi_{time}(e_1) <$*

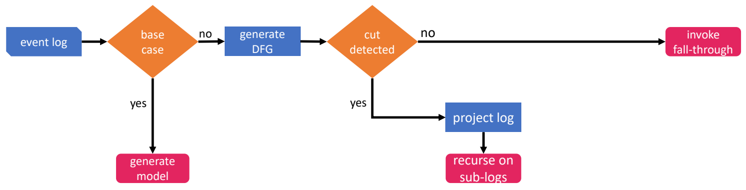


Fig. 3: Abstract overview of the Inductive Miner [19].

$\cdots < \pi_{time}(e_n)$, *and, for* $i, j \in \{1, \ldots, n\}$, $\pi_{case}(e_i) = \pi_{case}(e_j)$. *Also, for* $i, j \in \{1, \ldots, |\sigma|\}$, $e_i \neq e_j$. *In addition, for all* $\sigma, \sigma' \in L$, $\sigma \neq \sigma'$, $(\bigcup_{e \in \sigma}\{e\}) \cap (\bigcup_{e \in \sigma'}\{e\}) = \emptyset$. *For simplicity, we define* $\pi_{act}(L) = \bigcup_{\sigma \in L} \bigcup_{e \in \sigma}\{\pi_{act}(e)\}$

Let $L_{ex} = \{\langle e_1, e_2, e_3\rangle, \langle e_4, e_5\rangle\}$ be the translucent event log shown in Table 1. Then, $\pi_{act}(e_1) = a$, $\pi_{en}(e_1) = \{a, b, c, d\}$, $\pi_{act}(e_1) \in \pi_{en}(e_1)$, $\pi_{time}(e_1) =$ 2023-05-04 13:37:37, and $\pi_{case}(e_1) = 1$. In addition, $\pi_{act}(L_{ex}) = \{a, b, c, d, e\}$.

Our work extends the IM [19] to consider information on enabled activities. Therefore, we introduce directly-follows graphs and shortly explain the IM. A directly-follows graph is derived from a (translucent) event log and shows three aspects: start activities, end activities, and directly-follows relationships between activities recorded in the provided event log. Directly-follows relationships show which activities directly-follows an activity over all cases in an event log.

Definition 2 (Directly-Follows Graph). *Let* $L \subseteq \mathcal{U}_{ev}$ *be a (translucent) event log with* $\blacksquare \notin \pi_{act}(L)$ *and* $\blacktriangleright \notin \pi_{act}(L)$. *A Directly-Follows Graph (DFG) of* L *is a directed graph* $DFG^L = (V, E)$ *for which* $V = \pi_{act}(L) \cup \{\blacktriangleright, \blacksquare\}$, *where* $\blacktriangleright$ *is the start node,* $\blacksquare$ *is the end node, and* $E = (\{\blacktriangleright\} \times \bigcup_{\sigma \in L}\{\pi_{act}(\sigma_1)\}) \cup (\bigcup_{\sigma \in L}\{\pi_{act}(\sigma_{|\sigma|})\} \times \{\blacksquare\}) \cup (\bigcup_{\sigma \in L} \bigcup_{i \in \{1, \ldots, |\sigma|-1\}}\{(\pi_{act}(\sigma_i), \pi_{act}(\sigma_{i+1}))\})$.

The DFG of the event log shown in Table 1 is provided in Figure 2. As denoted in the DFG, activities a and d are start activities, and c and e are end activities. Furthermore, a is connected to b, and b is connected to c.

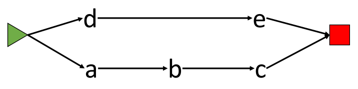


Fig. 2: DFG obtained from the event log depicted in Table 1.

The IM [19] is a well-known state-of-the-art process-discovery algorithm. An abstract overview of the IM is provided in Figure 3. The IM [19] works as follows. If the provided event log is a base case, a model is generated. If not, a DFG is generated. It is checked whether a cut on the graph is detected. There are for cuts: *exclusive-choice* ($\times$), *sequence* ($\rightarrow$), *parallel* ($+$), and *redo-loop* ($\circlearrowleft$). The cuts consider multiple arcs in the corresponding DFG and capture the relationships between various activities. If a cut is detected, the log is projected by considering the created activity partitions. Each activity is only part of exactly one partition. The final partition captures a single activity, resulting in a base case. If no cut is found, fall-throughs are applied. For more information, we refer to [19]. In the following, we define a cut.

Definition 3 (Cut). *Let $L \subseteq \mathcal{U}_{trace}$ be a translucent event log. A cut $(\oplus, A_1, \ldots, A_n)$ of L is a tuple of a control flow operator $\oplus \in \{\times, \rightarrow, +, \circlearrowleft\}$ and a partitioning of activities into $n \geq 2$ subsets, i.e., $\pi_{act}(L) = \bigcup_{i \in \{1,\ldots,n\}} A_i$ and $A_i \cap A_j = \emptyset$ for $1 \leq i < j \leq n$.*

In this work, we enhance DFGs to capture translucent information and, as a result, enable the IM [19] to consider information on enabled activities. In the remainder of this work, we show Petri nets as process model representations. An introduction to Petri nets is provided in [1,27].

4 Translucent Activity Relationships

Translucent activity relationships are a key concept in incorporating information on enabled activities in process mining. To discover translucent activity relationships, we use a sequence-based approach. This means that we walk through the cases of the provided translucent event log and compute the activity relationships by considering the executed activity of an event and the event's enabled activities as well as the successor event's enabled activities. We define three translucent activity relationships: *directly-follows*, *parallel*, and *exclusive-choice* relationships. We discuss their meaning and use for each of them. Furthermore, we introduce translucent start and end activities. To illustrate our approach, we refer to the example translucent event log provided in Table 1 that we denote in the following as L_{ex}.

4.1 Translucent Directly-Follows Relationship

The translucent directly-follows relationship is a natural extension of the traditional directly-follows relationship. Note that this relationship considers all enabled activities after an activity's execution, i.e., self-loops are considered, even if they do not appear in the sequences of executed activities.

Definition 4 (Translucent Directly-Follows Relationship). *Let $L \subseteq \mathcal{U}_{ev}$ be a translucent event log and $act \in \pi_{act}(L)$ be an activity. act has a translucent directly-follows relationship to all activities that are enabled after its execution and executed in L. The following defines the set of activities for an activity for which the conditions hold, $df^L(act) = \{en \in \pi_{en}(\sigma_{i+1}) \mid \sigma \in L \wedge i \in \{1, ..., |\sigma| - 1\} \wedge \pi_{act}(\sigma_i) = act\} \cap \pi_{act}(L)$.*

For L_{ex}, we denote the following translucent directly-follows relationships: $df^{L_{ex}}(a) = \{b, c\}$, $df^{L_{ex}}(b) = \{c\}$, $df^{L_{ex}}(c) = \emptyset$, $df^{L_{ex}}(d) = \{e\}$, and $df^{L_{ex}}(e) = \emptyset$.

4.2 Translucent Parallel Relationship

The translucent parallel relationship extends the previously mentioned translucent directly-follows relationship. Using the information on the executed activity of an event, the event's enabled activities, and the event's succeeding enabled activities, we conclude which activities are parallel to the executed activity. The parallel activities of the executed activity are the ones that were enabled during and after its execution.

Definition 5 (Translucent Parallel Relationship). *Let $L \subseteq \mathcal{U}_{ev}$ be a translucent event log and $act \in \pi_{act}(L)$ be an activity. act has a translucent parallel relationship to all activities that are enabled in its execution and are still enabled after its execution and are executed in L. The following defines the set of activities for an activity for which the conditions hold, $par^L(act) = \{en \in (\pi_{en}(\sigma_i) \cap \pi_{en}(\sigma_{i+1})) \setminus \{act\} \mid \sigma \in L \ \wedge \ i \in \{1, ..., |\sigma| - 1\} \ \wedge \ \pi_{act}(\sigma_i) = act\} \cap \pi_{act}(L)$.*

For L_{ex}, we denote the following translucent parallel relationships: $par^{L_{ex}}(a) = \{b, c\}$, $par^{L_{ex}}(b) = \{c\}$, $par^{L_{ex}}(c) = \emptyset$, $par^{L_{ex}}(d) = \emptyset$, and $par^{L_{ex}}(e) = \emptyset$.

4.3 Translucent Exclusive-Choice Relationship

The idea of the translucent exclusive-choice relationship is similar to translucent parallel relationships. Nevertheless, instead of focusing on "remaining" activities, we concentrate on "removed" activities. To discover "removed" activities, we use the information on the executed activity of an event and the difference between the event's enabled activities and the event's succeeding enabled activities.

Definition 6 (Translucent Exclusive-Choice Relationship). *Let $L \subseteq \mathcal{U}_{ev}$ be a translucent event log and $act \in \pi_{act}(L)$ be an activity. act has a translucent exclusive-choice relationship to all activities that are not enabled anymore after its execution and are executed in L. The following defines the set of activities for an activity for which the conditions hold, $exc^L(act) = \{en \in (\pi_{en}(\sigma_i) \setminus \pi_{en}(\sigma_{i+1})) \setminus \{act\} \mid \sigma \in L \ \wedge \ i \in \{1, ..., |\sigma| - 1\} \ \wedge \ \pi_{act}(\sigma_i) = act\} \cap \pi_{act}(L)$.*

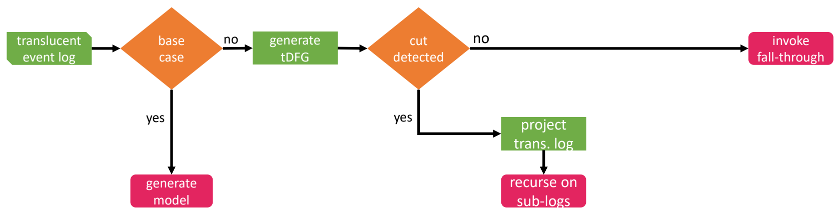
For L_{ex}, we denote the following translucent exclusive-choice relationships: $exc^{L_{ex}}(a) = \{d\}$, $exc^{L_{ex}}(b) = \emptyset$, $exc^{L_{ex}}(c) = \emptyset$, $exc^{L_{ex}}(d) = \{a, b, c\}$, and $exc^{L_{ex}}(e) = \emptyset$.

4.4 Translucent Start and End Activities

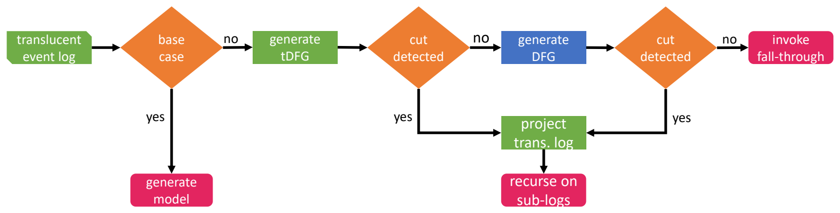
In addition to the previously defined translucent activity relationships, we introduce translucent start and end activities.

Translucent start activities are all enabled activities in traces' first events and executed at some point in the log.

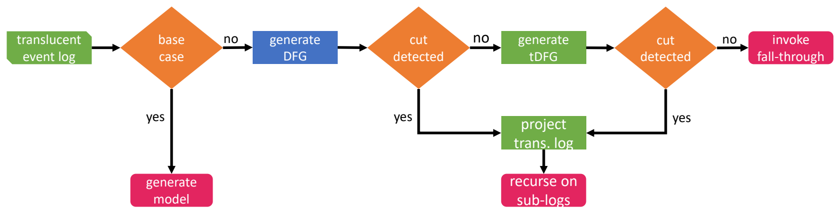
Definition 7 (Translucent Start Activities). *Let $L \subseteq \mathcal{U}_{ev}$ be a translucent event log. Translucent start activities are activities that are enabled at the start of traces, i.e., $Start^L = \bigcup_{\sigma \in L} \pi_{en}(\sigma_1) \cap \pi_{act}(L)$.*



(a) IM only using tDFGs (IMto).



(b) IM first using tDFG, then DFG (IMtf).



(c) IM first using DFG, then tDFG (IMts).

Fig. 4: Abstract overview of the different approaches.

For L_{ex}, we denote $Start^{L_{ex}} = \{a, b, c, d\}$ as translucent start activities. In the classic setting, only a and d are start activities.

Similar to translucent start activities, translucent end activities are all enabled activities of traces' last events of a translucent event log. Also, they have to be executed at some point in the log.

Definition 8 (Translucent End Activities). *Let $L \subseteq \mathcal{U}_{ev}$ be a translucent event log. Translucent end activities are activities that are enabled at the end of traces and parallel to the executed activity, i.e., $End^L = \bigcup_{\sigma \in L} \pi_{en}(\sigma_{|\sigma|}) \cap \pi_{act}(L)$.*

For L_{ex}, we denote the following translucent end activities: $End^{L_{ex}} = \{c, e\}$. This is similar to the classic setting.

5 Translucent Inductive Miner

After defining translucent activity relationships, we utilize them in process discovery. Translucent activity relationships are a general concept usable in various process-discovery algorithms. Despite their natural link to the Alpha algorithm [6], we show how to use them in the state-of-the-art discovery technique, the IM

[19]. We choose this algorithm based on its guarantees, such as soundness. Moreover, the IM [19] is fast compared to other methods, e.g., state-based regions. We present three approaches, depicted in Figure 4, to incorporate the translucent information. Note that the base case and cut detection work as defined by the IM [19]. The approach shown in Figure 4a replaces the DFG with a tDFG; the rest stays the same. We call this approach *IMto*. The approach shown in Figure 4b first uses a tDFG, but if no cut is detected, a DFG is constructed. We call this approach *IMtf*. The approach shown in Figure 4c first used a DFG, but if no cut is found, a tDFG is constructed on which cuts are searched. We call this approach *IMts*. Fall-throughs are applied as defined in the IM [19]. In the following, we focus on two important aspects. First, how to create a translucent DFG. Second, how logs are projected.

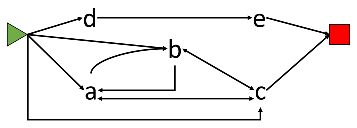


Fig. 5: tDFG obtained from the translucent event log depicted in Table 1.

5.1 Translucent DFG

Similar to DFGs (see Definition 2), translucent DFGs are constructed using activity nodes. However, to connect activity nodes, translucent information is used, i.e., translucent start and end activities, and translucent directly-follows and parallel relationships. We do not use translucent exclusive-choice relationships since they do not add or delete arcs, but they can be valuable for other techniques. Concerning translucent directly-follows relationships, we add an arc from each activity to every activity in the corresponding directly-follows set. Concerning translucent parallel relationships, we exploit the symmetry of parallel relationships, i.e., if an activity a is parallel to activity b, then b is parallel to a. Thus, we create two arcs, one for each direction between activities in a parallel relationship.

Definition 9 (Translucent Directly-Follows Graph). *Let $L \subseteq \mathcal{U}_{ev}$ be a translucent event log with $\blacksquare \notin \pi_{act}(L)$ and $\blacktriangleright \notin \pi_{act}(L)$. A translucent Directly-Follows Graph (tDFG) of L is a directed graph $tDFG^L = (V, E)$ for which $V = \pi_{act}(L) \cup \{\blacktriangleright, \blacksquare\}$ and $E = (\{\blacktriangleright\} \times Start^L) \cup (End^L \times \{\blacksquare\}) \cup (\bigcup_{act \in \pi_{act}(L)} \{act\} \times df^L(act)) \cup (\bigcup_{act \in \pi_{act}(L)} par^L(act) \times \{act\}) \cup (\bigcup_{act \in \pi_{act}(L)} \{act\} \times par^L(act))$. $\blacktriangleright$ is the start node, $\blacksquare$ is the end node.*

The tDFG of the event log shown in Table 1 is provided in Figure 5. As we observe in Figure 2, the DFG shows only sequential relationships and a choice between starting with either a or d, resulting in different paths. However, the tDFG depicted in Figure 5 reveals parallelism between a, b, and c, more start activities, but the same end activities as the DFG.

5.2 Log Splitting

The cuts of the IM [19] find disjoint sets of activities. The sets are used to split an event log into sub-logs. However, the splitting is only defined on event logs, not translucent event logs. Therefore, we have to create projected translucent sub-logs in which only events are included for which the executed activities fit the given set of activities. In the following, we list the definitions for splitting translucent event logs. These are based on the pre-existing definition [19].

Definition 10 (Exclusive-Choice Split). *Let $L \subseteq \mathcal{U}_{trace}$ be a translucent event log and $(\times, A_1, \ldots, A_n)$ be an exclusive-choice cut. Then, the exclusive-choice split is for $i \in \{1, \ldots, n\}$ defined as $L_i = \{\sigma \in L \mid \forall_{e \in \sigma} \pi_{act}(e) \in A_i\}$*

Definition 11 (Sequence Split). *Let $L \subseteq \mathcal{U}_{trace}$ be a translucent event log and $(\rightarrow, A_1, \ldots, A_n)$ be a sequence cut. Then, the sequence split is for $j \in \{1, \ldots, n\}$ defined as $L_j = \{\sigma^j \in L \mid \sigma^1 \cdot \sigma^2 \cdot \ldots \cdot \sigma^n \in L \wedge i \leq n \wedge \bigcup_{k \in \{1, \ldots, |\sigma^i|\}} \{\pi_{act}(\sigma^i_k)\} \subseteq A_i\}$*

Definition 12 (Concurrency Split). *Let $L \subseteq \mathcal{U}_{trace}$ be a translucent event log and $(\times, A_1, \ldots, A_n)$ be a concurrency cut. Then, the concurrency split is for $i \in \{1, \ldots, n\}$ defined as $L_i = \{\sigma\restriction_{A_i} \mid \sigma \in L\}$*

Definition 13 (Redo-Loop Split). *Let $L \subseteq \mathcal{U}_{trace}$ be a translucent event log and $(\circlearrowleft, A_1, \ldots, A_n)$ be a redo-loop cut. Then, the redo-loop split is for $i \in \{1, \ldots, n\}$ defined as $L_i = \{\sigma^2 \mid \sigma^1 \cdot \sigma^2 \cdot \sigma^3 \in L \wedge \bigcup_{j \in \{1, \ldots, |\sigma^2|\}} \{\pi_{act}(\sigma^2_j)\} \subseteq A_i \wedge (\sigma^1 = \langle\rangle \vee (\pi_{act}(\sigma^1_{|\sigma^1|}) \notin A_i)) \wedge (\sigma^3 = \langle\rangle \vee (\pi_{act}(\sigma^3_1) \notin A_i))\}$*

Note that the nodes of the tDFG consider only executed, not enabled, activities. Also, note that the translucent activity relationships are only defined on executed activities. Hence, activities that are not executed in a sub-log but are only being enabled do not cause issues. Applying the IM [19], IMtf, and IMts on the translucent event log contained in Table 1 results in the Petri net depicted in Figure 1a. Applying the IMto on the same translucent event log returns the Petri net depicted in Figure 1b. As we observe, the latter captures the behavior that we encountered in the tDFG and in the underlying translucent event log.

6 Evaluation

In this section, we evaluate our different approaches. As introduced in [13], enabled activities can be extracted from screenshots using template matching. However, to evaluate different scenarios and increase data availability and reproducibility, we create artificial translucent event logs. The method is similar to the approach in [13]. An overview is depicted in Figure 6. As shown, a process model is discovered from an existing event log. The process model and the event log are aligned, and by only considering fitting traces, we add the information on enabled activities by considering the replay state in the model. For details, we refer to [13].

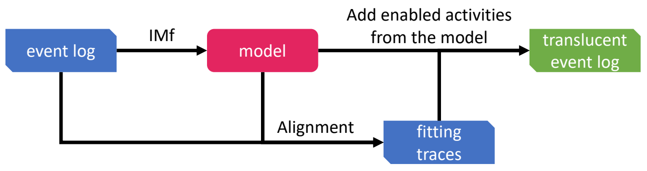


Fig. 6: Overview of creating translucent event logs.

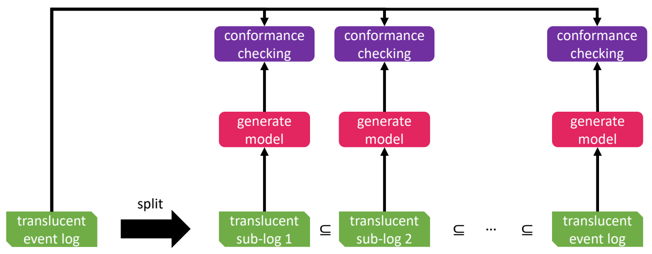


Fig. 7: Overview of our evaluation concept.

To evaluate whether translucent activity relationships provide value for process discovery, we compare the IM that does not use information on enabled activities with our three versions. Also, we evaluate e whether our approaches discover similar or better models with fewer variants than the IM [19]. To evaluate these aspects, we need to create such scenarios. An overview of how we create the evaluation scenarios is depicted in Figure 7. We split a translucent event log into multiple sublogs, each representing a certain number of variants. For each sublog, the IM [19] and our approaches discover process models. To check whether a more representative process model is discovered with less information, we compute fitness, precision, and F1 scores using the unsplit translucent event log. We use alignments with a standard cost function to compute fitness scores [7]. To compute precision scores, we use a modified approach of the escaping arcs approach, presented in [4,8,26]. Instead of only focusing on arcs representing behavior enabled in the log by considering executed activities and arcs based on behavior allowed in the model, we also consider the enabled activities as log behavior. Only fitting traces are considered. More information can be found in [14]. The F1 score is the harmonic mean between fitness and precision.

We implemented our approaches in Python using PM4Py [12][1]. We evaluate our process-discovery technique in two scenarios. First, an artificial scenario with a focus on parallelism. Second, three real-life scenarios.

[1] Our code, as well as the data, are provided here: `https://github.com/hherbertb/TranslucentActivityRelationships`

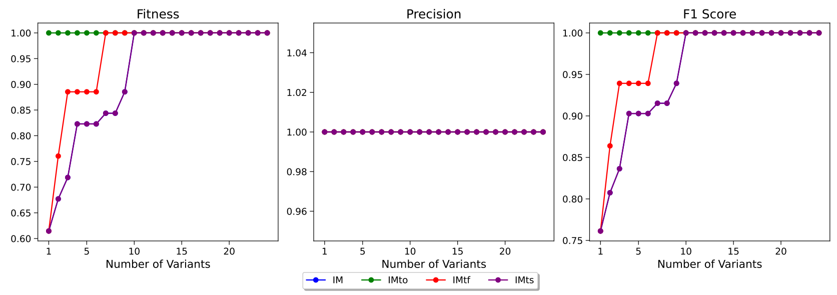


Fig. 8: Score comparison between the IM [19] and the different translucent variants using an artificial parallel scenario.

6.1 Artificial Scenario

In the evaluation based on an artificial parallelism scenario, we want to evaluate how fitting the discovered process models from the IM [19] and our approaches are. Since our various algorithms use translucent parallel relationships, we want to ratify their usefulness in process discovery. Besides parallelism, we want to check whether our approaches discover as least as good models as the IM [19].

To evaluate the usefulness of parallel translucent relationships, we created a scenario of four activities. Each activity has to be executed once per variant and in different orders, resulting in $4! = 24$ variants. We created 24 sublogs, each covering the first variant. The other variants in a sub-log are added in incremental order, i.e., the first sub-log consists of variant one, the second sub-log of variants one and two, the third sub-log of variants one, two, and three, etc. Each variant appears exactly once. For each sub-log, we discover four process models, one per algorithm. For each model, we compute its fitness, precision, and F1 score by considering the unsplit translucent event log. The results of the parallel scenario are shown in Figure 8. The IM and IMts share the same fitness values. The IMto has a perfect fitness score. The IMtf achieves a higher fitness score with fewer variants than the IM and IMts. When considering three variants, the fitness score for the IMtf is more than 0.1 higher than for the IM and IMts. In addition, only seven variants are needed to achieve a perfect fitness score; the IM and IMts need ten variants, nearly 42% of behavior. The interpretation of the F1 scores is the same as for fitness due to the same precision for all settings.

6.2 Real-life Scenario

To evaluate the performance of our approaches in comparison to the IM [19] in a real-life scenario, we use various event logs. An overview of the event logs is provided in Table 2. As shown in Figure 6 and discussed earlier, we preprocess the logs using process models, which we discover for each log. We use a Petri net discovered by the IM infrequent [20] using a 40% noise threshold. We do not use the IM [19] since this algorithm produces models that lead to translucent

Table 2: Overview of the evaluation event logs.

	Event Log		Translucent Event Log	
Name of the event log	# traces	#variants	#traces	#variants
Road Traffic Fine Management [23]	150,370	231	18,981	35
Sepsis [24]	1050	846	20	17
Hospital Billing [25]	100,000	1020	146,748	52

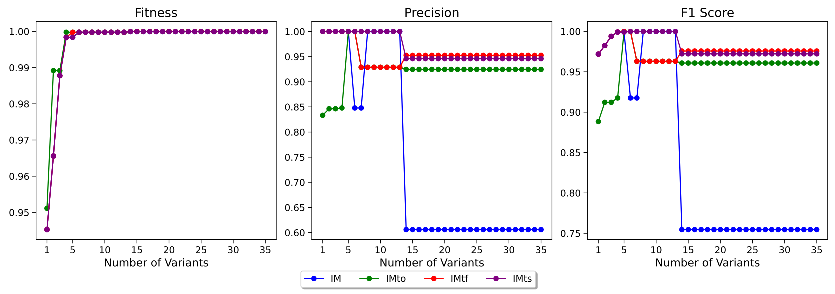
event logs with a high amount of enabled activities per event. We selected this noise threshold because the preprocessing required to create synthetic translucent event logs would be too time-consuming with a lower value. Moreover, a smaller threshold provided no value to our insights. As done in the artificial scenario, we consider only variants of the translucent event log. The first sub-log contains all traces of the most frequent variant, and the second sub-log contains all traces of the most and second most frequent variant, etc. Again, we compute the scores by using the unsplit translucent event log. Our result is depicted in Figure 9. In the following, we focus on the results for each log and provide a general statement at the end.

Concerning the road traffic fine management log [23] (see Figure 9a), we denote that the IMto achieves a higher fitness score earlier than the other approaches, followed by the IMtf. IM and IMts perform equally in terms of fitness. After six variants, the fitness value for all models across the different approaches is the same. Concerning precision, we observe more variation. The IMto performs worst when only a limited number of variants are considered. There is a drop for the IM when six or seven variants are included, caused by fall-throughs. When 14 variants are considered, a significant drop in precision is denoted for the IM, from 1 to roughly 0.61. The reason for this is the fall-throughs taking place after that point. Also, we denote that the IMtf and IMts perform similarly while the IMto performs slightly worse. The F1 score shows similar insights due to the heavy influence of precision. In Figure 10, two process models are depicted. Based on the translucent event log containing 14 variants, the IM and IMts discovered these models. While the model discovered by the IMts (Figure 10b) contains sequential behavior, the model discovered by the IM (Figure 10a) has a lot of parallel behavior. This is the reason for the lower precision score.

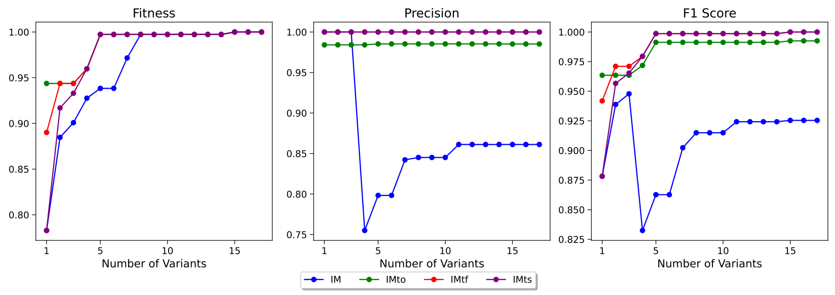
Concerning the sepsis log [24] (see Figure 9b), we observe that considering translucent variants boosts the fitness again from the start on. After considering eight variants, the IM achieves the same fitness. As before, the IMto starts with the highest fitness value, followed by the IMtf. The IMts performs better than the IM. Regarding precision, the IMtf and IMts perform equally with a score of 1. The IMto also provides a constant score of roughly 0.98. When including at least four variants, the precision score drops significantly and is never again close to the other approaches. When considering the F1 score, we observe that the IM performs the worst. IMtf and IMts perform equally after four variants. The IMto is the best choice when only one variant is available.

Concerning the hospital billing log [25] (see Figure 9c), we observe that the fitness values are the same for all approaches. Thus, the F1 score differences are only caused by precision. The IMts performs like the IM, which is precise most of the time. The IMto and IMtf perform similarly. However, after inducing variant 24, the score for the IMtf increases to 1, while the precision of the IMto drops.

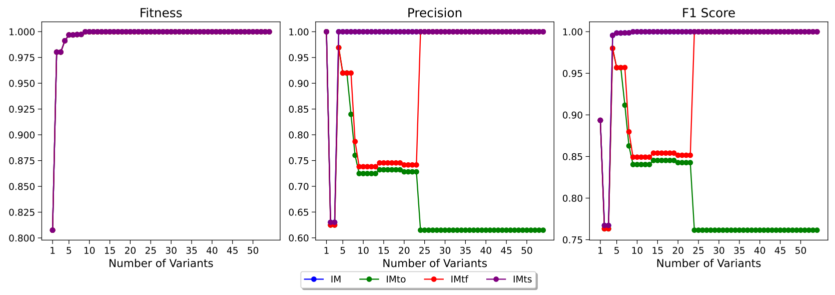
In general, including translucent relationships boosts performance, especially precision. Relying only on translucent relationships may increase fitness, but when only a limited number of variants are available, the precision score is lower.



(a) Results for the road traffic fine management log [23].

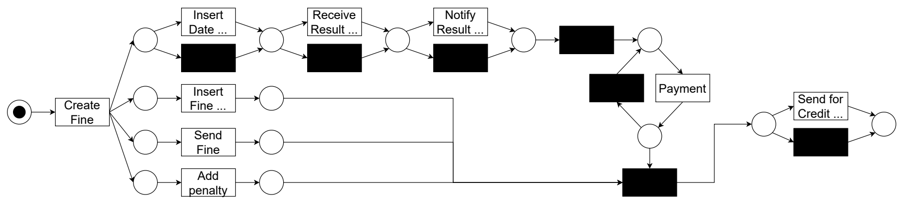


(b) Results for the sepsis log [24].

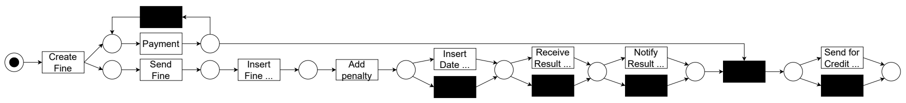


(c) Results for the hospital billing log [25].

Fig. 9: Results of different translucent event logs based on real-life data.



(a) Model discovered by the IM.



(b) Model discovered by the IMts.

Fig. 10: Models discovered by considering 14 variants of the modified road traffic fine management log [23].

Including translucent relationships in addition to the existing method (IMtf and IMts) prevents fall-throughs of the IM, which lowers the precision significantly. Overall, the IMts seem to be the most stable solution.

7 Conclusion

This paper presents how information on enabled activities can be used in state-of-the-art process-discovery techniques so that fewer variants are needed to discover an equivalent, if not even better, process model. We defined three translucent activity relationships based on information on enabled activities. These relationships are a general concept that can be introduced in existing process-discovery techniques. We used these relationships in combination with translucent start and end activities to construct a tDFG. The tDFG can be used by existing DFG-based techniques, like the IM, to use the information on enabled activities. We implemented the creation tDFGs and their usage at various points in the IM. As we conclude from our experiments, embedding these relationships boosts fitness and precision. However, only relying on these relationships may not be as beneficial as a mixed approach, combining DFG and tDFG. As observed in our experiments, using translucent relationships after the cut detection on a DFG fails (IMts) is the most stable solution. Since translucent event logs can be created when observing tasks done in a desktop environment, the process models discovered by using information on enabled activities provide a solid foundation for creating RPA bots. There are still pointers for future work. As we observed in our evaluation, information on enabled activities is valuable for process discovery. Thus, extending existing process-discovery techniques to incorporate this information is beneficial. In addition, we defined three relationships; however, it is possible to design more relationships, e.g., to capture causal relationships. It is also possible to design relationships beneficial for solving discovery problems such as long-term dependencies. Moreover, a stand-alone discovery technique besides the state-based region approach seems promising since the existing techniques are not designed to exploit information on enabled activities. Furthermore, analyzing activities that are enabled but never executed seems interesting since a log is a sample of real-world behavior. Overall, there is a need to incorporate information on enabled activities in process discovery by enhancing existing discovery techniques, defining more relationships, or designing new discovery techniques.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016). https://doi.org/10.1007/978-3-662-49851-4
2. van der Aalst, W.M.P.: Lucent process models and translucent event logs. Fundam. Informaticae **169**(1-2), 151–177 (2019). https://doi.org/10.3233/FI-2019-1842
3. van der Aalst, W.M.P.: Foundations of process discovery. In: Process Mining Handbook, pp. 37–75. Springer (2022). https://doi.org/10.1007/978-3-031-08848-3_2
4. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Mining Knowl. Discov. **2**(2), 182–192 (2012). https://doi.org/10.1002/WIDM.1045
5. van der Aalst, W.M.P., Rubin, V.A., Verbeek, H.M.W., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Softw. Syst. Model. **9**(1), 87–111 (2010). https://doi.org/10.1007/s10270-008-0106-z
6. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004). https://doi.org/10.1109/TKDE.2004.47
7. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Conformance checking using cost-based fitness analysis. In: EDOC. pp. 55–64. IEEE Computer Society (2011). https://doi.org/10.1109/EDOC.2011.12
8. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.P.: Alignment based precision checking. In: BPM Workshops. Springer (2012). https://doi.org/10.1007/978-3-642-36285-9_15
9. Augusto, A., Carmona, J., Verbeek, E.: Advanced process discovery techniques. In: Process Mining Handbook, pp. 76–107. Springer (2022). https://doi.org/10.1007/978-3-031-08848-3_3
10. Augusto, A., Conforti, R., Dumas, M., Rosa, M.L., Polyvyanyy, A.: Split miner: automated discovery of accurate and simple business process models from event logs. Knowl. Inf. Syst. **59**(2), 251–284 (2019). https://doi.org/10.1007/s10115-018-1214-x, `https://doi.org/10.1007/s10115-018-1214-x`
11. Augusto, A., Dumas, M., Rosa, M.L.: Automated discovery of process models with true concurrency and inclusive choices. In: Process Mining Workshops - ICPM. pp. 43–56. Springer (2020). https://doi.org/10.1007/978-3-030-72693-5_4
12. Berti, A., van Zelst, S., Schuster, D.: Pm4py: A process mining library for python. Software Impacts **17**, 100556 (2023). https://doi.org/https://doi.org/10.1016/j.simpa.2023.100556
13. Beyel, H.H., van der Aalst, W.M.P.: Creating translucent event logs to improve process discovery. In: Process Mining Workshops - ICPM. pp. 435–447. Springer (2022). https://doi.org/10.1007/978-3-031-27815-0_32

14. Beyel, H.H., van der Aalst, W.M.P.: Translucent precision: Exploiting enabling information to evaluate the quality of process models. In: RCIS. pp. 29–37. Springer (2024). https://doi.org/10.1007/978-3-031-59468-7_4
15. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving petri nets for finite transition systems. IEEE Trans. Computers **47**(8), 859–882 (1998). https://doi.org/10.1109/12.707587
16. Dumas, M., Rosa, M.L., Leno, V., Polyvyanyy, A., Maggi, F.M.: Robotic process mining. In: Process Mining Handbook, pp. 468–491. Springer (2022). https://doi.org/10.1007/978-3-031-08848-3_16
17. Ehrenfeucht, A., Rozenberg, G.: Partial (set) 2-structures. part I: basic notions and the representation problem. Acta Informatica **27**(4), 315–342 (1990). https://doi.org/10.1007/BF00264611
18. Ehrenfeucht, A., Rozenberg, G.: Partial (set) 2-structures. part II: state spaces of concurrent systems. Acta Informatica **27**(4), 343–368 (1990). https://doi.org/10.1007/BF00264612
19. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - A constructive approach. In: PETRI NETS. pp. 311–329. Springer (2013). https://doi.org/10.1007/978-3-642-38697-8_17
20. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Business Process Management Workshops. pp. 66–78. Springer (2013). https://doi.org/10.1007/978-3-319-06257-0_6
21. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from incomplete event logs. In: PETRI NETS. pp. 91–110. Springer (2014). https://doi.org/10.1007/978-3-319-07734-5_6
22. Leno, V., Polyvyanyy, A., Dumas, M., Rosa, M.L., Maggi, F.M.: Robotic process mining: Vision and challenges. Bus. Inf. Syst. Eng. **63**(3), 301–314 (2021). https://doi.org/10.1007/s12599-020-00641-4
23. de Leoni, M.M., Mannhardt, F.: Road traffic fine management process (2015). https://doi.org/10.4121/UUID:270FD440-1057-4FB9-89A9-B699B47990F5
24. Mannhardt, F.: Sepsis cases - event log (2016). https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460
25. Mannhardt, F.: Hospital billing - event log (2017). https://doi.org/10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741
26. Munoz-Gama, J., Carmona, J.: A fresh look at precision in process conformance. In: BPM. Springer (2010). https://doi.org/10.1007/978-3-642-15618-2_16
27. Reisig, W.: Petri Nets: An Introduction, EATCS Monographs on Theoretical Computer Science, vol. 4. Springer, Heidelberg (1985). https://doi.org/10.1007/978-3-642-69968-9
28. Solé, M., Carmona, J.: Light region-based techniques for process discovery. Fundam. Informaticae **113**(3-4), 343–376 (2011). https://doi.org/10.3233/FI-2011-612
29. Solé, M., Carmona, J.: Incremental process discovery. Trans. Petri Nets Other Model. Concurr. **5**, 221–242 (2012). https://doi.org/10.1007/978-3-642-29072-5_10
30. Solé, M., Carmona, J.: Region-based foldings in process discovery. IEEE Trans. Knowl. Data Eng. **25**(1), 192–205 (2013). https://doi.org/10.1109/TKDE.2011.192
31. Syed, R., Suriadi, S., Adams, M., Bandara, W., Leemans, S.J.J., Ouyang, C., ter Hofstede, A.H.M., van de Weerd, I., Wynn, M.T., Reijers, H.A.: Robotic process automation: Contemporary themes and challenges. Comput. Ind. **115**, 103162 (2020). https://doi.org/10.1016/j.compind.2019.103162