INCREMENTAL TRANSITION SYSTEM CONSTRUCTION WITH REFINEMENT FOR REGION-BASED PROCESS MINING

Shengrui Peng[1] and Helena Szczerbicka[1]

[1]L3S Research Center, Leibniz University Hannover, e-mail: {peng, hsz}@l3s.de

KEYWORDS

Region-based Process Mining, Incremental Algorithm, Transition System, Configuration Refinement.

ABSTRACT

Process discovery focuses on deriving process models from event logs to facilitate the analysis of complex workflows. Region-based approaches are known for their ability to effectively capture concurrency and handle invisible transitions, but they suffer from high computational demands and reliance on predefined state information. This paper introduces an incremental algorithm to construct *transition systems* (TS) from event logs, integrating a refinement strategy to optimize the representation of the state. The proposed method significantly reduces the complexity of the transition system by configuring state abstractions using past, future, or hybrid horizons. A refinement strategy is used to determine optimal configurations for state representation, ensuring model quality through precision and simplicity metrics. The approach is evaluated using a case study that demonstrates its effectiveness in improving scalability and efficiency in region-based process discovery.

1 INTRODUCTION

Process mining is a multifaceted field that bridges several disciplines, aiming at constructing precise process models from *event log* data to enable in-depth process analysis with the help of simulation. It is widely used to support applications in manufacturing and services where processes need to be continuously improved and adapted. *Event logs* can be understood as sequential records of operations performed. Modern information systems generate massive amounts of data, which poses significant challenges in efficiently constructing process models from these large datasets. Various *discovery algorithms* have been developed to address this challenge (Augusto et al., 2022), producing distinct types of process models, including *transition systems*, *Petri nets*, and *Business Process Modeling Notation* (van der Aalst, 2011). Among these approaches, the *region-based process discovery* approach is known to produce robust models capable of handling complex behaviors such as concurrency and invisible transitions. Petri net (PN) proposed in Petri (1962) is used to represent the process model specifically due to their solid mathematical foundations and ability to effectively represent concurrent processes.

Compared to other process discovery approaches, the main limitation of region-based algorithms is their high computational cost when handling large or highly variable logs. Typically, these algorithms need a transition system (TS) as an intermediate representation of the event log, from which a PN is derived using the *theory of regions* (Desel and Reisig, 1996). While many works focus on improving scalability and computational efficiency, a fundamental question is often overlooked, i.e., how to find a suitable (or even optimal) TS from a given event log. Since event logs generally do not provide state information, the first task is to extract it. Different abstractions can be applied to define states through their past (events before the state), future (events after the state), or both (van der Aalst et al., 2010). However, the modelers usually decide how these abstractions are combined. Thus, a systematic strategy is needed to identify a suitable configuration for representing states.

In this work, we propose an incremental algorithm for constructing transition systems from event logs to boost region-based algorithms. With abstractions, the equivalent states in *sub-TS*s can be identified and merged automatically, potentially reducing the size of resulting *super-TSs*. A *sub-TS* is a sequential unit TS derived from a single trace, and a *super-TS* is the TS constructed by integrating the *sub-TSs*. We then use two metrics, namely precision and simplicity, to evaluate and compare the *super-TS*s under different configurations. Finally, we introduce a refinement strategy to configure the abstractions.

This work is organized as follows. The section 2 reviews the literature, points out challenges and highlights the necessity of our approach. The section 3 covers fundamental definitions and concepts, especially abstractions applicable to event logs. The section 4 introduces our incremental algorithm and details the configuration refinement strategy. A numerical case study illustrating the efficacy of the proposed approach appears in section 5. Finally, section 6 provides conclusions and suggests future research directions.

2 LITERATURE REVIEW

The region-based process discovery approach originates from the Petri net (PN) synthesis problem proposed by

Communications of the ECMS, Volume 39, Issue 1,
Proceedings, ©ECMS Marco Scarpa, Salvatore Cavalieri,
Salvatore Serrano, Fabrizio De Vita (Editors) 2025
ISBN: 978-3-937 436-86-9/978-3-937 436-85-2
(CD) ISSN 2522-2414

(Desel and Reisig, 1996). The objective is to generate an *elementary PN* from a *(labeled) transition system* (TS), i.e., a concurrency model derived from sequential observations, using the *theory of regions* proposed in (Ehrenfeucht and Rozenberg, 1990). An *elementary PN* solves this synthesis problem *iff.* its reachability graph is isomorphic to the given TS. (Cortadella et al., 1995) also presented an early method for generating a safe PN from a (labeled) TS. Later, (Cortadella et al., 1998) introduced a broader class of (labeled) TS, called the excitation-closed TS. The solution is no longer restricted to elementary PN. Subsequently, region-based approaches are adapted for the *process mining problem* (see Definition 2). Typically, the event log is transformed into a TS, and then a PN is derived. One major issue in this approach is extracting state information from event logs, which provide only sequences of activities. In other words, the state information is not directly available from an event log.

An instance-based multi-step approach is proposed in (van Dongen and van der Aalst, 2004, 2005), focusing on constructing models at the instance level (i.e., *sub-TS*) and then aggregating them into an overall process model (i.e., *super-TS*). This enables analyzing individual instances without requiring a complete process model. Constructing models at the instance level avoids scalability issues, even with large data sets. However, this approach relies on causal relations from the event log, so instance-graph accuracy depends on log quality and detail. As an improvement, (van Dongen et al., 2007) propose an incremental approach to learn a PN from event data iteratively. This iterative strategy reduces memory requirements, making it feasible for large logs. However, while memory usage is reduced, computation time increases because all regions must be computed before identifying the minimal ones. As noted in (Solé and Carmona, 2012), this approach trades space for time.

The two-step approach in (van der Aalst et al., 2010) details how event logs are transformed into a TS and how a PN is derived from it. Unlike the earlier-mentioned iterative approach, which builds the TS incrementally, this method constructs a complete TS in one step, so it needs representative logs. The authors also discuss representing states as sequences of activities, applying various abstractions to the event log. However, they restricted the approach to *elementary* PNs. (Shershakov et al., 2017) address the issue of evaluating the transition systems generated by different configurations. The authors essentially use simplicity and precision as metrics. However, the author only considers the past (or history) of a state, and a numerical search is used to find suitable configurations.

Another issue with region-based approaches, as stated in (Augusto et al., 2022), is their high computational cost. Recent works address scalability constraints and propose efficient methods for constructing a TS from large event logs. For example, (Teren et al., 2023a,b) introduce a framework to decompose a PN into synchronizing sub-systems. In (Peng and Szczerbicka, 2024), the authors enhance the region-based process discovery algorithm by proposing an advanced minimal region generation approach that is twice as fast as the original.

Typically, these works assume that the TSs are already extracted from event logs, so they focus on improving the efficiency of learning PNs via the *theory of regions*. However, accurately extracting state information and selecting a suitable TS representation remain crucial for the final PN. Thus, simply stating how to extract state information is insufficient. It is also necessary to provide methods for evaluating different configurations, helping to find the most suitable TS representation at a moderate computational cost.

3 PRELIMINARIES

In this section, we explore the foundational concepts vital for the focus of this paper: incremental construction of a transition system from an event log and refining the configurations. Each topic is selected to offer a comprehensive theoretical background that enhances understanding of our proposed algorithm.

3.1 Region-based Process Discovery

Process mining is an interdisciplinary field that constructs detailed process models from *event logs* to allow in-depth analysis. As modern information systems generate vast amounts of data, efficiently derive process models from these datasets remains a challenge. Various *discovery algorithms* have been developed to address this, producing models such as *transition systems* (TS), *Petri nets* (PN), and *Business Process Modeling Notation* (van der Aalst, 2011). Region-based approaches typically use TS as an intermediate representation of event logs, from which a PN is derived. The definition of a (labeled) transition system is provided in Definition 3. Due to space limitations, details of PN and its derivation via region-based approaches are referred to in the literature cited in section 2.

Definition 1 (Language and Overapproximation (Solé and Carmona, 2012))**.** *The **language** of a system, denoted by $\mathcal{L}(\mathsf{S})$, is the set of all possible sequences feasible from the initial state. For two systems S_1 and S_2. S_2 is called an **overapproximation** of S_1, when $\mathcal{L}(\mathsf{S}_1) \subseteq \mathcal{L}(\mathsf{S}_2)$.*

Definition 2 (Process Mining Problem (Bergenthum et al., 2007))**.** *The objectives of the process mining problem can be articulated as follows: given an event log L from system S, we are searching for a preferably small finite marked PN, s.t. 1) $\mathcal{L}(\mathsf{S}) \subset \mathcal{L}(\mathsf{PN})$; and 2) $\mathcal{L}(\mathsf{PN}) \setminus \mathcal{L}(\mathsf{S})$ is minimal, which implies that the learned PN should be the tightest overapproximation over S.*

Definition 3 ((Labeled) Transition System)**.** *A transition system (TS) is a quadruple $\mathsf{TS} = (S, E, F, s_0)$, where S is the set of states, E is the set of events, $F : S \times E \times S$ is the set of **labeled** arcs, $s_0 \in S$ is the initial state.*

3.2 From Event Log to Transition System

The size of the extracted transition systems is highly related to the way the states of the process are represented. First, we define the trace and the event log:

Definition 4 (Trace and Event Log (van der Aalst, 2011))**.** *Given the universe of activities $\mathcal{A}$. A **trace** $\sigma =< a_1, \ldots, a_n > \in \mathcal{A}^*$ is a sequence of activities, where $\mathcal{A}^*$ is the Kleene closure of the set $\mathcal{A}$. An event log is a multiset of traces, i.e. $\mathsf{L} \in \mathcal{B}(\mathcal{A}^*)$.*

One of the issues in transforming an event log into a transition system is that the event logs contain various information about events, but there is no information about the states. This is typically a challenge for constructing a transition system, since it is a state-based framework. Thus, identification of the states is crucial. In the literature, a state can be identified by considering the ***past/future (P/F) strategy***: ① *past*, i.e., what has happened before the state; ② *future*, i.e., what will happen after the state; and ③ *hybrid*, i.e., consider both past and future (van der Aalst et al., 2010). In the rest of the work, we use P/F-1, P/F-2, and P/F-3 to represent the three different strategies accordingly. Take the trace σ_1 in Figure 1 as an example. After activity c is performed, the process has entered the state s. Based on the strategies stated above, the state s can be expressed as (a-b-c, -), (- , d-e-f-g) or (a-b-c, d-e-f-g) considering the P/F-1, the P/F-2, or P/F-3 respectively. A state is written in the form of a tuple, where the first entry represents the past of the state, and the second the future. The symbol '-' is used as a placeholder when the past or future is not used or when the state has no past or future. Thus, in this work, the state and its equivalence are defined as follows.

Definition 5 (State and Equivalence)**.** *Its past and / or future give a state with or without considering their orders, written as a tuple $s = (s_{past}, s_{future})$. The states s and s' are considered equivalent when their past and future are equivalent, i.e., $s = s'$, iff. $s_{past} = s'_{past}$ and $s_{future} = s'_{future}$.*

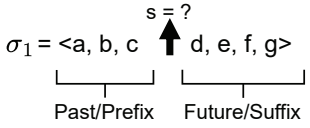


Figure 1: Example of state representation.

The *P/F strategy* sets the baseline for the state representation, which can be defined by ***horizons***. However, it needs to be further polished to fit the intended process. For this purpose, several other *abstractions* were discussed in (van der Aalst et al., 2010), i.e., the ***length***, the ***projection***, the ***frequency***, the ***order***, and the ***grouping***.

The ***horizons*** of the past and future, denoted by $\mathsf{h_p}$ and $\mathsf{h_f}$, respectively, define the level to which activity in the past and future affects the state. $\mathsf{h_p} = 0$ if the past is irrelevant to the state, and similarly $\mathsf{h_s} = 0$ if the future is irrelevant. For example, if we consider a configuration, that is, $\mathsf{h_p} = 2$ and $\mathsf{h_s} = 1$, for the state s in Figure 1, then it is represented as $s =$ (b-c, d). Note that the horizons are named as *window size* in other works such as (Shershakov et al., 2017). ***Lengths***, denoted by $\mathsf{l_p}$ and $\mathsf{l_f}$, act similarly to the ***horizons***, which further restricts the length of state representation. However, this abstraction is **not** considered in this work. First, these two abstractions share very similar functionality. Second, the parameter '***lengths***' is completely artificial and is decided by the modelers, adding more subjectivity to the model. If the length parameters $\mathsf{l_p} = 1$ and $\mathsf{l_f} = 1$ are applied to the state s, we obtain $s =$ (c , d)

Projecting the traces onto a subset of the activities provides the opportunity to filter out unimportant activities. Given a set of important activities $\boldsymbol{X}$, the ***projection*** of a trace σ onto $\boldsymbol{X}$ is denoted by $\sigma_{\uparrow \boldsymbol{X}}$. For example, the projection of a trace $\sigma =< a, b, a, c, d >$ on a set $\boldsymbol{X} = \{a, c\}$ is $\sigma_{\uparrow \boldsymbol{X}} =< a, a, c >$. This abstraction intends to help the modelers focus on the important activities of the process and prevent the construction of over-complicated models. However, applying the projection without *a priori* knowledge about the process can lead to over-simplification, and the resulting model can oversee some important behaviors.

The ***frequency*** defines whether the occurrences of the traces and the state-transitions are considered in the model. The frequencies can be used to filter out important behaviors of the process and to evaluate stochastic parameters. Neglecting the frequency can reduce the computational cost to a certain extent since only unique traces are considered. However, recent approaches address the value of stochastic characteristics in the models. In addition, the reduction obtained by ignoring the frequencies is far from significant compared to modern hardware improvements. Thus, frequencies are generally considered in this work.

The abstraction '***order***' determines whether the ordering of the past and/or future is important to the state representation. Consider two states based on the past $s_1 =$ (a-b-c, -) and $s_2 =$ (a-c-d, -). If the order of the past is irrelevant, then s_1 and s_2 are equivalent. If the order of the past is considered, then obviously $s_1 \neq s_2$.

Grouping similar activities together is another strategy that helps to reduce the size of the model. Technically, the recent advance in Large Language Models alleviates this task tremendously (Kourani et al., 2024). There are still some issues with this abstraction. First, grouping similar semantic events requires that the event logs be properly labeled. Second, it brings confusion to the model, where there are several semantic similar events, but they lead to completely different states of the process. Thus, we leave out this abstraction in the state representation. But address the possibility of using this abstraction in the *post-processing* to get a more precise model.

We summarize the usage of the different abstractions in this work. First, the ***horizons*** and the ***order*** are affecting directly the state representation. Second, the ***frequency*** is generally considered, i.e., it is set to True by default. Thirdly, the ***projection*** and ***grouping*** are used only in post-processing to further polish the result. Last, the ***lengths***

are omitted from our approach due to their redundancy and subjectivity.

3.3 Metrics

Given a transition system $\mathsf{TS} = (S, E, F, s_0)$, the ***simplicity*** of the transition system is given by (Buijs et al., 2012):

$$\mathsf{Simp}(\mathsf{TS}) = \frac{|E| + 1}{|S| + |F|}, \tag{1}$$

where $|S|$, $|E|$ and $|F|$ are the cardinality of the set S, E and F respectively. The ***precision*** of the TS is:

$$\mathsf{Prec}(\mathsf{TS}) = \frac{1}{|S|} \cdot \sum_{s \in S} \mathsf{Prec}(\mathsf{s}) \tag{2}$$

with

$$\mathsf{Prec}(\mathsf{s}) = \frac{1}{\mathsf{NoV}(s)} \cdot \sum_{i=1}^{\mathsf{NoV}(s)} \frac{|s\bullet| - |\hat{s\bullet}_i|}{|s\bullet|}$$

where $\mathsf{Prec}(s)$ is the partial precision for state s, $\mathsf{NoV}(s)$ is a number of all visits of state s during a replay of the full transition system. $\hat{s\bullet}_i$ is a set of penalized output transitions of state s, i.e., transitions that do not have active counterparts compared to the precise model. Details on definitions and implementations can be found in (Shershakov et al., 2017).

4 INCREMENTAL PROCESS MINING WITH STATE IDENTIFICATION

The goal of the incremental algorithm is to learn a suitable transition system representation from a given event log. Note that, different from the algorithm proposed in (van der Aalst et al., 2010), the completeness of the event log is not required. Thus, the central issue remains in configuring the abstractions introduced section 3. The relevant abstractions are summarized in Table 1, which are used to form a Python dictionary like the input CONFIG in the incremental algorithm.

In subsection 3.2, we have reasoned the general usage of ***frequency***, categorized ***projection*** and ***grouping*** to post-processing, and dropped the ***lengths***. Thus, these abstractions do not affect the state representation in this work. The abstractions that we need to consider in the state representation are ***horizons*** and ***order***, i.e., $\mathsf{h_p}$, $\mathsf{h_f}$ and is_order. Although we have reduced the number of relevant abstractions in the state representation, combining ***horizons*** and ***order*** can still lead to a combinatorial problem. Theoretically, there are infinite number of configurations, since the past or future can be infinitely long. Thus, a strategy for the incremental algorithm is required.

4.1 P/F Refinement and Incremental Algorithm

Finding a suitable or optimal abstraction configuration (CONFIG) for state representation is difficult. Typically,

Table 1: Notions and Parameters

Key	Value	Description
h_p	integers	horizon past
h_f	integers	horizon future
is_order	boolean	order
$\boldsymbol{X}$	set of activities	projection
is_grouping	boolean	grouping

Table 2: Constructing TS for L^* with 251 traces and 247 activities using P/F-1 (Shershakov et al., 2017).

$\mathsf{h_p}$	#S	#ST	Simplicity	Precision
1	248	1755	0.1240	0.3791
3	3603	4838	0.0290	0.8892
5	5601	6129	0.0210	0.9671
7	6515	6806	0.0190	0.9836
10	7228	7392	0.0170	0.9927
15	7840	7905	0.0160	0.9968
∞	8088	8087	0.0153	1.0000

#S: number of states, #ST: number of state-transitions.
Dataset: `https://fluxicon.com/blog/2015/05/bpi-challenge-2015/`

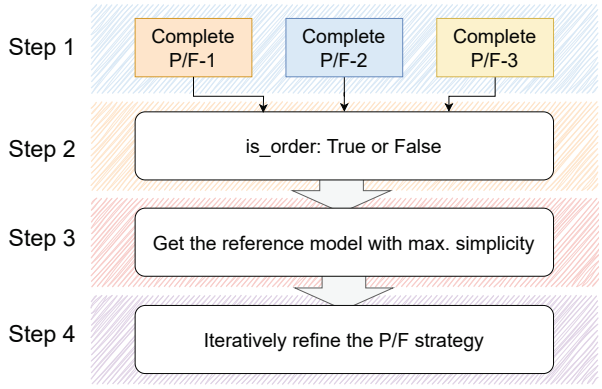


Figure 2: General Refinement Strategy.

there is no *a priori* knowledge about the characteristics of the state in an event log. Decisions are often made based on the experience of the modelers (van der Aalst et al., 2010) or by searching all combinations (Shershakov et al., 2017). Fortunately, certain state-representation features can help. Since is_order is binary, the main complexity lies in choosing the horizons $\mathsf{h_p}$ and $\mathsf{h_f}$.

Table 2 shows how the size of TS (i.e., number of states and state-transitions) learned from L^* changes under P/F-1 as $\mathsf{h_p}$ varies. A larger $\mathsf{h_p}$ yields a more complex but precise TS. Its precision reaches 1 when $\mathsf{h_p} = \infty$, where ∞ denotes the maximum length of past or future. For L^*, the maximal $\mathsf{h_p}$ is 83. Thus, the best P/F-1 setting lies between $\mathsf{h_p} = 1$ (the simplest model) and $\mathsf{h_p} = \infty$ (the most precise). In this example, $\mathsf{h_p} = 3$ seems optimal, offering a substantial precision gain. However, simplicity drops dramatically for higher $\mathsf{h_p}$. In addition, a greater $\mathsf{h_p}$ adds complexity, underscoring the importance of transforming the TS into a PN for a more compact representation (Peng

and Szczerbicka, 2024), although that is not our focus here.

Based on these findings, we can derive a refinement strategy. Knowing that minimal values of $\mathsf{h_p}$ and/or $\mathsf{h_f}$ (i.e., 1) yield the simplest models and simplicity decreases as these parameters grow, we can start with 1 and increase them incrementally. The gain in precision and simplicity can then serve as the stopping criterion, balancing simplicity and precision.

The general refinement strategy is depicted in Figure 2. In step 1, the complete P/F strategies are considered, i.e., $\mathsf{h_p} = \infty$ and/or $\mathsf{h_f} = \infty$. This serves two purposes. First, reference models are required when calculating the precision. As mentioned in subsection 3.3, the precision is a characteristic of the current model against a reference model. Reference models are generally the models with the highest precision. By combining different configurations, we obtain six distinct settings, yielding six different TSs. In step 3, the TS with the highest simplicity is selected as the reference model, and its P/F strategy is used for iterative refinement in step 4. This procedure is demonstrated in Algorithm 1 as a pseudo-code. In this work, the P/F refinement algorithm is stopped when the user-defined min_gain is reached. The precision gain, τ, is the difference between the precision of the current TS and that of the last TS.

The incremental algorithm in Algorithm 2 demonstrates briefly the idea of how a TS is created. First, the set of state-transitions F of the super-TS is initialized as an empty list. This allows us to evaluate the frequencies and stochastic parameters. Then the state-transitions are obtained from traces using the get_st_from_trace function, which are appended to the list F. The algorithm runs until all the traces in the event log L are processed. Let the algorithm run until the event log is empty, which has the benefit that it will also process the traces added to the event log after the algorithm is triggered. The super-TS is created by the function create_ts_from_st with the input F and CONFIG.

Algorithm 1 P/F refinement (ts_pf_refinement)

Require: L, $\boldsymbol{X}$, is_grouping, min_gain
Ensure:
$\mathsf{TS_{ref}} = \text{get\_reference\_model}(L)$
$\mathsf{CONFIG} = \text{get\_initial\_config}(\mathsf{TS_{ref}}, \boldsymbol{X}, \text{is\_grouping})$
$\text{simp\_0}, \text{prec\_0} = 0$
$\tau = 1$
while $\tau \geq \text{min\_gain}$ **do**
    $\mathsf{TS} = \text{get\_ts\_incremental}(L, \mathsf{CONFIG})$
    $\text{prec}_1 = \text{get\_metrics}(\mathsf{TS})$ #Equation 1Equation 2
    $\tau = \max(\text{simp\_1} - \text{simp\_0}, \text{prec\_1} - \text{prec\_0})$
    $\text{simp\_1} = \text{simp\_0}$
    $\text{prec\_1} = \text{prec\_0}$
    $\mathsf{CONFIG} = \text{update\_config}(\mathsf{CONFIG})$
return TS

Algorithm 2 Incremental Algorithm (get_ts_incremental)

Require: L, CONFIG
Ensure:
$\mathsf{F} = [\,]$
while $\mathsf{L} \neq \emptyset$ **do**
    $\sigma = L.pop()$
    $\mathsf{F}_\sigma = \text{get\_st\_from\_trace}(\sigma, \mathsf{CONFIG})$
    $\mathsf{F}.\text{append}(\mathsf{F}_\sigma)$
$\mathsf{TS} = \text{create\_ts\_from\_st}(\mathsf{F}, \mathsf{CONFIG})$
return TS

4.2 Tools Support & Implementation

The incremental algorithms including the numerical case study presented in section 5 are implemented in Python. The code can be found in our Git repository (link: `https://git.com/peng_luh/ecms_2025`). To reduce the work on visualization of transition systems, the TransitionSystem class from the PM4PY package (Berti et al., 2023) is used as the base class.

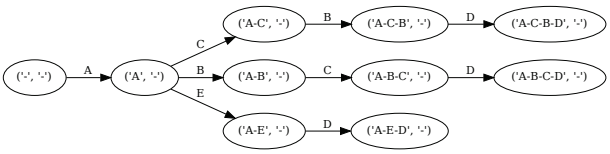
5 NUMERICAL CASE STUDY

To demonstrate the concepts introduced in the last sections, we use the following simple event log as an example: $\mathsf{L}_1 = \left\{[A, B, C, D]^5, [A, C, B, D]^2, [A, E, D]^2\right\}$. The event log is given as a multiset, where the power is used to count the frequency of the sequences. In order to get a view on the difference between the P/F strategies and how they are interacting with the other abstractions, three different *super-transition systems* are constructed, which are depicted in Figure 3 with their configurations, respectively.

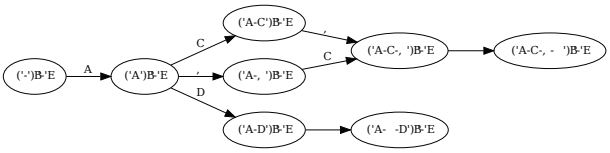
First, there are 10 states in Super-TS-1 (Figure 3a) but only 6 states in Super-TS-3 (Figure 3c), where the complete past ($\mathsf{h_p} = \infty$) and only the last event ($\mathsf{h_p} = 1$) are considered in the state representation, respectively. Indeed, the Super-TS-1 is much more rigid than Super-TS-3 regarding system behaviors. The Super-TS-1 is basically sequential. There is only one decision to be made in the state (A, -). It mimics a system behavior such as a production process, where resources are distributed at (A, -) to three production lines, where they work independently. Even though the Super-TS-3 own fewer states, it depicts more behaviors than the other two. Thus, it is a more general representation of the event log. We observe that there are concurrency, synchronization, and loops in the process.

In Super-TS-1 we also notice that there is a concurrent behavior between ⟦(A-C, -), B, (A-C-B, -) ⟧and ⟦(A-B, -), C, (A-B-C, -)⟧. In some cases, the order of the past events is irrelevant. By ignoring the order of the events, we obtain a slightly different super-transition system depicted in Figure 3b, where the state (A-C-B, -) and the state (A-B-C, -) are merged into one. Thus, the states (A-B, -) and (A-C, -) are synchronized in the state (A-B-C, -) in Super-TS-2. However, the order of the past/future could be irrelevant when the horizon is short enough. For example, in Super-TS-3, the states depend only on the last event.

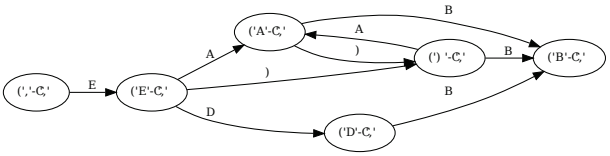
Following the strategy outlined in section 4, the reference models are constructed and their simplicities are evaluated, as shown in Table 3. In this case, the combinations $(\infty, -1, \text{False})$ and $(-1, \infty, \text{False})$, highlighted



(a) Super-TS-1 with $\mathsf{h_{past}} = \infty$ and is_order = True



(b) Super-TS-2 with $\mathsf{h_{past}} = \infty$ and is_order = False



(c) Super-TS-3 with $\mathsf{h_{past}} = 1$ and is_order = False

Figure 3: Super-TSs with $\mathsf{h_{future}} = 0$, $\boldsymbol{X} = \emptyset$, is_frequency = False, is_grouping = None

Table 3: Reference Models and Simplicities

$\mathsf{h_p}$	$\mathsf{h_f}$	is_order	Simplicity
∞	-1	True	0.3158
∞	-1	False	0.3750
-1	∞	True	0.3158
-1	∞	False	0.3750
∞	∞	True	0.2400
∞	∞	False	0.3158

in green, represent the *complete P/F-1 without order* and *complete P/F-2 without order*, respectively. These configurations yield super-TSs with the highest simplicity. For this example, we select the super-TS resulting from $(\infty, -1, \mathsf{False})$ as the reference model, and its configuration serves as the root for refinement. This step determines whether order should be considered. Here, is_order is set to False and carried forward to the next refinement step. The values of $\mathsf{h_p}$ and $\mathsf{h_f}$ dictate further refinement: If either is -1, it is excluded from refinement; if ∞, it undergoes refinement in the next step. Thus, only $\mathsf{h_p}$ will be refined in the next step. The numerical results of the refinement on the horizon $\mathsf{h_p}$ are presented in Table 4. We observe that the precision reaches 1 at $\mathsf{h_p} = 2$. However, the algorithm does not stop when precision reaches 1 but continues until the improvement becomes negligible compared to a user-defined threshold. Although precision cannot be further increased beyond $\mathsf{h_p} = 2$, a noticeable gain in simplicity is observed at $\mathsf{h_p} = 3$. The algorithm continues until $\mathsf{h_p} = 4$ is evaluated and a plateau is detected. At this point, the model with $\mathsf{h_p} = 3$ is selected as the optimal configuration. Details regarding the implementation of this numerical example can be found in the given GitHub repository subsection 4.2.

Table 4: Numerical results of P/F refinement on L_1.

$\mathsf{h_p}$	#S	#ST	Simplicity	Precision
1	6	9	0.4000	0.8333
2	9	9	0.3833	1.0000
3	8	8	0.3750	1.0000
4	8	8	0.3750	1.0000

#S: number of states, #ST: number of state-transitions.

6 CONCLUSION

This paper introduced an incremental approach that enhances the construction and refinement of transition systems (TS) from event logs. Addressing computational challenges, our method optimizes the representation of states by systematically configuring state abstractions based on past, future, and hybrid activity horizons. The proposed refinement strategy ensures that transition systems are both precise and simple, striking a balance between model accuracy and computational efficiency. Through a detailed numerical case study, we demonstrated that our approach significantly reduces the complexity of transition systems while maintaining high model quality. The results confirm that our incremental strategy enables the construction of more efficient and interpretable process models, making it a valuable contribution to the field of process mining and workflow analysis.

Future work will extend this approach to incorporate a region-based process discovery approach, allowing for adaptive changes in the resulting Petri nets model. In addition, integrating the method into large-scale enterprise process mining applications will further validate its practical effectiveness in handling complex, data-driven workflows. Lastly, comparing the proposed approach to existing approaches would help measure the computational effort and show the superiority over existing ones.

Acknowledgments

The authors gratefully acknowledge the financial support by the DFG (German Research Foundation) for the Project "Offshore-Plan" (Porject nummer: 411010215).

References

Adriano Augusto, Josep Carmona, and Eric Verbeek. Advanced Process Discovery Techniques. In Wil M. P. Van Der Aalst and Josep Carmona, editors, *Process Mining Handbook*, volume 448, pages 76–107. Springer International Publishing, Cham, 2022. doi: 10.1007/978-3-031-08848-3_3.

R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process Mining Based on Regions of Languages. In G. Alonso, P. Dadam, and M. Rosemann, editors, *Business Process Management*, volume 4714, pages 375–383. Springer Berlin Heidelberg, 2007. doi: 10.1007/978-3-540-75183-0_27.

A. Berti, S. van Zelst, and D. Schuster. PM4Py: A process mining library for Python. *Software Impacts*, 17:1–7, September 2023. doi: 10.1016/j.simpa.2023.100556.

J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. In *On the Move to Meaningful Internet Systems: OTM 2012*, volume 7565, pages 305–322. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi: 10.1007/978-3-642-33606-5_19.

J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Synthesizing Petri nets from state-based models. In *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, pages 164–171. IEEE, Nov. 5–9 1995. doi: 10.1109/ICCAD.1995.480008.

J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri nets from finite transition systems. *IEEE Transactions on Computers*, 47(8):859–882, August 1998. doi: 10.1109/12.707587.

J. Desel and W. Reisig. The synthesis problem of Petri nets. *Acta Informatica*, 33(4):297–315, 1996. doi: 10.1007/s002360050046.

A. Ehrenfeucht and G. Rozenberg. Partial (set) 2-structures: Part II: state spaces of concurrent systems. *Acta Informatica*, 27(4):343–368, March 1990. doi: 10.1007/BF00264612.

Humam Kourani, Alessandro Berti, Daniel Schuster, and Wil Van Der Aalst. Evaluating Large Language Models on Business Process Modeling: Framework, Benchmark, and Self-Improvement Analysis. 2024. doi: 10.13140/RG.2.2.11821.70880.

S. Peng and H Szczerbicka. Improvements to the region-based petri nets synthesis algorithm for process mining. In *Proceedings of the 16th EAI International Conference on Simulation Tools and Techniques*, Bratislava, 2024. EAI.

C. A. Petri. *Kommunikation mit Automaten*. Phd thesis, Fakultät für Mathematik und Physik der Technischen Hochschule Darmstadt, 1962. Available at `https://edoc.sub.uni-hamburg.de/informatik/volltexte/2011/160/`.

S.A. Shershakov, A.A. Kalenkova, and I.A. Lomazova. Transition Systems Reduction: Balancing Between Precision and Simplicity. In M. Koutny, J. Kleijn, and W. Penczek, editors, *Transactions on Petri Nets and Other Models of Concurrency XII*, volume 10470, pages 119–139. Springer Berlin Heidelberg, Berlin, Heidelberg, 2017. doi: 10.1007/978-3-662-55862-1_6.

M. Solé and J. Carmona. Incremental process discovery. In Kurt Jensen, Susanna Donatelli, and Jetty Kleijn, editors, *Transactions on Petri Nets and Other Models of Concurrency V*, volume 6900, pages 221–242. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-29072-5_10.

V. Teren, J. Cortadella, and T. Villa. Generation of synchronizing state machines from a transition system: A region-based approach. *International Journal of Applied Mathematics and Computer Science*, 33(1), 2023a. doi: 10.34768/amcs-2023-0011.

Viktor Teren, Jordi Cortadella, and Tiziano Villa. Seto: A Framework for the Decomposition of Petri Nets and Transition Systems. In *2023 26th Euromicro Conference on Digital System Design (DSD)*, pages 669–677, Golem, Albania, September 2023b. IEEE. doi: 10.1109/DSD60849.2023.00096.

W. M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi: 10.1007/978-3-642-19345-3.

W. M. P. van der Aalst, V. Rubin, H. M. W. Verbeek, B. F. van Dongen, E. Kindler, and C. W. Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software & Systems Modeling*, 9(1):87–111, January 2010. doi: 10.1007/s10270-008-0106-z.

B. F. van Dongen and W. M. P. van der Aalst. Multi-phase Process Mining: Building Instance Graphs. In *Conceptual Modeling – ER 2004*, volume 3288, pages 362–376. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. doi: 10.1007/978-3-540-30464-7_29.

B.F. van Dongen and W.M.P. van der Aalst. Multi-phase process mining : Aggregating instance graphs into epcs and petri nets. In *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management*, 2005.

B.F. van Dongen, N. Busi, G.M. Pinna, and W.M.P van der Aalst. An iterative algorithm for applying the theory of regions in process mining. In *Proceedings of the Workshop on Formal Approaches to Business Processes and Web Services*, pages 36–55, Eindhoven, 2007. Publishing house of university of Podlasie.

AUTHOR BIOGRAPHIES

Shengrui Peng went to Leibniz University of Hanover, where he started with *Civil Engineering* as a Bachelor's and obtained a Master's degree in *Computational Methods in Engineering* in 2019. Immediately after graduation, he joined *L3S Research Center* as a Ph.D. candidate. He has published several papers on modeling, simulation-based optimization, and process mining. He is strongly interested in the concept of combining process mining techniques with advanced AI approaches. His e-mail address is: `peng@l3s.de` and his profile can be found at `https://www.linkedin.com/in/shengrui-peng-9884926a/`.

Helena Szczerbicka is Professor Emeritus of the Faculty of Electrical Engineering and Computer Science at the Leibniz University of Hanover, Germany. Her research interests focus on modeling, simulation, and optimization. She has served on the Board of Directors of the Society of Modeling and Computer Simulation International SCS for many years. She has received much recognition for her various scholarly publications, achievements, and professional activities that focus on the importance of modeling and simulation in its interaction with other disciplines. Her email address is: `hsz@sim.uni-hannover.de`. Her website is `https://www.l3s.de/people/members/helena-szczerbicka/`