Transition Systems Reduction: Balancing between Precision and Simplicity*

Sergey A. Shershakov, Anna A. Kalenkova, and Irina A. Lomazova

National Research University Higher School of Economics,
20 Myasnitskaya st., 101000 Moscow, Russia,
sshershakov@hse.ru,
WWW home page: http://pais.hse.ru

Abstract Transition systems are a powerful formalism, which is widely used for process model representation. A number of approaches were proposed in the process mining field to tackle the problem of constructing transition systems from event logs. Existing approaches discover transition systems that are either too large or too small. In this paper we propose an original approach to discover transition systems that perfectly fit event logs and whose size is adjustable depending on the user's need. The proposed approach allows achieving a required balance between simple and precise models.

Keywords: transition systems, process mining, model reduction, process model quality

1 Introduction

Process mining is a relatively new discipline, whose basic research and practical purpose is to extract process models from data given in the form of *event logs*, checking existing models for conformance to actual processes and improving them. *Transition systems* are extensively used to formalize processes extracted from event logs. A transition system can be constructed from an event log by using prefix-based techniques in a very natural way [2]. We consider several metrics that describe the model's quality [7]. Replay fitness quantifies the extent to which a process model can reproduce the behavior recorded in a log. Complexity of the model is estimated by simplicity and precision (the *metrics*, which shows how precise the model is in respect to the event log).

The major weakness of models constructed from real-life event logs is their size. Despite the fact that there are a number of approaches aimed to reduce the size of transition systems [2], application of the existing approaches results in either too large or too small models. In the former case the model size is big enough for being readable. Furthermore, it becomes difficult or even impossible

* This work is supported by the Basic Research Program at the National Research University Higher School of Economics in 2016 and the Russian Foundation for Basic Research, project No. 15-37-21103.

to apply existing transition system analysis techniques that are sensitive to the size of input models. For example, the state-based region algorithm [8] has an exponential complexity dependence on the size of the input model, so its applicability is limited with fairly small models. In the latter case, due to states merging a rather small model implies considerably much of extra behavior, which makes the model less precise and thus less applicable.

The main goal of our study is to develop an approach for reducing the size of a transition system mined from an event log in a flexible manner. This paper describes an original 3-step algorithm achieving the goal by using a variable-size window based on a state frequency characteristic. The approach preserves (perfect) fitness of a model and balances between its simplicity and precision by introducing a set of adjustable parameters.

Thus, the *main contributions* are as follows: (1) an original method for reducing transition systems and justification of its applicability; (2) a set of experimental results which show the advantages of the proposed approach as compared with existing methods; an openly available proof of the concept through implementation in a set of ProM plug-ins.

The remaining part of the paper is organized as follows. Section 2 gives an overview of related work in the context of inferring transition systems and their application in the process mining domain. Section 3 introduces basic concepts used further in this paper. A detailed description of the proposed algorithm is given in Section 4. A novel precision calculation algorithm, some significant implementation details, and experimental results are discussed in Section 5. Finally, Section 6 concludes the paper and discusses some directions for future work.

2 Related work

A number of works concerning inferring transition systems from event traces exist. Biermann and Feldman in their work [6] proposed a k-tails algorithm which merges states of a FSM by basing on the similarity of their behavior. The algorithm falls into the class of prefix tree merging methods. Angluin [4] proposed a method of prefix tree states merging based on a notion of k-reversibility. In [11], Lorenzoli et al. proposed a GK-tail approach, an extension of the k-tail algorithm, dealing with parametrized finite state automata.

Cook and Wolf in their work [9] introduced *process discovery*, a new data analysis technique in the context of software engineering processes. They considered automatic generation of a formal model describing an ongoing process from captured event data. A new Markov method was developed specifically for this purpose. Moreover, two existing methods, the k-tail and RNet (based on neural networks) ones, were adopted for the process discovery technique.

Process discovery along with *conformance checking* and *process enhancement* form the basis of *process mining* [3], which deals with various types of process models including Petri nets, transition systems, fuzzy maps, C-nets, BPMN and others. In the context of process mining, transition systems are considered both as a self-independent model and an intermediate model for building another

type of model on its basis. In the latter case, one should mention region-based approaches discussed in [5,10,8,15].

The leading role of a transition system as an intermediate representation of a process is discussed in [2]. The paper considers a number of different strategies to construct a transition system that is more suitable to be a base for a resulting final Petri net model with respect to desirable metrics. Nevertheless, all discussed strategies are based on inferring algorithms with a fixed window.

3 Preliminaries

This section introduces basic concepts related to event logs, transition systems and some other notations that are needed for explaining the approach.

$\mathcal{P}(A)$ denotes the power set of A — the set of all subsets of A. For a given set A, A^* is the set of all finite sequences over A.

Definition 1 (Event Trace, Event Log). *Let A be a set of activities. The* (event) trace *is a sequence $\sigma = \langle a_1, a_2, ..., a_i, ..., a_n \rangle \in A^*$. By $\sigma(i) = a_i$ we denote i-th element (event) of the trace. The* $[i, k]$-subtrace *of trace σ ended at i-th event a_i is defined as*

$$\sigma[i,k] = \begin{cases} \langle \sigma(1), \sigma(2), ..., \sigma(i) \rangle, & \text{if } k > i; \\ \langle \sigma(i-k+1), ..., \sigma(i) \rangle, & \text{if } 1 \leq k \leq i; \\ \langle \rangle, & \text{if } k = 0. \end{cases} \tag{1}$$

The complete subtrace *of the trace σ ended at i-th event a_i is $\sigma[i] = \sigma[i, i]$. By $|\sigma|$ we denote a trace length. For $k \leq i$, k denotes the length of the subtrace. $L \in \mathcal{P}(A^*)$is an* event log *and $|L|$ is a log size that is equal to a number of all traces.*

We assume, that event logs do not contain process states explicitly. This way, we need to deduce the desirable states from an event log based on some approach.

In [13], four approaches to determine the state in a log were proposed. They are *past*, *future*, *past and future* and *explicit knowledge*. In this paper we consider only *past* approach, according to which a state is constructed based on the prefix of a trace. Then, the order of activities is important. Hence, we apply sequence policy [13] for determining a state.

Definition 2. *A* labeled transition system *is a tuple $TS = (S, E, T, s_0, AS)$, where S is a* state space*, E is a* set of labels*, $T \subseteq S \times E \times S$ is* set of transitions*, $s_0 \in S$ is an initial state, and $AS \subseteq S$ is a set of* accepting (final) states*. We denote the set of* output *(*input*) transitions of a state $s \in S$ as $s\bullet = \{t = (s, e, s') \in T \mid e \in E, s' \in S\}$ ($\bullet s = \{t = (s', e, s) \in T \mid e \in E, s' \in S\}$).*

Definition 3 (k-window transition system). *Let $L \in \mathcal{P}(A^*)$ be a log over set of activities A and let $k \in \mathbb{N}$ be a natural number called* window size*. $TS(L, k) =$*

(S, E, T, s_0, AS) *is a k-window (labeled) transition system built for log* L *and window size* k*, where* $S = \{s_0\} \cup \{\sigma[i,k] \mid \sigma \in L, 1 \leq i \leq |\sigma|, k \leq i\}$ $T = \{(s_0, \sigma(1), \sigma[1,k]) \,|\, \sigma \in L\} \cup \{(\sigma[i-1,k], \sigma(i), \sigma[i,k]) \,|\, \sigma \in L, 2 \leq i \leq |\sigma|\}$*,* $AS = \{s \in S \mid s = \sigma[|\sigma|,k], \sigma \in L\}$*, and* $E = A$*.*

Definition 4 (Full transition system). *Let* $L \in \mathcal{P}(A^*)$ *be a log over set* A *of activities. The* full transition system $TS(L) = (S, E, T, s_0, AS)$ *for the log* L *is a labeled transition system built for log* L*, where* $S = \{s_0\} \cup \{\sigma[i] \mid \sigma \in L, 1 \leq i \leq |\sigma|\}$*,* $T = \{(s_0, \sigma(1), \sigma[1]) \,|\, \sigma \in L\} \cup \{(\sigma[i-1], \sigma(i), \sigma[i]) \,|\, \sigma \in L, 2 \leq i \leq |\sigma|\}$*,* $AS = \{s \in S \mid s = \sigma[|\sigma|], \sigma \in L\}$*, and* $E = A$*.*

Definition 5. *Let* $TS(L) = (S, E, T, s_0, AS)$ *be a transition system over a set of activities* $E = A$*, and let* $\sigma = \langle a_1, ..., a_n \rangle$ *be a trace over* A *and* $n = |\sigma|$*. We say that trace* σ *can be replayed in transition system* $TS(L)$ *if there is a sequence of states* $\langle s_0, ..., s_n \rangle$ *such that* $\exists t_1 = (s_0, a_1, s_1), t_2 = (s_1, a_2, s_2), ..., t_n = (s_{n-1}, a_n, s_n)$ *where* $s_0, s_1, ..., s_n \in S, t_1, t_2, ..., t_n \in T$*. We denote this as* $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} ... \xrightarrow{a_n} s_n$*. We say that trace* σ *can be* partially replayed *by its prefix in a transition system* TS *if* $\exists k < n : \exists s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} ... \xrightarrow{a_k} s_k$ *and* $\nexists s_k \xrightarrow{a_{k+1}} s_{k+1} \xrightarrow{a_{k+2}} ... \xrightarrow{a_n} s_n$*. We denote* $\sigma^+(TS) = \langle a_0, a_1, ..., a_k \rangle$ *and* $\sigma^-(TS) = \langle a_{k+1}, ..., a_n \rangle$*. Hence,* $\sigma(TS) = \sigma^+(TS) + \sigma^-(TS)$ *where* $+$ *denotes concatenation of two sequences.*

To measure the quality of resulting models, we consider three quality metrics. We based them primarily on the work [7] and adopted them for transition systems. *Fitness* quantifies the extent to which a transition system can reproduce traces recorded in a log. *Simplicity* quantifies the complexity of a model. Simplicity is measured by comparing the size of a given transition system $TS(L)$ with the simplest possible transition system, which is the flower model (Fig. 1a). *Precision* compares transition system $TS(L)$ with the full transition system built for log L, considering the latter to be the most precise.

Definition 6 (Metrics). *Let* L *be an event log and let* $TS(L) = (S, E, T, s_0, AS)$ *be a transition system built for* L*.* Fitness *is defined to be the ratio of the number of traces from log* L *that can be fully replayed in transition system* $TS(L)$ *to the total number of all traces. Log* L perfectly fits *transition system* $TS(L)$ *iff all traces of* L *can be fully replayed in* $TS(L)$*.*

Simplicity *of* $TS(L)$ *is:*

$$Simpl(TS(L)) = \frac{|E| + 1}{|T| + |S|}.$$

Precision *of* $TS(L)$ *is:*

$$Prec(TS(L)) = \frac{1}{|S|} \cdot \sum_{s \in S} Prec(s), \quad Prec(s) = \frac{1}{NoV(s)} \cdot \sum_{i=1}^{NoV(s)} \frac{|s\bullet| - |\widehat{s\bullet}_i|}{|s\bullet|},$$

where $Prec(s)$ *is a* partial precision *for state* s*,* $NoV(s)$ *is a number of all visits of state* s *during a replay of the full transition system.* $\widehat{s\bullet}_i$ *is a set of* penalized

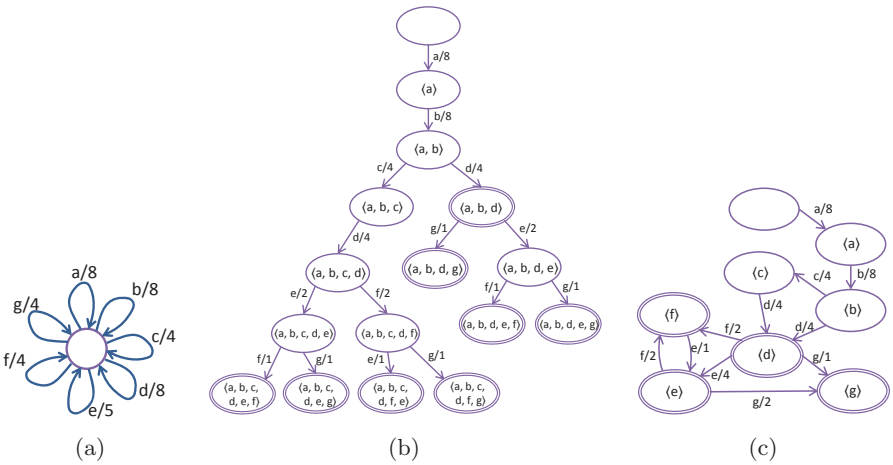


Figure 1: (a) "Flower" model for log L_1; (b) $TS_1(L_1)$ built for log L_1; (c) transition system built for log L_1 with a fixed window of size 1

output transitions *of state s, i.e., transitions that do not have active counterparts as compared to the precise (full TS) model.*

4 Algorithm Description

For the clarification of the approach, the following motivating example is considered. Let L_1 be an event log that is defined as follows:

$$L_1 = \{\langle a,b,c,d,e,f\rangle, \langle a,b,c,d,e,g\rangle, \langle a,b,c,d,f,e\rangle, \langle a,b,c,d,f,g\rangle, \\ \langle a,b,d\rangle, \langle a,b,d,g\rangle, \langle a,b,d,e,f\rangle, \langle a,b,d,e,g\rangle\} \quad (2)$$

In addition to the previously discussed flower model, one can build a number of other models that perfectly fit log L_1. A model built with unlimited window size is depicted on Figure 1b. This model is a *full transition system* by the definition. Another model built by an algorithm with a fixed window of size 1 is depicted on Figure 1c. Although all these models perfectly fit log L_1, none of them is satisfactory in simplicity and precision at the same time. Thus, we are interested in a trade off between these metrics.

The proposed approach incorporates a 3-steps algorithm sequentially building 3 transition systems. The first transition system, TS_1, is built from an event log. The second (TS_2) and the third (TS_3) transition systems are built from TS_1 and TS_2, respectively. Finally, TS_3 is considered as a desirable result.

The main point of the proposed approach is dynamic variation of the window used for deducing states. For this very purpose, two adjustable parameters are involved into the approach. The first one, *Threshold*, affects the size of the

intermediate transition system (TS_2). The second one, *Vwsc*, is a linear factor used for the dynamic calculation of a variable window size while building the resulting model (TS_3). Each step of the algorithm along with both parameters is thoroughly discussed in the following sections.

4.1 Constructing a Full Transition System (Step 1)

The first step of the approach is to construct a full transition system and define a special labeling function mapping every transition to a natural number that determines its *frequency characteristic*.

Definition 7 (Frequency characteristic). *Let $L \in \mathcal{P}(A^*)$be a log over set of activities A and let $TS(L) = (S, E, T, s_0, AS)$ be a full transition system for the L. A frequency characteristic of $TS(L)$ is a function $f : T \to \mathbb{N}$ defined for $t = (\sigma[j-1], \sigma(j), \sigma[j])$ as $f(t) = |L'|$, where $L' \subseteq L$ is the maximum subset of L, and $\forall \sigma' \in L' \quad \exists l \in \mathbb{N}, l > 0 : \sigma'[l-1] = \sigma[j-1], \sigma'(l) = \sigma(j), \sigma'[l] = \sigma[j]$.*

Note that for $l = 1$, $\sigma(0) = \langle\rangle = s_0$ by (1).

Frequency characteristic determines for every transition t a number of traces in log L that start with prefixes $\sigma[j]$. The entire procedure of building a full transition system is presented in Algorithm 1.

We denote a full transition system for a given log L as $TS_1(L) = TS(L)$. $TS_1(L_1)$ built for log L_1 with function f is depicted on Figure 1b; it is a tree by the construction. It is easy to see that fitness of the full transition system ($TS_1(L)$) is perfect (equals 1). This is inherent in the algorithm since it builds for each trace in a log a full chain of states following one after another that corresponds to a sequence of events in the trace.

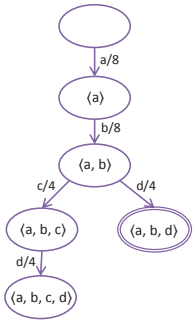


Figure 2: $TS_2(L_1)$ (condensed) built from $TS_1(L_1)$ with $Threshold = 0.33$

4.2 Constructing a Condensed Transition System (Step 2)

The second step of our approach involves cutting some branches of the full transition system with frequency values less than a *cutting threshold* parameter; we refer to it as f_1. Having f_1 set, we can exclude from a model all the states and transitions that correspond to behavior in the event log which is rarely observed. This results in simplifying the tree structure and reduction of the number of states and transitions.

Definition 8 (Condensed Transition System). *Let $TS_1(L) = (S_1, E_1, T_1, s_0, AS_1)$ be a full transition system constructed for log L and let f be a frequency characteristic. The Threshold is a real number from $[0; 1]$ determining a cutting threshold f_1 as follows: $f_1 = round(|L| \cdot Threshold) - 1$. The value $f_1 + 1 = round(|L| \cdot Threshold)$ is a* minimum preserved frequency *for $TS_1(L)$.*

Algorithm 1: Building a full transition system for a given log L

Input : an event log L
Output : a full transition system; $TS_1(L) = (S, E, T, s_0, AS)$; f is a frequency characteristic;
begin
  $S \leftarrow \{s_0\}$;
  for $\sigma \in L$ **do**
    $s \leftarrow s_0$;
    for $i \leftarrow 1$ **to** $|\sigma|$ **do**
      $s' \leftarrow \sigma[i]$;
      $t \leftarrow (s, \sigma(i), s')$;
      if $t \notin T$ **then**
        $T \leftarrow T \cup \{t\}$;
        $f(t) \leftarrow 1$;
      else
        $f(t) \leftarrow f(t) + 1$;
      $S \leftarrow S \cup \{s'\}$;
      $E \leftarrow E \cup \{\sigma(i)\}$;
      if $i = |\sigma|$ **then**
        $AS \leftarrow AS \cup \{s\}$;
      $s \leftarrow s'$;

Algorithm 2: Building a condensed transition system $TS_2(L)$

Input : an event log L; a full transition system $TS_1(L) = (S_1, E_1, T_1, s_0, AS_1)$; f is a frequency characteristic; $Threshold$ is a real number determining a cutting threshold and a minimum preserved frequency;
Output : $TS_2(L) = (S_2, E_2, T_2, s_0, AS_2)$ is a condensed transition system;
begin
  $f_1 = round(|L| \cdot Threshold) - 1$;
  for $t \in T_1$ **do**
    if $f(t) > f_1$ **then**
      $T_2 \leftarrow T_2 \cup \{t\}$;
  $S_2 \leftarrow \{s_0\}$;
  for $t = (s, a, s') \in T_2$ **do**
    $S_2 \leftarrow S_2 \cup \{s'\}$;
  $E_2 \leftarrow E_1$;

A condensed transition system $TS_2(L)$ *built for* $TS_1(L)$ *with function* f *and a given cutting threshold* f_1 *is a transition system* $TS_2(L) = (S_2, E_2, T_2, s_0, AS_2)$, *where* $S_2 \subseteq S_1$, $E_2 = E_1$,$T_2 \subseteq T_1$, $AS_2 \subseteq AS_1$ *and* $T_2 = \{t \mid t \in T_1 \ \& \ f(t) > f_1\}$, $S_2 = \{s_0\} \cup \{s \mid s \in S_1 \ \& \ \exists t = (s', a, s) \in T_2\}$, $AS_2 = AS_1 \cap S_2$.

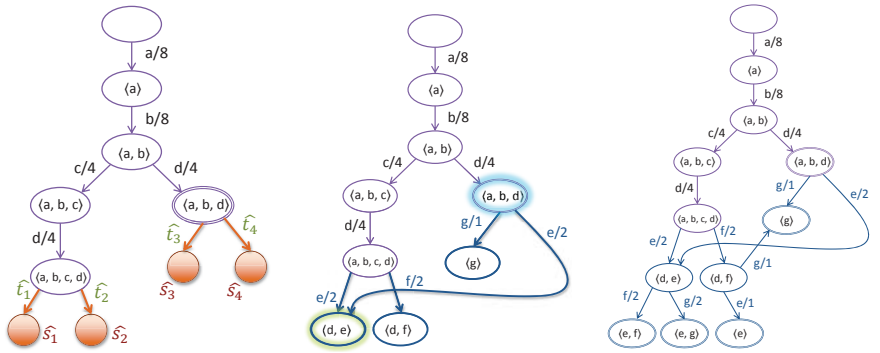
Note that the frequencies of transitions diminish on the way to the leaves of $TS_1(L)$ and $TS_2(L)$. Hence, the exclusion of transition t_k from a $TS_1(L)$ implies the exclusion of a total subtree that has state s_k as a root. Thus, $TS_2(L)$ obtained as a result of cutting with a given threshold, cannot be disconnected.

The entire procedure of construction a condensed transition system is presented in Algorithm 2. For log L_1 with the size $|L| = 8$ and $Threshold = 0.33$, we have $f_1 = 2$. A $TS_2(L)$ built for the log L_1 and $f_1 = 2$ is depicted in Figure 2. It is easy to see that not all the traces from the log can be replayed on $TS_2(L)$ as its fitness is not perfect. Therefore, we cannot consider this model as a final result.

4.3 Constructing a Reduced Transition System (Step 3)

In this section, we propose an approach to convert $TS_2(L)$ to a model with perfect fitness and a size that it less than the size of $TS_1(L)$.

Our proposal is to construct a new transition system $TS_3(L)$ based on $TS_2(L)$ by adding missing states and transitions in order to fully replay all the traces.



(a) (b) (c)

Figure 3: (a) TS_3 under a restoring algorithm: temporary states and transitions after *the first stage*; (b) restored states and transitions after *the second stage*; (c) $TS_3(L_1)$ (reduced) built from $TS_2(L_1)$ with $Threshold = 0.33$ for log L_1

Unlike building the full transition system, in this case we use partial subtrace $\sigma[i, k]$ for representing newly added states of $TS_3(L)$. The important point here is that parameter k is proportional to the frequency of a corresponding input transition.

The algorithm implementing the proposed approach includes a few steps performed iteratively. Its main part is represented in Algorithm 5. In the beginning, we create a copy of $TS_2(L)$, which is denoted as $TS_3(L)$.Then, we try to replay all the traces σ from event log L until there is no trace that cannot be fully replayed in the transition system under construction. Along with replay, we build on additional elements of $TS_3(L)$ such that an increasing number of traces can be replayed.

Coming back to the example with log L_1, we consider transition system $TS_3(L_1)$ copied from $TS_2(L_1)$ depicted in Figure 2. We try to replay log L_1 on it and perform its transformation. Let $\sigma = \langle a, b, c, d, e, f \rangle$. The only prefix of σ that can be successfully replayed is $\sigma^+(TS_3(L_1)) = \langle a, b, c, d \rangle$. Correspondingly, the unreplayable suffix of σ is $\sigma^-(TS_3(L_1)) = \langle e, f \rangle$.

Algorithm 3 replays a single trace σ and also gets as its input transition system $TS_3(L)$ and a frequency characteristic f (discussed above). Moreover, it uses a special function ξ, which maps every trace σ onto number j that determines element $\sigma(j)$ splitting σ into σ^+ and σ^-, $\sigma(j) \in \sigma^-$, and set TT of temporary transitions. The algorithm starts with initial state s_0 as a current state s and the first element $\sigma(1)$ of a trace as a current element $\sigma(i)$. Then it tries to find an appropriate transition t starting with current state s and marked by symbol $\sigma(i)$.

We use an example in Figure 3 to illustrate the proposed approach. Next, there are three possible cases. In the first one (for instance, during the replay $\sigma = \langle a, b, d \rangle$), $t = (s, \sigma(i), s')$ exists with some $s' \in S_2$ and it is a regular one ($t \notin TT$). In this case current state s is changed to s' and the next element

$\sigma(i+1)$ is processed. In the second case (if $\sigma = \langle a, b, d, g \rangle$), t does not exist. For that case a new temporary transition $t = (s, \sigma(i), \widehat{s})$ is added to both the set of transitions and the set of temporary transitions. The end state $\widehat{s}$ is a special temporary state too. It is not marked by any substring and is unique for any temporary transition. For newly added transition t frequency characteristic f is defined to be equal to 1; It means transition t "fired" only once. In the third case, t exists and it is a temporary one ($t \in TT$). This is the case when transition t "fires" one more time; hence, its frequency should be increased by one.

In both the second and the third cases replaying of the current trace σ is broken and $\xi(\sigma)$ is set to the position of the first unreplayable element. This is the reason why the temporary state $\widehat{s}$ cannot still be marked with any subtrace and, consequently, one cannot guarantee that $\widehat{s}$ will be preserved in $TS_3(L)$.

This way, at each iteration Algorithm 5 tries to replay as many traces from the log as possible. For each state the first unreplayable element is determined and a new temporary transition for the element along with a temporary state are built. $TS_3(L_1)$ with temporary transitions and states marked by symbols e, f, g and e are depicted in Figure 3a. Note, the total frequency for all temporary transitions is equal to a number of traces that were not able to be replayed at the first iteration.

Trace $\sigma_5 = \langle a, b, d \rangle$ from log L_1 is an example of a trace that can be replayed at the very first iteration. Once all the traces from the log can be replayed, the reconstruction of $TS_3(L)$ is successfully ended.

As long as there is at least one temporary transition/state in $TS_3(L)$, it has to be converted to a regular one. It is done by Algorithm 4, which enumerates all uncompleted traces. For each such trace σ, the last regular state s, temporary transition t and temporary state $\widehat{s}$ are obtained; they correspond to the first unplayable element $\sigma(\xi(\sigma))$. Then state $\widehat{s}$ is converted to a regular state by being marked with subtrace $\sigma[\xi(\sigma), m]$ of trace σ ended by element $\sigma(\xi(\sigma))$ with length m that *is proportional to the frequency* of temporary transition t. Furthermore, the dedicated state s_{0ws} is used for the case of *zero window size*, which accumulates all rare behavior. This results in the *shortest subtrace* marking such a state than its counterpart in the full transition system. Note that temporary states and transitions are always converted to regular ones.

The latter results in a probability of coinciding numbers of previously distinguishable states. In such a case, all matching states are merged and the total number of states and transitions in the resulting transition system is decreased. Figure 3b shows how two states marked with subtrace $\langle d, e \rangle$ are merged to one state.

After the algorithm is finished, no temporary transitions and states are present in $TS_3(L)$ anymore. Moreover, all previously unreplayable elements in the traces can now be replayed in $TS_3(L)$ as new states for them have been established. At the end of algorithm's iteration each trace from the log can be replayed at least by one more element than before the iteration. Since the length of each trace is finite, the number of algorithm's iterations is also finite. Thus, the algorithm finally stops. The resulting $TS_3(L_1)$ is depicted in Figure 3c.

Algorithm 3: Function `ReplayTrace` of the algorithm of building a reduced transition system $TS_3(L)$

Input : trace σ;
reduced transition system $TS_3(L) = (S_3, E_3, T_3, s_0, AS_3)$; frequency characteristic f; set of completely replayed traces $CompleteTraces \subseteq L$; function ξ mapping each trace to a number of the first unreplayable symbol ; set of temporary transitions TT;
Output : *true*, if a trace is replayed completely, *false* otherwise

Function `ReplayTrace`(σ, $TS_3(L)$, f, $CompleteTraces$, ξ, TT)*: Boolean*

```
/* Already completed */
if σ ∈ CompleteTraces
then
    return true;
s ← s_0;
for i ← 1 to |σ| do
    if ∃s' : t = (s, σ(i), s') ∈ T_3 then
        if s' = ŝ then
            f(t) ← f(t) + 1;
            ξ(σ) = i;
            return false;
        s ← s';
    else
        S_3 ← S_3 ⋃ {ŝ};
        t ← (s, σ(i), ŝ);
        T_3 ← T_3 ⋃ {t};
        TT ← TT ⋃ {t};
        f(t) = 1;
        ξ(σ) = i;
        return false;
/* Trace's complete */
ξ(σ) = |σ| + 1;
return true;
```

Algorithm 4: Procedure `RestateTS` of the algorithm of building a reduced transition system $TS_3(L)$

Input : log L;
reduced transition system $TS_3(L) = (S_3, E_3, T_3, s_0, AS_3)$; frequency characteristic f; set of completely replayed traces $CompleteTraces \subseteq L$; function ξ mapping each trace to a number of the first unreplayable symbol; multiplicative factor for fixed window size $Vwsc \in \mathbb{R}$; set of temporary transitions TT;

```
/* Converts temporaries */
Procedure RestateTS(L, TS_3(L), f,
CompleteTraces, ξ, Vwsc, TT)
    for σ ∈ L do
        i ← ξ(σ);
        /* If the trace is already
        complete */
        if i = |σ| + 1 then
            return;
        s ← σ[i − 1];
        t ← (s, σ(i), ŝ);
        if t ∉ TT then
            return;
        maxWndSize ← max_σ(|σ|);
        wndSize ← round(maxWndSize ·
        f(t) · Vwsc ÷ |L|);
        /* a special 'trash' state */
        if wndSize = 0 then
            s' ← s_0ws;
        else
            s' ← σ[i, wndSize];
        t' ← (s, σ(i), s');
        /* Replace the temporary
        transition and the state by
        regular ones */
        S_3 ← S_3 \ {ŝ} ∪ {s'};
        T_3 ← T_3 \ {t} ∪ {t'};
        TT ← TT \ {t};
        f(t') ← f(t);
        if i = |σ| then
            AS_3 ← AS_3 ∪ {s'};
    return;
```

Algorithm 5: Building a reduced transition system $TS_3(L)$

Input : an event log L;
a condensed transition system $TS_2(L) = (S_2, E_2, T_2, s_0, AS_2)$; a frequency characteristic f; a multiplicative factor $Vwsc \in \mathbb{R}$ for fixed window size;
Output : a reduced transition system $TS_3(L) = (S_3, E_3, T_3, s_0, AS_3)$;
Data: a set of completely replayed traces $CompleteTraces \subseteq L$;
a function mapping each trace to a number of first unreplayable symbol ξ;
a set of temporary transitions TT;

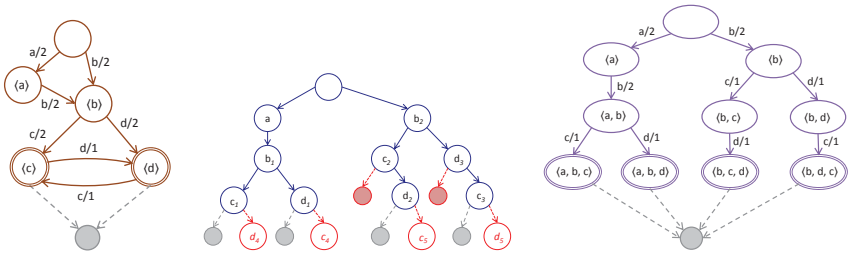
```
/* Main part of the algorithm                                    */
begin
    TS_3(L) ← TS_2(L);
    repeat
        unreplayableTraces ← false;
        for σ ∈ L do
            if ReplayTrace(σ, TS_3(L), f, CompleteTraces, ξ, TT) = false
            then
                unreplayableTraces ← true;
        if unreplayableTraces = true then
            RestateTS (L, TS_3(L), f, CompleteTraces, ξ, Vwsc, TT);
    until unreplayableTraces = false;
```

In Algorithm 5 *Vwsc* is an additional parameter used to combine a varying window size approach with a classical fixed window size approach. It is a real value from $[0, 1]$ determining a maximum size of a state window during the reconstruction phase of $TS_3(L)$.

Finally, considering fitness of reduced transition system $TS_3(L)$ one can postulate the following proposition.

Theorem 1. *Let L be a log and let $TS_3(L)$ be a reduced transition system based on condensed transition system $TS_2(L)$. Then, $TS_3(L)$ perfectly fits L.*

Proof. Let $\sigma = \langle a_1, a_2, ..., a_n \rangle \in A^*$ be a trace of log L and $\sigma = \sigma^+(TS_2(L)) + \sigma^-(TS_2(L))$, where $\sigma^+(TS_2(L))$ is a trace prefix that can be "replayed" on $TS_2(L)$ and $\sigma^-(TS_2(L))$ is a trace suffix that cannot be "replayed" on $TS_2(L)$. $\sigma^+(TS_2(L))$ can be "replayed" on $TS_3(L)$ by the construction. Now we need to prove that the entire sequence σ can be "replayed" on $TS_3(L)$. We will prove that iteratively for sequences $\langle \sigma(1), ..., \sigma(i) \rangle$, where i varies from $|\sigma^+(TS_2(L))|$ to $|\sigma|$. Basis of induction: the proposition valid for $i = |\sigma^+(TS_2(L))|$, since $\sigma^+(TS_2(L))$ can be "replayed" on $TS_3(L)$. Step of induction: the trace $\langle \sigma(1), ..., \sigma(i) \rangle$ can be "replayed". Now let us prove that the trace $\langle \sigma(1), ..., \sigma(i+1) \rangle$ can be "replayed" as well. According to Algorithms 3 and 4, we "replay" $\langle \sigma(1), ..., \sigma(i) \rangle$ and add a new state $\widehat{s}$ (if it has not been added previously) and a new edge $t = (s, \sigma(i+1), \widehat{s})$ correspondingly. Thus, trace $\langle \sigma(1), ..., \sigma(i+1) \rangle$ now can be replayed, and that proves the step of induction. □



(a) (b) (c)

Figure 4: Transition systems built for log L_2: (a) 1-window size (fixed) $TS_f(L_2)$; (b) unfolding graph obtained after $TS_f(L_2)$ simulated $TS_1(L_2)$; (c) full $TS_1(L_2)$

5 Evaluation and Discussion

In this section we evaluate the proposed approach on real-life event logs. In the beginning of the section, an algorithm for precision calculation is introduced.

5.1 Metrics Calculation

Calculation of metrics for all three transition systems is performed throughout their building.

As we have shown above, *fitness* of $TS_1(L)$ and $TS_3(L)$ is perfect. Further, *simplicity* of a model is easily calculated on the basis of the number of model's elements.

We propose an algorithm calculating *precision* metrics for a given transition system $TS(L)$ based on an idea of simulation [12]. The algorithm assumes that $TS(L)$ perfectly fits log L. Suppose that $TS(L)$ can simulate $TS_1(L)$. All extra behavior observed for $TS(L)$ is penalized. Finally, a normalized total penalty forms the basis of a precision value.

The approach for calculation of precision is described in Algorithm 6. The algorithm consists of two main steps. First, the algorithm iteratively calculates so-called *partial precisions* for every state in $TS(L)$ (Algorithm 9). Second, the algorithm sums partial precisions (Algorithm 7) and calculates the average over the number of states.

To get an illustration of this idea, consider a following log:

$$L_2 = \{\langle a, b, c\rangle, \langle a, b, d\rangle, \langle b, c, d\rangle, \langle b, d, c\rangle\}$$

Full transition system $TS_1(L_2)$ for the log is depicted in Figure 4c. It is considered to be the *most precise reference model*. Transition system $TS_f(L_2)$ in Figure 4a is built with a fixed window size (equal to 1). As a more general model than $TS_1(L_2)$ it allows more behavior than the reference model. All extra behavior is counted as *penalty* and influence precision.

Precision calculation invokes simulation routine implemented as a recursive procedure (Algorithm 8). For the example above, $TS(L) = TS_f(L_2)$ and

Algorithm 6: Calculating precision for transition system $TS(L)$

Input : general (perfectly fit) transition system $TS(L) = (S, E, T, s_0, AS)$ built for log L; reference full transition system $TS_1(L) = (S_1, E_1, T_1, s_{01}, AS_1)$;
Output : value of precision $Prec(TS(L))$ for $TS(L)$;
Data: partial function η mapping each state of $TS(L)$ to a real number determining the state's "partial precision"; partial function θ mapping each state $s \in TS(L)$ to a natural number determining how many times state s has been visited;

```
/* Main part of the algorithm  */
begin
    CalcStatePrecision (s_0, s_01,
    TS(L), TS_1(L), η, θ);
    Prec(TS(L)) ←SumPartialPrecisions
    (TS(L), η);
```

Algorithm 7: Function `SumPartialPrecisions` calculating total precision of $TS(L)$

Input : general (perfectly fit) transition system $TS(L) = (S, E, T, s_0, AS)$ built for log L; function η mapping each state of $TS(L)$ to a real number determining state's "partial precision";
Output : value of precision $Prec(TS(L))$ for $TS(L)$;

Function `SumPartialPrecisions(`$TS(L)$, η`)`: *Real*

```
    sum ← 0;
    for s ∈ S do
        sum ← sum + η(s);
    res ← sum/|S|;
    return res;
```

$TS_1(L) = TS_1(L_2)$. Initially $s = s_0 \in TS(L)$ and $s_1 = s_{01} \in TS_1(l)$ are passed as parameters to the procedure. For each output transition of a current state s, the algorithm tries to find a transition labeled with the same event among output transitions of a current state s_1. It is possible to have not more than one transition labeled with the same event, since all transition systems considered in the paper are deterministic by the construction.

For example, consider transition $(s_0, a, \langle a \rangle)$ in $TS_f(L_2)$. It has a matching transition $(s_0, a, \langle a \rangle)$ in $TS_1(L_2)$. The procedure is recursively called with $s = \langle a \rangle$ and $s_1 = \langle a \rangle$. This step also produces an input edge to vertex a in an unfolding graph depicted on Figure 4b. This step has to be repeated for transitions $(\langle a \rangle), b, \langle b \rangle)$ and $(\langle a \rangle), b, \langle a, b \rangle)$ (obtains b_1 in the unfolding graph) and transitions $(\langle b \rangle), c, \langle c \rangle)$ and $(\langle a, b \rangle), c, \langle a, b, c \rangle)$ (obtains c_1 in the unfolding graph). Here, $\langle c \rangle$ in $TS_f(L_2)$ is an accepting state; it contains a virtual output transition (depicted as a gray dashed arrow) to a virtual final state. Similarly, $\langle a, b, c \rangle$ from $TS_1(L_2)$ also has a virtual final transition, which maintains balance (depicted as a dashed output edge in the unfolding graph).

Together with that, $\langle c \rangle$ has transition $(\langle c \rangle), d, \langle d \rangle)$ to state $\langle d \rangle$, which has no counterpart in $TS_1(L_2)$ (edge (c_1, d_4) in the graph). The algorithm penalizes this extra transition and calculates partial precision for state $\langle c \rangle$ as a difference between a number of output transitions and a number of penalized output transitions divided by a total number of output transitions.

Algorithm 8: Procedure `CalcStatePrecision` calculating "partial precision for entire states" as function η

Input : current state s of $TS(L)$;
current state s_1 of $TS_1(L)$; general (perfectly fit) transition system $TS(L) = (S, E, T, s_0, AS)$ built for log L; reference full transition system $TS_1(L) = (S_1, E_1, T_1, s_{01}, AS_1)$; partial function η mapping each state of $TS(L)$ to a real number determining the state's "partial precision"; partial function θ mapping each state $s \in TS(L)$ to a natural number determining how many times state s has been visited;

Procedure `CalcStatePrecision`(s, s_1, $TS(L)$, $TS_1(L)$, η, θ)

```
pen ← 0;
for t = (s, a, s') ∈ s• do
    /* If no matching trans.                                    */
    if ∃t_1 = (s_1, a, s'_1) ∈ TS_1(L) then
        CalcStatePrecision (s', s'_1, TS(L), TS_1(L), η);
    else
        pen ← pen + 1;
/* Number of output trans-s                                     */
otn ← |s • |;
if s ∈ AS then
    otn ← otn + 1
    if s_1 ∉ AS_1 then
        pen ← pen + 1;
if otn ≠ 0 then
    partPartPrec = (otn − pen)/otn;
    RecalcStatePrecision (s, partPartPrec);
```

Algorithm 9: Procedure `RecalcStatePrecision` refines the value of a state's "partial precision"

Input : state $s \in S$ of $TS(L) = (S, E, T, s_0, AS)$;
function η mapping each state of $TS(L)$ to a real number determining the state's "partial precision"; partial function θ mapping each state $s \in S$ to a natural number determining how many times state s has been visited; new state's partial precision *pprec* for refining;

Procedure `RecalcStatePrecision`(s, η, θ, *pprec*)

```
/* If either η or θ is not defined for this state               */
if η(s) is not defined then
    η(s) ← 0;
if θ(s) is not defined then
    θ(s) ← 0;
stPrec ← η(s) · θ(s);
θ(s) ← θ(s) + 1;
η(s) ← (stPrec + pprec)/θ(s);
```

Since state $\langle a, b, c\rangle$ of $TS_1(L_2)$ does not allow any further moves, the algorithm leaves the current iteration of the procedure and, thereby, returns to a higher level, to state $\langle b\rangle$ of $TS_f(L_2)$ and $\langle a, b\rangle$ of $TS_1(L_2)$. Then, the algorithm repeats the same steps for all unvisited output transitions.

During its work, the algorithm can normally visit some states of a more general model ($TS_f(L_2)$ in the example) more than once. For each such visit a value of partial precision for the state is recalculated (Algorithm 9).

Finally, after all states of the reference model have been visited during the simulation, values of ultimate partial precision for each state of $TS(L)$ are represented by η. The last step is to calculate an average value (Algorithm 7), which is the required value of the model's precision.

5.2 Implementation Details

To evaluate the proposed approach, we have developed a number of routines for the ProM toolkit [16]. The routines are implemented as plug-ins with several entry points intended for different sets of input parameters.

`Build and reduce transition systems (xi)` plug-in combines routines for building $TS_1(L)$, $TS_2(L)$, $TS_3(L)$ for a given input log L, along with calculation metrics for each transition system built. In the simplest case, the plug-in obtains at its input only event log L and provides ability to configure the settings as follows. (1) Specify a maximum window size for building $TS_1(L)$. Default size is unlimited (that is set at a value of -1). By setting the size to a natural number, one can make algorithm act with a fixed window. We use this option for building reference models with fixed windows. (2) Specify a value of *Threshold* parameter used for building $TS_2(L)$. (3) Specify a value of multiplicative factor *Vwsc* used when building $TS_3(L)$.

Once successfully finished, the plug-in produces at its output *three transition systems*, and, what is the most important for analyzing the results, a *hierarchical report*. The report represents a set of characteristics organized in a tree structure. They include metrics for each built transition system and additional attributes calculated during the algorithm's operation. We created a special "view" plug-in for browsing such reports, which allows to export information as a JSON-structure or HTML-formatted text.

The both plug-ins are openly available to download on `http://pais.hse.ru`.

5.3 Experiments and Discussion

We evaluated our approach on a set of event logs, both artificial and real-life. In the following sections we consider "BPI challenge" logs [1] `L3` (11 traces, 89 activities) and `L4` (251 traces, 247 activities). Full transition systems built for the logs have the following frequency characteristics:

- $TS_1(L3)$: Maximum Window Size: 71; 514 states; 513 transitions;
- $TS_1(L4)$: Maximum Window Size: 83; 8088 states; 8087 transitions.

Our goal is to compare metrics of models built with an existing fixed window algorithm and models built with the algorithms proposed in this paper.

Table 1: Dependence of model's metrics on parameters *Threshold* and *Vwsc* with an unlimited window size

Settings		TS_2(*L3*): Condensed				TS_3(*L3*): Reduced				TS_2(*L4*): Condensed				TS_3(*L4*): Reduced			
Thres-hold	VWSC	TThr	States #	Transs #	Act #	States #	Transs #	Simpl	Prec	TThr	States #	Transs #	Act #	States #	Transs #	Simpl	Prec
0	1	1	514	513	89	514	513	0,0876	1	0	8088	8087	247	8088	8087	0,0153	1
0,05	1	1	514	513	89	514	513	0,0876	1	13	37	36	20	320	1573	0,131	0,567
0,1	1	1	514	513	89	514	513	0,0876	1	25	12	11	10	327	1595	0,129	0,5482
0,25	1	3	11	10	9	470	478	0,0949	0,9911	63	9	8	8	320	1531	0,134	0,5585
0,33	1	4	9	8	8	464	474	0,0959	0,9905	83	9	8	8	320	1531	0,134	0,5585
0,5	1	6	7	6	6	470	478	0,0949	0,9911	126	6	5	5	308	1540	0,1342	0,5523
0,65	1	7	6	6	6	470	478	0,0949	0,9911	163	3	2	2	289	1490	0,1394	0,5595
0,75	1	8	7	6	6	470	478	0,0949	0,9911	188	3	2	2	289	1490	0,1394	0,5595
0,85	1	9	7	6	6	470	478	0,0949	0,9911	213	3	2	2	289	1490	0,1394	0,5595
0,95	1	10	7	6	6	470	478	0,0949	0,9911	238	2	1	1	308	1544	0,1339	0,5581
1	1	11	1	0	0	470	478	0,0949	0,9911	251	2	1	1	308	1544	0,1339	0,5581
0,25	0,5	3	11	10	9	397	438	0,1078	0,9505	63	9	8	8	173	992	0,2129	0,5514
0,33	0,5	4	9	8	8	395	438	0,108	0,9477	83	9	8	8	173	992	0,2129	0,5514
0,5	0,5	6	7	6	6	397	438	0,1078	0,9505	126	6	5	5	147	869	0,2441	0,556
0,75	0,5	8	7	6	6	397	438	0,1078	0,9505	188	3	2	2	141	866	0,2463	0,5809
0,25	0,25	3	11	10	9	312	403	0,1259	0,8764	63	9	8	8	73	645	0,3454	0,5116
0,33	0,25	4	9	8	8	311	405	0,1257	0,8739	83	9	8	8	73	645	0,3454	0,5116
0,5	0,25	6	7	6	6	309	400	0,1269	0,8771	126	6	5	5	78	639	0,3459	0,5387
0,75	0,25	8	7	6	6	309	400	0,1269	0,8771	188	3	2	2	86	709	0,3119	0,4843
0,33	**0,12**	4	9	8	8	139	320	**0,1961**	**0,7012**	83	9	8	8	59	565	**0,3974**	**0,5333**
0,33	**0,05**	4	9	8	8	55	201	**0,3516**	**0,676**	83	9	8	8	26	404	**0,5767**	**0,5266**

Table 2: Metrics of k-window models

k (wnd size)	L3				L4			
	States #	Transs #	Simpl	Prec	States #	Transs #	Simpl	Prec
1	90	273	0,2479	0,6117	248	1755	0,124	0,3791
2	274	371	0,1395	0,8574	1756	3602	0,046	0,7607
3	372	418	0,1139	0,9457	3603	4838	0,029	0,8892
4	419	437	0,1051	0,9828	4839	5600	0,024	0,946
5	438	451	0,1012	0,9882	5601	6129	0,021	0,9671
7	464	474	0,0959	0,9914	6515	6806	0,019	0,9836
10	492	497	0,091	0,9961	7228	7392	0,017	0,9929
15	508	508	0,0886	0,999	7840	7905	0,016	0,9968
20	513	513	0,0877	0,999	8037	8054	0,015	0,9991

Table 1 shows metrics of condensed and reduced models constructed with unlimited window size. Metrics of k-window models for the logs are presented in Table 2. Comparing the results from both tables, one can consider the following outcome. By using a *fixed window* approach presented in [2] the maximum reduction is reached for a parameter $k = 1$. This is the smallest model that can be built by the algorithm. Moreover, Tables 2 shows that increasing windows size (k) to a value greater than 7 does not significantly impact simplicity and precision.

In our algorithms, there are two parameters affecting a model size: *Threshold* and *Vwsc*. Parameter *Threshold* has a limited impact on the model size. As it is shown in Table 1, dependence of the model size from *Threshold* is nonlinear in the entire domain of *Threshold*. For log $L4$, better simplicity results are in the vicinity of the points 0.2 and 0.8 and worse in the range ends. This is because in the case of dismissive small *Threshold* model $TS_2(L)$ is similar to $TS_1(L)$; so, application area of a variable size window lies near the tree leaves. In such a case, noticeable reduction of a model is achievable only for traces with similar suffixes.

Conversely, the closer a value of *Threshold* to 1, the lesser $TS_2(L)$ is built. Consequently, at the stage of building $TS_3(L)$ model most of it is to be reconstructed. Preliminary experiments show that selecting *Threshold* from a range $[0.2; 0.8]$ leads to better results; nevertheless, further elaboration of the parameter's impact is needed.

By decreasing the value of *Vwsc* parameter from 1 to 0, *significant reduction of a resulting model size* is achieved (last two rows of Table 1). For example, for log $L3$ and a value of $Vwsc = 0.05$ we have $Simpl(TS_3(L3)) = 0.3516$ and $Prec(TS_3(L3)) = 0.676$ versus $Simpl(TS_f(L3)) = 0.2479$ and $Prec(TS_f(L3)) = 0.6117$ for the case of one window size model. Generally, by adjusting the value of *Vwsc*, one can obtain a resulting model in a wide range of sizes. Unlike parameter *Threshold*, parameter *Vwsc* gives a linear dependence of the model size. Moreover, by varying values both of *Threshold* and *Vwsc* it is possible to enhance mutual influence of the parameters on each other. That allows flexible balancing between precision and simplicity.

6 Conclusion

This paper presented a new approach for reducing transition systems, based on an inference algorithm with a varied window size. In contrast to the existing approaches the approach presented in this paper shows advantages of flexible adjustment of a resulting model size. This way, estimation of achievability of the main goal is made on the basis of numerical characteristics of resulting models, both absolute and integral metrics. Future work is aimed at further investigation of impacts of various algorithm coefficients on time costs of a region-based algorithm applied to resulting transition systems. Moreover, we plan to investigate the quality metrics of Petri nets obtained as an outcome of the algorithm.

References

1. http://fluxicon.com/blog/2015/05/bpi-challenge-2015/
2. van der Aalst, W.M.P., Rubin, V., Verbeek, H.M.W., Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Software & Systems Modeling 9(1), 87–111 (2008), http://dx.doi.org/10.1007/s10270-008-0106-z
3. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer (2011)

4. Angluin, D.: Inference of reversible languages. J. ACM 29(3), 741–765 (Jul 1982), http://doi.acm.org/10.1145/322326.322334
5. Badouel, E., Darondeau, P.: Lectures on Petri Nets I: Basic Models: Advances in Petri Nets, chap. Theory of regions, pp. 529–586. Springer Berlin Heidelberg, Berlin, Heidelberg (1998), http://dx.doi.org/10.1007/3-540-65306-6_22
6. Biermann, A.W., Feldman, J.A.: On the synthesis of finite-state machines from samples of their behavior. IEEE Trans. Comput. 21(6), 592–597 (Jun 1972), http://dx.doi.org/10.1109/TC.1972.5009015
7. Buijs, J., Dongen, B., Aalst, W.: On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. In: Meersman, R., Rinderle, S., Dadam, P., Zhou, X. (eds.) OTM Federated Conferences, 20th International Conference on Cooperative Information Systems (CoopIS 2012). Lecture Notes in Computer Science, vol. 7565, pp. 305–322. Springer-Verlag, Berlin (2012)
8. Carmona, J., Cortadella, J., Kishinevsky, M.: A region-based algorithm for discovering petri nets from event logs. In: BPM. pp. 358–373 (2008)
9. Cook, J.E., Wolf, A.L.: Discovering models of software processes from event-based data. ACM Trans. Softw. Eng. Methodol. 7(3), 215–249 (Jul 1998), http://doi.acm.org/10.1145/287000.287001
10. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving petri nets from finite transition systems. IEEE Trans. Comput. 47(8), 859–882 (Aug 1998), http://dx.doi.org/10.1109/12.707587
11. Lorenzoli, D., Mariani, L., Pezzè, M.: Inferring state-based behavior models. In: 4th International Workshop on Dynamic Analysis (WODA 2006) co-located with the 28th International Conference on Software Engineering (ICSE 2006). pp. 25–32. ACM Press (2006)
12. Park, D.M.R.: Concurrency and automata on infinite sequences. In: Theoretical Computer Science, 5th GI-Conference, Karlsruhe, Germany, March 23-25, 1981, Proceedings. pp. 167–183 (1981), http://dx.doi.org/10.1007/BFb0017309
13. Rubin, V., Dongen, B.F.V., Kindler, E., Günther, C.W.: Process mining: A two-step approach using transition systems and regions. Tech. rep., BPM Center Report BPM-06-30, BPM Center (2006)
14. Shershakov, S.A., Lomazova, I.A., Kalenkova, A.A.: Method for transition systems reduction based on state frequencies (preliminary version). Tech. rep., Higher School of Economics (2016), http://dx.doi.org/10.13140/RG.2.1.3438.6328
15. Sole, M., Carmona, J.: Region-based foldings in process discovery. IEEE Transactions on Knowledge and Data Engineering 25(1), 192–205 (2013)
16. Verbeek, H., Buijs, J., Dongen, B., Aalst, W.: ProM 6: The Process Mining Toolkit. In: Rosa, M.L. (ed.) Proc. of BPM Demonstration Track 2010. CEUR Workshop Proceedings, vol. 615, pp. 34–39 (2010)