Computing and Informatics, Vol. 43, 2024, 149–180, doi: 10.31577/cai_2024_1_149

ALIGNMENT-BASED CONFORMANCE CHECKING OF HIERARCHICAL PROCESS MODELS

Lu Wang*, Xiao Han

College of Computer Science and Engineering
Shandong University of Science and Technology
Qingdao 266590, China
e-mail: `wanglu253@126.com`

Man Qi

Department of Computing
Canterbury Christ Church University
Canterbury CT1 1QU, UK

Kang Wang

College of Electronic Information Engineering
Shandong University of Science and Technology
Qingdao 266590, China

Peng Li

College of Computer Science and Engineering
Shandong University of Science and Technology
Qingdao 266590, China
&
IT Management Department
The Affiliated Hospital of Qingdao University
Qingdao 266003, China

* Corresponding author

Abstract. Process mining has received much attention in the field of business process management. Event logs that are generated from information systems can be correlated with the process models for conformance checking. The process models describe event activities at an abstraction level. However, hierarchical business processes, as a kind of typical complex process scenario, describe sub-processes invocation and multi-instantiation patterns. As existing conformance, checking approaches cannot identify sub-processes within hierarchical process models, they cannot be used for conformance checking of hierarchical process models. To handle this limitation, a definition of hierarchically alignment sequences is presented in this paper. Meanwhile, a novel conformance checking approach for hierarchical process models and event logs is proposed. The proposed method has been implemented within the ProM toolkit, which is an open-source process mining software. To evaluate the effectiveness of the proposed approach, both artificial and real-world event logs are utilized in a comparative analysis against existing state-of-the-art approaches.

Keywords: Hierarchical process models, Petri nets, event logs, hierarchical alignment trees, conformance checking

1 INTRODUCTION

As a new and emerging research area within the field of business process management [1], process mining aims to build a bridge between traditional model-driven methods and new data-driven methods [2]. Rounded and efficient business process management models can be built by process mining. The main scenarios of process mining include process discovery, conformance checking, and enhancement. Based on event logs, process models can be produced by process discovery. Conformance checking is employed to verify whether the actual behavior recorded in event logs is consistent with the model, and vice versa. The information about the actual process recorded in the event logs is used to extend or improve an existing process model, which is the idea of enhancement. The primary focus of this paper is on conformance checking.

Conformance checking is an important part of process mining, which is used to detect the matching degree between the process models and event logs. To check to what extent the model can replay the logs related to the process. The purpose of conformance checking is to identify potential risks and problems in the business processes, such as violations of laws and regulations, security hazards, operational errors, etc. Conformance checking helps organizations assess the risks and establish a sound management model. The management model can guarantee the compliance and effectiveness of the business processes, and reduce the risks and costs of the organizations. The performance and competitiveness of the organizations can also be improved by conformance checking.

Current process mining techniques generally assume that the events are associated with activities in the process models, with these activities being at the same level of abstraction [3]. Nonetheless, this assumption may not always hold, particularly for complex processes where activities take place at varying levels of abstraction. An example of this is seen in business process outsourcing scenarios, where a company may contract out a portion of its business to another organization, creating a hierarchical relationship in which the outsourced process is considered a sub-process of the original process. The relationships between these activities and events are complex and are not easily identified by the current process mining techniques.

To address this limitation, the concept of hierarchical process models is proposed in [3]. A new process discovery technique is also proposed to mine hierarchical process models by the event logs with a life-cycle. The hierarchical process models can describe the complex relationships between events and activities on different abstraction levels. The sub-processes and multi-instance markers can be represented by hierarchical business processes. Different from the traditional flat process model, the hierarchical process model can divide the models into multiple levels, and each level represents a different abstraction level. The understandability and maintainability of the models can be improved by the hierarchical structure, and the complexity of modeling can also be reduced. Employing hierarchical process models can aid organizations in comprehending and handling intricate business processes. With the discovery approaches of hierarchical process models proposed, the corresponding conformance checking techniques are not mature enough.

To measure the quality of the hierarchical process model, the nesting relationships within the hierarchical model should be mined. For the conformance checking of hierarchical process models, the nesting relationships within different levels should be considered. The relationships between each level need to be identified in the conformance checking. Present conformance checking methods are primarily developed for flat process models and event logs, such as replay, alignment, and footprint comparison, which do not apply to hierarchical process models with hierarchical structures. The concept of hierarchical structure brings new challenges to the conformance checking of hierarchical process models. To address this challenge, a method of conformance checking based on alignment for hierarchical process models is proposed in this paper. The method aims to explore and provide the following contributions.

- First, the method is specifically designed for hierarchical process models.
- Second, the nested relationships between tasks can be distinguished and mined from the hierarchical process model.
- Third, the hierarchical event logs are constructed by the nested relationships of tasks.
- Lastly, conformance checking of hierarchical process models and hierarchical event logs is investigated, to obtain the corresponding alignment sequence tree.

Based on the simulation logs and real logs, the method of conformance checking for the hierarchical models is compared with the current methods for the flat models. Experiments are carried out to demonstrate the effectiveness of the proposed approach for hierarchical process models.

The paper is further structured as follows: Section 2 provides an overview of related works, while Section 3 revisits fundamental concepts such as Petri nets, hierarchical process models, and event logs. In Section 4, we present our proposed conformance checking method for hierarchical process models and hierarchical event logs. The effectiveness of our approach is demonstrated through experiments in Section 5. Finally, Section 6 concludes the paper and discusses potential directions for future work.

2 RELATED WORKS

Conformance checking primarily concerns the quantification of the degree to which the execution sequences of models, as recorded in the logs, match their actual executions. Conformance checking methods detect deviations in execution processes from their prescribed behavior by comparing the actual execution processes against the prescribed ones. Therefore, conformance checking can determine whether organizations are operating as expected. It can also help organizations identify process problems and risks. Thus, organizations can take timely measures for improvement.

Conformance checking is a technique that links process models with process data. A conformance checking method based on event logs is proposed by Rozinat et al. [4]. This method can automatically compare the event logs with the process models to find deviations in the execution. In [5], a conformance checking method based on comparing process models and event logs is proposed. The method transforms the process models into Petri nets and transforms the event logs into "process traces". Several comparisons are made to evaluate the consistency between the actual execution processes and the models. In [6], the explicit and implicit disparities that exist between the process models and the event logs are captured. These differences can be utilized to perform automated analysis and optimization of the processes. An online conformance checking approach is proposed in [7], which can detect in real-time whether the process execution conforms to the prescribed behavior.

Conformance checking techniques include replay, alignment, and footprint comparison. Carmona et al. [8] and van der Aalst [9] describe two conformance checking approaches, namely log replay and trace alignment. The recorded event logs and the simulated processes of process models are used in log replay. The differences between the simulated processes and the actual processes are used to analyze the conformance and quality of the process models. In token-based log replay [10], the remaining tokens in the model after replay are aggregated. The conformance of the process model is determined by the sum of all redundant and generated tokens. The differences between the actual execution traces and the expected traces are compared in alignment. The differences are used to identify the specific deviations and anomalies. Most of the literature uses trace alignment algorithms to assess the conformance of process models. Therefore, trace alignment algorithms are considered as the present standard for conformance checking techniques [11]. A similar framework of alignment is proposed by Bose et al. [12] and Prabhakara et al. [13]. The differences between the models and the executed processes are detected by aligning the process models with the event logs of the actual execution in this framework. The optimal alignment of the model to the logs can be calculated by extending the basic alignment method [11].

Finding the optimal alignment [14] between models and traces is essentially an optimization problem. By using the A* algorithm, the task of computing an optimal alignment between a model and a trace can be converted to solving the shortest path problem [15]. In addition to the method of obtaining the optimal alignment by calculating the cost function, there is also an alignment algorithm based on insertion planning [16]. The alignment process is transformed into a planning problem. To find the optimal alignment, the insertion planning language is used to define the planning domain. To find the optimal alignment, an integer linear programming algorithm is proposed in [17]. The alignment cost function is expanded to incorporate dimensions such as time, data, and resources.

A decomposition technique for alignment is proposed in [18], which can reduce the calculation time. Petri net decomposition is used to decompose the process models into subnets [19]. The subnets are aligned with the corresponding traces in the event logs. The concept of single-entry, single-exist (SESE) [20] is used in Petri net decomposition. Complex event logs can be decomposed into simple sublogs by SESE. Thus, the conformance checking of large event logs and large process models can be handled. An alignment method based on heuristic is proposed by Song et al. [21]. This method can automatically identify the defects and discrepancies of industrial-scale process models. for repairing industrial-scale process design models. The defects and discrepancies can be used to repair the models. Although most alignment methods are dedicated to process modeling languages, declarative process modeling languages [22] can also be used to create alignments. In [23], an aligning method for event logs and declarative models is proposed. The location and severity of the deviations can be diagnosed by this method. Existing conformance checking approaches are defined on the flat process models, which are not suitable for hierarchical models with sub-process.

For the hierarchical process model, the conformance checking method Acorn based on BPMN is proposed in [24]. The semantics of complex patterns such as sub-processes and multi-instances are analyzed based on BPMN [25]. The alignment algorithm is designed and optimized. And the calculation method of fitness is also given in the final. A method to transform hierarchical models with sub-processes into flat models is presented in [26]. However, this transformation rule does not apply to event logs with multi-instance behavior in sub-processes. An approach to transforming hierarchical models with multi-instances into flat models is proposed in [27]. Thus, the current methods can be employed to assess the quality of models.

Usually, the event logs and models are large in reality. The transformation methods for conformance checking consume a lot of time and memory. Moreover, the existing conformance checking methods of the hierarchical process models are either only for special process models [24] or transform the hierarchical process model into a flat model and then apply the existing conformance checking approaches to flat process models [26, 27]. There is no approach to obtaining the conformance of hierarchical process models by analyzing the relationships of the hierarchical structures.

This paper proposes a conformance checking approach for hierarchical process models to address the aforementioned issues. The conformance checking of hierarchical models can be stratified. Then the hierarchical segmented alignment sequences are integrated into an overall alignment sequence. Then the hierarchical process model can be measured by using conformance evaluation indexes of the flat process model. Experimental results show that the method can greatly save time to align logs and models.

3 PRELIMINARIES

This section provides a brief overview of Petri nets and Petri nets with nested transitions, as well as introduces the notations used for event logs.

Definition 1 (Petri net [28])**.** Let $N_e = (P, T; F)$ be a Petri net, where

1. P is a finite set of places;
2. T is a finite set of transitions with $P \cup T = \emptyset$ and $P \cap T = \emptyset$; and
3. $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs.

Definition 2 (Labeled Petri net [2])**.** Let $PN = (N_e, A, l)$ be a labeled Petri net, where

1. N_e is a Petri net;
2. A is a finite set of activities; and
3. $l := T \rightarrow A^\tau$ is a function, where $A^\tau = A \cup \tau$ and $\tau \notin A$ represents the labels of all invisible transitions.

Definition 3 (Petri nets with nested transitions [26])**.** Let $PN_N = (PN, N)$ be a Petri net with nested transition, where

1. $PN = (N_e, A, l)$ is a labeled Petri net;
2. $N : T \rightarrow \{A, N\}$ is a nested transition function, where $\forall t \in T, N(t) = A$ is a normal transition, $N(t) = N$ is a nested transition.

PN_N is a Petri net with nested transitions. $T_a = t \in T \mid N(t) = A$ is a set of normal transitions, and $T_n = t \in T \mid N(t) = N$ is a set of nested transitions. T_{a0} is a normal transition in the top Petri net PN_{N0} of PN_N. T_{n0} is a nested transition in the top Petri net PN_{N0} of PN_N. Figure 1 displays the representation of a nested

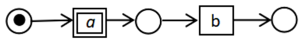


Figure 1. A Petri net with a nested transition pn_n

transition (e.g., a) using a double-line rectangle, and an ordinary transition (e.g., b) using a single-line rectangle.

Definition 4 (Hierarchical process models [26])**.** Let $HPN = (PN_{N0}, HPN(PN_{N0}))$ be a hierarchical process model, where:

1. PN_{N0} is a root node, i.e. the top-level Petri net with nested transitions;
2. $HPN(PN_{N0})$ is the sub-model of PN_{N0}, such that:
   - $HPN(PN_{N0}) = \emptyset$ if $T_{N0} = \emptyset$; otherwise
   - $HPN(PN_{N0}) = \{(t_i, PN_{ni}, HPN(PN_{ni})) \mid t_i \in T_{n0}\}$, where PN_{ni} is a Petri net with nested transitions that are nested by t_i.

An instance of a hierarchical Petri net is illustrated in Figure 2. One nested transition and three normal transitions are contained in the top-level hpn, which refers to a Petri net. To be more precise, a is a nested transition, which refers to PN_{N1}. A nested transition is contained in PN_{N1}, denoted as b, which refers to a sub-process PN_{N2}. As no one nested transition is contained in PN_{N2}, the recursive definition terminates at this level.

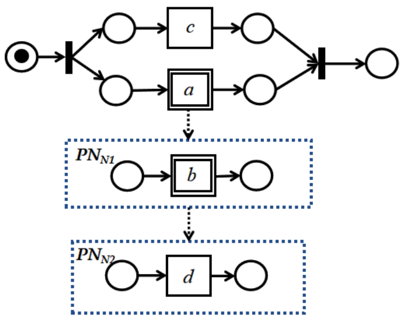


Figure 2. An instance of hierarchical Petri net hpn

Definition 5 (Event, Attribute [2])**.** Let ξ be the event universe, i.e., the set of all possible event identifiers, U_A be the activity universe, and U_T be the time universe, U_L be the transaction type universe. For any event $e \in \xi$, $\sharp n(e)$ is the value of attribute n for event e. The following standard attributes are involved: (1) $\sharp act(e) \in$

U_A is the activity name of e; (2) $\sharp time(e) \in U_T$ is the timestamp of e; (3) $\sharp trans(e) \in U_L$ is the transaction type of e.

In this paper, we solely focus on two types of lifecycles: star and complete.

Definition 6 (Classifier [2])**.** A classifier is a function $C : \xi \longrightarrow U_A \times U_L$ that assigns a representative name to each event for the purpose of analysis. For all events, $e \in \xi$, $C(e) = (\sharp act(e), \sharp trans(e))$, i.e. e represents the name of the event.

In the following, the standard classifier of events is represented as the pair of activity name and transaction type, i.e., $e \in \xi$, $e = C(e) = (\sharp act(e), \sharp trans(e))$. With the help of the classifier, we define an event log as a combination of activity name and lifecycle that represents an event in a simple manner. That is $(b, start)$ can be written as b_s, and $(b, complete)$ can be written as b_c.

Definition 7 (Lifecycle Event Log)**.** A lifecycle event log $L \in \boldsymbol{B}((U_A \times U_L)^*)$ is a multi-set of traces, and a trace $\sigma \in (U_A \times U_L)^*$ is a sequence of activities with lifecycle information.

For example, $L = \{\langle a_s, c_s, a_c, b_s, b_c, c_c \rangle^3\}$ denotes an event log of three traces, and each trace has six events.

Definition 8 (Alignment [29])**.** Let $A \subseteq U_A$ be a set of activities, and $\sigma \in (U_A \times U_L)^*$ be a trace, $HPN = (PN_{N0}, HPN(PN_{N0}))$ be a hierarchical process model. $(e_i, a_i) \in (A^{\gg} \times T^{\gg}) \setminus \{(\gg, \gg)\}$ represents a movement. An alignment, denoted by $\gamma = (e_1, a_1)(e_2, a_2)\dots(e_K, a_K)$ between σ and HPN refers to a valid sequence of movements that satisfy the following conditions:

1. $\pi_1(\gamma)_{|A} = \sigma$, and
2. $M_{in}[\pi_2(\gamma)_{|T}\rangle M_{fi}$.

The alignment between σ and HPN is shown in Table 1. In the alignment, the top row corresponds to the trace σ, while the bottom row corresponds to a full firing sequence of the hierarchical process model. Each activity in σ is matched with a transition that has the same label.

σ	e_1	e_2	$\dots$	e_i
firing sequence of *HPN*	a_1	a_2	$\dots$	a_i

Table 1. Alignment matrix

1. If $e \in U_A$ and $a = \gg$, (e, a) is a movement on a log;
2. If $e = \gg$ and $a \in T$, (e, a) is a movement on a model;
3. If $e \in U_A$ and $a \in T$, (e, a) is a synchronous movement; and
4. Illegal movements otherwise.

In the remainder of this paper, we consider movement on a log, movement on a model and synchronous movement being the legal movements.

4 CONFORMANCE CHECKING OF HIERARCHICAL PROCESS MODELS

The conformance checking of the hierarchical process models starts from hierarchical models and lifecycle event logs. Finding corresponding hierarchical event logs and hierarchical alignments of the hierarchical models is the key to the approach. The framework of the conformance checking approach is depicted in Figure 3, which encompasses the following steps:

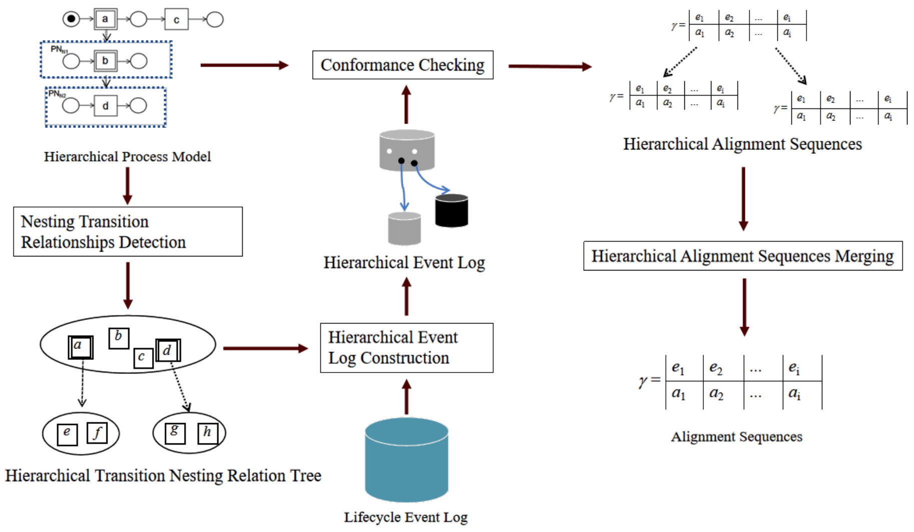

Hierarchical Process Model
Nesting Transition Relationships Detection
Hierarchical Transition Nesting Relation Tree
Conformance Checking
Hierarchical Event Log
Hierarchical Event Log Construction
Lifecycle Event Log
Hierarchical Alignment Sequences
Hierarchical Alignment Sequences Merging
Alignment Sequences

Figure 3. A Petri net with a nested transition pn_n

Phase 1: Nesting Transition Relationships Detection. Given a hierarchical process model *hpn* as input, we first detect nesting relations among the models. The output is a nested transition relations tree *ang*, which describes all possible nesting relationships in the hierarchical process model.

Phase 2: Hierarchical Event Log Construction. Given a lifecycle event log xlog and the nested transition relations tree *ang* as input, the nesting relations within the model are utilized to construct a hierarchical event log *hlog* recursively.

Phase 3: Nesting Relation Alignment Tree Construction. Given hierarchical event log *hlog* and hierarchical process model *hpn* as input, we recursively examine the conformance of the hierarchical structure to construct a nesting relation alignment tree *hat*.

Phase 4: Hierarchical Alignment Trees Merging. The nesting alignment relation tree can be merged to obtain the final result.

4.1 Nesting Relationships Within the Hierarchical Process Model Detect

A hierarchical process model can express multiple complex patterns such as multi-instance and sub-process, which can better describe the possible system state. In order to evaluate the quality of the hierarchical process model, it is necessary to extract the nesting relationships inherent within the model. The nesting transition relation trees and the hierarchical event logs can be constructed by the nesting relationships. (Then the hierarchical event logs can be obtained by the nesting relation trees.) The definition and the method for mining the nesting relationships within the hierarchical model are given as follows.

Definition 9 (Hierarchical Transition Nesting Relation Tree)**.** Let $HPN = (PN_{N0}, HPN(PN_{N0}))$ be a hierarchical process model, T_{n0} is the set of the top-level nested transitions. $HNT = (T_{n0}, HNT(T_{n0}))$ is the hierarchical transition nesting relationship tree of HPN, where:

1. T_{n0} is the set of the root node, i.e. the set of nested transitions in the top-level Petri net;
2. $HNT(T_{n0})$ is the corresponding sub-process of nested transitions in T_{n0}, such that:
   - $HNT(T_{n0}) = \emptyset$ if $T_{n0} = \emptyset$; otherwise,
   - $HNT(T_{n0}) = \{t_i, T_{ni}, HNT(T_{ni}) \mid t_i \in T_{n0}\}$, where T_{ni} is the set of nested transitions in the corresponding nested models of t_i.

Algorithm 1 shows how to get a hierarchical transition nesting relation tree. The idea is by a hierarchical process model as input, to detect all the nested models in the hierarchical model. All the nested transitions in the nested models should also be located. That is, to find the nesting relationships and recursively construct the hierarchical transition nesting relation tree.

Algorithm 1 *TransitiveNestingTree*()

```
Input: hierarchical process model $hpn$
Output: hierarchical transition nesting relation tree $ang$
1: $activityNestedSet[] \longrightarrow \emptyset$
2: $activityPariSet[] \longrightarrow \emptyset$
3: $ang$ = **new** $TransitionNestingGraph()$;
4: **if** $hpn \neq \emptyset$
5:     $activityNestedSet[]$ = getActivityNestedSet($hpn$);
6:     $activityPariSet[]$ = getActivityPair($hpn$);
7:     $ang = ActivityGraphConstruction(activityPariSet)$;
8: **return** $ang$.
```

In Algorithm 1, the variables are initialized (Lines 1–3). The function getActivityNestedSet() is employed to recognize the set of nested transitions that exist within the hierarchical model. And the function getActivityPair() is used to mine the set of nested transition association pairs (Lines 5–7). By the nested transition association pairs, the algorithm ActivityGraphConstruction() (Algorithm 4) can be utilized to construct the transition nesting relation tree *ang*. The algorithm traverses all activities of each layer in the hierarchical process model. Assuming the hierarchical process model has n layers, with the number of activities in each layer being L_1, L_2, L_3, …, and L_n, respectively. Therefore, the computational complexity is $O(n \times \max(L_i))$, for $1 \le i \le n$.

The function getActivityNestedSet() is called in Algorithm 1, which is used to return all the nested transitions in the hierarchical mode. The details are described in Algorithm 2.

Algorithm 2 getActivityNestedSet()

```
Input: hierarchical process model $hpn$
Output: the set of nested transitions $activityNestedSet[]$
1: $t = $ new $Transition()$;
2: if $hpn \neq \emptyset$
3:     for ($Transition\ t : pn$.getTransitions())
4:         if (!nestedTransitionLabels.contains($t$.getLabel()))
5:             $t \longrightarrow activityNestedSet[]$;
6:         getActivityNestedSet($hpn_i$);
7: return $activityNestedSet[]$.
```

Algorithm 2 recursively traverses the top-level Petri net to find the nested transitions. And all the nested transitions are stored in the set *activitySet*[]. The algorithm traverses all nested transitions of each layer in the hierarchical process model. Assuming the hierarchical process model has n layers, with the number of nested transitions in each layer being M_1, M_2, M_3, …, and M_n, respectively. Therefore, the computational complexity is $O(n \times \max(M_i))$, for $1 \le i \le n$.

The function getActivityPair() is called in Algorithm 1, which is used to return all the nested transition association pairs in the hierarchical model. The details are described in Algorithm 3.

Algorithm 3 recursively traverses the hierarchical model nested in the nested transition t. The nested transition t is extracted from *activityNestedSet*[] (Line 1). The nested transition t is assigned to source (Line 2). The hierarchical model hpn_i nested in t will be found (Line 3). The set of nested transitions in hpn_i will be obtained and stored in *target*. The nested transition association pairs (*source*, *target*) are stored in the set *activityPairSet*[]. The algorithm traverses the set *activityNestedSet*[], which stores the nested transitions. Therefore, the computational complexity is $O(n \times \max(M_i))$, for $1 \le i \le n$.

The function activityGraphConstruction() is called in Algorithm 1, which is used to return the hierarchical transition nesting relation tree in the hierarchical model. The details are described in Algorithm 4.

Algorithm 3 getActivityPair()

```
Input: hierarchical process model $hpn$
Output: the set of nested transition pairs $activityPairSet[]$
1: for (Transition $t$ : $activityNestedSet[]$)
2:     $source = t$;
3:     $hpn_i = HPN(PN_{N0})$;
4:     if $hpn_i \neq \emptyset$
5:         $target$ = getActivityNestedSet($hpn_i$);
6:         $(source, target) \longrightarrow activityPairSet[]$;
7:         getActivityPair($hpn_i$);
8: return $activityPairSet[]$.
```

Algorithm 4 *activityGraphConstruction*()

```
Input: the set of nested transition association pairs $activityPariSet[]$
Output: hierarchical transition nesting relation tree $ang$
1: $ap$ = new $TransitionPair()$;
2: $ang$ = new $TransitionNestingGraph()$;
3:     for ($ap$ : $activityPairSet[]$)
4:         $ap$.getSourceActivity() $\longrightarrow ang.vertex$;
5:         $ap$.getTargetActivity() $\longrightarrow ang.vertex$;
6:         ($ap$.getSourceActivity(), $ap$.getTargetActivity()) $\longrightarrow ang.edge$;
7:     and Constructing nested transition relation tree: $ang$;
8: return $ang$.
```

Algorithm 4 recursively traverses the nested transition association pairs. The nested transition association pairs *ap* is extracted from *activityPairSet*[] (Line 3). The node and edges of *ap* are stored in *ang* (Lines 4–6). The algorithm traverses the set *activityPairSet*[], which stores the pairs of nested transitions. Therefore, the computational complexity is $O(n \times \max(M_i))$, for $1 \leq i \leq n$.

By Algorithms 1, 2, 3, and 4, we can obtain the hierarchical transition nesting relation tree *ang* of *hpn*, which is shown in Figure 4. In the tree, (1) a, c is the root nodes; (2) b is nested in a and d is nested in b. The transition nesting tree is $Tr = \{T_{n0}, T_{n1}, T_{n2}\} = \{a, c, b, d\}$.

4.2 Hierarchical Event Log Construction

By the hierarchical transition nesting relation tree, the event logs with lifecycle can be layered. The hierarchical event log is constructed by event logs with lifecycle and hierarchical transition nesting tree. The definition of a hierarchical event log is given as follows.

Definition 10 (Hierarchical Event Log)**.** Let $HPN = (PN_{N0}, HPN(PN_{N0}))$ be a hierarchical process model, $Tr = \{T_{n0}, T_{n1}, \ldots, T_{nn}\}$ is the set of nested tran-

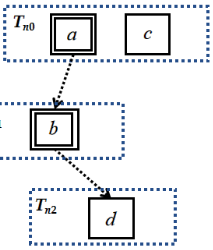


Figure 4. Hierarchical transition nesting relation tree *ang*

sitions, $HNT = (T_{n0}, HNT(T_{n0}))$ is the hierarchical transition nesting relationship tree of HPN. $HL = (rootLog, HL(rootLog))$ is the hierarchical event log of HPN, where:

1. $rootLog$ is the root event log of HL; and
2. $HL(rootLog)$ is the sub-logs of $rootLog$, such that:
   - $HL(rootLog) = \emptyset$ if $NA(rootLog) = \emptyset$; otherwise,
   - $HL(rootLog) = \{(na, NLog_{na}, HL(NLog_{na})) \mid na \in NA(rootLog)\}$, where $NLog_{na}$ is the sub-log of HL nested by na.

The nested transition relation tree *ang* and a lifecycle event log *xlog* are taking as input, a hierarchical event log *hlog* is recursively constructed. The details are described in Algorithm 5.

Algorithm 5 *constructHierarchicalLog*()

```
Input: the nested transition relation tree $ang$ and the lifecycle event log $xlog$
Output: the hierarchical event log $hlog$
1: for ($n$ : getAllNestedActivities($ang$))
2:     $n \longrightarrow allNestedActivities[]$;
3: $rootActivities$ = getAllRootActivities($ang$);
4: $topLevelActivities$ = getTopLevelActivitySet($ang$);
5: $hEventLog$.setMainLog($mainLog$);
6: for (String $rootNestedActivity$ : $rootActivities$)
7:     $hEventLog$.setSubLogMapping($subLogMapping$);
8:     $subsubLogMapping$.put($eventClassActivity$, $constructHierarchicalLog$());
9: return $hlog$.
```

In Algorithm 5, the nodes are extracted from *ang*, and stored in *allNestedActivities*[] (Lines 1–2). The root event logs and the sub-logs nested in root logs are constructed by *ang* and *xlog* (Lines 3–4). The structure of the hierarchical event log is assigned to *mainLog* (Line 5). The algorithm recursively traverses the nested activities in root logs and assigns them to the hierarchical event log structure *submainLog*

(Lines 6–8). The algorithm traverses each node of the nested transition relation tree and perform the corresponding operations on each node. The depths of the nested transition relation tree are n. The maximum number of nodes in each layer is $\max(M_i)$. Therefore, the computational complexity is $O(n \times \max(M_i))$, for $1 \leq i \leq n$.

By Algorithm 5, take hpn_1 (Figure 5) and $L_1 = \{\langle a_s, b_s, b_c, a_c\rangle^{90}, \langle a_s, b_s, a_c, b_c\rangle^{1}\}$ as input, we can obtain hierarchical event log hl_1, which is shown in Figure 6. The root log is $rootLog = \{\langle a_s, a_c\rangle^{91}\}$, and the set of nested activity is $NA(rootLog) = \{a\}$. The corresponding sub-logs of a is $NLoga = \{\langle b_s, b_c\rangle^{90}\}$.

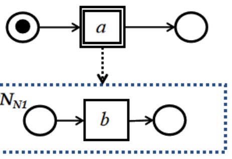


Figure 5. Hierarchical process model hpn_1

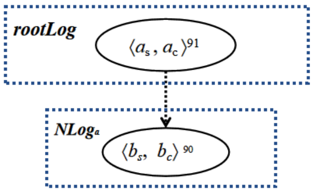


Figure 6. The nested relationship of hierarchical event log hl_1

4.3 Conformance Checking Based on Alignment

As the hierarchies in the hierarchical models and the hierarchical logs, the sub-models at each level need to be traversed. Conformance checking of sub-models and the corresponding sub-logs can be investigated. In theory, we can use the existing technology to take the conformance checking between the sub-models and sub-logs.

4.3.1 The Idea of Hierarchical Align

The A^* algorithm can be used to align the sub-models and the corresponding sub-logs at every level. A nesting relation alignment tree can be recursively constructed by the alignment of the sub-models and the sub-logs, which formal description is as follows:

Definition 11 (Hierarchical Alignment Tree)**.** Let $HPN = (PN_{N0}, HPN(PN_{N0}))$ be a hierarchical process model, $Tr = \{T_{n0}, T_{n1}, \ldots, T_{nn}\}$ is the set of nested transitions, $HNT = (T_{n0}, HNT(T_{n0}))$ is the hierarchical transition nesting relationship tree of HPN, $HL = (rootLog, HL(rootLog))$ is the hierarchical event log of HPN. $HAT = (rootA, HAT(rootA))$ is the hierarchical alignment tree of HPN and HNT, where:

1. $rootA$ is the root alignment of HAT, i.e. the alignment of PN_{N0} and $rootLog$; and
2. $HAT(rootA)$ is the nested alignment of $rootA$, such that:
   - $HAT(rootA) = \emptyset$ if $T_{n0} = \emptyset \wedge NA((rootLog) = \emptyset$; otherwise
   - $HAT(rootA) = \{(t_i, na, NAlign_{na}, HAT(NAlign_{na})) \mid t_i \in T_{n0}, na \in NA(rootLog)\}$, where $NAlign_{na}$ is the sub-alignment of the model nested in t_i and the logs of na.

Algorithm 6 takes hierarchical event log *hlog* and hierarchical process model *hpn* as input, a hierarchical alignment tree *hat* is recursively constructed. The details are described as follows.

Algorithm 6 *HierarchicallyAlignedSeqConstruction*()

```
Input: a hierarchical process model hpn and a hierarchical event log hlog
Output: hierarchical alignment tree hat
1: $xlog = hlog$.getMainLog();
2: $pn = hpn$.getPn();
3: $hat = replayLog(context, pn, xlog)$;
4: **When** $hpn_i \neq \emptyset$ and $hlog_i \neq \emptyset$
5:     $xlog = hlog$.getsubLogMapping().get($nestingActivityPariSet$);
6:     $pn = pn$.getXEventClass2hpn().get($eventClassName2EventClass$
                  .get($t$.toString()).getPn();
7:     $hat_i = replayLog(context, pn, xlog)$;
8: **return** $hat$.
```

In Algorithm 6, the root log *hlog* is extracted from the hierarchical log *hlog* (Line 1). The top-level Petri net *pn* is extracted from the hierarchical model *hpn* (Line 2). The top-level alignment is obtained by aligning *pn* and *xlog* (Line 3). The algorithm recursively aligns the nested model hpn_i and the nested log $hlog_i$ to get the hierarchical alignment tree *hat*. The algorithm traverses the process models and event logs of each layer and perform alignment. The levels of the process model are n. The activities of the process model in each layer is L_i. The length of the event log is σ_i. Therefore, the computational complexity is $O(n \times \max(\sigma_i, L_i))$, for $1 \leq i \leq n$.

Take the hierarchical model hpn_2 (in Figure 7) and the corresponding event log with lifecycle

$$\begin{aligned} L2 = \{&\langle a_s, c_s, d_s, d_c, b_s, c_c, a_c, b_c\rangle^{79}, \langle a_s, c_s, d_s, d_c, c_c, a_c, b_s, b_c\rangle^{99},\\ &\langle b_s, a_s, c_s, b_c, d_s, d_c, c_c, a_c\rangle^{96}, \langle a_s, b_s, c_s, d_s, d_c, c_c, a_c, b_c\rangle^{86},\\ &\langle a_s, c_s, d_s, b_s, d_c, c_c, a_c, b_c\rangle^{78}, \langle a_s, c_s, d_s, d_c, c_c, b_s, a_c, b_c\rangle^{82},\\ &\langle a_s, b_s, c_s, d_s, b_c, d_c, c_c, a_c\rangle^{98}, \langle a_s, c_s, b_c, d_s, d_c, c_c, b_c, a_c\rangle^{85},\\ &\langle a_s, c_s, b_s, d_s, b_c, d_c, c_c, a_c, b_c\rangle^{100}\} \end{aligned}$$

as an example illustrating how to get the hierarchical alignment tree.

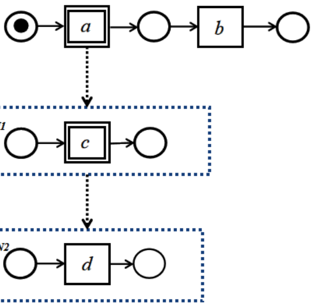


Figure 7. Hierarchical process model hpn_2

As Figure 7 shows, the top level of hpn_2 contains a nested transition a, and a normal transition b. The nested Petri net PN_{N1} is nested in a, and PN_{N1} with a nested transition c. The nested Petri net PN_{N2} is nested in c and PN_{N2} with a normal transition d.

By Algorithm 5, taking hpn_2 and L_2 as input, we can obtain the hierarchical event log hl_2. The root log is

$$\begin{aligned} rootLog = \{&\langle a_s, b_s, a_c, b_c\rangle^{79}, \langle a_s, a_c, b_s, b_c\rangle^{99}, \langle b_s, a_s, b_c, a_c\rangle^{96}, \langle a_s, b_s, a_c, b_c\rangle^{86},\\ &\langle a_s, b_s, a_c, b_c\rangle^{78}, \langle a_s, b_s, a_c, b_c\rangle^{82}, \langle a_s, b_s, b_c, a_c\rangle^{98}, \langle a_s, b_c, b_c, a_c\rangle^{85},\\ &\langle a_s, b_s, b_c, a_c, b_c\rangle^{100}\}, \end{aligned}$$

and the set of nested activities is $NA(rootLog) = \{a\}$. The corresponding sub-log to a is

$$\begin{aligned} NLog_a = \{&\langle c_s, c_c\rangle^{79}, \langle c_s, c_c\rangle^{99}, \langle c_s, c_c\rangle^{96}, \langle c_s, c_c\rangle^{86}, \langle c_s, c_c\rangle^{78}, \langle c_s, c_c\rangle^{82}, \langle c_s, c_c\rangle^{98},\\ &\langle c_s, c_c\rangle^{85}, \langle c_s, c_c\rangle^{100}\}, \end{aligned}$$

and the set of nested activities is $NA(NLog_a) = \{c\}$. The corresponding sub-log to

c is

$$NLog_c = \{\langle d_s, d_c\rangle^{79}, \langle d_s, d_c\rangle^{99}, \langle d_s, d_c\rangle^{96}, \langle d_s, d_c\rangle^{86}, \langle d_s, d_c\rangle^{78}, \langle d_s, d_c\rangle^{82}, \langle d_s, d_c\rangle^{98}, \langle d_s, d_c\rangle^{85}, \langle d_s, d_c\rangle^{100}\}.$$

The nesting relationship is shown in Figure 8.

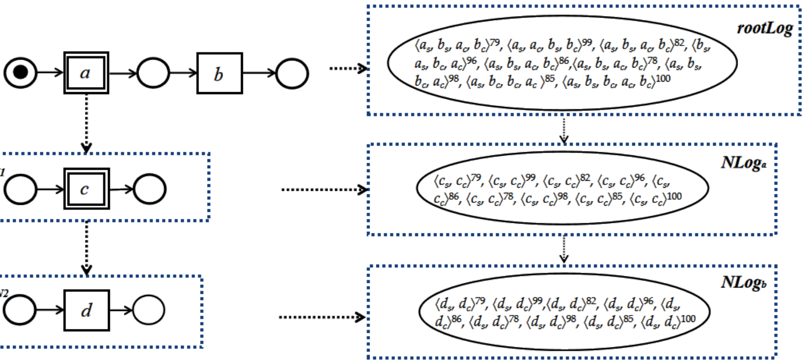

a
b
rootLog
⟨as, bs, ac, bc⟩79, ⟨as, ac, bs, bc⟩99, ⟨as, bs, ac, bc⟩82, ⟨bs, as, bc, ac⟩96, ⟨as, bs, ac, bc⟩86,⟨as, bs, ac, bc⟩78, ⟨as, bs, bc, ac⟩98, ⟨as, bc, bc, ac ⟩85, ⟨as, bs, bc, ac, bc⟩100
PNN1
c
NLoga
⟨cs, cc⟩79, ⟨cs, cc⟩99, ⟨cs, cc⟩82, ⟨cs, cc⟩96, ⟨cs, cc⟩86, ⟨cs, cc⟩78, ⟨cs, cc⟩98, ⟨cs, cc⟩85, ⟨cs, cc⟩100
PNN2
d
NLogb
⟨ds, dc⟩79, ⟨ds, dc⟩99,⟨ds, dc⟩82, ⟨ds, dc⟩96, ⟨ds, dc⟩86, ⟨ds, dc⟩78, ⟨ds, dc⟩98, ⟨ds, dc⟩85, ⟨ds, dc⟩100

Figure 8. The hierarchical event log hl_2 corresponds to the event log with lifecycle L_2

By Algorithm 6, taking hpn_2 and hl_2 as input, we can obtain the hierarchical alignment tree *hat* in Figure 9. Part of the results is shown as follows:

For top-level Petri net and root log, there are many possible cases of alignment, three of which are listed as follows:

$$x_1 = \begin{array}{|c|c|} \hline a & b \\ \hline \gg & b \\ \hline \end{array}, \quad x_2 = \begin{array}{|c|c|} \hline a & b \\ \hline a & b \\ \hline \end{array}, \quad x_3 = \begin{array}{|c|c|} \hline a & b \\ \hline a & \gg \\ \hline \end{array}.$$

For nested Petri net PN_{N1} and sub-log $NLog_a$, the possible alignment has the following two possible cases:

$$y_1 = \begin{array}{|c|} c \\ \hline \gg \end{array}, \quad y_2 = \begin{array}{|c|} c \\ \hline c \end{array}.$$

For nested Petri net PN_{N2} and sub-log $NLog_b$, the possible alignment has the following two possible cases:

$$z_1 = \begin{array}{|c|} d \\ \hline \gg \end{array}, \quad z_2 = \begin{array}{|c|} d \\ \hline d \end{array}.$$

The hierarchical alignment tree is shown in Figure 9.

4.3.2 Alignments

By taking the hierarchical alignment tree *hat* as input, the final alignment r is obtained by traversing and merging *hat*. The details are described in Algorithm 7.

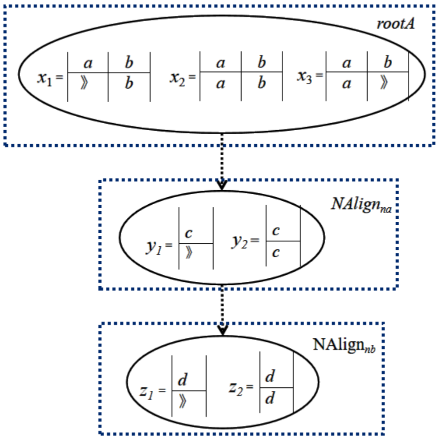


Figure 9. The hierarchical alignment tree *hat*

In Algorithm 7, the element in *rootA* should be pushed into the queue *st* when *hat* is not null (Lines 4–6). The head element in *st* is taken out, and the pointer points to the next element (Lines 7–9). The nodes in *st* should be traversed. If $roota.tag_i$ $==$ 1, then the elements before the node should be pushed into *st* (Lines 10–13). The next level of hat_i is recursively traversed. The algorithm traverses each node

Algorithm 7 *AlignedSeqConstruction*()

```
Input: the hierarchical alignment tree $hat$
Output: alignment $r$
1: $rootA = HNT(rootlog)$;
2: $st \longrightarrow \emptyset$
3: if $hat \neq \emptyset$
4:     if $rootA! = Null$
5:         $st$.push($rootA$);
6:         $rootA = HAT(rootA) \longrightarrow data$;
7:     $rootA = s$.top();
8:     $s$.pop();
9:     $r = rootA \longrightarrow child$;
10:     if $r \longrightarrow child == Null \,||\, r \longrightarrow chlid == tag$
11:         if (!$nestedTransitionLabels.contains(roota$.getLabel()))
12:             $r$.pop($roota \longrightarrow data$);
13:         else $r$.pop($roota \longrightarrow data$) and $roota.tagi == 1$;
14:         $rootA = r$ and $r = r \longrightarrow chlid$;
15:         AlignedSeqConstruction($hat_i$);
16: return $r$.
```

of the hierarchical alignment tree. The depths of the hierarchical alignment tree are n. The maximum number of nodes in each layer refers to $max(L_i)$, for $1 \leq i \leq n$. Therefore, the computational complexity is $O(n \times \max(L_i))$.

By Algorithm 7, the hierarchical alignment tree *hat* (in Figure 9) is merged, which is depicted in Figures 10, 11, 12, and 13.

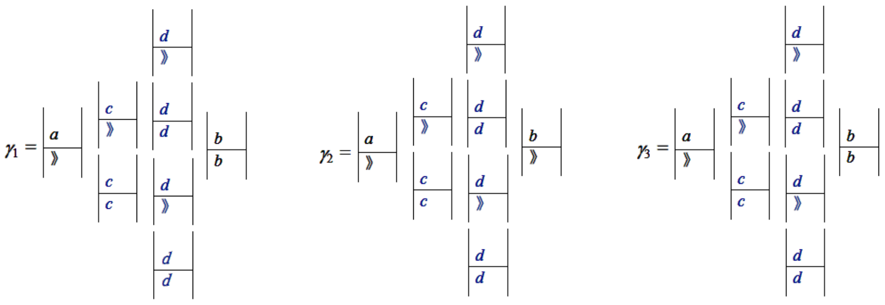


Figure 10. Three possible hierarchical alignment trees of hpn_2 and hl_2

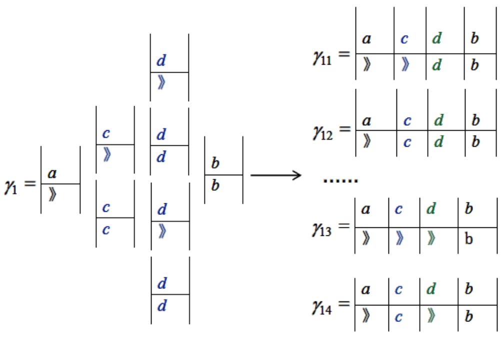


Figure 11. The merged alignment r_1 of *hat*

The elements of $rootA = \{r_1, r_2, r_3, \dots\}$ are pushed in the queue st. Algorithm 7 takes out the head elements r_1 in st and traverses it. As $tag_i == 1$, then node a is a nested transition. The elements before a should be pushed into st. The hierarchical alignment nested in a should be recursively traversed. The other node in $rootA$ should be traversed in the same way.

5 TOOL IMPLEMENTATION AND EXPERIMENTAL EVALUATION

Experiments are conducted to demonstrate the practicality and effectiveness of the proposed conformance checking approach in this paper. Firstly, the tool implementation of the conformance checking approach is introduced. Then, the conformance

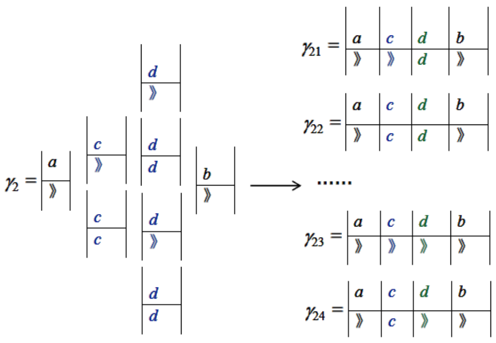


Figure 12. The merged alignment r_2 of *hat*

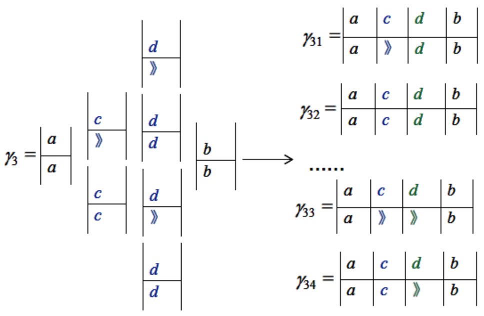
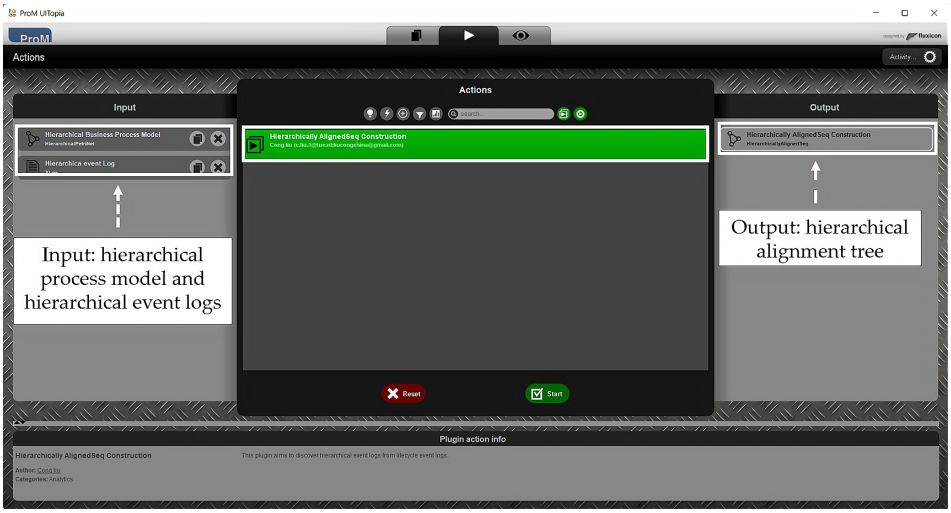


Figure 13. The merged alignment r_3 of *hat*

checking for hierarchical models is compared with existing conformance checking methods based on simulation logs and real logs.

5.1 Tool Implementation

The conformance checking approaches suggested in this paper have been integrated into ProM (`http://promtools.org`) through an extension. ProM is an open-source platform that offers plug-ins to support a range of process mining techniques. The approaches in this paper: are 1) using XES event logs with lifecycle as input; 2) user input noise threshold; and 3) the conformance checking outcomes of the hierarchical models and logs are obtained by utilizing the corresponding hierarchical process models as input. The plug-in Hierarchically AlignedSeq Construction (HAC) for the method is implemented, the details are shown in Figure 14.


ProM UITopia
ProM
designed by fluxicon
Actions
Activity...
Input
Hierarchical Business Process Model
HierarchicalPetriNet
Hierarchica event Log
XLog
Input: hierarchical process model and hierarchical event logs
Actions
search...
Hierarchically AlignedSeq Construction
Cong liu (c.liu.3@tue.nl;liucongchina@gmail.com)
Output
Hierarchically AlignedSeq Construction
HierarchicallyAlignedSeq
Output: hierarchical alignment tree
Reset
Start
Plugin action info

Hierarchically AlignedSeq Construction

This plugin aims to discover hierarchical event logs from lifecycle event logs.

Author: Cong liu
Categories: Analytics

Figure 14. The details of the plug-in

5.2 Experiment Data

The input data of the approach are event logs with lifecycle and hierarchical models with sub-process. We got 2 simulation hierarchical models and 2 real hierarchical models as the experiment data. The detail of the hierarchical process models is given as follow:

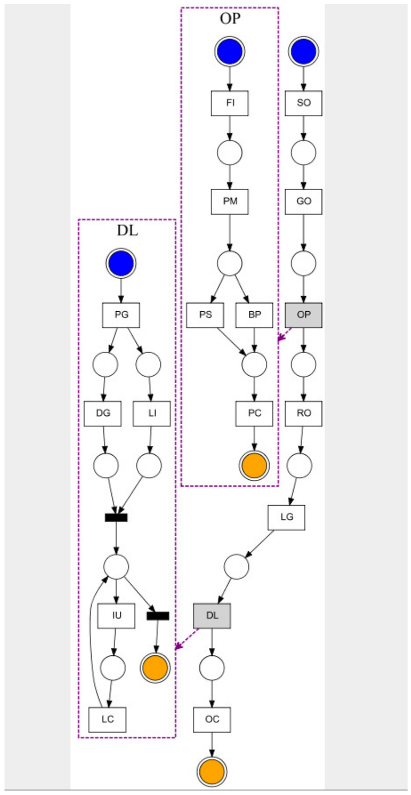
The 2 simulation models hpn_1 and hpn_2 are mentioned above. The 2 real models are hpn_3 and hpn_4 given in Figure 15 respectively. And the corresponding datasets are public datasets *TSECLog* and *CRMCLog*.

The source of the datasets:

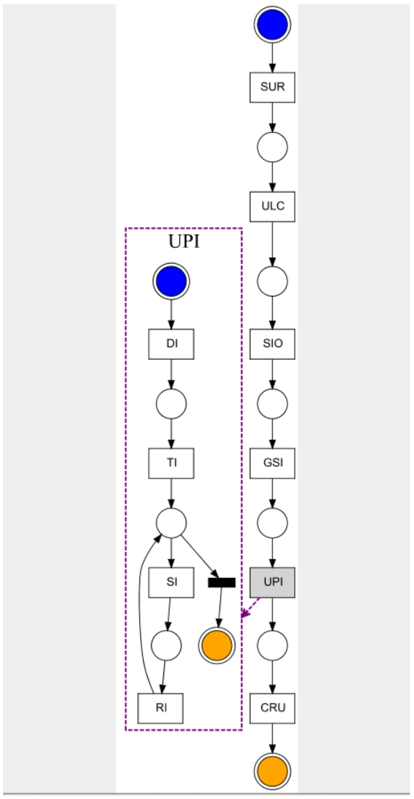
1. *TSECLog*: the dataset is generated based on the transnational e-commerce scenario. When a user submits an order, the merchant contacts a third-party logistics provider. Once the user receives the items and completes the payment, the order is considered complete. The process involves two sub-processes: the third-party logistics process and the payment process.
2. *CRMCLog*: the dataset is created from the upgrade process of Netflix Asgard, an open-source cloud resource management tool that runs on Amazon Web Services. The entire process includes preparation before the upgrade and the upgrade process. The upgrade process is a sub-process of preparation before the upgrade.

The basic information about the datasets and the model size is respectively given in Table 2 and Table 3.

The link of data and program is:
`https://pan.baidu.com/s/1Soku4gJWzX5Js6ICglfnOg?pwd=9ecc`.



a)



b)

Figure 15. a) The hierarchical process model *hpn_3* of *TSECLog* and b) the hierarchical process model *hpn_4* of *CRMCLog*

Hierarchical Model	Transitions	Places	Connections
hpn_1	2	4	4
hpn_2	4	7	8
hpn_3	19	21	40
hpn_4	11	12	22

Table 2. The information on the model size

Event logs	Traces	Events	Activities
L_1	91	360	4
L_2	802	6 516	8
TSECLog	522	14 616	20
CRMCLog	626	27 544	34

Table 3. The information of the datasets

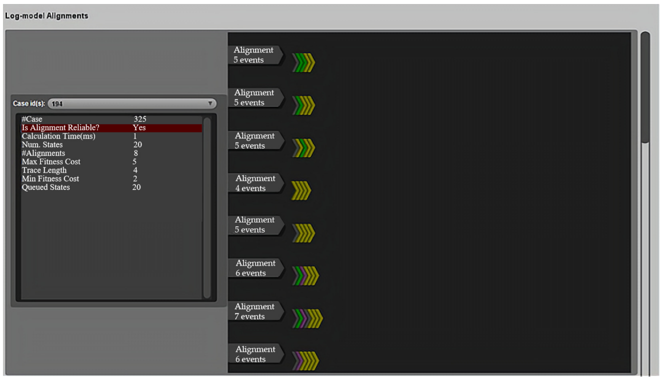
5.3 Experiments Evaluation

The experiment results based on simulation models and logs are illustrated in this paragraph. To evaluate the effectiveness of the conformance checking approach (HAC) proposed in this paper, the Alignment based Hierarchical Business Process Model Compliance Detection algorithm (HAC) was quantitatively compared with Convert a Hierarchical Petri Net to a Flat Petri Net (CHFP) [26] and Conformance Checking by Decomposition (DCC) [18] based on simulation event logs and real event logs.

In order to validate the effectiveness of the proposed method, simulation experiments (Subsection 5.3.1) were performed. Applying HAC, CHFP, and DCC to hierarchical process models of varying sizes (Subsection 5.3.2), the effectiveness and adaptability of HAC can be evaluated. Additionally, event logs with different lengths in terms of lifecycle are used to evaluate the performance and effectiveness of HAC (Subsection 5.3.3). Finally, the performance of HAC at different noise levels is evaluated by applying the approach to various noise thresholds (Subsection 5.3.4).

5.3.1 Simulation Experiment Results

Taking hpn_2 and L_2 as an example, part of the alignment results is shown in Figure 16.



#Case	325
Is Alignment Reliable?	Yes
Calculation Time(ms)	1
Num. States	20
#Alignments	8
Max Fitness Cost	5
Trace Length	4
Min Fitness Cost	2
Queued States	20

Figure 16. Part of the alignment results of hpn_2 and L_2

In the alignments in Figure 16, green is a synchronous movement, yellow is a movement on a log, and purple is a movement on a model. Figure 16 shows 8 possible alignments for case 194 with trace length 4, and the best alignment result contains 2 synchronous moves and 2 log moves. The conformance checking method presented in this paper enables the evaluation of hierarchical structures within hierarchical models for conformance. This approach can be utilized to identify and diagnose conformance issues in complex systems. To illustrate the contribution, based on hpn_2 and L_2, the overall fitness and the fitness for each level are analyzed by HAC, which is shown in Figure 17.

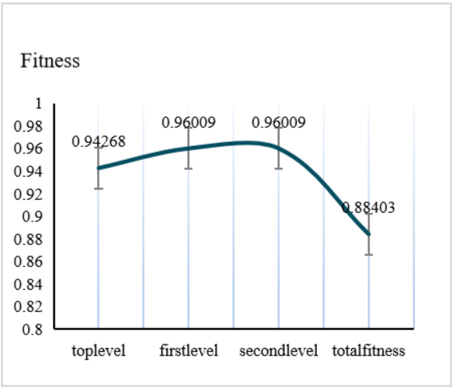


Figure 17. Visualization of conformance for the hpn2 run example

The line chart is used to describe the fitness of hierarchical models hpn_2. *toplevelfitness*, *firstlevelfitness*, and *secondlevelfitness* represent the fitness of the top-level model, first-layer model, and second-layer model of the hierarchical model, respectively. *totalfitness* indicates the fitness of the overall hierarchical model.

Figure 17 shows that the *toplevelfitness*, *firstlevelfitness*, and *secondlevelfitness* of the hierarchical process model hpn_2 are 0.94268, 0.96009, and 0.96009, respectively. Its *totalfitness* is 0.88403. The alignment result of each level is higher than the overall alignment result. It is because the mapping relationship between the levels is not taken into account when the hierarchical process model and the hierarchical log are used for the hierarchical alignment. This point will be further improved in future research.

5.3.2 Different Sizes of Hierarchical Process Models

To assess the practicality and effectiveness of the conformance checking method, the 4 hierarchical models hpn_1, hpn_2, hpn_3, and hpn_4 in Table 2 are used as inputs to compare the computation time of alignment with different sizes. hpn_1 is the smallest, with the least transition, places, and connections. hpn_3 is the largest, with the most transition, places, and connections.

Taking the hierarchical models hpn_1, hpn_2, hpn_3, and hpn_4 as input, HAC, CHFP, and DCC are run to alignment, respectively. The performance of HAC is demonstrated by comparing the computation time.

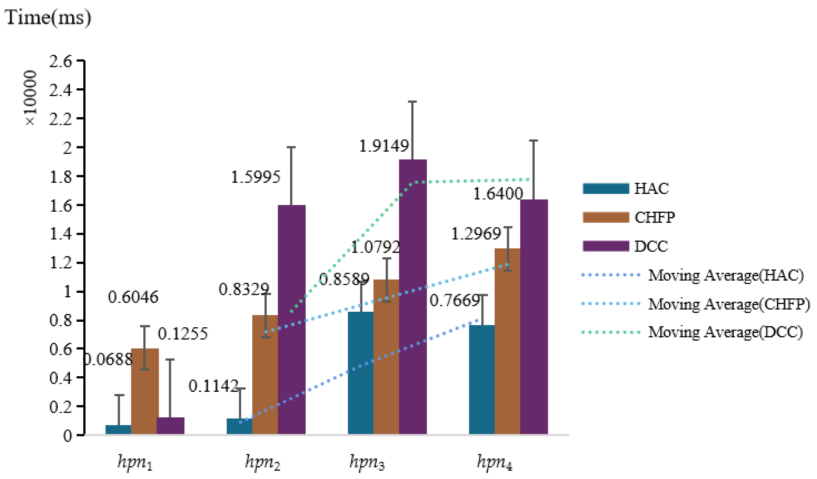


Figure 18. Comparison of computational time for different sizes of models

Figure 18 shows that, for hpn_1 (the smallest size), the computing time of HAC is 688 ms, the computing time of CHFP is 6 046 ms, and the computing time of DCC is 1 255 ms. For hpn_3 (the largest scale), the computing time of HAC is 8 589 ms. The computing time of CHFP is 10 792 ms. The computing time of DCC is 19 149 ms. The computing time of HAC is the lowest with the same size as the hierarchical model. For DCC with small models, the larger the model size, the longer times are used to decompose the process model. But, DCC has a great advantage in computing time when dealing with large process models. From the moving average trend line in Figure 18, we observe that the computing times of HAC, CHFP, and DCC increase with the model sizes increase. For all the models, the computing times of HAC are lower than CHFP, and DCC.

5.3.3 Different Log Lengths

Using logs of varying sizes as input for the same hierarchical process model, the computing time is used to verify the effectiveness and availability of HAC, CHFP, and DCC. The hierarchical model hpn_2 is utilized in this experiment, and the information on the logs with different log lengths is given in Table 4.

Table 4 shows that *prom_1* is the minimum length with 368 traces and 2 962 events. The *prom_2* contains 500 traces and 4 100 events. The *prom_3* contains 802 traces and 6 516 events. The *prom_4* is the maximum length with 897 traces and 7 281 events. The experiment results are shown in Figure 19.

Event Logs	Traces	Events	Activities
prom_1	368	2 962	4
prom_2	500	4 100	8
prom_3	802	6 516	20
prom_4	897	7 281	34

Table 4. The information of the event logs

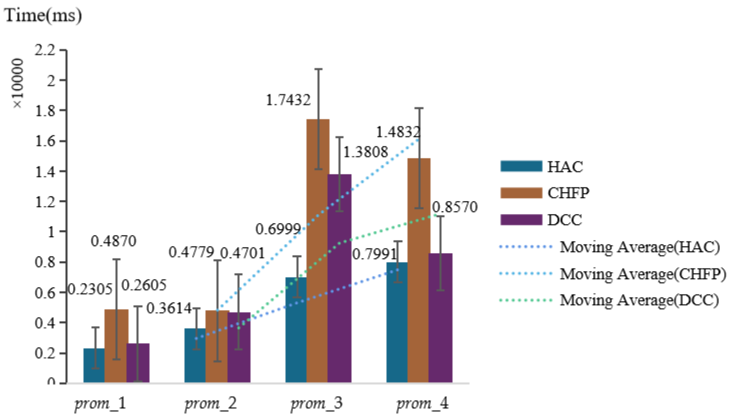


Figure 19. Comparison of computational time for different sizes of logs

From Figure 19, we observe that HAC, CHFP, and DCC perform well with *prom_1* (the minimum log length), the computation time are 2 305 ms, 4 870 ms, and 2 605 ms, respectively. However, once the log size increases, the performance of the CHFP and DCC will degrade. For *prom_3*, the computation time of HAC, CHFP, and DCC is different. The computation time of HAC is 6 999 ms, which is less than 17 432 ms of CHFP and 13 808 ms of DCC. That is the more activities in logs, the more difficult to calculate the alignments. Figure 19 shows that, by the simulation experiment with the same log lengths, the performance of HAC is better than CHFP and DCC.

5.3.4 Different Noise Thresholds

Taking logs with different noise thresholds as input, the performance of HAC is verified. The logs L_1, L_2, *TSECLog*, and *CRMCLog* in Table 3 are used in this experiment. L_1 contains the least events, which are 360. *CRMCLog* contains the most events, which are 27 544. The plug-in "Add Noise to Log Filter" is used to add noises to the logs. We add noise levels in percentages 2.0, 3.0, and 5.0 to each log. Then the logs with noises are used in this experiment. The effectiveness and practicability of HAC are evaluated by comparing the computation time with logs containing different noise thresholds. The experiment results are shown in Figure 20.

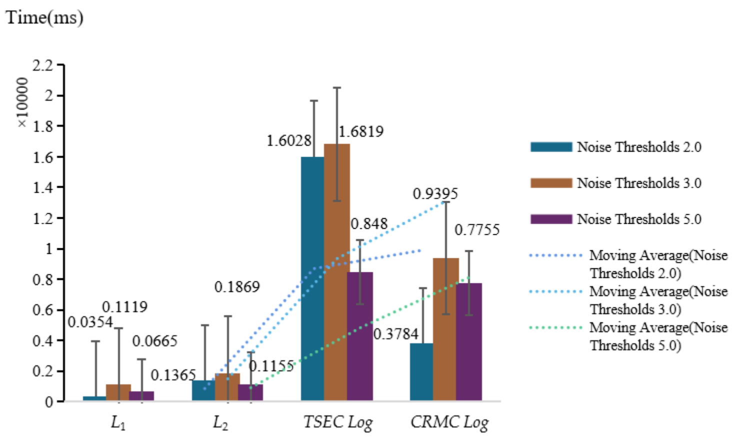


Figure 20. Comparison of computational time at different noise thresholds

Figure 20 shows that the performance of HAC is not significantly impacted by different noise thresholds when L_1 and L_2 contain fewer events. For L_1 with noise thresholds 2.0, 3.0, and 5.0, the computation times of HAC are 354 ms, 1 119 ms, and 665 ms, respectively. For L_2 with noise thresholds 2.0, 3.0, and 5.0, the computation times of HAC are 1 365 ms, 1 869 ms, and 1 155 ms, respectively. For *TSECLog* with noise thresholds 2.0, 3.0, and 5.0, the computation times of HAC are 16 028 ms, 16 819 ms, and 8 480 ms, respectively. *CRMCLog* with noise thresholds 2.0, 3.0, and 5.0, the computation time are 3 784 ms, 9 395 ms, and 7 755 ms, respectively. It shows that when *TSECLog* and *CRMCLog* with more events are used, the performance of HAC within various noises is different obviously. That is because as the number of events increases, more alignments should be calculated. Therefore, to obtain more accurate alignment results, the noises in logs need to be preprocessed.

6 CONCLUSIONS

The conformance checking approach for hierarchical process models is proposed in this paper. As the hierarchical structures in hierarchical process models, it is hard to check the conformance. And most of the existing conformance checking methods are only for the flat process models. The nested relationships in hierarchical models are researched. The existing conformance checking approaches is used to detect the sub-models at each level. The results of each level are merged to analyze the conformance of hierarchical models. The effectiveness of the proposed conformance checking approach is evaluated using both real-world and simulated hierarchical models. The computational time for different sizes of models and different noise thresholds among the same size logs is compared in the experiments. Experimental results confirm the practicality and effectiveness of the conformance checking approach.

After the logs are layered, the activity names in the logs are only considered. The lifecycle information of the logs is ignored in the approach. The method proposed in this paper can only be applied to hierarchical models that involve a single process instance. And the noise threshold in this method lacks of systematic setting means. Future work will focus on researching the hierarchical structure in event logs. The other feature information of the logs will be considered to optimize the approach, and to improve the conformance checking approach to detect the hierarchical models with multi-instances.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (No. 61902222), the TaiShan Scholars Program of Shandong Province (No. ts20190-936, No. tsqn201909109), the Natural Science Excellent Youth Foundation of Shandong Province (ZR2021YQ45), the Youth Innovation Science and Technology Team Foundation of Shandong Higher School (No. QC2021948080), the Education Ministry Humanities and Social Science Research Youth Fund Project of China (No. 20-YJCZH159), the Natural Science Foundation Youth Fund Project of Shandong Province (No. ZR2022QF020), and the National Natural Science Foundation of China (No. 72101137), the Education Ministry Humanities and Social Science Research Youth Fund Project of China (No. 21YJCZH150), the Natural Science Foundation of Shandong Province (No. ZR2021MF117), and the Key R & D Program (Soft Science) Project of Shandong Province (No. 2022RKY02009).

REFERENCES

[1] Kumar, A.: Business Process Management. Routledge, 2018, doi: 10.4324/9781315646749.

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

[3] Liu, C.: Hierarchical Business Process Discovery: Identifying Sub-Processes Using Lifecycle Information. 2020 IEEE International Conference on Web Services (ICWS), 2020, pp. 423–427, doi: 10.1109/ICWS49710.2020.00062.

[4] Rozinat, A.—van der Aalst, W. M. P.: Conformance Checking of Processes Based on Monitoring Real Behavior. Information Systems, Vol. 33, 2008, No. 1, pp. 64–95, doi: 10.1016/j.is.2007.07.001.

[5] Carmona, J.—van Dongen, B.—Weidlich, M.: Conformance Checking: Foundations, Milestones and Challenges. In: van der Aalst, W. M. P., Carmona, J. (Eds.): Process Mining Handbook. Springer, Cham, Lecture Notes in Business Information Processing, Vol. 448, 2022, pp. 155–190, doi: 10.1007/978-3-031-08848-3_5.

[6] van der Aalst, W. M. P.—Rubin, V.—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. Software & Systems Modeling, Vol. 9, 2010, No. 1, pp. 87–111, doi: 10.1007/s10270-008-0106-z.

[7] Burattin, A.—van Zelst, S. J.—Armas-Cervantes, A.—van Dongen, B. F.—Carmona, J.: Online Conformance Checking Using Behavioural Patterns. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (Eds.): Business Process Management (BPM 2018). Springer, Cham, Lecture Notes in Computer Science, Vol. 11080, 2018, pp. 250–267, doi: 10.1007/978-3-319-98648-7_15.

[8] Carmona, J.—van Dongen, B. F.—Solti, A.—Weidlich, M.: Conformance Checking: Relating Processes and Models. Springer, 2018, doi: 10.1007/978-3-319-99414-7.

[9] van der Aalst, W.: Process Mining: Overview and Opportunities. ACM Transactions on Management Information Systems (TMIS), Vol. 3, 2012, No. 2, Art. No. 7, doi: 10.1145/2229156.2229157.

[10] van der Aalst, W.—Adriansyah, A.—van Dongen, B.: Replaying History on Process Models for Conformance Checking and Performance Analysis. WIREs Data Mining and Knowledge Discovery, Vol. 2, 2012, No. 2, pp. 182–192, doi: 10.1002/widm.1045.

[11] Dunzer, S.—Stierle, M.—Matzner, M.—Baier, S.: Conformance Checking: A State-of-the-Art Literature Review. Proceedings of the 11th International Conference on Subject-Oriented Business Process Management (S-BPM ONE '19), 2019, doi: 10.1145/3329007.3329014.

[12] Jagadeesh Chandra Bose, R. P.—van der Aalst, W. M. P.: Process Diagnostics Using Trace Alignment: Opportunities, Issues, and Challenges. Information Systems, Vol. 37, 2012, No. 2, pp. 117–141, doi: 10.1016/j.is.2011.08.003.

[13] Jagadeesh Chandra Bose, R. P.—van der Aalst, W.: Trace Alignment in Process Mining: Opportunities for Process Diagnostics. In: Hull, R., Mendling, J., Tai, S. (Eds.): Business Process Management (BPM 2010). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6336, 2010, pp. 227–242, doi: 10.1007/978-3-642-15618-2_17.

[14] Schuster, D.—Föcking, N.—van Zelst, S. J.—van der Aalst, W. M. P.: Conformance Checking for Trace Fragments Using Infix and Postfix Alignments. In: Sellami, M., Ceravolo, P., Reijers, H. A., Gaaloul, W., Panetto, H. (Eds.): Cooperative Information Systems (CoopIS 2022). Springer, Cham, Lecture Notes in Computer Science, Vol. 13591, 2022, pp. 299–310, doi: 10.1007/978-3-031-17834-4_18.

[15] De Giacomo, G.—Maggi, F. M.—Marrella, A.—Patrizi, F.: On the Disruptive Effectiveness of Automated Planning for LTLf-Based Trace Alignment. Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 31, 2017, No. 1, doi: 10.1609/aaai.v31i1.11020.

[16] van Dongen, B. F.: Conformance Checking: A Systemic View. In: Marrella, A., Weber, B. (Eds.): Business Process Management Workshops (BPM 2021). Springer, Cham, Lecture Notes in Business Information Processing, Vol. 436, 2021, pp. 61–72, doi: 10.1007/978-3-030-94343-1_5.

[17] De Leoni, M.—van der Aalst, W. M. P.: Aligning Event Logs and Process Models for Multi-Perspective Conformance Checking: An Approach Based on Integer Linear Programming. In: Daniel, F., Wang, J., Weber, B. (Eds.): Business Process Management (BPM 2013). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 8094, 2013, pp. 113–129, doi: 10.1007/978-3-642-40176-3_10.

[18] Munoz-Gama, J.: Decomposing for Fitness Checking. Conformance Checking and Diagnosis in Process Mining: Comparing Observed and Modeled Processes, Springer, Cham, Lecture Notes in Business Information Processing, Vol. 270, 2016, pp. 129–139, doi: 10.1007/978-3-319-49451-7_13.

[19] van der Aalst, W. M. P.: Decomposing Petri Nets for Process Mining: A Generic Approach. Distributed and Parallel Databases, Vol. 31, 2013, No. 4, pp. 471–507, doi: 10.1007/s10619-013-7127-5.

[20] Munoz-Gama, J.—Carmona, J.—van der Aalst, W. M. P.: Single-Entry Single-Exit Decomposed Conformance Checking. Information Systems, Vol. 46, 2014, pp. 102–122, doi: 10.1016/j.is.2014.04.003.

[21] Song, W.—Xia, X.—Jacobsen, H. A.—Zhang, P.—Hu, H.: Efficient Alignment Between Event Logs and Process Models. IEEE Transactions on Services Computing, Vol. 10, 2017, No. 1, pp. 136–149, doi: 10.1109/TSC.2016.2601094.

[22] Sommers, D.—Sidorova, N.—van Dongen, B.: Aligning Event Logs to Resource-Constrained ν-Petri Nets. In: Bernardinello, L., Petrucci, L. (Eds.): Applications and Theory of Petri Nets and Concurrency (PETRI NETS 2022). Springer, Cham, Lecture Notes in Computer Science, Vol. 13288, 2022, pp. 325–345, doi: 10.1007/978-3-031-06653-5_17.

[23] De Leoni, M.—Maggi, F. M.—van der Aalst, W. M. P.: Aligning Event Logs and Declarative Process Models for Conformance Checking. In: Barros, A., Gal, A., Kindler, E. (Eds.): Business Process Management (BPM 2012). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7481, 2012, pp. 82–97, doi: 10.1007/978-3-642-32885-5_6.

[24] Wang, Y.—Wen, L.—Yan, Z.: Alignment Based Conformance Checking Algorithm for BPMN 2.0 Model. Journal of Computer Research and Development, Vol. 54, 2017, No. 9, pp. 1920–1930, doi: 10.7544/issn1000-1239.2017.20160756 (in Chinese).

[25] Wang, Y.—Wen, L.—Yan, Z.—Sun, B.—Wang, J.: Discovering BPMN Models with Sub-Processes and Multi-Instance Markers. In: Debruyne, C., Panetto, H., Meersman, R., Dillon, T., Weichhart, G., An, Y., Ardagna, C. A. (Eds.): On the Move to Meaningful Internet Systems: OTM 2015 Conferences. Springer, Cham, Lecture Notes in Computer Science, Vol. 9415, 2015, pp. 185–201, doi: 10.1007/978-3-319-26148-5_11.

[26] Liu, C.—Cheng, L.—Zeng, Q.—Wen, L.—Ouyang, C.: Petri Net-Based Hierarchical Business Process Discovery. Computer Integrated Manufacturing Systems, Vol. 26, 2020, No. 6, pp. 1525–1537, doi: 10.13196/j.cims.2020.06.009 (in Chinese).

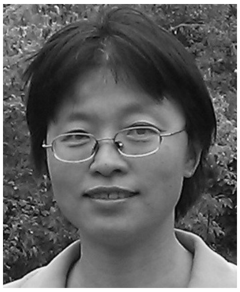
[27] Wang, Y.—Liu, C.—Wen, L.—Zeng, Q.—Cheng, L.: Hierarchical Multi-Instance Processes Model Discovery Approach. Computer Integrated Manufacturing System, Vol. 28, 2022, No. 10, pp. 3246–3255, doi: 10.13196/j.cims.2022.10.020 (in Chinese).

[28] Reisig, W.: Petri Nets: An Introduction. Springer, 2012, doi: 10.1007/978-3-642-69968-9_2.

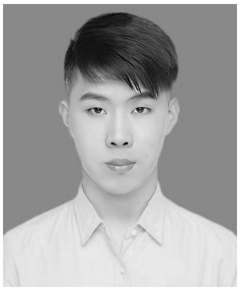
[29] Adriansyah, A.—Munoz-Gama, J.—Carmona, J.—van Dongen, B. F.—van der Aalst, W. M. P.: Alignment Based Precision Checking. In: La Rosa, M., Soffer, P. (Eds.): Business Process Management Workshops (BPM 2012). Springer, Berlin, Heidelberg, Lecture Notes in Business Information Processing, Vol. 132, 2013, pp. 137–149, doi: 10.1007/978-3-642-36285-9_15.

Lu Wang received her B.Sc. and Ph.D. degrees in computer software and theory from the Shandong University of Science and Technology, Qingdao, China, in 2013 and 2018, respectively. She is currently a Lecturer of computer science and technology at the Shandong University of Science and Technology, Qingdao, China. Her current research interests include business process management, process mining, workflow and Petri nets.

Xiao Han received her B.Sc. degree in information engineering from the Shandong University of Science and Technology, Qingdao, China, in 2020, and now she is pursuing her M.Sc. degree in computer science and technology from the Shandong University of Science and Technology, Qingdao, China. Her current research interests include process mining, conformance checking and Petri nets.



Man Qi is Senior Lecturer in computing at the Canterbury Christ Church University. Her research interests are in cyber security, data intelligence, IoT and HCI. She published over 80 research papers including over 30 journal papers and is the editorial board member for 5 international journals. She has been the Ph.D. external examiners for many universities in Australia and UK. She has served as Chair/Program Committee Member for around 50 international conferences and been long term reviewer for many international journals. She is Fellow of British Computer Society (FBCS) and Fellow of Higher Education Academy (FHEA).



Kang Wang received his B.Sc. degree in computer science and technology from the Binzhou University, BinZhou, China, in 2020, and now he is pursuing his M.Sc. degree in electronic science and technology from the Shandong University of Science and Technology, Qingdao, China. His current research interests include process mining and Petri nets.

Peng Li received his B.Sc. degree from the College of Computer Science, Qufu Normal University, China, in 2007, and his M.Sc. degree from the College of Computer Science and Engineering, Shandong University of Science and Technology, China, in 2010. He is currently pursuing the Ph.D. degree with the Shandong University of Science and Technology and also is a Senior Engineer with the Affiliated Hospital of Qingdao University.