Business Process Retrieval from Large Model Repositories for Industry 4.0

Rui Zhu, *Member, IEEE*, Yue Huang, Ling Liu, *Fellow, IEEE*, Wei Zhou,

Xuan Zhang, Yeting Chen* and Li Cai*

Abstract—The process model repository has demonstrated unprecedented success in a variety of industrial and process as a service scenarios. With the rapid increase of massive business process-related data under Industry 4.0, effectively retrieval of process models from large process model repositories becomes a critical challenge for process mining, process deployment and process model acquisition. To accelerate the retrieval of process models from a large process repository, existing retrieval methods rely solely on building single dimension process model indices. In this paper we show that this single dimension indexing approach is not only inefficient but also cumbersome for supporting high performance retrieval services over large process model repositories. We propose a new business process model indexing and retrieval with structure and behavior fusion. In the indexing stage, we propose a process model index generation paradigm method with two novel features. First, our index algorithm can transform the *trace equivalent process model* (TEPM) with complex structures into a process tree, which can better capture process sequence semantics than the existing approach based on block structured process model. Second, we improve the method for computing the process tree edit distance for measuring process model similarity by introducing the process tree similarity method, which can distinguish leaf nodes and non-leaf nodes and improve the limitations of the traditional edit distance algorithm. Extensive experiments using real world process repositories demonstrate that the proposed methods are under polynomial time in both the model index generation and model querying stages, and offer superior retrieval performance compared to existing process model retrieval methods in terms of efficiency, search capability and scope.

Index Terms—business process model, complete finite prefix unfolding, Industry 4.0, process model repository, process retrieval

1 Introduction

In the field of business process management (BPM), the process model repository[1-3] (PMR) was proposed to query, store and analyze process-related data. The PMR can be used in wide and significant scenarios in process analytics and business intelligence, such as formal model verification[4], next event prediction[5], heterogeneous event data matching[6, 7], natural language requirement generation[8] and process model reuse[9].

Industrial process management also needs the support of the PMR in Industry 4.0. Massive business processes[10] are contained in many production control systems, such as product design, production equipment, development, and process perception for most core industrial products. For instance, the lithography process and chip manufacturing process of the famous Dutch ASML lithography machine company have essentially adopted the industrial process, which is the business process in industrial manufacturing used to control the production and manufacturing of chips[13]. Besides the PMR of the CNR Group's control product implementation has accumulated more than 200,000 process models[14], and Suncorp Bank in Australia has more than 6,000 process models[15]. Nearly 136,000 business rules were extracted from the 2 million lines of COBOL code in the Volkswagen Financial Car Rental Legacy System built in the 1980s in Germany[16].

For massive and complex process models, how to effectively retrieve the process models from large model repositories for Industry 4.0 is an urgent problem that needs to be solved. Compared with conventional process management, large PMR for Industry 4.0 should have the following four characteristics, and shorted for VVCC:

(1) Volume: As core intangible assets, business process models are essential approaches for organizations and their business process improvement strategies[17-19], and decision-making through process models also are contemporary companies' smart knowledge-based solutions by Industry 4.0 to compete in the worldwide scenario [20-23]. With the continuous advancement of global Industry 4.0, it is an indisputable fact that massive amounts of industrial process data have gradually accumulated in enterprises.

(2) Velocity: With the growth and integration of

- *R. Zhu, W. Zhou, X. Zhang, and L. Cai are with the School of Software, Yunnan University, Kunming 650091, China. E-mail: rzhu@ynu.edu.cn, zwei@ynu.edu.cn, zhxuan@ynu.edu.cn, caili@ynu.edu.cn.*
- *Y. Huang is with the School of Software, Shandong University, Ji'nan 250101, China. E-mail: xgyxhy@163.com.*
- *L. Liu is with the School of Computer Science, Georgia Institute of Technology Atlanta, Georgia, USA. E-mail: ling.liu@cc.gatech.edu.*
- *Y. Chen is with the School of Economics and Management, Yunnan Normal University, Kunming 650500, China. E-mail: 17801037267@163.com.*

Our deepest gratitude goes to the anonymous reviewers for their careful work and thoughtful suggestions that have helped to improve this paper substantially. This work was supported by National Natural Science Foundation of China under grants 62362067, 62002310; Yunnan Provincial Natural Science Foundation Fundamental Research Project under grant 202101AT070004; Science Foundation of Yunnan Jinzhi Expert Workstation under grant 202205AF150006; Yunnan Provincial Key Laboratory of Software Engineering Open Fund Project under grant 2023SE205; Yunnan Xing Dian Talents Support Plan.

(Corresponding author: Yeting Chen and Li Cai)

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

modern technologies, including BPM, service workflow, Internet of Things, cloud computing, service-oriented architecture cyber-physical systems, cyber-physical production workflows, robotic process automation in Industry 4.0, millions of business nodes, process fragments and the rapidly growing number of business rules [24, 25]. In recent decades, industry has paid attention to the management, coordination and optimization of project workflows. Besides, this trend has further accelerated in response to the growing prominence of digital engineering practices [26].

(3) Complexity: The inherent complexity of the process model has troubled stakeholders for a long time[27]. The fundamental reason is that the process model is a type of special data with a graphical representation and behavioral semantics. A small change of structure may result in a major change of behavior. The structure and behavior of the model are interdependent and coupled. However, existing approaches establish different storing and indexing dimensions by distinguishing structure and behavior when constructing process management systems. These methods can manage cases, when the model scale and number of nodes are relatively small. However, with the massive increase in the scale and number of models, it is obvious that the existing distinguishing structure and behavior can no longer effectively overcome the complexity of the model.

(4) Fast-Changing: With the continuous development of IIoT, industrial big data and cyber-physical systems , business processes are facing the challenge of fast-changing application scenarios and contexts[28]. A large number of model redesign[29], re-engineering[30] and refactoring[19] methods have been proposed to cope with the constant changes of industrial processes during the implementation process to respond quickly to the current complex and changeable production environment, such as flawed workflow controls, industrial transformation during the pandemic and unstable international scenarios.

In conclusion, the traditional method of separating structure and behavior for retrieval requires at least two or more times when both structure and behavior need to be retrieved, and there is no correlation between the two retrievals, thereby decreasing the retrieval efficacy of models. In addition, the PMR for Industry 4.0 is confronted by VVCC characteristics. Massive and complex process models must be swiftly retrieved from PMR using multidimensional association retrieval. The integrated structure and behavior retrieval proposed is therefore more appropriate for PMR management in Industry 4.0. An important challenge is how to manage techniques efficiently, accurately and easily.

In this paper, in order to efficiently query large model repositories for Industry 4.0, we propose a new method which support business process model indexing and retrieval with structure and behavior fusion. Complete finite prefix unfolding technology is used to extract the relationship between transitions from the process model and excluding relationships with conflicts, and then build an index (i.e., process tree) to greatly reduce the number of candidate models in the large PMR, and finally evaluate the process behavior similarity to redetermine candidates in the restriction stage model.

The contributions of this paper can be summarized as follows:

(1) To efficiently retrieve massive process models in the current Industry 4.0 environment, we propose a supporting structure and behavior fusion business process model indexing and retrieval method. It enhances the traditional single-dimensional retrieval method and improves the efficiency of the query.

(2) We propose a process model index generation paradigm which improves the existing process model index generation capabilities and ranges. Compared with traditional index generation methods, the proposed method can transform the *trace equivalent process model* (TEPM) with complex structures into a process tree (e.g., Fig.1), whereas traditional methods can only transform the *block-structured process model* (BSPM)[31] structure into a behavior-equivalent process tree.

(3) We improve process tree edit distance calculating methods for model similarity measurement and propose a calculation method for process tree similarity. The proposed method improves the traditional tree edit distance algorithm through distinguishing leaf nodes and non-leaf nodes.

The remainder of this paper is organized as follows: In Section 2, current indexing and search techniques for business process models are introduced. In Section 3, the related concepts involved in this study are introduced. In Section 4, the construction method for the model index based on the process tree is discussed in detail. In Section 5, a measurement method for model similarity is proposed. In Section 6, experiments for analyzing the proposed methods are presented. In Section 7, the study is summarized and directions for future work are suggested.

2 Related Work

Process retrieval involves entering search conditions in a specific format to the PMR, and then returning a set of process models that satisfy the requirements[32]. A process model is a type of special data that has both a graphical representation and behavioral semantics. Therefore, compared with querying traditional database systems, process retrieval is an important and challenging task that is still in the initial stage[28, 33-35]. Existing process retrieval methods can be classified into three categories according to the research content: process index

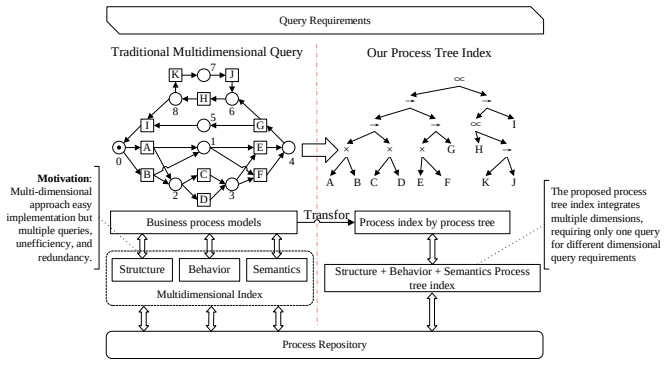

Fig.1. Motivational example of transferring a process model to process tree

construction, matching algorithm, and Process Query Language (PQL) [36-39]. Process query language, as a high-level interface, is generally determined by the low-level storage structure and index construction. The main focus in this study is the two other decisive aspects: process index construction and matching algorithm of the query.

The process index is considered to be an important enabling technology used to speed up model retrieval. Huang et al. [40] proposed an OPTR_index based on the quantitative ordering relation with time and probability constraints to query large process model repositories in a smart city cloud. Jin et al.[41] proposed a method to build a B+ tree based on the path as a structure-based precise index. Hofstede et al. [42] defined a process model query language based on the semantic relationship between tasks in the process model that is independent of any specific process modeling symbols to build the model index, thereby allowing users to formulate queries in a similar manner to modeling the conditional model. Mahleko et al.[43] proposed a business process index matching method based on finite state machine modeling. Beheshti et al.[34] introduced an extensible large-scale process data querying and analysis platform called ProcessAtlas, and provided services for discovering, extracting, and analyzing process knowledge graphs.

The matching algorithm is an important basis for searching PMR. Techniques for process model search can be divided into two main groups: (1) query based on graph structure; and (2) query based on behavior semantics[14, 31, 44, 45]. Jin et al.[46] used an index called TaskEdgeIndex for query processing, and estimated the minimum number of edges required to measure the structure similarity of business process models. Zha et al. [47] proposed transition adjacent relations to evaluate the model behavior similarity, and used an index to support behavior-based similarity model retrieval. Leopold et al.[44] addressed the textual description similarity problem and proposed a technique that can search textual as well as model-based process descriptions. Gómez-López et al.[48] proposed an architecture that integrates the business process, business process instance, and business data models using their meta-models to take advantage of the three models, and the technologies support querying the three isolated models. Brdjanin et al.[49] implemented an online web-based model-driven tool called AMADEOS, which automatically derives conceptual database models from process models that are represented by different notation and also serialized differently. Huang et al.[50] presented an approach to automating business process consolidation by applying process topic clustering based on business process libraries using a graph mining algorithm to extract process patterns, identify frequent subgraphs under the same process topic, fill the pertinent subgraph information into a table of frequent process subgraphs, and finally merge these frequent subgraphs to obtain merged business processes using a process merging algorithm.

Although some progress was made in the aforementioned studies, there are still relatively few research results regarding new VVCC features in industrial processes. In most of the aforementioned studies, the process model was regarded as a graph with both structure and behavior for storage and retrieval. However, many graph algorithms are computationally expensive (e.g., many are NP-complete) and the size of business process models essentially determines the runtime cost. Therefore, simply improving the accuracy of process retrieval can no longer meet the requirements for process retrieval efficiency in the current industrial big data era. Based on the above discussion, in this study, a new process model storage and index structure for efficient query analysis is proposed.

To the best of the authors' knowledge, this study is the first to express the behavior and structure of a model simultaneously, and it has no need to establish multiple retrieval dimensions. Using the method proposed in this study only requires calculating the similarity between tree structures, which can greatly improve the retrieval efficiency of the model.

3 PRELIMINARIES

3.1 Business Process and Process Tree

In this section, the basic concepts and notation related to Petri nets and complete finite prefix unfolding are reviewed, and will be used to support the discussion that follows.

Definition 1. Business process model. A business process model is a 4-tuple $N=(P, T; F, M_0)$ that satisfies the following:

(1) P is a place set and T is a transition set, where $P\cap T\neq\Phi$ and $P\cup T\neq\Phi$, and a place or transition is generically called a node.

(2) $F\subseteq P\times T\cup T\times P$ is the flow relation and generically called an arc.

(3) $M_0\subseteq P$ is the initial marking of $(P, T; F)$.

(4) For a node $x \in P \cap T$, the preset of x is denoted by $\cdot x$ and the postset of x is denoted by $x\cdot$.

The Petri net system is used to represent the process model, as Petri nets have a strict mathematical representation and are the most popular.. For other related concepts of Petri nets, please refer to [51, 52].

Definition 2. Process tree[53]. Let $N=(P, T; F, M_0)$ be a process model. Then a process tree can be defined:

(1) If $t \in T$, t is a process tree.

(2) Let $t_1,t_2,\ldots t_n$ (n>0) be process trees, $\odot$ be the behavior of $t_1,t_2,\ldots t_n$, and $\odot(t_1,t_2,\ldots t_n)$ be a process tree.

In this study, four types of relationship symbols are used: sequence ($\rightarrow$), parallel ($\wedge$), choice ($\times$), and loop ($\circlearrowleft$). $\odot$ indicates one of the relationships. $\odot$ represents the relationship between multiple transitions. $\odot$ can be empty, or contain one or more relationships. For related concepts about process trees, refer to [53, 54].

3.2 Complete Finite Prefix Unfolding

To extract the behavior of a business process, it is necessary to analyze the process model. The traditional analysis method mainly uses the reachable tree or reachable graph[55, 56], but these methods encounter the problem of state space explosion. To avoid this problem, Esparza et al. [57, 58] proposed complete finite prefix unfolding. It can expand the process model into a branching process that contains an occurrence net and cut-off events. It uses the set of possible extensions of the branching process and continuously expands the branching process until the set of possible extensions becomes an empty set or a cut-off event is encountered.

Definition 3. Occurrence net[58]. A 3-tuple $o=(B, E; F')$ is an occurrence net, where B denotes the conditions set and E denotes the events set. $B \cap E \neq \Phi$ and $B \cup E \neq \Phi$, and $F' \subseteq B \times E \cup E \times B$, which satisfy the following:

(1) $\forall b \in B, |\cdot b| \leq 1$.

(2) $\forall e \in E$, E is not in self-conflict.

(3) $B \cup E$ is a finite set.

(4) $(B, E; F')$ is an acyclic net.

Min(N) denotes the set of minimal nodes of $B \cup E$ with respect to the transitive closure of F'.

Given two nodes $x, y \in B \cup E$, the relationship between x and y is as follows:

(1) There is a path from x to y, and the relationship between x and y is denoted by $x<y$.

(2) There is a condition b such that the path from b to x does not intersect the path from b to y, and the relationship between x and y is denoted by $x\#y$.

(3) If neither $x<y$, $y<x$, nor $x\#$y, the relationship between x and y is denoted by x**co**y.

Definition 4. Branching processes. Let $N=(P, T; F, M_0)$ be a business process model and $\Pi=(o, h)$ be a branch process corresponding to N, where $o=(B, E; F')$ is an occurrence net and h is a homomorphism that satisfy the following:

(1) $h(B) \subseteq P$, $h(E) \subseteq T$, $h(F') \subseteq F$.

(2) Min(o) and M_0 is a bijection relationship.

(3) For each $e_1, e_2 \in E$, if $\cdot e_1 = \cdot e_2 \wedge h(e_1)=h(e_2)$, then $e_1=e_2$.

Definition 5. Configurations. A configuration C of an occurrence net is a set of events that satisfy the following:

(1) $e \in C \Rightarrow \forall e' \prec e$: $e' \in C$, where C is causally closed.

(2) $\forall e, e' \in C$: $\neg(e \# e')$, where C is conflict-free.

$\forall e \in E$, the local configuration of e is a set of non-conflicting events that include e itself, which is denoted by $[e]$. Additionally, $\forall e_1 \in [e]$, e_1 satisfies $[e_1] \prec [e]$ and any $e_2 \in [e]$ has $\neg e_2 \# e_1$.

The symbol $\prec$ indicates the adequate order relationship between configurations, and refines $\subset$; that is, if [e]$\subset$[e'], then [e]$\prec$[e'] [58].

Definition 6. Cut. For a configuration C, the cut of C is a co-set, which is defined as $Cut(C)=(Min(N) \cup C\cdot)\backslash \cdot C$.

$Cut([e])$ represents the conditions reached by the configuration $[t]$. For any $b \in Cut([e])$, b=$\emptyset$ must hold. $h(Cut(C))$ is denoted by $Mark(C)$.

Definition 7. Cut-off event. Let $\Pi=(o,h)$ be a branch process corresponding to N, where $o=(B, E; F')$ is an occurrence net. An event e is a cut-off event iff Π contains a local configuration $[e']$ such that $Mark([e])=Mark([e'])$ and $[e'] \prec [e]$, which is denoted by $corr(e)=e'$, and also referred to as e' is cut off by e.

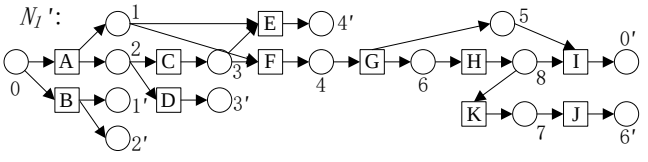

Fig.2. Unfolding instance of N in Fig.1.

N' in Fig.1 is an unfolding instance of N in Fig.2. N' demonstrates the branching processes of N, where $[A]=\{A\}$, $[G]=\{A, C, F, G\}$, and $[J]=\{A, C, F, G, H, K, J\}$. Because $Mark([G])=\{5, 6\}$, $Mark([J])=\{5, 6\}$, and $[G] \prec [J]$, J is a cut-off event and $corr(J)=G$.

4 Architecture of PMR for Industry 4.0

The architecture of PMR for Industry 4.0 is shown in Fig.3. The architecture follows a multilayer model and consists of four parts: Interface, Index Generation, Query process, and Storage. In the context of Industry 4.0, multiple intelligent manufacturing activities in the cyber-physical space are driven and controlled by business process models. Process engineers can store and retrieve business process models through the PMR interface.

While the model is being stored, it is input to the index generation module, through which the process model is unfolded to obtain branching processes for subsequent operations. Then a process tree is generated and nodes of the tree are merged based on the extracted transition

relations. Next, a partial relation matrix is obtained as an index of the model. Finally, the process model with its corresponding index is stored in the PMR. The index generation module consists of two main components: process tree and partial relationship matrix generation. The details of this part are presented in Section 5.

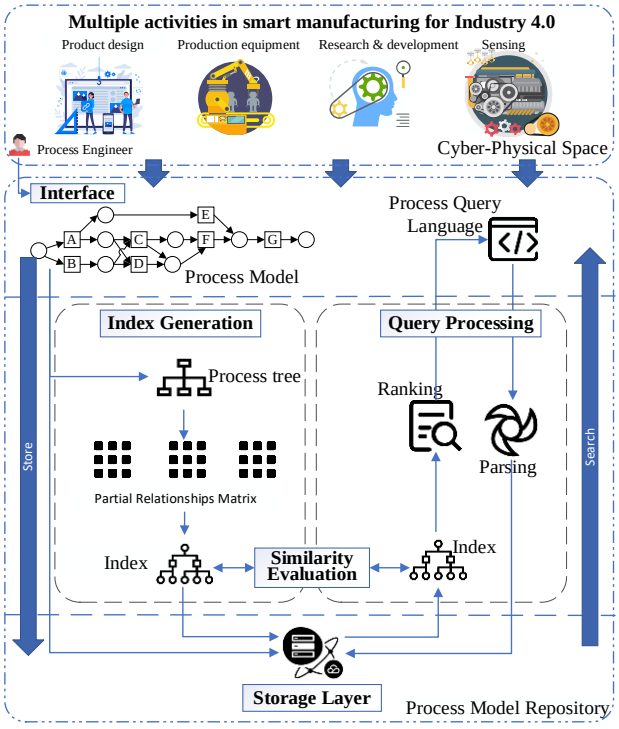

Fig.3. Architecture of PMR for Industry 4.0.

During the search stage, process engineers can input PQL using the interface to specify query constraints. After the semantic analysis of PQL, the constraint conditions are input into the PMR and an inverted table of query results is generated based on similar values between the process tree and the model index. Finally, through the ranking of the query results, the final query results are returned. The search phase is a method used to improve the retrieval efficiency using the process tree obtained in the store phase. The details of this part are presented in Section 6.

5 Index Generation

To improve the retrieval efficiency, the process tree is used as the index of the process model, which incorporates the structure and behavior of the model, changes the traditional single-dimensional retrieval method, and improves the existing process model index generation capabilities and ranges. The algorithm for the index construction of the process model can be divided into six steps:

(1) Complete finite prefix unfolding. Unfold the model to obtain branching processes for subsequent operations.

(2) Extract transition relationships. Determine whether there is a mergeable relationship between each pair of transitions in the model, and if such a relationship exists, save it in the relationship matrix *RM*.

(3) Determine whether there is a reconfigurable transition relationship. Remove the conflicting relationships and select the high priority relationships from the *RM*. Then, obtain a high-priority and conflict-free set of relationships list *RL*. If the *RL* is empty, go to (4); otherwise, go to (5).

(4) Generate the partial relationship matrix *pRM*. A complex structure exists in the model, which has no corresponding behavior-equivalent process tree. Therefore, in this step, according to the branching processes obtained in the first step, select the locally smallest complex structure in the process model and transform it into a *pRM*. Then go to (6).

(5) Reconstruct the relationships. Merge, in turn, the relationships in *RM*.

(6) Determine if there are multiple transitions. If so, go to (1); otherwise, output the process tree.

In Algorithm 1, the input is a process model and the output is a process tree. Consider that there may be "fault structures" in the actual input process model. These "fault structures" do not appear in the branching process, and thus are not recognized by subsequent operations, but they cause the algorithm to enter into an infinite loop. Hence, adding a variable ***flag*** to the pseudocode prevents the algorithm from entering into an infinite loop.

ExtractRelation() in Algorithm 1 is detailed in Algorithm 2 in Section 5.1, *GetRList*() and *Refactor*() are detailed in Algorithms 3 and 4 in Section 5.2, and *PRMGeneration*() is detailed in Section 5.3.

Algorithm 1. GenerationOfTheProcessTree

Input: a model $N=(P, T; F, M_0)$
Output: a process tree t

```
1   flag=true
2   while flag:
3     flag=false
4     get a branching process Π=(o,h) with N
5     RM=ExtractRelation(N, Π)
6     RL=getRList(RM)
7     if (RL.size()==0) {
8       N'=PRMGeneration(N, Π)
9       flag=N'.T.size()==N.T.size()
10      if (flag) {N=N'} }
11    else foreach( ⊙(e1, e2)∈RL ) {
12      if (⊙(e1, e2) ⊂ N.T) {
13        p=Refactor(N, ⊙(e1, e2))
14        flag=true }}
15  return N.T
```

5.1 Determine the Transition Relationship

In this section, the method for extracting model behavior based on the unfolding net is introduced in detail. First, some relevant information can be easily obtained from the branch process.

The specific information required is shown in Table 1. It

TABLE 1
INFORMATION ABOUT THE UNFOLDING IN FIG.1

o.T	*CM*											*oRM*											*corr*	*h*
	A	B	C	D	E	F	G	H	I	J	K	A	B	C	D	E	F	G	H	I	J	K		
A	✓												<	<	<	<	<	<	<	<	<	<		A
B	✓	✓										>		#	#	<	#	#	#	#	#	#		B
C	✓		✓									>	#		<	#	<	<	<	<	<	<		C
D	✓		✓	✓		✓						>	#	>		#	#	<	#	#	#	#		D
E	✓	✓			✓							>	>	#	#		#	#	#	#	#	#	F	E
F	✓		✓			✓						>	#	>	#	#		<	<	<	<	<		F
G	✓		✓	✓		✓	✓					>	#	>	>	#	>		#	#	#	#	F	G
H	✓		✓			✓		✓				>	#	>	#	#	>	#		#	<	#		H
I	✓		✓			✓			✓			>	#	>	#	#	>	#	#		#	<		I
J	✓		✓			✓		✓		✓		>	#	>	#	#	>	#	>	#		#		J
K	✓		✓			✓			✓		✓	>	#	>	#	#	>	#	#	>	#			K

CM[x][y] indicates that the configuration of x contains y. "<," "#," and "co" can be found in Definition 3, and the symbol ">" represents inverse "<." corr denotes the cut-off events. h is a homomorphism of the branching processes.

includes two two-dimensional arrays, *CM* and *oRM*; and two one-dimensional arrays, *corr* and *h*. *CM* represents the configuration relationship between events. For example, $CM[F][A]=\checkmark$ indicates that the configuration of *F* contains *A*. *oRM* represents the relationship between events in the occurrence net.

The above information can determine the transition relationships in the business process model. The following four conditions are provided for determining the behavior.

Let $N=(P, T; F, M_0)$ be a business process; $\Pi=(o,h)$ be a branch process corresponding to *N*, where $o=(B, E; F')$ is an occurrence net; and *h* be a homomorphism, where $e_1, e_2 \in E$, $t_1, t_2 \in T$, $h(e_1)=t_1$, and $h(e_2)=t_2$.

Definition 8. Sequence judgment condition. e_1 and e_2 satisfy the following conditions for a sequence relationship between t_1 and t_2, which is denoted by $\rightarrow(t_1, t_2)$:

- $oRM[e_1][e_2]=<$: the relationship between e_1 and e_2 is $e_1<e_2$.
- For every $e\in E$ and $e\neq e_1 \wedge e\neq e_2$,

(1) $CM[e_1][e]=CM[e_2][e]$ and $CM[e][e_1]=CM[e][e_2]$;

(2) when $h(e)=h(e_2)$, $CM[e_1][e]\neq\checkmark$ and $h(\cdot\cdot e)=h(e_1)$ must hold; and

(3) when $corr(e)=e_1$, either $h(e)\neq h(e_1)$ or $oRM[e][e_2]\neq <$, or the presence of $e\in E$ and $e\neq e_1 \wedge e\neq e_2$ makes $CM[e][e']\neq CM[e_2][e']$.

Definition 9. Parallel judgment condition. e_1 and e_2 satisfy the following conditions for a parallel relationship between t_1 and t_2, which is denoted by $\wedge(t_1, t_2)$ or $\wedge(t_2, t_1)$:

- $oRM[e_1][e_2]=$ **co**: the relationship between e_1 and e_2 is e_1 **co** e_2.
- For every $e\in E$ and $e\neq e_1 \wedge e\neq e_2$,

(1) $CM[e_1][e]=CM[e_2][e]$ and $CM[e][e_1]=CM[e][e_2]$; (2) when $h(e)=h(e_1)$, $CM[e_2][e]=\checkmark$ must hold; and

(3) when $h(e)=h(e_2)$, $CM[e_1][e]=\checkmark$ must hold.

Definition 10. Choice judgment condition. e_1 and e_2 satisfy the following conditions for a choice relationship between t_1 and t_2, which is denoted by $\times(t_1, t_2)$ or $\times(t_2, t_1)$:

- $oRM[e_1][e_2]=\#$: the relationship between e_1 and e_2 is $e_1 \# e_2$.
- For every $e\in E$ and $e\neq e_1 \wedge e\neq e_2$, $CM[e_1][e]=CM[e_2][e]$.
- $corr(e_1)=e_2$ or $corr(e_2)=e_1$: an event cuts off the other event.

Definition 11. Loop judgment condition. e_1 and e_2 satisfy the following conditions for a loop relationship between t_1 and t_2. If t_1 is executed first, it is denoted by $\circlearrowleft(t_1, t_2)$. If t_2 is executed first, it is denoted by $\circlearrowleft(t_2, t_1)$:

- $oRM[corr(e_2)][e_1]=<$ and $oRM[e_1][e_2]=<$.
- For every $e\in E$ and $e\neq e_1 \wedge e\neq e_2$, $CM[e_1][e]=CM[e_2][e]$.
- For every $e\in E$ and $e\neq e_1 \wedge e\neq corr(e_2)$, $CM[e_1][e]=CM[corr(e_2)][e]$.
- If $h(e_2)\neq h(corr(e_2))$, there is a loop relationship between t_2 and t_1, and t_1 is executed first. If $h(e_2)=h(corr(e_2))$, there is also a loop relationship between t_2 and t_1, but t_2 is executed first.

Table 1 shows all the event information of the unfolding instance P' in Fig.2. From the information in Table 1 and the above four relationship judgment methods, the four relationships ×(A, B), ×(C, D), ×(E, F) and →(K, J) can be extracted from Fig.1. Using the above four relationship judgment methods, Algorithm 2 provides the pseudocode of the *ExtractRelation*() algorithm for generating *RM*.

Algorithm 2. ExtractRelation

Input: a model $N=(P, T; F, M_0)$, $\Pi=(o,h)$
Output: *RM*

```
get an branching processes Π=(o,h) with p
E=o.E, get corr, h, oRM, CM form Π
RM=[len(p.T)][len(p.T)]
foreach (e1∈E){
  foreach (e2∈E and e1!=e2){
    if isSequnence(e1, e2) {
      RM[h(e1)][h(e2)].add(→(h(e1), h(e2))) }
    if isIteration(e1, e2) {
      RM[h(e1)][h(e2)].add(↻(h(e1), h(e2))) }
    if isChoice(e1, e2) {
      RM[h(e1)][h(e2)].add(×(h(e1), h(e2))) }
    if isConcurrency(e1, e2) {
      RM[h(e1)][h(e2)].add(∧(h(e1), h(e2))) } }}
return RM
```

5.2 Select and Merge the Transition Relationships

Next, it is necessary to select mergeable relationships,

which satisfy the condition that they will not affect other relationships after reconstruction. The relationships extracted above are determined according to events in the branching process. Definition 12 is as follows.

Definition 12. Combinable relationship judgment condition. When $\odot(t_1, t_2)$ satisfies the following conditions, the relationship between t_1 and t_2 can be combined. For any $\odot'(t_1', t_2') \in RM$ and $\{t_1, t_2\} \cap \{t_1', t_2'\} \neq \Phi$:

(1)$\odot$ only represents one relationship;

(2)$\odot = \rightarrow$ when $\odot' = \rightarrow$;

(3)$\odot = \times$ when $\odot' = \times$ or $\odot' = \wedge$;

(4)$\odot = \wedge$ when $\odot' = \wedge$; and

(5)$\odot = \circlearrowleft$ when $\odot' \neq \circlearrowleft$or $t_1' \neq t_2 \wedge t_1 \neq t_2'$.

Merging iterative relations merges the flow relations in different directions into one place, which easily affects other transition relations. Therefore, iterative relations are not merged when there are other mergeable relations. Algorithm 3 is the transition relation algorithm *GetRList*() for selecting and merging.

Algorithm 3. GetRList

Input: a behavior matrix RM
Output: *RL*

```
1   tRL= {} IRL= {} RL= {}
2   for ( i=0; i <RM.size; i++) {
3     for ( j=0 ; j<RM.size; j++) {
4       if ( i≠j and len(RM[i][j].⊙)≠0) {
5         tRL.add(RM[i][j]) } }}
6   foreach (⊙(t1, t2)∈RM){
7       Falg=len(⊙)==1
8       if (!falg) foreach (⊙'(t1', t2')∈tR){
9         if ({t1, t2}∩{t1', t2'}≠∅) {
10          if(⊙==→) {
11            if(⊙'!=→){flag=false, break}}
12          else if(⊙==×) {
13            if(⊙'==→ or ⊙'==↻) {
14              flag=false, break}}
15          else if(⊙==↻)  {
16            if(⊙'!=↻ or t1'==t2 or t1==t2') {
17              flag=false, break}}
18          else if(⊙==∧) {
19            if(⊙'!=∧) {flag=false, break}} }}
20        if falg (
21          if(⊙==↻) {IRL.add(⊙(t1, t2))}
22          else {RL.add(⊙(t1, t2))}
23  if(IRL.size > 0) {return RL}
24  return IRL
```

The basic idea of merging transition relationships is shown in Fig.4: delete the original two transitions and add a new transition. Simultaneously, selectively make the new transition inherit parts of the arcs of the old transitions.

The retained arcs are indicated by the blue and red flow relationships in Fig.4, where red indicates the arcs pointing to the new transition and blue indicates the arcs starting from the new transition. The retained arcs can be divided into two types according to the transition relationship:

(1) The relationship is not a loop relationship. At this time, after excluding the conditions that need to be deleted, the new transition inherits the arcs of the two old transitions.

(2) The relationship is a loop relationship, which allows the new transition to inherit the arcs of the transition that executed first in the loop relationship (i.e., loop(1) in Fig.4). However, sometimes, the successor of the first transition is a condition that needs to be deleted (i.e., loop(2) in Fig.4). At this time, the new transition a inherits the arcs that point to the transition a_1 that executed first in the loop relationship, and the arcs that starting to the transition a_2 that executed last.

Next, it is necessary to delete some of the conditions and their associated arcs, including the conditions *delC* in Fig.4. The conditions to be deleted can be divided into three scenarios:

(1) does not belong to the initial modality, but only connects the two old transitions (i.e., the black condition in Fig.4);

(2) No other transitions are related; that is, both the preset and postset are empty (may occur after removing part of the arcs); and

(3) if there are two conditions of the same preset and the same postset, delete one of them.

Case (1) of the selection of the conditions is likely to affect the selection of the arcs and needs to be determined before the arcs are selected. The latter two cases of the selection of the conditions are affected by the selection of the arcs, and need to be judged after the arcs are selected.

Refactor() is used to merge transitions, where the selection of arcs corresponds to lines 5–8 of Algorithm 4. The three scenarios for the deletion conditions correspond to rows 2, 10, and 11–15 of Algorithm 4.

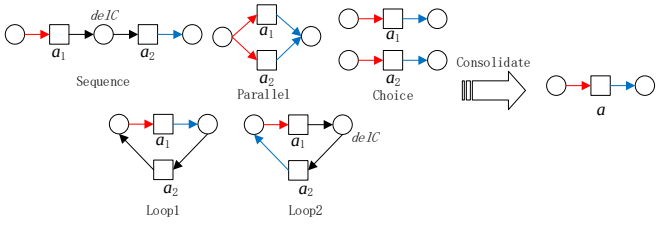

Fig.4. Schematic of Algorithm 4: Refactor.
Delete the original transitions, add transitions, and selectively make the new transitions inherit parts of the arcs of the original transitions. Red represents arcs to be retained that point to a new transition, and blue represents arcs to be retained that start from a new transition.

Algorithm 4. Refactor

Input: a model $N=(P, T; F, M_0)$, $\odot(t_1, t_2)$
Output: a model $N=(P, T; F, M_0)$

1 *newT*=$\odot(t_1, t_2)$, *delT*={t_1, t_2}

delP={*p*|*p*∈*N.P*-*N.M*$_0$ **and** ·*p*∩*p*·⊆*delT*}
delF={*e*$_1$×*e*$_2$|*e*$_1$×*e*$_2$∈*N.F* **and** {*e*$_1$, *e*$_2$}∩(*delT*∪ *delP*) hØ}
if (⊙==♉)
  F'=*N.F* – *delF*+{*p*×*newt*|*p*∈(·*t*$_1$-*delP*)}
    +{*newt*×p|*p*∈((*t*$_1$·⊆*delP*?*t*$_2$·:*t*$_1$·)-*delP*)}
  else
    F' =*N.F* – *delF*+{*newt*×p|p∈((*t*$_1$· ∪ *t*$_2$·)
    –(·*t*$_1$ ∪ ·*t*$_2$) –*delP*)}+{p×*newt*|p∈((·*t*$_1$
    ∪ ·*t*$_2$)-(*t*$_1$· ∪ *t*$_2$·)-*delP*)}
  P'=*N.P'*-*delP*, *M*$_0$'=*N.M*$_0$
foreach (*p*∈*P'*){
  if (·*p*==Ø **and** *p*·==Ø) {*P'*=*P'*-*p*}
  else foreach (*p'*∈*P'* **and** *p*≠*p'*){
    if(*p*∈*M*$_0$ XNOR *p'*∈*M*$_0$ **and** ·*p*=·*p'* **and** *p*·=*p*]·){
    F'={*e*$_1$×*e*$_2$|*e*$_1$×*e*$_2$∈*F'* **and** {*e*$_1$, *e*$_2$}∩{*p*}=Ø}
      if(*p*∈*M*$_0$) {*M*$_0$'=*M*$_0$'-{*p*}}
      P'=*P'*-{*p*}, **break** }}}
return *N'*=(*P'*, *N.T*+{*newt*}-*delT*; *F'*, *M*$_0$')

5.3 Partial Relationship Matrix Generation

The model N_2 in Fig.5 has no corresponding behavior-equivalent process tree. Because there are ∧(*d*, *e*), ∧(*b*, *c*), ∧(*b*, *c*), →(*b*, *d*), →(*b*, *e*), and →(*c*, *e*) in N_2. The transitions *b*, *e*, *c* form →(∧(*b*, *c*), *e*). When *d* is going to join →(∧(*b*, *c*), *e*), then it forms →(∧(*b*, *c*), ∧(*d*, *e*)) or →(∧(→(*b*, *d*), *c*), *e*). The former causes the relationship between *c* and *d* to change from parallel to sequential, and the latter causes the relationship between *d* and *e* to change from parallel to sequential. Both combinations are wrong. In fact, no combination can produce a behavior-equivalent process tree of N_2.

Therefore, how to select the "appropriate" complex structure for *PRMGeneration*() is the problem to be solved in this section.

The basic steps of the algorithm *PRMGeneration*() are described in detail as follows and illustrated by the process model N_2 in Fig.5:

Let *Π*=(*o*,*h*) be a branch process corresponding to N_2=(*P*, *T*; *F*, *M*$_0$), where *o*=(*B*, *E*; *F'*) is an occurrence net; *h* be a homomorphism; *CM* represent the configuration relationship between events from *Π*; and *corr* denote the cut-off event.

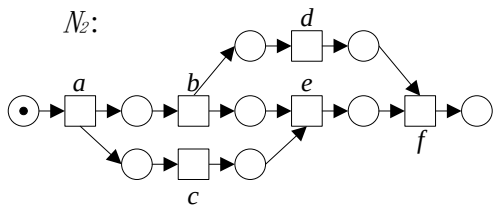

Fig.5. Example of process model N_2 containing a complex structure.

(1) Construct the binary matrix *CM'*. The matrix *CM'* is constructed from *CM* and the homomorphism *h*. For every *e*$_1$, *e*$_2$∈*E*, *CM'*[*h*(*e*$_1$)][*h*(*e*$_2$)]=*CM'*[*h*(*e*$_1$)][*h*(*e*$_2$)]|*CM*[*e*$_2$][*e*$_1$]. The *CM'* of N_2 is shown in Table 2.

(2) Construct the binary matrix *hCM'*. The binary matrix *CM'* is constructed from *CM'* and *corr*. Let *hCM'*= *CM'*, and for any *corr*(*e*$_1$)=*e*$_2$ and ∀*e*∈*E*, let *hCM'*[*e*][*e*$_1$]=*hCM'*[*e*][*e*$_1$]|*hCM'*[*e*][*e*$_2$]. There is no cut-off event in N_2; hence, *hCM'* of N_2 is the same as *CM'*.

TABLE 2
CM' IN FIG.5

o.T	*CM'*					
	a	*b*	*c*	*d*	*e*	*f*
a	✓					
b	✓	✓				
c	✓		✓			
d	✓		✓	✓		
e	✓	✓	✓		✓	
f	✓	✓	✓	✓	✓	✓

(3) The global relation is extracted to construct the relation matrix *pRM*. The relationship between *t*$_1$, *t*$_2$∈*T*∧*t*$_1$≠*t*$_2$ is judged using a quadratic nested cycle. First, *pRM*=*RM* is set to reduce the number of judgments. The judgment method of the remaining transition relationship is as follows:

I. When *hCM'*[*t*$_1$][*t*$_2$]=✓ and *hCM'*[*t*$_2$][*t*$_1$]=✓, ♉(*t*$_1$, *t*$_2$).

II. When *hCM'*[*t*$_1$][*t*$_2$]=✓ and *hCM'*[*t*$_2$][*t*$_1$]≠✓, →(*t*$_1$, *t*$_2$).

III. When *hCM'*[*t*$_1$][*t*$_2$]≠✓ and *hCM'*[*t*$_2$][*t*$_1$]≠✓, transition set *Ts*={*t*|*t*∈*T* and *CM'*[*t*][*t*$_2$]|*CM'*[*t*][*t*$_1$]}, and conditions set *C*=(*Min*(*o*)∪*Ts*·)\·*Ts* are constructed. The calculation method of *C* is similar to that in Definition 6. If any C[{*t*$_1$, *t*$_2$}>, then ∧(*t*$_1$, *t*$_2$); otherwise, ×(*t*$_1$, *t*$_2$).

(4) Retain transitions that form a complex structure in *pRM*. When *t*$_1$∈*T* is removed from *pRM*, *t*$_1$ has the same relationships with any *t*$_2$∈*T*∧*t*$_1$≠*t*$_2$. Loop to check whether the transitions in *pRM* can be removed, until any transitions does not satisfy the removal condition.

(5) Refactor the behavior. Use *Refactor*() to refactor the transition relationships in *pRM*.

Finally, through the above basic steps of the algorithm *PRMGeneration*(), the index of process model N_2 is obtained as Fig.6.

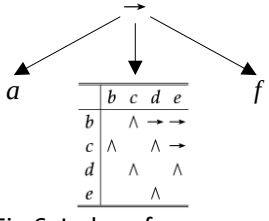

Fig.6. Index of process model N_2.

6 QUERY PROCESSING

After the index for the business process is built, the next step is how to query using the index. The process tree is used as the index; hence, the similarity between the process trees needs to be measured.

6.1 Process Tree Edit Distance

The tree edit distance is mainly inspired by the traditional concept of the string editing distance. The definition of

the process tree edit distance is as follows:

Definition 12. Process tree edit distance. The edit distance between process trees t_1 and t_2 is expressed as $\delta(t_1, t_2)$, where $\delta(t_1, t_2)$ denotes the sum of the minimum overhead of the basic operations required to convert t_1 and t_2 into the same process tree.

The basic operation[59] does not distinguish leaf nodes and non-leaf nodes. The cost of deleting node T is denoted by $Del(T)$, and the cost of modifying the relationship between nodes t_1 and t_2 is denoted by $Rel(t_1, t_2)$.

A process tree is different from a tree in that branch nodes represent the relationships between transitions (leaf nodes). Hence, the deleting node is set only for leaf nodes and the modifying node is set only for branch nodes.

For example, once the node ∧(*a*, *b*) is deleted, and the nodes *a* and *b* will be removed. Only the∧operation symbol does not make any sense. Similarly, only deleting node *a*, then get node ∧(*b*),∧also does not make sense.

Therefore, the deleting node is set as an operation on the leaf nodes.

Calculating the edit distance for t_1=→(*a*, ×(*b*, *c*), *d*) and t_2=→(*a*, ∧(*b*, *c*), *d*) requires modifying node x(*b*, *c*) in t_1 or node ∧(*b*, *c*) in t_2. The two nodes are included in *b* and *c*. Deleting ×(*b*, *c*) and adding ∧(*b, c*)is not reasonable. Thus, modifying nodes is an operation on the branch nodes (relationship). Additionally, modifying the amendment of ×(*b*, *c*) to ∧(*b*, *c*) is equivalent to modifying the relationship between *b* and *c*.

The traditional tree edit distance calculation adopts a type of dynamic recursion[59]. However, combined with the characteristics of the process tree, it does not need dynamic recursion to calculate the edit distance. Based on Definition 12, the calculation of the process trees' edit distance requires only two components: the cost of the transitions that need to be removed and the cost of the transition relationships that need to be modified.

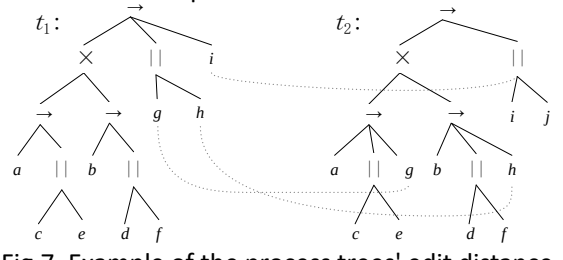

Fig.7. Example of the process trees' edit distance.

The operation overhead of deleting a leaf node *t* is denoted by $Del(t)=p$, and the overhead of modifying the relationship between node t_1 and t_2 is denoted by $Rel(t_1, t_2)=q$. The calculation method for the process trees' edit distance is refined into the following three steps, and t_1 and t_2 in Fig.7 are taken as examples to illustrate it:

(1) $Retain=\{t \mid t \in \text{leaves}(t_1) \text{ and } t \in \text{leaves}(t_2)\}$, where leaves($t_1$) represents all the leaf nodes of t_1. *Retain* is the nodes to be reserved in process trees t_1 and t_2. In Fig.7, *Retain*={*a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*, *I*).

(2) Calculate the cost of deleting nodes. In t_1 and t_2, the cost of deleting nodes is $p\times(|leaves(t_1)|+|leaves(t_2)|-2\times|Retain|)$. However, it is not necessary to delete the nodes in t_1 and t_2 because they will not affect subsequent operations.

(3) $Rel=\{(x,y) \mid \odot(x,y) \in t_1 \wedge \odot'(x,y) \in t_2 \wedge \odot \neq \odot'\}$. *Rel* is the branch nodes to be modify in process trees t_1 and t_2. In Fig.7, *Rel*={(*a*, *h*), (*c*, *h*), (*e*, *h*), (*b*, *g*), (*d*, *g*), (*f*, *g*), (*g*, *h*)}, and the cost of modifying the branch nodes is $q\times|Rel|$. In the actual calculation, it is not necessary to calculate the specific matrix, and the transition relationship can be obtained by tracing the nearest common parent node between leaf nodes.

6.2 Process Model Query

The similarity between the process trees is calculated using the process tree edit distance. The definition of process tree similarity is provided in the following.

Definition 13. Union set value. For process trees t_1 and t_2, let the cost of deleting node *a* be denoted by $Del(a)=p$, and the cost of modifying the relationship between node a_1 and a_2 be denoted by $Rel(a_1, a_1)=q$. Then the union set value between t_1 and t_2 is

$$|t_1 \cup t_2| = (|t_1| + |t_2|) \times p + \frac{|Retain(t_1,t_2)| \times (|Retain(t_1,t_2)| - 1)}{2} \times q \,. \quad (1)$$

Definition 14. Process tree similarity. The similarity between process trees t_1 and t_2 is

$$simTree(t_1,t_2) = \frac{|t_1 \cap t_2|}{|t_1 \cup t_2|} = \frac{|t_1 \cup t_2| - \varphi(t_1,t_2)}{|t_1 \cup t_2|} = 1 - \frac{\varphi(t_1,t_2)}{|t_1 \cup t_2|} \,. \quad (2)$$

Two process trees are presented, as shown in Fig.8. The similarity of the model is measured. The similarity between t_1 and t_2 process trees is measured by setting q=1 and p=1, and $simTree(t_1, t_2)$ =0.8 is obtained.

Using Definition 6, a process model retrieval method based on process tree similarity measurement is proposed. The pseudocode is shown in Algorithm 5. The input retrieval condition *QC* can be a process model or process tree.

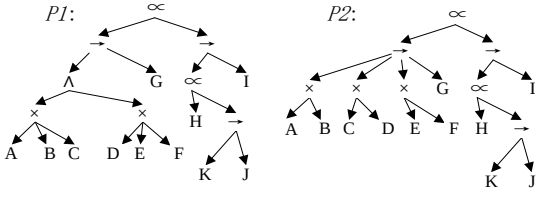

Fig.8. Process trees.

Algorithm 5. Similarity calculation for the process model

Input: The retrieval condition QC, process model set to be retrieved NS, and index set TS
Output: a behavior list *IRL*
1 **if (***qc* is not tree**)**
2 get a process tree *t* from *qc*

```
3    else t=qc
4    t=standardize(t)
5    rTS=[]
6    foreach (t'∈TS)
7      similarity=simTree(t, t')
8       if (similarity > 0)
9          Insert t' into rTS with similarity
10   return rTS
```

7 Evaluation

The main aim of the experiment for process model retrieval is to analyze its feasibility and efficiency.

In this section, the detailed process model retrieval case is analyzed to verify the feasibility of index generation. Then, the retrieval efficiency of the query process is analyzed. Finally, the performance of Algorithm 5 is evaluated.

The authors uploaded the experimental data and source code for this algorithm to GitHub[1]. Using the method in this study, a prototype system was developed[2]. Notably, the experimental data utilised in this paper have passed our tests to ensure that they are free of noise interference.

7.1 Experiment Datasets

The experimental datasets used to evaluate the performance of the algorithms are divided into three groups:

(1) AP group: 10,000 randomly and automatically generated models were recorded using PLG[60]. These PLG-generated models contain four basic structures, that is, sequence, selection, concurrency, and iteration, and their node labels are composed of character numbers.

(2) BP group: 1,000 process models proposed by Polyvyanyy et al.[61]. These models contain not only the four basic structures, but also some complex non-TEPM structures.

(3) CP group: This repository contains the hand-drawn BPMN dataset published in [62]. 651 representative real-life process models were contained. This group contains BSPM models, TEPM models with complex structures, and non-TEPM models.

(4) DP group: 20 representative artificially constructed process models. This group contains BSPM models, TEPM models with complex structures, and non-TEPM models. Specific models can be found in Table 5.

TABLE 3
Experimental Datasets

	counts	*pmin*	*pmax*	*psum*	*tmin*	*tmax*	*tsum*	*fmin*	*fmax*	*fsum*
AP	10000	7	28	133184	6	29	126633	12	66	287088
BP	1000	5	84	13516	5	86	16631	10	84	32398
CP	651	7	62	13839	7	54	13410	11	62	28151
DP	20	5	15	155	5	16	173	14	15	424

pmin, pmax, and pavg denote the minimum, maximum, and average numbers of places of models in the dataset. tmin, tmax, and tavg denote the minimum, maximum, and average numbers of transitions of models in the dataset. fmin, fmax, and favg denote the minimum, maximum, and average numbers of arcs of models in the dataset.

Table 3 shows the specific conditions of the experimental data: *counts* represents the total number of models in the dataset.

7.2 Correctness Verification of Index Generation

To verify the correctness of index generation in transforming a complex structure, DP experimental data were used for analysis. The 20 process models in the DP group,as shown in Table 4, can be divided into three types:

(1) The models were obtained using random superposition and a combination of sequence, parallel, choice, and loop. The models correspond to the process models numbered 1–7. These models are TEPM models, and include some BSPM models.

(2) By modifying some classic process model cases (e.g., courier protocol, accurate colored, and colored reader writer), the obtained process models correspond to the process models numbered 8–15, which are TEPM models with complex structures.

(3) Representative non-TEPM models were selected as BP group data, and the simple BSPM structure in these models was deleted to reduce the scale of the model and make it easy to analyze. The models correspond to the process models numbered 16–20.

Table 5 shows that the algorithm presented in this study has advantages over other methods regarding dealing with TEPM models with complex structures, and better preserves model behavior. Table 5 compares the proposed Algorithm 1 with [53, 54] in the dataset DP. The [63] converts BSPMs into behavioral-equivalent process trees by identifying basic blocks, but cannot handle non-BSPMs. The [53, 54] convert all models into process trees, but generalize non-BSPM to flower models. The [64] does not generate process trees, but can convert non-BSPMs to BSPMs, and then obtain the corresponding process tree by identifying the basic block. However, this method cannot be applied to the loop structure. To sum up, the existing methods cannot convert the models in Table 5 into behavioral-equivalent process trees.

7.3 Indexing Efficiency Analysis

Before the performance of the process model index building algorithm is analyzed, the time complexity of Algorithm 1 is analyzed. The time complexity of *GenerationOfTheProcessTree*() is determined using the complexity of the complete finite prefix unfolding, complexity of *ExtractRelation*(), complexity of *Refactor*(), and number of iterations (process tree depth).

First, consider the complexity of the algorithm in the worst case. The complete finite prefix unfolding algorithm is $O(|T|\cdot R^{s})$ in the worst case, where $|T|$ is the number of transitions in the model, R is the number of non-

[1]https://github.com/zhu-rui/Process-model-repository [2]http://47.110.142.51:8080/ProcessModel/

TABLE 4 (REFER TO SUPPLEMENTAL FILE)
20 EXPERIMENTAL CASES IN THE DP DATASETS

TABLE 5
RESEARCH ON PROCESS MODEL RETRIEVAL

NUM	WHETHER TO CONVERT				WHETHER TO LOSE BEHAVIORAL				WHETHER TO GENERALIZE TO A FLOWER MODEL				WHETHER TO GENERALIZE TO A BLOCK STRUCTURE				WHETHER TO HANDLE LOOP			
	OUR	[63]	[53, 54]	[64]	OUR	[63]	[53, 54]	[64]	OUR	[63]	[53, 54]	[64]	OUR	[63]	[53, 54]	[64]	OUR	[63]	[53, 54]	[64]
1	✓	✓	✓	×	×	×	×	⊥	×	×	×	⊥	×	×	×	⊥	✓	✓	✓	⊥
2	✓	×	✓	✓	×	⊥	✓	×	×	⊥	✓	×	×	⊥	×	✓	⊘	⊘	⊘	⊘
3	✓	×	✓	×	×	⊥	✓	⊥	×	⊥	✓	⊥	×	⊥	×	⊥	✓	⊥	✓	⊥
4	✓	✓	✓	×	×	×	×	⊥	×	×	×	⊥	×	×	×	⊥	✓	✓	✓	⊥
5	✓	×	✓	×	×	⊥	✓	⊥	×	⊥	✓	⊥	×	⊥	×	⊥	✓	⊥	✓	⊥
6	✓	×	✓	×	×	⊥	✓	⊥	×	⊥	✓	⊥	×	⊥	×	⊥	✓	⊥	✓	⊥
7	✓	×	✓	×	×	⊥	✓	⊥	×	⊥	✓	⊥	×	⊥	×	⊥	✓	⊥	✓	⊥
8	✓	×	✓	×	×	⊥	✓	⊥	×	⊥	✓	⊥	×	⊥	×	⊥	✓	⊥	✓	⊥
9	✓	×	✓	✓	×	⊥	✓	×	×	⊥	✓	×	×	⊥	×	✓	⊘	⊘	⊘	⊘
10	✓	×	✓	×	×	⊥	✓	⊥	×	⊥	✓	⊥	×	⊥	×	⊥	✓	⊥	✓	⊥
11	✓	×	✓	✓	×	⊥	✓	×	×	⊥	✓	×	×	⊥	×	✓	⊘	⊘	⊘	⊘
12	✓	×	✓	×	×	⊥	✓	⊥	×	⊥	✓	⊥	×	⊥	×	⊥	✓	⊥	✓	⊥
13	✓	×	✓	×	×	⊥	✓	⊥	×	⊥	✓	⊥	×	⊥	×	⊥	✓	⊥	✓	⊥
14	✓	×	✓	×	×	⊥	✓	⊥	×	⊥	✓	⊥	×	⊥	×	⊥	✓	⊥	✓	⊥
15	✓	×	✓	×	×	⊥	✓	⊥	×	⊥	✓	⊥	×	⊥	×	⊥	✓	⊥	✓	⊥
16	✓	×	✓	✓	×	⊥	✓	×	×	⊥	✓	×	×	⊥	×	✓	⊘	⊘	⊘	⊘
17	✓	×	✓	✓	×	⊥	✓	×	×	⊥	✓	×	×	⊥	×	✓	⊘	⊘	⊘	⊘
18	✓	×	✓	✓	×	⊥	✓	×	×	⊥	✓	×	×	⊥	×	✓	⊘	⊘	⊘	⊘
19	✓	×	✓	✓	×	⊥	✓	×	×	⊥	✓	×	×	⊥	×	✓	⊘	⊘	⊘	⊘
20	✓	×	✓	✓	×	⊥	✓	×	×	⊥	✓	×	×	⊥	×	✓	⊘	⊘	⊘	⊘

"✓" and "×" represent "yes" and "no." "⊥" indicates that the case cannot be converted and there are no other values. "⊘"indicates that there is no loop structure in the case.

truncation conditions in the unfolding network, and ξ is the maximum number of transitions in the outflow and entry.

Compute the complexity of the active relationship *ExtractRelation*() to $O(|E|^3)$, where $|E|$ is the number of events in the expanded network. The complexity of *Refactor*() is $O(|loopRL|^4)$, where $|loopRL|$ is the number of transitions contained in a complex structure that needs to be generalized. Therefore, the time complexity of the process model index building algorithm is at worst $O(h(|A|\cdot R^{\xi}+|E|^3+|loopRL|^4))$, where h is the depth of the process tree.

Therefore, generally, the time complexity of the process model index building algorithm is determined by the complexity of *ExtractRelation*() and number of iterations (depth of process tree). Hence, typically, the time complexity of *GenerationOfTheProcessTree*() is $O(|T|^3 \cdot h)$.

Next, the performance of the index building algorithm is analyzed. Figs. 9, 10 and 11 show time curves for datasets AP, BP and CP. The horizontal axis is the number of simplified models and the vertical axis is the time spent. The models in dataset AP are BSPMs, and some of the models in dataset BP contain complex structures. The time-varying curve of dataset BP is more uneven. Therefore, in most cases, complex structures are more time-consuming and unstable.

7.4 Retrieval Efficiency Analysis

The time complexity of the calculation of the process tree similarity is determined by the complexity of the nodes to be deleted and the behavior to be modified.

The complexity of deleting behavior is $O(|leaves(t_1)|\times|leaves(t_2)|)$. The complexity of modifying behavior is $O(max(depth(t_1), depth(t_2))\times|Retain(t_1,t_2)|^2)$, where $leaves(t_1)$ represents all the leaf nodes of t_1, $depth(t_1)$ is the depth of process tree t_1.

Thus, the time complexity of the process tree similarity calculation is $O(|leaves(t_1)|\times|leaves(t_2)|+max(depth(t_1), depth(t_2))\times|Retain(t_1,t_2)|^2)$. The worst time complexity of the process tree similarity calculation is $O(max(depth(t_1), depth(t_2))\times|Retain(t_1, t_2)|^2)$, which occurs when $leaves(t_1)$ is identical to $leaves(t_2)$. The best time complexity is $O(|leaves(t_1)|\times|leaves(t_2)|)$, which occurs when the leaf nodes of two trees are totally different or have low similarity.

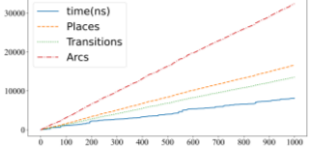

Fig.9. Indexing time statistics on AP.

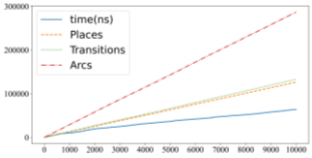

Fig.10. Indexing time statistics on BP.

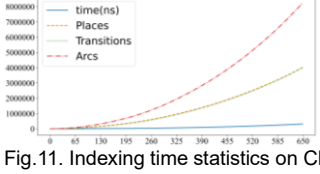

Fig.11. Indexing time statistics on CP

To test the performance of the retrieval algorithm on AP, six different query conditions (labeled AQ1–AQ6) were constructed. AQ1 contained 10 transitions, and each subsequent query condition had 10 more transitions than the previous one. The query conditions AQ1–AQ6 were combined with AP group models of different sizes (100–1,000) for the query. The reponse time statistics are shown in Fig.12, and the retrieval time is in milliseconds.

To test the performance of the retrieval algorithm on BP and CP, six different query conditions (labeled BQ1–BQ6 and CQ1-CQ6) were also constructed, respectively. The number of transitions in BQ1–BQ6 and CQ1-CQ6 were

TABLE 6
COMPARISON OF THE RETRIEVAL CAPABILITIES OF EXISTING METHODS

Paper	Index to describe	Querying technique	Types		Ranges				Index descriptions		
			Structure	Behavior	Overall structure	Overall behavior	Each behavior	Loop structure	Graphics	Matrix	Text
[65]	Tasks are next to the relationship tree	Graph edit distance for semantic workflow	✓	✓	✓				✓		✓
[66]	Complete trigger sequence	A* Search algorithm		✓	✓			✓	✓		✓
[67]	Behavioral feature	Filter- verification		✓		✓	✓	✓		✓	
[68]	Behavioral feature set	Direct retrieval of behavioral characteristics		✓		✓	✓	✓		✓	
[40]	ORTP Index	Time and probability constrained retrieval		✓		✓	✓			✓	
[69]	Process Graph	Graph matching	✓		✓			✓	✓		
[70]	Process Graph	Graph matching	✓		✓			✓	✓		
[71]	EPC	Graph matching	✓		✓			✓	✓		
Ours	**Process tree**	**Process tree Edit distance**	✓	✓	✓	✓	✓	✓	✓	✓	✓

the same as that in AQ1–AQ6; however, the transition labels were different. The transition labels in AQ1–AQ6 were characters, whereas those in BQ1–BQ6 and CQ1-CQ6 were phrases. The time curve statistics for dataset BP are shown in Fig.13, and for dataset CP are shown in Fig.14.

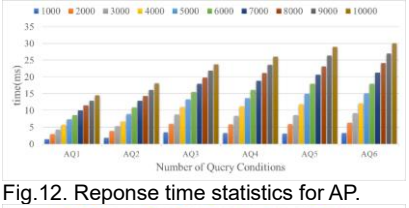

Fig.12. Reponse time statistics for AP.

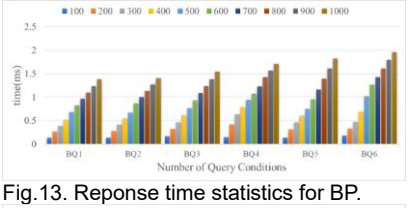

Fig.13. Reponse time statistics for BP.

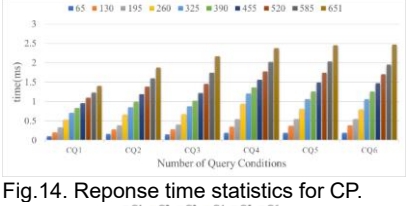

Fig.14. Reponse time statistics for CP.

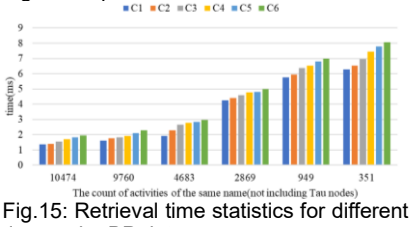

Fig.15: Retrieval time statistics for different levels of transition repetition on the BP datasets.

The reponse time of the method proposed by Polyvyanyy et al.[61] was in seconds on dataset BP. The algorithm was much faster than [61] in milliseconds on dataset BP. Similar, our approach was much faster on data [62].

The transition repetition rate among different models in dataset AP was high, whereas the transition repetition rate among different models in dataset BP and CP was low. Therefore, the retrieval time of dataset BP and CP was almost one-tenth of dataset AP.

There were 13,516 transitions in dataset BP, including duplicate name transitions and tau transitions[72], and 10,474 normal transitions. In order to process the synonyms, the [73] use Word2vec to standardize synonyms in the BP group. Comparing the similarity between two transitions using Word2vec, can get a number between 0 and 1. The closer to 1, the higher the similarity between two transitions. Using the same way to handle CP dataset, the result of reponse time on the three dataset are shown in Table 7. Our research findings indicate that our proposed method exhibits significantly improved response time than [61] on datasets AP, BP, and CP.

TABLE 7
QUANTITATIVE COMPARISON ON RESPONSE TIME

	AP		*BP*		*CP*	
	Our	[61]	Our	[61]	Our	[61]
Q1	14.4866	46.876	1.3773	3.537	1.4034	3.7028
Q2	18.087	66.396	1.4031	3.9151	1.8718	4.2487
Q3	23.815	76.104	1.5387	4.4812	2.1643	4.5554
Q4	26.0554	82.913	1.711	4.8221	2.3705	5.2504
Q5	28.9556	93.596	1.8243	5.5011	2.4413	5.901
Q6	29.9889	101.695	1.9584	5.923	2.456	6.4924

The focus of this study is the structure and behavior of the model, not including tag semantics. The similarity thresholds were set to 1, 0.9, 0.8, 0.7, 0.6, and 0.5. If the values of similarity higher than this threshold, the synonyms were replaced. When the similarity thresholds were set to 1, the synonyms were not replaced. The final six similarity thresholds corresponded to 10,474, 9760, 4683, 2869, 949, and 351 transitions without duplicate name transitions and tau nodes. The retrieving time statistics for different levels of transition repetition are shown in Fig.15. The results show that the higher the transition repetition, the longer it took to retrieve models.

7.5 Compared with Existing Methods

The method was compared with other business process retrieval algorithms, and the results are shown in Table 6.

In Table 6, the "Types" are divided into "Structure" and "Behavior," which refer to the search types of the search algorithms.

The "Ranges" is divided into four types: "Overall structure" refers to whether the overall structure of the model can be directly input to retrieve the model; "Overall

behavior" refers to whether the overall behavior of the model can be directly input for retrieval; "Each behavior" refers to whether the behavior between transitions can be directly input for retrieval; and "Loop structure" refers to whether the loop structure can be retrieved.

"Index descriptions" can be divided into three types, that is, "Graphics," "Matrix," and "Text," which refer to the index structure of the expression. "Graphics" refers to the index to be rendered in the form of graphics; "Matrix" refers the index to be rendered in the form of a matrix, and is not required in the form of the matrix storage model; and "Text" means that the index can be presented as text, and that the text can express the index information directly.

In terms of the retrieval scope, the methods [65-67] in Table 6 need to input a model or model fragment during retrieval instead of directly inputting behavior for retrieval.

The methods[40, 68] directly store transition behavior; hence, the overall behavior of the model can be retrieved, but they do not directly input the model to retrieve models with a similar structure. The methods [69-71] are graph retrieval approaches, and cannot be retrieved by behavior. The process tree not only contains the behaviors between transitions, but also the control flow pattern contained[74], in which the structural characteristics of the model are implied. Therefore, the retrieval method can combine behavior and structure simultaneously.

8 Conclusion

In this study, a behavior-based retrieval method for a business process model repository was proposed. First, the approach uses a process tree as the index of the business process model. Second, the similarity between different models is calculated by measuring the similarity between indices.

The behavior-based business data retrieval method proposed has certain promising applications and practical significance. However, there are still many areas to be improved in this research. For example, business database retrieval in the present study is only integrated from a behavioral perspective and is not sufficiently comprehensive. Only four types of relationships between transitions are considered. In follow-up research, the characteristics of business processes will be analyzed in a more comprehensive and detailed manner, and a more efficient retrieval method will be established.

References

[1] M. L. Rosa *et al.*, "APROMORE: An advanced process model repository," *Expert Systems with Applications,* vol. 38, no. 6, pp. 7029-7040, 2011.
[2] H. Leopold, J. Mendling, H. A. Reijers, and M. L. Rosa, "Simplifying process model abstraction: Techniques for generating model names," *Information Systems,* vol. 39, no. 1, pp. 134-151, 2014.
[3] M. Dumas, L. García-Bañuelos, M. L. Rosa, and R. Uba, "Fast detection of exact clones in business process model repositories," *Information Systems,* vol. 38, no. 4, pp. 619-633, 2013.
[4] F. Corradini, F. Fornari, A. Polini, B. Re, and F. Tiezzi, "A formal approach to modeling and verification of business process collaborations," *Science of Computer Programming,* vol. 166, pp. 35-70, 2018.
[5] B. A. Tama and M. Comuzzi, "An empirical comparison of classification techniques for next event prediction using business process event logs," *Expert Systems with Applications,* vol. 129, pp. 233-245, 2019.
[6] S. Song, Y. Gao, C. Wang, X. Zhu, J. Wang, and S. Y. Philip, "Matching heterogeneous events with patterns," *IEEE Transactions on Knowledge and Data Engineering,* vol. 29, no. 8, pp. 1695-1708, 2017.
[7] Y. Gao, S. Song, X. Zhu, J. Wang, X. Lian, and L. Zou, "Matching heterogeneous event data," *IEEE Transactions on Knowledge and Data Engineering,* vol. 30, no. 11, pp. 2157-2170, 2018.
[8] B. Aysolmaz, H. Leopold, H. A. Reijers, and O. Demirörs, "A semi-automated approach for generating natural language requirements documents based on business process models," *Information and Software Technology,* vol. 93, pp. 14-29, 2018.
[9] R. S. Veitch and L. F. Seymour, "Measuring Business Process Model Reuse in a Process Repository," in *International Conference on Business Process Management*, 2019: Springer, pp. 733-744.
[10] E. Oztemel and S. Gursev, "Literature review of Industry 4.0 and related technologies," *Journal of Intelligent Manufacturing,* vol. 31, no. 1, pp. 127-182, 2020.
[11] M. Javaid, A. Haleem, R. Vaishya, S. Bahl, R. Suman, and A. Vaish, "Industry 4.0 technologies and their applications in fighting COVID-19 pandemic," *Diabetes & Metabolic Syndrome: Clinical Research & Reviews,* vol. 14, no. 4, pp. 419-422, 2020/07/01/ 2020.
[12] L. S. Dalenogare, G. B. Benitez, N. F. Ayala, and A. G. Frank, "The expected contribution of Industry 4.0 technologies for industrial performance," *International Journal of Production Economics,* vol. 204, pp. 383-394, 2018/10/01/ 2018.
[13] M. Leemans, W. M. Van Der Aalst, M. G. Van Den Brand, R. R. Schiffelers, and L. Lensink, "Software Process Analysis Methodology–A Methodology Based on Lessons Learned in Embracing Legacy Software," presented at the 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2018.
[14] J. Wang, T. Jin, R. Wong, and L. Wen, "Querying business process model repositories: A survey of current approaches and issues," *World Wide Web-internet & Web Information Systems,* vol. 17, no. 3, pp. 427-454, 2014.
[15] T. Jin, J. Wang, M. L. Rosa, A. Ter Hofstede, and L. Wen, "Efficient querying of large process model repositories," *Computers in Industry,* vol. 64, no. 1, pp. 41-49, 2013.
[16] H. Sneed and C. Verhoef, "Re-implementing a legacy system," *Journal of Systems and Software,* vol. 155, pp. 162-184, 2019.
[17] W. Aalst, *Process Mining: Data Science in Action*. Springer Publishing Company, Incorporated, 2016.
[18] M. M. Queiroz, S. F. Wamba, M. C. Machado, and R. Telles, "Smart production systems drivers for business process management improvement," *Business Process Management Journal,* 2020.
[19] R. Perez-Castillo, M. Fernandez-Ropero, and M. Piattini, "Business process model refactoring applying IBUPROFEN. An industrial evaluation," *Journal of Systems and Software,* vol. 147, pp. 86-103, 2019.
[20] R. Lombardi, "Knowledge transfer and organizational performance and business process: past, present and future researches," *Business Process Management Journal,* 2019.
[21] H. Kir and N. Erdogan, "A knowledge-intensive adaptive business process management framework," *Information Systems,* vol. 95, p. 101639, 2021.
[22] J. Mendling *et al.*, "Blockchains for business process management-challenges and opportunities," *ACM Transactions on Management Information Systems (TMIS),* vol. 9, no. 1, pp. 1-16, 2018.
[23] A. Corallo, M. Lazoi, and M. Lezzi, "Cybersecurity in the context of industry 4.0: A structured classification of critical assets and business impacts," *Computers in industry,* vol. 114, p. 103165, 2020.
[24] W. Viriyasitavat and D. Hoonsopon, "Blockchain characteristics and consensus in modern business processes," *Journal of Industrial Information Integration,* vol. 13, pp. 32-39, 2019.
[25] M. Nardelli, S. Nastic, S. Dustdar, M. Villari, and R. Ranjan, "Osmotic flow: Osmotic computing+ iot workflow," *IEEE Cloud Computing,* vol. 4, no. 2, pp. 68-75, 2017.
[26] C. Benghi, "Automated verification for collaborative workflows in a Digital Plan of Work," *Automation in Construction,* vol. 107, p. 102926, 2019.
[27] G. Jošt, M. Heričko, and G. Polančič, "Theoretical foundations and implementation of business process diagrams' complexity management technique based on highlights," *Software & Systems Modeling,* vol. 18, no. 2, pp. 1079-1095, 2019.
[28] A. Polyvyanyy, C. Ouyang, A. Barros, and W. M. van der Aalst, "Process querying: Enabling business intelligence through query-based

process analytics," *Decision Support Systems,* vol. 100, pp. 41-56, 2017.
[29] M. Cho, M. Song, M. Comuzzi, and S. Yoo, "Evaluating the effect of best practices for business process redesign: An evidence-based approach based on process mining techniques," *Decision Support Systems,* vol. 104, pp. 92-103, 2017.
[30] M. AbdEllatif, M. S. Farhan, and N. S. Shehata, "Overcoming business process reengineering obstacles using ontology-based knowledge map methodology," *Future Computing and Informatics Journal,* vol. 3, no. 1, pp. 7-28, 2018.
[31] W. Song, H.-A. Jacobsen, S. Cheung, H. Liu, and X. Ma, "Workflow refactoring for maximizing concurrency and block-structuredness," *IEEE Transactions on Services Computing,* 2018.
[32] Y. Zhang, G. Cui, Z. Shu, and T. Jie, "IFOA4WSC: A quick and effective algorithm for QoS-aware service composition," *International Journal of Web & Grid Services,* vol. 12, no. 1, p. 81, 2016.
[33] M. Leyer, D. Iren, and B. Aysolmaz, "Identification and analysis of handovers in organisations using process model repositories," *Business Process Management Journal,* 2020.
[34] A. Beheshti, B. Benatallah, and H. R. Motahari‐Nezhad, "ProcessAtlas: a scalable and extensible platform for business process analytics," *Software: Practice and Experience,* vol. 48, no. 4, pp. 842-866, 2018.
[35] F. Corradini, F. Fornari, A. Polini, B. Re, and F. Tiezzi, "RePROSitory: a Repository Platform for Sharing Business PROcess modelS," *BPM (PhD/Demos),* vol. 2420, pp. 149-153, 2019.
[36] A. Polyvyanyy, A. Pika, and A. H. ter Hofstede, "Scenario-based process querying for compliance, reuse, and standardization," *Information Systems,* vol. 93, p. 101563, 2020.
[37] S. Yongchareon, C. Liu, and X. Zhao, "Reusing artifact-centric business process models: a behavioral consistent specialization approach," *Computing,* pp. 1-37, 2020.
[38] M. Camargo, M. Dumas, and O. González-Rojas, "Automated discovery of business process simulation models from event logs," *Decision Support Systems,* vol. 134, p. 113284, 2020.
[39] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, and G. Bruno, "Automated discovery of structured process models from event logs: The discover-and-structure approach," *Data & Knowledge Engineering,* vol. 117, pp. 373-392, 2018.
[40] H. Huang *et al.*, "Efficiently querying large process model repositories in smart city cloud workflow systems based on quantitative ordering relations," *Information Sciences,* vol. 495, pp. 100-115, 2019.
[41] T. Jin, J. Wang, N. Wu, M. La Rosa, and A. H. Ter Hofstede, "Efficient and accurate retrieval of business process models through indexing," presented at the OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", 2010.
[42] A. H. Ter Hofstede, C. Ouyang, M. La Rosa, L. Song, J. Wang, and A. Polyvyanyy, "APQL: A process-model query language," presented at the Asia-Pacific Conference on Business Process Management, 2013.
[43] B. Mahleko and A. Wombacher, "Indexing business processes based on annotated finite state automata," presented at the 2006 IEEE International Conference on Web Services (ICWS'06), 2006.
[44] H. Leopold, H. van der Aa, F. Pittke, M. Raffel, J. Mendling, and H. A. Reijers, "Searching textual and model-based process descriptions based on a unified data format," *Software & Systems Modeling,* vol. 18, no. 2, pp. 1179-1194, 2019.
[45] M. Ponti, L. Ribeiro, T. Nazare, T. Bui, and J. Collomosse, *Everything You Wanted to Know about Deep Learning for Computer Vision but Were Afraid to Ask*. 2017, pp. 17-41.
[46] T. Jin, J. Wang, and L. Wen, "Efficient Retrieval of Similar Business Process Models Based on Structure," vol. 7044, pp. 56-63, 2011.
[47] H. Zha, J. Wang, L. Wen, C. Wang, and J. Sun, "A workflow net similarity measure based on transition adjacency relations," *Computers in Industry,* vol. 61, no. 5, pp. 463-471, 2010.
[48] M. T. Gómez-López, A. M. R. Quintero, L. Parody, J. M. P. Álvarez, and M. Reichert, "An architecture for querying business process, business process instances, and business data models," in *International Conference on Business Process Management*, 2017: Springer, pp. 757-769.
[49] D. Brdjanin, S. Ilic, G. Banjac, D. Banjac, and S. Maric, "Automatic derivation of conceptual database models from differently serialized business process models," *Software and Systems Modeling,* pp. 1-27, 2020.
[50] Y. Huang, W. Li, Z. Liang, Y. Xue, and X. Wang, "Efficient business process consolidation: combining topic features with structure matching," *Soft Computing,* vol. 22, no. 2, pp. 645-657, 2018.
[51] W. M. P. V. D. Aalst, "Process discovery from event data: Relating models and logs through abstractions," *Wiley Interdisciplinary Reviews Data Mining & Knowledge Discovery,* vol. 8, no. 3, p. e1244, 2018.
[52] R. Zhu, T. Li, Q. Mo, Z.-L. He, Q. Yu, and Y.-Q. Wang, "Data-Driven Bilayer Software Process Mining," *Ruan Jian Xue Bao/Journal of Software,* vol. 28, pp. 3455-3483, 2018.
doi: 10.13328/j.cnki.jos.005304.
[53] S. J. J. Leemans, D. Fahland, and W. M. P. V. D. Aalst, "Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour," presented at the Business Process Management Workshops, 2013.
[54] S. J. J. Leemans, D. Fahland, and W. M. P. V. D. Aalst, "Discovering Block-Structured Process Models from Event Logs - A Constructive Approach," presented at the Petri Nets, 2013.
[55] P. Darondeau, "Equality of languages coincides with isomorphism of reachable state graphs for bounded and persistent Petri nets," *Inf. Process. Lett.,* vol. 94, no. 6, pp. 241-245, / 2005, doi: 10.1016/j.ipl.2005.03.002.
[56] J. Wang, Y. Du, and S. Yu, "Coloured Logic Petri Nets and analysis of their reachable trees," *Enterp. Inf. Syst.,* vol. 9, no. 8, pp. 900-919, / 2015, doi: 10.1080/17517575.2013.879924.
[57] K. L. McMillan, "A Technique of State Space Search Based on Unfolding," *Formal Methods Syst. Des.,* vol. 6, no. 1, pp. 45-65, / 1995, doi: 10.1007/BF01384314.
[58] J. Esparza, S. Römer, and W. Vogler, "An Improvement of McMillan's Unfolding Algorithm," 1996.
[59] S. Ontañón, "An overview of distance and similarity functions for structured data," *Artificial Intelligence Review,* vol. 53, no. 7, pp. 5309-5351, 2020.
[60] A. Burattin and A. Sperduti, "PLG: A Framework for the Generation of Business Process Models and Their Execution Logs," presented at the Business Process Management, 2010.
[61] A. Polyvyanyy, A. Pika, and A. H. M. T. Hofstede, "Scenario-based process querying for compliance, reuse, and standardization," *Information Systems,* vol. 93, p. 101563, 2020.
[62] Schäfer, Bernhard, Han van der Aa, Henrik Leopold, and Heiner Stuckenschmidt. "Sketch2BPMN: Automatic recognition of hand-drawn BPMN models." In International Conference on Advanced Information Systems Engineering, pp. 344-360. Cham: Springer International Publishing, 2021.
[63] M. A. B. Ahmadon and S. Yamaguchi, "Convertibility and Conversion Algorithm of Well-Structured Workflow Net to Process Tree," presented at the Proceedings of the 2013 First International Symposium on Computing and Networking, 2013.
[64] A. Polyvyanyy, L. Garc¨ªa-Ba£¿uelos, D. Fahland, and M. Weske, "Maximal Structuring of Acyclic Process Models," *Computer Science,* 2011.
[65] S. Jinyong, G. Tianlong, W. Lijie, Q. Junyan, and M. Yu, "Retrieval of Similar Semantic Workflows Based on Behavioral and Structural Characteristics," *Journal of Computer Research and Development,* vol. 54(9), pp. 1880-1891, 2017, doi: 10. 7544/ issn l000-1239. 2017. 20160755.
[66] D. ZH, W. LJ, H. HW, and W. JM, "Behavioral Similarity Algorithm for Process Models Based on Firing Sequence Collection," *Journal of Software,* vol. 26, no. 3, pp. 449-459, 2015.
[67] M. Kunze, M. Weidlich, and M. Weske, "Querying process models by behavior inclusion," *Software & Systems Modeling,* vol. 14, no. 3, 2015.
[68] J. Tao, J. Wang, and L. Wen, "Querying Business Process Models Based on Semantics," presented at the Database Systems for Advanced Applications, 2011.
[69] Z. Yan, R. M. Dijkman, and P. W. P. J. Grefen, "FNet: An Index for Advanced Business Process Querying," presented at the BPM, 2012.
[70] P. Delfmann, D. Breuker, M. Matzner, and J. Becker, "Supporting Information Systems Analysis Through Conceptual Model Query – The Diagramed Model Query Language (DMQL)," *Communications of the Association for Information Systems,* vol. Vol. 37, no. Article 24, pp. 473-509, 2015.
[71] P. Delfmann, M. Steinhorst, H. A. Dietrich, and J. Becker, "The generic model query language GMQL – Conceptual specification, implementation, and runtime evaluation," *Information Systems,* vol. 47, no. C, pp. 129-177, 2015.
[72] L. Wen, J. Wang, W. Aalst, B. Huang, and J. Sun, "Mining process models with prime invisible tasks," *Data & Knowledge Engineering,* vol. 69, no. 10, pp. 999-1021, 2010.
[73] K. W. Church, "Word2Vec," *Natural Language Engineering,* vol. 23, no. 1, pp. 155-162, 2016, doi: 10.1017/S1351324916000334.
[74] W. van der Aalst, J. Buijs, and B. van Dongen, "Towards Improving the Representational Bias of Process Mining," presented at the SIMPDA, 2011.

Rui Zhu (Member, IEEE) received the B.S. degree in software engineering and the M.S. and Ph.D degrees in system analysis and integration from Yunnan University, Kunming, China in 2006, 2010 and 2016, respectively. He is currently a Head and Associate Professor of Artificial Intelligence with the School of Software, Yunnan University, Yunnan, China. He was a visiting scholar with Peking University from 2019 to 2020. He is also the Core Scientist of the Yunnan Key Laboratory of Software Engineering and the Yunnan Software Engineering Academic Team. In 2022, he received the Yunnan Province Xing Dian Talents Support Plan Young Scholar Title. His research interests include business process management, process mining, service computing and deep learning. He has published around 70 research papers in national and international conferences and journals.

Yue Huang received the BSc degree in software engineering from the Wuhan Polytechnic University, Wuhan, China, in 2018, and the MSc degree in software engineering from the Yunnan University, Kunming, China, in 2021. She is currently working toward the PhD degree with the software and data engineering research center, Shandong University, Jinan, China. Her main research interests include federated learning, models bussiness processes analysis and service computing.

Ling Liu, Ph.D., is a Professor in the College of Computing at Georgia Institute of Technology and an elected IEEE Fellow. She directs the research programs in Distributed Data Intensive Systems Lab (DiSL), examining performance, availability, security, privacy, trust and data management issues in big data systems, cloud computing and distributed computing systems. Liu and the DiSL research group have been working on various aspects of distributed data intensive systems, ranging from Big Data systems and data analytics, Cloud Computing and cloud datacenters, distributed systems, decentralized and social computing, mobile and location based services, sensor network and event stream processing, to service oriented computing and architectures.

Wei Zhou received the Ph.D. degree from the University of Chinese Academy of Sciences. He is currently a Full Professor with the Software School, Yunnan University. His current research interests include the distributed data intensive computing and network security. He host a number of National Natural Science Foundation projects.

Xuan Zhang received the B.S. and M.S. degrees in computer science and the Ph.D. degree in system analysis and integration from Yunnan University, Kunming, China. She is currently a Professor with the School of Software, Yunnan University, Yunnan, China. She is also the Core Scientist of the Yunnan Key Laboratory of Software Engineering and the Yunnan Software Engineering Academic Team. She has been a principal investigator for more than 30 national, provincial, and private grants and contracts. She is the author of three books and more than 100 articles. Her research interests include knowledge graph(KG), natural language processing (NLP), business process management and service computing.

Yeting Chen(Member, IEEE) received a Master's degree in Management Science and Engineering from Yunnan University in Kunming, China in 2013, and a Ph.D. in Economics from Central University of Finance and Economics in Beijing, China in 2019. The current Director and Associate Professor of the Department of Digital Economy at the School of Economics and Management, Yunnan Normal University, Yunnan Province, China. In 2020, she was awarded the title of Young Scholar in the Yunnan Xingdian Talent Support Program. Her research interests include business process management and service computing. She has published about 30 research papers.

Li Cai received the MS degree in computer application from Yunnan University, China, in 2007, and the PhD degree in computer software and theory from Fudan University, China, in 2020. From 1997 to 2002, she was a research assistant with Network Center. Since 2010, she has been an associate professor with the School of Software, Yunnan University, China. Her researchinterests include machine learning, business process management, service computing and data quality.