Efficient and Exact Query of Large Process Model Repositories in Cloud Workflow Systems

Hua Huang, Rong Peng, and Zaiwen Feng

Abstract—As cloud computing platforms are widely accepted by more and more enterprises and individuals, the underlying cloud workflow systems accumulate large numbers of business process models. Retrieving and recommending the most similar process models according to the tenant's requirements become extremely important, for it is not only beneficial to promote the reuse of the existing model assets, but also helpful to reduce the error rate of the modeling process. Since the scales of cloud workflow repositories become bigger and bigger, developing efficient and exact query approaches is urgent. To this end, an improved two-stage exact query approach based on graph structure is proposed. In the filtering stage, the *composite task index*, which consists of the label, join-attribute and split-attribute of a task, is adopted to acquire candidate models, which can greatly reduce the number of process models needed to be tested by a time-consuming verification algorithm. In the verification stage, a novel subgraph isomorphism test based on *task code* is proposed to refine the candidate model set. Experiments are conducted on six synthetic model sets and two real model sets. The results demonstrate that the presented approach can significantly improve the query efficiency and reduce the query response time.

Index Terms—Cloud workflow, business process management, exact process query, composite index, subgraph isomorphism

1 Introduction

Cloud workflow systems are a class of workflow management systems that are deployed in various cloud computing environments. Cloud workflow systems can be widely used as platform software (or middleware services) to facilitate the usage of cloud services. Every tenant can design, configure and run his/her business processes on it [1], [2]. With the increase of tenants' applications, the underlying cloud workflow systems accumulate a large number of process descriptions (i.e., business process models) [3]. For example, there are more than 2,000 enterprise users (namely tenants) and 100,000 business processes (roughly 50 business processes per tenant) in the cloud workflow system of Ceramic Cloud Service Platform (CCSP for short, http://www.pasp.cn). So how to efficiently and exactly query large process model repositories in a cloud workflow system is challenging.

Efficient retrieval of similar process models or process fragments is helpful for users (e.g., enterprise tenants) to select, customize and establish their new processes from existing process model repositories [4]. For instance, in a cloud workflow system, when a tenant wants to create a new process model, building it from scratch is time-consuming and error-prone [24]; instead, building it by customizing a suitable existing process model is time-saving and helpful to promote the reuse and preserve the best practice. Thus, an efficient process retrieval algorithm is ultimate important. Unfortunately, as the scale of the process model repository in a cloud workflow system may be huge and will keep on growing with the increase of tenants, it is not easy to achieve efficient and exact retrieval for a given process fragment.

As described in [25], most business process models can be modeled as graph with behavior and operation semantics. Based on this assumption, Wang et al. [25] group process model queries into five categories: *i*) exact query based on graph structure; *ii*) similarity query based on graph structure; *iii*) exact query based on behavior semantics; *iv*) similarity query based on behavior semantics; *v*) query based on operation semantics. In this paper, we focus on improving the efficiency of exact query based on graph structure in cloud workflow systems.

In order to improve the process query efficiency, the filtering-verification framework is often adopted to reduce the number of process models needed to be checked by subgraph isomorphism algorithms. Generally, in the filtering stage, various indexes are built to locate candidate process models that satisfy the given retrieval conditions; in the verification stage, only the candidate models need to be checked by time-consuming subgraph isomorphism tests, such as Ullman's subgraph isomorphism algorithm [14], [16], [26].

In a cloud workflow based cloud platform, customized services are provided based on customized workflows. The customized services always have the same or similar business backgrounds and application contexts, which indicates that their underlying process models have many tasks with the same or similar labels and common subprocesses (i.e., subgraphs). As a result, it lessens the filtering capability of many exact querying approaches [9], [10], [11], [12], [13], [14], [25], such as the single task index in [14]. As cloud

- *H. Huang is with the State Key Laboratory of Software Engineering, School of Computer, Wuhan University, Wuhan 430072, and the School of Information Engineering, Jingdezhen Ceramic Institute, Jingdezhen 333001. E-mail: jdz_qq@qq.com.*
- *R. Peng and Z. Feng are with the State Key Laboratory of Software Engineering, School of Computer, Wuhan University, Wuhan 430072. E-mail: {rongpeng, zwfeng}@whu.edu.cn.*

Manuscript received 7 Feb. 2015; revised 30 Aug. 2015; accepted 16 Sept. 2015. Date of publication 23 Sept. 2015; date of current version 5 Oct. 2018. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TSC.2015.2481409

platforms are widely accepted by more and more enterprises and individuals, the scale of the process model repositories will become larger than before. Namely, the quick exact query will be increasingly challenged. To solve this problem, improvements from the following two aspects should be considered: *i*) enhance the filtering capability of the index in the filtering stage, e.g., constructing a powerful composite index; *ii*) accelerate the exact query algorithm in the verification stage. Therefore, we put forward an efficient and exact query approach based on graph structure through the *composite task index* and a novel subgraph isomorphism test based on *task code*. The former is helpful to promote the filtering capability, and the latter contributes to accelerate the verification process. In summary, the proposed approach has the following contributions:

- Construct a *composite task index* by combining the label, join-attribute and split-attribute of a task to improve the filtering capability.
- Propose a novel subgraph isomorphism algorithm based on *task code*, which utilizes both the neighborhood information and the structural features of a task node.
- Conduct extensive experiments over synthetic and real datasets to demonstrate the effectiveness and efficiency of the proposed approach.

The rest of this paper is as follows. Section 2 discusses related work. Section 3 introduces the definitions used in this paper, while Section 4 describes the subgraph isomorphism test based on *task code*, and Section 5 shows how to query the large process model repository in a cloud workflow system through the *composite task index* and the subgraph isomorphism test based on *task code*. Section 6 presents our experimental evaluation. Section 7 draws out the conclusion.

2 Related Work

Currently, most process query approaches [5], [6], [7], [8], [9], [10] mainly adopt process similarity calculation to find the similar processes or process fragments. Many of them are devoted to how to evaluate the similarity between processes. Thus, the algorithms adopted in these approaches always retrieve similar processes by comparing the process model to be queried (denoted as the query model) with each process model in the repository. Namely, these methods focus on improving the precision and the recall of process retrieval, rather than on how to improve the retrieval efficiency. For instance, [5] proposes an approach to improve process model matching quality through user feedbacks; [6] proposes an automated approach to query a process model repository based on structural similarity and semantic similarity; [10] proposes an approach based on graph distance to measure the similarity between processes with variable time constraints for exact process retrieval. In a word, these approaches do not focus on improving the efficiency of process retrieval.

To improve the query efficiency, some researchers propose several process query approaches based on indexes. This work is inspired by the filtering and verification approach used in graph indexing algorithms, in which indexes are used to reduce the number of graphs needed to be checked by subgraph isomorphism test. In fact, different graph indexing algorithms use different graph features as indexes. For instance, GraphGrep [27] constructs an index on variable-length paths, upon which the database fingerprint and query pattern are built. GIndex [28], [29] creates an index on frequent and discriminative subgraphs. TreePi [30] constructs an index on discriminative frequent subtrees. Tree $+\Delta$ [31] builds an index on frequent subtrees and on-demand discriminative subgraphs. FG-Index [32] indexes on frequent subgraphs. FG*-index [33] is the upgraded version of FG-Index with a feature-index and a FAQ-index. Although the above graph indexing algorithms are capable of improving the efficiency of graph matching, none of them concentrate on how to utilize the features of process models, such as the labels and behavior attributes of nodes.

In the domain of process retrieval, several approaches are proposed to utilize special indexes to facilitate business process model query. For instance, [35] proposes a language based on BPMN (BPMN-Q), which exploits the robust indexing infrastructure available in the RDBMS to speed up the process query. [15] proposes path based indexes to speed up exact queries on business process model repositories. [17] proposes a business process query algorithm based on the index FNet, which performs two orders of magnitude faster than querying techniques built on traditional RDBMS. Moreover, [20] proposes a technique for indexing process models that relies on their alternative representations (called untangling). Experiments show that it can achieve accurate and efficient retrieval of process models based on behavior semantics. In contrast, our work focuses on efficient and accurate retrieval of large process model repositories based on graph structure.

In this respect, [14] proposes a two-stage approach using the filtering-verification framework for query evaluation. Since the approach is a typical exact query of large process model repositories based on graph structure, which can improve the query efficiency, it is selected as a benchmark approach. In this approach, a candidate model set is first obtained by using the *task index* in the filtering stage; in the verification stage, the candidate set is refined by adopting Ullman's subgraph isomorphism algorithm. As the index of the filtering stage is a *single task index* (only considering the labels of tasks), the filtering capability is still not strong enough if there exist numerous tasks with the same or similar labels in process models, especially in cloud workflow systems where large number of similar tenants' process models exist. Therefore, improving the exact query efficiency of process models from large process model repositories in cloud workflow systems is the focus of this paper.

3 Preliminaries

In order to improve the filtering capability of the *task index*, we construct a *composite task index* by composing the join-attribute and split-attribute of each task in process models represented as CNets (Definition in Section 3.1). Through the *composite task index*, more process models that do not satisfy the exact query can be filtered; and then, the candidate model set will be smaller and less time will be needed for the verification stage. Moreover, the *task code* is designed to improve the pruning capability of the subgraph isomorphism test used in the verification stage. Therefore, our proposal can greatly improve the efficiency of the exact

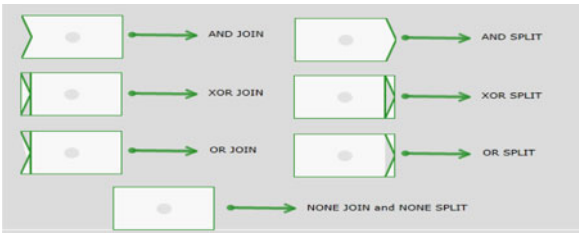


Fig. 1. Symbols of join-attributes and split-attributes of tasks.

retrieval based on graph structure. For better understanding, some related preliminaries are introduced below.

3.1 Model Definition

For the sake of dealing with different formats in a uniform way, a cloud workflow model (CNet) extended from YAWL [14], [36] is defined to model business processes. In CNet, nodes denote the tasks for executing activities, and edges denote the flow relations between tasks. The formal definitions of tasks and CNets are presented as follows.

Definition 1 (Task). *A task* t *is a 5-tuple,* t = (id,label,type, join,split)*, where:*

- id *specifies the unique identifier of a task.*
- label ∈ L *specifies a specific label of a task, where* L *is the set of labels.*
- type ∈ *{atomic task, compound task} specifies the type of a task. A task is compound means that the task can be refined to a process model.*
- join ∈ *{NONE, AND, OR, XOR} is the join-attribute of a task which specifies the join behavior of a task. Join-attribute "NONE" means that the task has no join behavior.*
- split ∈ *{NONE,AND,OR,XOR} is the split-attribute of a task which specifies the split behavior of a task. Split-attribute "NONE" means that the task has no split behavior.*

The symbols used to represent join-attributes and split-attributes of tasks are shown in Fig. 1.

Definition 2 (CNet). *A CNet is a 7-tuple, CNet =* (V,E,D,D_w, D_r,R,T_r)*,in which:*

- V = *{*Start*}* ∪ *{*End*}* ∪ T *is a set of nodes, where* T *is a set of tasks except the* Start *node and the* End *node.*
- E = ⟨Start × T⟩ ∪ ⟨T × T⟩ ∪ ⟨T × End⟩ *is a set of flow relations (namely edges), which describes the execution orders between tasks.*
- D *is a set of data elements, and each data element* d ∈ D *is written or read by tasks.*
- D_w: D→2^T *describes what data element is written by which tasks.*
- D_r: D→2^T *describes what data element is read by which tasks.*
- R *is a set of resources used in the CNet, and each resource* r ∈ R *could be a person or a role or a group of people.*
- T_r: T→2^R *is an assignment function, which describes what task is assigned to which resources.*

Fig. 2 exemplifies some business process models represented as CNets. The models CN1 to CN4 and the query model *q* can be created with the visual modeling tool, which can be accessed by http://cbpm.pasp.cn. In Fig. 2, the node with red rectangle means that the task is compound and other nodes with green rectangles are atomic tasks. In the notation, only the control-flow perspective of a process model is shown, and the data and resource perspectives are not graphically represented.

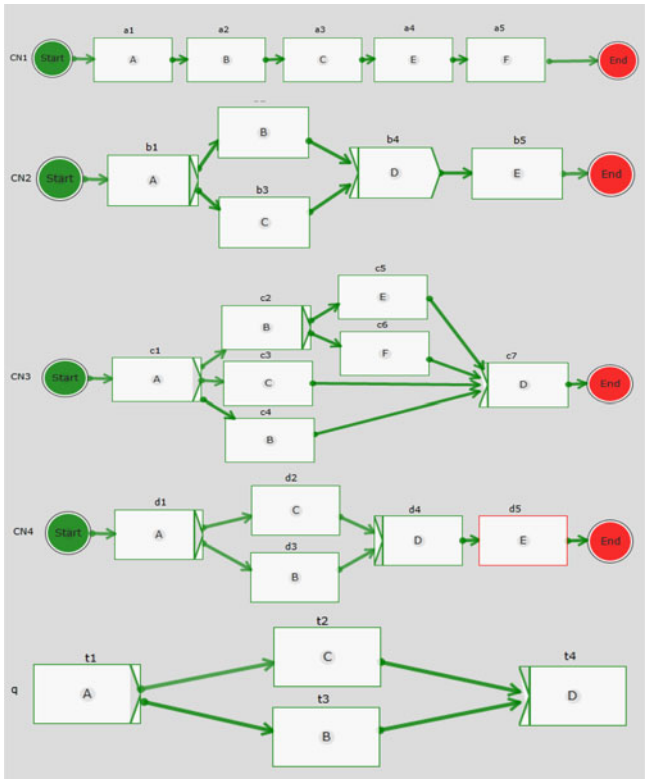


Fig. 2. Examples of process models represented as CNets.

In this paper, a query model is a CNet or a CNet fragment, and the result is defined as all CNets that cover the query model. For better understanding the concept of the "CNet cover" and "CNet query", the definition of subgraph isomorphism is first introduced.

Definition 3 (Subgraph Isomorphism). *Given two graphs G_a and G, Graph G_a is subgraph isomorphic to G, if and only if* ∃ *a subgraph* G_b ⊆ G*, there is a 1-1 correspondence between the distinct nodes of G_a and G_b that preserves adjacency.*

Definition 4 (CNet cover). *Given two CNets CN1 and CN2,* $CN1 = (V_1, E_1, D_1, Dw_1, D_{r1}, R_1, T_{r1})$ *is covered by* $CN2 = (V_2, E_2, D_2, D_{w2}, D_{r2}, R_2, T_{r2})$*, denoted as* $CN1 \subseteq CN2$*, which means that CN1 is isomorphic to a subgraph of CN2, iff there exists a function* f : $V_1 \to V_2 \cup E_1 \to E_2$*, such that:*

i) $\forall t \in V_1/\{Start, End\} : \exists f(t) \in V_2/\{Start, End\}$ *and* $t.label = f(t).label \Lambda t.type = f(t).type \Lambda t.join = f(t).join \Lambda t.split() = f(t).split$*, i.e., function* f *preserves all the features of tasks.*

ii) $\forall (v_1, v_2) \in E_1 : \exists (f(v_1), f(v_2)) \in E_2$*, i.e., function* f *preserves the flow relations.*

Definition 5 (CNet query). *Let* R *be a CNet model repository and* q *be a query model. The result of querying* q *in* R *is* $R_q = \{r \in R | q \subseteq r\}$.

TABLE 1
A Sample of the Task Index in Fig. 2

task.label	*model list*
"A"	CN1,CN2,CN3,CN4
"B"	CN1,CN2,CN3,CN4
"C"	CN1,CN2,CN3,CN4
"D"	CN2,CN3,CN4
"E"	CN1, CN2,CN3,CN4
"F"	CN1, CN3

Example 1. Assume that a process model repository R consists of four models in Fig. 2 and a query model q is also shown in Fig. 2. If we query q in R, then according to Definition 5, only model CN3 covers q, i.e., $R_q = \{\mathrm{CN3}\}$.

3.2 Index Construction

To improve the query efficiency, the Jin et al. in [14] apply an inverted index (namely *task index*) to speed up the query process. The *task index* stores the mapping from process tasks to process models by taking the form of set of pairs (*task*, *model list*) where *task* = *task.label* denotes a task with its label and *model list* denotes the set of models with the same task labels. The detail of the *task index* can be found in [14]. Given the model repository R shown in Fig. 2, its *task indexes* are shown in Table 1.

Table 1 shows that the *task index* is a single index, which only considers the label of a task. If there are a large number of tasks with the same labels in a process model repository, the filtering capability of the *task index* will be weakened. Especially in cloud workflow systems, the vendors provide process customization services, which increase the possibilities of different tenants' process models having the same task labels. Therefore, if we only consider the task label in the *task index*, the filtering capability is not enough. For example, given the model repository $R = \{\mathrm{CN1}, \mathrm{CN2}, \mathrm{CN3}, \mathrm{CN4}\}$ and the query model q as shown in Fig. 2, using the *task index* shown in Table 1 to obtain the candidate models that dominate the model query q, the candidate model set will be {CN2,CN3,CN4}, in which only CN1 is filtered. To enhance the filtering capability of the *task index*, inspired by the composite index of database technology, a *composite task index* is proposed.

According to Definition 4, a task is identical to another one means that they have the same labels, types, join-attributes and split-attributes (the data and resource perspective of a task is not considered in this paper). Generally, in the cloud workflow models, most tasks are atomic and only a few of them are compound [38]. Therefore, for the sake of reducing the cost of constructing the composite index, the type of a task is not included in the proposed *composite task index*. Thus, we only apply the join-attribute and split-attribute of a task with its label to construct a *composite task index* to improve the filtering capability of the *single task index*. The mapping from tasks to process models is stored in the *composite task index*es, which take the form of set of pairs (*composite-task*, *model list*), where *composite-task* = *task.label#task.join#task.split* denotes a specific task with the specific label, join-attribute and split-attribute, and *model list* denotes the set of models where the specific task occurs.

TABLE 2
A Sample of the Composite Task Index in Fig. 2

task.label	*task.join*	*task.split*	*model list*
"A"	NONE	NONE	CN1
"A"	NONE	XOR	CN2,CN4
"A"	NONE	OR	CN3
"B"	NONE	NONE	CN1,CN2,CN3,CN4
"B"	NONE	XOR	CN3
"C"	NONE	NONE	CN1,CN2,CN3,CN4
"D"	OR	NONE	CN3
"D"	OR	AND	CN2
"D"	XOR	NONE	CN4
"E"	NONE	NONE	CN1,CN2,CN3,CN4
"F"	NONE	NONE	CN1,CN3

For a given model, all its tasks are first extracted. Then the mappings between the tasks and the models are set up in the *composite task index* considering the label, join-attribute and split-attribute. Given the model repository R in Fig. 2, its *composite task indexes* are shown in Table 2.

Using the *composite task indexes* in Table 2 to filter the process model repository R, the candidate model set is {CN3}. Compared with the *single task indexes*, the filtering capability of the *composite task index* is obviously more powerful.

Given the process model repository R, let m be the number of models and n be the average number of tasks in process models. The time complexity of constructing the *composite task index* in R is O(mn), which is the same as the time complexity of constructing the *single task index*. However, building the *composite task index* is still time-consuming. Therefore, it is often done off-line in advance.

4 Subgraph Isomorphism Test Based on Task Code

According to Definition 4, a process model q_1 covers another process model q_2 means that q_1 contains all the tasks and flow relations of q_2, i.e., q_2 is isomorphic to a subgraph of q_1. To reduce the time cost of complex subgraph isomorphism algorithms in the verification stage, we adopt the neighborhood information and structural features of tasks to construct the data structure *task code*, to improve the pruning capability of the subgraph isomorphic algorithm. The definition of the *task code* and related concepts are presented first.

4.1 Definition of Task Code and Related Concepts

Definition 6 (Task code). *Given a task* t *in a CNet model, its* task code *is defined as* TC(t) = *[*L(t),Pre(t),Post(t)*], in which:*

i) L(t) *is the label of* t*;*

ii) $Pre(t) = (join, prenum, preset)$ *denotes the join-information of* t*, where* join *is the join-attribute of* t, prenum *is the number of the direct precursor tasks (namely join-tasks) of* t*, and* $preset = \{[l_i, (ind_{i1}, ind_{i2}, \ldots, ind_{im}), (outd_{i1}, outd_{i2}, \ldots, outd_{im})]^+\}$ *is the in-degree and out-degree information of* t*'s join-tasks in terms of labels, where* $\mathrm{l_i}$ *is a task label,* $(ind_{i1}, ind_{i2}, \ldots, ind_{im})$ *and* $(outd_{i1}, outd_{i2}, \ldots, outd_{im})$ *are the in-degree and out-degree list of* t*'s join-tasks with the label* $\mathrm{l_i}$*, such that* $ind_{i1} \geq ind_{i2} \geq \cdots \geq ind_{im}$ *and* $outd_{i1} \geq outd_{i2} \geq \cdots \geq outd_{im}$*;*

iii) Post(t) = (split,postnum,postset) *denotes the split-information of a task* t, *where* split *is the split-attribute of* t, postnum *is the number of the direct posterior tasks (namely split-tasks) of* t, *and* $postset = \{[l_i, (ind_{i1}, ind_{i2}, \ldots, ind_{in}), (outd_{i1}, outd_{i2}, \ldots, outd_{in})]^+\}$ *is the in-degree and out-degree information of* t*'s split-tasks in terms of labels, where* l_i *is a task label,* $(ind_{i1}, ind_{i2}, \ldots, ind_{in})$ *and* $(outd_{i1}, outd_{i2}, \ldots, outd_{in})$ *are the in-degree and out-degree list of* t*'s split-tasks with the label* l_i, *such that* $ind_{i1} \geq ind_{i2} \geq \cdots \geq ind_{in}$ *and* $outd_{i1} \geq outd_{i2} \geq \cdots \geq outd_{in}$.

Example 2. The *task codes* of the tasks c_1,c_2 and c_7 in CN3 in Fig. 2 are as follows:

- *TC(c_1)* = ["A",("NONE",1,{["*Start*",(0),(1)]}),("OR",3, {["B",(1,1),(2,1)],["C",(1),(1)]})];
- *TC(c_2)* = ["B",("NONE",1,{["A",(1),(3)]}),("XOR",2, {["E",(1),(1)], ["F",(1),(1)]})];
- *TC(c_7)* = ["D",("OR",4,{["B",(1),(1)], ["C",(1),(1)], ["E", (1),(1)],["F",(1),(1)]}),("NONE",1,{["*End*",(1),(0)]})].

To find candidate tasks according to a specific task (i.e., *candidate-task* shown in Definition 8) in the query model, the concept of presequence should be defined first.

Definition 7 (Presequence). *Given two sequences of integers in descending order:* U = (u_1,u_2,...,u_m) *and* O = (o_1,o_2,...,o_n), U *is called a presequence of* O, *iff:*

i) $m \leq n$;
ii) $\forall u_i \in U, |\{o_j | o_j \geq u_i \Lambda o_j \in O\}| \geq i$, *where* $1 \leq i \leq m$ *and* $1 \leq j \leq n$. *Example 3 illustrates the presequence relationship.*

Example 3. Given three sequences $U_1 = \{7, 5, 4, 2\}$, $U_2 = \{9, 8, 7, 4, 2\}$ and $U_3 = \{11, 9, 6, 5, 4, 3\}$, for each element *u* in U_1, there are enough elements in U_3 that are no smaller than u. Thus, U_1 is a presequence of U_3. But U_2 is not a presequence of U_3 because that there are only two elements in U_3 that are no smaller than the third largest element "7" in U_2.

Intuitively, if mapping tasks of a specific task *t* in the query model *q* can be found in any candidate model *c*, the join-information and split-information of *t* should be preserved by all mapping tasks. In order to find the candidate tasks that may match a task in the query model *q*, a definition of *candidate-task* based on the presequence is given below.

Definition 8 (Candidate-Task). *Assume that there are two tasks* t_1 *and* t_2 *in a query model* q *and a candidate model* c *respectively,* t_2 *is a* candidate-task *of* t_1 *iff:*

i) $L(t_1) = L(t_2)$;
ii) $Pre(t_1).join = Pre(t_2).join$ *and* $Post(t_1).split = Post(t_2).split$;
iii) $Pre(t_1).prenum \leq Pre(t_2).prenum$ *and* $Post(t_1).postnum \leq Post(t_2).postnum$;
iv) $\forall l_i \in Pre(t_1).preset, \exists l_j \in Pre(t_2).preset$, *such that* $l_i = l_j$ *and the in-degree list* $(ind_{i1}, ind_{i2}, \ldots, ind_{im})$ *and out-degree list* $(outd_{i1}, outd_{i2}, \ldots, outd_{im})$ *of* l_i *are the presequences of the in-degree list* $(ind_{j1}, ind_{j2}, \ldots, ind_{jn})$ *and out-degree list* $(outd_{j1}, outd_{j2}, \ldots, outd_{jn})$ *of* l_j *respectively;*
v) $\forall l_i \in Post(t_1).postset, \exists l_j \in Post(t_2).postset$, *such that* $l_i = l_j$ *and the in-degree list* $(ind_{i1}, ind_{i2}, \ldots, ind_{iu})$ *and out-degree list* $(outd_{i1}, outd_{i2}, \ldots, outd_{iu})$ *of* l_i *are the presequences of the in-degree list* $(ind_{j1}, ind_{j2}, \ldots, ind_{jv})$ *and out-degree list* $(outd_{j1}, outd_{j2}, \ldots, outd_{jv})$ *of* l_j *respectively.*

According to Definition 4, if a candidate model *c* covers a query model *q*, the features and the neighborhood information of each task t_i in *q* should be preserved by at least one corresponding task t_j in *c*, which means that it must satisfy the following two conditions: *i)* t_j must be a candidate-task of t_i; and *ii)* t_j can match t_i.

In fact, if a task t_j in *c* is not a candidate-task of t_i in *q*, t_j cannot match t_i. For instance, in Fig. 2, the task c_2 with label "B" in CN3 is not a candidate-task of the task t_3 with label "B" in the query model *q*, thus c_2 cannot match t_3. This means that the *candidate-task* can be used to find the candidate tasks in the candidate model *c* for matching a task in the query model *q*. For each task t_i in *q*, we need to find all candidate tasks that may match t_i (denoted as *CT(t_i)*) in *c*. *CT(t_i)* is defined as Eq. (1).

$$CT(t_i) = \{ct_j | j = 1, 2, \ldots, n\}. \quad (1)$$

Where ct_j is the *j*th *candidate-task* of t_i in the candidate model *c*.

In order to verify whether a *candidate-task* t_j of t_i in *c* matches t_i in *q*, the concepts of *Position Sequence of Join-Tasks* (*PSJT*) and *Position Sequence of Split-Tasks* (*PSST*) should be presented first.

Definition 9 (Position Sequence of Join-Tasks). *Given a query model* q *and its matching sequence (denoted as* MSOQ, *which is used to store the matched tasks in the query model* q *during the subgraph isomorphism test), the* Position Sequence of Join-Tasks *of the current task* t_i *to be matched is a sequence of number, denoted as* $PSJT(t_i) = \{s_1, s_2, \ldots, s_n, k\}$, *where* $s_1, s_2, \ldots, s_n$ *are the position number of* t_i*'s join-tasks that have been matched in* MSOQ *(if there is no matched join-tasks, then* $s_1, s_2, \ldots, s_n$ *is equal to 0), and* k *is the number of* t_i*'s join-tasks that have not been matched.*

Definition 10 (Position sequence of split-tasks). *Given a query model* q *and its matching sequence* MSOQ, *the* Position Sequence of Split-Tasks *of the current task* t_i *to be matched is a sequence of number, denoted as* $PSST(t_i) = \{s_1, s_2, \ldots, s_n, k\}$, *where* $s_1, s_2, \ldots, s_n$ *are the position number of* t_i*'s split-tasks that have been matched in* MSOQ *(if there is no matched split-tasks, then* $s_1, s_2, \ldots, s_n$ *is equal to 0), and* k *is the number of* t_i*'s split-tasks that have not been matched.*

In order to build *PSJT(t_i)* and *PSST(t_i)* for each task t_i in *q*, it is necessary to record the position number of a task when it is pushed into the matching sequence *MSOQ*. Hence, it is straightforward to obtain *PSJT(t_i)* and *PSST(t_i)* by traversing the join-tasks and the split-tasks of the task t_i. Similarly, we can also define and build the *PSJT(t_j)* and *PSST(t_j)* for each candidate-task t_j of t_i in the candidate model *c*. According to Definitions 9 and 10, we can apply the following Lemma 1 to verify whether t_j matches t_i.

Lemma 1. *Given two tasks* t_i *and* t_j *in a query model* q *and a candidate model* c, *respectively,* t_j *is a candidate-task of* t_i, $PSJT(t_i) = \{r_1, r_2, \ldots, r_m, k_1\}$, $PSJT(t_j) = \{u_1, u_2, \ldots, u_n, k_2\}$, $PSST(t_i) = \{s_1, s_2, \ldots, s_h, m_1\}$, $PSST(t_j) = \{v_1, v_2, \ldots, v_k, m_2\}$, t_j *matches* t_i *iff:* i) $\{r_1, r_2, \ldots, r_m\}$ *is a subsequence of* $\{u_1, u_2, \ldots, u_n\}$ *and* $\{s_1, s_2, \ldots, s_h\}$ *is a subsequence of* $\{v_1, v_2, \ldots, v_k\}$; ii) $k_1 \leq k_2$ *and* $m_1 \leq m_2$.

Proof (By Adequacy and Necessity).

Adequacy. Given two tasks t_i and t_j in a query model q and a candidate model c, respectively, based on Definitions 9 and 10, we can build the *PSJT* and *PSST* of t_i and t_j, respectively when t_j is a candidate-task of t_i, i.e., the label, join-attribute and split-attribute of t_j is the same as the ones of t_i. Meantime, if $PSJT(t_i) = \{r_1, r_2, \ldots, r_m, k_1\}$, $PSJT(t_j) = \{u_1, u_2, \ldots, u_n, k_2\}$, $PSST(t_i) = \{s_1, s_2, \ldots, s_h, m_1\}$, $PSST(t_j) = \{v_1, v_2, \ldots, v_k, m_2\}$, such that : *i)* $\{r_1, r_2, \ldots, r_m\}$ is a subsequence of $\{u_1, u_2, \ldots, u_n\}$ and $\{s_1, s_2, \ldots, s_h\}$ is a subsequence of $\{v_1, v_2, \ldots, v_k\}$; *ii)* $k_1 \leq k_2$ and $m_1 \leq m_2$. It means that t_j preserves the neighbor information of t_i. According to Definitions 3 and 4, we can infer that t_j matches t_i.

Necessity. *i)* If $\{r_1, r_2, \ldots, r_m\}$ is not a subsequence of $\{u_1, u_2, \ldots, u_n\}$ or $\{s_1, s_2, \ldots, s_h\}$ is not a subsequence of $\{v_1, v_2, \ldots, v_k\}$, there exists at least one element r_i in $\{r_1, r_2, \ldots, r_m\}$ or s_i in $\{s_1, s_2, \ldots, s_h\}$ that is not contained in $\{u_1, u_2, \ldots, u_n\}$ or $\{v_1, v_2, \ldots, v_k\}$. It means that some join-tasks or split-tasks of task t_j cannot match r_i or s_i. Thus, t_j cannot match t_i; *ii)* If $k_1 > k_2$ or $m_1 > m_2$, there exists at least one join-task or split-task of t_i that cannot be matched with any neighbor of t_j. Thus, t_j cannot match t_i. □

Example 4. Fig. 3 shows a match between the query model q and the candidate model CN3 in Fig. 2 by using the matching sequences *MSOQ* and *MSOC*, where *MSOC* is used to store the partially matched tasks in the candidate model during the subgraph isomorphism test and the construct method of *MSOC* is the same as *MSOQ* (The process of constructing *MSOQ* and *MSOC* is shown in Algorithm 1). In Fig. 3, we assume that the current task to be matched is t_4, and the current candidate task is c_7. Then, as shown in Fig. 3, $PSJT(t_4) = \{2, 1\}$, $PSST(t_4) = \{0, 0\}$, $PSJT(c_7) = \{2, 3\}$ and $PSST(c_7) = \{0, 1\}$. According to Lemma 1, we can infer that c_7 matches t_4.

When all tasks of a query model q can find the corresponding matching tasks in a candidate model c, a match between the query model q and the candidate model c is found, which means q is subgraph isomorphic to c, i.e., c covers q. That is described in Lemma 2.

Lemma 2. *When the number of the matched tasks in* MSOQ *and* MSOC *is equal to the number of tasks in the query model* q, *i.e.,* $|MSOQ| = |MSOC| = |q|$ 1/2, *the pair of match sequence* MSOQ *and* MSOC *is a match mapping between* q *and* c, *i.e.,* q *is subgraph isomorphic to* c.

Proof (By Induction).

i) When $|MSOQ| = |MSOC| = 1$, obviously, the task in *MSOQ* matches the one in *MSOC*.

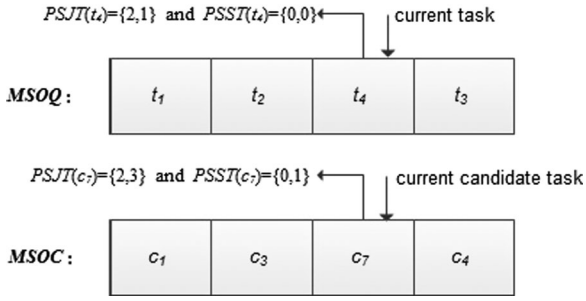


Fig. 3. *PSJT* and *PSST* of the current task in *q* and CN3 in Fig. 2.

ii) Suppose that when $|MSOQ| = |MSOC| = n$, the subgraph consisting of the tasks in *MSOQ* matches the corresponding subgraph consisting of the tasks in *MSOC*.

iii) Then, assume $|MSOQ| = |MSOC| = n + 1$, the subgraph consisting of the first n tasks in *MSOQ* matches the corresponding subgraph consisting of the first n tasks in *MSOC*, and the next task to be matched is t_{n+1}. According to Lemma 1, all the information of the candidate tasks of t_{n+1} is checked after matching t_{n+1}. Thus, when $|MSOQ| = |MSOC| = n + 1$, the subgraph consisting of the tasks in *MSOQ* matches the corresponding subgraph consisting of the *tasks* in MSOC.

According to the principle of mathematical induction, Lemma 2 is proved. □

4.2 Algorithm for Subgraph Isomorphism Test Based on *Task Code*

According to Definitions 6, 7, 8, 9, and10 and Lemmas 1-2, the match process of subgraph isomorphism test based on *task code* is conducted as shown in Algorithm 1.

In Algorithm 1, *MSOQ* and *MSOC* are initialized to *null*. Among the process, Line 4 is used to obtain the candidate tasks *CT*(*t*) of *t* by Eq. (1); Function GetLastTask(*MSOQ*) is used to obtain the last task in *MSOQ*, and Function VerifyTaskMatch(*v*,*t*) is used to check whether task *v* in a candidate model *c* matches task *t* in the query model *q* according to Lemma 2.

Unlike traditional subgraph isomorphism test algorithms, in which the filtering and the verification are divided into two different stages, Algorithm 1 combines the filtering process and the verification process together so that false positive candidate tasks can be filtered as early as possible. Meantime, in the matching process, if the sum of the in-degree and out-degree of a task is larger, more edges will be tested at a time. Thus, the performance of our subgraph isomorphism test is greatly improved.

5 Process of Exact Query Based on Graph Structure

According to the discussion in Section 2, we select the approach proposed in [14] as the benchmark. Based on whether considering the task similarity (see Section 5.2), two algorithms are proposed: Algorithm 2 (Exact query without task similarity) and Algorithm 3 (Exact query with task similarity).

Algorithm 1. Subgraph Isomorphism Test based on *Task Code* (SITC for short)

```
Input: q      //the query model represented as CNet
       c      // the candidate model represented as CNet
       STCQ// the descending sequence of task codes
              of all tasks in q
       STCC // the descending sequence of task codes
              of all tasks in c
Output: if c covers q, return true, else return false
 1 MSOQ = null, MSOC = null;
 2 while STCQ is not null do
 3 { Move the 1st task t in STCQ to MSOQ;
 4   CT(t) = GetCandidateTasks(t,q,c,STCQ,STCC);
 5   for each v∈CT(t) do
 6   { if VerifyTaskMatch(v,t) = true then
 7       { if v does not exist in MSOC then
 8           { push v into MSOC;
 9             for each t' ∈ neighbors of t do
10               if t' does not exist in MSOQ then
11                {t = t';
12                 push t into MSOQ and delete t from STCQ;
13                 goto Line 4;
14:              }
15:              }
16              else
17              {  delete v from MSOC;
19              }
20         }
21       }
22 if |MSOQ|! = |MSOC|then
23   { delete t from MSOQ and add t to STCQ;
24     if |MSOQ|! = 0 then
25     {  t = GetLastTask(MSOQ);
26        goto Line 4;
27:    }
28     else
29        return false;
30   }
31   else if |MSOQ| = |q1/2 then
32          return true;
33 }
```

5.1 Query Processing without Task Similarity

Being similar to the corresponding algorithm (without label similarity) proposed in [14] (denote as QTSU), the query processing in our approach is also divided into two stages, namely, a filtering stage and a verification stage. In the first stage, we use the *composite task index* to filter the cloud workflow repository, and get the candidate model set. And then, SITC is adopted to check whether the query model q is subgraph isomorphic to each candidate model c. The details of exact query based on *composite task index* and SITC without task similarity are shown in Algorithm 2.

In Algorithm 2, Lines 1, 2, 3, 4, 5, and 6 specify the filtering stage and Lines 7, 8, 9, 10, 11, 12, 13, and 14 specify the verification stage. In the filtering stage, it first extracts all tasks from the query model q; and then, for each extracted task, it uses the *composite task index* to locate the models containing the same index and create a temporary model set to store them; finally, by Line 6, the intersection of these candidate model sets is computed to get the candidate models containing all the extracted tasks, and store them into R_q. In the verification stage, Line 7 obtains the *task code* sequence of the query model q and Line 8 sorts the *STCQ* in descending order according to the sum of the in-degree and out-degree of the task. Lines 9, 10, 11, 12, 13, and 14 work as a refiner to get the final model set. For each candidate model c in R_q, Line 10 obtains its *task code* sequence and Line 11 sorts the *STCQ* in descending order. And then, function SITC, as described in Algorithm 1, is used to verify whether q is subgraph isomorphic to c. If q is not subgraph isomorphic to c, c will be deleted from the set R_q as shown in Line 13. Therefore, at last, the models left in R_q are those who can cover the query model q.

Algorithm 2. Exact Query without Task Similarity

```
Input:  q   // the query model represented as CNet
        R   // the CNet model repository
Output: R_q // a set of CNet models covering q
//the filtering stage
 1 tasks = ExtractTask(q);
 2 for each task in tasks do
 3 {model_set = getModelListByCompositeTaskIndex(task,R);
 4   add model_set to model_set_lists;
 5 }
 6 R_q = GetIntersection(model_set_lists);
//the verification stage
 7 STCQ = ExtractTaskCode(q);
 8 STCQ = SortByDesc(STCQ);
 9 for each c in R_q do
10  { STCC = ExtractTaskCode(c);
11    STCC = SortByDesc(STCC);
12    if SITC(q,c,STCQ,STCC) = false then
13       delete c from R_q;
14 }
15 return R_q;
```

In Algorithm 2, suppose that $|R1/2$ is the number of process models in the repository R, k is the number of tasks in the query model q, m is the number of candidate models obtained by the *single task index* proposed in [14] and n is the number of candidate models obtained with the *composite task index*. According to the definition of the *composite task index*, it consists of the label, the join-attribute and the split-attribute. Both the join-attribute and the split-attribute have four values: NONE, AND, XOR, and OR. Thus, on the average, the filtering capability of the *composite task index* is as 4*4 times as the *single task index*. Correspondently, as to find candidate models which contain all the k tasks of a query model, the filtering capability of the *composite task index* is as $(4^*4)^k$ times as the *single task index*. Therefore, on the average, the number of candidate models obtained with the *composite task index* will be $(1/16)^k$ of the number of candidate models obtained with the *single task index*, i.e., $n = (1/16)^k m$. Therefore, the efficiency of querying using the *composite task index* is much better than using the *single task index*.

At the same time, the verification stage in Algorithm 2 costs less time than QTSU. Given a query model q and a candidate model c, define v as the average number of tasks in candidate models, d as the average degree of all tasks in a process model, u as the number of tasks in q. According to

[16], for a given model graph with p nodes, the Ullman's subgraph isomorphism algorithm (denoted as USIA) finds all isomorphism in a time roughly proportional to p^3. So, the time complexity of testing a candidate model by USIA is $O(v^3)$, while the time complexity of each test by SITC in Algorithm 2 is $O(vud)$. In practice, $d < v$ and $u < v$, thus $O(vud) \ll O(v^3)$. This means that in the verification stage, SITC costs less time than USIA even if the number of candidate models in Algorithm 2 is the same as the one in QTSU.

Actually, the number of candidate models obtained with the *composite task index* is much smaller than the one obtained with the *single task index*. Therefore, Algorithm 2 has better retrieval efficiency than QTSU.

5.2 Query Processing with Task Similarity

In the modeling process, different tenants may choose different names (labels) for the identical tasks in process models. To recognize the same tasks with different or similar labels, [14] presents the concept of semantic similarity between labels and constructs the *label index*; [20] expands basic query primitives (e.g., *CanOccurAll* is expanded to *CanOccurAllExpanded*) by constructing the sets of similar labels. In addition, [5] adopts user feedback to adapt the label similarity to get higher matching quality.

Based on the modeling features of CNet, the similarity between tasks depends on not only the semantic similarity between their labels, but also the similarity of their behavior attributes, such as their join-attributes and split-attributes. Therefore, to improve the tasks matching quality and efficiency, both the label semantic similarity and the similarity of the join-attribute and split-attribute should be considered. So the following equation, Eq. (2), is proposed to calculate the degree of similarity between tasks t_1 and t_2.

$$\begin{aligned}&\mathrm{tasksim}(t_1, t_2)\\ &= \begin{pmatrix} 0, & \text{if } labelsim(t_1.label, t_2.label) < \theta_1 \\ 0, & \text{if } flowsim(t_1, t_2) < \theta_2 \\ labelsim(t_1.label, t_2.label) \times flowsim(t_1, t_2), & \text{else} \end{pmatrix}.\end{aligned} \tag{2}$$

In Eq. (2), θ_1 is the label similarity threshold, θ_2 is the flow similarity threshold, function *labelsim*(l_1,l_2) is the degree of semantic similarity between labels l_1 and l_2, and *attrisim*(a_1, a_2) is the degree of similarity between behavior attributes a_1 and a_2. To facilitate comparison, the equation of *labelsim*(l_1, l_2) used in the following experiments is the same as the corresponding equation in [14]. As the flow similarity between two tasks t_1 and t_2 is determined by the similarity of their join-attributes and split-attributes, the following Eq. (3) is proposed to calculate *flowsim*(t_1, t_2)

$$\begin{aligned}flowsim(t_1, t_2) =& (attrisim(t_1.join, t_2.join) \\ &+ attrisim(t_1.split, t_2.split))/2.\end{aligned} \tag{3}$$

The default value of *attrisim*(a_1,a_2) can be obtained from Table 3, which can be reset according to the user's preference.

Example 5. Given two tasks t_1 and t_2, the join-attributes of t_1 and t_2 are "AND" and "OR", and the split-attributes of t_1 and t_2 are "XOR" and "AND". According to Eq. (3) and Table 3, the flow similarity between t_1 and t_2 is : *flowsim*(t_1, t_2) = (*attrisim*("AND","OR")+*attrisim*("XOR","AND"))/ 2 = (0.75+ 0.25)/2 = 0.5.

TABLE 3
The Degree of Similarity between Attributes a_1,a_2

a_2 \ a_1	NONE	AND	OR	XOR
NONE	1	0	0	0
AND	0	1	0.75	0.25
OR	0	0.75	1	0.5
XOR	0	0.25	0.5	1

Assume θ is the task similarity threshold, θ_1 is the label similarity threshold, and θ_2 is the flow similarity threshold. According to Eq. (2), the relation among θ and θ_1, θ_2 is denoted as Eq. (4).

$$\theta = \theta_1 \times \theta_2. \tag{4}$$

The flow similarity threshold θ_2 can be set according to users' preferences. In the following experiments, we suppose $\theta_2 = 0.5$. According to Eq. (4), $\theta = 0.5\theta_1$.

To take task similarity into account, some changes should be made in the filtering stage and verification stage of Algorithm 2, respectively. For instance, during the process of verifying whether the query model q is subgraph isomorphic to a candidate model c, the selection criterion of candidate tasks is different. Thus, the concept of *similar candidate-task* should be defined.

Definition 11 (Similar candidate-task). *Given two tasks* t_1 *and* t_2 *in a query model* q *and a candidate model* c, *respectively*, t_2 *is a* similar candidate-task *of* t_1 *iff:*

- *i)* $\mathrm{tasksim}(t_1, t_2) \geq \theta$;
- *ii)* $Pre(t_1).prenum \leq Pre(t_2).prenum$ *and* $Post(t_1).postnum \leq Post(t_2).postnum$;
- *iii)* $\forall l_i \in Pre(t_1).preset$, $\exists l_j \in Pre(t_2).preset$, *such that* $labelsim(l_i, l_j) \geq \theta_1$ *and the in-degree list* $(ind_{i1}, ind_{i2}, \ldots, ind_{im})$ *and out-degree list* $(outd_{i1}, outd_{i2}, \ldots, outd_{im})$ *of* l_i *is a presequence of the in-degree list* $(ind_{j1}, ind_{j2}, \ldots, ind_{jn})$ *and out-degree list* $(outd_{j1}, outd_{j2}, \ldots, outd_{jn})$ *of* l_j, *respectively;*
- *iv)* $\forall l_i \in Post(t_1).postset$, $\exists l_j \in Post(t_2).postset$, *such that* $labelsim(l_i, l_j) \geq \theta_1$ *and the in-degree list* $(ind_{i1}, ind_{i2}, \ldots, ind_{iu})$ *and out-degree list* $(outd_{i1}, outd_{i2}, \ldots, outd_{iu})$ *of* l_i *is a presequence of the in-degree list* $(ind_{j1}, ind_{j2}, \ldots, ind_{jv})$ *and out-degree list* $(outd_{j1}, outd_{j2}, \ldots, outd_{jv})$ *of* l_j, *respectively.*

According to Definition 11, it is obvious that the threshold of task similarity can affect the size of the candidate model set. The algorithm for exact query with task similarity is as follows.

Algorithm 3 differs from Algorithm 2 only on Lines 3, 4, 5, 6, and 7 and Line 16. Line 3 initializes the *model_set*, Line 4 obtains all the similar tasks of a given task according to Eq. (2). Lines 5, 6, and 7 collect the models where the given task or its similar tasks occur. In Line 16, the candidate tasks are obtained by SITC based on Definition 11.

Algorithm 3. Exact Query with Task Similarity

Input: q // the query model represented as CNet
R // the CNet model repository
Output: R_q // a set of CNet models covering q
//the filtering stage
1 $tasks$ = ExtractTask(q);
2 for each $task$ in $tasks$ do
3 { $model\_set = null$;
4 $similartasks$ = getSimilarTask($task$, R);
5 for each $similartask$ in $similartasks$ do
6 {$model\_set = model\_set$ + getModelListByCompositeTaskIndex($similartask$, R);
7 }
8 add $model\_set$ to $model\_set\_lists$;
9 }
10 R_q = GetIntersection($model\_set\_lists$);
//the verification stage
11 $STCQ$ = ExtractTaskCode(q);
12 $STCQ$ = SortByDesc($STCQ$);
13 for each c in R_q do
14 { $STCC$ = ExtractTaskCode(c);
15 $STCC$ = SortByDesc($STCC$);
16 if SITC(q,c,$STCQ$,$STCC$) = $false$ then
17 delete c from R_q;
18 }
19 return R_q;

To compare Algorithm 3 with the corresponding algorithm that is used for dealing with semantic similarity between labels in [14], we suppose that the flow similarity threshold θ_2 in Algorithm 3 is set to 0.5 and the flow similarity obeys uniform distribution. According to uniform distribution and Table 3, the probability that the flow similarity of any two tasks t_1 and t_2 is greater than 0.5 is 17/32. Thus, if only the flow similarity is taken into consideration, the probability of existence of k tasks with similar join-attributes and split-attributes is approximately equal to $(17/32)^k$. Given a process model repository R and a query model q, suppose that m and n are the numbers of candidate models obtained with the *single task index* in the corresponding algorithm (with label similarity) in [14] and the *composite task index* in Algorithm 3, respectively under the same label similarity threshold. Then, based on the above analysis, $n = (17/32)^k m$. Besides, each verification using SITC also costs less time than USIA according to the analysis in Section 5.1. Therefore, the performance of Algorithm 3 is better.

6 Experiments and Evaluation

To evaluate our approach, we developed a Cloud Workflow Management Platform (CWMP), which can be accessed by http://platform.pasp.cn. It includes the model generating tool, the model importing tool, the visual modeling tool, the model querying tool and the result displaying tool. The model generating tool is used to generate synthetic process models. All the labels of the tasks in the synthetic process models are selected from the function words of WordNet. The type, join-attribute and split-attribute of each task are randomly assigned. The rules used for generating the control-flow perspective in our generator come from [37]. The model importing tool is used to import models

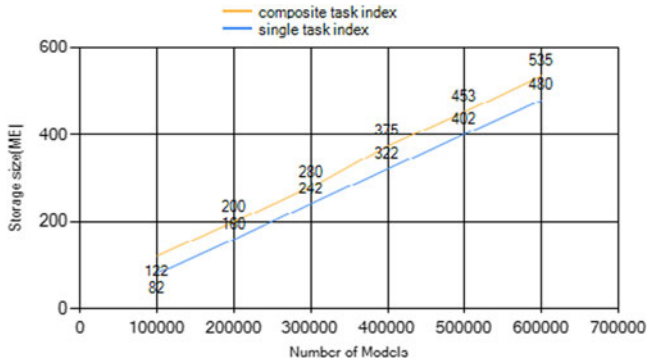


Fig. 4. Index storage size comparison.

represented by YAWL and convert them to CNets. All the CNets can be visually displayed or modified in the visual modeling tool (http: //cbpm.pasp.cn). All query algorithms proposed in this paper have been implemented in the model querying tool, and the following experiments are all based on it. The result displaying tool is used to display the results and analyses of experiments.

In CWMP, we conducted several experiments on both synthetic data and real data. In the experiments, all the indexes, including the *single task index*, the *composite task index* and the *label index*, are managed by Apache Lucene [22], which is a search engine designed for efficient text search. According to the discussion in Section 3.2, the time complexity of constructing *composite task indexes* is the same as the one of constructing *single task indexes*. The comparison between the storage needed by the *composite task indexes* and the *single task indexes* is shown in Fig. 4.

As shown in Fig. 4, with the increase of the size of the repository, the storage sizes needed by *composite task indexes* and *single task indexes* increase correspondingly. And the storage size needed by *composite task indexes* is about 22 percent bigger than the one needed by *single task indexes*, since *composite task indexes* need to index more information. Since the storage size for *composite task indexes* is just about 1 percent of the one that is required by the corresponding model repository, the storage cost of constructing *composite task indexes* is acceptable. At the same time, whenever a new model is added to the repository, neither the *single task indexes* nor the *composite task indexes* need to be reconstructed. They only need to be updated.

Before conducting the query experiments, we compared the efficiency between SITC and USIA. We randomly generate six CNet models as candidate models by the model generating tool. The number of the tasks in the six CNet models are 30,50,70,90,110, and 130, respectively. Meantime, we randomly extract 10 process model fragments from the six candidate models as the query model q_i $(1 \leq i \leq 10)$, in which the number of tasks varies from 2 to 20. For each query model, we conduct six subgraph isomorphism tests with the candidate models by SITC and USIA, respectively. The time comparison between two subgraph isomorphism algorithms is shown in Fig. 5.

As shown in Fig. 5, with the increase of the number of tasks in candidate models, both SITC and USIA cost more time. For the candidate models with the same number of tasks, the time cost of SITC is much shorter than USIA. At the same time, the more tasks the candidate model contains,

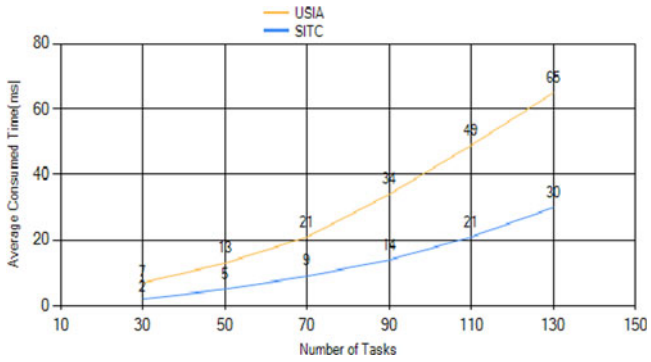


Fig. 5. Time comparison between SITC and USIA.

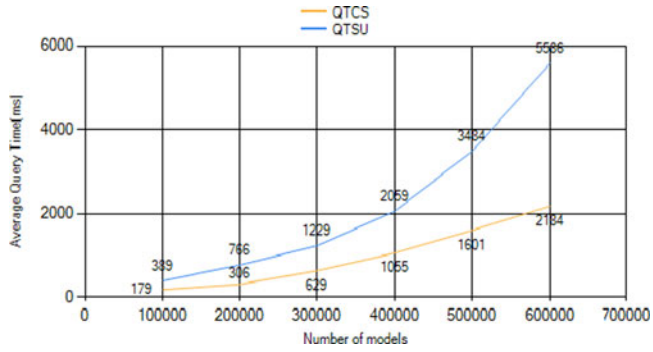


Fig. 6. Query time comparison without considering task similarity.

the more time can be saved by SITC. The reason can be found in the algorithm analysis in Section 4.2.

6.1 Experiments on Synthetic Process Models

To compare the query efficiency between Algorithm 2 (denoted as QTCS) and QTSU, several experiments are conducted on the synthetic process model repository R, in which all cloud workflow models are generated by the model generating tool according to the features of cloud workflow. E.g., tasks in cloud workflow models always have the same or similar labels and cloud workflow models always have common process fragments.

Finally, in the repository R, we generate six large process model sets, and the number of models in these model sets is 100,000, 200,000, 300,000, 400,000, 500,000, and 600,000, respectively. In these model sets, the number of tasks ranges from 2 to 50, for process models should be decomposed if they have more than 50 elements to ensure the readability and understandability according to 7PMG [38]; and the number of edges ranges from 2 to 308. Thus, the total number of tasks in these model sets ranges from 2,321,568 to 13,944,907, and the total number of edges ranges from 3,207,539 to 19,265,511. After that, we use the model generating tool to produce 10 process model fragments as the query model q_i $(1 \leq i \leq 10)$, in which the number of tasks ranges from 2 to 30 and the number of edges ranges from 1 to 70. Then, these 10 query models are leveraged to conduct the exact query in each model sets by QTCS and QTSU, respectively. The response time of each exact query is recorded in the result data table.

As shown in Fig. 6, with the increase of the number of models, the time cost of QTCS and QTSU increases at the same time. Meantime, the average query time of QTCS is much shorter than the one needed by QTSU. The amount of time saved by QTCS increases with the increase of the number of models in the model set, which means that QTCS achieves better query efficiency than QTSU.

In order to analyze the experiment results, we define R_S as the candidate mode set obtained by using the *single task index*, R_c as the candidate model set obtained by using the *composite task index*, T_S as the traversal time of the *single task index*, T_c as the traversal time of the *composite task index* (in fact, $T_S \approx T_c$), T_{vs} as the time required to check whether a candidate model is an exact match by using USIA, and T_{vc} as the time required to check whether a candidate model covers the query model q by SITC.

The query response time of QTSU and QTCS can be calculated according to Eq. . (5) and Eq. (6) respectively:

$$T^3 = T_c + |R_c| \times T_{vc}, \tag{5}$$

$$T' = T_c + |R_c| \times T_{vc}. \tag{6}$$

According to the theoretical analysis in Section 5.1, $|R_c| \ll |R_s|$ and $|T_{vc}| < |T_{vs}1/2$, thus $T\prime \ll T$, i.e., the query response time of QTCS is much shorter than QTSU. To quantitatively compare the query efficiency between QTCS and QTSU, the average speedup rate is defined as follows.

Definition 12 *(Average Speedup Rate).*

Given two query algorithms A1 and A2, by conducting the query tests n times, the average query time of A1 and A2 are $t_{11}, t_{12}, \ldots, t_{1n}$ and $t_{21}, t_{22}, \ldots, t_{2n}$ respectively. When compared with A1, the average speedup rate (denoted as *asr*(A1,A2)) of A2 can be obtained by Eq. (7):

$$asr(\mathrm{A1}, \mathrm{A2}) = \left(\sum_{i=1}^{n} t_{1i} - \sum_{i=1}^{n} t_{2i}\right) \Big/ \sum_{i=1}^{n} t_{1i}. \tag{7}$$

According to Definition 12, compared with the baseline algorithm from [14] that is shown as QTSU in Fig. 6, our new algorithm (QTCS in Fig. 6) resulted in a 57.4 percent speedup in average, which complies with the result of the theoretical analysis. Therefore, we can draw a conclusion that QTCS can greatly improve the efficiency of the exact query based on graph structure in large process model repositories without considering task similarity.

6.2 Experiments on Real Process Models

To observe the relation between the query time and the similarity threshold, several experiments are conducted by Algorithm 3 proposed in this paper (denoted as TS-QTCS), the corresponding algorithm proposed in [14] (denoted as TS-QTSU) and the one proposed in [20] (denoted as TS-QTBU), respectively. The experiments are conducted on a SAP reference model repository R'. To construct R', about 600 SAP reference models are first transformed into YAWL models with ProM [39] and then imported into the database of CWMP with the model importing tool. The visual modeling tool is used to create 10 real process model fragments as the query model q_i $(1 \leq i \leq 10)$, in which the tasks and flow relations are abstracted from R'. For each query model q_i, by TS-QTCS, TS-QTSU, and TS-QTBU, respectively, several query experiments are performed with different label similarity thresholds θ_1(θ_1 varies from 0.5 to 1 with the growing

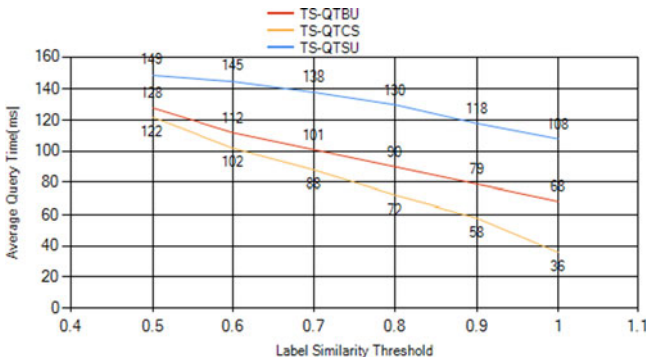


Fig. 7. Query time comparison with different label similarity threshold.

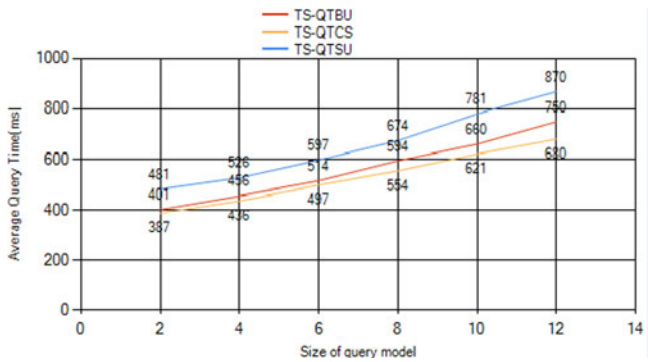


Fig. 8. Query time comparison with different size of query models.

step 0.1). The final results of experiments on the SAP reference model repository R' are shown in Fig. 7.

As shown in Fig. 7, with the increase of the similarity threshold, the average query time of TS-QTCS, TS-QTSU and TS-QTBU reduces, respectively. The reason is that when the similarity threshold increases, the number of similar task labels decreases, which lead to the decrease of the number of the candidate model set. Moreover, the figure shows that as to the same label similarity threshold the average query response time of TS-QTCS is much shorter than TS-QTSU and TS-QTBU, and the higher the label similarity threshold is, the greater the amount of time can be saved by TS-QTCS.

In addition, in order to observe the relation between the query time and the size of the query model (i.e., the number of tasks in the query model) and to evaluate the total efficiency of using TS-QTCS, TS-QTSU, and TS-QTBU on large process model repositories in cloud workflow systems or other process-aware information systems (e.g., Suncorp, BIT Process Library, etc. [21]), several experiments are conducted on the model repository of CCSP, in which there are about 100,000 business process models (denoted as R'').

First, 60 real process model fragments are selected as the query models, in which the tasks and flow relations are abstracted from R''. The 60 query models are grouped into six query model sets (each model set contains 10 query models) according to the number of tasks in each query model, which are 2,4,6,8,10 and 12, respectively.

Then, for each query model set, TS-QTCS, TS-QTSU, and TS-QTBU are used respectively to conduct query experiments with the same label similarity threshold $\theta_1(\theta_1 = 0.8)$. The results of experiments on R'' are shown in Fig. 8.

As shown in Fig. 8, with the increase of the number of tasks in the query model, the average query time of TS-QTCS, TS-QTSU, and TS-QTBU increases too. At the same time, the average query time of TS-QTCS is much shorter than the one needed by TS-QTSU and TS-QTBU. Namely, TS-QTCS is more efficient than TS-QTSU and TS-QTBU with the same label similarity threshold.

In summary, compared with TS-QTSU and TS-QTBU (according to the data recorded in Figs. 7 and 8,) TS-QTCS resulted in 30 and 16 percent speedup in average respectively, when considering task similarity.

Based on all the above analysis, we can come to the conclusion that our approach can greatly improve the efficiency of exact query based on graph structure of large process model repositories no matter whether task similarity is considered or not. In a word, our approach can achieve efficient and exact query of large process model repositories in cloud workflow systems.

7 Conclusion and Future Work

This paper focuses on improving the efficiency of graph structure based exact query on large process model repositories in cloud workflow systems. The main idea of this paper is inspired by the concept of composite index from the field of database management technology. The *composite task index*, including the label, join-attribute and split-attribute of a task, is leveraged to reduce the number of the candidate models in the filtering stage, and then the subgraph isomorphism test based on *task code* is proposed to refine the candidate models and get the more exact matching models. To make our approach more flexible, the task similarity is taken into account to give the modeler more choices when they launch a retrieval. Extensive experiments are conducted to evaluate the proposed algorithms. The experiment results show that our approach can obtain better query efficiency than existing approaches at the expense of affordable storage space.

Further investigation can be carried out from the following aspects: *i*) to further differentiate two tasks, the *composite task index* can be extended to include more task attributes, such as its data or resource attribute; *ii*) another valuable work is to extend our approach in parallel computing environment (e.g., Hadoop Ecosystem) to improve the retrieval efficiency of super-large process model repositories.

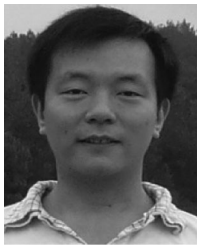
Acknowledgments

The authors would like to thank Jianmin Wang, Lijie Wen and Tao Jin (the authors in [14]) for they provided the SAP reference models. This work is supported by the National 973 Basic Research Program of China under Grant No. 2014CB340404, the National Natural Science Foundation of China under Grant Nos. 61170026,61373037 and 61100017, the National Science and Technology Ministry of China under Grant Nos. 2012BAH25F02 and 2013BAF02B01 and the Fundamental Research Funds for the Central Universities of China under Grant Nos. 2012211020203 and 2042014kf0237. Prof. R. Peng is the corresponding author.

References

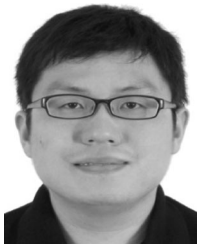
[1] T. Baeyens, "BPM in the Cloud," in *Proc. BPM*, 2013, pp. 10–16,
[2] X. Liu, D. Yuan, G. Zhang, et al., *The Design Of Cloud Workflow Systems*. Berlin, Germany: Springer, 2012.

[3] X.-z. Chai and J. Cao, "Cloud computing oriented workflow technology," *J. Chinese Comput. Syst.*, vol. 33, no. 1, pp. 90–95, 2012 (in Chinese with English abstract).
[4] M. Dumas, L. García-Bañuelos, and R. M. Dijkman, "Similarity search of business process models," *IEEE Data Eng. Bull.*, vol. 32, no. 3, pp. 23–28, Sept. 2009.
[5] Klinkmüller, Christopher, et al., "Listen to me: Improving process model matching through user feedback," in *Proc. 12th Int. Conf. Bus. Process Manage.*, 2014, 84–100.
[6] A. Awad, A. Polyvyanyy, and M. Weske, "Semantic querying of business process models," in *Proc. IEEE 12th Int. Conf. Enterprise Distrib. Object Comput.*, 2008, pp. 85–94.
[7] C. Gilbert, B. Payam, and M. Klaus, "Probabilistic matchmaking methods for automated service discovery," *IEEE Trans. Service Comput.*, vol. 7, no. 4, pp. 654–666, Oct. 2014.
[8] S. Sakr and A. Awad "A framework for querying graph-based business process models," in *Proc. ACM 19th Int. Conf. World Wide Web.*, 2010, pp. 1297–1300.
[9] T. Jin, J. Wang, and L. Wen "Querying business process models based on semantics," in *Proc. 16th Int. Conf. Database Syst. Adv. Appl.*, 2011, pp. 164–178.
[10] M. Yinglong, Z. Xiaolan, and L. Ke, "A graph distance based metric for data oriented workflow retrieval with variable time constraints," *Expert Syst. Appl.*, vol. 41, pp. 1377–1388, 2014.
[11] Z. C. Huang, J. P. Huai, X. D. Liu, X. Li, and J. J. Zhu, "Automatic service discovery framework based on business process similarity," *J. Softw.*, vol. 23, no. 3, pp. 489–503, 2012.
[12] M. Kunze and M. Weske, "Metric trees for efficient similarity search in large process model repositories," in *Proc. In:Bus. Process Manag. Workshops*, 2010, pp. 535–546.
[13] T. Jin, J. Wang, and L. Wen, "Efficient retrieval of similar workflow models based on behavior," in *Proc. 14th Asia-Pacific Web Conf. Web Technol. Appl.*, 2012, pp. 677–684.
[14] T. Jin, J. Wang, M. La Rosa, A. ter Hofstede, and L. Wen, "Efficient querying of large process model repositories," *Comput. Ind.*, vol. 64, pp. 41–49, 2013.
[15] T. Jin, J. Wang, N. Wu, M. L. Rosa, and A. H. M. ter Hofstede, "Efficient and accurate retrieval of business process models through indexing," in *Proc. OTM Conf.*, 2010, pp. 402–409.
[16] J. Ullmann. "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, no. 1, pp. 31–42, 1976.
[17] Z. Yan, R. Dijkman, and P. Grefen, "FNet: An index for advanced business process querying,"in *Proc. 10th Int. Conf. Bus. Process Manage.*, 2012, pp. 246–261.
[18] M. Qiao, R. Akkiraju, and A. J. Rembert, "Towards efficient business process clustering and retrieval: combining language modeling and structure matching," in *Proc. 9th Int. Conf. Bus. Process Manage.*, 2011, pp. 199–214.
[19] M. Becker and R. Laue, "A comparative survey of business process similarity measures," *Comput. Ind.*, vol. 63, no. 2, pp. 148–167, 2012.
[20] A. Polyvyanyy, M. La Rosa, and A. H. M. Ter Hofstede, "Indexing and efficient instance-based retrieval of process models using untanglings," in *Proc. 26th Int. Conf. Adv. Inf. Syst. Eng.*, 2014, pp. 439–456.
[21] D. Patrick and S. Höhenberger, "Supporting business process improvement through business process weakness pattern collections, in *Proc. 12th Int. Conf. Wirtschaftsinformatik*, Mar. 2015, Osnabrück, Germany, pp. 4–6.
[22] Apache, Lucene web-site [Online]. Available: http://lucene.apache.org, 2013.
[23] Q. Shao, P. Sun, Y. Chen, "WISE: A workflow information search engine," in *ICDE*, 2009, pp. 1491–1494.
[24] K. Shahzad, M. Elias, and P. Johannesson, "Requirements for a business process model repository: A stakeholders' perspective," *Business Inform. Syst.*, vol. 47, pp. 158–170, 2010.
[25] J. Wang, T. Jin, W. Raymond, K. Wong, and L. Wen, "Querying business process model repositories," *World Wide Web*, vol. 17, pp. 427–454, 2014.
[26] H. Jiang, H. Wang, P. S. Yu, and S. Zhou, "GString: a novel approach for efficient search in graph databases," In *Proc. IEEE 23rd Int. Conf. Data Eng.*, Apr. 2007, pp. 566–575.
[27] D. Shasha, J. T. L. Wang, and R. Giugno, "Algorithms and applications of tree and graph searching," in *Proc. 21st ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Syst.*, 2002, pp. 39–52.
[28] X. Yan, P. S. Yu, and J. Han, "Graph indexing: A frequent structure-based approach," in *Proc. SIGMOD Conf.*, 2004, pp. 335–346.
[29] X. Yan, P. S. Yu, and J. Han, "Graph indexing based on discriminative frequent structure analysis," *ACM Trans. Database Syst.*, vol. 30, no. 4, pp. 960–993, 2005.
[30] S. Zhang, M. Hu, J. Yang, "TreePi: A novel graph indexing method," in *Proc. IEEE 23rd Int. Conf. Data Eng.* Apr. 2007, pp. 966–975,.
[31] P. Zhao, J. X. Yu, and P. S. Yu, "Graph indexing: Tree + Delta >= graph," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 938–949.
[32] J. Cheng, Y. Ke, W. Ng, and A. Lu, "Fg-index: Towards verification-free query processing on graph databases," in *Proc. SIGMOD Conf.*, 2007, pp. 857–872.
[33] J. Cheng, Y. Ke, and W. Ng, "Efficient query processing on graph databases," *ACM Trans. Database Syst.*, vol. 34, no. 1, pp. 1–44, 2009.
[34] F. Zhe, P. Yun, C. Byron, X. Jianliang, and B. Sourav S. "Towards efficient authenticated subgraph query service in outsourced graph databases," *IEEE Trans. Service Comput.*, vol. 7, no. 4, pp. 696–713, Aug. 2014.
[35] A. Awad and S. Sakr, "Querying graph-based repositories of business process models," in *Proc. DASFAA Workshops*, 2010, pp. 33–44.
[36] A. H. M. Hofstede, W. M. P. Aalst, M. Adams, and N. Russell, *Modern Business Process Automation:YAWL and Its Supports Environment*. New York, NY, USA: Springer, 2010.
[37] M. T. Wynn, H. M. W. E. Verbeek, W. M. P. van der Aalst, A. H. M. ter Hofstede, and D. Edmond, "Reduction rules for YAWL workflows with cancellation regions and OR-joins," *Inform. Softw. Technol.*, vol. 51, no. 6, pp. 1010–1020, 2009.
[38] J. Mendling, H. A. Reijers, and W. M. P. van der Aalst, "Seven Process Modeling Guide-lines (7PMG)," *Inform. Softw. Technol.*, vol. 52, no. 2, pp. 127–136, 2010.
[39] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst, "The ProM framework: a new era in process mining tool support," in *Proc. 26th Int. Conf. Appl. Theory Petri Nets*, 2005, pp. 444–454.



Hua Huang is currently working toward the PhD degree in the School of Computer, Wuhan University, China. He is an associate professor in the School of Information Engineering, Jingdezhen Ceramic Institute. His research interests include cloud computing and business process model management.

Rong Peng is a professor and doctoral supervisor in the School of Computer, Wuhan University, China. Her research interests include requirements engineering, software engineering, and cloud computing, etc.



Zaiwen Feng is a lecturer in the School of Computer, Wuhan University, China. His research interests include business process management, service computing, and software requirements modeling and so on.