The Journal of Systems and Software 113 (2016) 59–75

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Run-based exception prediction for workflows

Yain-Whar Si [a,*], Kin-Kuan Hoi [a], Robert P. Biuk-Aghai [a], Simon Fong [a], Defu Zhang [b]

[a] Department of Computer and Information Science, University of Macau, Macao
[b] Department of Computer Science, Xiamen University, China

ARTICLE INFO

Article history:
Received 31 January 2015
Revised 10 November 2015
Accepted 14 November 2015
Available online 26 November 2015

Keywords:
Workflow
Run
Temporal exception prediction

ABSTRACT

Events such as iteration of activities or lack of available resources can cause temporal exceptions in business processes. Exception prediction can improve the quality of workflow execution since preventive actions can be taken to reduce the occurrence of exceptions. Thus, it is crucial to provide an accurate and efficient temporal exception prediction capability for workflow management systems. In this paper, we propose a run-based exception prediction algorithm to predict temporal exceptions in workflows. The proposed algorithm is divided into two phases, design-time and run time. At design-time, all possible runs are generated from a workflow and their estimated execution time and mapping probability are calculated. At run time, temporal exceptions are predicted by analyzing the runs. Simulation experiments are performed to evaluate the proposed approach using five workflow models having different characteristics. Simulation experiments show that our approach is efficient and produces good results in prediction accuracy.

1. Introduction

A workflow is a collection of tasks organized to achieve business goals (Li and Yang 2005; Pan, Tang et al. 2005; Son, Kim et al. 2005; Dumas, Garcıa-Banuelos et al. 2011). Various aspects of workflow management systems have been extensively reported in the literature, such as verifying and optimizing workflows (Cao, Jin et al. 2013). One of the most important issues in workflow management is the time management for executing its activities. For instance, a workflow can be designed with a set of tasks to be executed by staff for delivery of goods. The process begins when an order for goods is received and ends when the goods are delivered to the customer and the payment is received. Any delay in completing the process can lead to financial loss, damaged reputation, or customer dissatisfaction. Therefore, predicting temporal exceptions and taking necessary measures to avoid delays in execution of the processes can significantly reduce costs and in the long run make businesses more competitive.

A Workflow Management System (WFMS) is designed to manage the workflows and organize the flow of tasks such that a task is performed by the right person/application (resource) at the right time (del Foyo and Silva 2008). Since time plays an important role in a WFMS, it is necessary to provide time-related constraints, such as bounded execution durations of tasks and absolute deadlines of workflows (Eder and Pichler 2005). However, some unexpected events can cause time violations as some of the tasks need to be repeated or necessary resources are unavailable. A temporal exception occurs when a time constraint defined at design time is violated during run time (Xie, Yu et al. 2009).

Exception prediction can be useful in improving the quality of workflow execution since preventive actions can be taken to reduce the occurrence of exceptions. When a workflow instance is predicted to have a high probability of violating its deadline constraints, a workflow administrator can allocate more resources, assigning a higher priority to reduce the likelihood of deadline violation (Grigori, Casati et al. 2001). Thus, it is crucial to provide an accurate and efficient temporal exception prediction capability in a WFMS.

A number of exception prediction methods have been proposed in the literature (Eder, Gruber et al. 2000; Eder and Pichler 2002; Leong, Si et al. 2012; Yu, Xie et al. 2013). However, these approaches do not consider a number of critical issues in the control flow of workflows. In this paper, we propose a novel temporal exception prediction method for workflows addressing these control flow aspects. Specifically, in our approach, the concept of runs (the complete execution path) of a workflow is used for predicting temporal exceptions.

1.1. Motivation

Various approaches to workflow exception prediction can be found in the literature (Eder, Gruber et al. 2000; Eder and Pichler 2002; Leong, Si et al. 2012; Yu, Xie et al. 2013). However, they

* Corresponding author. Tel.: +853 8822 4454; fax: +853 8822 2426.
E-mail addresses: fstasp@umac.mo (Y.-W. Si), neti1723@hotmail.com (K.-K. Hoi), robertb@umac.mo (R.P. Biuk-Aghai), ccfong@umac.mo (S. Fong), dfzhang@xmu.edu.cn (D. Zhang).

http://dx.doi.org/10.1016/j.jss.2015.11.024
0164-1212/

provide limited support for predicting temporal exceptions. Firstly recent work (Eder, Gruber et al. 2000; Eder and Pichler 2002; Yu, Xie et al. 2013) conducted exception prediction at a specific time point such as at the beginning or end of a task. Secondly existing work (Eder, Gruber et al. 2000; Eder and Pichler 2002; Leong, Si et al. 2012; Yu, Xie et al. 2013) ignores some situations in parallel control structures. Specifically, during prediction these algorithms choose a particular task of one of the branches of the AND gateway and make conclusions based on the result of that prediction. Therefore, simultaneous predictions at several locations on different branches can produce inconsistent conclusions. In our approach, a single unified conclusion for prediction at a specific time point is drawn based on the exception probability of the given workflow instance while taking into account all AND/XOR branches. Thirdly these algorithms only provide limited support for iterative control structures which is used to control the repetitive execution of one or more tasks. In this paper, we introduce a run-based prediction algorithm to overcome these problems.

1.2. Contributions

The main contributions of our run-based exception prediction algorithm are as follows:

(1) The proposed algorithm can predict exceptions in workflows involving parallel control structures while taking into account all parallel branches.
(2) It can not only predict exceptions at any time point but also can predict exceptions for workflows that consist of crossing loops and nested loops.
(3) Our algorithm can be used with additional parameters in exception prediction, for instance, the number of iterations for loops and the threshold for exceptions according to users' preferences.
(4) The proposed algorithm enables finer control of prediction processes since users are allowed to set thresholds in making predictions. Specifically, the proposed run-based exception prediction algorithm not only considers the choice probabilities in selective control structures, but also the exception probability of an affected-run, and exception probability of an instance.

2. Literature review

In (Eder, Gruber et al. 2000), researchers proposed an algorithm for handling alternative execution paths by augmenting each task node with temporal information. In (Eder, Gruber et al. 2000), two explicit time constraints are defined; they are lower (lbc) and upper (ubc) bound constraint, which are used to define the minimum and maximum time distance between source event and destination event. Based on these two time constraints, the algorithm computes the earliest point in time a task node can finish when the shortest (longest) path from start node to the task node is taken, called best case E-value (worst case E-value); and they compute the latest point in time a task node has to finish in order to meet the overall deadline when the shortest (longest) path is taken, called best case L-value (worst case L-value). The exception prediction is conducted by comparing the lbc and ubc to E-value and L-value to check whether the time constraints are violated or not. When a workflow is executing, at a given time point that a task ends its execution, if the current time is smaller than or equal to the best-case E-value of the task, it will predict no exception happened, otherwise exception happened is predicted; if the current time is greater than or equal to best case L-value of the task, it will predict an exception happened, otherwise no exception happened is predicted.

In (Eder and Pichler 2002), the researchers introduced an approach for enhancing the estimation of duration of workflow and the likelihood of time constraints violations by constructing a new structure called duration histogram, which is used to present temporal information and different probability values for different branches at XOR-split nodes. A duration histogram of a node is a (n x 2)-matrix, where each row holds one tuple (p, d), representing the probability p of the remaining execution time within d between this node and the end node. In this paper, we improve the workflow graph by attaching the duration histogram to every node; the exception prediction is conducted by checking the remaining execution time in the duration histogram for the current task node.

In (Leong, Si et al. 2012), the researchers introduce a critical path based approach for predicting the occurrence of temporal exceptions in workflows. In this approach, firstly it generates all possible execution paths at design time, and secondly it calculates the head deadline (HD) from the longest path and the earliest completion time (CET) from the shortest path according to the execution status during run time, and finally it compares CET to HD to make the prediction: if the CET is greater than HD, it will predict an exception happened, otherwise no exception happened.

In (Yu, Xie et al. 2013), the researchers propose an algorithm based on historical temporal data for handling workflow time exceptions (deadline constraint violations). In this algorithm, they introduce a set of time cumulative distribution functions, such as execution time, queuing time, and total processing time etc. In their approach, a time probability model of the process is used to model the time indeterminacy of the execution time of tasks. By analyzing the time probability model and the workload of the available resources, the algorithm predicts the temporal exceptions of the workflow.

The related work stated above have some limitations in exception prediction. First, exceptions cannot be predicted at any time point (Eder, Gruber et al. 2000; Eder and Pichler 2002; Yu, Xie et al. 2013). For instance, prediction must be performed at the beginning or end of a task. Second, no crossing and nested loops are supported (Eder, Gruber et al. 2000; Eder and Pichler 2002; Yu, Xie et al. 2013). Although it is supported in (Leong, Si et al. 2012), it has limitations in flexibility, as it predefines a maximum iterative number for each loop. In our approach, we not only consider crossing and nested loops, but also allow users to define the probability of iteration. Third, (Eder, Gruber et al. 2000; Eder and Pichler 2002; Leong, Si et al. 2012; Yu, Xie et al. 2013) use temporal information of a chosen node to make the exception prediction. Since they only focus on one specific node, simultaneous prediction on different nodes can cause inconsistent conclusions. For instance, simultaneous prediction at different nodes in AND branches can produce inconclusive results. Therefore, in this paper, we present an algorithm that takes into account all possible execution paths of the workflow instance for making a unified conclusion. Fourth, (Eder, Gruber et al. 2000; Leong, Si et al. 2012) provide a direct yes/no answer for exception prediction without considering probabilities. Therefore, users have no information about the likelihood of the occurrence of exceptions. Although (Eder and Pichler 2002; Yu, Xie et al. 2013) provide an approach which uses probabilities for exception prediction, these values are defined at design time and they are not updated during run time. Note that some execution paths defined at design time may not map into any of the execution paths during actual execution. In these situations, the accuracy of the prediction could be affected. In this paper, we propose an algorithm which calculates the exception probability of an instance at run time. Table 1 shows the comparison of the above four papers to our run-based exception prediction algorithm, where S, PS, and NS denote "Supported", "Partially-Supported", and "Not-Supported".

3. Background

In this section we define the fundamental elements of the workflow used in the rest of this paper.

Table 1
Comparison of the proposed run-based exception prediction algorithm with existing work.

	(Eder, Gruber et al. 2000)	(Eder and Pichler 2002)	(Leong, Si et al. 2012)	(Yu, Xie et al. 2013)	Run-based algorithm
Sequential routing	S	S	S	S	S
Selective routing	S	S	S	S	S
Parallel routing	PS	PS	PS	PS	S
Structured loop	NS	PS	PS	PS	S
Nested loop	NS	NS	PS	NS	S
Crossing loop	NS	NS	PS	NS	S
Branch probability	NS	PS	NS	PS	S
Overtime probability	NS	NS	NS	NS	S

∗ S = Supported, PS = Partially-Supported, NS = Not-Supported

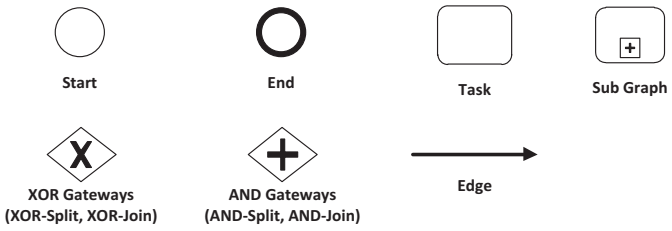

Fig. 1. BPMN workflow notations.

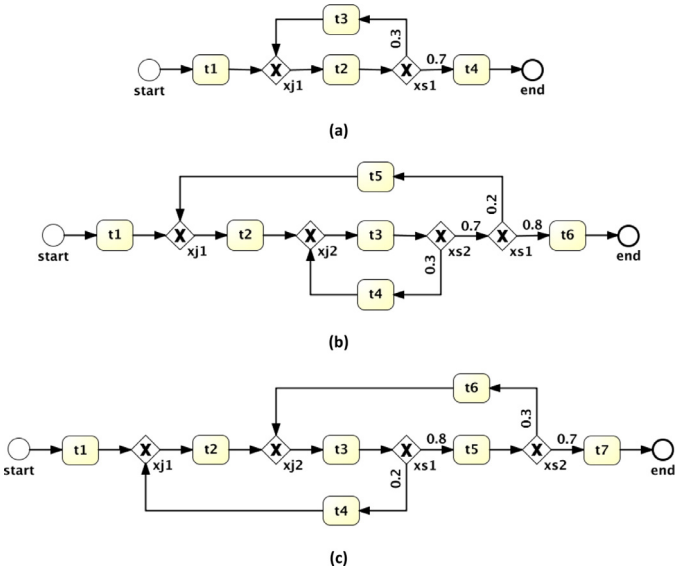

Fig. 2. Three types of loop: (a) structured loop (b) nested loop (c) crossing loop.

3.1. Workflow and iterative control structure

A workflow is defined as a network of tasks with rules that determine the sequence of tasks that should be carried out by resources in order to achieve business goals. A resource is a generic name for a person, a machine, or an application program. A workflow schema/model is a description of a workflow (Son, Kim et al. 2005). A process (workflow) instance represents an instance of a workflow that is being executed.

To visualize a workflow schema (specification), we can represent it as a directed acyclic graph, which is composed of nodes and edges (Eder, Gruber et al. 2000; Bierbaumer, Eder et al. 2005; Eder and Pichler 2005). Three types of nodes can be identified: activities, events, and gateways. Activities can be further classified into two types, atomic activities and composite activities. An atomic activity represents a naturally defined task. It is a basic unit of work that can be executed directly, and cannot be further divided. A composite activity is a workflow and can be decomposed into a number of nodes and edges. Events capture the start and end of a workflow. Gateways denote the flow of execution between activities. There are four types of gateways, namely AND-split (executing all immediate successor nodes simultaneously), AND-join (executing the immediate successor node once all immediate predecessor nodes have been executed), XOR-split (choosing only one among the immediate successor nodes for executing), and XOR-join (executing the immediate successor node once any one of its immediate predecessor nodes has been executed) (Kiepuszewski, ter Hofstede et al. 2003; van der Aalst, ter Hofstede et al. 2003). Edges define the execution sequence of nodes. The combination of gateways and edges constitute the control structures of the workflow. There are four basic control structures: sequential, parallel (AND), selective (XOR) and iterative (LOOP) control structures (Jin and Myoung 2001; Li and Yang 2005). Fig. 1 shows the notations that will be used in this paper.

An iterative control structure is a control structure that repeatedly executes one or more tasks. Therefore, if a workflow contains an iterative control structure, it is more likely that temporal constraints of a workflow will be violated since a loop can be repeated a number of times causing a significant increase in execution time. According to (van der Aalst, ter Hofstede et al. 2003; Leong, Si et al. 2012), three types of iterative control structure can be identified, namely, structured loop, nested loop and crossing loop. Examples of these control structures are shown in Fig. 2. A structured loop is a set of tasks that is continually repeated until a certain condition is reached. A nested loop is a loop that completely contains another loop and all tasks from the inner loop are also embedded in the outer loop. A crossing loop is a loop that contains part of another loop. In a crossing loop, only some of the tasks in the inner loop are embedded within the outer loop. In these control structures, the first iteration of the outer loop will trigger the inner loop, which executes until completion. When the inner loop completes, the outer loop is executed for its next iteration, triggering the inner loop again, and so on until no more iteration of the outer loop is possible.

3.2. Run

A run is a directed acyclic graph that represents one possible execution path of the workflow schema. A run consists of all or a subset of nodes and edges within a workflow schema beginning from the start node to the end node (van der Aalst, Hirnschall et al. 2002; Dumas, García-Banuelos et al. 2011). A run has three characteristics: (a) For any parallel control structure in the run, all branches of the control structure will be executed. (b) For any selective control structure in the run, only one of its branches will be executed. (c) For each

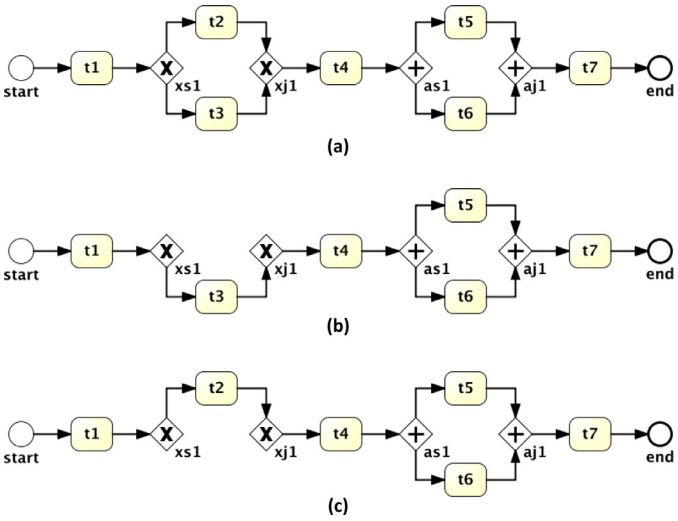

Fig. 3. Two runs (b) and (c) are constructed from the workflow in (a).

iterative control structure in the run, the number of iterations is exactly defined. In other words, two runs are different if they contain at least one different successor node for a selective control structure, or if iterative control structures in these runs are executed for a different number of iterations. Fig. 3 shows an example of two runs constructed from a workflow.

4. Run-based exception prediction for workflows

4.1. Assumptions

Firstly, we assume that workflows are well-structured. For a workflow to be well-structured means that every control structure in it can be transformed into a composite node. The workflow consists of *n* sequential nodes and each node is either an atomic node or a composite node. A composite node is a workflow without start node and end node. But, a composite node may contain parallel, conditional, or iterative control structures. In a well-structured workflow, control structures can be nested but any crossings within these structures are not allowed. However, one of the objectives of this research is to predict temporal exceptions for workflows that contain crossing loops. Thus in this paper, we allow loops to be nested or crossed. However, we assume that other control structures such as parallel and selective control structure cannot be crossed (Eder, Gruber et al. 2000; Bierbaumer, Eder et al. 2005; Xie, Yu et al. 2009). Secondly, since the main focus of this paper is about predicting temporal exceptions, we assume that there are sufficient resources to handle all the tasks, meaning that there will be no conflict in sharing the resources. Finally, we assume that each iteration of a loop is dependent; meaning that the probability of iteration of the loop occurring is gradually diminished as the number of iterations is increased no matter if it is triggered by the loop itself or the outer loop which contains it.

4.2. DEFINITIONS

In this section, we use formulas to define the terms used in the run-based exception prediction algorithm.

Definition 1. (Workflow schema): A workflow schema is a description of a workflow. It is commonly defined with a set of nodes and directed edges, beginning with a start node and finishing with an end node. Each node in the workflow denotes an activity or a gateway, and each directed edge denotes a transition relationship between any two nodes.

An example of a workflow schema is depicted in Fig. 4.

Workflow schema (specification) $W = (st_0, et_0, T, AS, AJ, XS, XJ, E)$

- st_0 is the start node representing the initial point of the workflow
- ed_0 is the end node representing the final point of the workflow
- T is a set of tasks, denoted by $T = \{t_1, t_2, \dots, t_i\}$, where t_i is one of the tasks
- AS is a set of AND-split gateways, denoted by $AS = \{as_1, as_2, \dots, as_i\}$, where as_i is one of the AND-split gateways
- AJ is a set of AND-join gateways, denoted by $AJ = \{aj_1, aj_2, \dots, aj_i\}$, where aj_i is one of the AND-join gateways
- XS is a set of XOR-split gateways, denoted by $XS = \{xs_1, xs_2, \dots, xs_i\}$,where xs_i is one of the XOR-split gateways
- XJ is a set of XOR-join gateways, denoted by $XJ = \{xj_1, xj_2, \dots, xj_i\}$,where xj_i is one of the XOR-join gateways
  - Each split node as_n (xs_m) has a unique corresponding join node aj_n (xj_m) that can form a meaningful control flow structure, where $\forall as_n \in AS, \forall aj_n \in AJ, \forall xs_m \in XS, \forall xj_m \in XJ$
- E is a set of directed edges, denoted by $E \subset V \times V = \{e_1, e_2, \dots, e_i\}$, where e_i is the flow from node element v_n to v_m ($v_n, v_m \in V$); it defines the execution order of nodes
  - V is a set of node elements which can be further divided into disjoint sets of T, AS, AJ, XS, XJ
  - The mapping function InEdge(v_i) returns the set of directed edges that goes to v_i
  - The mapping function OutEdge(v_i) returns the set of directed edges that comes from v_i

Definition 2. (Looping gateway): Looping Gateways are used to encapsulate iterative control structures. The formation of an iterative control structure can be verified if there is a path consisting of a series of flows from an XOR-join to its corresponding XOR-split gateway.

Let XS^L be the set of XOR-split gateways that can combine with their corresponding XOR-join gateways to construct an iterative control structure.

Let XJ^L be the set of XOR-join gateways that can combine with their corresponding XOR-split gateways to construct an iterative control structure.

For example, the following equation shows the looping gateway with respect to Fig. 4.

$XS^L = \{xs1, xs2\}, XJ^L = \{xj1, xj2\}$

Definition 3. (Sub-graph): In a workflow schema, an iterative control structure can be encapsulated into a node element called sub-graph, which in turn contains a workflow-graph itself and other parameters. Intuitively, a sub-graph can be derived from the iterative control structure by duplicating all node elements and directed edges between the XOR-split and XOR-join gateway.

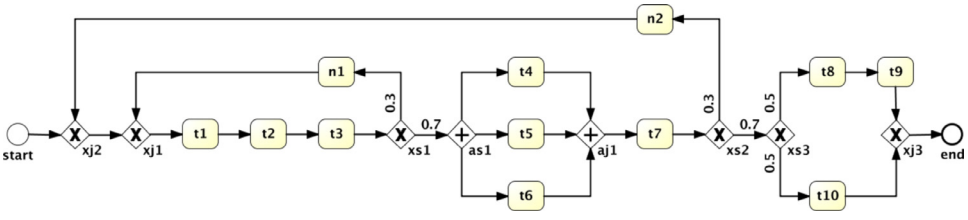

Fig. 4. A workflow schema.

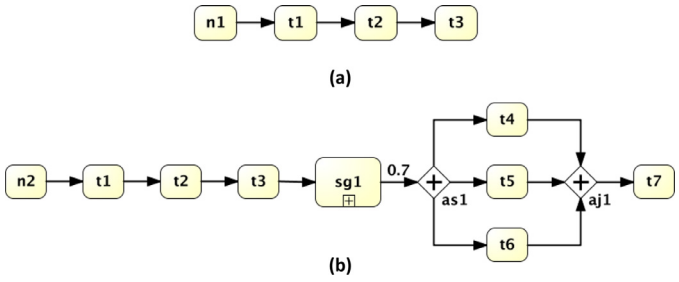

Fig. 5. Two sub-graphs with respect to Fig. 4, where (a) is the inner loop, (b) is the outer loop.

Sub-graph $sg_i = Block(xs_n, xj_n) \cup Block(xj_n, xs_n) = (T_i, AS_i, AJ_i, XS_i, XJ_i, E_i, SG_i, op_i)$

- The mapping function $Block(v_i, v_j)$ returns all unique nodes and edges on all possible paths from v_i to v_j
  $Block(v_i, v_j) = (T_k, AS_k, AJ_k, XS_k, XJ_k, E_k)$
- $T_i = T_x \cup T_y$,where $T_x \in Block(xs_n, xj_n), T_y \in Block(xj_n, xs_n)$
- $AS_i = AS_x \cup AS_y$,where $AS_x \in Block(xs_n, xj_n), AS_y \in Block(xj_n, xs_n)$
- $AJ_i = AJ_x \cup AJ_y$,where $AJ_x \in Block(xs_n, xj_n), AJ_y \in Block(xj_n, xs_n)$
- $XS_i = XS_x \cup XS_y$,where $XS_x \in Block(xs_n, xj_n), XS_y \in Block(xj_n, xs_n)$
- $XJ_i = XJ_x \cup XJ_y$,where $XJ_x \in Block(xs_n, xj_n), XJ_y \in Block(xj_n, xs_n)$
- $E_i = E_x \cup E_y$,where $E_x \in Block(xs_n, xj_n), E_y \in Block(xj_n, xs_n)$
- SG_i is a set of sub-graphs, $SG_i = SG_x \cup SG_y$,where $SG_x \in Block(xs_n, xj_n), SG_y \in Block(xj_n, xs_n)$
- op_i is the probability of occurrence for sg_i, representing the likelihood of each iteration for it
- $(xs_n, xj_n), (xj_n, xs_n) \in XS^L \times XJ^L$

For example, the following equations show two sub-graphs with respect to Fig. 4. These sub-graphs are shown in Fig. 5, where (a) is the inner loop and (b) is the outer loop, namely sg1 and sg2, respectively.

sg1 = (T_1, AS_1, AJ_1, XS_1, XJ_1, E_1, SG_1, op_1)

- T_1 = {t1, t2, t3, n1}, $AS_1 = \emptyset$, $AJ_1 = \emptyset$, $XS_1 = \emptyset$, $XJ_1 = \emptyset$, $SG_1 = \emptyset$, $op_1 = 0.3$

sg2 = (T_2, AS_2, AJ_2, XS_2, XJ_2, E_2, SG_2, op_2)

- T_2 = {t1, t2, t3, t4, t5, t6, t7}, AS_2 = {as_1}, AJ_2 = {aj_1}, $XS_2 = \emptyset$, $XJ_2 = \emptyset$, SG_2 = {sg1}, $op_2 = 0.3$

Definition 4. (Reduced-graph): A reduced graph is an acyclic directed graph derived from a workflow graph that transforms all iterative control structures to sub-graphs.

Reduced-graph $G = (st_0, ed_0, T', AS', AJ', XS', XJ', E', SG)$

- $T' = T - \cup T_k$, where $T_k \in Block(xs_n, xj_n), \forall (xs_n, xj_n) \in XS^L \times XJ^L$
  $Block(v_i, v_j)$ returns all nodes and eges between v_i and v_j
  XS^L (XJ^L) is the set of XOR-split (XOR-join) gateways for loops
- $AS' = AS - \cup AS_k$, where $AS_k \in Block(xs_n, xj_n), \forall (xs_n, xj_n) \in XS^L \times XJ^L$
- $AJ' = AJ - \cup AJ_k$, where $AJ_k \in Block(xs_n, xj_n), \forall (xs_n, xj_n) \in XS^L \times XJ^L$
- $XS' = XS - \cup XS_k - XS^L$,$\times XJ^L$where $XS_k \in Block(xs_n, xj_n), \forall (xs_n, xj_n) \in XS^L$
- $XJ' = XJ - \cup XJ_k - XJ^L$, where $XJ_k \in Block(xs_n, xj_n), \forall (xs_n, xj_n) \in XS^L \times XJ^L$
- $E' = (E - \cup E_k) \cup (\cup Edge(Source(xs_n), sg_i) \cup Edge(sg_i, Target(xs_n))$
  $E_k \in Block(xs_n, xj_n), sg_i = Block(xs_n, xj_n) \cup Block(xj_n, xs_n), \forall (xs_n, xj_n) \in XS^L \times XJ^L, \forall (xj_n, xs_n) \in XJ^L \times XS^L$
  The mapping function $Source(v_i)$ returns the set of directed predecessor nodes of v_i
  The mapping function $Target(v_i)$ returns the set of directed successor nodes of v_i
  The function $Edge(v_i, v_j)$ creates a directed edges from v_i to v_j
- SG is a set of sub-graphs, denoted by $SG = \{sg_1, sg_2, \dots, sg_i\}$

The following equations show how a reduced-graph can be constructed from the workflow schema given in Fig. 4. The reduced-graph is depicted in Fig. 6.

G = (st_0, ed_0, T′, AS′, AJ′, XS′, XJ′, E′, SG)

- st_0 = start, ed_0 = end
- T′ = {t1, t2, t3, t4, t5, t6, t7, t8, t9, t10}
- AS′ = {as1}, AJ′ = {aj1}, XS′ = {xs3}, XJ′ = {xj3}, SG = {sg1, sg2}

Definition 5. (Process instance): A process instance comprises a network of activities from the start node to the end node of a workflow, representing an actual execution of a workflow. It also contains runtime information related to the workflow, such actual start time and actual end time of a task.

Let P be a n tuple of process instances, denoted by $P = (p_1, p_2, \dots, p_n)$

- p_n is one of the process instances, it is a m tuple of tasks, denoted by $p_n = (t'_{ab}, t'_{cd}, \dots, t'_{ik})$
  where t'_{ik} is the k^{th} execution instance of task t_i in T

For a process instance p_n, it has following properties:

- The amount of time for executing p_n is denoted by $Instance\_Exc(p_n) = End(p_n) - Start(p_n)$,
  where $End(p_n)$ and $Start(p_n)$ is the end and start time of p_n respectively
- Actual execution time of task t'_{ik} in p_n is the amount of actual time for executing the task
  It is denoted by $Task\_Exe(t'_{ik}) = End\_Time(t'_{ik}) - Start\_Time(t'_{ik})$,where $\forall t'_{ik} \in p_n$
  $Start\_Time(t'_{ik})$ is the actual start time of task t'_{ik}, representing the moment when the task begins execution
  $End\_Time(t'_{ik})$ is the actual end time of task t'_{ik}, representing the moment when the task finishes execution

Definition 6. (Run): A run represents one execution path of workflow graph that consists of all nodes and directed edges between start and end node. A run can be considered as one specific type of process that contains the same set of nodes and directed edges. A run is uniquely identified by the set of nodes and/or directed edges.

Let R be a set of runs, denoted by $R = \{r_1, r_2, \dots, r_i\}$

- r_i is one of the possible execution paths of G, denoted by $r_i = (st_0, ed_0, T_i, AS_i, AJ_i, E_i, S_i)$
  st_0 is the start node of G
  ed_0 is the end node of G
  $T_i \subseteq T'$, where $T' \in G$
  $AS_i \subseteq AS'$, where $AS' \in G$
  $AJ_i \subseteq AJ'$, where $AJ' \in G$
  $E_i \subseteq E'$, where $E' \in G$
  S_i is a set of sub-runs from different sub-graphs, denoted by $S_i = \{sr_{jk}, sr_{nm} \dots\}$,where $sr_{jk} \in SR_j, sr_{nm} \in SR_n$
  SR_j is the set of sub-runs for sub-graph sg_j, sr_{jk} is one of the sub-runs in SR_j

Definition 7. (Sub-run): A sub-run represents a possible execution path of a sub-graph. However each sub-graph within a reduced-graph may iterate one or more times. Therefore we annotate each sub-run of a sub-graph with a counter which describes the exact number of iterations of the sub-graph. In other words, sub-runs of a sub-graph represent all possible execution paths which are generated as a result of each possible iteration of the sub-graph.

Let RSG_i be a set of runs of sg_i, denoted as $RSG_i = \{rsg_{i1}, rsg_{i2}, \dots rsg_{ik}\}$

- $rsg_{ij} = (T_{ij}, AS_{ij}, AJ_{ij}, E_{ij}, SG_{ij},)$
  $T_{ij} \subseteq T_i$, where $T_i \in sg_i$
  $AS_{ij} \subseteq AS_i$, where $AS_i \in sg_i$
  $AJ_{ij} \subseteq AJ_i$, where $AJ_i \in sg_i$
  $E_{ij} \subseteq E_i$, where $E_i \in sg_i$
  $SG_{ij} \subseteq SG_i$, where $SG_i \in sg_i$

Let SR_i be a set of sub-runs of sg_i, denoted by $SR_i = \{sr_{i1}, sr_{i2}, \dots sr_{in}\}$

- sr_{in} is one of the sub-runs of sg_i, denoted by $sr_{in} = (RSG_{in}, nf_{in})$
  $RSG_{in} = \{rsg_{ia}, rsg_{ib}, \dots, rsg_{ij}\}$, where $(\forall\, rsg_{ij} \in RSG_i), (|RSG_{in}| = nf_{in})$
  nf_{ij} is the number of iterative frequency

The generation of sub-runs for *sg1* and *sg2* given in Fig. 5 is described in following paragraph.

For illustration purposes we assume that each sub-graph can iterate no more than two times. First we generate the runs of the corresponding sub-graphs. In this example, RSG_1 represents the set of runs of sub-graph *sg1* and rsg_{11} is one of the runs of *sg1*. Next, we define SR_1 which is the set of sub-runs of *sg1* which contains two sub-runs sr_{11} and sr_{12}. Each sub-run is generated by repeating the path contained in the run of the sub-graph with respect to the interactive frequency. For instance, sr_{11} is generated by repeating

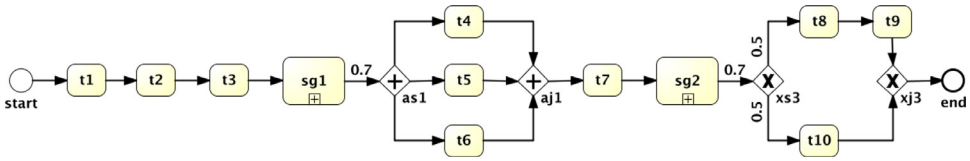

Fig. 6. Reduced-graph with respect to Fig. 4.

the path contained in the run of the sub-graph RSG_{11} for one time. Likewise, sr_{12} is generated by repeating the path contained in the run of the sub-graph RSG_{11} for two times.

Run of sub-graph and sub-run for sg1:

$RSG_1 = \{rsg_{11}\}$

- $rsg_{11} = (T_{11}, AS_{11}, AJ_{11}, XS_{11}, XJ_{11}, E_{11}, SG_{11})$
  $T_{11} = \{t1, t2, t3, n1\}, AS_1 = \emptyset, AJ_1 = \emptyset, XS_1 = \emptyset, XJ_1 = \emptyset, SG_1 = \emptyset$

$SR_1 = \{sr_{11}, sr_{12}\}$

- $sr_{11} = (RSG_{11}, nf_{11})$, where $RSG_{11} = \{rsg_{11}\}$, $nf_{11} = 1$
- $sr_{12} = (RSG_{12}, nf_{12})$, where $RSG_{12} = \{rsg_{11}, rsg_{11}\}$, $nf_{12} = 2$

Run of sub-graph and sub-run for sg2:

$RSG_2 = \{rsg_{21}\}$

- $rsg_{21} = (T_{21}, AS_{21}, AJ_{21}, XS_{21}, XJ_{21}, E_{21}, SG_{21})$
  $T_{21} = \{t1, t2, t3, t4, t5, t6, t7\}, AS_{21} = \{as_1\}, AJ_{21} = \{aj_1\}, XS_{21} = \emptyset, XJ_{21} = \emptyset, SG_{21} = \{sg1\}$

$SR_2 = \{sr_{21}, sr_{22}\}$

- $sr_{21} = (RSG_{21}, nf_{21})$, where $RSG_{21} = \{rsg_{21}\}$, $nf_{21} = 1$
- $sr_{22} = (RSG_{22}, nf_{22})$, where $RSG_{22} = \{rsg_{21}, rsg_{21}\}$, $nf_{22} = 2$

After all sub-runs are generated, runs with respect to Fig. 6 can be created. In the following equations, R is the set of runs and $r1$ is one of the runs. Moreover, $r1$ is a tuple containing $st_0, ed_0, T_1, AS_1, AJ_1, XS_1, XJ_1, E_1, \mathcal{S}_1$.

$R = \{r1, r2, r3, r4, r5, r6, r7, r8, r9\}$

- $r_1 = (st_0, ed_0, T_1, AS_1, AJ_1, XS_1, XJ_1, E_1, \mathcal{S}_1)$
  $T_1 = \{t1, t2, t3, t4, t5, t6, t7, t8, t9\}, AS_1 = \{as1\}, AJ_1 = \{aj1\}, XS_1 = \{xs3\}, XJ_1 = \{xj3\}, \mathcal{S}_1 = \emptyset$
- $r_2 = (st_0, ed_0, T_2, AS_2, AJ_2, XS_2, XJ_2, E_2, \mathcal{S}_2)$
  $T_2 = \{t1, t2, t3, t4, t5, t6, t7, t10\}, AS_2 = \{as1\}, AJ_2 = \{aj1\}, XS_2 = \{xs3\}, XJ_2 = \{xj3\}, \mathcal{S}_2 = \emptyset$
- $r_3 = (st_0, ed_0, T_3, AS_3, AJ_3, XS_3, XJ_3, E_3, \mathcal{S}_3)$
  $T_3 = \{t1, t2, t3, t4, t5, t6, t7, t8, t9\}, AS_3 = \{as1\}, AJ_3 = \{aj1\}, XS_3 = \{xs3\}, XJ_3 = \{xj3\}, \mathcal{S}_3 = \{sr_{11}\}$
- $r_4 = (st_0, ed_0, T_4, AS_4, AJ_4, XS_4, XJ_4, E_4, \mathcal{S}_4)$
  $T_4 = \{t1, t2, t3, t4, t5, t6, t7, t10\}, AS_4 = \{as1\}, AJ_4 = \{aj1\}, XS_4 = \{xs3\}, XJ_4 = \{xj3\}, \mathcal{S}_4 = \{sr_{11}\}$
- $r_5 = (st_0, ed_0, T_5, AS_5, AJ_5, XS_5, XJ_5, E_5, \mathcal{S}_5)$
  $T_5 = \{t1, t2, t3, t4, t5, t6, t7, t8, t9\}, AS_5 = \{as1\}, AJ_5 = \{aj1\}, XS_5 = \{xs3\}, XJ_5 = \{xj3\}, \mathcal{S}_5 = \{sr_{12}\}$
- $r_6 = (st_0, ed_0, T_6, AS_6, AJ_6, XS_6, XJ_6, E_6, \mathcal{S}_6)$
  $T_6 = \{t1, t2, t3, t4, t5, t6, t7, t10\}, AS_6 = \{as1\}, AJ_6 = \{aj1\}, XS_6 = \{xs3\}, XJ_6 = \{xj3\}, \mathcal{S}_6 = \{sr_{12}\}$
- $r_7 = (st_0, ed_0, T_7, AS_7, AJ_7, XS_7, XJ_7, E_7, \mathcal{S}_7)$
  $T_7 = \{t1, t2, t3, t4, t5, t6, t7, t8, t9\}, AS_7 = \{as1\}, AJ_7 = \{aj1\}, XS_7 = \{xs3\}, XJ_7 = \{xj3\}, \mathcal{S}_7 = \{sr_{21}\}$
- $r_8 = (st_0, ed_0, T_8, AS_8, AJ_8, XS_8, XJ_8, E_8, \mathcal{S}_8)$
  $T_8 = \{t1, t2, t3, t4, t5, t6, t7, t10\}, AS_8 = \{as1\}, AJ_8 = \{aj1\}, XS_8 = \{xs3\}, XJ_8 = \{xj3\}, \mathcal{S}_8 = \{sr_{21}\}$
- $r_9 = (st_0, ed_0, T_9, AS_9, AJ_9, XS_9, XJ_9, E_9, \mathcal{S}_9)$
  $T_9 = \{t1, t2, t3, t4, t5, t6, t7, t8, t9\}, AS_9 = \{as1\}, AJ_9 = \{aj1\}, XS_9 = \{xs3\}, XJ_9 = \{xj3\}, \mathcal{S}_9 = \{sr_{22}\}$
- $r_{10} = (st_0, ed_0, T_{10}, AS_{10}, AJ_{10}, XS_{10}, XJ_{10}, E_{10}, \mathcal{S}_{10})$
  $T_{10} = \{t1, t2, t3, t4, t5, t6, t7, t10\}, AS_{10} = \{as1\}, AJ_{10} = \{aj1\}, XS_{10} = \{xs3\}, XJ_{10} = \{xj3\}, \mathcal{S}_{10} = \{sr_{22}\}$
- $r_{11} = (st_0, ed_0, T_{11}, AS_{11}, AJ_{11}, XS_{11}, XJ_{11}, E_{11}, \mathcal{S}_{11})$
  $T_{11} = \{t1, t2, t3, t4, t5, t6, t7, t8, t9\}, AS_{11} = \{as1\}, AJ_{11} = \{aj1\}, XS_{11} = \{xs3\}, XJ_{11} = \{xj3\}, \mathcal{S}_{11} = \{sr_{11}, sr_{21}\}$
- $r_{12} = (st_0, ed_0, T_{12}, AS_{12}, AJ_{12}, XS_{12}, XJ_{12}, E_{12}, \mathcal{S}_{12})$
  $T_{12} = \{t1, t2, t3, t4, t5, t6, t7, t10\}, AS_{12} = \{as1\}, AJ_{12} = \{aj1\}, XS_{12} = \{xs3\}, XJ_{12} = \{xj3\}, \mathcal{S}_{12} = \{sr_{11}, sr_{21}\}$
- $r_{13} = (st_0, ed_0, T_{13}, AS_{13}, AJ_{13}, XS_{13}, XJ_{13}, E_{13}, \mathcal{S}_{13})$
  $T_{13} = \{t1, t2, t3, t4, t5, t6, t7, t8, t9\}, AS_{13} = \{as1\}, AJ_{13} = \{aj1\}, XS_{13} = \{xs3\}, XJ_{13} = \{xj3\}, \mathcal{S}_{13} = \{sr_{12}, sr_{21}\}$
- $r_{14} = (st_0, ed_0, T_{14}, AS_{14}, AJ_{14}, XS_{14}, XJ_{14}, E_{14}, \mathcal{S}_{14})$
  $T_{14} = \{t1, t2, t3, t4, t5, t6, t7, t10\}, AS_{14} = \{as1\}, AJ_{14} = \{aj1\}, XS_{14} = \{xs3\}, XJ_{14} = \{xj3\}, \mathcal{S}_{14} = \{sr_{12}, sr_{21}\}$
- $r_{15} = (st_0, ed_0, T_{15}, AS_{15}, AJ_{15}, XS_{15}, XJ_{15}, E_{15}, \mathcal{S}_{15})$
  $T_{15} = \{t1, t2, t3, t4, t5, t6, t7, t8, t9\}, AS_{15} = \{as1\}, AJ_{15} = \{aj1\}, XS_{15} = \{xs3\}, XJ_{15} = \{xj3\}, \mathcal{S}_{15} = \{sr_{12}, sr_{22}\}$
- $r_{16} = (st_0, ed_0, T_{16}, AS_{16}, AJ_{16}, XS_{16}, XJ_{16}, E_{16}, \mathcal{S}_{16})$
  $T_{16} = \{t1, t2, t3, t4, t5, t6, t7, t10\}, AS_{16} = \{as1\}, AJ_{16} = \{aj1\}, XS_{16} = \{xs3\}, XJ_{16} = \{xj3\}, \mathcal{S}_{16} = \{sr_{12}, sr_{22}\}$
- $r_{17} = (st_0, ed_0, T_{17}, AS_{17}, AJ_{17}, XS_{17}, XJ_{17}, E_{17}, \mathcal{S}_{17})$
  $T_{17} = \{t1, t2, t3, t4, t5, t6, t7, t8, t9\}, AS_{17} = \{as1\}, AJ_{17} = \{aj1\}, XS_{17} = \{xs3\}, XJ_{17} = \{xj3\}, \mathcal{S}_{17} = \{sr_{11}, sr_{22}\}$
- $r_{18} = (st_0, ed_0, T_{18}, AS_{18}, AJ_{18}, XS_{18}, XJ_{18}, E_{18}, \mathcal{S}_{18})$
  $T_{18} = \{t1, t2, t3, t4, t5, t6, t7, t10\}, AS_{18} = \{as1\}, AJ_{18} = \{aj1\}, XS_{18} = \{xs3\}, XJ_{18} = \{xj3\}, \mathcal{S}_{18} = \{sr_{11}, sr_{22}\}$

We summarize the relationships among workflow schema, reduced-graph, sub-graph, run of reduced-graph, run of sub-graph, sub-run, and run in Fig. 7.

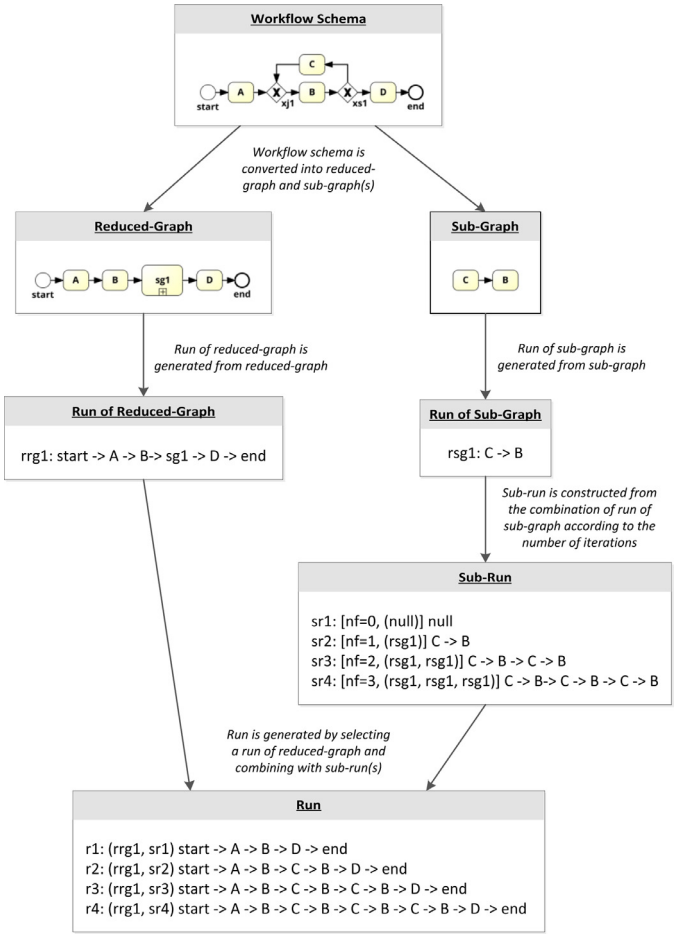

Fig. 7. The relationship among workflow schema, reduced-graph, sub-graph, run of reduced-graph, run of sub-graph, sub-run, and run.

Definition 7.1. (Cumulative looping probability): It represents how likely the iterative control structure is to iterate one or more times. A high cumulative looping probability indicates that the loop is likely to iterate more often. The probability can be defined by an administrator's estimations or average values from past executions.

Let CLP_{ij} be the cumulative looping probability of sr_{ij} for sg_i, denoted by $CLP_{ij} = op_{i_1} \times op_{i_2} \times \ldots \times op_{i_n}$

- op_i is the probability of the looping occurrence, where $op_i \in sg_i$
- $n = nf_{ij}$, it is the number of iterative frequency, where $nf_{ij} \in sr_{ij}$
- sr_{ij} is one of the sub-runs for sg_i

Definition 7.2. (Threshold for cumulative looping probability): The minimum value of cumulative looping probability required for a loop to iterate one more time. It is used to decide whether a new iteration is required for the respective iterative control structure. The threshold can be defined by the administrator's estimations. A low threshold will allow a loop to iterate more often whereas a high

threshold will prohibit a loop from further iteration.

Let β_{loop} be the threshold for cumulative looping probability

- If $CLP_i > \beta_{loop}$, the sub-run sr_{ij} will be considered to iterate once more
- If $CLP_i \leq \beta_{loop}$, the process instance p_i will be considered not to iterate once more

Definition 8. (Task allowable execution time): Each task in the workflow is assigned with a maximum allowable execution time by workflow designers. These constraints can be derived from historical records of task execution. Examples of Task Allowable Execution Time include "An insurance claim should be approved within 2 days" and "the delivery of goods should take no more than one week".

Let $Task\_Allow(t_i)$ be the allowable execution time of task t_i, it is the amount of predefined time for executing the task
In this paper, the execution time of a task is controlled by the uniform distribution that has two parameters,
which are the minimum and maximum execution time of the task. Thus, we define $Task\_Allow(t_i) = \frac{a_i + b_i}{2}$

The following definitions (from 9 to 11) are used for calculating temporal properties for a run. Since selective control structures such as XOR-splits have been taken into account when runs are generated, we will only focus on the temporal properties of AND blocks in the runs.

Definition 9. (Estimated execution time of AND block): An AND block consists of all elements between a pair of AND-split and its corresponding AND-join gateway. It contains all possible paths between the gateways and one of the possible paths is called a branch. The estimated execution time of an AND block is the execution time of the branch that has the maximum execution time among all the branches.

Let $AND\_Est(AND\_Block(as_n, aj_n))$ be the estimated execution time of AND control block

- The mapping function $AND\_Block(as_n, aj_n) = \{b_1, b_2, \dots, b_k\}$
- b_k is one of the branches between as_n and aj_n, it is 3-tuple sets, denoted by $b_k = (T_k, AS_k, AJ_k)$
  T_k is the set of tasks in the current AND control block, but not in any other of its inner AND control blocks
  $T_k = \{t_i | (\exists t_i \notin T_x) \wedge (\forall T_x \in b_j) \wedge (\forall b_j \in AND\_Block(as_m, aj_m))\}$
  AS_k, AJ_k is the set of AND-split, AND-join gateways that directly belongs to b_k respectively
- Let $Branch\_Est(b_k)$ be the estimated execution time of branch b_k
- $AND\_Est(AND\_Block(as_n, aj_n)) = Max(Branch\_Est(b_k))$, where $\forall b_k \in AND\_Block(as_n, aj_n)$
- $Branch\_Est(b_k) = \sum Task\_Allow(t_i) + \sum AND\_Est(AND\_Block(as_m, aj_m))$
  where $(\forall t_i \in T_k), (\forall (as_m, aj_m) \subseteq AS_k \times AJ_k), (T_k, AS_k, AJ_k \in b_k)$

Definition 9.1. (AND branch slack time): The difference between the estimated execution time of "AND" control block and one of its branches is called AND branch slack time.

Let $Branch\_Slack(b_k)$ be the slack time of branch b_k
$Branch\_Slack(b_k) = AND\_Est(AND\_Block(as_n, aj_n)) - Branch\_Est(b_k)$, where $b_k \in AND\_Block(as_n, aj_n)$

Definition 10. (Estimated execution time of sub-run): The estimated execution time of a sub-run is the estimated time required to complete the sub-run. It is equal to the sum of all estimated execution times of the runs of the corresponding sub-graph. The estimated execution time of the run of a sub-graph is the sum of all task allowable execution times that are not in a parallel control structure and all estimated execution times of AND blocks.

Let $Subrun\_Est(sr_{in})$ be the estimated execution time of sr_{in}

- $Subrun\_Est(sr_{in}) = \sum RunOfSubGraph\_Est(bsr_{ij})$, where $\forall bsr_{ij} \in BSR_{in}$
- $RunOfSubGraph\_Est(bsr_{ij}) = \sum Task\_Allow(t_k) + \sum AND\_Est(AND\_Block(as_n, aj_n))$
  where $(\forall t_k \in \Gamma_{subrun}), (\forall (as_n, aj_n) \subseteq AS_{ij} \times AJ_{ij}))$
- Γ_{subrun} is the set of tasks in sr_i, but not in any other of its inner AND control blocks
  $\Gamma_{subrun} = \{t_k | (\exists t_k \notin T_x) \wedge (\forall T_x \in b_j) \wedge (\forall b_j \in AND\_Block(as_n, aj_n))\}$

Definition 11. (Estimated execution time of run): The estimated execution time of run is the estimated time required to complete the run. It is equal to the sum of all task allowable execution times that is not in a parallel control structure and all estimated execution times of AND blocks.

Let $Run\_Est(r_i)$ be the estimated execution time of r_i

- $Run\_Est(r_i) = \sum Task\_Allow(t_k) + \sum AND\_Est(AND\_Block(a_n, aj_n)) + \sum Subrun\_Est(sr_{ij})$
  where $(\forall t_k \in \Gamma_{run}), (\forall (as_n, aj_n) \subseteq AS_i \times AJ_i)), (\forall sr_{ij} \in S_i)$
- Γ_{run} is the set of tasks in r_i, but not in any other of its inner AND control blocks
  $\Gamma_{run} = \{t_k | (\exists t_k \notin T_x) \wedge (\forall T_x \in b_j) \wedge (\forall b_j \in AND\_Block(as_n, aj_n))\}$

Definition 12. (Exceeded Execution Time of Task): The exceeded execution time of a task is the difference between the task execution time and the task allowable execution time.

Let $Task\_Exc(t'_{ik})$ be the exceeded execution time of task t'_{ik}
$Task\_Exc(t'_{ik}) = Task\_Exe(t'_{ik}) - Task\_Allow(t_i)$, where $t'_{ik} \in p_n$, t'_{ik} is the k^{th} execution instance of task t_i in T

Definition 13. (Exceeded execution time of AND block): The exceeded execution time of an AND block is the sum of the Exceeded Execution Times of all tasks of the longest branch within the AND block.

Let $AND\_Exc(AND\_Block(as_n, aj_n))$ be the exceeded execution time of an AND control block

- Let $Branch\_Exc(b_k)$ be the exceeded execution time of branch b_k
- $AND\_Esc(AND\_Block(as_n, aj_n)) = Max(Branch\_Esc(b_k))$, where $\forall b_k \in AND\_Block(as_n, aj_n)$
- $Branch\_Esc(b_k) = \begin{cases} 0 & , if\ \lambda \leq 0 \\ \lambda & , otherwise \end{cases}$
  where $\lambda = \sum Task\_Exc(t_i) + \sum AND\_Exc(AND\_Block(as_n, aj_n)) - Branch\_Slack(b_k)$
  where $(\forall t_i \in T_k), (\forall (as_m, aj_m) \subseteq AS_k \times AJ_k), (T_k, AS_k, AJ_k \in b_k)$

Definition 14. (Exceeded execution time of sub-run): The **exceeded** execution time of sub-run is calculated by summing up the exceeded execution time of all runs of a sub-graph. The exceeded execution time of the run of a sub-graph is the sum of all exceeded execution times of all the tasks that are not in parallel control structures and all exceeded execution times of AND blocks.

Let $Subrun\_Exc(sr_i)$ be the exceeded execution time of sr_i

- $Subrun\_Exc(sr_{in}) = \sum RunOfSubGraph\_Exc(bsr_{ij})$, where $\forall bsr_{ij} \in BSR_{in}$
- $RunOfSubGraph\_Exc(bsr_{ij}) = \sum Task\_Exc(t_k) + \sum AND\_Exc(AND\_Block(as_n, aj_n))$
  where $(\forall t_k \in \Gamma_{subrun}), (\forall (as_n, aj_n) \subseteq AS_{ij} \times AJ_{ij}))$
- Γ_{subrun} is the set of tasks in sr_i, but not in any other of its inner AND control blocks
  $\Gamma_{subrun} = \{t_k | (\exists t_k \notin T_x) \wedge (\forall T_x \in b_j) \wedge (\forall b_j \in AND\_Block(as_n, aj_n))\}$

Definition 15. (Exceeded execution time of run): The exceeded execution time of a run is the sum of exceeded execution times of all the tasks that are not in parallel control structures, and all exceeded execution times of AND blocks, and all exceeded execution times of sub-runs.

Let $Run\_Exc(r_i)$ be the exceeded execution time of r_i

- $Run\_Exc(r_i) = \sum Task\_Exc(t_k) + \sum AND\_Exc(AND\_Block(as_n, aj_n)) + \sum Subrun\_Exc(sr_{ij})$
  where $(\forall t_k \in \Gamma_{run}), (\forall (as_n, aj_n) \subseteq AS_i \times AJ_i)), (\forall sr_{ij} \in S_i)$
- Γ_{run} is the set of tasks in r_i, but not in any other of its inner AND control blocks
  $\Gamma_{run} = \{t_k | (\exists t_k \notin T_x) \wedge (\forall T_x \in b_j) \wedge (\forall b_j \in AND\_Block(as_n, aj_n))\}$

Definition 16. (Affected-runs): **Affected**-runs are the runs which are mapped into the process instance after the prediction point (a time point where the exception prediction is performed) is identified.

Let AR_i be the set of affected-runs that process instance p_i will map into under the prediction point θ
It is denoted by $AR_i = \{r_a, r_b, \dots, r_k\}$, where $(r_k \in R) \wedge (AR_i \subseteq R)$

Definition 17. (Deadline constraint of workflow): It is the hard deadline of the workflow. In this paper, the deadline is obtained by averaging the execution time of process instances from the history logs.

Let dl be the deadline constraint of the workflow
- If $Instance\_Exc(p_i) > dl$, the process instance p_i violates the deadline constraint
- If $Instance\_Exc(p_i) < dl$, the process instance p_i doesn't violate the deadline constraint

Definition 18. (Exception probability of an affected-run): It is the ratio of Run overtime (the difference between execution time of the run and deadline) to deadline constraint. It represents the likelihood of having an exception for the run. A high exception probability of an affected-run indicates that the run is likely to have an exception in the future.

Let REP_i be the exception probability of an affected-run for run r_i, denoted by $REP_i = \frac{ot_i}{dl}$, where $r_i \in AR$
- ot_i is the run overtime of run r_i, denoted by $ot_i = Run\_Exc(r_i) + Run\_Est(r_i) - dl$

Definition 18.1. (Threshold for exception probability of an affected-run): An Affected-Run is considered to have an exception if its exception probability exceeds the threshold.

Let β_{run} be the threshold for exception probability of an affected-run
- If $REP_i > \beta_{run}$, the run r_i will be considered having an exception when it is mapped by the process instance
- If $REP_i \leq \beta_{run}$, the run r_i will be considered having no exception when it is mapped by the process instance
- $r_i \in AR$

Definition 19. (Choice probability): Each directed edge after an XOR-split gateway contains a probability value that represents the likelihood of being selected. These probabilities can be defined by an administrator based on the logs of previous executions.

Let $Choice\_Prob(e_j)$ be the probability of a directed edge e_j, representing the likelihood of selecting the edge

Definition 20. (Mapping probability of sub-run): It is the likelihood of the sub-run being mapped by the sub-graph.

Let $SubrunProb(sr_{ik})$ be the mapping probability of sub-run sr_{ik}
$Subrun\_Prob(sr_{ik}) = \prod Choice\_Prob(e_n) \times CLP_{nm}$, where $(\forall e_n \in E_i), (E_i \in sr_{ik})$

Definition 21. (Mapping probability of run): It is the likelihood of the run being mapped by the process instance.

Let $Run\_Prob(r_i)$ represent the likelihood of the run being mapped to the process instance
$Run\_Prob(r_i) = \prod Choice\_Prob(e_k) \times \prod Subrun\_Prob(sr_{nm})$
where $(\forall e_n \in E_i), (\forall sr_{nm} \in S_i), (E_i, S_i \in r_i)$

Definition 22. (Exception probability of an instance): It is the ratio of the sum of the probabilities of the affected-runs which have exceeded the threshold to the sum of the probabilities of all the affected-runs. A high exception probability of an instance indicates that the process instance is likely to have exceptions in the future.

Let IEP_i be the exception probability of an instance for process instance p_i, denoted by $p_i = \frac{\sum Run\_Prob(r_n)}{\sum Run\_Prob(r_m)}$
- r_n are runs where $REP_n > \beta_{run}$ $(r_n \in AR)$
- r_m are all runs in AR

Definition 22.1. (Threshold for exception probability of an instance): A process instance which has the probability higher than the threshold will be flagged for exception.

Let $\beta_{instance}$ be the threshold for exception probability of an instance
- If $IEP_i > \beta_{instance}$, the process instance p_i will be considered having an exception
- If $IEP_i \leq \beta_{instance}$, the process instance p_i will be considered having no exception

4.3. Overview of the algorithm

The run-based exception prediction for workflows is divided into two phases, design time and run time, as shown in Fig. 8. At design time, all possible runs are generated and calculated, which will be used for prediction at run time. The detailed steps for exception prediction are given in Fig. 8.

At design-time, four steps are performed. Firstly, a user specifies the task allowable execution time, choice probability, and sets the cumulative looping probability threshold for the directed acyclic graph of workflow schema. Secondly, we convert the workflow graph into a reduced-graph by transforming all iterative control structures into sub-graphs. Thirdly, all runs and sub-runs from reduced-graph and sub-graphs are generated. Fourthly, we calculate the estimated execution time of runs from the task allowable execution time, and compute the mapping probability of runs from choice probability and cumulative looping probability.

At run-time, four steps are performed. Firstly, the algorithm generates the affected runs from the current process instance with respect to the prediction point. Secondly, we calculate the exceeded execution time of runs for each affected-run by comparing the current actual execution time to the estimated execution time. Thirdly, the algorithm calculates the exception probability of each affected-run by comparing the run overtime to the deadline constraints. Fourthly, the algorithm calculates the exception probability of the process instance. For all affected-runs, if the exception probability of an affected-run computed from the previous step is larger than the threshold for exception probability of an affected-run, the mapping probability of the affected-run will be added to the exception probability of this instance. Finally, if the exception probability of the instance is larger than the threshold, the workflow is predicted to have an exception in the future. Otherwise there will be no exception.

Six probability estimations and three thresholds used in the proposed exception prediction algorithm are summarized in Table 2.

Among six probability estimations from the table, only two of them (1 and 3) are either input by the Administrator or averaged from past executions. The rest of the estimations are calculated by the algorithm. All the thresholds are input by the Administrator.

4.4. Algorithms

There are six main modules in our run-based exception prediction program. The module diagram and the overall relationship among them are shown in Fig. 9. All these modules are implemented in Java. The BPMN Parser is used for interpreting the workflow schema. The MXML Parser is used for reading the process instances from the log files. The Reduced-Graph Generator is used to convert the workflow schema into a reduced-graph and a set of sub-graphs. The Run Generator is used for generating a set of runs. The Process Instance Analyzer is used to get the running status for a process instance, such as which tasks are executed/executing, and how much time is used. The Exception Predictor is used for conducting the exception prediction.

4.5. Example

In this section, we provide two examples to illustrate our run-based exception prediction algorithm. The first example shows how our run-based exception prediction algorithm can be used to predict

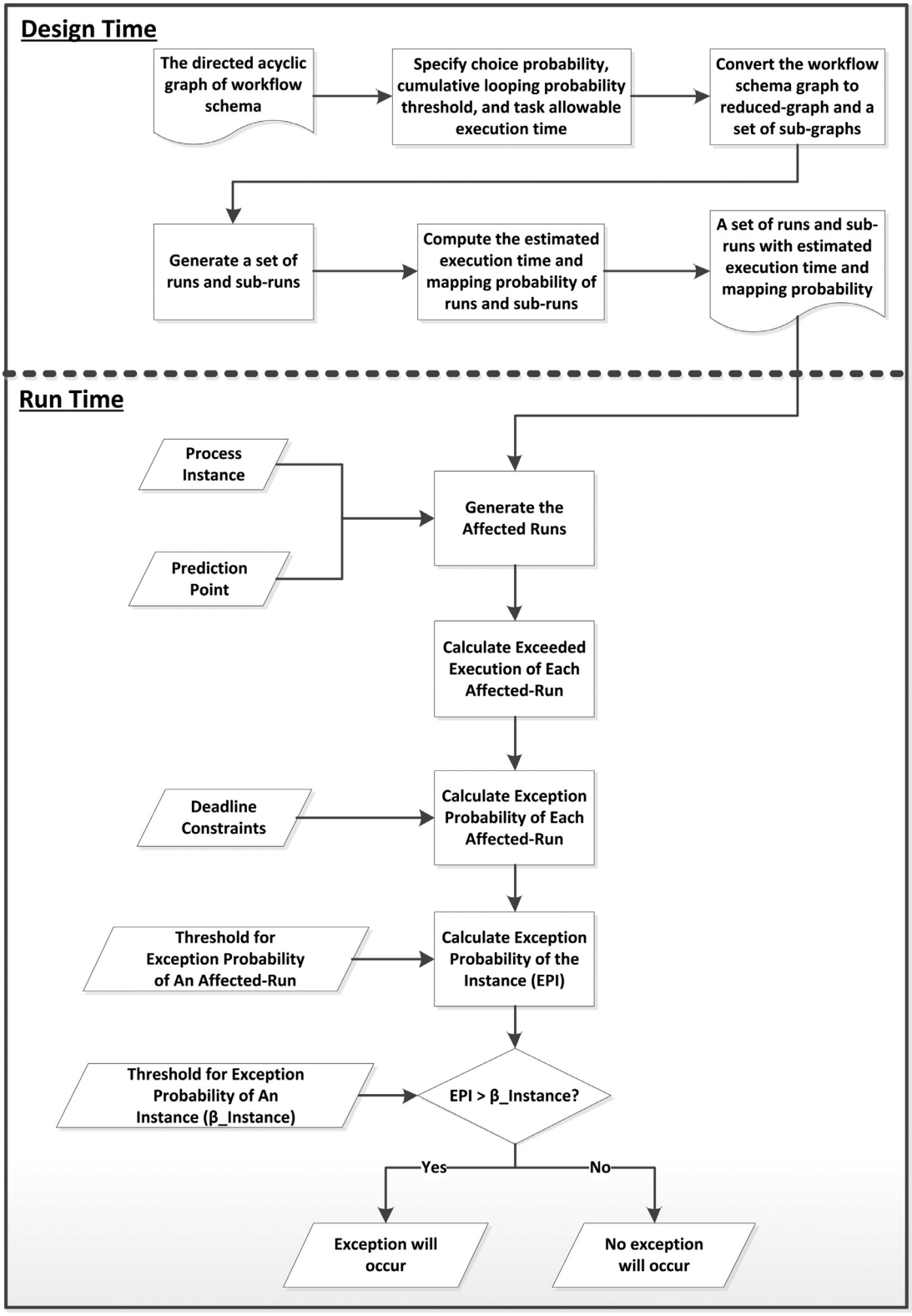

Fig. 8. Steps of Predicting Temporal Exceptions.

exceptions for a workflow that contains a nested loop. The second example shows how to deal with a workflow that includes a crossing loop.

Example 1

The workflow schema given in Fig. 10 contains one nested loop. Table 3 shows the tasks allowable execution times which are calculated from the uniform distribution of the two parameters (minimum a, and maximum b). The cumulative looping probability threshold, run exception probability threshold, instance exception probability threshold, and prediction point are set to 0.01, 0.1, 0.3, and 0.5, respectively.

First, the workflow schema is transformed into sub-graphs and reduced-graphs. There are two loops that need to be converted, shown in Fig. 11. The sub-graphs for the inner and outer loops are

Table 2
Probability estimations and thresholds used in the algorithm.

	Probability estimation	Source	Corresponding threshold	Source
1.	Cumulative looping probability (Definition 7.1)	Administrator's estimation or average values from past executions.	Threshold for cumulative looping probability (Definition 7.2)	Administrator's estimation
2.	Exception probability of an affected-run (Definition 18)	The ratio of run overtime to deadline. Calculated by the proposed algorithm.	Threshold for exception probability of an affected-run (Definition 18.1)	Administrator's estimation
3.	Choice probability (Definition 19)	Administrator's estimation or average values from past executions.		
4.	Mapping probability of sub-run (Definition 20)	Calculated from choice probability and cumulative looping probability		
5.	Mapping probability of run (Definition 21)	Calculated from choice probability and mapping probability of sub-run		
6.	Exception probability of an instance (Definition 22)	Calculated from the mapping probability of run	Threshold for exception probability of an instance (Definition 22.1)	Administrator's estimation

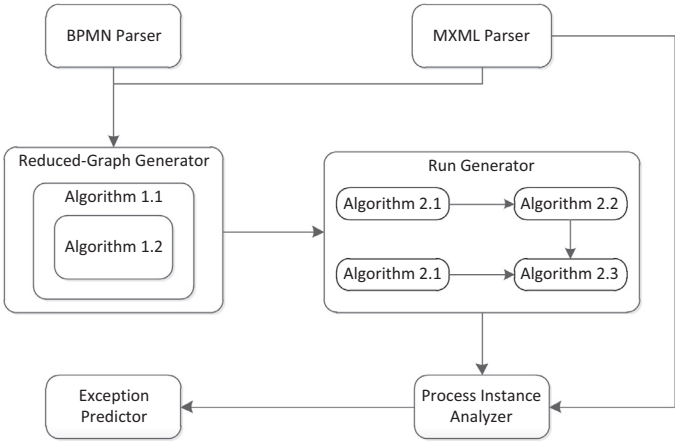

Fig. 9. Six main modules used in our run-based exception prediction program.

Table 3
Tasks allowable execution times for Example 1.

Task	a (min)	b (max)	Task	a (min)	b (max)
A	5000	10000	G	5000	10000
B	5000	10000	H	5000	10000
C	5000	10000	I	5000	10000
D	5000	10000	N1	0	0
E	5000	10000	N2	0	0
F	5000	10000			

called sg1 in Fig. 11 (a), and sg2 in Fig. 11 (b) respectively. After this conversion, a reduced-graph shown in Fig. 11 (c) is created.

Next, the algorithm generates the following run from the reduced-graph. This step is depicted in the left branch of the flow chart given in Fig. 7.

id —— (d, p) { start → …→ end }
rrg1 ---- (52500, 0.4900) { start -> A -> B -> sg1 -> C -> D -> E -> sg2 -> H -> I -> end }

Next, following runs of sub-graph and sub-runs can be generated from the sub-graph 1 (sg1) according to the right branch of the flow chart given in Fig. 7. Each sub-run is presented as "id [nf=x, (sg.rsg…)] —- (d, p)", where id is the unique key, nf=x represents the number of iterative frequency for the sub-run, (sg.rsg…) represents that the id-th sub-run is constructed from the combination of the run(s) of the sub-graph sg.rsg, d is the estimated execution time, p is the probability.

Run of sub-graph List
id —— (d, p) { start → …→ end }
sg1.rsg1 ---- (15000, 0.3000) { n1 -> A -> B }
Sub-Run List
id [nf=x, (sg.rsg…)] —— (d, p)
sg1.sr1 [nf=1, (sg1.rsg1)] ----- (15000, 0.3000)
sg1.sr2 [nf=2, (sg1.rsg1, sg1.rsg1)] ----- (30000, 0.0900)
sg1.sr3 [nf=3, (sg1.rsg1, sg1.rsg1, sg1.rsg1)] ----- (45000, 0.0270)
sg1.sr4 [nf=0, (null)] ----- (0, 1.0000)

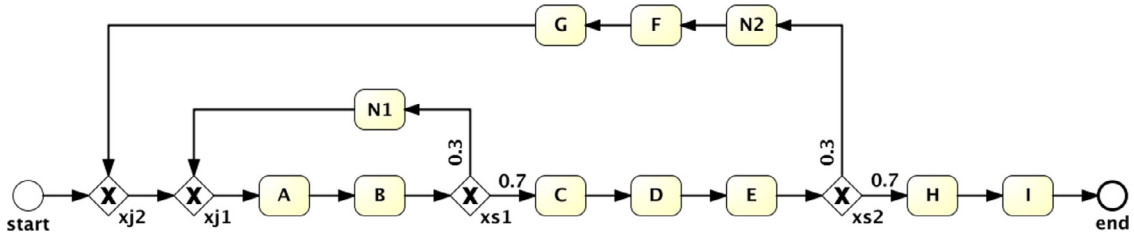

Fig. 10. Workflow schema that contains one nested loop.

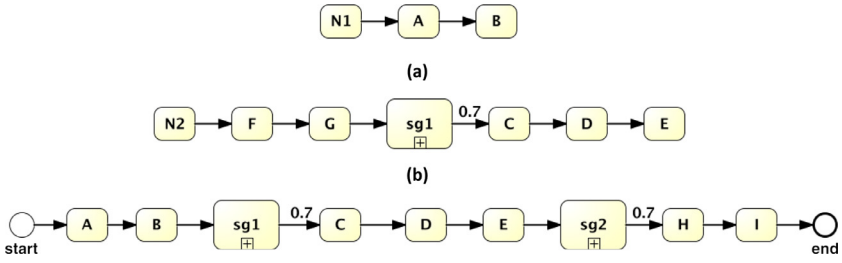

Fig. 11. (a) Sub-graph for the inner loop (sg1) (b) Sub-graph for outer loop (sg2) (c) Reduced-graph for the workflow schema from Fig. 10.

Table 4
Summary of Four predictions status.

	Process instance	Deadline	Prediction point	Executed tasks	Executing tasks
1st prediction	p1	83938	41969	$A_1, B_1, N1_1, A_2, B_2, C_1$	D
2nd prediction	p2	83938	41969	$A_1, B_1, N1_1, A_2, B_2, N1_2, A_3$	B
3rd prediction	p3	83938	41969	$A_1, B_1, C_1, D_1, E_1, N2_1$	F
4th prediction	p4	83938	41969	$A_1, B_1, C_1, D_1, E_1, N2_1, F_1$	G

Likewise, runs of sub-graph and sub-runs can be generated from the sub-graph 2 (sg2) as follows:

Run of sub-graph List

id — (d, p) { start → ···→ end }

sg2.rsg1 ---- (52500, 0.2100) { n2 -> F -> G -> A -> B -> sg1 -> C -> D -> E }

Sub-Run List

id [nf=x, (sg.rsg…)] — (d, p)

sg2.sr1 [nf=1, (sg2.rsg1)] ----- (52500, 0.2100)

sg2.sr2 [nf=2, (sg2.rsg1, sg2.rsg1)] ----- (105000, 0.0441)

sg2.sr3 [nf=3, (sg2.rsg1, sg2.rsg1, sg2.rsg1)] ----- (157500, 0.0093)

sg2.sr4 [nf=0, (null)] ----- (0, 1.0000)

Finally, following 16 runs can be generated for the workflow schema from Fig. 10.

id (rrg, sg.sr...) — (d, p) { start -> … -> end }

r1 (rrg1, sg1.sr1, sg2.sr1) ----- (120000, 0.0309) { start -> A -> B -> sg1 -> C -> D -> E -> sg2 -> H -> I -> end }

r2 (rrg1, sg1.sr2, sg2.sr1) ----- (135000, 0.0093) { start -> A -> B -> sg1 -> C -> D -> E -> sg2 -> H -> I -> end }

r3 (rrg1, sg1.sr3, sg2.sr1) ----- (150000, 0.0028) { start -> A -> B -> sg1 -> C -> D -> E -> sg2 -> H -> I -> end }

r4 (rrg1, sg1.sr4, sg2.sr1) ----- (105000, 0.1029) { start -> A -> B -> sg1 -> C -> D -> E -> sg2 -> H -> I -> end }

r5 (rrg1, sg1.sr1, sg2.sr2) ----- (172500, 0.0065) { start -> A -> B -> sg1 -> C -> D -> E -> sg2 -> H -> I -> end }

r6 (rrg1, sg1.sr2, sg2.sr2) ----- (187500, 0.0019) { start -> A -> B -> sg1 -> C -> D -> E -> sg2 -> H -> I -> end }

r7 (rrg1, sg1.sr3, sg2.sr2) ----- (202500, 0.0006) { start -> A -> B -> sg1 -> C -> D -> E -> sg2 -> H -> I -> end }

r8 (rrg1, sg1.sr4, sg2.sr2) ----- (157500, 0.0216) { start -> A -> B -> sg1 -> C -> D -> E -> sg2 -> H -> I -> end }

r9 (rrg1, sg1.sr1, sg2.sr3) ----- (225000, 0.0014) { start -> A -> B -> sg1 -> C -> D -> E -> sg2 -> H -> I -> end }

r10 (rrg1, sg1.sr2, sg2.sr3) ----- (240000, 0.0004) { start -> A -> B -> sg1 -> C -> D -> E -> sg2 -> H -> I -> end }

r11 (rrg1, sg1.sr3, sg2.sr3) ----- (255000, 0.0001) { start -> A -> B -> sg1 -> C -> D -> E -> sg2 -> H -> I -> end }

r12 (rrg1, sg1.sr4, sg2.sr3) ----- (210000, 0.0045) { start -> A -> B -> sg1 -> C -> D -> E -> sg2 -> H -> I -> end }

r13 (rrg1, sg1.sr1, sg2.sr4) ----- (67500, 0.1470) { start -> A -> B -> sg1 -> C -> D -> E -> sg2 -> H -> I -> end }

r14 (rrg1, sg1.sr2, sg2.sr4) ----- (82500, 0.0441) { start -> A -> B -> sg1 -> C -> D -> E -> sg2 -> H -> I -> end }

r15 (rrg1, sg1.sr3, sg2.sr4) ----- (97500, 0.0132) { start -> A -> B -> sg1 -> C -> D -> E -> sg2 -> H -> I -> end }

r16 (rrg1, sg1.sr4, sg2.sr4) ----- (52500, 0.4900) { start -> A -> B -> sg1 -> C -> D -> E -> sg2 -> H -> I -> end }

After all runs are generated, the exception prediction process can begin. We use following four process instances extracted from the log file for illustration. Each process instance is presented as "id (pd) —- $node_k$=nd, …, node=nd", where "id" is the unique identifier, "pd" is the actual execution time of the process instance, and $node_k$=nd represents the actual execution time "nd" of the "$node_K$" where the subscript k denotes the k^{th} execution instance of the "node" in the process instance.

id (pd) — $node_k$=nd, …, $node_k$=nd

p1 (66848) ---- $start_1$=0, A_1=5232, B_1=8435, $N1_1$=0, A_2=9831, B_2=7837, C_1=9286, D_1=6406, E_1=7057, H_1=5272, I_1=7492, end_1=0

p2 (83623) ---- $start_1$=0, A_1=9941, B_1=7379, $N1_1$=0, A_2=5747, B_2=6533, $N1_2$=0, A_3=7484, B_3=7661, C_1=9441, D_1=5236, E_1=9985, H_1=7717, I_1=6499, end_1=0

p3 (213573) ---- $start_1$=0, A_1=7697, B_1=9246, C_1=9217, D_1=7208, E_1=5064, $N2_1$=0, F_1=7741, G_1=9184, A_2=8884, B_2=5110, $N1_1$=0, A_3=5648, B_3=5297, $N1_2$=0, A_4=5653, B_4=7466, $N1_3$=0, A_5=7042, B_5=7723, $N1_4$=0, A_6=6193, B_6=5414, C_2=7055, D_2=6732, E_2=9298, $N2_2$=0, F_2=9695, G_2=7251, A_7=8260, B_7=8437, C_3=6259, D_3=6748, E_3=7827, H_1=6329, I_1=9895, end_1=0

p4 (97225) ---- $start_1$=0, A_1=6336, B_1=5630, C_1=5428, D_1=6189, E_1=5791, $N2_1$=0, F_1=6672, G_1=7360, A_2=8321, B_2=6724, C_2=8203, D_2=9671, E_2=5666, H_1=6788, I_1=8446, end_1=0

For this example, we set the deadline of the workflow at 83938 (time units) and prediction point at 41969 for the above four process instances. From these prediction points, executed tasks and currently executing tasks can be identified. The summary of the four prediction points is shown in Table 4.

Based on the information given in Table 4, the exceeded execution time of tasks can be calculated. The exceeded execution time of a task is the difference between the task execution time and the task allowable execution time. The result of the calculation is shown in Table 5. Next, we identify the affected runs based on Definition 16 and they are r1, r2, r3, r5, r6, r7, r9, r10, r11, r13, r14, r15. Affected-runs are the runs which are mapped into the four process instances after the prediction point. Then, the exceeded execution times of these runs can be computed. The exceeded execution time of a run is the sum of the exceeded execution times of all the tasks that are not in a parallel control structure, and all exceeded execution times of AND blocks, and all exceeded execution times of sub-runs. The results of the calculation are shown in Table 6.

Finally, we compute the exception probability of these affected-runs and compare them with the user defined threshold to determine whether these runs will have an exception or not. Exception probability of an affected-run is the ratio of run overtime (the difference between execution time of the run and the deadline) to the deadline constraint. For example: exception probability of affected-run r1 = (Run_Exc + Run_Est − Deadline) ÷ Deadline = (3121 + 120000 − 83938) ÷ 83938 = 0.47 which is greater than threshold 0.1. Therefore, r1 is predicted to have an exception. Likewise, we find that affected runs r1, r2, r3, r5, r6, r7, r9, r10, r11, r15 are predicted to have exceptions. For the first prediction point in Table 7, the resulted exception probability of an instance is 0.2599, which is less than the threshold for exception probability of an instance (0.3 in this case). Therefore we conclude that there will be no exception in the future. Likewise, the results for 2nd, 3rd, and 4th predictions are summarized in Table 7.

Example 2

The aim of Example 2 is to show how to process a crossing loop in a given workflow. Fig. 12 shows the workflow schema used in Example 2. In general, the procedures for processing crossing loops and nested loops are exactly the same since crossing loops can be converted into equivalent nested loops. In order to demonstrate this conversion, we first show how to transform the workflow schema in Fig. 12 into a reduced-graph.

Table 5
Exceeded execution time of tasks for Example 1.

	Task(s)	Exceeded execution time of task	Task(s)	Exceeded execution time of task
1st prediction	A_1	−2268	B_2	337
	B_1	935	C_1	1786
	A_2	2331		
2nd prediction	A_1	2441	B_2	−967
	B_1	−121	A_3	−16
	A_2	−1753		
3rd prediction	A_1	197	D_1	−292
	B_1	1746	E_1	−2436
	C_1	1717		
4th prediction	A_1	−1164	D_1	−1311
	B_1	−1870	E_1	−1709
	C_1	−2072	F_1	−828

Table 6
Exceeded execution time of runs for Example 1.

	Run(s)	Exceeded execution time of run
1st prediction	r1, r2, r3, r5, r6, r7,r9, r10, r11, r13, r14, r15	3121
2nd prediction	r2, r3, r6, r7, r10, r11, r14, r15	-416
3rd prediction	r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12	932
4th prediction	r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12	−8954

Table 7
Summary of prediction for Example 1.

	Affected run(s)	Run(s) predicted to have an exception	Exception?	Exception probability of an instance
1st prediction	r1, r2, r3, r5, r6, r7,r9, r10, r11, r13, r14, r15	r1, r2, r3, r5, r6, r7, r9, r10, r11, r15	No	0.2599 (see 1)
2nd prediction	r2, r3, r6, r7,r10, r11, r14, r15	r2, r3, r6, r7, r10, r11, r15	Yes	0.3909 (see 2)
3rd prediction	r1, r2, r3, r4, r5, r6,r7, r8, r9, r10, r11, r12	r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12	Yes	1.0000 (see 3)
4th prediction	r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12	r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12	Yes	1.0000 (see 3)

1: 0.0309 + 0.0093 + 0.0028 + 0.0065 + 0.0019 + 0.0006 + 0.0014 + 0.0004 + 0.0001 + 0.0132 = 0.0671, 0.0309 + 0.0093 + 0.0028 + 0.0065 + 0.0019 + 0.0006 + 0.0014 + 0.0004 + 0.0001 + 0.1470 + 0.0441 + 0.0132 = 0.2582
0.0671 ÷ 0.2582 = 0.2599
2: 0.0093 + 0.0028 + 0.0019 + 0.0006 + 0.0004 + 0.0001 + 0.0132 = 0.0283, 0.0093 + 0.0028 + 0.0019 + 0.0006 + 0.0004 + 0.0001 + 0.0441 + 0.0132 = 0.0724, 0.0283 ÷ 0.0724 = 0.03909
3: 0.0309 + 0.0093 + 0.0028 + 0.1029 + 0.0069 + 0.0019 + 0.0006 + 0.0216 + 0.0014 + 0.0004 + 0.0001 + 0.0045 = 0.1833
0.1833 ÷ 0.1833 = 1.0000

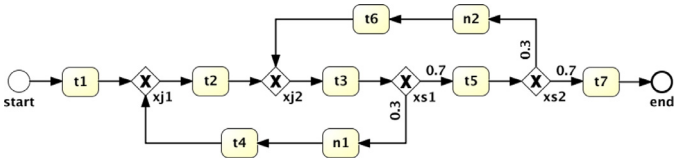

Fig. 12. Workflow schema that contains one crossing loop.

Firstly, the inner loop is converted into a sub-graph (sg1) as shown in Fig. 13 (a). Next, the resulting sub-graph is used to replace the inner loop from the workflow schema. The revised workflow schema is shown in Fig. 13 (b). Likewise, we continue to convert the outer loop into a sub-graph (sg2) as shown in Fig. 13 (c). The reduced-graph derived from the above transformation is shown in Fig. 13 (d).

Next, we show how crossing loops can be converted into equivalent nested loops. If we unfold the sub-graph sg1 in Fig. 13 (b), we can get the graph shown in Fig. 14, a workflow that contains only a nested loop.

Finally, we can generate the following runs from Fig. 14. Note that the runs generated from the workflow in Fig. 12 and the runs generated from the workflow in Fig. 14 are exactly the same. Using the same approach from Example 1, we can predict the exception for the generated runs. Due to space limitations, we will not show the remaining steps of the prediction.

Runs of the workflow in **Fig. 12**

start -> t1 -> t2 -> t3 -> t5 -> t7 -> end
start -> t1 -> t2 -> t3 -> n1 -> t4 -> t2 -> t3 -> t5 -> t7 -> end
start -> t1 -> t2 -> t3 -> t5 -> n2 -> t6 -> t3 -> t5 -> t7 -> end
start -> t1 -> t2 -> t3 -> n1 -> t4 -> t2 -> t3 -> t5 -> n2 -> t6 -> t3 -> t5 -> t7 -> end
start -> t1 -> t2 -> t3 -> t5 -> n2 -> t6 -> t3 -> n1 -> t4 -> t2 -> t3 -> t5 -> t7 -> end

Runs of the workflow in **Fig. 14**

start -> t1 -> t2 -> t3 -> t5 -> t7 -> end
start -> t1 -> t2 -> t3 -> n1 -> t4 -> t2 -> t3 -> t5 -> t7 -> end
start -> t1 -> t2 -> t3 -> t5 -> n2 -> t6 -> t3 -> t5 -> t7 -> end
start -> t1 -> t2 -> t3 -> n1 -> t4 -> t2 -> t3 -> t5 -> n2 -> t6 -> t3 -> t5 -> t7 -> end
start -> t1 -> t2 -> t3 -> t5 -> n2 -> t6 -> t3 -> n1 -> t4 -> t2 -> t3 -> t5 -> t7 -> end

5. Simulation experiment

In our experiment, we use five different workflow schemas to simulate our run-based exception prediction algorithm. First, workflow schemas are created by using Signavio Process Editor (Signavio) (Signavio, 2015) and are stored in BPMN format. Next, these schemas are edited so that they are compatible with the BIMP Simulator (BIMP) (BIMP, 2015) and BPMN parser. Note that the details for

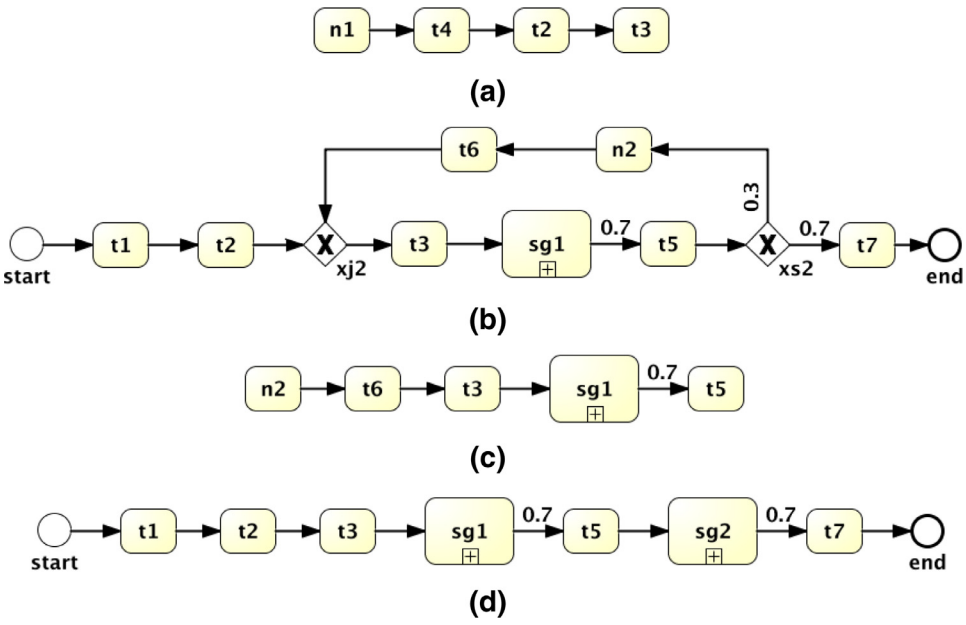

Fig. 13. (a) Sub-graph for inner loop containing t2, t3, t4 and n1 (b) Reduced-Graph after converting the inner loop into sg1 (c) Sub-graph for outer loop containing t3, t5, t6, n2, and sg1 (d) Reduced-graph for workflow schema in Fig. 12.

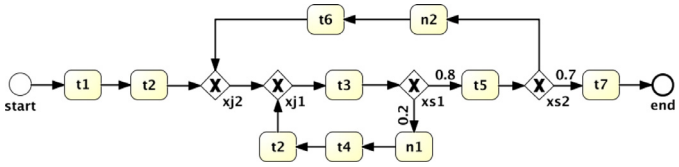

Fig. 14. Equivalent workflow schema of Fig. 12.

the modification are shown in the appendix. Thirdly, we select a uniform distribution as execution time distribution for each task, and input the branch probability for each XOR-split gateway in the BIMP Simulator. Fourthly, we generate two sets of 10,000 process instances in the BIMP Simulator, creating two MXML files, where one is used for prediction and another one is used to calculate deadlines.

In the following experiment, the deadlines are calculated from the average of actual execution times of process instances for the workflows generated from the MXML files. In addition, the task allowable execution times are calculated from the uniform distribution of the two parameters (minimum a, and maximum b) in the BIMP Simulator. The time unit used is milliseconds. For scenario I, we adopted the same workflow schema given in (Eder and Pichler 2002). However, since the task execution times in (Eder and Pichler 2002) are constant, we extend (Eder and Pichler 2002) to suit our experiment setting. Let c be the execution time of a task in (Eder and Pichler 2002), then we set parameter $a = c$, and parameter $b = a + 5000$ ms. For other scenarios, as no execution time information is provided from the workflow schema, we set the following two rules for parameters a and b.

(1) If no XOR/AND branches exist in the workflow, then the execution time of the workflow is equal to the sum of all executed tasks. Therefore, we set a=5000 and b=10000 for all tasks.
(2) If XOR/AND branches exist in the workflow, then the execution time of the workflow is likely to be affected by which branch is selected. Therefore, we set parameter a and b for each XOR/AND branch differently.

Moreover, the cumulative looping probability threshold, and the threshold for exception probability of an affected-run are set to 0.01 for all simulation experiments. We measure the accuracy of our run-based exception prediction algorithm for five different "*thresholds for exception probability of an instance*" (0.3, 0.2, 0.1, 0.05, and 0.01) with respect to ten prediction points. We use relative positions to the deadline as the prediction points since the execution times of workflow schemas are different from one scenario to another. The formula for calculating the accuracy can be defined as follows:

$$\text{Accuracy} = \frac{\text{Number of correct prediction}}{\text{Total number of instances intersected by the predicion point}}$$

5.1. SCENARIO I

Fig. 15 shows the workflow schema used in this scenario, it consists of sequential, conditional, and parallel control structures only. One of the interesting characteristics of this scenario is that the workflow schema includes nested parallel control structures leading to 16 execution paths (runs).

Table A.1 in the Appendix shows the basic information of the uniform distribution for each task, where a and b are the two arguments that are minimum and maximum values for uniform distribution. From Fig. 16 we can see that the accuracy for threshold 0.01 is the lowest among all threshold values. One of the reasons is that, for this threshold, the number of instances predicted as having exceptions increases but they actually do not have an exception. The accuracy of other thresholds are above 50% at the 0% prediction point, above 60% at the 10–50% prediction points, and above 75% at the remaining prediction points. From the above experiment, we can see that our run-based exception prediction algorithm achieves good results in workflows which consist of sequential, conditional, and parallel control structures.

5.2. SCENARIO II

In this scenario, we analyze the prediction result for a workflow that contains a structured loop in the second half of the schema. Note that the iterative control structure in Fig. 17 contains one selective control structure which has two branches (one with task t7 and the other one with task t8). Because of such embedded structures, each iteration can produce twice the number of runs with respect to the

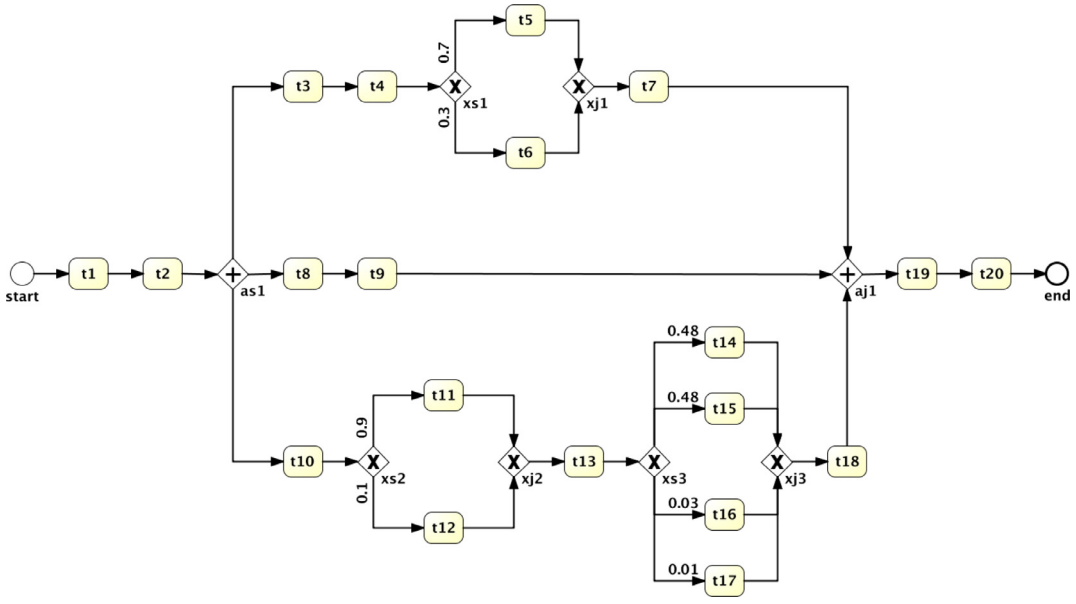

Fig. 15. Workflow schema that consists of sequential, selective, and parallel control structures only.

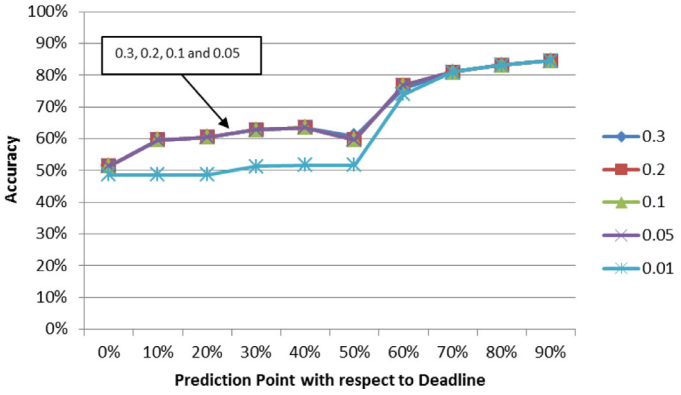

Fig. 16. Accuracy of different thresholds for exception probability for scenario I.

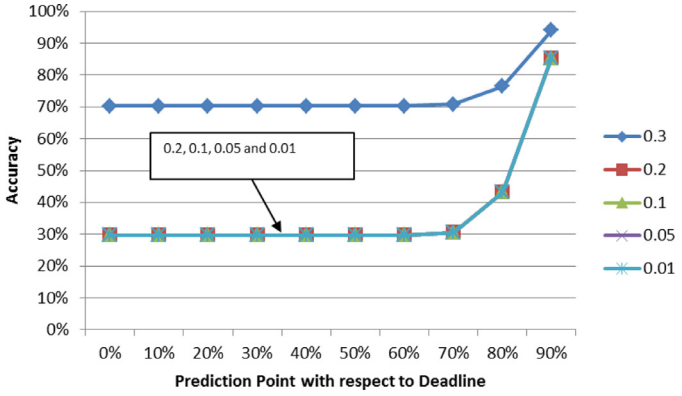

Fig. 18. Accuracy of different thresholds for exception probability for scenario II.

number of iterations. For example, if the number of iterations is 0, 1, 2, and 3, there are 2, 4, 8, 16 runs, respectively.

Table A.2 in the Appendix shows the parameters used in the uniform distribution for the workflow. In some workflow schemas, back edges in a loop may not contain any tasks. A back edge is used to connect the XOR split gateway with the XOR join gateway in a loop. For instance, a back edge in Fig. 17 is used to connect xs1 to xj1. Without a task embedded in a back edge of a loop (imagine that t9 is not included in the workflow), we cannot identify how many times that particular loop has been iterated by just analyzing the BIMP simulator result. In order to overcome this problem, we add an additional null task to the beginning of all back edges in the loops of all the workflows in our experiments. Note that the parameters a and b for n1 (null task) are 0. The deadline of the workflow for this scenario is 53015. From Fig. 18, we can see that the threshold of 0.3 provides the best prediction result and minimum of 70% accuracy is recorded for all prediction points. We can also observe that for threshold 0.3, the accuracy is the same for all prediction points except the last two. The accuracy increases to almost 95% when the prediction point is getting closer to the deadline. The change in accuracy at the later stage of prediction points is caused by the fact that a loop is located at the latter half of the workflow and there is a lack of information about the number of iterations of the loop at the beginning of execution. From the experiment results, we can also observe that the accuracy for the

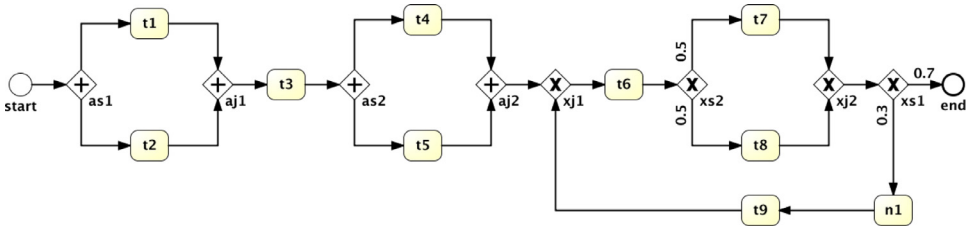

Fig. 17. Workflow schema that contains one structured loop in the latter half of the workflow schema.

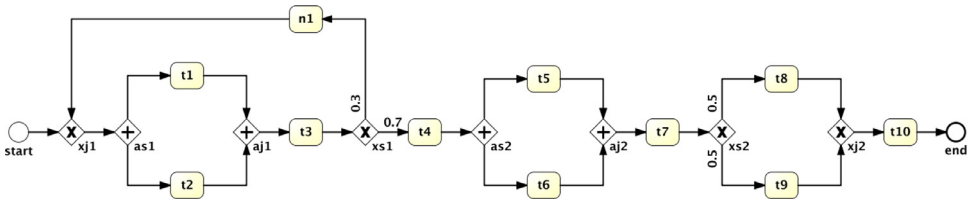

Fig. 19. Workflow schema that contains one structured loop in the former half of the workflow schema.

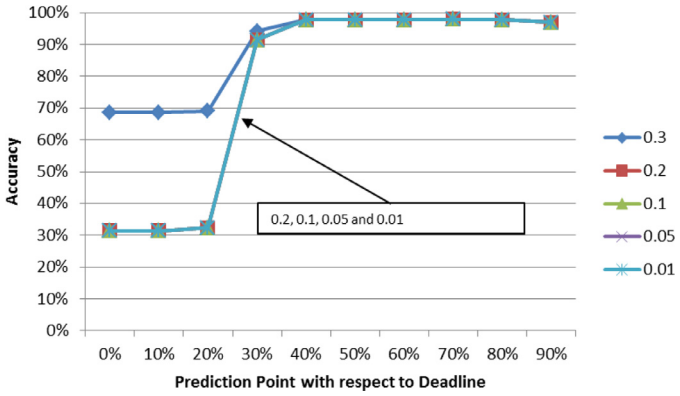

Fig. 20. Accuracy of different thresholds for exception probability for scenario III.

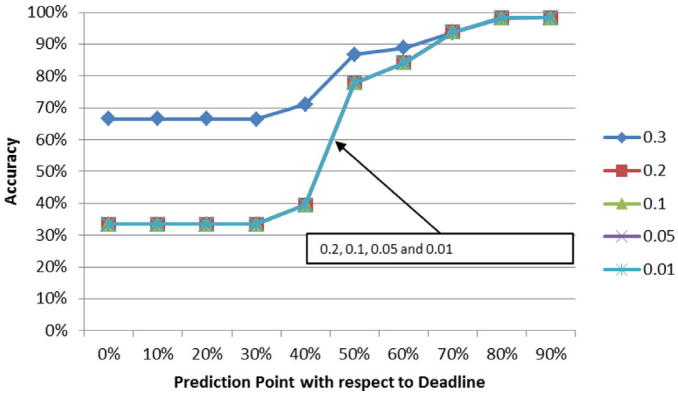

Fig. 22. Accuracy of different thresholds for exception probability for scenario IV.

remaining thresholds is relatively low for all prediction points except the last two. From our analysis, we find that low accuracy is mainly caused by those workflow instances wrongly labeled as having an exception in the future.

5.3. SCENARIO III

In this scenario, we analyze the accuracy of our proposed algorithm based on a workflow that contains a structured loop at the beginning of the schema. The workflow schema used in the experiment is depicted in Fig. 19. Note that a parallel control structure (with t1 and t2) is also nested in the structured loop.

The deadline calculated for this experiment is 65307. The parameters for uniform distribution for scenario III are shown in Table A.3 in the Appendix. From Fig. 20, we can see that the 0.3 threshold has a better result compared to the others. The accuracy is approximately 70% from the beginning and increases up to 90% at the 30% prediction point. In addition, we can also notice that even after the 30% prediction point, it maintains a high accuracy result. For the remaining thresholds, 90% accuracy is only achieved after the prediction point is at 30% or later. The reason behind this outcome is the fact that the majority of instances in the prediction set have completed execution of the structured loop at around 30% of the prediction point. Therefore only a small number of workflow instances in the prediction set are relevant for prediction at points beyond 30% with respect to the deadline. This experiment also highlights the fact that the iterative control structures can significantly affect the accuracy of the prediction.

5.4. SCENARIO IV

For scenario IV, we use a workflow that contains one nested loop located at the beginning of the schema which is shown in Fig. 21. Note that in this schema, the number of runs is determined by the number of iterations of the two loops since it does not contain any selective and parallel control structures.

Table A.4 in the Appendix shows the information used in the uniform distribution for the workflow schema in Fig. 21. The deadline for this experiment is set to 83,938. From Fig. 22 we can observe that the accuracy for threshold 0.3 has a good result compared to others, maintaining the rates at above 65% or higher throughout the experiment. The significant increase in accuracy around the 50% prediction point is caused by the fact that most of the executions of loops are completed around that time point. However, other thresholds have a low accuracy at the first half of the prediction points.

5.5. SCENARIO V

In this last scenario, we use a workflow schema that contains one nested loop and one structured loop. The schema is depicted in Fig. 23.

The arguments used in the uniform distribution are shown in Table A.5 in the Appendix. From Fig. 24, we can observe that the result is quite different from the previous experiment. In this experiment, each loop can be iterated a maximum of 3 times (without exceeding 0.01 cumulative looping probability threshold), as a result 64 (4∗4∗4) runs can be generated for the workflow schema. In addition,

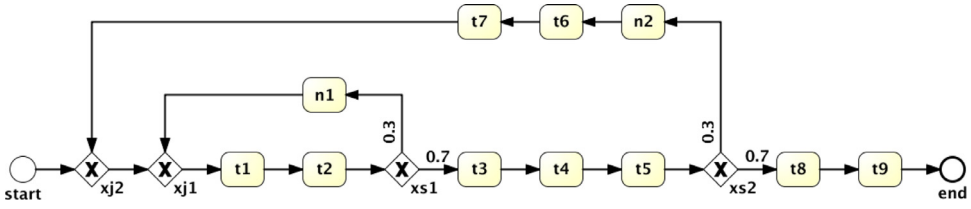

Fig. 21. Workflow schema that contains one nested loop in the former half of the workflow schema.

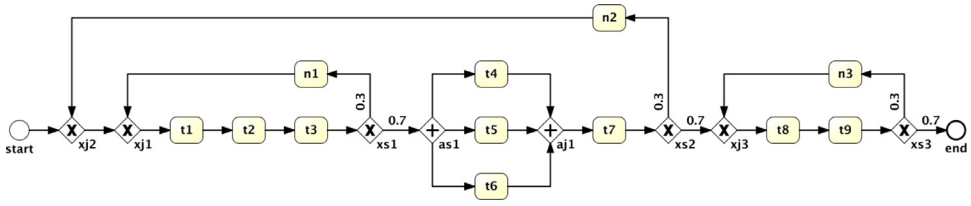

Fig. 23. Workflow schema that contains one nested loop and one structured loop.

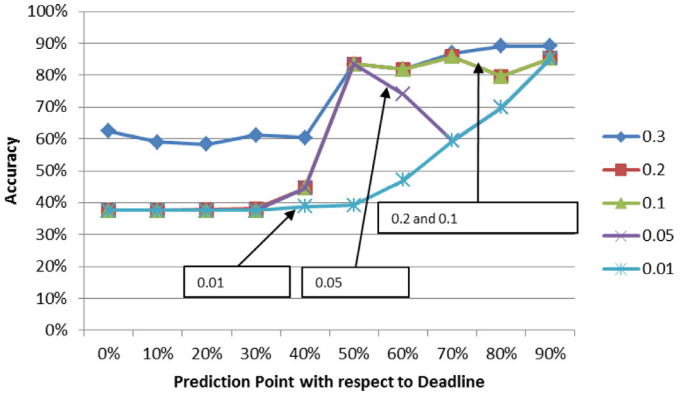

Fig. 24. Accuracy of different thresholds for exception probability for scenario V.

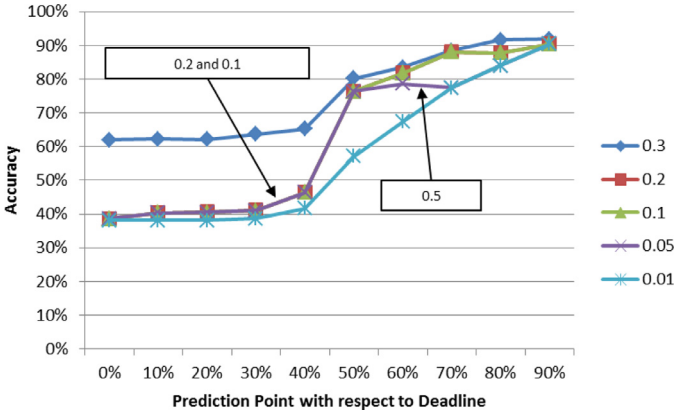

Fig. 25. Average of different thresholds for exception probability of above five scenarios.

the deadline for the experiment is set to 93779. From Fig. 24, we can see the threshold 0.3 achieves a better result compared to the others. The accuracy is approximately 60% or higher throughout the experiment. We also find that the increase in accuracy from around the 50% prediction point is caused by the fact that the execution of loops is completed around that point.

5.6. Summary of scenarios

From the above five scenarios, our run-based exception prediction algorithm can provide a good prediction result when the threshold for exception probability of an instance is 0.3. The average accuracy is about 60% when the prediction is around 40% of the deadline, and it is above 80% after 50% of the deadline (see Fig. 25). However, our run-based exception prediction algorithm achieves less satisfactory results when the threshold for exception probability of an instance is 0.01. Thus, the value assigned to the threshold for exception probability of an instance significantly impacts the achieved result. Finally, we can conclude that our run-based exception prediction algorithm can provide a good prediction result regardless of whether complex looping structures are contained in the workflows.

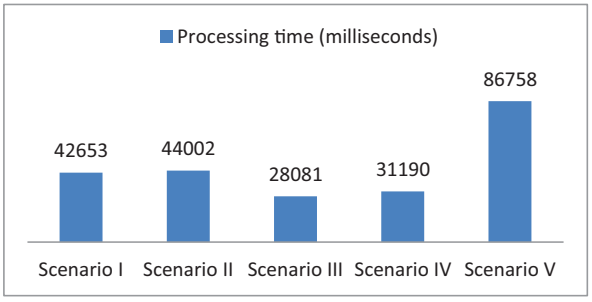

Fig. 26. Processing time for scenarios I, II, III, IV, and V.

All the experiment scenarios were executed on a computer running Windows 7 Ultimate with Intel® Core™ i7-2600K CPU and 8 GB of RAM. The version of the Java programming language used was 1.7.0. The processing time for running each of above five scenarios is shown in Fig. 26. The time unit is in milliseconds. The total processing time for simulating these scenarios was 232684 ms, that is less than 4 min, and the average processing time for each scenario was 46536 mis, that is less than 1 min.

6. Conclusion

In this paper, we propose a run-based exception prediction algorithm to predict temporal exceptions for a given workflow. In this algorithm, we use a set of runs generated at design time to predict temporal exceptions at run time. The advantages of our approach are: (1) it checks all branches of parallel control structures; (2) it analyzes all possible runs that could have mapped into the process instance; (3) it can predict temporal exceptions in workflows that consists of structured loops, nest loops, and the simple crossing loop shown in Example 2.; (4) it not only uses choice probability but also exception probability of an affected-run, and exception probability of an instance to provide more control over the prediction process. In addition, from the experiments given in Section 5 we can see that our exception prediction algorithm achieves desirable results.

As for future work, we plan to develop a handling method to resolve the temporal exceptions when they are detected. Specifically, we plan to take into account the resource perspectives in prediction since resource sharing is one of the major sources of temporal exceptions. In addition, we will devise a handling method to rearrange the allocation of resources in order to solve temporal exceptions or minimize loss. The branch probability used in this paper is a constant value defined by the workflow administrator based on estimations, or empirically generated from history logs. In the future, we intend to develop an algorithm to re-calculate or re-define the branch probability dynamically in order to improve accuracy. In a real-world situation, a workflow can be more complex than the workflows which are used as examples in this paper. Therefore further performance optimization approaches could be applied to increase the efficiency of the proposed approach. One of the limitations of the current approach is that, the algorithm for generating runs only supports the simple crossing loop demonstrated in Example 2. We are planning to further investigate the issues of generating runs from unstructured workflows involving other formations of crossing loops.

Acknowledgments

This research was funded by the University of Macau under grants RG074/09-10S/SYW/FST, MYRG2015-00054-FST and MYRG041(Y1-L1)-FST13-SYW.

Appendix

Table A.1
Two arguments used in the uniform distribution for scenario I.

Task	a (min)	b (max)	Task	a (min)	b (max)
t1	2000	7000	t11	9000	14000
t2	5000	10000	t12	6000	11000
t3	16000	21000	t13	4000	9000
t4	1000	6000	t14	1000	6000
t5	3000	8000	t15	6000	11000
t6	11000	16000	t16	15000	20000
t7	6000	11000	t17	15000	20000
t8	25000	30000	t18	3000	8000
t9	2000	7000	t19	3000	8000
t10	5000	10000	t20	2000	7000

Table A.2
Two arguments used in the uniform distribution for scenario II.

Task	a (min)	b (max)	Task	a (min)	b (max)
t1	5000	10000	t6	5000	10000
t2	7000	12000	t7	5000	10000
t3	5000	10000	t8	7000	12000
t4	5000	10000	t9	5000	10000
t5	7000	12000	n1	0	0

Table A.3
Two arguments used in the uniform distribution for scenario III.

Task	a (min)	b (max)	Task	a (min)	b (max)
t1	5000	10000	t7	5000	10000
t2	7000	12000	t8	5000	10000
t3	5000	10000	t9	7000	12000
t4	5000	10000	t10	5000	10000
t5	5000	10000	n1	0	0
t6	7000	12000			

Table A.4
Two arguments used in the uniform distribution for scenario IV.

Task	a (min)	b (max)	Task	a (min)	b (max)
t1	5000	10000	t7	5000	10000
t2	5000	10000	t8	5000	10000
t3	5000	10000	t9	5000	10000
t4	5000	10000	n1	0	0
t5	5000	10000	n2	0	0
t6	5000	10000			

Table A.5
Two arguments used in the uniform distribution for scenario V.

Task	a (min)	b (max)	Task	a (min)	b (max)
t1	5000	10000	t7	5000	10000
t2	5000	10000	t8	5000	10000
t3	5000	10000	t9	5000	10000
t4	5000	10000	n1	0	0
t5	7000	12000	n2	0	0
t6	8000	13000	n3	0	0

References

Bierbaumer, M., Eder, J., et al., 2005. Accelerating workflows with fixed date constraints. In: Delcambre, L., Kop, C., Mayr, H., Mylopoulos, J., Pastor, O. (Eds.), Conceptual Modeling – ER 2005, 3716:. Springer, Berlin Heidelberg, pp. 337–352.

BIMP. "BIMP - the business process simulator" from http://bimp.cs.ut.ee/ (accessed 01.12.15).

Cao, H., Jin, H., et al., 2013. Petri net based Grid workflow verification and optimization. J. Supercomput. 66 (3), 1215–1230.

del Foyo, P.M.G., Silva, J.R., 2008. Using time Petri nets for modelling and verification of timed constrained workflow systems. ABCM Symp. Ser. Mechatron.

Dumas, M., Garcıa-Banuelos, L., et al., 2011. Extended choice relation framework for workflow testing. In: Proceedings of the 12th Symposium on Programming Languages and Software Tools. Tallinn.

Eder, J., Gruber, W., et al., 2000. Temporal modeling of workflows with conditional execution paths. In: Ibrahim, M., Küng, J., Revell, N. (Eds.), Database and Expert Systems Applications, 1873:. Springer, Berlin Heidelberg, pp. 243–253.

Eder, J., Pichler, H., 2002. Duration histograms for workflow systems. In: Rolland, C., Brinkkemper, S., Saeki, M. (Eds.), Engineering Information Systems in the Internet Context, 103:. Springer, US, pp. 239–253.

Eder, J., Pichler, H., 2005. Probabilistic calculation of execution intervals for workflows. In: Proceedings of the 12th International Symposium on Temporal Representation and Reasoning. IEEE.

Grigori, D., Casati, F., et al., 2001. Improving business process quality through exception understanding, prediction, and prevention. In: Proceedings of the 27th International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc.

Jin, H.S., Myoung, H.K., 2001. Improving the performance of time-constrained workflow processing. J. Syst. Softw. 58 (3), 211–219.

Kiepuszewski, B., ter Hofstede, A.H.M., et al., 2003. Fundamentals of control flow in workflows. Acta Informatica 39 (3), 143–209.

Leong, I.-F., Si, Y.-W., et al., 2012. Predicting temporal exceptions in concurrent workflows. Models for Capital. Web Eng. Adv.: Trends and Discov. G. I. Alkhatib, IGI Global 196–218.

Li, H., Yang, Y., 2005. Dynamic checking of temporal constraints for concurrent workflows. Electron. Commer. Res. Appl. 4 (2), 124–142.

Pan, Y., Tang, Y., et al., 2005. A workflow model based on fuzzy-timing petri nets. In: Proceedings of the Ninth International Conference on Computer Supported Cooperative Work in Design. IEEE.

Signavio "Signavio process editor home page.". from http://www.signavio.com/ (accessed 01.12.15).

Son, J.H., Kim, J.S., et al., 2005. Extracting the workflow critical path from the extended well-formed workflow schema. J. Comput. Syst. Sci. 70 (1), 86–106.

van der Aalst, W.M.P., Hirnschall, A., et al.Pidduck, A., Ozsu, M.T., Mylopoulos, J., Woo, C. (Eds.), 2002. An alternative way to analyze workflow graphs. Advanced Information Systems Engineering 2348, 535–552.

van der Aalst, W.M.P., ter Hofstede, A.H.M., et al., 2003. Workflow patterns. Distrib. Parallel Databases 14 (1), 5–51.

Xie, T., Yu, Y., et al., 2009. A time exception handling algorithm of temporal workflow. In: Proceedings of IEEE International Symposium on Parallel and Distributed Processing with Applications. IEEE.

Yu, Y., Xie, T., et al., 2013. A handling algorithm for workflow time exception based on history logs. J. Supercomput. 63 (1), 89–106.

Yain-Whar Si is an assistant professor at the University of Macau. He holds a Ph.D. degree in Information Technology from the Queensland University of Technology, Brisbane. His research interests are in the areas of business process management and decision support systems.

Kin-Kuan Hoi received a M.Sc. in E-commerce Technology from the University of Macau. His research interests include business process management and e-commerce systems.

Robert P. Biuk-Aghai is an assistant professor of computing sciences at the University of Macau. His research interests include information visualization and computer-supported collaborative work. He received a Ph.D. in computing sciences from the University of Technology, Sydney.

Simon Fong graduated from La Trobe University, Australia, with a 1st Class Honors B.Eng. Computer Systems degree and a Ph.D. Computer Science degree in 1993 and 1998, respectively. Simon is now working as an Associate Professor at the Computer and Information Science Department of the University of Macau. Prior to joining the University of Macau, he worked as an Assistant Professor in the School of Computer Engineering, Nanyang Technological University, Singapore. Dr. Fong has published over 288 international conference and peer-reviewed journal papers, mostly in Data-mining and Machine-learning.

Defu Zhang is a full professor of computer science at the Xiamen University in China. His research interests include cloud computing, big data, and computational intelligence. He received a Ph.D. in computer science from the Huazhong University of Science and Technology in China.