Streaming Event Compliance

Streaming event data compliance checking in Python

Jingjing Huo	376323
Sabya Shaikh	384606
Zheqi Lyu	378653

1 Project Overview

1.1 Background

Process mining is widely used to visualize processes, analyze processes performance based on event logs. Unfortunately, the majority techniques handle the offline events data. Moreover, some techniques are applied on live event streams under the assumption, that the input stream has no spurious events which leads to the degradation of results quality on the real data. In order to release the assumption, we build probabilistic automata to filter out these spurious events.

1.2 Goal

- Our project will be able to effectively check event compliance,i.e when anomalous pattern is detected on the streaming data, alerts will be triggered and spurious events will be filtered out using probabilistic automata with different prefix length.
- Our system will handle 1000,000 streaming events in 1 hour.
- Eventually, the user must be able to access web-service via the client using HTTP post request to provide the streaming events in order to check event compliance.

1.3 Assumptions

- The order of event arrival corresponds to the order of execution.
- The event is spurious, if any of the automata alerts that event to be spurious.

1.4 Constraints

- The filtering of event will be done taking into account only recent history.
- Validity of real life data is hard to determine because of the lack of ground-truth.

1.5 Related Business Cases

Event logs are important evidence to see if a company or organization works good. Until now a mass of event logs are recorded by information systems. In order to make it more useful, some visualization and evaluation should be done. Recently, some techniques are used for analysing the underlying processes in many different fields. Such as:

Service - Insurance [2]

Organization: (Netherlands)
Process: *Dental care process analysis, 2017*
Description: Solving of many operational problems much quicker by combining Lean tools with process mining. Using process mining, VGZ was able to visualize the flow of the dental care process within weeks. This directly pointed out bottlenecks and it demonstrated that there were long waiting times when the work was handed over from medical advisors to experts and vice-versa. By applying the traditional Lean tools, such as 5x Why, CZ was able to pinpoint the actual root causes.

Results:

- Reduction of the throughput time by 40%.

Service - Public Sector[2]

Organization: Copyright mediator company(Italy)
Process: *Event Licence Approval, 2016*
Description: His case study applied process mining techniques to event licence approval process to expose deviations and performance issues. Specifically, the study involved process discovery of the "as is" model and the conformance checking of the "as is" process to the expected process.
Aim: finding out the root of the problem that was affecting company's core processes.
Results:

- Anomalies and bottlenecks were clearly detected.
- It was found that core processes didn't perform well because of the lack of quality data and transparent communications.

2 Definitions

In this section, some terms and logic about this project are specified.

2.1 Basic Terms

Case_id: It refers to the case_id in Event log.
In the below example, there will be two cases, with CaseID 1 and 2.

WINDOW_SIZE: It is prefix size of automata nodes.
In the below example, the WINDOW_SIZE = 2

MAXIMUN_WINDOW_SIZE: It is the maximum of all the WINDOW_SIZE provided by user.
For example, user provides WINDOW_SIZE as [1,2,3,4,5], then
MAXIMUN_WINDOW_SIZE = 5

windowsMemory: It contains the last n+1 events of one case in the program memory, where n is MAXIMUN_WINDOW_SIZE and the current event is at the last position of the windowsMemory.

For example: If MAXIMUN_WINDOW_SIZE=4, WINDOW_SIZE = 2
For **case_id**= CaseIDX, the WindowsMemory = [‘A’, ‘B’, ‘C’, ’D’, ‘E’]
‘A’,’B’,’C’,’D’,’E’ events belong to CaseIDX and here ‘E’ is recent and current event for the CaseIDX.

Sink_Node: It is the events for which the compliance needs to be checked.
In the below example, “C,D” is supposed to be the Sink_Node.

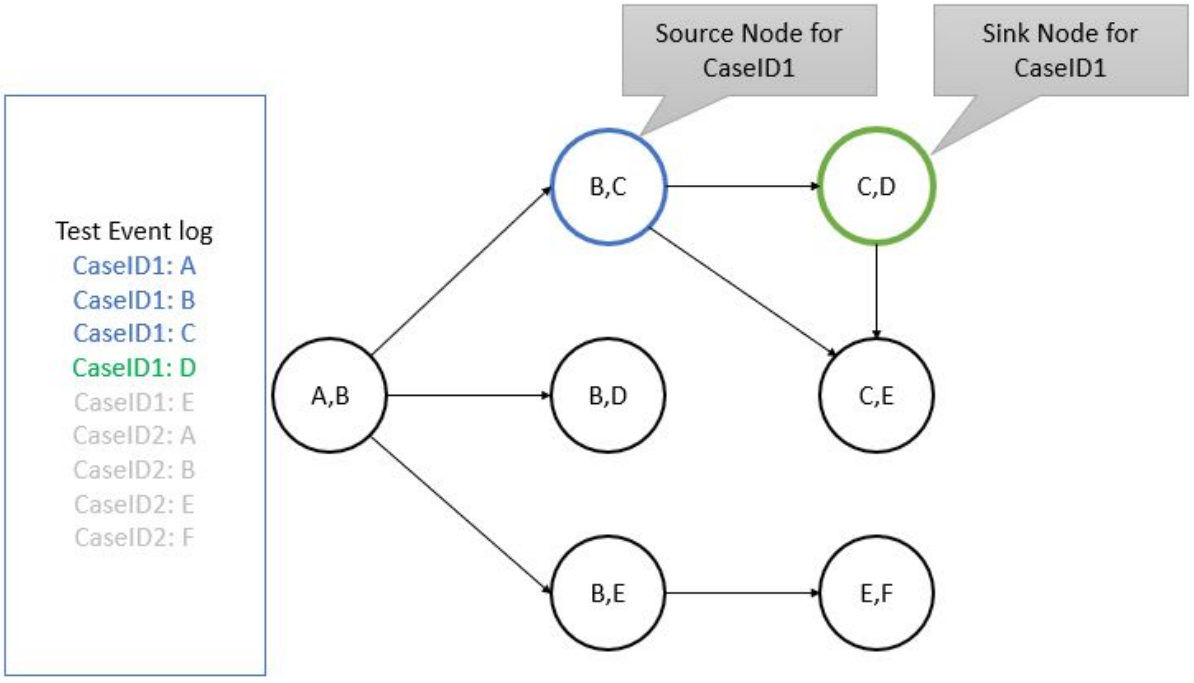
Source_Node: In the below example, the Source_Node is “B,C”

client_uuid: The username that is given by the user while running client. This is used by server to uniquely identify which client has requested the services

Connections: The occurrence of events from “Source_Node” to “Sink_Node”.

Test Event Log: Event log file used for testing.

Training Event Log: Event log used for training.



In the above example, This example is for WINDOW_SIZE =2
Events already processed for case_id = CaseID1 are A,B,C
Event being processed for case_id = CaseID1 is D
Event not yet processed for case_id = CaseID1 is E
Events for case_id =CaseID2 are not yet processed - A,B,E,F

2.2 Business logic

The alert is triggered only when business asks for it.
For example, in our system, the deviations can occur in two cases :
Case 1: The incoming event is never supposed to occur after a particular event;
Case 2: The incoming events probability to occur after a particular event is lesser than the threshold probability.

“Triggering alerts only when deviations have occured” saves time and CPU cycles for the server.

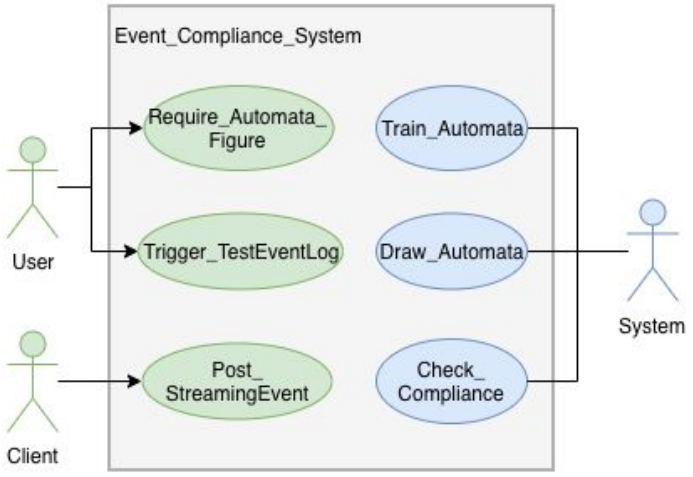
To measure the correctness of the alerted event we will need the below listed information in the alerts send to user:

- Case_id of the event which is spurios
- Current Source_Node in automata.
- Current Sink_Node.
- The reason the alert was triggered:
  case 1 : Due to lower probability of occurrence of Sink_Node than threshold;
  case 2: There is no path between Souce_Node and Sink_Node in Automata.
- Expected sink_nodes/ probability

3 Requirement Analysis

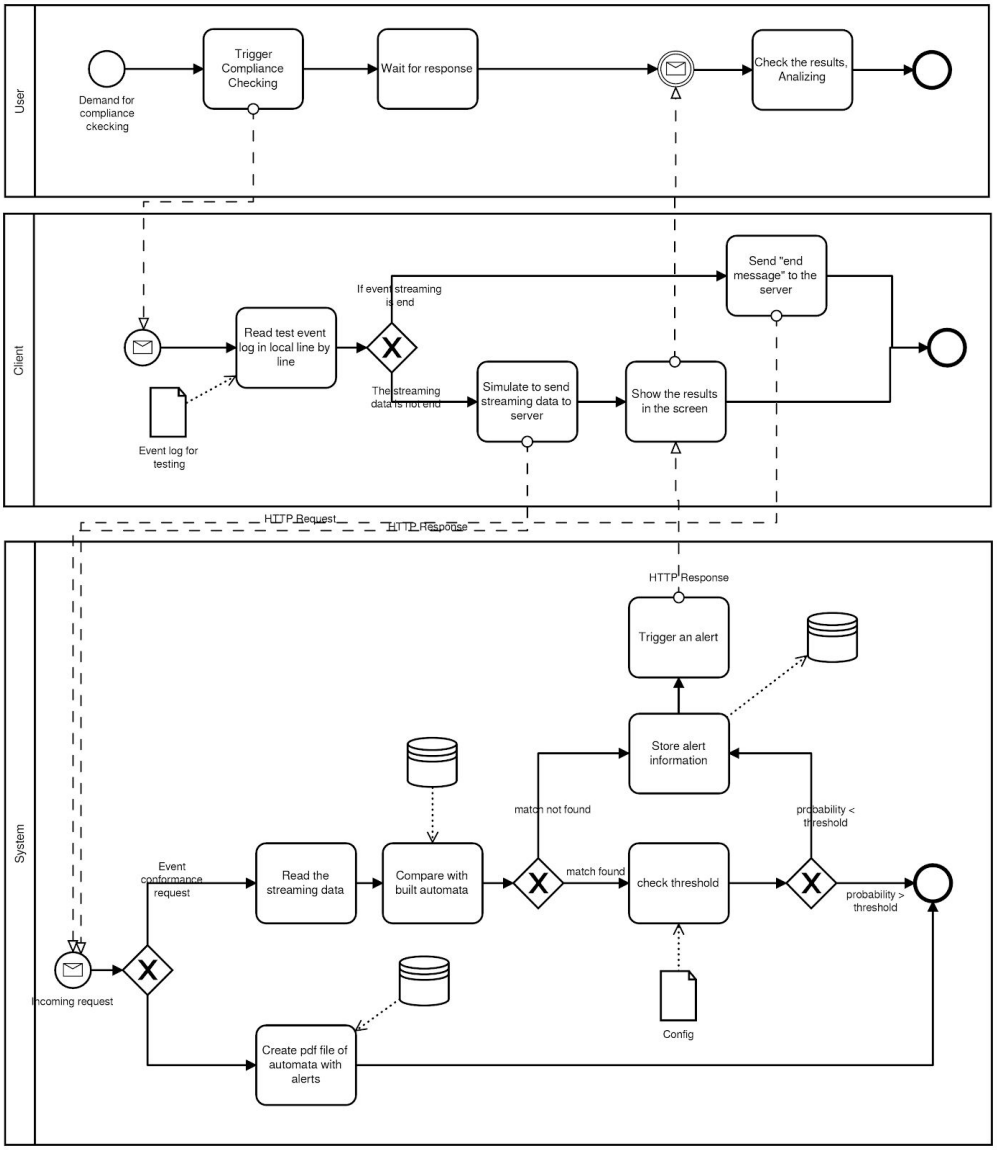
3.1 User Cases and Characters

The following **Use case Diagram** describes the interactions among the elements of our event compliance system, in particular, user, client and system:

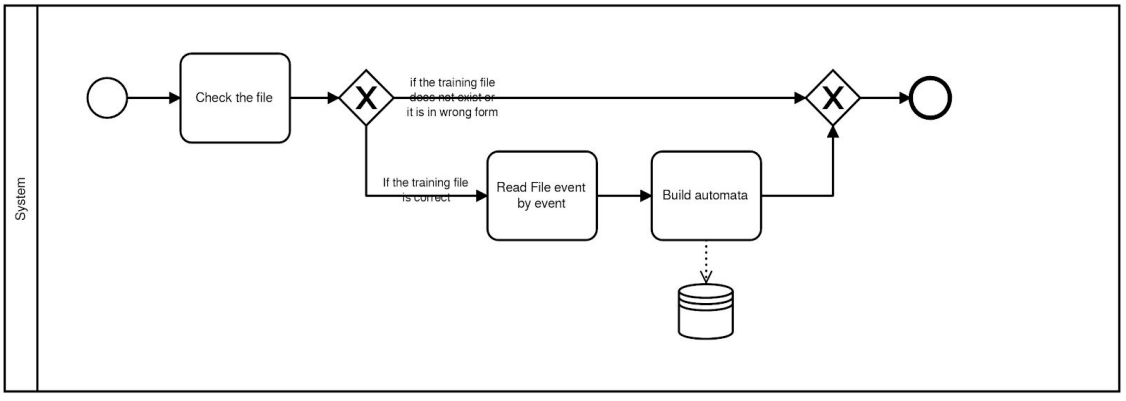


3.2 Function model (BPMN)

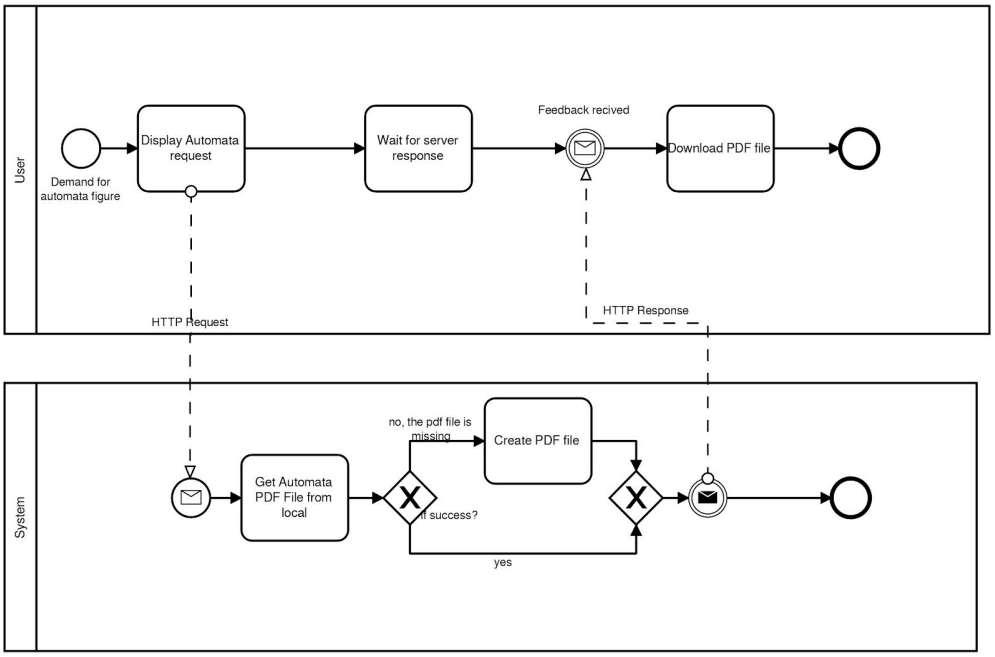
Check Compliance:



Building Automata:



Display Automata:

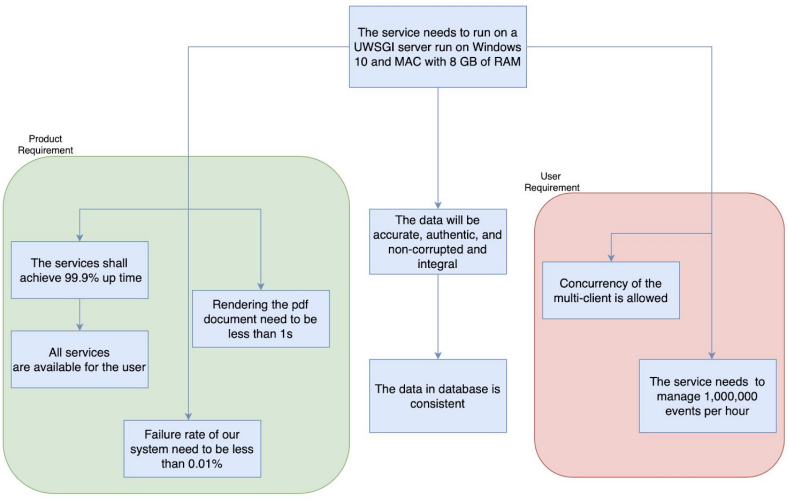


3.3 Non Functional Requirements

3.3.1 Software Quality Attributes

- Performance
  - 1,000,000 events will be processed in 1 hour. Multi streaming data come in at the same time.
  - Generation of alert in case of deviation of a single event shall take less than 4 ms seconds for 95% of the cases.
- Reliability
  - Failure of compliance checking is when an event is spurious but alert wasn't triggered.The failure rate of our system shall be less than 0.01%. This means while processing 300 events in one second, a failure can occur in every 0.5min. This failure depends on the quality of our automata and the training event log.
  - The failure that our system can not build automata or generate the automata figure shall be 0.01%. If this failure occurs we need to restart our training phase to ensure that our automata is created successfully.
- Availability
  - The services shall achieve 99.9% uptime.
    In case the server is busy building automata, the user shall be notified with busy status- This would happen rarely as we consider training the automata only once initially.
- Integrity
  - The data shall be 100% accurate.
    The data maintained in the system will be accurate, authentic, and non-corrupted at any point of time because the training data is build only once with high supervision. Thereafter, no changes are allowed on automata, hence, maintaining the consistency.

3.3.2 Allocation and Flow-Down of Requirements



3.4 Data Structure

The structure of Automata and Alert logs are stored in **MySQL** database.

Entiry connection:

source_node	sink_node	count	probability
varchar(350)-PK	varchar(350)-PK	int(11)	float

source_node: A node in automata. It is event/events previously received
sink_node: It is event/events currently received.
count: The number of times the connection with same source_node and sink_node occurs in automata
probability: The ratio between the *count* column to the total number of connections the source_node(node) has(degree field from node entity)

Entity node:

node	degree
varchar(350)-PK	int(11)

node: A node in automata.
degree: No of connections it has with other nodes

Entity alertrecord:

client_id	source_node	sink_node	alert_cause	alert_count

varchar(250)-PK	varchar(250)-PK	varchar(250)-PK	varchar(1)	float

client_id: The client that requested the service for a particular event. Also called as client_uuid in the program code

source_node: Same as Automata

sink_node:

Case 1: If the event can occur after the current source_node but probability of this event to occur is lower than threshold then the sink_node is that particular event with reference to Sink Entity;

Case 2: If the event should never occur after the current source_node then then the sink_node is that particular event with no reference to existing Automata.

alert_cause: Type: Char: 'M' or 'T'.

- 'T'(Threshold): When the alert is due to lower probability of occurrence of Sink_Node than threshold.
- 'M'(Missing Sink_Node from Automata): When there is no path between Source_Node and Sink_Node in Automata.

alert_count: Type: Integer. The number of times the alert was triggered for a particular path from Souce_Node to Sink_Node.

Entity client:

client_name	**status**
varchar(250)-PK	tinyint(1)

client_name: The client that requested the service for a particular event. Also called as client_uuid in the program code

status: The status indicates if the compliance checking is done for the client.

0- Compliance checking is not done

1- Compliance checking is done

4 Technical Requirements

Operating Environment:

Database: MySQL 8.0

Server system: Python with Flask

Operating system: Mac OS and Windows

Hardware Interface:

RAM: 8GB or higher

OS: Windows 64 Bit, Mac OS 64 Bit

Processor Name: Intel Core i5 or higher

Number of Cores: 2

Processor base frequency: 2 GHz- 2.5 GHz

5 Implementation Details/ Algorithms

In this session, we discuss the implementation and algorithms. The code is available in GitHub under https://github.com/lvzheqi/StreamingEventCompliance.git.

The project is named as: **StreamingEventCompliance** and it consists of two parts:

- Client - The client reads the tracelog file provided by user and simulates a streaming environment for event logs. Each event is thrown to server to check for compliance.
- Server- The server reads tracelog file locally stored every time the server is run. The trace log is used to train the probabilistic automata. On receiving the event submitted by client the server runs the compliance checking algorithm and returns alert in case of deviation.

5.1 Client

In this section we discuss the different functionalities the client offers the user.

5.1.1 Menu Options

Based on the arguments passed during execution of *client.py* the menu option will be displayed.

- If one argument is passed, we consider it to be username and assume the compliance checking is already done and hence user has option only to see the deviation pdf or exiting the client session.

```
There are two services:
       Press 2, if you want to show the deviation pdf
       Press 3, if you want to exit
```

  Selecting 2 will allow user to receive a pdf file containing deviations. This file is fetched from server database based on the username.
  Selecting 3 will allow user to exit the loop and end the client session.

- If two arguments are passed the client offers three options to the user as shown in screenshot below.

```
There are two services:
       Press 1, if you want to do the compliance checking
       Press 2, if you want to show the deviation pdf
       Press 3, if you want to exit
[17:44:10:68863][ Note  ]: : you can interrupt with CTR_C, once you start to do the compliance checking
```

  If this user has already done compliance checking before, then there is option of re-running the compliance check.

Selecting 1 will allow the user to run compliance checker service from the server for the file provided by the user while initiating the client

Selecting 2 will allow user to receive a pdf file with deviation if the user had already done the compliance check else it will display error message

Selecting 3 will allow user to exit the loop and end the client session.

5.1.2. Client-side Algorithm for Streaming

Step 1: Check if username/client_uuid provided is not being used currently by other client. If yes go to Step:7 else go to next step.
Step 2: Read Trace log and convert it to event log.
Step 3: Sort events based on timestamp.
Step 4: For each event start a thread to stream this data to server via post request. Event is sent in the form of dictionary

event={'case_id': string, 'activity": string }.

Step 5: Thread receives response from the server. Based on each response body the alert message is displayed on screen.
Step 6: The user is informed of the completion of compliance checker and given an option to render pdf containing automata.

5.2 Server

In this section we discuss in detail regarding the different functionalities the server offers the client, the data structure used in order to achieve those functionalities, server-side configuration that can be used to change the processing of server and also the algorithms used.

The server provides the following functionalities.

- Build Automata
- Compliance checking
- Generate Alert Automata PDF

5.2.1 Data Structures

Some of the important data structures used for programing the above functionalities are listed and described below.

- **CaseMemorizer.dictionary_cases** is used to store the cases that are being processed currently.
  It's of the dictionary type containing several sub_dicts. The last n (MAXIMUN_WINDOW_SIZE) events and all the events that are not processed in a case will be stored in the list of corresponding sub_dict.
  For eg: If n=4, then first 4 events are the latest events that have been processed and the 5th event is always the current event which has to be processed in that case_id.

Initially the CaseMemorizer.dictionary_cases is assigned ' * ' when there there are no events processed for a case_id. If a new event "a" has occurred with that case_id then

```
CaseMemorizer.dictionary_cases= {
        {
                'case_id1' : [ * , * , * , * , 'a']
        }
}
```

During processing the event 'a', other new events from this case_id1 will add into the tail. When the processing of the event 'a' is finished, an old event will be removed from the head. 'a' is pushed one step towards head. Then the event 'b' that should be processed next will be added at the 5th position (tail).

```
CaseMemorizer.dictionary_cases = {
        {
                'case_id1' : [ * , * , * , 'a', 'b']
        }
}
```

- **windowsMemory:** It contains the list of activities for one case_id that we need for processing the current event, i.e. event 'e' in the following case.

  windowsMemory = ['a', 'b' , 'c' , 'd' , 'e']

  Here, the windowsMemory is the first n(MAXIMUN_WINDOW_SIZE) events in corresponding sub_dict in CaseMemorizer.dictionary_cases

- **autos** dictionary is created based on automata class with window size(int), nodes (dictionary) and connections(list). This is the automata that is created after training.

```
autos= {
        1 : { window size: 1,nodes: {node: '', degree: ''}, connection_list : "list"} ,
        2 : { window size: 2,nodes: {node: '', degree: ''}, connection_list : "list"} ,
        3 : { window size: 3,nodes: {node: '', degree: ''}, connection_list : "list"} ,
        4 : { window size: 4,nodes: {node: '', degree: ''}, connection_list : "list"} ,
}
```

- **ClientCaseMemorizer.dictionary_cases**: It is same like **CaseMemorizer.dictionary_cases** used during compliance checking to store cases and their respective events being processed along with client_uuid.

```
ClientCaseMemorizer.dictionary_cases = {
        'client_uuid1': {
                        'case_id1' : [ * , * , * , 'a', 'b']
                        'case_id2' : [ 'y' , 'x , 'z' , 'a', 'b']
                },
        'client_uuid2': {
                        'case_id3' : [ * , * , * , 'apple', 'banana']
                        'case_id4' : [ * , 'cherry' , 'apple' , 'banana', 'grapes']
                }
        }
```

- **alert_log**: It is dictionary containing the alerts for different clients and various window sizes.
  alert_log ={
  ‘client_uuid1’: {‘windowsize’: AlertRecord}
  }
  AlertRecord is the db entity
  Client_id :varchar(250)-PK
  source_node: varchar(250)-PK
  sink_node:varchar(250)-PK
  alert_cause: varchar(1)
  alert_count: float

- **response:** It is a dictionary that contains the compliance checking output that is sent to client.
  response[WINDOW_SIZE] {
  'body': ‘M’/’T’/’OK’ (Mandatory)
  'case_id': case id (Optional)
  'source_node': source node of the current connection, with separator `,` (Optional)
  'sink_node': sink node of the current connection, with separator `,` (Optional)
  'cause': current connection (If alerts due to lesser probability)
  'expect': expected sink_nodes for current source_node/ minimum expected probability= threshold(Optional)
  }

  response[WINDOW_SIZE][‘body’]= ‘T’ indicate that the alert was due to probability of connection being lower than threshold
  response[WINDOW_SIZE][‘body’]= M indicate that there is no connection with that source_node and sink_node
  response[WINDOW_SIZE][‘body’]=’OK’ indicates that the event being checked is not spurious

5.2.2 Server Configuration

We can change parameters at the server side by manipulating the content in the file - *config.ini* (under project folder).
The parameters that can be changed are:
Window Size: the default value is [1,2,3,4], and one can change it into whatever he wants to have, like [1,2,3,4,5,6] or [2,4,6] etc.
Threshold: This is the threshold probability a connection can have. If probability of a connection is below threshold then such connections are considered to be spurious and alerts are raised. The default value is 0.2.
Path of the event log for training: the default path is *data/Simple_Training2.xes*. If one wants to change it, the file should under the project folder.

Checking type: The type of compliance checking:

1. Option: DELETE_M_EVENT: This means when one event is detected as a spurious event, if the reason is it is not in the automata(the alert type is 'M'), we delete this event from the corresponding case, then do the following checking.
2. Option: KEEP_ALL_EVENTS(default): This means that we keep all events during the compliance checking despite being spurious or not.

Alert type: The type of response of alert:

1. Option: RETURN_ONE(default): If one event alerts during checking with window size 1, we don't continue doing checking with bigger window size, and only return one alert into the client, because in this case, this event will obviously alert when checking with bigger window size.
2. Option: RETURN_ALL_ALERTS: This means when one event is spurious, we will return all alerts for different window size.

5.2.3 Functionalities Offered By Server

As mentioned above, the server provides the following functionalities.

- Build Automata
- Compliance checking
- Generate Alert Automata PDF

5.2.3.1 Build Automata

Running the server.py file will initially train automata using the trace log file stored locally.

Algorithm for building automata:
Step 0: Initiate the *threadpool* with fix number of thread.
Step 1: A new event is read and appended to *CaseMemorizer.dictionary_cases* based on the client_uuid and case_id.
Step 2: Create a new Task for this event and add to the threadpool.
Step 3: Using the *CaseMemorizer*, the *windowsMemory* is calculated which has the latest n (MAXIMUN_WINDOW_SIZE) activities and the current activity which is being processed(in-total n+1 activities).
Step 4: This *windowsMemory* is further used to calculate source_node and sink_node for different window sizes.

For example:
MAXIMUN_WINDOW_SIZE =4
windowsMemory = ['a', 'b' , 'c' , 'd' , 'e']

window size	1	2	3	4
source_node	d	c,d	b,c,d	a,b,c,d
sink_node	e	d,e	c,d,e	b,c,d,e

***Note**: To calculate the source_node and sink_node we need only n latest activities for a particular case_id hence, we only store the lastest n activities plus current processed activity in the windowsMemory.*

Step 5: The calculated *source_node* and *sink_node* is inserted into **autos** dictionary by incrementing the count if the the connection and nodes already exist.
Step 6: After running the execution of all the threads the probability is calculated.
Step 7: Finally all the autos data is inserted into Database.

5.2.3.2 Compliance checking

The server exposes a service for compliance checking available at *http://0.0.0.0:5000/compliance-checker?uuid=<client_uuid>*
The client requests the server using this service providing client_uuid as path parameter and event in body parameter(json).

Alogorithm for compliance checking:
Step 1: Every event sent to server, a thread is created.
Step 2: Event is added to ClientCaseMemorizer.dictionary_cases.
Step 3: Calculate the windows memory for that case_id.
For example, if incoming event “a” is the first event for a case_id and maximum window size is 4 then the windows memory will be [*, *, *, *, 'a'].
As the events keep coming, we add the latest event at tail and move the previous events in windows memory by 1 to the left.
For eg, if we now receive “b” the windows memory will be [*, *, *, ‘a’, 'b'].
Step 4: Calculate the sink_node and source_node using this windows memory.

	Windows Memory		[*,*,*,’a’, ‘b’]	[a,b,c,d,e]
1	Windows size = 1	source_node	a	d
		sink_node	b	e
2	Windows size = 2	source_node	None	c,d
		sink_node	a,b	d,e
3	Windows size = 3	source_node	-	b,c,d
		sink_node	-	c,d,e
4	Windows size = 4	source_node	-	a,b,c,d
		sink_node	-	b,c,d,e

Step 5: Create a Connection object using the source_node and sink_node calculated above.
Step 6: This connection object is checked in ***autos*** object which was created while training automata(automata is preloaded into computer memory in autos object for speeding up).

Case 1: source_node is not None:This means the source_node and sink_node connection must be available in the automatas connection object else it is a spurious event.
Case 2: source_node is None: This means the sink_node is the starting node (starting events)
If 'case 1' go to Step 7 if 'case 2' go to Step 10.
Step 7: The automata's connection list is scanned to check if newly created connection object is available in it. If available then probability of this connection is checked. The threshold of the probability is set to default = 0.2.
Step 8: If the connection exists and probability is equal or greater than threshold then the event is considered as normal event and response object's body is set to OK. Go to Step 12
Step 9: If the connection is not available in automata or probability is lesser than threshold then it is considered to be spurious event and alert is raised. Go to Step 11.
Step 10:: Check if sink_node exists in nodes dictionary. If sink_node does not exists alert is raised.Go to Step 11.
Step 11: Alert details are inserted into ***alert_log*** object which is finally inserted into ***alertrecord*** table once all the events are processed from same client.
Corresponding response object is set with values based on alert type etc.
Step 12: Return the response variable to client.

5.2.3.3 Generate Alert Automata PDF

The server exposes a service for getting PDF with deviations at
*http://0.0.0.0:5000/*show-deviation-pdf*?uuid=<client_uuid>*
The client requests the server using this service providing client_uuid as path parameter.
The url with pdf is displayed by the client on the console.

Alogorithm for rendering PDF with deviations:
Step 1: Check if a deviations pdf with requested client_uuid already exists at server side.
If exists then go to step 5 else go to step 2
Step 2: Get the alogs(the dictionary having all alerts for a client)
Step 3: If alert logs exists in alogs then go to Step 4 else go to Step 6 because compliance checking is not done
Step 4: Create a PDF with deviations
Step 4.i : Create nodes using the nodes from autos object
Step 4.ii: Create edges using connection from autos object for which there are no alerts
Step 4.iii: For all sink_node and source_node in alertlog create nodes in graph with red outline
Step 4.iv: For all connections in alertlog create edges in graph with various colors
green- If the cause of this alert was due to probability lesser than threshold
red - If the cause of this alert was due to missing sink_node
Step 5: Send the PDF to client got to Step 7
Step 6: Send empty string to client which suggests the client that the compliance checking is not done and user must first to do compliance checking
Step 7: End

6 Project Execution

In this section we discuss how to run entire project.

6.2 Server

To run server application, we need to first run the server.py file.

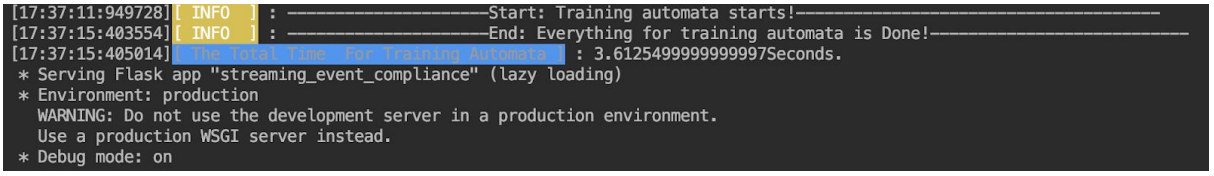
Note: *Start server before starting client.*

Locate the project directory - StreamingEventCompliance

Run the command: *python server.py.*

You'll see the configuration details on the console.

Once the training is completed you can see the below data on the screen.



6.2 Client

Once the server is up and idle the client can request the server.

To run client application , we need to run the client.py file with either one argument or two arguments

Enter the project directory - StreamingEventCompliance

Run the command:

python client/client.py <username>

or

python client/client.py <username> <absolute_filepath>

- Argument 1: **username** - This is an arbitrary username that user can provide to the client. The user must use the same name for any other runs in future to display the deviations pdf corresponding to the file the user had shared in first run for compliance checking. If only one argument is passed it is considered as username.
- Argument 2: **Absolute filepath -** It is the absolute path of the file that the user wants to check compliance for. It must have .xes extension. For running the file you could use "*Example.xes*" available in client directory inside StreamingEventCompliance project.

7. Testing

In this section we test various functionalities of server and test different possibilities of output for various parameters

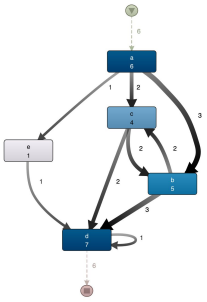
7.1 Mult-client Check

When someone is using uuid ‘cli’ to do compliance checking, then this uuid cannot be used again at the same time.

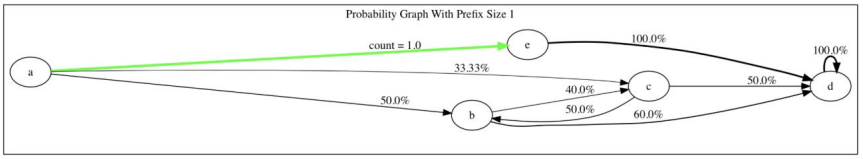
```
(lab) 075-233:client xuer$ python client.py ec1 ../data/SEC_XES/DUP/dup-global-c.xes
[14:46:41:503393][ Refuse  ] : The user with the same name is currently doing the compliance checking, please try it with other name!
```

7.2 Correctness of Compliance Checking

1. Using the event log *data/Simple_Training2.xes* to do training, setting threshold as 0.3. Then using the same event log to do compliance checking.
The screenshot using Disco (with 100% Activities and 100% Paths) is provided, from this we can see that only the threshold from ‘a’ to ‘e’ is around 0.167 which is lower than 0.3. So our system should only give one alert of type T for the connection of a to e.

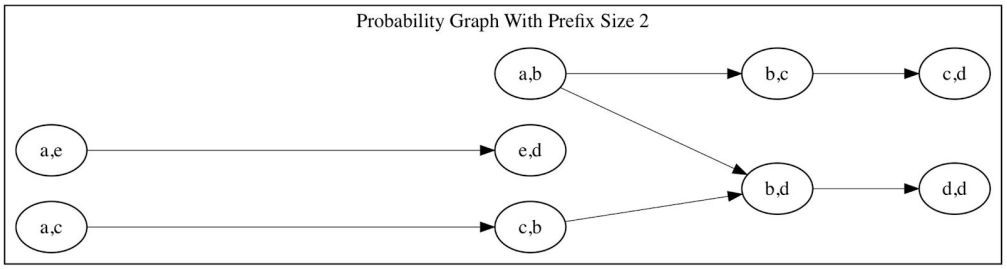


```
[17:43:18:377650][ Alert T  ] :  The threshold of the connection in case 'Case3.0' is too low.
   The minimal expected probability from  a --> e :  0.3
              The true probability from  a --> e :  0.166667
```



2. The event log from *data/Simple_Training2.xes* is follows:

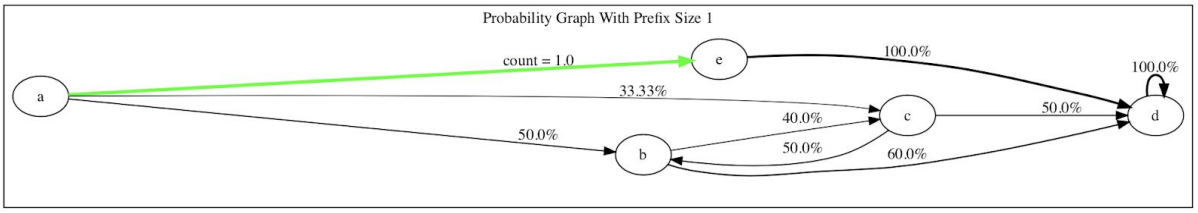
Case3.0: a c b d Case2.0: a c b d Case1.2: a b d d
Case1.1: a b c d Case1.0: a b c d Case2.1: a c b d

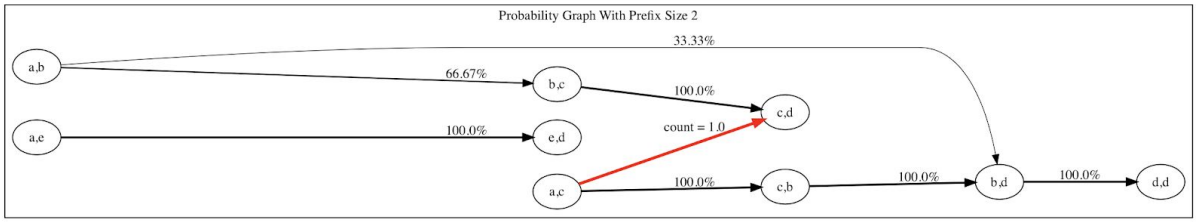


Deleting the activity ‘c’ from Case2.0 and activity ‘b’ from Case2.1 to get the file data/Simple_Training1.xes. And cases are:

Case3.0: a c b d Case2.0: a b d Case1.2: a b d d
Case1.1: a b c d Case1.0: a b c d Case2.1: a c d

Combining the Probability Graph above we can see that using this event log to do compliance checking there will no other alerts occuring in window size ‘1’, except that Alert T described in part 1. But with window size 2, there will be only one new alert from ‘ac’ to ‘cd’. Our system gives exactly the same result as expected.





7.3 Testing Parameter changes

We can change parameters like ALERT_TYPE, CHECKING_TYPE and WINDOW_SIZE mentioned in server configuration. Parameters are changed to different values and their expected output is discussed below.

Alert type:
For the case ‘17409’, if we use option RETURN_ALL_ALERTS, then the alerts of type missing will return four times with different window size, while if we use option RETURN_ONE, only the alert with the smallest window size will be return.

1. RETURN_ALL_ALERTS

```
[14:9:35:430682]Alert M :  no such connection in case '174090'
    The connection: W_Complete_Application --> W_Complete_Application
[14:9:35:430716]Alert M :  no such connection in case '174090'
    The connection: App_Pre_Acceptation,W_Complete_Application --> W_Complete_Application,W_Complete_Application
[14:9:35:430791]Alert M :  no such connection in case '174090'
    The connection: App_Incomplete_Submission,App_Pre_Acceptation,W_Complete_Application --> App_Pre_Acceptation,W_Complete_Application,W_Compl
ete_Application
[14:9:35:430908]Alert M :  no such connection in case '174090'
    The connection: App_Fully_Submission,App_Incomplete_Submission,App_Pre_Acceptation,W_Complete_Application --> App_Incomplete_Submission,App
_Pre_Acceptation,W_Complete_Application,W_Complete_Application
```

2. RETURN_ONE(default)

```
[14:21:19:574441]Alert M :  no such connection in case '174111'
    The connection: W_Complete_Application --> W_Handling_Leads
[14:21:19:577666]Alert M :  no such connection in case '174090'
    The connection: W_Complete_Application --> W_Complete_Application
[14:21:19:580968]Alert M :  no such connection in case '174111'
    The connection: W_Handling_Leads --> W_Complete_Application
```

Checking type:
For case ‘173709’, it has 7 events as below, the first alert will occur when we check compliance for the event ‘Err_Pre_Acceptation-complete’, if we use option DELETE_M_EVENT, after we get the alert that no connection from App_Incomplete_Submission-complete to Err_Pre_Acceptation-complete, we delete the

Err_Pre_Acceptation-complete, then the second alert will become no connection from App_Incomplete_Submission-complete to W_Complete_Application-complete and so on.

	Activity	Resource	Date	Time	(case) creator	(case) variant	(case) variant-index
1	App_Fully_Submission-complete	112	01.10.2011	09:57:42	Fluxicon Disco	Variant 36	36
2	App_Incomplete_Submission-complete	112	01.10.2011	09:57:43	Fluxicon Disco	Variant 36	36
3	Err_Pre_Acceptation-complete	112	01.10.2011	09:58:27	Fluxicon Disco	Variant 36	36
4	W_Complete_Application-complete	112	01.10.2011	09:58:27	Fluxicon Disco	Variant 36	36
5	W_Complete_Application-complete	10912	01.10.2011	10:26:42	Fluxicon Disco	Variant 36	36
6	W_Complete_Application-complete	10912	01.10.2011	10:27:07	Fluxicon Disco	Variant 36	36
7	W_Complete_Application-complete	10912	01.10.2011	11:40:40	Fluxicon Disco	Variant 36	36

1. KEEP_ALL_EVENTS (default)

```
[14:37:46:914673]Alert M :  no such connection in case '173709'
    The connection: App_Incomplete_Submission --> Err_Pre_Acceptation
    The expected connection:
         App_Incomplete_Submission --> App_Rejection :  0.275144
         App_Incomplete_Submission --> App_Pre_Acceptation :  0.353482
         App_Incomplete_Submission --> W_Handling_Leads :  0.36619
         App_Incomplete_Submission --> W_Fraud_Detection :  0.00518351
[14:37:46:918566]Alert M :  no such connection in case '173709'
    The connection: Err_Pre_Acceptation --> W_Complete_Application
[14:37:47:161989]Alert M :  no such connection in case '173709'
    The connection: Err_Pre_Acceptation,W_Complete_Application --> W_Complete_Application,W_Complete_Application
[14:37:47:573742]Alert M :  no such connection in case '173709'
    The connection: Err_Pre_Acceptation,W_Complete_Application,W_Complete_Application --> W_Complete_Application,W_Complete_Application,W_Complete_Application
[14:37:48:193552]Alert M :  no such connection in case '173709'
    The connection: Err_Pre_Acceptation,W_Complete_Application,W_Complete_Application,W_Complete_Application --> W_Complete_Application,W_Complete_Application
,W_Complete_Application,W_Complete_Application
```

2. DELETE_M_EVENT

```
[14:34:46:870922]Alert M :  no such connection in case '173709'
    The connection: App_Incomplete_Submission --> Err_Pre_Acceptation
    The expected connection:
         App_Incomplete_Submission --> App_Rejection :  0.275144
         App_Incomplete_Submission --> App_Pre_Acceptation :  0.353482
         App_Incomplete_Submission --> W_Handling_Leads :  0.36619
         App_Incomplete_Submission --> W_Fraud_Detection :  0.00518351
[14:34:46:874841]Alert M :  no such connection in case '173709'
    The connection: App_Incomplete_Submission --> W_Complete_Application
    The expected connection:
         App_Incomplete_Submission --> App_Rejection :  0.275144
         App_Incomplete_Submission --> App_Pre_Acceptation :  0.353482
         App_Incomplete_Submission --> W_Handling_Leads :  0.36619
         App_Incomplete_Submission --> W_Fraud_Detection :  0.00518351
[14:34:46:878291]Alert M :  no such connection in case '173709'
    The connection: App_Incomplete_Submission --> W_Complete_Application
    The expected connection:
         App_Incomplete_Submission --> App_Rejection :  0.275144
         App_Incomplete_Submission --> App_Pre_Acceptation :  0.353482
         App_Incomplete_Submission --> W_Handling_Leads :  0.36619
         App_Incomplete_Submission --> W_Fraud_Detection :  0.00518351
[14:34:46:881782]Alert M :  no such connection in case '173709'
    The connection: App_Incomplete_Submission --> W_Complete_Application
    The expected connection:
         App_Incomplete_Submission --> App_Rejection :  0.275144
         App_Incomplete_Submission --> App_Pre_Acceptation :  0.353482
         App_Incomplete_Submission --> W_Handling_Leads :  0.36619
         App_Incomplete_Submission --> W_Fraud_Detection :  0.00518351
[14:34:46:885171]Alert M :  no such connection in case '173709'
    The connection: App_Incomplete_Submission --> W_Complete_Application
    The expected connection:
         App_Incomplete_Submission --> App_Rejection :  0.275144
         App_Incomplete_Submission --> App_Pre_Acceptation :  0.353482
         App_Incomplete_Submission --> W_Handling_Leads :  0.36619
         App_Incomplete_Submission --> W_Fraud_Detection :  0.00518351
```

Window Size: the default value is [1,2,3,4], and one can change it into whatever he wants to have, like [1,2,3,4,5,6] or [2,4,6] etc.[1, 2, 3]
If the user give the window size [3, 4, 6], then as we can see in the following picture, the checking begins with window size 3, and then do 4 and 5.

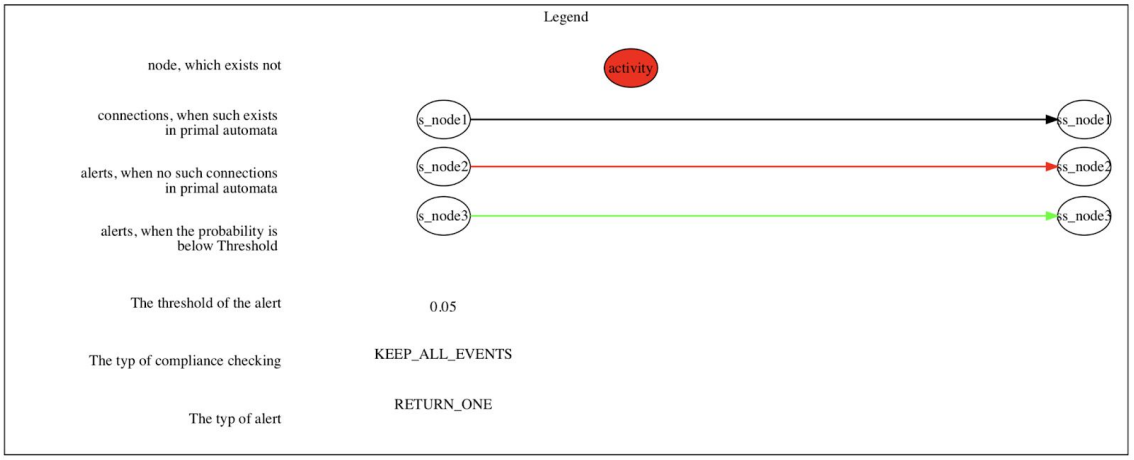
```
[16:12:51:756706]DEFAULT : WINDOW_SIZE: [1, 2, 3, 4] CHECKING_TYPE: KEEP_ALL_EVENTS  ALERT_TYPE: RETURN_ONE
[16:12:51:757894]PATH : /Users/jingjinghuo/Documents/PycharmProjects/StreamingEventCompliance/streaming_event_compliance/../data/A4.xes
[16:12:51:757930]USER-DEFINED : WINDOW_SIZE: [3, 4, 6]  MAXIMUN_WINDOW_SIZE: 6  CHECKING_TYPE: KEEP_ALL_EVENTS  ALERT_TYPE: RETURN_ALL_ALERTS
```

```
(StreamEC) 080-139:StreamingEventCompliance jingjinghuo$ python client/client.py z219 client/Example_EventLogForTesting.xes
There are two services:
        Press 1, if you want to do the compliance checking
        Press 2, if you want to show the deviation pdf
        Press 3, if you want to exit
[16:12:59:719676]Note: : you can interrupt with CTR_C, once you start to do the compliance checking
1
[16:13:0:476201][ INFO  ] : ----------------start to do compliance checking, please wait--------------------
[16:13:0:547927]Alert M :  no such start node' App_Fully_Submission,App_Incomplete_Submission,Err_Pre_Acceptation 'in case ' 173709'
    The expected start node:
        ' App_Fully_Submission,App_Incomplete_Submission,App_Pre_Acceptation ' with probability:  0.623441
        ' App_Fully_Submission,App_Incomplete_Submission,W_Handling_Leads ' with probability:  0.337905
        ' App_Fully_Submission,App_Incomplete_Submission,W_Fraud_Detection ' with probability:  0.0386534
[16:13:0:598899]Alert M :  no such connection in case '173709'
    The connection: App_Fully_Submission,App_Incomplete_Submission,Err_Pre_Acceptation --> App_Incomplete_Submission,Err_Pre_Acceptation,W_Complete_Applicatio
n
[16:13:0:598942]Alert M :  no such start node' App_Fully_Submission,App_Incomplete_Submission,Err_Pre_Acceptation,W_Complete_Application 'in case ' 173709'
    The expected start node:
        ' App_Fully_Submission,App_Incomplete_Submission,App_Pre_Acceptation,W_Complete_Application ' with probability:  0.623441
        ' App_Fully_Submission,App_Incomplete_Submission,W_Handling_Leads,W_Handling_Leads ' with probability:  0.337905
        ' App_Fully_Submission,App_Incomplete_Submission,W_Fraud_Detection,W_Fraud_Detection ' with probability:  0.0386534
[16:13:0:602815]Alert M :  no such connection in case '173709'
    The connection: App_Incomplete_Submission,Err_Pre_Acceptation,W_Complete_Application --> Err_Pre_Acceptation,W_Complete_Application,W_Complete_Application
[16:13:0:602856]Alert M :  no such connection in case '173709'
    The connection: App_Fully_Submission,App_Incomplete_Submission,Err_Pre_Acceptation,W_Complete_Application --> App_Incomplete_Submission,Err_Pre_Acceptatio
n,W_Complete_Application,W_Complete_Application
[16:13:0:663951]Alert M :  no such connection in case '173709'
    The connection: Err_Pre_Acceptation,W_Complete_Application,W_Complete_Application --> W_Complete_Application,W_Complete_Application,W_Complete_Application
[16:13:0:663992]Alert M :  no such connection in case '173709'
    The connection: App_Incomplete_Submission,Err_Pre_Acceptation,W_Complete_Application,W_Complete_Application --> Err_Pre_Acceptation,W_Complete_Application
,W_Complete_Application,W_Complete_Application
[16:13:0:664015]Alert M :  no such start node' App_Fully_Submission,App_Incomplete_Submission,Err_Pre_Acceptation,W_Complete_Application,W_Complete_Applicatio
n,W_Complete_Application 'in case ' 173709'
    The expected start node:
        ' App_Fully_Submission,App_Incomplete_Submission,App_Pre_Acceptation,W_Complete_Application,W_Complete_Application,W_Complete_Application ' with proba
bility:  0.49
        ' App_Fully_Submission,App_Incomplete_Submission,W_Handling_Leads,W_Handling_Leads,App_Pre_Acceptation,W_Complete_Application ' with probability:  0.2
7625
        ' App_Fully_Submission,App_Incomplete_Submission,W_Handling_Leads,W_Handling_Leads,W_Handling_Leads,W_Handling_Leads ' with probability:  0.06125
        ' App_Fully_Submission,App_Incomplete_Submission,App_Pre_Acceptation,W_Complete_Application,W_Complete_Application,App_Acceptation ' with probability:
  0.13375
        ' App_Fully_Submission,App_Incomplete_Submission,W_Fraud_Detection,W_Fraud_Detection,W_Fraud_Detection,W_Fraud_Detection ' with probability:  0.03875
```

7.4 Generation of Alert Automata PDF

In this part, we show the legend of the deviation PDF and an example of the alert automata.

Legend: The green line represents the connection probability below the threshold, the red line is for the missing connection, which doesn't occur in the training automata and the black line shows the connections without alerts. We use different colors to refer to the different types of alert and the thickness of the line to indicate the counts of the alert in that connection.



Example: The figure below shows the alert automata with 3 different window size. The event e exists not in the training phase, while it appears in the testing event log. When we remove the all alerts (all connection and node with color), we can directly get the probability automata for training data. The expected trace of the training automata is obviously a,b,c,b,d. The unexpected connections in event log are a->c; a->e->c; d->d; a,b->d; a,c->b and a,c,b->d

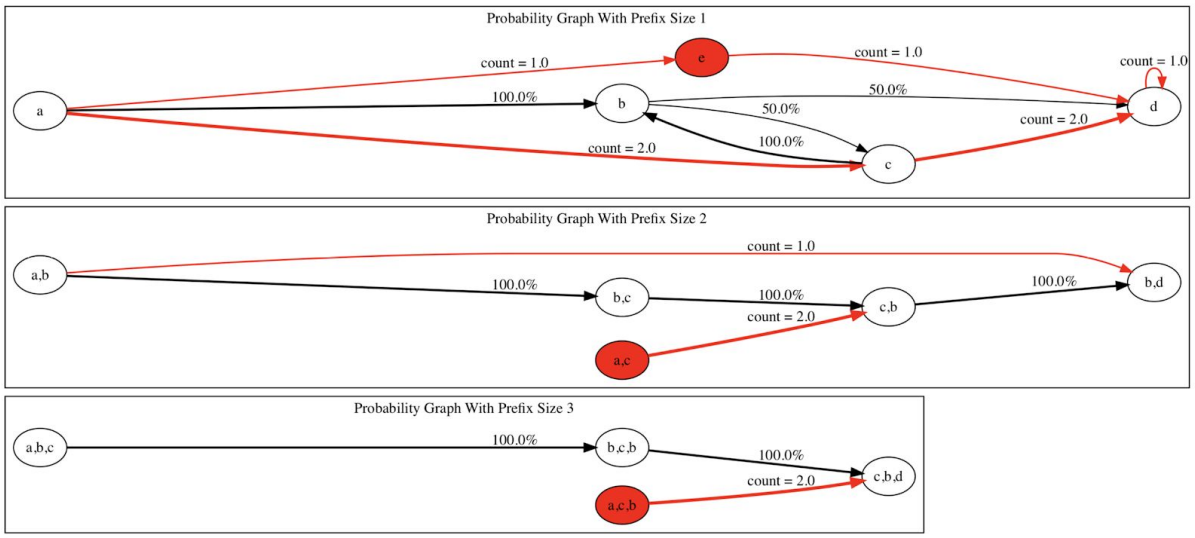


Figure: Training data: *dup-no-a.xes*, Testing data: *Simple_Testing1.xes*

8 Assessment

In this section, we first report a survey on the quality of Real Life Data and Synthetic Event Logs. Then, the performance of conformance checking is evaluated on both efficiency and effectiveness. All programs are implemented in Python, and experiments run on a computer with 2.7GHz CPU and 16GB RAM.

8.1 Comparison of Runtime

To test the efficiency of algorithm, we use the real life event logs from BPI-Challenge 2018[3]. Using Disco we filter a part of data as training event log and rest as validation sets. The runtime with different size of test event log using particular type of alert and compliance checking lists as following.

Parameter_table:

WINDOW_SIZE	THRESHOLD	CHECKING TYPE	ALERT TYPE
[1, 2, 3]	0.05	KEEP_ALL_EVENTS	RETURN_ONE

8.1.1 Comparison with varying Training/Testing event logs

In table 1 and 2 we use the same parameters mentioned in the Parameter_table. In table 1 we compare the average runtime of event logs with different event log of different sizes.

Table1: Using *C2018_Train.xes* as training event log - 104454 Events(358.9M)

TEST EVENTLOG	**TOTAL EVENTS**	**RESULTS**	**TOTAL RUNTIME (sec.)**	**AVG EVENTS PROCESSED (per sec.)**

C2018_180517_end.xes (8,5M)	2595	OK:5803; Alert T:604; Alert M:603;	2.854297	909.155564
C2018_140517_end.xes (185,2M)	56620	OK:132034; Alert T:12950; Alert M:11097;	62.964901	899.231145
C2018_090517_end.xes (685,3M)	201052	OK:460126; Alert T:49180; Alert M:41947;	222.013047	905.586418
C2018_050517_end.xes (994,8M)	303776	OK:695465; Alert T:73431; Alert M:63519;	330.611001	918.832099
C2018_Train.xes(itself)	104454	OK:295504; Alert T:16012; Alert M:0;	104.864594	996.084532

Observation: If we use the same parameters for different logs, the speed does not vary much.

Table2: Using *C2018_Training_310316.xes* as training event log - 417622 Events(1.37G)

TEST EVENTLOG	**TOTAL EVENTS**	**RESULTS**	**TOTAL RUNTIME (sec.)**	**AVG EVENTS PROCESSED (per sec.)**
C2018_180517_end.xes (8,5M)	2595	OK:5768; Alert T:725; Alert M:580;	2.981806	870.277946
C2018_050517_end.xes (994.8M)	303776	OK:690364; Alert T:85896; Alert M:61657;	336.163009	903.656833

Observation: Compare this with the results of the same event log in table 1, we can see that the size of the event log for training will not affect the speed very much.

8.1.2 Comparison with varying Parameters

We test by changing different parameters from Parameter_table. The results and observations are discussed below. *Testing Parameter* is the parameter that is being tested. We change the different values for that testing parameter and check for the runtime. For all the following tests, we using *C2018_Train.xes* as training event log - 104454 Events(358.9G)

Testing Parameter: WINDOW_SIZE
TESTING EVENTLOG: *C2018_180517_end.xes*(2595 Events 49 Cases[8,5M])
CHECKING TYPE : KEEP_ALL_EVENTS

ALERT TYPE: RETURN_ALL_ALERTS

WINDOW_SIZE	**RESULTS**	**TOTAL RUNTIME (sec.)**	**AVG EVENTS PROCESSED (per sec.)**
[1]	OK:2116; Alert T:226; Alert M:253;	2.755969	941.5925947
[1,2,3,4,5]	OK:8698; Alert T:885; Alert M:3098;	8.493865	305.514627
[1,2,3,4,5,6,7,8]	OK:10437; Alert T:1336; Alert M:5608;	32.131246	80.762508

Observation:

1. We can see that changing the parameter “WINDOW_SIZE” has great effect on the speed, the more window size we use, the slower the compliance checking gets.
2. Explanation of the results: Using *C2018_180517_end.xes* as testing event log, we have several cases as following:
    - For the case of WINDOW_SIZE [1], we have results OK:2116; Alert T:226; Alert M:253, the sum of these values is exactly the number of events in that testing event log.
    - For the case of WINDOW_SIZE [1,2,3], we have results OK:5803; Alert T:604; Alert M:1329, for window size 1 and 2 the number of results that will be generated is exactly the number of events(because we add start event at the very beginning of each case, so for window size 2 the number of node(two events) is equal to the number of events), but for the window which bigger than 2, for example for window size 3, the number of valid nodes is the number of events mimus the number of cases, so the total number of the results plus the number of cases will be the 3 times of the number of events in the testing event log.
    - Similarly for WINDOW_SIZE [1,2,3,4,5] the sum of the results should the 5 times of the number events in the testing event log minus the 6 times of number of cases.

Testing Parameter: THRESHOLD

TESTING EVENTLOG: *C2018_140517_end.xes*(56620 Events [185,2M])
WINDOW_SIZE: [1,2,3]
CHECKING TYPE : KEEP_ALL_EVENTS
ALERT TYPE: RETURN_ALL_ALERTS

THRESHOLD	**RESULTS**	**TOTAL RUNTIME (sec.)**	**AVG EVENTS PROCESSED (per sec.)**

0.05	OK:132034; Alert T:12950; Alert M:24071;	71.128831	796.020393
0.1	OK:126522; Alert T:18462; Alert M:24071;	71.09079	796.446347
0.3	OK:98792; Alert T:46192; Alert M:24071;	72.110116	785.188031
0.5	OK:73672; Alert T:71312; Alert M:24071;	75.237867	752.546587

Observation:

1. From results in this table, we can see the bigger the threshold, the more Alert T are generated, but the number of Alert M will not change;
2. Also, compare the **AVG EVENTS PROCESSED** in this table with the the **AVG EVENTS PROCESSED** of the same event log Table1, we can see the ALERT TYPE also has great on speed, ie. using RETURN_ALL_ALERTS will make the process slower;

Testing Parameter: CHECKING TYPE
TESTING EVENTLOG: *C2018_090517_end.xes*(201052 Events [685.3M])
WINDOW_SIZE: [1,2,3]
THRESHOLD:0.05
ALERT TYPE: RETURN_ALL_ALERTS

CHECKING TYPE	**RESULTS**	**TOTAL RUNTIME (sec.)**	**AVG EVENTS PROCESSED (per sec.)**
KEEP_ALL_EVENTS	OK:460126; Alert T:49180; Alert M:90643;	246.331338	816.185231
DELETE_M_EVENT	OK:493541; Alert T:56140; Alert M:50250;	229.955756	874.307317

Observation: From results in this table, we can see that when the DELETE_M_EVENT is used, the number of Alert M will decrease, but this parameter doesn't have much effect on the speed.

Testing Parameter: ALERT TYPE
TESTING EVENTLOG: *C2018_050517_end.xes*(303776 Events [994.8M])
WINDOW_SIZE: [1,2,3]
THRESHOLD:0.05

CHECKING TYPE:KEEP_ALL_EVENTS

ALERT TYPE	RESULTS	TOTAL RUNTIME (sec.)	AVG EVENTS PROCESSED (per sec.)
RETURN_ALL_ALERTS	OK:695465; Alert T:73431; Alert M:137428;	384.166592	790.740284
RETURN_ONE	OK:695465; Alert T:73431; Alert M:63519;	329.228726	922.689838

Observation: This result is consistent with the observation before(observation 2 in the part Testing Parameter: THRESHOLD), using the RETURN_ONE will much faster.

8.2 Comparison with Alignment

We used 3 different data structures[4].They are:

Skip (three activities where the middle one can be skipped).

Duplicates (an activity appears at two positions in the process).

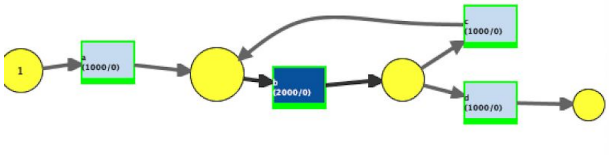
Non-free-choice (a choice in the later part of the process depends on an earlier choice).

1. Randomly inject noise with additional variable x

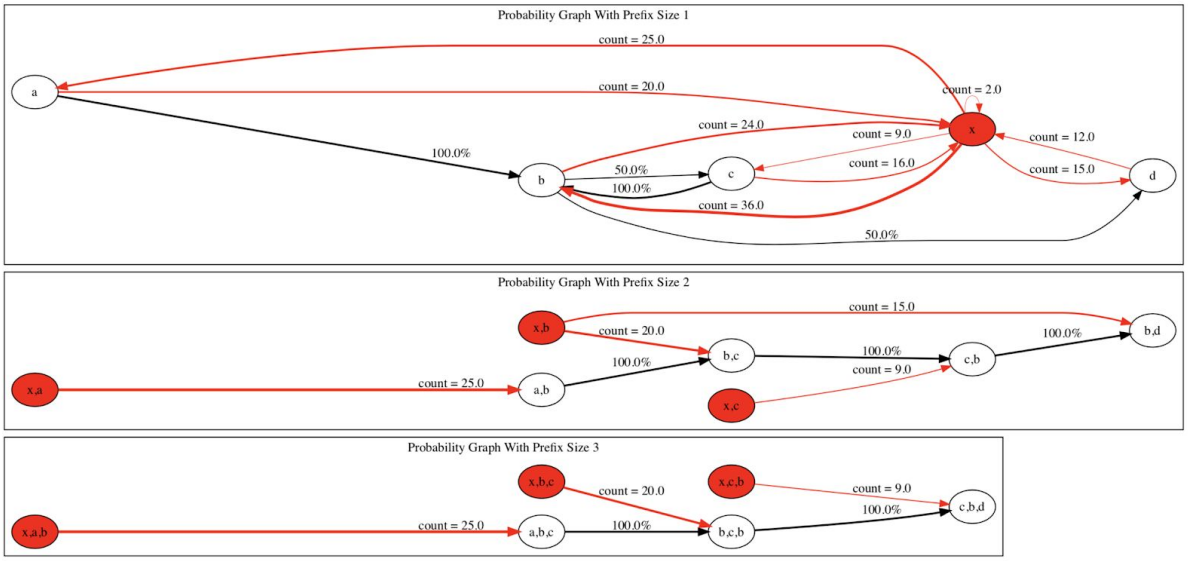
It is obvious, compliance checking deviation gives us a more intuitive feeling where the x causes alert. By prefix size 1, the total number of alerts returned by compliance checking is twice of the number of *move log only* by alignment except x as start point and x as end point. Because in probability automata, the start point and end point are unimportant.
For different structure of data, the behaviour of compliance checking is consistent with alignment.

Here we show the a concrete **example** of alignment by ProM, compliance checking deviation graph by our project and the Directly-Following-Graph by Disco.

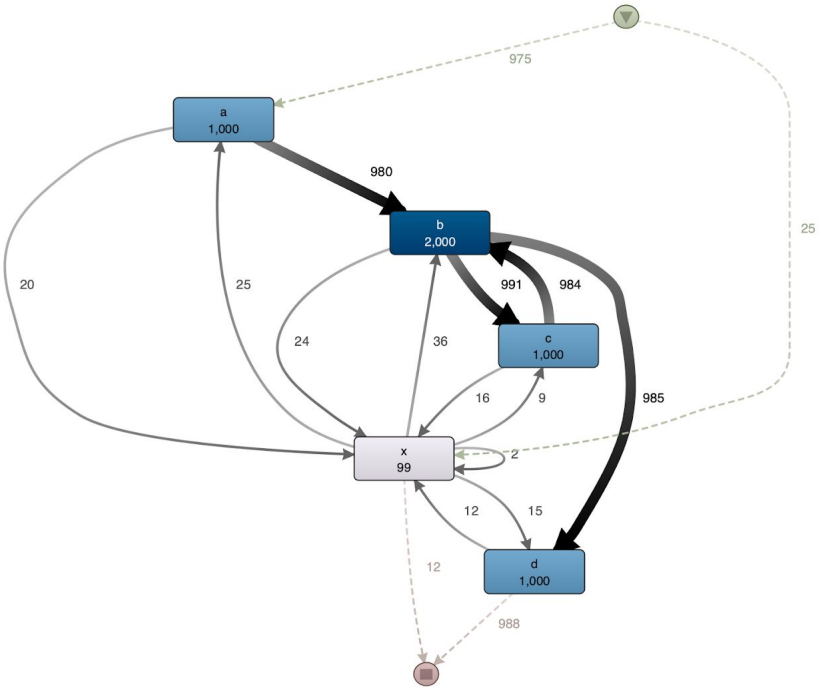
We used *dup-no-a.xes* as model/ training event log and *dup-global-a.xes* as testing event log.



The following are done by using KEEP_ALL_EVENT as CHECKING_TYPE and RETURN_ONE as ALERT_TYPE.



The following shows the Directly-Following-Graph of *dup-global-a.xes*.



Obersevation:

We have totally 99 moves on log during alignment, and 161 alerts for compliance checking. According to the Directly-Following-Graph, 99*2 = 161+12+25, where 12 traces end at x and 25 traces begin with x.

2. Randomly remove the element

In this situation for prefix 1, the total number of alerts returned by the compliance checking is equal to the number of *move model only by* alignment.

For example, the following alignment leads to 1 “move on” model.

Model	a	b	c

Log	a	<<	c

Compliance checking by our project for WINDOW_SIZE=1 and such deviations leads to 1 alert indicating connection a->c is not valid. Hence, the number of 'move on' = number of alerts (from compliance checker)

3. Other

Using compliance checking with different window size, we can fetch more informations behind the event log. The figure below shows the dependency of the choice. If we event d is followed by event a, then event c is followed by event d. Similarly, f (g) is followed by event c,d (e,d). That means, There is no choice that g follows event c,d.

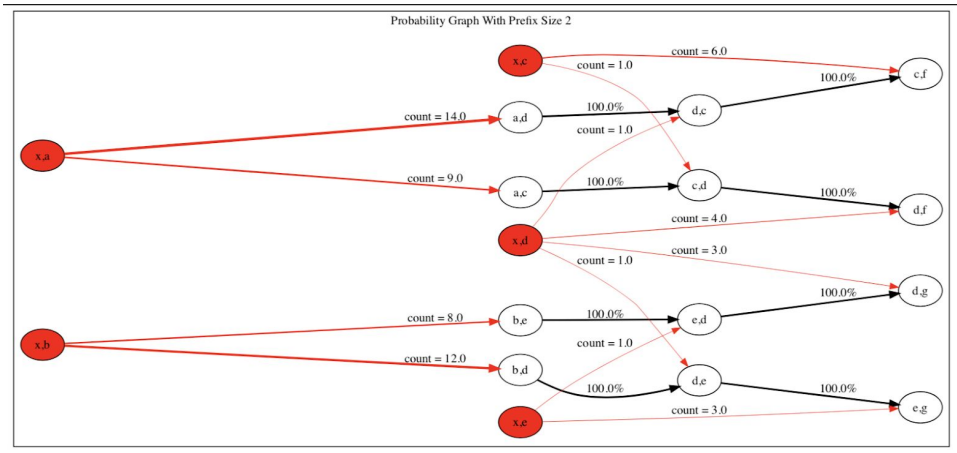


Figure: Training data: *nfc-no-a.xes* and testing data: *nfc-global-a.xes.* x is the noise variable. We apply KEEP_ALL_EVENT and RETURN_ONE to do the compliance checking, where x,_ are the spurious.

9 Dockerizing the Application

To dockerise the application follow the below steps:

Step 1: Open streaming_event_compliance/__init__.py
Step 2: Change **deploy = True** (Default = False)
Step 3: Open `StreamingEventCompliance/docker` file at terminal
Step 4: Run `sh build-run-db.sh` to pull the mysql:5.7 database and create `mysqldb` container.
Step 5: Run `sh build-compliancechecker.sh` to build `streameventcompliance` image.
Step 6: Run `sh run-compliancechecker.sh` to create `compliancechecker` container.
(Step 7): Run `sh clean-compliancechecker.sh` to stop and remove the `compliancechecker` container

Note: *If you are using docker tool box then client needs to access the services via this url http://**docker-machine**-IP:5000/<service_name> instead of this http://**0.0.0.0**:5000/<service_name>*

Reference

[1] S.J. van Zelst, M. Fani Sani, A. Ostovar, R. Conforti, M. La Rosa: Filtering spurious events from event streams of business processes. In: Proceedings of the CAISE (2018)
[2] HSPI Management Consulting, Process mining: a database of applications (2017) <http://www.hspi.it/wp-content/uploads/2017/11/HSPI_Process_Mining_Database_v1.1-Nov_17.pdf>
[3] van Dongen, B.F. (Boudewijn); Borchert, F. (Florian) (2018) BPI Challenge 2018. Eindhoven University of Technology. Dataset. <https://doi.org/10.4121/uuid:3301445f-95e8-4ff0-98a4-901f1f204972>
[4] van der Aalst, W.M.P. (Wil) (2017) Testing Representational Biases. Eindhoven University of Technology. Dataset. <https://doi.org/10.4121/uuid:25d6eef5-c427-42b5-ab38-5e512cca08a9>