Paper SAS5641-2016

Improve Your Business through Process Mining

David R. Duling and Emily (Yan) Gao, SAS Institute Inc.

ABSTRACT

Looking for new ways to improve your business? Try mining your own data! Event log data is a side product of information systems generated for audit and security purposes and is seldom analyzed, especially in combination with business data. Along with the cloud computing era, more event log data has been accumulated and analysts are searching for innovative ways to take advantage of all data resources in order to get valuable insights. Process mining, a new field for discovering business patterns from event log data, has recently proved useful for business applications. Process mining shares some algorithms with data mining but it is more focused on interpretation of the detected patterns rather than prediction. Analysis of these patterns can lead to improvements in the efficiency of common existing and planned business processes. Through process mining, analysts can uncover hidden relationships between resources and activities and make changes to improve organizational structure. This paper shows you how to use SAS® Analytics to gain insights from a real event log data.

INTRODUCTION

Outcomes such as revenues, profits, and customer turnover are visible results of many intermediate activities and events within a business. Countless influencers lurk beneath the surface of your business to affect these outcomes (Blickle et al., 2010). Event log data is a side product of information systems, recording many steps of business operations often for the purposes of auditing and security. Process mining is an emerging field that focuses on analyzing event logs with data mining tools for the purpose of business process improvement. Process mining has three main forms: process discovery, conformance check, and enhancement of business processes. Process discovery depicts the underlying process models that your business follows. The conformance check detects all events that deviated from the expected process model and measures how well actual processes conform to the expected process models by replaying event logs against the expected process model. Figure 1 shows positioning of the three main types of process mining.

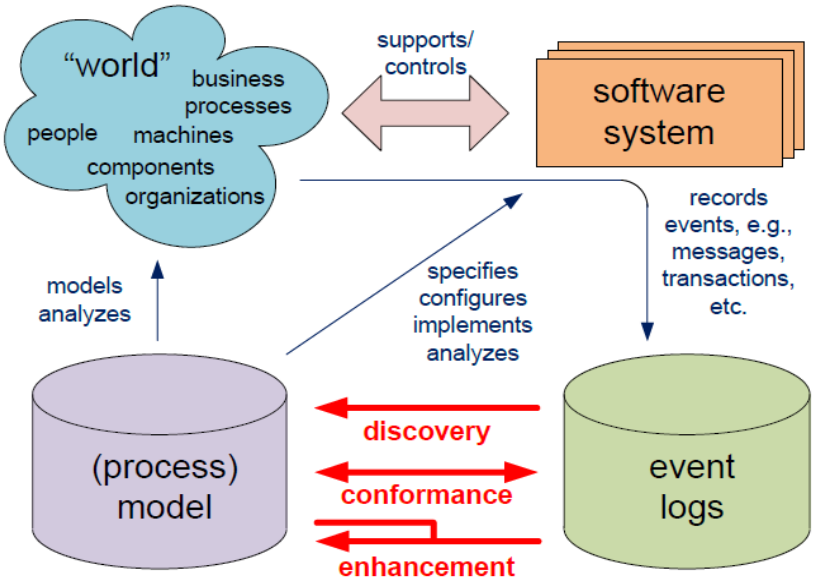

Figure 1. Positioning of the Three Main Types of Process Mining (Wil M. P. van der Aalst. 2011)

In our research, we implemented process discovery and conformance check using SAS.

This paper demonstrates the workflow of process mining as Figure 2 shows, through a case study based on the Business Process Intelligence (BPI) Challenge 2012 data, which are loan approval event logs provided by a bank from Netherlands.

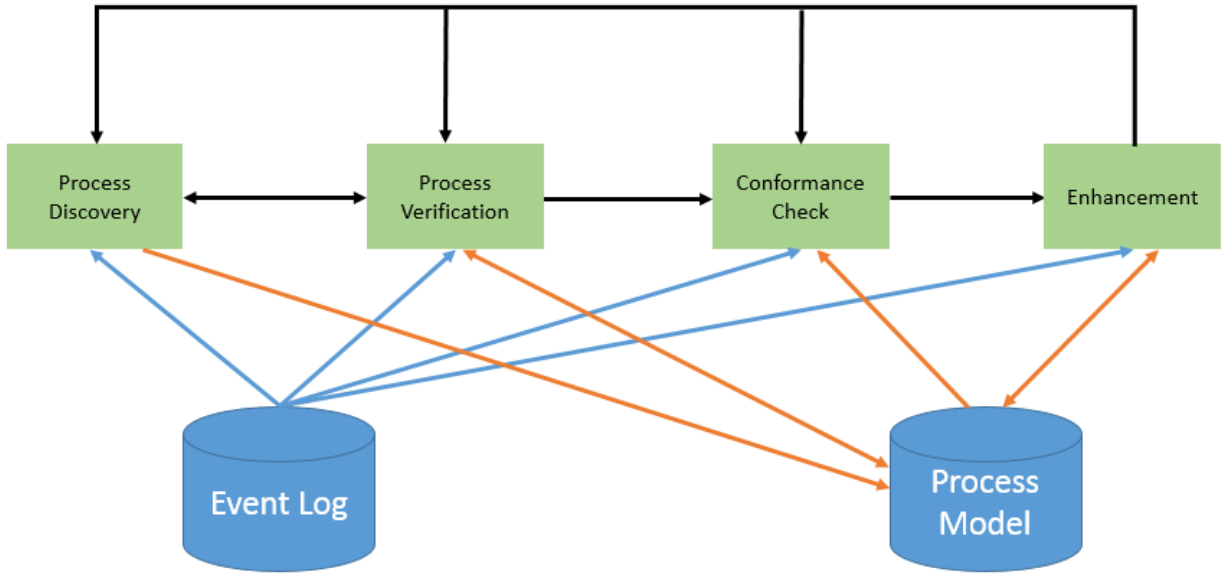

Figure 2. Workflow of Process Mining

PROCESS MODEL

Several notations exist to model operational business processes, such as Business Process Model and Notation (BPMN), Petri Nets, Unified Modeling Language (UML), and Event-driven Process Chains (EPCs), and so on. These notations have in common that processes are described in terms of activities and interactions between activities. In this research, only BPMN and Petri Nets models are used.

BPMN is a graphical notation that depicts the end to end flow of a business process. BPMN 2.0 was released in January 2011. The notation has been specifically designed to coordinate the sequence of processes and the messages that flow between different process participants in a related set of activities. Figure 3 shows the expected loan application process model expressed in terms of BPMN 2.0.

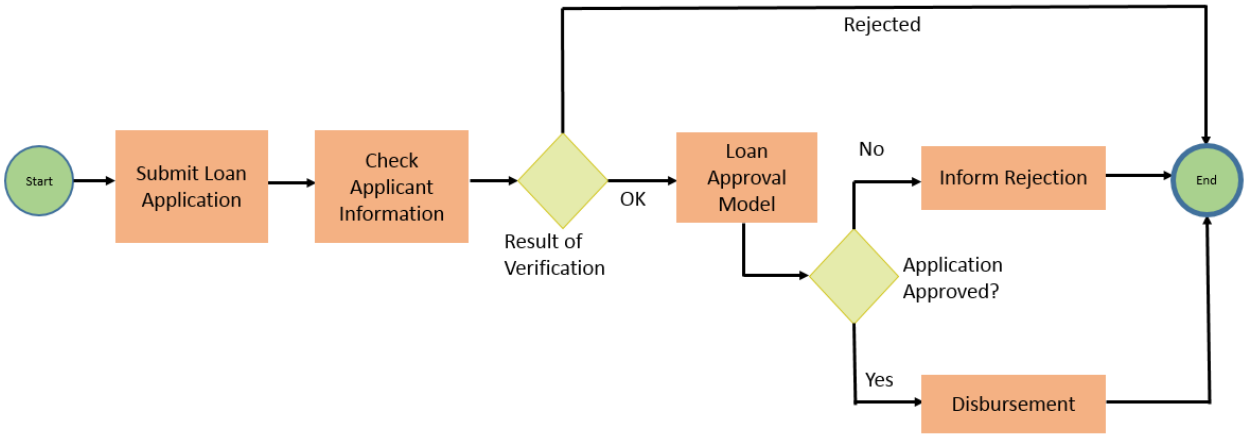

Figure 3. BPMN 2.0 Sample

The sample model has five activities displayed in rectangles and two gateways displayed as diamonds. The gateways determine which path is taken in the process based on the application data.

PROCESS LOGS

It is impossible to do process mining without process logs. The requirements for the schema of the process logs depend on your analysis goals. The process log should include at least a Case ID and an Event ID. Events within a case should be ordered or include an Event Order or Event Timestamp to correctly determine the sequence of activities. Process logs can contain more information such as attributes of cases, event, and other items. Key-value pairs are the best way to store event logs, since they are flexible and extensible. Figure 4 shows a fragment of a process log.

case id	event id	properties				
		timestamp	activity	resource	cost	...
1	35654423	30-12-2010:11.02	register request	Pete	50	...
	35654424	31-12-2010:10.06	examine thoroughly	Sue	400	...
	35654425	05-01-2011:15.12	check ticket	Mike	100	...
	35654426	06-01-2011:11.18	decide	Sara	200	...
	35654427	07-01-2011:14.24	reject request	Pete	200	...
2	35654483	30-12-2010:11.32	register request	Mike	50	...
	35654485	30-12-2010:12.12	check ticket	Mike	100	...
	35654487	30-12-2010:14.16	examine casually	Pete	400	...
	35654488	05-01-2011:11.22	decide	Sara	200	...
	35654489	08-01-2011:12.05	pay compensation	Ellen	200	...

Figure 4. Fragment of a Process Log (Wil M. P. van der Aalst. 2011)

Process log files are often encoded as an XML file standard named XES (eXtensible Event Stream). This paper shows the use of XES files. Figure 5 shows a fragment of an XES file.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- This file has been generated with the OpenXES library. It conforms -->
<!-- to the XML serialization of the XES standard for log storage and -->
<!-- management. -->
<!-- XES standard version: 1.0 -->
<!-- OpenXES library version: 1.0RC7 -->
<!-- OpenXES is available from http://www.openxes.org/ -->
<log xes.version="1.0" xes.features="nested-attributes" openxes.version="1.0RC7" xmlns="http://www.xes-standard.org/">
    <extension name="Metadata_Time" prefix="meta_time" uri="http://www.xes-standard.org/meta_time.xesext"/>
    <extension name="Lifecycle" prefix="lifecycle" uri="http://www.xes-standard.org/lifecycle.xesext"/>
    <extension name="Metadata_Lifecycle" prefix="meta_life" uri="http://www.xes-standard.org/meta_life.xesext"/>
    <extension name="Organizational" prefix="org" uri="http://www.xes-standard.org/org.xesext"/>
    <extension name="Metadata_Organizational" prefix="meta_org" uri="http://www.xes-standard.org/meta_org.xesext"/>
    <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
    <extension name="Metadata_Concept" prefix="meta_concept" uri="http://www.xes-standard.org/meta_concept.xesext"/>
    <extension name="3TU metadata" prefix="meta_3TU" uri="http://www.xes-standard.org/meta_3TU.xesext"/>
    <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
    <extension name="General metadata" prefix="meta_general" uri="http://www.xes-standard.org/meta_general.xesext"/>
    <extension name="Semantic" prefix="semantic" uri="http://www.xes-standard.org/semantic.xesext"/>
    <global scope="trace">
        <date key="REG_DATE" value="1970-01-01T00:00:00.000+01:00"/>
        <string key="AMOUNT_REQ" value="UNKNOWN"/>
        <string key="concept:name" value="UNKNOWN"/>
    </global>
    <global scope="event">
        <date key="time:timestamp" value="1970-01-01T00:00:00.000+01:00"/>
        <string key="lifecycle:transition" value="UNKNOWN"/>
        <string key="concept:name" value="UNKNOWN"/>
    </global>
    <classifier name="Activity classifier" keys="concept:name lifecycle:transition"/>
    <classifier name="Resource classifier" keys="org:resource"/>
    <trace>
        <date key="REG_DATE" value="2011-10-01T16:39:55.798+02:00"/>
        <string key="concept:name" value="173772"/>
        <string key="AMOUNT_REQ" value="3000"/>
        <event>
            <string key="org:resource" value="112"/>
            <string key="lifecycle:transition" value="COMPLETE"/>
            <string key="concept:name" value="A_SUBMITTED"/>
            <date key="time:timestamp" value="2011-10-01T16:39:55.798+02:00"/>
        </event>
        <event>
            <string key="org:resource" value="112"/>
            <string key="lifecycle:transition" value="COMPLETE"/>
            <string key="concept:name" value="A_PARTLYSUBMITTED"/>
            <date key="time:timestamp" value="2011-10-01T16:39:55.898+02:00"/>
        </event>
        <event>
            <string key="org:resource" value="112"/>
            <string key="lifecycle:transition" value="COMPLETE"/>
            <string key="concept:name" value="A_DECLINED"/>
            <date key="time:timestamp" value="2011-10-01T16:40:31.075+02:00"/>
        </event>
    </trace>
```

Figure 5. Fragment of an XES file

PROCESS DISCOVERY

The following steps describe our process discovery journey:

1. Filter out incomplete cases and useless events. Activities that involve human resource efforts often have three lifecycle stages: schedule, start, and complete. Filtering out schedule stage does not hurt the performance of process model and makes the discovery process efficient.
2. Combine an activity and its lifecycle stage into a new column named Event. For example, activity *W_Valideren aanvraag* and its two stages START and COMPLETE are treated as two different events *W_Valideren aanvraag(START)* and *W_Valideren aanvraag(COMPLETE)* after combination.
3. Sort the event logs based on the event timestamp and create a new column named VISIT to record event sequence by each case. Discover process model using sequence discovery algorithm and the code snippet as follows. The SAS SEQUENCE procedure performs multiple item sequence discovery on transaction data. For example, the SEQUENCE procedure would identify something like, “Of those customers who purchase a new computer, 25% of them will purchase a laser printer in the next quarter.” Sequence discovery is not limited to customer purchase analysis, if you treat each event as item and event sequence as customer visit, then you can use sequence discovery to find out sequence relationship between events. To perform a sequence discovery with SAS, you must first run the ASSOC procedure to create an output data set of the assembled items. The ITEMS= and NITEMS= options are set to discover sequences of pairs of events and the support option is set to return all discovered sequences.

```
proc sort data=event_logs;
   by case_id event_order;
run;

data event_logs;
   set event_logs;
   by case_id event_order;
   if first.case_id then visit = 0;
   visit + 1;
   retain visit;
run;

proc dmdb data=event_logs dmdbcat=catSeq;
   id case_id;
   class event;
run;

proc assoc data=event_logs dmdbcat=catSeq items=2 out=assocOut
support=0;
   customer case_id;
   target event;
run;

proc sequence data=event_logs dmdbcat=catSeq assoc=assocOut out=seqOut
nitems=2 support=0;
   customer case_id;
   target event;
   visit visit/window=1;
run;
```

4. Verify the discovered model and improve or correct it manually. This step is necessary and very important, since the discovery algorithm is based on frequency rather than business knowledge. Some high-frequency connections between events will be included in the model even though they are not reasonable from a business perspective. For such unreasonable connections, you can drop them

manually from the model. Conversely, some insignificant connections might be excluded due to low frequency but should be included from business perspective.

PROCESS REPLAY

Replay uses event logs and process model as input, and event logs are “replayed” on top of the process model. Discrepancies between logs and a model can be detected, and then quantified by replaying the log. By using visualization, analysts can clearly see which activities have most deviations and which activities should be improved. If the Log data record contains an activity timestamp, then replay can also be used to detect bottlenecks.

There are several ways to measure fitness, the degree by which event logs conform to process model. The most naive approach is to count case quantities that can be fully replayed by the model from Start to End, divide the total case count, and calculate fitness. However, trace-level fitness is too strict and even if a case has 99 of 100 events matched with the model, the fitness is still zero. Under most conditions, trace-level fitness cannot differentiate the fitness of process models, so event-level fitness is more common.

Event-level fitness formula is Fitness = ½ * (1 - *m*/*c*) + ½ * (1 - *r*/*p*), here *m* signifies missing tokens, *c* signifies consumed tokens, *r* signifies remaining tokens, and *p* signifies produced tokens (van der Aalst, 2011).

To illustrate the algorithm of event-level fitness, we will replay two cases on top of a simple model. Table 1 includes the cases to be replayed and Figure 6 shows the process model. The process model is drawn in Petri Nets format, but its process model is the same as Figure 3. All events are represented with single letters.

Case ID	Trace
1	{a, b }
2	{a, b, d}

Table 1. Cases to be replayed

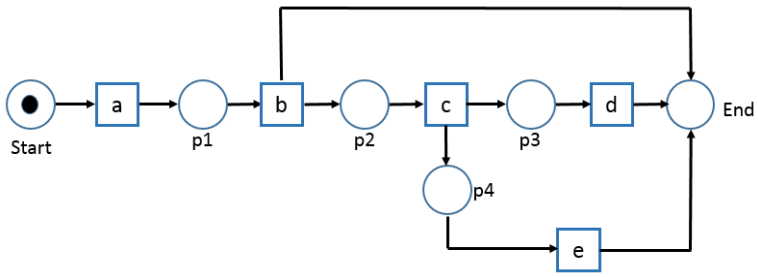

Figure 6. Process Model to Be Replayed

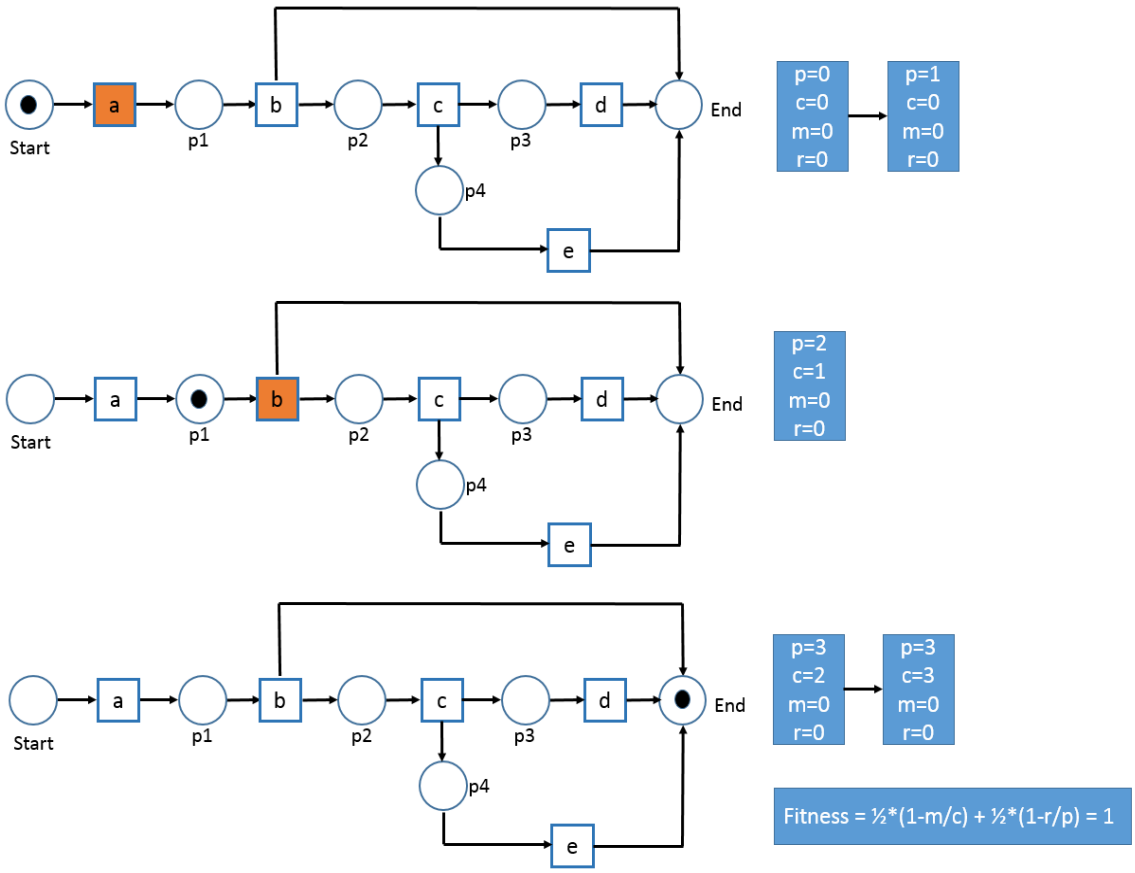

Figure 7. Replay Case 1 on Top of Process Model

Figure 7 illustrates the process of replaying *case* 1= {*a*, *b*}. Initially, $p = c = 0$ and all places are empty. Then the environment produces a token for place *start* and the *p* counter is updated: $p = 1$. The first event (*a*) can be replayed. After firing *a*, we have $p = 2$, $c = 1$, $m = 0$, and $r = 0$. Now the second event (*b*) can be replayed. After firing *b*, we have $p = 3$, $c = 2$, $m = 0$, and $r = 0$. In the final state End, the environment consumes the token from place end. Hence, the final result is $p = c = 3$ and $m = r = 0$.

$$Fitness\ (case\ 1) = ½ * (1 - m/c) + ½ * (1 - r/p) = 1$$

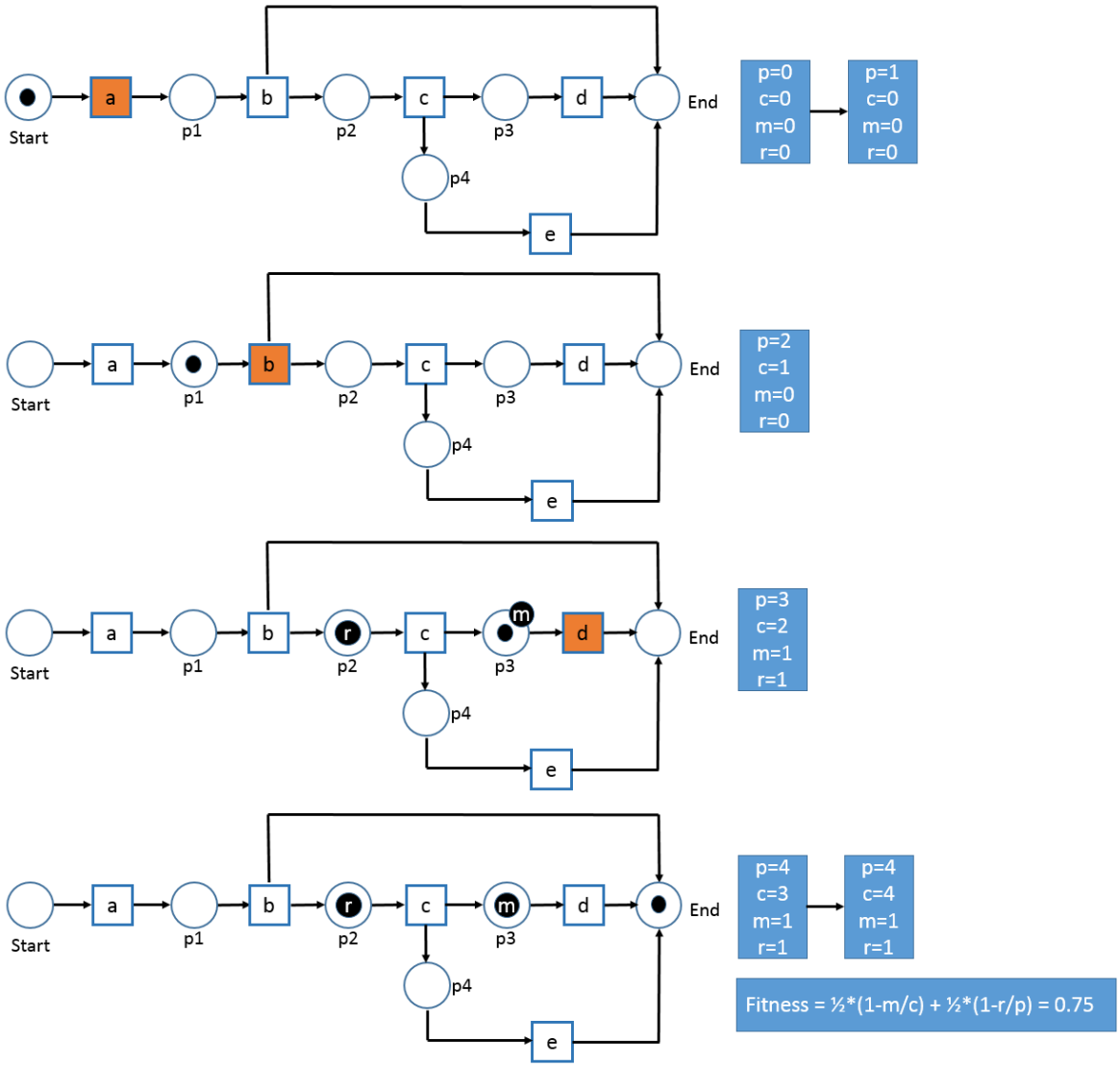

Figure 8. Replay Case 2 on Top of Process Model

Figure 8 illustrates the process of replaying *case* 2 = {*a*, *b*, *d*}. Initially, *p* = *c* = 0 and all places are empty. Then the environment produces a token for place start and the *p* counter is updated: *p* = 1. The first event (*a*) can be replayed. After firing a, we have *p* = 2, *c* = 1, *m* = 0, and *r* = 0. The second event (*b*) can be replayed. After firing *b*, we have *p* = 3, *c* = 2, *m* = 0, and *r* = 0. The current token is either put at place *End* or at place *p2*. Now we try to replay the third event (*d)*. This is not possible, because transition *d* is not enabled. To fire *d*, we need to add a token to place *p3* and record the missing token. The *m* counter is incremented and we tag place *p3* to remember that a token was missing. Meanwhile, neither token *p2* nor *End* is consumed, so we record one of them as the remaining token, and the *r* counter is incremented. The *p* and c counter are updated as usual. Therefore, after firing *d*, we have *p* = 4, *c* = 3, *m* = 1, and *r* = 1. In the final state, the environment consumes the token from place End. Hence, the final result is *p* = *c* = 4, and *m* = *r* = 1.

$$Fitness\ (case\ 2) = \tfrac{1}{2} * (1 - m/c) + \tfrac{1}{2} * (1 - r/p) = 0.75$$

IMPLEMENTATION

The implementation is presented in SAS code. Readers can contact the authors for more information. Some steps are displayed as macros to condense the content of the paper. Contact the authors for copies of the macros.

LOAD PROCESS LOG

This section shows how to load XES-format event logs in to several SAS data sets and flatten all information into one wide table. We focused on canceled cases where no offer was made from data in the BPI Challenge 2012 data as an example to illustrate the procedure with an understandable sample.

```
libname datalib '.';
filename  BPIChall '\\SGF2016\Data\BPI_Challenge_2012.xes';
filename  SXLEMAP '\\SGF2016\Data\bpi_challenge_2012.map';

%PM_Load_Data(
   In_XES_File = BPIChall,
   In_XML_MAP_File = SXLEMAP,
   Out_Event_Log_DS = datalib.bpi_challenge_2012);

proc sql;
   create table cancelled_cases as
   select distinct case_id from datalib.bpi_challenge_2012
   where activity eq 'A_CANCELLED'
     and case_id not in (select distinct case_id from
     datalib.bpi_challenge_2012 where substr(activity,1,2) eq 'O_');

   create table datalib.cancelled_cases_logs as
   select * from datalib.bpi_challenge_2012
   where case_id in (select case_id from cancelled_cases);
quit;
```

DATA EXPLORATION AND MANIPULATION

This section shows how to get basic information about raw event logs, such as case quantities, event quantities, and so on.

```
%PM_Event_Summary(In_Event_Log_DS = datalib.cancelled_cases_logs);
```

```
case_count=1167
event_count=22291
resource_count=59
activity_count=8
end_state_count=2
trace_count=178
```

Output 1. Output from %PM_Event_Summary

Next, we combined the activity and its lifecycle stage into a new column event, and removed the "SCHEDULE" stage.

```
proc sql;
   create table datalib.cancelled_cases_logs2 as
   select case_id, event_order, cats(activity,'(', lifecycle_transition,
')') as event length = 50, org_resource, amount_req, reg_date, dt_activity
   from datalib.bpi_challenge_2012
   where case_id in (select case_id from cancelled_cases)
     and lifecycle_transition ne 'SCHEDULE';
quit;
```

```
filename tracesum 'freq_traces.csv';
%PM_Trace_Frequency(
   In_Event_Log_DS = datalib.cancelled_cases_logs2,
   In_Event_Column_Name = event,
   Out_Trace_Freq_DS = datalib.freq_traces,
   Out_Trace_Network_DS = datalib.trace_network,
   Out_Trace_Network_CSV_File = tracesum);
```

The process model of full logs is displayed in Figure 9, which was generated using the freq_traces.csv data file and the ProcessFlow_Filter.html file that we wrote for process flow visualization. This utility includes fields to control the nodes and links based on frequencies. This is used to focus the view on the core business process and select areas for attention.

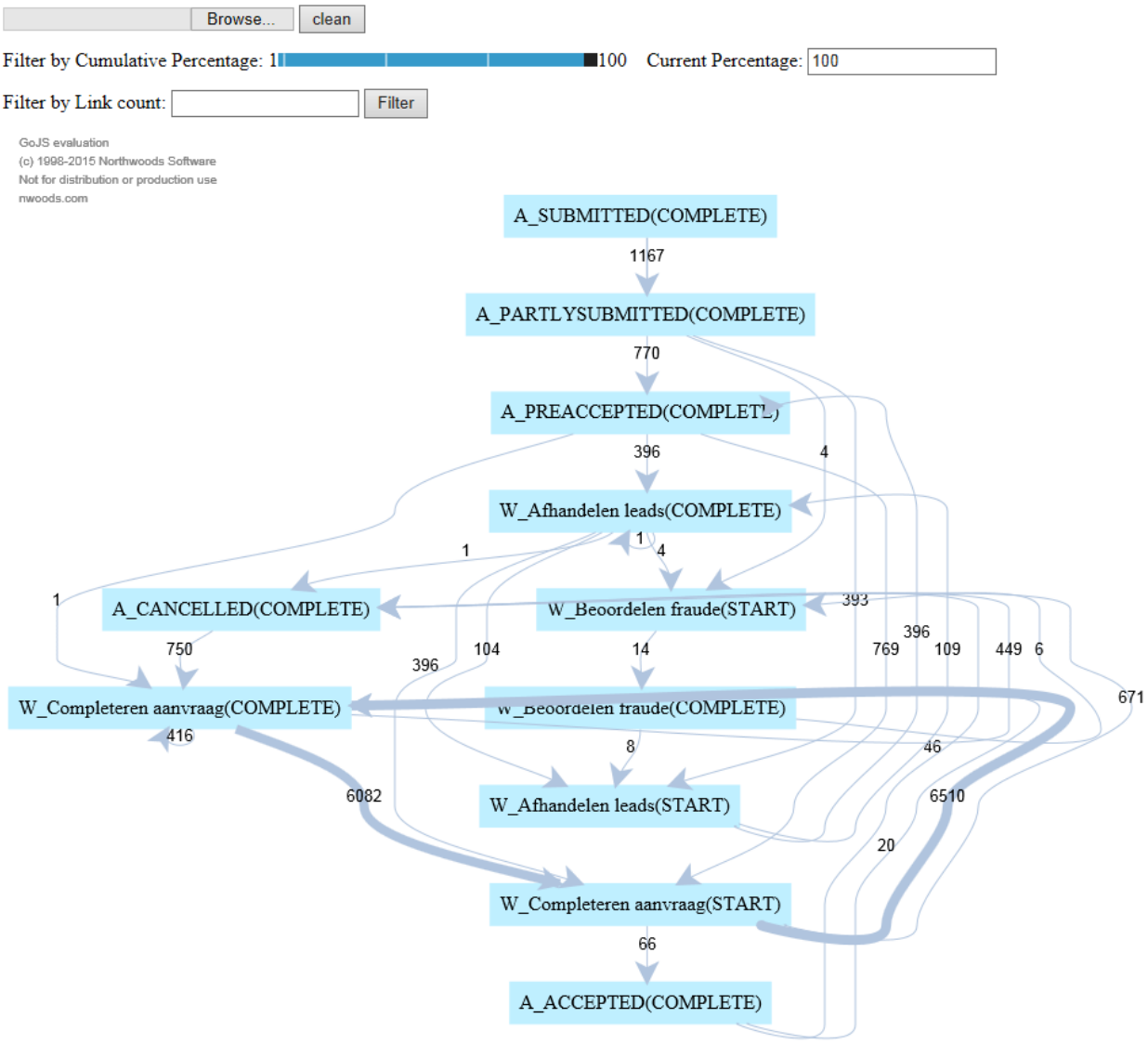

Figure 9. Process Model of Full Logs

This utility includes a slider and input fields to modify the display. We used these controls to generate a display that best illustrates the core process flow.

In Figure 9, arrows indicate transitions from one activity to the next. Line thickness indicates frequency of occurrences of that transition. From this figure, we found that most frequent connections exist between

W_Completeren aanvraag(START) and *W_Completeren aanvraag(COMPLETE)*, so improving performance of activity *W_Completeren aanvraag* most result in enhanced performance of this process.

DISCOVER PROCESS MODEL

The third section is to discover the model and calculate its fitness. The core discovery code uses SEQUENCE procedure process described earlier.

```
filename model 'discovered_model.csv';
filename modeldif 'discovered_model_diff.csv';
%PM_Discovery
(
   In_Event_Log_DS = datalib.cancelled_cases_logs2,
   In_Trace_Network_DS = datalib.trace_network,
   In_Assocs_Support = 2,
   In_Assocs_PCTSUP = 0,
   In_Sequence_Support = 2,
   Out_Model_DS = datalib.discovered_model,
   Out_Fitness_DS = datalib.discovered_model_fitness,
   Out_Graph_CSV_File = model,
   Out_Diff_Graph_CSV_File = modeldif
  ) ;
```

```
    event-level fitness is 99.99%
```

Output 2. Output from %PM_Discovery

VERIFICATION

Here we show you how to verify model manually and evaluate model fitness after manual correction. After further analysis on the discovered model, we thought one self-loop from *W_Completeren aanvraag(COMPLETE)* to itself is not reasonable and should be removed, because all human efforts involved activities should first start, then complete, so we deleted this link programmatically from the discovered model and re-evaluate performance of final model as Figure 10 shows.

```
data datalib.discovered_model;
   set datalib.discovered_model;
   if from_activity eq 'W_Completeren aanvraag(COMPLETE)' and to_activity
eq 'W_Completeren aanvraag(COMPLETE)' then delete;
run;

filename fnlmodel 'final_model.csv';
filename fnlmddif 'final_model_diff.csv';
%PM_Model_Evaluation(
   In_Trace_Network_DS = datalib.trace_network,
   In_Process_Model_DS = datalib.discovered_model,
   Out_Fitness_DS = datalib.discovered_model_fitness,
   Out_Graph_CSV_File= fnlmodel,
   Out_Diff_Graph_CSV_File=fnlmddif);
```

```
    event-level fitness is  98.09%
```

Output 3. Output from %PM_Model_Evaluation

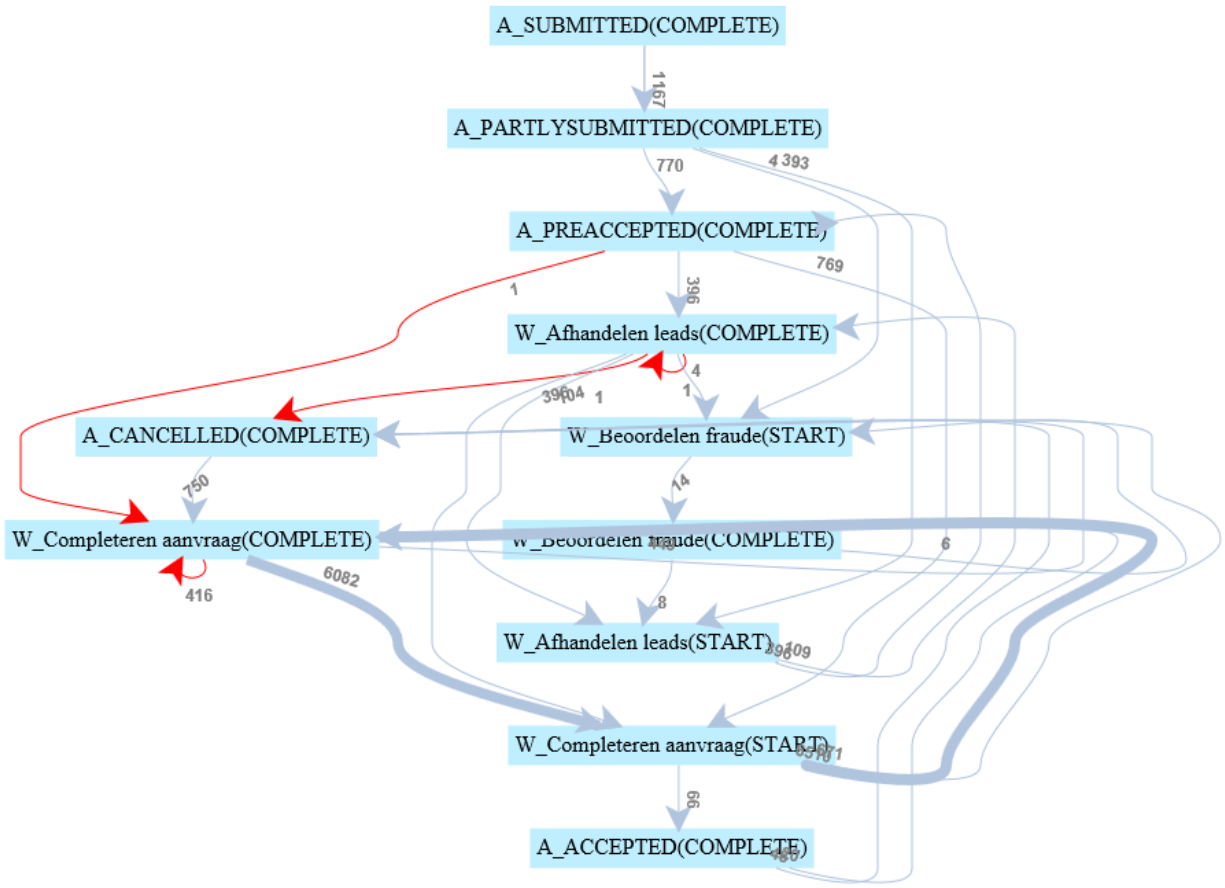

Figure 10. The final discovered process model

Figure 10 was generated by using ProcessFlow_noFilter.html and data final_model_diff.csv. In Figure 10, all links included in full logs but missed from the discovered model are displayed in red. If you want to see the model only, and don't want to see any missed links, you can open data final_model.csv in ProcessFlow_noFilter.html.

CONVERT PROCESS MODEL INTO BPMN FILE

Here we show you how to convert process model into BPMN format.

```
%PM_Convert_Model_to_BPMN(
   In_Process_Model_DS = datalib.discovered_model,
   In_With_GateWay = no,
   Out_BPMN_DS = datalib.bpmn);

filename  BPMNFN 'bpi_challenge_2012_bpmn.bpmn';
%PM_Export_Model_to_BPMN_File(
   In_BPMN_Model_DS = datalib.bpmn,
   In_Process_Name = BPI_Challenge_2012_Demo,
   Out_BPMN_File = BPMNFN);
```

RUN CONFORMANCE CHECK

The sixth section is to run conformance check. We use logs of canceled cases without making an offer as data to be checked.

```
proc sql;
   create table cancelled_cases as
```

```
    select distinct case_id from datalib.bpi_challenge_2012
    where activity eq 'A_CANCELLED'
      and case_id not in (select distinct case_id from
       datalib.bpi_challenge_2012 where substr(activity,1,2) eq 'O_');

    create table completed_cases_logs as
    select case_id, event_order, org_resource, amount_req, reg_date,
    cats(activity,'(', lifecycle_transition, ')') as event length = 50,
    dt_activity
    from datalib.bpi_challenge_2012
    where case_id in (select case_id from cancelled_cases)
      and lifecycle_transition ne 'SCHEDULE';
quit;

filename BPMNFN 'bpi_challenge_2012_bpmn_exported.bpmn';
filename BPMNMAP '\\SGF2016\Data\bpmn_xml_map.map';
%PM_Compliance_in_Batch(
    In_BPMN_XML_File = BPMNFN,
    In_BPMN_XML_Map_File = BPMNMAP,
    In_Trace_DS = completed_cases_logs,
    In_TraceID_Column_Name = case_id,
    Out_PetriNet_DS = process_model_petrinet,
    Out_Replay_Full_Log_DS = datalib.compliance_of_full_logs,
    Out_Fitness_Full_Log_DS = datalib.compliance_fitness_of_full_logs,
    Out_Replay_Trace_DS = datalib.compliance_of_cases,
    Out_Fitness_Trace_DS = ,
    Out_Deviated_Step_DS = datalib.compliance_problematic_steps);

filename compcsv 'compliance_result.csv';
%PM_Compliance_Data2CSV(
    In_Trace_DS = completed_cases_logs,
    In_TraceID_Column_Name = case_id,
    In_Deviated_Step_DS = datalib.compliance_problematic_steps,
    In_Process_Model_DS = datalib.discovered_model,
    In_Color = red,
    Out_Graph_CSV_File = compcsv);
```

Table Out_Fitness_Full_Log_DS saves the overall fitness of conformance check and it is 98.09%, which is same as the event-level fitness of the final discovered model. Figure 11 was generated by using ProcessFlow_noFilter.html and data compliance_result.csv.

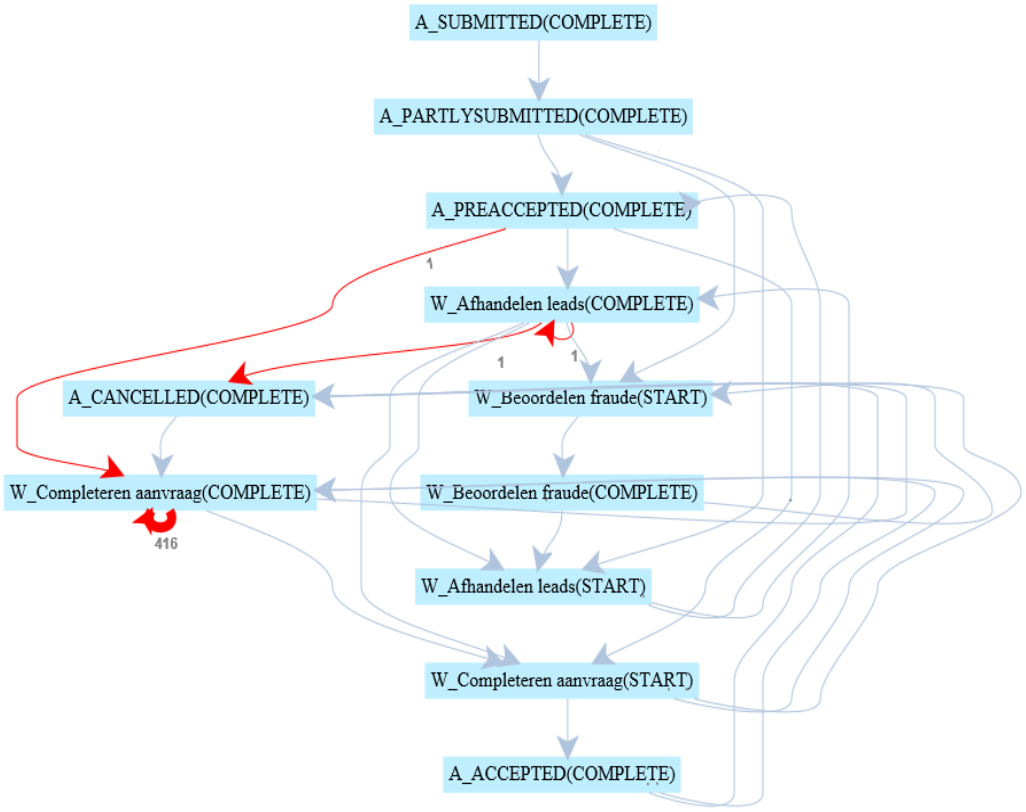

Figure 11. Deviations found by conformance check

DIAGNOSE DEVIATIONS

Here we show you how to diagnose deviations, which is necessary for business enhancement. From Figure 11, we noticed that there were 416 links marked as deviations and we'd like to analyze the root reasons why, so many deviations happened. In this paper, we show you an example on how to find out common patterns of deviations by using decision tree method.

```
proc sql;
   create table labelled_cases as
   select completed_cases_logs.case_id, event, event_order, org_resource,
          dt_activity, reg_date, amount_req,
          max(count, 0) as deviated_flag,
          intck('DTDAY', reg_date, dt_activity) as duration
   from completed_cases_logs
   left join datalib.compliance_problematic_steps
   on cats(completed_cases_logs.case_id,'') =
      compliance_problematic_steps.case_id
     and cats(completed_cases_logs.event_order,'') =
      compliance_problematic_steps.token
   order by case_id, event_order;
quit;

proc hpsplit data = labelled_cases missing=popularity;
  criterion fastchaid ;
  input event org_resource duration / level = nom;
  target deviated_flag  / level = nom;
  prune entropy;
run;quit;
```

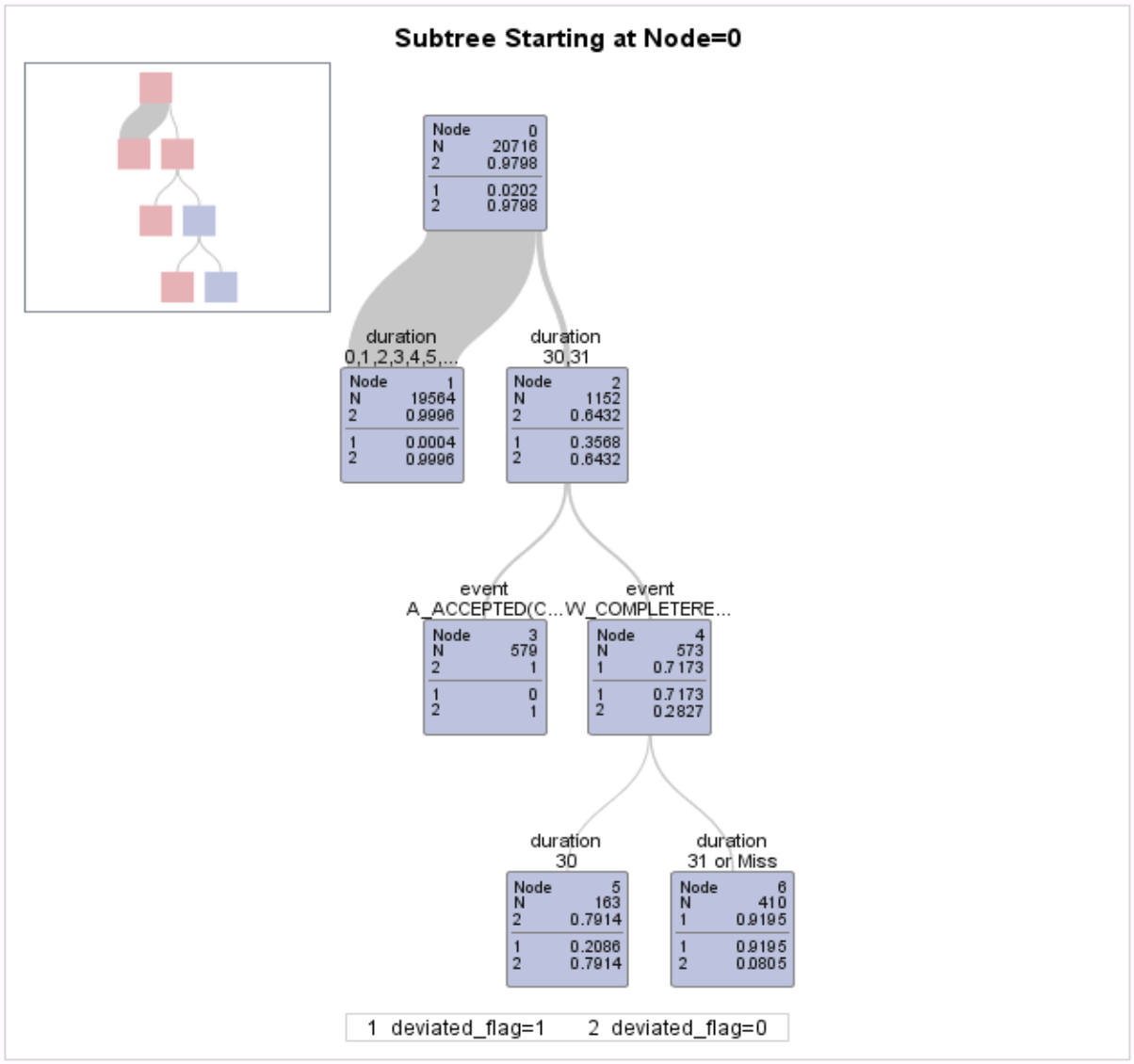

Figure 12. Decision tree of deviations

From Figure 12, we found that deviations are highly related to duration. If the duration is equal or larger than 31 days, then the probability of deviation is more than 90%, so we suspected there might be some rules related to duration and we did an analysis on case cancellation and duration.

```
%PM_Analyze_Cases(
   In_Event_Log_DS = datalib.cancelled_cases_logs,
   Out_Case_Summary_DS = datalib.case_summary,
   Out_Case_Duration_DS = datalib.va_case_endpoint_by_duration,
   Out_Case_Duration_Summary_DS = datalib.va_case_endpoint_by_duration2);
```

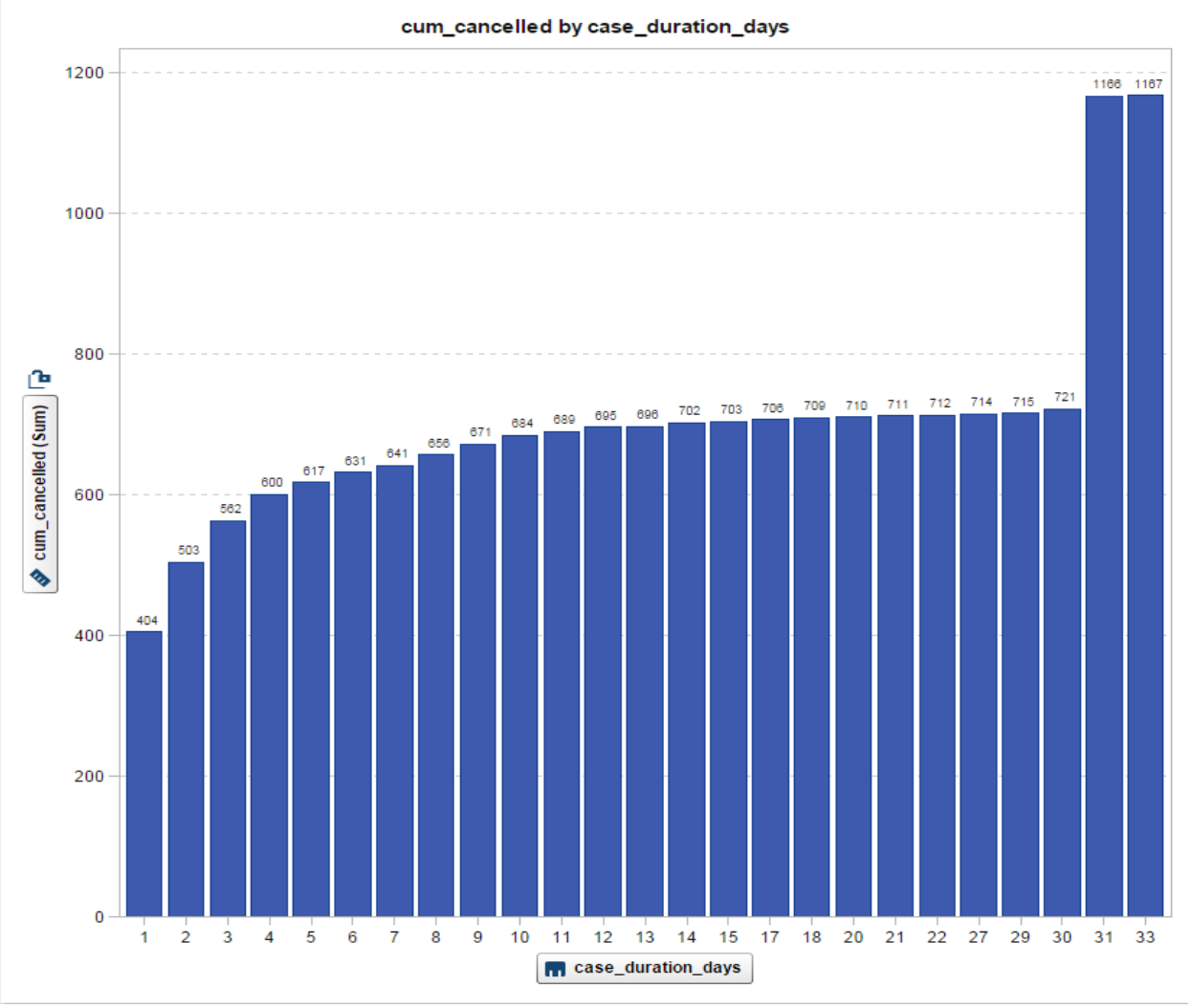

Figure 13. Cancellation of Loan Applications by Duration

Figure 13 was generated by using SAS® Visual Analytics and data set Out_Case_Duration_Summary_DS. From Figure 13, we see that there is an obvious increase in case cancellation at Day 31 and Day 33, which might be a company policy that all uncompleted cases should be forcedly terminated after 30 days.

CONCLUSION

Business Process Improvement is a popular topic; however, very little attention has been given to mining event logs. Analysis of patterns hidden in event logs is key to improvements in the efficiency of existing business processes. Process mining is a new field focused on process data analysis that leverages algorithms found in traditional data mining combined with several unique algorithms. In this paper, we demonstrated how to use SAS to seamlessly integrate process mining and data mining through a case study. SAS users now have the power to know their process.

REFERENCES

Blickle, Tobias, Helge Hess, Joerg Klueckmann, Mike Lees, and Bruce Williams. 2010. *Process Intelligence for Dummies*, Indiana, Indianapolis: Wiley Publishing, Inc.

van der Aalst, Wil M. P.. 2011. *Process Mining*. Heidelberg, Germany: Springer.

ACKNOWLEDGMENTS

The author would like to thank Jet (Zhongjian) Gong from SAS Institute for his help with the process flow visualization. Also, Robert Chu and Xudong Sun from SAS Institute for reviewing and providing feedback on this paper.

RECOMMENDED READING

van der Aalst, Wil M. P.. 2011. *Process Mining*. Heidelberg, Germany: Springer.

Bautista, Arjel D., Lalit Wangikar, and Syed M. Kumail Akbar. 2012. “Process-Mining Driven Optimization of a Consumer Loan Approvals Process.“ *The 2012 BPIC Challenge*. New York, NY: CKM Advisors.

Bose, R.P. Jagadeesh Chandra and Wil M.P. van der Aalst. 2012. “Process Mining Applied to the BPI Challenge 2012: Divide and Conquer While Discerning Resources.“ *The 2012 BPIC Challenge*. The Netherlands: Eindhoven University of Technology.

CONTACT INFORMATION

David R. Duling
SAS Institute Inc.
Cary, North Carolina, 27513
Email: david.duling@sas.com
Phone: 919-531-5267

Emily (Yan) Gao
SAS Institute Inc.
Beijing, China, 100102
Email: yan.gao@sas.com
Phone: +86 10-8319-3355