Business Process Intelligence

Report Assignment Part 2

Business Process Intelligence
course

Summer semester 2022
semester

prof. dr. ir. Wil van der Aalst
lecturer

Group 77: Niklas Föcking (423932), Filiz Günal (431174), David Wenderdel (423885)
students

July 4, 2022
submission date

Q1 Knowing the Event Log

(a) The following information are visualized in the “Explore event log”-view in ProM:

- Time period: 2010-01-13 08:40:25 - 2014-01-03 13:20:58
- Number of cases: 4,580
- Number of events: 21,348
- Number of activities (event classes): 14
- Number of trace variants: 226

The “Summary”-tab of the “Log Visualizer”-view provides the following statistics for each activity (see Table 1).

Activity	**Occurrences**	**Occ. as start actitity**	**Occ. as end actitity**
Take in charge ticket	5060	74	1
Resolve ticket	4983	2	10
Assign seriousness	4938	4384	-
Closed	4574	-	4557
Wait	1463	1	8
Require upgrade	119	-	3
Insert ticket	118	118	-
Create SW anomaly	67	1	-
Resolve SW anomaly	13	-	-
Schedule intervention	5	-	-
VERIFIED	3	-	1
INVALID	2	-	-
RESOLVED	2	-	-
DUPLICATE	1	-	-

Table 1: Statistics per activity

Now, we want to discuss three properties of the event log that can be derived from the previous statistics. We observe that *Take in charge ticket* occurs 5,060 times, and the event log contains 4,580 traces. Thus, we know that there is a loop in the process: it must be possible to take a ticket in charge several times. The same holds for *Resolve ticket* and *Assign seriousness.*

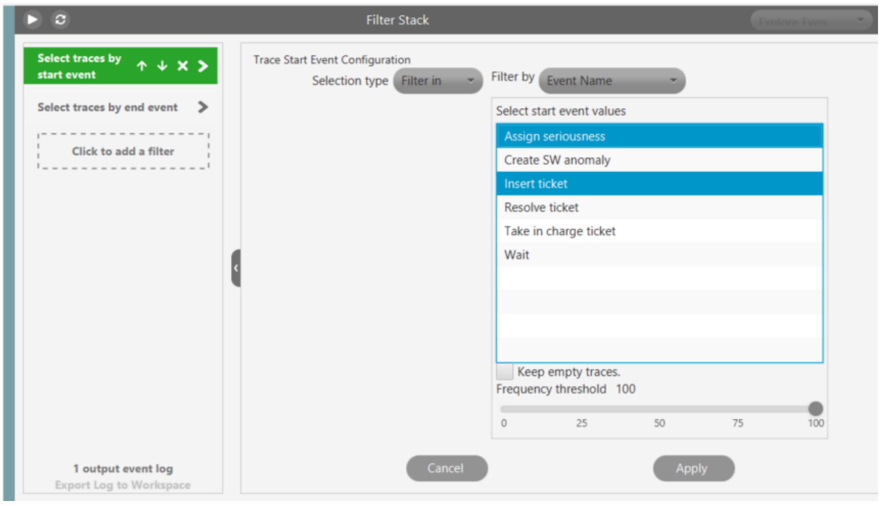
We observe that the log contains four activities that are very infrequent and do not follow the same naming convention because they contain only upper case characters: *VERIFIED*, *INVALID*, *RESOLVED*, and *DUPLICATE.* Without additional knowledge, one might assume that these activities are in the event log because of errors or manual interventions into a digital process (e.g. changing values in the underlying database directly). But these are only first assumption and have to be checked in later stages of the analysis.

From the frequencies of start and end activities, we observe that the log may contain incomplete traces. There are several activities, like *Resolve ticket*, *Take in charge ticket*, *Wait*, *Require upgrade*, *Create SW anomaly*, and *VERIFIED*, that are start respective end activities in a very small amount of traces.

(b) 1. As already mentioned in exercise a, there are a some activities that are start respective end activities in a very small number of traces. These are *Resolve ticket*, *Wait*, *Require upgrade*, *Create SW anomaly*, *Take in charge ticket*, and *VERIFIED*. Furthermore, there is a short description of the process in the introduction of the exercise sheet. This description states that tickets are archived after the activity *Closed* appears. After this activities, tickets are no longer in the queue of tickets that have to be solved. With this background information as well as the distribution of end activities in the traces, we assume that traces are only complete if they end with the activity *Closed*.

When looking at start activities, we observe that 4,384 of 4,580 cases start with *Assign seriousness*. Additionally, there are 118 cases that start with *Insert ticket*, and *Insert ticket* occurs exactly in 118 traces. Thus, when *Insert ticket* is contained in a trace, it is always a start activity. Therefore, we assume that traces that start with *Assign seriousness* or *Insert ticket* are complete. Traces that start with other activities are considered to be incomplete. All in all, incomplete traces are all traces that do not start with *Assign seriousness* or *Insert ticket* or which do not end with *Closed*.

2. To filter the log, we use the “Event filter on event log”-view of ProM. Here, we add two filters of the types “Select traces by start event” (see Figure 1) and “Select traces by end event”. In the configuration of the filters, we select the values according to part 1. We apply the filter and export the event log using the button “Export Log to Workspace”. The filtered log contains 4,479 traces. So, we removed 101 traces.



Trace Start Event Configuration

Selection type: Filter in

Filter by: Event Name

Select start event values:
- Assign seriousness
- Create SW anomaly
- Insert ticket
- Resolve ticket
- Take in charge ticket
- Wait

Keep empty traces.

Frequency threshold 100

Figure 1: The figure shows the filter that is applied for start activities.

(c) For question 1 and 2, we use the same methods to extract the results as in exercise a.

- Number of cases: 4,479
- Number of events: 20,937
- Number of activities (event classes): 14
- Number of trace variants: 193

Activity	**Occurrences**
Take in charge ticket	4949
Resolve ticket	4881
Assign seriousness	4918
Closed	4494
Wait	1402
Require upgrade	104
Insert ticket	114
Create SW anomaly	57
Resolve SW anomaly	8
Schedule intervention	5
VERIFIED	2
INVALID	1
RESOLVED	1
DUPLICATE	1

Table 2: Statistics per activity

We use a pie chart to visualize the frequencies of the ten most frequent traces (see Figure 2). We observe that the most frequent variant covers more than half of the cases of the log. Furthermore, the four most frequent variants cover together approx. 75 % of the cases in the log.

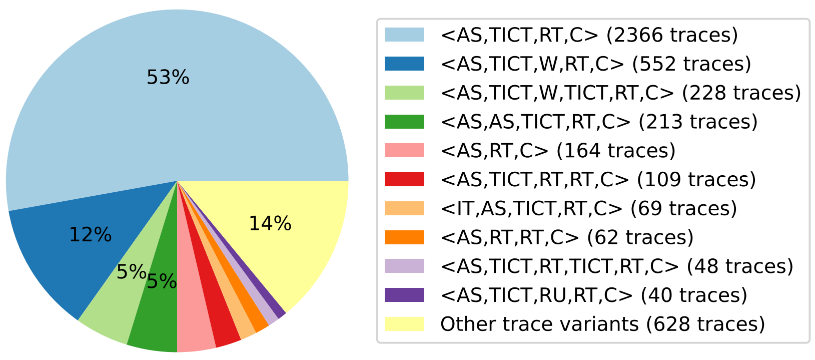
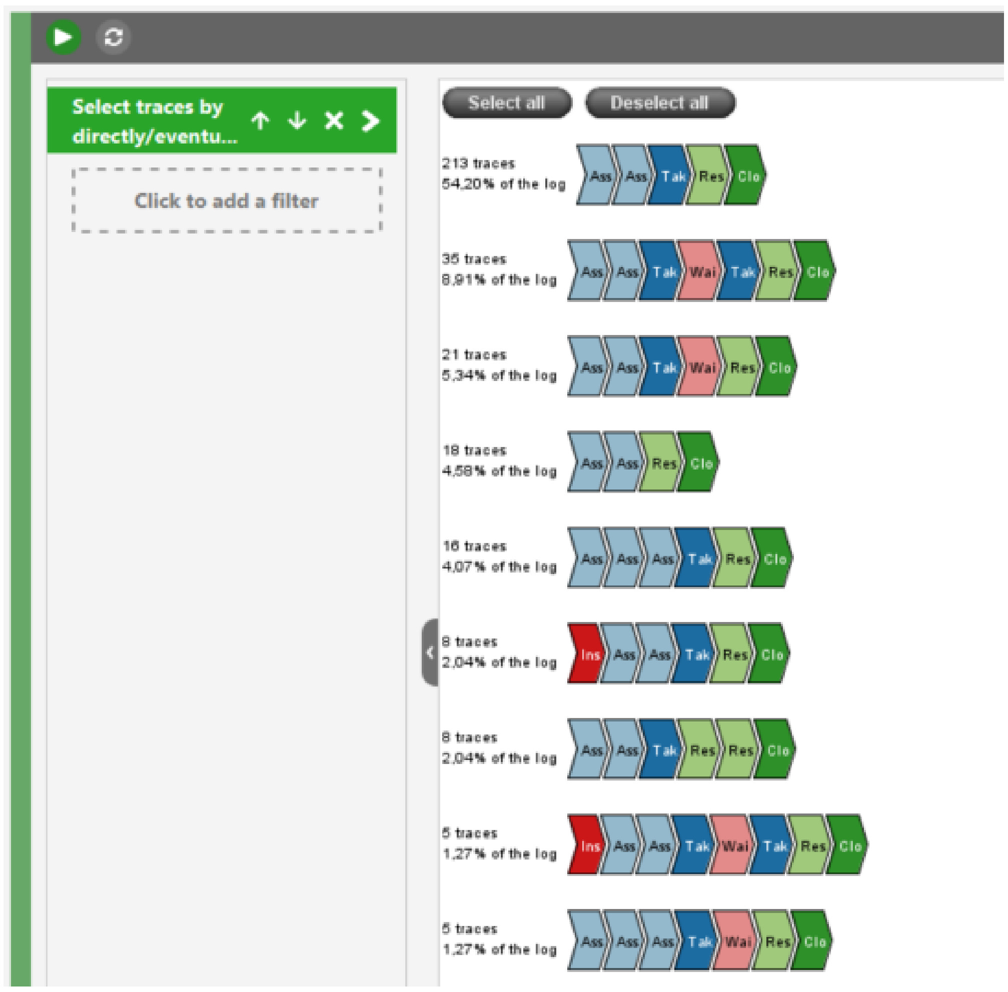


Figure 2: The figure shows the distribution of the ten most frequent traces. The activity names are abbreviated, i.e. we only use the first letter of each word in the activity name. For example, *Take in charge ticket* is transformed to *TICT*.

We use the plugin "Inductive visual miner" to compute the case duration statistics. There, we click "data analysis". The log has an average trace duration of approx. 41 days, a minimum case duration of approx. 31 days, and a maximum case duration of 60 days (see Figure 30 in the appendix).

To list the activities that appear multiple times in at least one of the traces, we perform the following approach. We open the view "Event filter on event log" and then add a filter of the type "Select traces by directly/eventually follows of events". As the attribute, we select "concept:name", and as the follow type, we select "Eventually followed". As reference and follower events, we always use the same activity label. So, we start with *Assign seriousness* and apply the filter (see Figure 3). If the filtered event log contains at least one trace, we know that there is a trace that contains the activity at least twice. We perform the procedure for every activity. The following activities occur multiple times in at least one trace: *Assign seriousness*, *Closed*, *Create SW anomaly*, *Require upgrade*, *Resolve SW anomaly*, *Resolve ticket*, *Take in charge ticket*, and *Wait*.



```
213 traces 54,20% of the log   Ass Ass Tak Res Clo
35 traces 8,91% of the log     Ass Ass Tak Wai Tak Res Clo
21 traces 5,34% of the log     Ass Ass Tak Wai Res Clo
18 traces 4,58% of the log     Ass Ass Res Clo
16 traces 4,07% of the log     Ass Ass Ass Tak Res Clo
8 traces 2,04% of the log      Ins Ass Ass Tak Res Clo
8 traces 2,04% of the log      Ass Ass Tak Res Res Clo
5 traces 1,27% of the log      Ins Ass Ass Tak Wai Tak Res Clo
5 traces 1,27% of the log      Ass Ass Ass Tak Wai Res Clo
```

Figure 3: Excerpt of the filtered log that contains only traces where *Assign seriousness* eventually follows *Assign seriousness*

(d) In Figure 4, we visualize the distribution of the case-attributes *Ticket type* and *Membership*. For each of the column charts, we use one KPI and one dimension. These are the PQL-Commands for the attribute *Membership*:

Dimension: `"case_table_csv"."MEMBERSHIP"`

KPI: `COUNT("case_table_csv"."MEMBERSHIP")`

These are the PQL-Commands for the attribute *Ticket type*:

Dimension: `"case_table_csv"."TICKET TYPE"`

KPI: `COUNT("case_table_csv"."TICKET TYPE")`

We observe that the company has approx. 2500 customers with a premium membership and more than 2000 customers with a gold membership. Regarding the types of the tickets, most tickets have the types *Question* or *Task*, whereas tickets of the type *Incident* are less common.

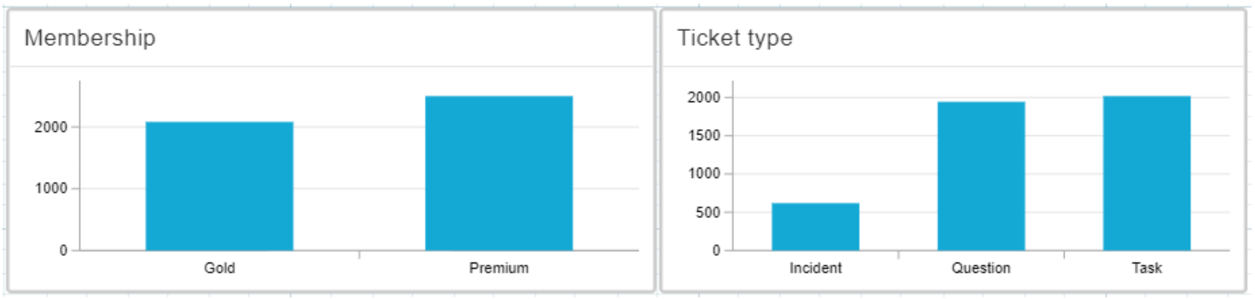


Figure 4: The distribution of the values for the case-attributes *Ticket type* and *Membership*

In Figure 5, we show the total workload per resource. We use the following dimension and the following KPI:

Dimension: `"event_table_csv"."RESOURCE"`

KPI: `COUNT("event_table_csv"."ACTIVITY")`

Res3, *Res4*, and *Res8* have the highest workload. In contrast to that, *Res5*, *Res6*, and *Res7* have a small workload.

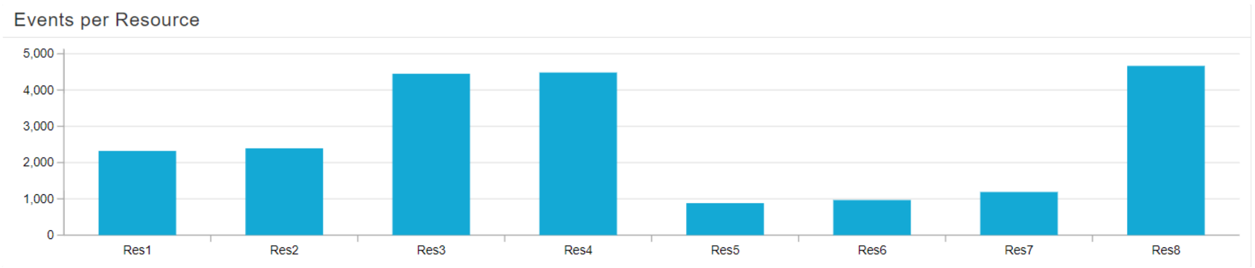


Figure 5: Total workload per resource

In Figure 6, we show the distribution of throughput times using a histogram. We use the following Dimension:

```
CALC_THROUGHPUT(ALL_OCCURRENCE['Process Start'] TO ALL_OCCURRENCE['Process
          End'], REMAP_TIMESTAMPS("event_table_csv"."TIMESTAMP", DAYS))
```

As the time resolution, we use days. We configure exactly ten bins. We observe a peak of throughput times that take 31-34 days. Additionally, there is a minor peak at throughput times of 43-49 days.

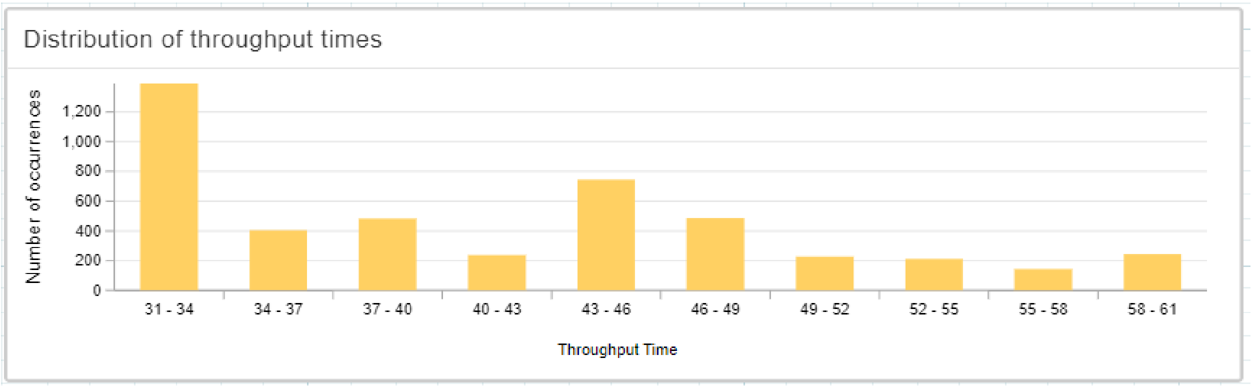


Figure 6: Distribution of the throughput times (in days) shown in Celonis.

Q2 Discovery

(a) As a first step, we create two sublogs as a basis for comparing the two time-frames. In order to achieve a filtering on event level, we apply plugin "Log Filtering" (see Figure 31 in the appendix). We set the plugin configuration to filter on event level and retain those events which have a timestamp between 13/01/2010 and 30/09/2012. The resulting log might still contain incomplete traces. Hence, we switch to view "Event filter on event log" and – as already done in exercise 1 – select only those traces that start either with *Insert ticket* or *Assign seriousness* and end with *Closed* (see Figure 32 in the appendix). We proceed analogously with the second sublog that contains only events at or after 01/10/2012. The final characteristics of both sublogs are depicted in Table 3.

	log-before	**log-after**
# unique activities	13	12
# cases	3,708	730
# trace variants	146	83

Table 3: Selected statistics for the two created sublogs.

(b) We would like to discover process models based on the given logs. Since there is not **the** one algorithm to do that, we try three different approaches – heuristic mining, state-based regions, and inductive mining. For heuristic mining, we use the plugin "Interactive

Data-aware Heuristic Miner (iDHM)". For inductive mining, we use the plugin "Mine Petri Net with Inductive Miner". For state-based regions, we use an approach that consists of two steps. First, based on the log, we create a transition system using the plugin "Mine transition system". Using the plugin "Convert to Petri Net using Regions", we create a Petri net based on the given transition system.

We test several different parameter settings for the different approaches. In order to compare the quality of the discovered models, we compute precision and fitness for all discovered process models using the plugin stated in the exercise description. The results for the best models per approach are shown in Table 4. All these models are depicted in the appendix (see Figure 33 ff.). Note that we provide the parameter settings of the different discovery runs through footnotes. As long as no specifications are made, the default settings are selected.

		log-pre-complete			**log-post-complete**	
		fitness (%)	precision (%)		fitness (%)	precision (%)
Heuristic	M_{pre-h_1}[1]	99.3	84.2	M_{post-h_1}[2]	98.8	92.2
	M_{pre-h_2}[3]	99.5	82.1	M_{post-h_2}[3]	98.7	92.5
Regions	M_{pre-r_1}	98.2	85.6	M_{post-r_1}	98.2	81.8
	M_{pre-r_2}[4]	95.4	80.6	M_{post-r_2}[4]	96.3	92.8
Inductive	M_{pre-i_1}[5]	94.5	89.4	M_{post-i_1}[6]	94.2	88.7
	M_{pre-i_2}[7]	100	45.7	M_{post-i_2}[7]	100	37.6

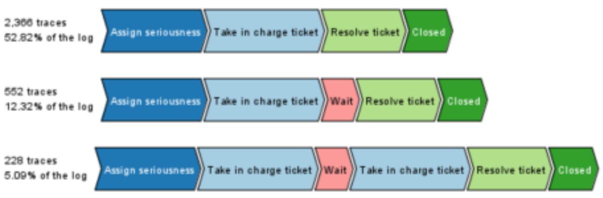
Table 4: Fitness and precision values for the discovered process models. As long as no specifications are made, the default settings are selected.

As specified in the exercise description, we select the models with the highest precision as M_{pre} and M_{post}. With respect to our statistics, that is M_{pre-i_1} for M_{pre} (see Figure 37) and M_{post-r_2} for M_{post} (see Figure 42). Please note that for readability reasons, we decided to move the figures into the appendix, although this was requested otherwise in the task description.

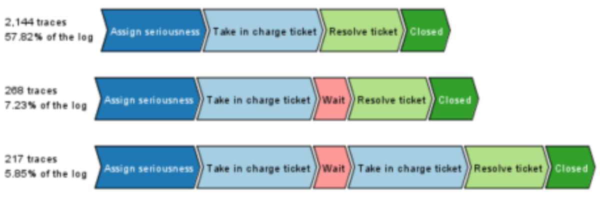
(c) 1. Having found the process models, we now consider the most frequent variants of the three different logs, which are shown in Figure 7. Looking at the three most-frequent variants of *log-complete* first (Figure 7a), we detect that the most frequent variant (consisting of four events) makes up more than 50% of the whole log, i.e. more than every second case has this behaviour. The next most frequent variant (with an additional "Wait" event in the middle of the trace) accounts for almost only 10 percent. It is easily recognizable that the most frequent variants of *log-complete* are equal to the ones of *log-pre-complete* (Figure 7b). Also

[1]frequency: 0, dependency: 0.045
[2]frequency: 0, dependency: 0.1
[3]frequency: 0, dependency: 0.5
[4]transition system size limit: 100, collection size limit: 20, merge states with identical inflow and outflow
[5]variant *IMf* with noise threshold of 0.1
[6]variant *IMf* with noise threshold of 0.2
[7]variant *IM*

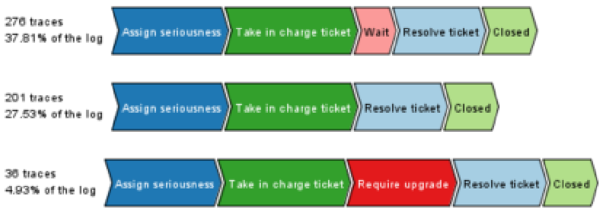
the distribution is similar but even more focused on the most frequent variant (approximately 58%). Looking at the frequencies, we realize that *log-pre-complete* contains a large portion of all traces while *log-post-complete* seems to be a smaller log. In *log-post-complete*, the two most frequent traces of *log-complete* respective *log-pre-complete* are the most frequent ones, too, but in reverse order (Figure 7c). Hence, it seems that there was a concept drift and the process behaviour changed.



(a) *log-complete*



(b) *log-pre-complete*



(c) *log-post-complete*

Figure 7: The three most frequent traces for logs *log-complete* (a), *log-pre-complete* (b) and *log-post-complete* (c) and their frequencies.

2. As a next step, we try to replay the most frequent trace of each log with the previously discovered (best) process models. The most frequent trace of *log-complete* and *log-pre-complete* is $\sigma_{compl} = \sigma_{pre} = \langle AS, TICT, RT, C\rangle$ (using the same abbreviations as in exercise 1c). The most frequent trace of *log-post-complete* is $\sigma_{post} = \langle AS, TICT, W, RT, C\rangle$. In order to give valid firing sequences, we labeled the transitions and places of the Petri net of M_{pre} (see Figure 37) and M_{post} (see Figure 42).

σ_{compl} **and** σ_{pre} **on** M_{pre} : $\langle t2, t3, t4, t5, t8, t10, t15, t17, t18, t19, t23\rangle$

σ_{compl} **and** σ_{pre} **on** M_{post} : $\langle t2, t6, t16, t30\rangle$

σ_{post} **on** M_{pre} : $\langle t2, t3, t4, t5, t8, t9, t11, t13, t14, t15, t17, t18, t19, t23\rangle$

σ_{post} **on** M_{post} : $\langle t2, t6, t12, t25, t41\rangle$

The most frequent traces of each (sub)log are replayable in both models.

(d) 1. M_{pre} starts with an optional execution of *Insert ticket*, followed by a parallel process part. Both parallel parts have one mandatory activity, namely *Take in charge ticket* respectively *Assign seriousness*. While the parallel part with *Take in charge ticket* is completed with the execution of this activity, *Assign seriousness* can optionally be followed by various activities. After that, both parallel sequences come back together again and must execute

Resolve ticket as well as *Closed* while some other activities are again optional. The shortest path through the petri net is $\langle t2, t3, t4$ *(Assign seriousness)*$, t5$ *(Take in charge ticket)*, $t8, t10, t15, t17, t18$ *(Resolve ticket)*$, t19, t23$ *(Closed)*$\rangle$.

2. The behaviour of M_{post} is a bit more difficult to capture, because it consists of several duplicated transitions, i.e. transitions that have the same visible label. At the beginning, the process can choose between *Insert ticket* and *Assign seriousness*. However, if the process executes *Insert ticket* first, it enforces the firing of *Assign seriousness* directly afterwards. After the execution of *Assign seriousness*, it can choose between a variety of possible activities. While most of them are optional, some very few are mandatory to reach the final place. This final place is only reachable by taking *Closed* before. In order to reach one of the *Closed* activities in turn, one has to execute *Resolve ticket* before. The shortest path through the petri net is $\langle t2$ *(Assign seriousness)*$, t4$ *(Resolve ticket)*$, t22$ *(Closed)*$\rangle$.

3. As previously described, both models have very similar mandatory activities. However, M_{pre} has one additional mandatory activity in comparison too M_{post}, namely *Take in charge ticket*. Both models have in common that there are no loops and also no activities that can be executed twice. Some key characteristics, such as that a process always starts with *Insert ticket* or *Assign seriousness* and always ends with *Closed* is modeled similarly in both Petri nets.

If we wanted to evaluate both models, we would probably choose model M_{pre} as the "better" one because it better balances the four forces which were presented in the lecture. In terms of precision and fitness, both models perform equally good as they achieve precision values around 90% and fitness values of even 95%. Considering the third force, i.e. generalization, the two models generalize in different ways. While M_{pre} uses many silent transitions to model optional behaviour, M_{post} uses multiple transitions with equal labels to generalize. However, both approaches are possible and generalize equally well. The difference between both models and the reason why we categorized M_{pre} as the "better" one is simplicity. M_{post} consists of almost twice as many transitions as M_{pre} and offers many decisions. In contrast, M_{pre} is much simpler to understand and has one "main line".

Q3 Conformance Checking

(a) We filter the log to contain only traces with *Ticket type = task* using the "Event Filter on Event log"-view. There, we add a filter of the type "Select traces by trace attributes" and filter by *Ticket type* with the operator *in* and the desired value *Task*. The filtered event log contains 2,018 traces, 10,283 events, 14 activities, and 201 trace variants. Using the filtered log, we mine a process tree using the plugin *Mine process tree with Inductive Miner*, select the variant *IMf* and a noise threshold of 0.2. The resulting process tree is depicted in Figure 8 using the graphviz-visualization. The corresponding Petri net is depicted in the appendix (see Figure 46).

(b) First, we convert the discovered process tree to a Petri net using ProM. To perform conformance checking, we use this Petri net as well as the original (unfiltered) log. We use

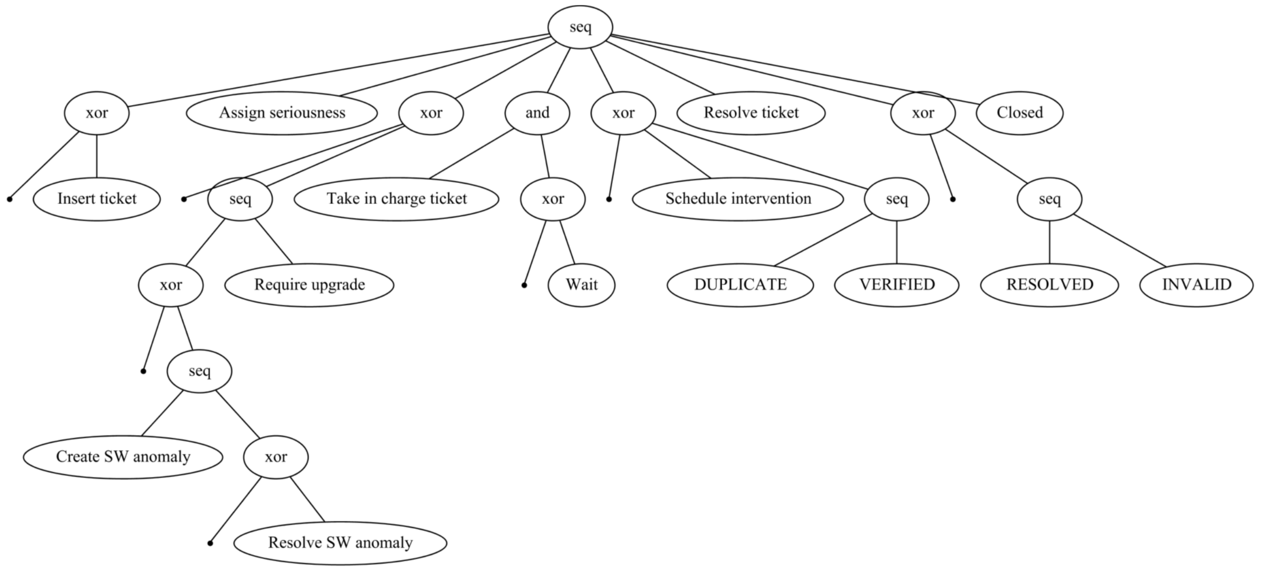


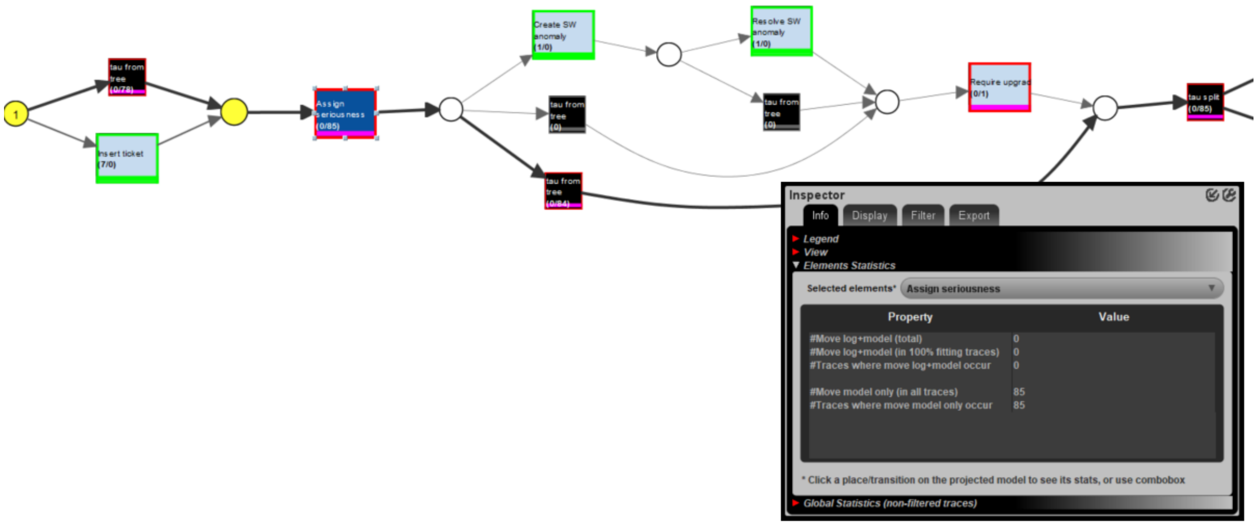
Figure 8: Process tree discovered using the filtered log and the Inductive Miner

the plugin "Multi-perspective Process Explorer" to derive the alignment-based fitness and precision (as in Q2). The fitness is 94.1%, the precision is 92%. We use the plugin "Replay a Log on Petri Net for Conformance Analysis" to derive the percentage of perfectly fitting traces. Therefore, after executing the plugin, we change to the view "Project Alignment to log". In the dropdown on the righthand side, we choose "Trace fitness" as the "Alignment Statistic". There appears a row with the title "#Cases with value 1.00". This row states that there are 3,008 perfectly fitting traces. So, $3,008/4,580 = 65.68\%$ of the traces are perfectly fitting.

Now, we apply Heuristic Miner ("Interactive Data-aware Heuristic Miner (iDHM)") on the orignal (not filtered) log. We choose Alpha Miner as the dependency heuristic and the values 0.053 and 0.872 for frequency and dependency, respectively. The new process model (see Figure 47 in the appendix) performs slightly better in terms of fitness and precision. The fitness for the new model improved to 95.3% while precision increased to 100%. Also the number of perfectly fitting traces improved to 3,193 (=69.71%).

(c) First, we want to find two transitions with a high amount of model moves. Therefore, we use the plugin "Replay a log on Petri Net for Conformance Analysis". As inputs, we use the discovered Petri net for exercise a and the log `log-flat`. After executing the plugin, the plugin visualizes performance information on the model. For example, for each transition, we can directly read the number of traces containing a model move for the given transition in the underlying alignments. The transitions that have the highest amounts of model moves are *Assign seriousness* (85 model moves) and *Take in charge ticket* (298 model moves).

In a first step, we have a deeper look in the model moves for the transition *Assign seriousness.* Thus, we create a filtered log that only contains traces that have a model move on this transition using the *filter*-tab of the plugin *Replay a log on Petri Net for Conformance Analysis.* An excerpt of the resulting process model with conformance annotations for the filtered log is depicted in Figure 9. According to the element statistics, there is no trace that contains a log move for the activity *Assign seriousness*. Therefore, because we know that all the filtered traces contain a model move, we know that all these traces do not contain the activity *Assign seriousness*. However, the model enforces the execution of the transition that corresponds to the activity *Assign seriousness*. One possible solution is to make *Assign seriousness* skip-able in the the model by introducing a silent transition. For a deeper analysis, we export the filtered log and have a look at the different variants. We observe that only three traces of the filtered log correspond to complete traces according to our definition from Q1. Because there are only very few deviations for completed traces, changing the model seems to be a poor option.



Selected elements* Assign seriousness

Property	Value
#Move log+model (total)	0
#Move log+model (in 100% fitting traces)	0
#Traces where move log+model occur	0
#Move model only (in all traces)	85
#Traces where move model only occur	85

* Click a place/transition on the projected model to see its stats, or use combobox

Global Statistics (non-filtered traces)

Figure 9: Process model with conformance annotations for traces that contain a model move on *Assign seriousness*

Next, we filter the log based on model moves for the activity *Take in charge ticket* using a "Movement containment filter". Here, it also holds that most of the traces simply does not contain the activity *Take in charge ticket*, but the model enforces *Take in charge ticket* to be executed in every valid firing sequence of the model (see excerpt of the model in Figure 10). We can solve this problem by making *Take in charge ticket* skip-able. Therefore, we would introduce a new silent transition. It has the same preset and the same postset as the activity labeled with *Take in charge ticket*.

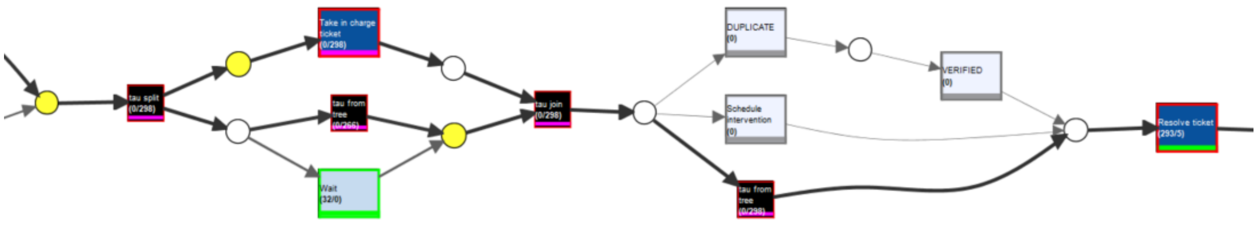
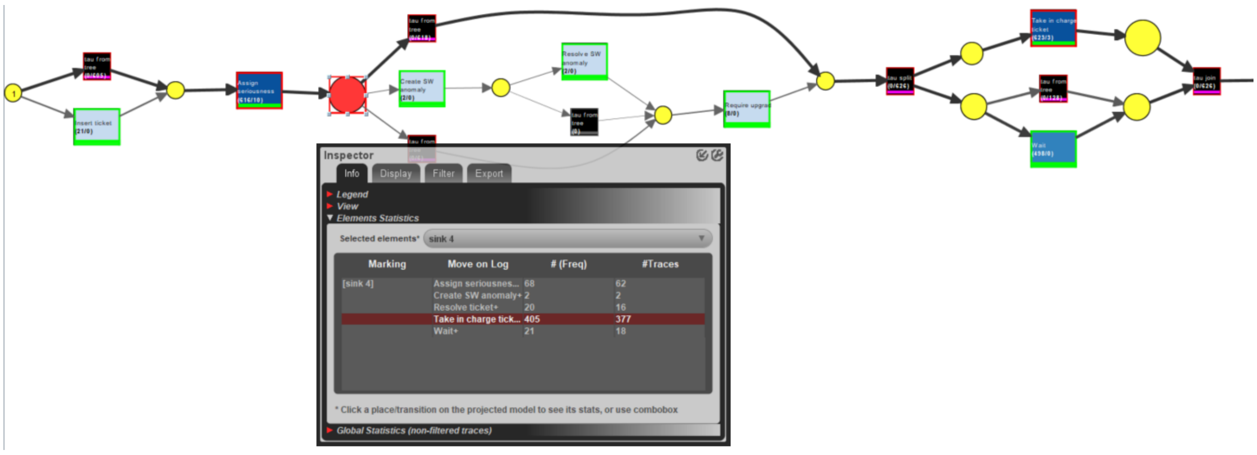


Figure 10: Process model with conformance annotations for traces that contain a model move on *Take in charge ticket*

In a next step, we are interested in finding log moves, i.e. moves that are present in the log but cannot be performed in the model in the current state. Log moves are displayed in the marking in which they occur. In a first step, we want to find the activities with the most performed log moves. Therefore, we use the "Move Containment Filter" and filter for one log move after another. Thus, we can derive the number of log moves per activity. The two activities with the highest number of log moves are *Take in charge ticket* (626 traces with log moves on this activity) and *Assign seriousness* (393 traces). We note that these are the same activities that also have a large number of model moves according to previous section. First, we focus on the activity *Take in charge ticket* and filter the log such that it only contains traces with log moves on this activity. An excerpt of the resulting model with annotated information is visualized in Figure 11.



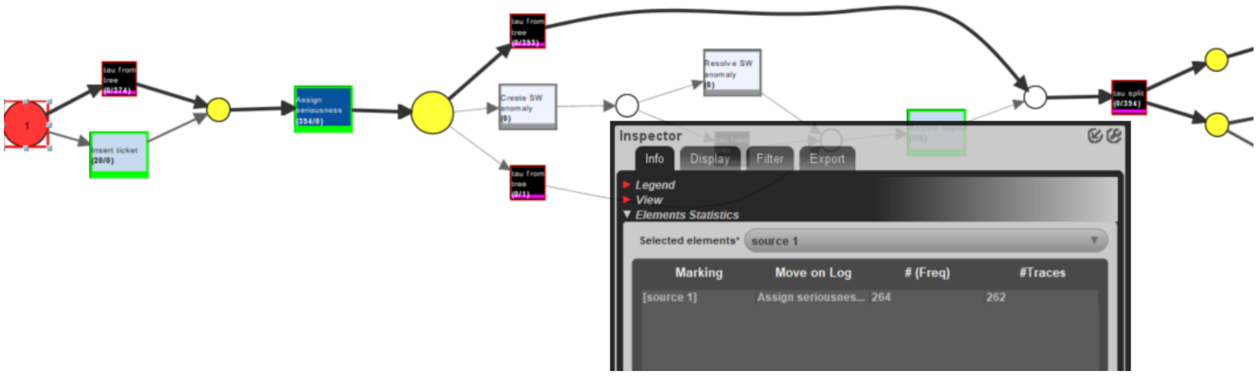
Marking	Move on Log	# (Freq)	#Traces
[sink 4]	Assign seriousnes...	68	62
	Create SW anomaly+	2	2
	Resolve ticket+	20	16
	Take in charge tick...	405	377
	Wait+	21	18

* Click a place/transition on the projected model to see its stats, or use combobox

Figure 11: Process model with conformance annotations for traces that contain a log move on *Take in charge ticket*

One can see that there are a lot of synchronous moves for the transition labeled with *Take in charge ticket*. Additionally, there is always at least one further log move for the activity *Take in charge ticket*. Figure 11 highlights a marking, which is reached before (not necessary directly before) the transition *Take in charge ticket* is enabled. In this marking, there are 405 log moves in 377 traces – some traces contain multiple log moves. In addition, there are also log moves in markings that are reached after the transition has fired (e.g. in the marking that marks the postset of the transition labeled with *Take in charge ticket*, and one of the places in the pre- or post-set of the transition labeled with *Wait*). Because there are synchronous moves as well as log moves, we know that *Take in charge ticket* often happens multiple times in the traces of the event log. In contrast, the model enforces exactly one execution of the transition labeled with *Take in charge ticket*. To prove our assumption, we export the log that is filtered based on the containment of the log move and visualize it using the "Explore event log"-view (see Figure 48 in the appendix for an excerpt). Only three out of 626 traces do not contain *Take in charge ticket* multiple time. Thus, we are correct and the main problem of the process model is that it does not allow for repetitions of the activity.

Next, we analyse the log moves for the activity *Assign seriousness*. We use the same analysis setup as before and create the process model with annotated conformance data for the log that is filtered based on log moves for the activity *Assign seriousness* (see Figure 12). The largest amount of log moves is present in one of the first three states of the state space of the process model. Furthermore, it holds for all traces of the filtered log that there is a synchronous move on *Assign seriousness*. Because of that, we know that all the traces that contain a log move must execute *Assign seriousness* multiple times.



Selected elements* source 1

Marking	Move on Log	# (Freq)	#Traces
[source 1]	Assign seriousnes...	264	262

Figure 12: Process model with conformance annotations for traces that contain a log move on *Assign seriousness*

Again, to check our assumption, we visualize the different variants of the filtered log (see Figure 13). We observe that there are always repetitions of the activity *Assign seriousness* in these traces. Furthermore, this repetition happens in the beginning of the trace for a majority of cases.

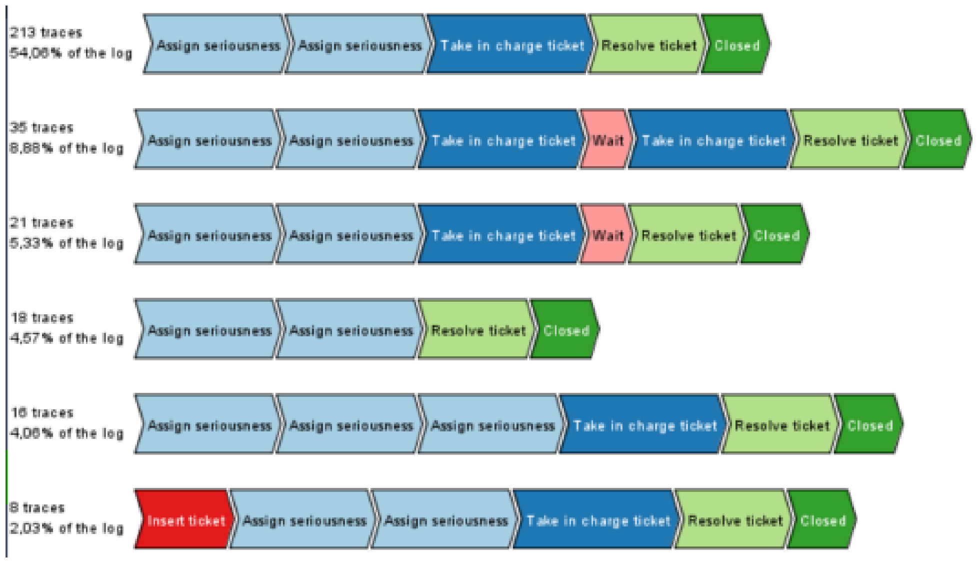


Figure 13: Excerpt of variants that contain a log move on *Assign seriousness*

Q4 Decision Mining

(a) In the following, we would like to find explanations for some specific behaviour regarding (the repeated execution of) activity *Wait*. Therefore, we create a case-based situation table in Celonis consisting of the following columns:

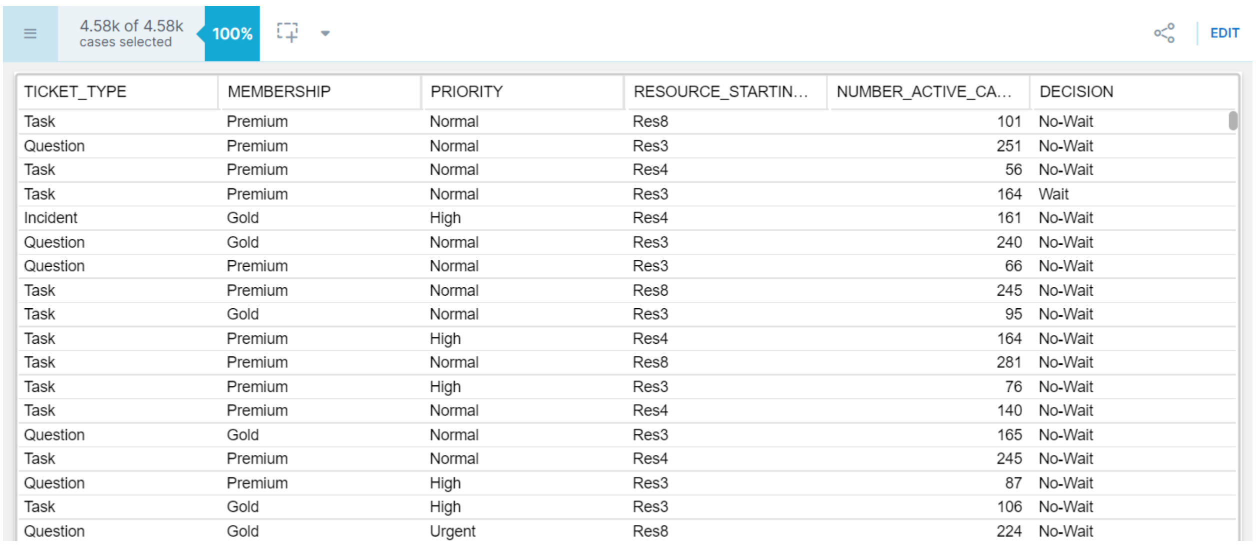
1. `"case_table_csv"."TICKET TYPE"`
2. `"case_table_csv"."MEMBERSHIP"`
3. `PU_LAST("case_table_csv", "event_table_csv"."PRIORITY")`
4. `PU_FIRST("case_table_csv", "event_table_csv"."RESOURCE")`
5. `PU_FIRST("case_table_csv", KPI("No. of active cases at event"))`

   with a stored formula "No. of active cases at event" as:

   ```
   RUNNING_SUM(
     CASE WHEN ACTIVITY_LAG("event_table_csv"."ACTIVITY") IS NULL
     THEN 1
     WHEN ACTIVITY_LEAD("event_table_csv"."ACTIVITY") IS NULL
     THEN -1
     ELSE 0
     END,
     ORDER BY ("event_table_csv"."TIMESTAMP")
   )
   ```

6. `CASE WHEN PROCESS EQUALS 'Wait' THEN 'Wait' ELSE 'No-Wait' END`

An excerpt of the resulting situation table is shown in Figure 14.


4.58k of 4.58k cases selected 100% EDIT

TICKET_TYPE	MEMBERSHIP	PRIORITY	RESOURCE_STARTIN...	NUMBER_ACTIVE_CA...	DECISION
Task	Premium	Normal	Res8	101	No-Wait
Question	Premium	Normal	Res3	251	No-Wait
Task	Premium	Normal	Res4	56	No-Wait
Task	Premium	Normal	Res3	164	Wait
Incident	Gold	High	Res4	161	No-Wait
Question	Gold	Normal	Res3	240	No-Wait
Question	Premium	Normal	Res3	66	No-Wait
Task	Premium	Normal	Res8	245	No-Wait
Task	Gold	Normal	Res3	95	No-Wait
Task	Premium	High	Res4	164	No-Wait
Task	Premium	Normal	Res8	281	No-Wait
Task	Premium	High	Res3	76	No-Wait
Task	Premium	Normal	Res4	140	No-Wait
Question	Gold	Normal	Res3	165	No-Wait
Task	Premium	Normal	Res4	245	No-Wait
Question	Premium	High	Res3	87	No-Wait
Task	Gold	High	Res3	106	No-Wait
Question	Gold	Urgent	Res8	224	No-Wait

Figure 14: Screenshot of the first rows of the situation table created in Celonis.

(b) Having the situation table, we would like to predict the decision column based on the other attributes. Therefore, we export the situation table in Celonis and import it to RapidMiner. There, we define column "DECISION" as "label" and train a decision tree within a Cross Validation operator (workflow is depicted in Figure 49 in the appendix). We use a maximal depth of 5 and a minimal gain of 0.01. The resulting tree is depicted in Figure 15.

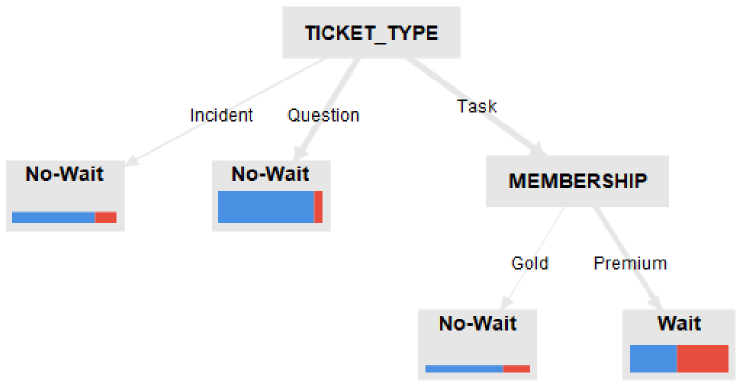


Figure 15: Decision tree predicting the decision trained in RapidMiner.

The trained tree looks concise. The only attributes considered are ticket type and membership. Only for tickets of type "Task" and belonging to a premium membership, the tree predicts decision "Wait". Approximately 52% of these cases, which make up about 36% of the entire event log, are classified correctly. Another leaf containing many cases is described by ticket type "Question": More than 40% of all tickets are represented by this class and therefore classified as "No-Wait" (correct for more than 92%).

Note that it is possible to get a more detailed (i.e. deeper) decision tree by reducing the minimal gain parameter. This tree would then – for example – consider the attribute describing the number of running cases at case start as well. However, since the overall accuracy is slightly worse for this decision tree, we discard it.

Q5 Performance

(a) Since one of the manager claimed that the throughput time is not the real throughput time, we update the throughput time with the help of the following PQL command:

```
CASE WHEN PROCESS EQUALS 'Closed'
THEN
  CALC_THROUGHPUT(
    ALL_OCCURRENCE['Process Start']
    TO
    LAST_OCCURRENCE['Resolve ticket'],
    REMAP_TIMESTAMPS("event_table_csv"."TIMESTAMP", HOURS)
  )
```

```
ELSE
  CALC_THROUGHPUT(
    ALL_OCCURRENCE['Process Start']
    TO
    ALL_OCCURRENCE['Process End'],
    REMAP_TIMESTAMPS("event_table_csv"."TIMESTAMP", HOURS)
  )
END
```

As we have to use this command later again, we store it in a saved formula called `REAL_THROUGHPUT_TIME_HOURS` and get the distribution as shown in figure Figure 16.

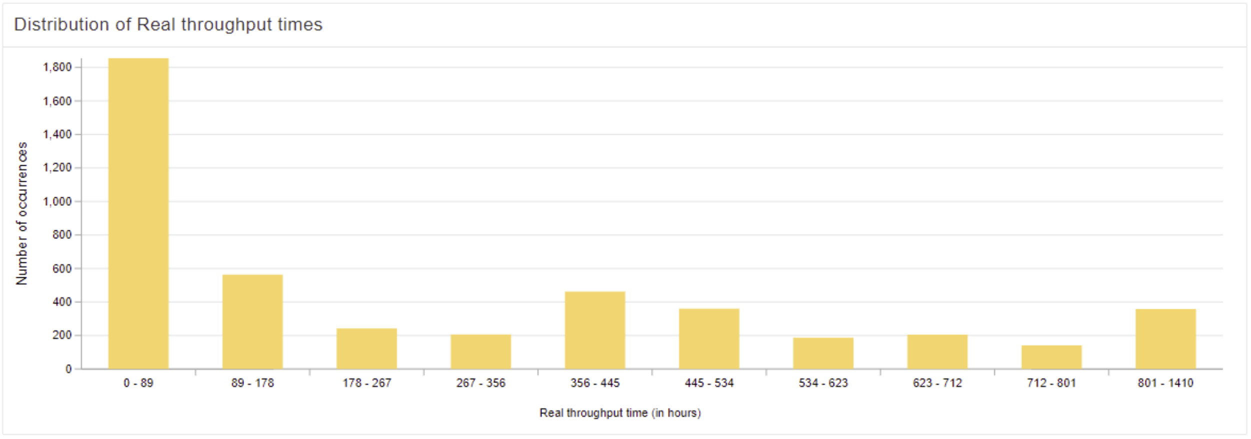
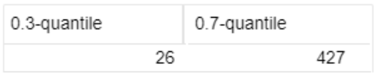


Figure 16: Distribution of the real throughput times (hours) shown in Celonis.

(b) To categorize the throughput times into *Short*, *Medium* and *Long*, we first have to compute the 0.3 and 0.7-quantile. Therefore, we construct the following PQL command:

```
QUANTILE(KPI("REAL_THROUGHPUT_TIME_HOURS"), 0.3)
```

The PQL command for the 0.7-quantile looks analogous. With these commands, we get the thresholds as shown in Figure 17.



0.3-quantile	0.7-quantile
26	427

Figure 17: 0.3 and 0.7-quantile for the throughput times calculated in Celonis.

(c) Now that we are able to categorize the throughput times, we create a case-based situation table (see Figure 18) consisting of the following columns:

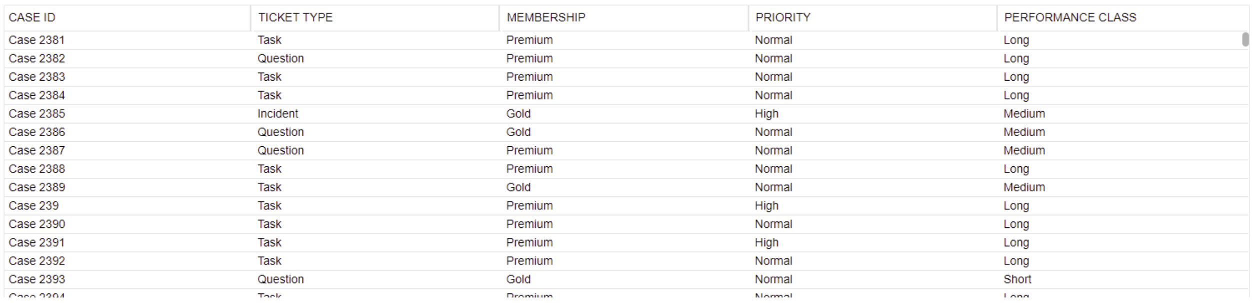
Case ID (not asked for, additionally): `"case_table_csv"."CASE ID"`

Ticket type : `"case_table_csv"."TICKET TYPE"`

Membership : `"case_table_csv"."MEMBERSHIP"`

Priority : `PU_LAST("case_table_csv", "event_table_csv"."PRIORITY")`

Performance class :
```
CASE
    WHEN KPI("REAL_THROUGHPUT_TIME_HOURS") < 26
    THEN 'Short'
    WHEN KPI("REAL_THROUGHPUT_TIME_HOURS") > 427
    THEN 'Long'
    ELSE 'Medium'
END
```



CASE ID	TICKET TYPE	MEMBERSHIP	PRIORITY	PERFORMANCE CLASS
Case 2381	Task	Premium	Normal	Long
Case 2382	Question	Premium	Normal	Long
Case 2383	Task	Premium	Normal	Long
Case 2384	Task	Premium	Normal	Long
Case 2385	Incident	Gold	High	Medium
Case 2386	Question	Gold	Normal	Medium
Case 2387	Question	Premium	Normal	Medium
Case 2388	Task	Premium	Normal	Long
Case 2389	Task	Gold	Normal	Medium
Case 239	Task	Premium	High	Long
Case 2390	Task	Premium	Normal	Long
Case 2391	Task	Premium	High	Long
Case 2392	Task	Premium	Normal	Long
Case 2393	Question	Gold	Normal	Short

Figure 18: Case-based situation table showing selected case attributes and the performance class regarding the real throughput times in Celonis.

(d) Based on the created situation table, we would like to train a decision tree, which helps us to predict the performance class of a case based on its other attributes. Therefore, we import the situation table in RapidMiner, define “CASE ID” as role “id” and “PERFORMANCE CLASS” as “label”. Then, we train the decision tree (criterion information gain, maximal depth 5, minimal gain 0.01) inside a Cross Validation operator. The resulting decision tree is shown in Figure 19 and has an overall accuracy of approximately 75%. It performs best for performance class “Short” and worst for class “Medium” (see Figure 50 in the appendix).

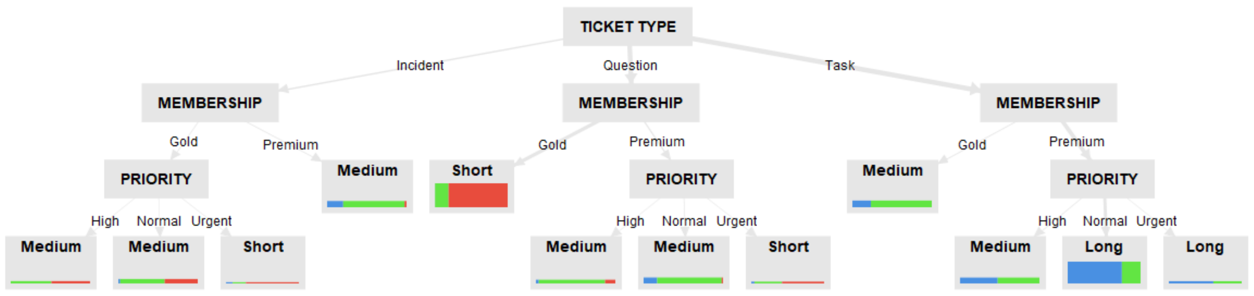


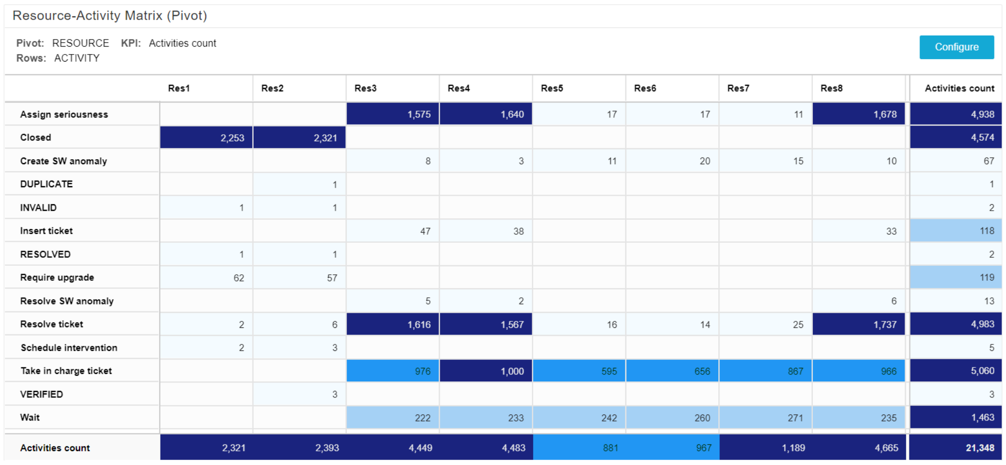
Figure 19: Decision tree predicting the performance class trained in RapidMiner.

The first thing we observe in the decision tree is that tickets of type “Task” never belong to nor are classified as performance class “Short”. In contrast, performance class “Long” is never predicted for both the “Incident” and “Question” ticket types.

Almost 30% of all cases are of type “Task”, belong to a premium membership and have normal priority. Those cases are then predicted to have a long performance which is true for approximately 75% of these cases. Another common combination of attributes is described by ticket type "Question" and membership "Gold". More than 30% of all cases belong to this classification and are predicted as short then. Almost all predictions of performance class "Short" belong to this leaf (97%) and a large majority is predicted correctly (more than 80%). The third ticket type, namely "Incident", makes up about 13% of all cases and is almost always predicted to belong to performance class "Medium".

Q6 Organizational Mining

(a) In the following section, we are going to have a deeper look at the resources involved in the process. Therefore, we use a pivot table in order to create a resource-activity matrix (see Figure 20). For the dimensions, we use the resource (`"event_table_csv"."RESOURCE"`) and the activity (`"event_table_csv"."ACTIVITY"`). As our KPI, we count the number of activities (`COUNT("event_table_csv"."ACTIVITY")`). With the help of the options of our activities count data series, we add a color mapping to make the values of the table easier to recognize. With that, we detect that – for example – activity *Closed* is only executed by resources *Res1* and *Res2* while activity *Assign seriousness* is only executed by all other resources.


Resource-Activity Matrix (Pivot)

Pivot: RESOURCE KPI: Activities count
Rows: ACTIVITY

	Res1	Res2	Res3	Res4	Res5	Res6	Res7	Res8	Activities count
Assign seriousness			1,575	1,640	17	17	11	1,678	4,938
Closed	2,253	2,321							4,574
Create SW anomaly			8	3	11	20	15	10	67
DUPLICATE		1							1
INVALID	1	1							2
Insert ticket			47	38				33	118
RESOLVED	1	1							2
Require upgrade	62	57							119
Resolve SW anomaly			5	2				6	13
Resolve ticket	2	6	1,616	1,567	16	14	25	1,737	4,983
Schedule intervention	2	3							5
Take in charge ticket			976	1,000	595	656	867	966	5,060
VERIFIED		3							3
Wait			222	233	242	260	271	235	1,463
Activities count	2,321	2,393	4,449	4,483	881	967	1,189	4,665	21,348

Figure 20: Resource-activity matrix as pivot table in Celonis.

We would like to get a better overview about the activity distributions for some selected resources using pie charts (see Figure 21). As our dimension, we always select the activity (`"event_table_csv"."ACTIVITY"`) while our KPI is the activity count again (`COUNT("event_table_csv"."ACTIVITY")`). To better compare the charts, we select the activity as sorting attribute. Finally, we filter each pie chart by the regarding resource for which we want to show the activity distribution (e.g. for resource *Res1*: `FILTER "event_table_csv"."RESOURCE" LIKE 'Res1'`). We noticed that there was another possibility for this filtering shown in the instruction and also discussed in the forum (using `CASE WHEN` instead of `FILTER`). However, we keep our usage of `FILTER` here, since the results do not differ and the legend of the resulting plots are cleaner, since only the labels that actually occur are depicted.

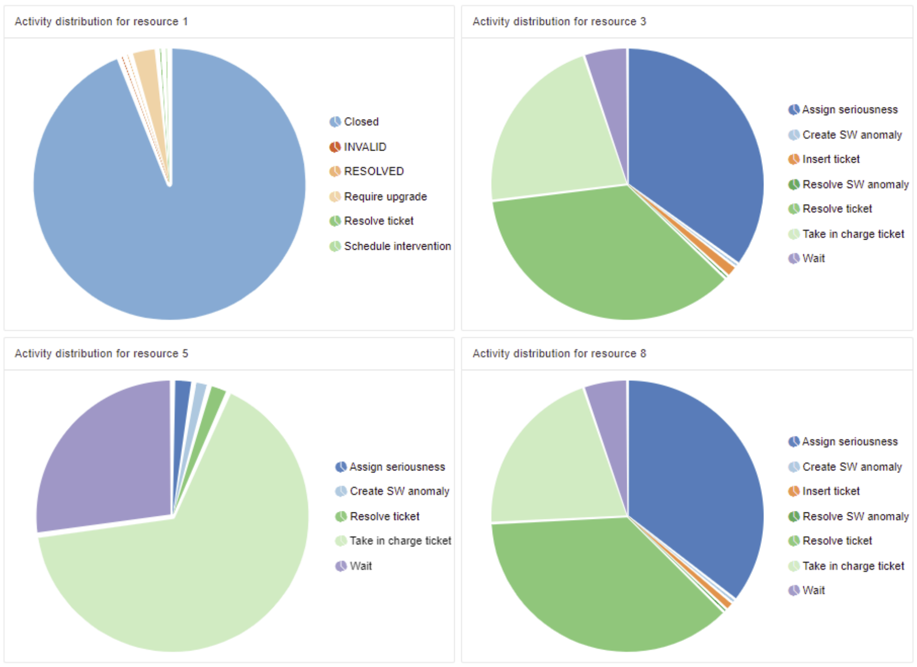


Figure 21: Pie charts showing the distribution of activities per selected resources in Celonis.

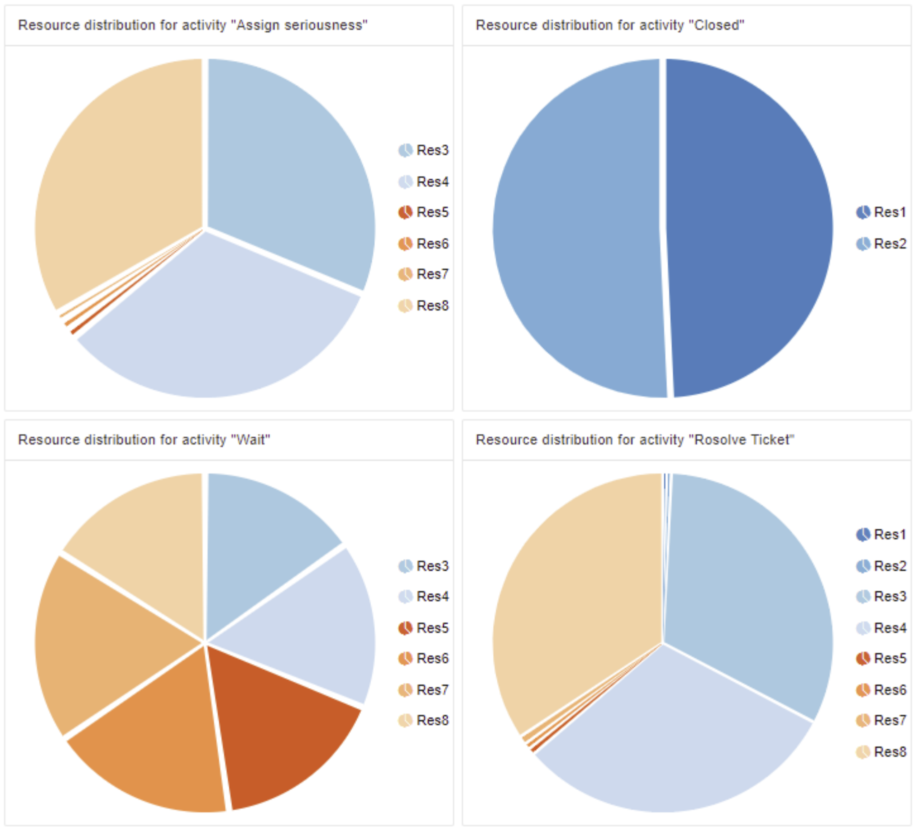
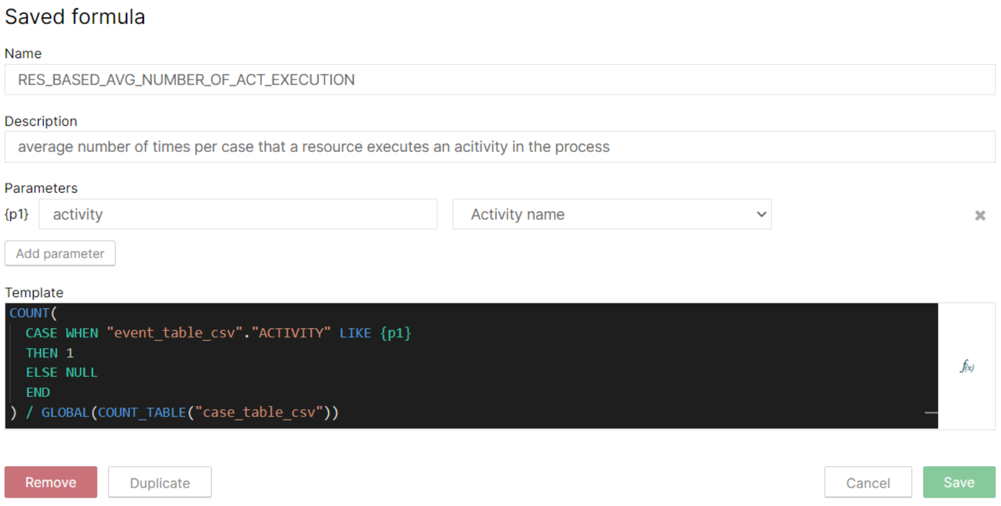


Figure 22: Pie charts showing the distribution of resources per selected activities in Celonis.

Now, we proceed in a similar way for plotting some resource distributions for selected activities (see Figure 22). As our dimension, we now select the resource (`"event_table_csv"."RESOURCE"`) while our KPI is the activity count again (`COUNT("event_table_csv"."ACTIVITY")`). Analogous to the previous pie charts we use the resource as sorting attribute. Finally, we filter each pie chart by the regarding activity for which we want to show the resource distribution (e.g. for activity *Closed*: `FILTER "event_table_csv"."ACTIVITY" LIKE 'Closed'`).

(b) As a next step, we create a resource-activity matrix showing the average number of times per case that a resource r executes an activity a in the process. Therefore, we create a resource-based situation table. This table contains the resource (`"event_table_csv"."RESOURCE"`) as first column. To create the further columns for all activities, we define a custom formula which we can reuse later (see Figure 23). Since each column should represent a different activity, we use the activity name as input parameter. Then, we divide the number of occurrences of this activity by the total number of cases.


Saved formula

Name: RES_BASED_AVG_NUMBER_OF_ACT_EXECUTION

Description: average number of times per case that a resource executes an acitivity in the process

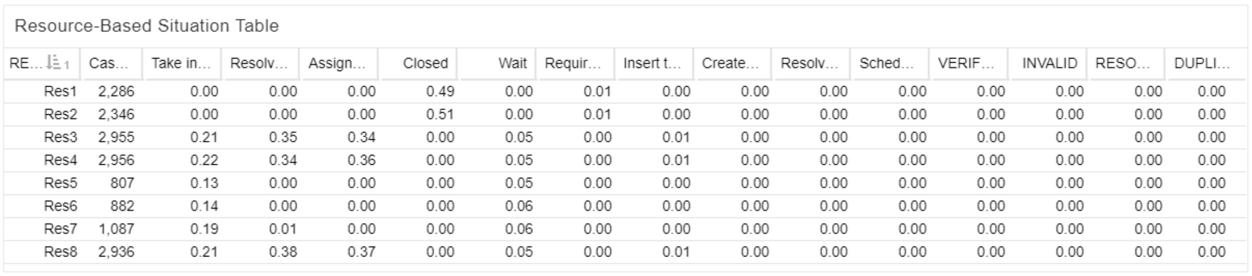
Parameters: {p1} activity — Activity name

Template:

```
COUNT(
  CASE WHEN "event_table_csv"."ACTIVITY" LIKE {p1}
  THEN 1
  ELSE NULL
  END
) / GLOBAL(COUNT_TABLE("case_table_csv"))
```

Figure 23: Pre-defined resp. saved formula for calculating the average number of activity executions per resource per process in Celonis.

With the pre-defined formula, we add one dimension for each activity (formatted as decimal number and rounded to two decimal places), e.g. for activity *Closed*: `KPI("RES_BASED_AVG_NUMBER_OF_ACT_EXECUTION", 'Closed')`. The resulting table is shown in Figure 24.


Resource-Based Situation Table

RE...	Cas...	Take in...	Resolv...	Assign...	Closed	Wait	Requir...	Insert t...	Create...	Resolv...	Sched...	VERIF...	INVALID	RESO...	DUPLI...
Res1	2,286	0.00	0.00	0.00	0.49	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Res2	2,346	0.00	0.00	0.00	0.51	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Res3	2,955	0.21	0.35	0.34	0.00	0.05	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Res4	2,956	0.22	0.34	0.36	0.00	0.05	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Res5	807	0.13	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Res6	882	0.14	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Res7	1,087	0.19	0.01	0.00	0.00	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Res8	2,936	0.21	0.38	0.37	0.00	0.05	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00

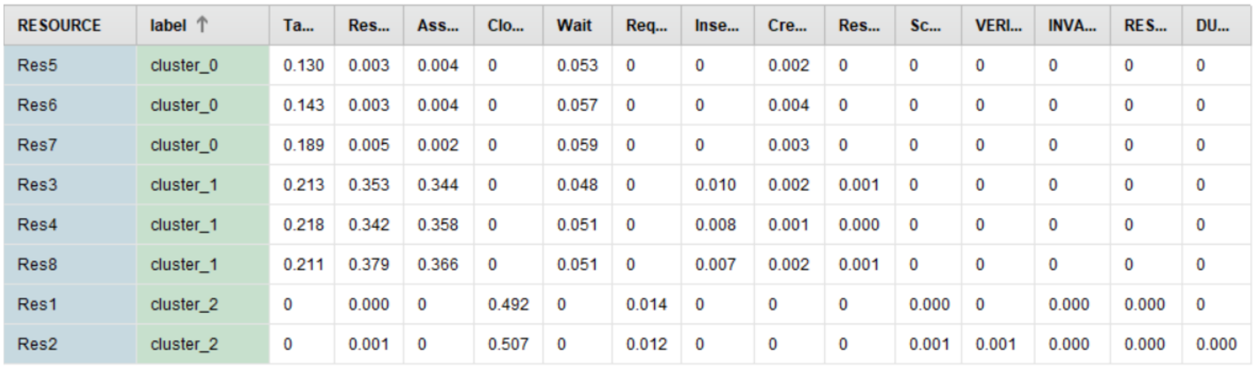
Figure 24: Resource-activity matrix as resource-based situation table in Celonis.

(c) We want to use the previously built resource-based situation table to find clusters of resources. Therefore, we export the situation table from Celonis and import it to RapidMiner. In the process designer, we use the Set Role operator to define column “RESOURCE” as “id”. After doing so, we apply a k-means clustering operator with $k = 3$. Obviously, the clustering returns three clusters (as depicted in Figure 25):

Cluster 0 consists of resources *Res5*, *Res6* and *Res7*. These resources typically execute activities *Take in charge ticket* and *Wait*. Less often, they execute *Assign seriousness*, *Create SW anomaly* or *Resolve ticket*.

Cluster 1 consists of resources *Res3*, *Res4* and *Res8*. These resources typically execute activities *Take in charge ticket*, *Resolve ticket* and *Assign seriousness*. Less often, they execute *Wait*, *Create SW anomaly*, *Insert ticket* or *Resolve SW anomaly*.

Cluster 2 consists of resources *Res1* and *Res2*. These resources typically execute activity *Closed*. Very rarely, they execute *DUPLICATE*, *INVALID*, *RESOLVED*, *VERIFIED*, *Require upgrade*, *Resolve ticket* or *Schedule intervention*.



RESOURCE	label ↑	Ta...	Res...	Ass...	Clo...	Wait	Req...	Inse...	Cre...	Res...	Sc...	VERI...	INVA...	RES...	DU...
Res5	cluster_0	0.130	0.003	0.004	0	0.053	0	0	0.002	0	0	0	0	0	0
Res6	cluster_0	0.143	0.003	0.004	0	0.057	0	0	0.004	0	0	0	0	0	0
Res7	cluster_0	0.189	0.005	0.002	0	0.059	0	0	0.003	0	0	0	0	0	0
Res3	cluster_1	0.213	0.353	0.344	0	0.048	0	0.010	0.002	0.001	0	0	0	0	0
Res4	cluster_1	0.218	0.342	0.358	0	0.051	0	0.008	0.001	0.000	0	0	0	0	0
Res8	cluster_1	0.211	0.379	0.366	0	0.051	0	0.007	0.002	0.001	0	0	0	0	0
Res1	cluster_2	0	0.000	0	0.492	0	0.014	0	0	0	0.000	0	0.000	0.000	0
Res2	cluster_2	0	0.001	0	0.507	0	0.012	0	0	0	0.001	0.001	0.000	0.000	0.000

Figure 25: Centroid table showing the results of k-means clustering ($k = 3$) in RapidMiner.

(d) In the following, we would like to have a deeper look at handover of work between different resources. Therefore, we create a situation table where each row represents a pair of resources where handover of work takes place. The table consists of the following columns:

Source: `SOURCE("event_table_csv"."RESOURCE")`

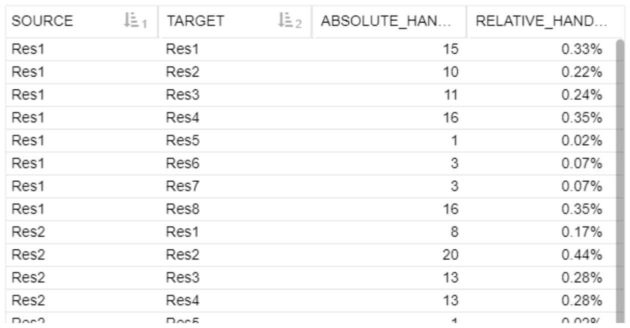
Target: `TARGET("event_table_csv"."RESOURCE")`

Abs. number of handovers: `COUNT(SOURCE("event_table_csv"."ACTIVITY"))`

Rel. number of handovers: `COUNT(SOURCE("event_table_csv"."ACTIVITY")) / GLOBAL(TABLE_COUNT("case_table_csv"))`

An excerpt of the resulting situation table is shown in Figure 26.

(e) We use the previously created situation table and generate a social network in RapidMiner. So that we do not see all connections, but only the stronger ones, we filter only those



SOURCE	TARGET	ABSOLUTE_HAN...	RELATIVE_HAND...
Res1	Res1	15	0.33%
Res1	Res2	10	0.22%
Res1	Res3	11	0.24%
Res1	Res4	16	0.35%
Res1	Res5	1	0.02%
Res1	Res6	3	0.07%
Res1	Res7	3	0.07%
Res1	Res8	16	0.35%
Res2	Res1	8	0.17%
Res2	Res2	20	0.44%
Res2	Res3	13	0.28%
Res2	Res4	13	0.28%

Figure 26: Situation table showing simple handover of work in Celonis.

pairs whose relative frequency is greater than or equal to 0.1. After applying the transition graph operator, we get the social network as depicted in Figure 27 (using KKLayout).

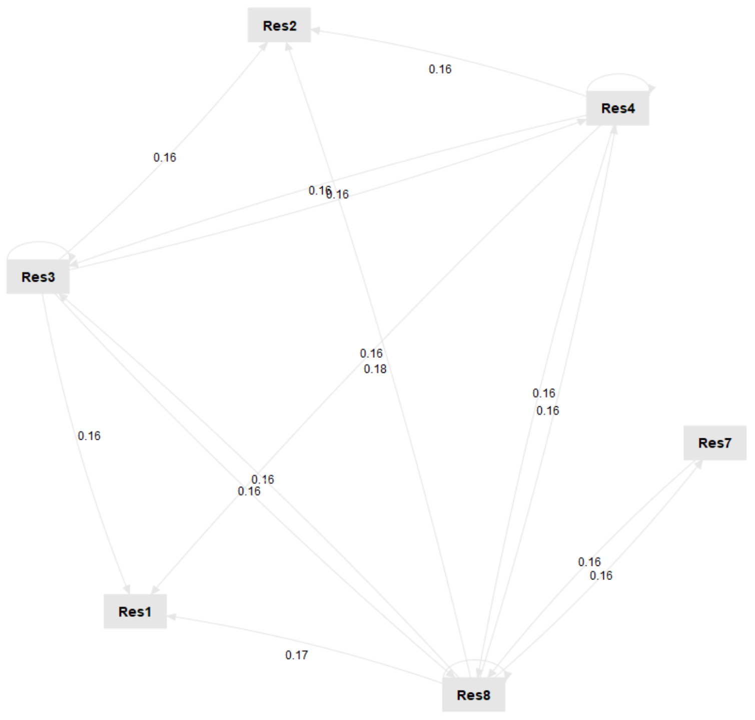


Figure 27: Social network showing handover of work between resources in RapidMiner.

(f) We use the previously discovered clusters of resources to recreate the situation table from before to show the handover of work between the clusters. Therefore, we add a new OLAP table in Celonis consisting of the following columns:

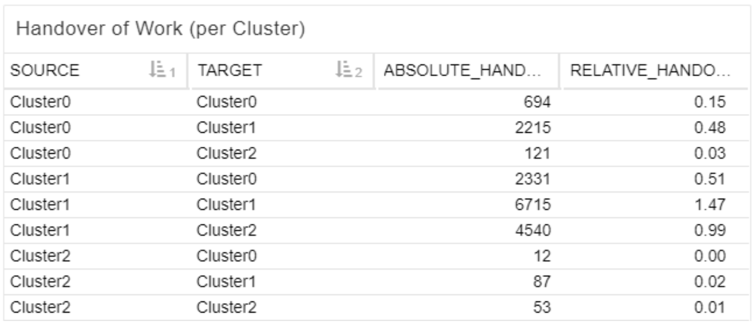
Source:
```
CASE WHEN SOURCE("event_table_csv"."RESOURCE") IN ('Res1', 'Res2')
THEN 'Cluster2'
WHEN SOURCE("event_table_csv"."RESOURCE") IN ('Res5', 'Res6', 'Res7')
THEN 'Cluster0'
ELSE 'Cluster1'
END
```

Target:
```
CASE WHEN TARGET("event_table_csv"."RESOURCE") IN ('Res1', 'Res2')
    THEN 'Cluster2'
    WHEN TARGET("event_table_csv"."RESOURCE") IN ('Res5', 'Res6', 'Res7')
    THEN 'Cluster0'
    ELSE 'Cluster1'
    END
```

Abs. number of handovers: `COUNT(SOURCE("event_table_csv"."ACTIVITY"))`

Rel. number of handovers:
```
COUNT(SOURCE("event_table_csv"."ACTIVITY")) /
    GLOBAL(COUNT_TABLE("case_table_csv"))
```

The resulting situation table is depicted in Figure 28.



Handover of Work (per Cluster)

SOURCE	TARGET	ABSOLUTE_HAND...	RELATIVE_HANDO...
Cluster0	Cluster0	694	0.15
Cluster0	Cluster1	2215	0.48
Cluster0	Cluster2	121	0.03
Cluster1	Cluster0	2331	0.51
Cluster1	Cluster1	6715	1.47
Cluster1	Cluster2	4540	0.99
Cluster2	Cluster0	12	0.00
Cluster2	Cluster1	87	0.02
Cluster2	Cluster2	53	0.01

Figure 28: Situation table showing simple handover of work between clusters of resources in Celonis.

(g) We use the previously created situation table and generate a social network in RapidMiner. As before, we do not want to see all connections, but only the stronger ones, we filter only those pairs whose relative frequency is greater than or equal to 0.01. After applying the transition graph operator, we get the social network as depicted in Figure 29 (using FRLayout).

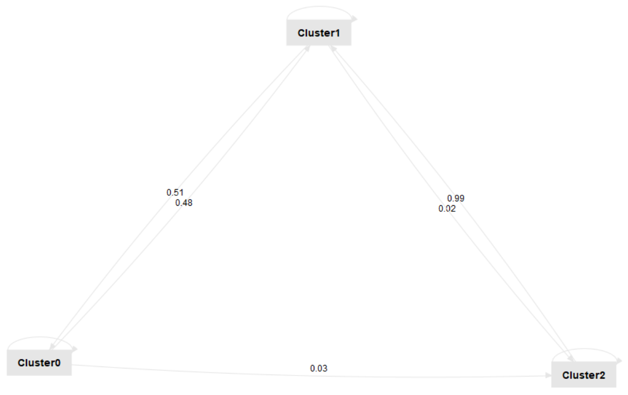
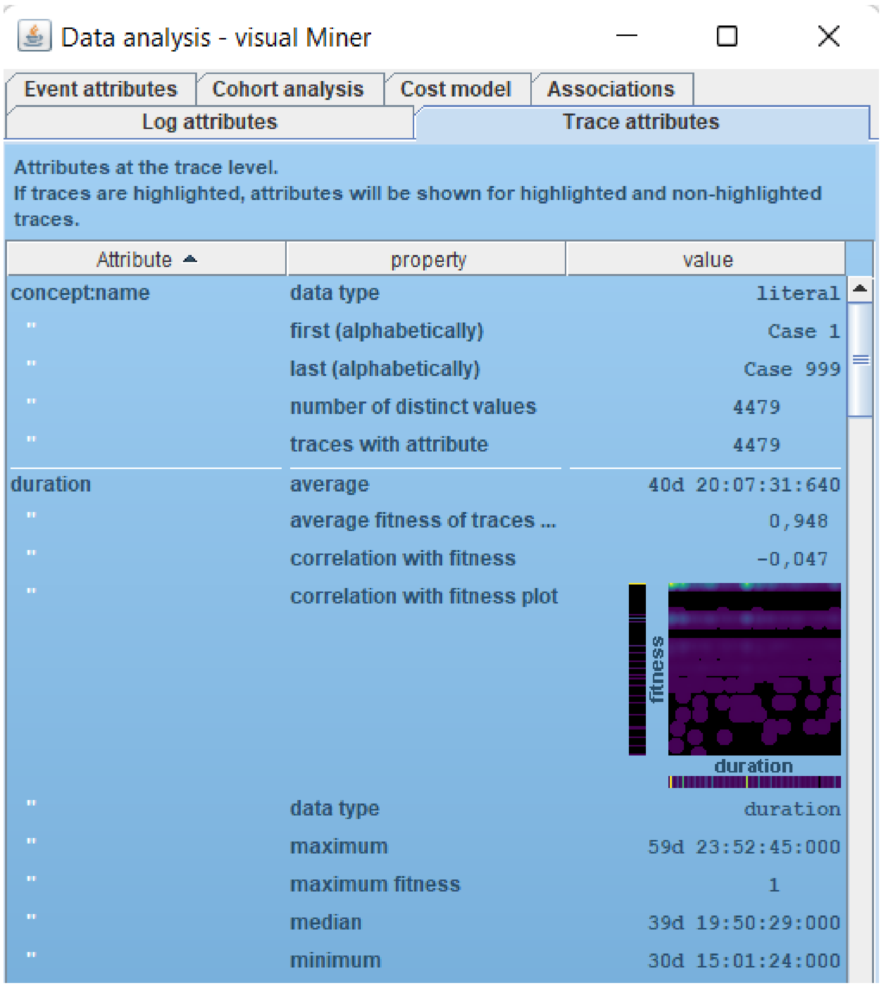


Figure 29: Social network showing handover of work between clusters of resources in RapidMiner.

Appendix

Case duration statistics (Exercise 1c)

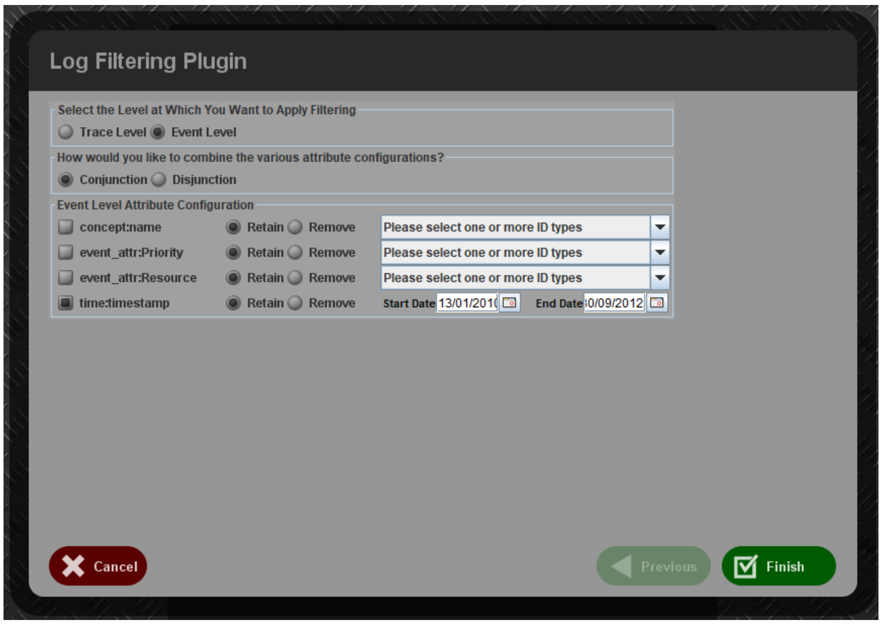


Attributes at the trace level.
If traces are highlighted, attributes will be shown for highlighted and non-highlighted traces.

Attribute	property	value
concept:name	data type	literal
"	first (alphabetically)	Case 1
"	last (alphabetically)	Case 999
"	number of distinct values	4479
"	traces with attribute	4479
duration	average	40d 20:07:31:640
"	average fitness of traces ...	0,948
"	correlation with fitness	-0,047
"	correlation with fitness plot	
"	data type	duration
"	maximum	59d 23:52:45:000
"	maximum fitness	1
"	median	39d 19:50:29:000
"	minimum	30d 15:01:24:000

Figure 30: Duration statistics for the given event log with complete traces

Filtering (Exercise 2a)



Log Filtering Plugin

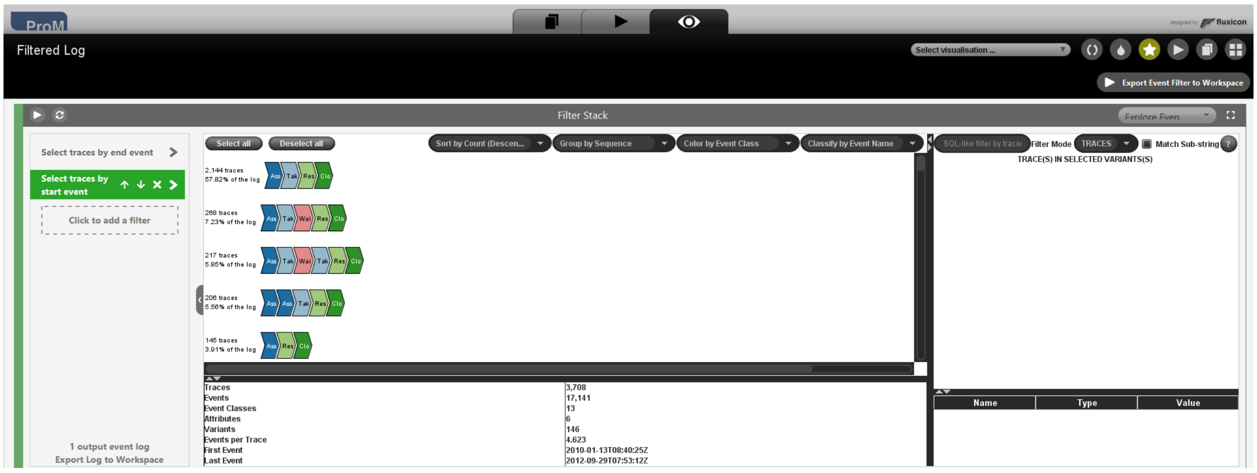
Select the Level at Which You Want to Apply Filtering
Trace Level [✓ Event Level]

How would you like to combine the various attribute configurations?
[✓ Conjunction] Disjunction

Event Level Attribute Configuration

concept:name	[✓ Retain] Remove	Please select one or more ID types
event_attr:Priority	[✓ Retain] Remove	Please select one or more ID types
event_attr:Resource	[✓ Retain] Remove	Please select one or more ID types
[✓ time:timestamp]	[✓ Retain] Remove	Start Date 13/01/201(End Date)0/09/2012

Figure 31: Applying plugin “Log filtering” to *log-flat* in *ProM*.



Filtered Log

2,144 traces, 57.82% of the log	Ass Tak Res Clo
268 traces, 7.23% of the log	Ass Tak Wai Res Clo
217 traces, 5.85% of the log	Ass Tak Wai Tak Res Clo
206 traces, 5.56% of the log	Ass Ass Tak Res Clo
145 traces, 3.91% of the log	Ass Res Clo

Traces	3,708
Events	17,141
Event Classes	13
Attributes	6
Variants	146
Events per Trace	4.623
First Event	2010-01-13T08:40:25Z
Last Event	2012-09-29T07:53:12Z

Name	Type	Value

Figure 32: Using view “Event filter on event log” on pre-filtered *log-flat* in *ProM*.

Process model M_{pre-h_1} (Exercise 2b)

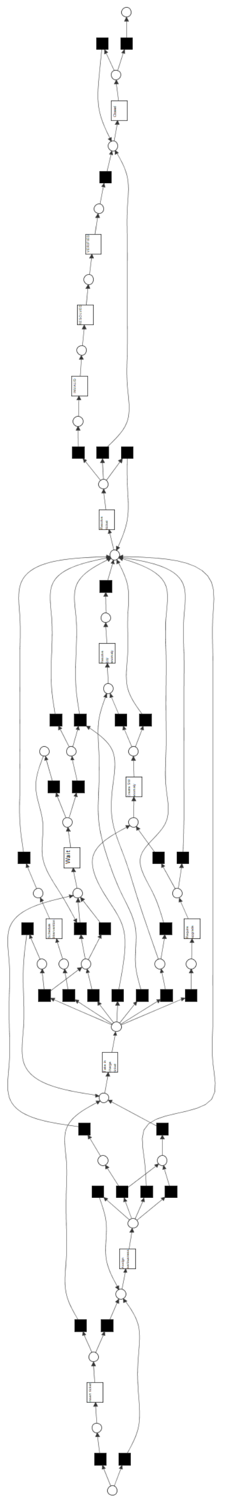


Figure 33: Petri net discovered using *log-pre-complete* and Heuristic Miner (frequency of 0, dependency of 0.045).

Process model M_{pre-h_2} (Exercise 2b)

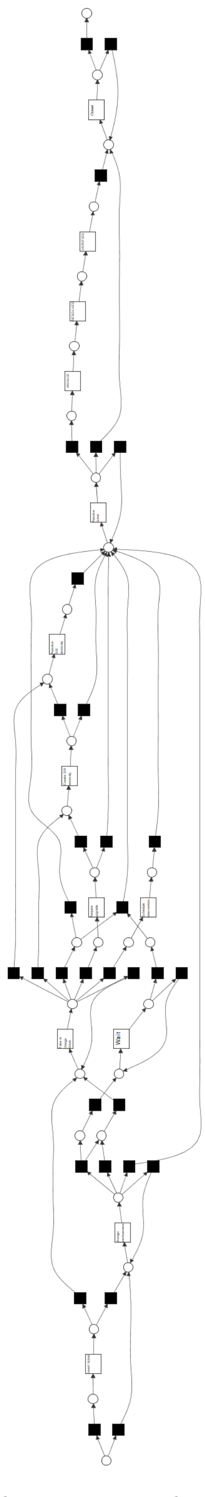


Figure 34: Petri net discovered using *log-pre-complete* and Heuristic Miner (frequency of 0, dependency of 0.5).

Process model M_{pre-r_1} (Exercise 2b)

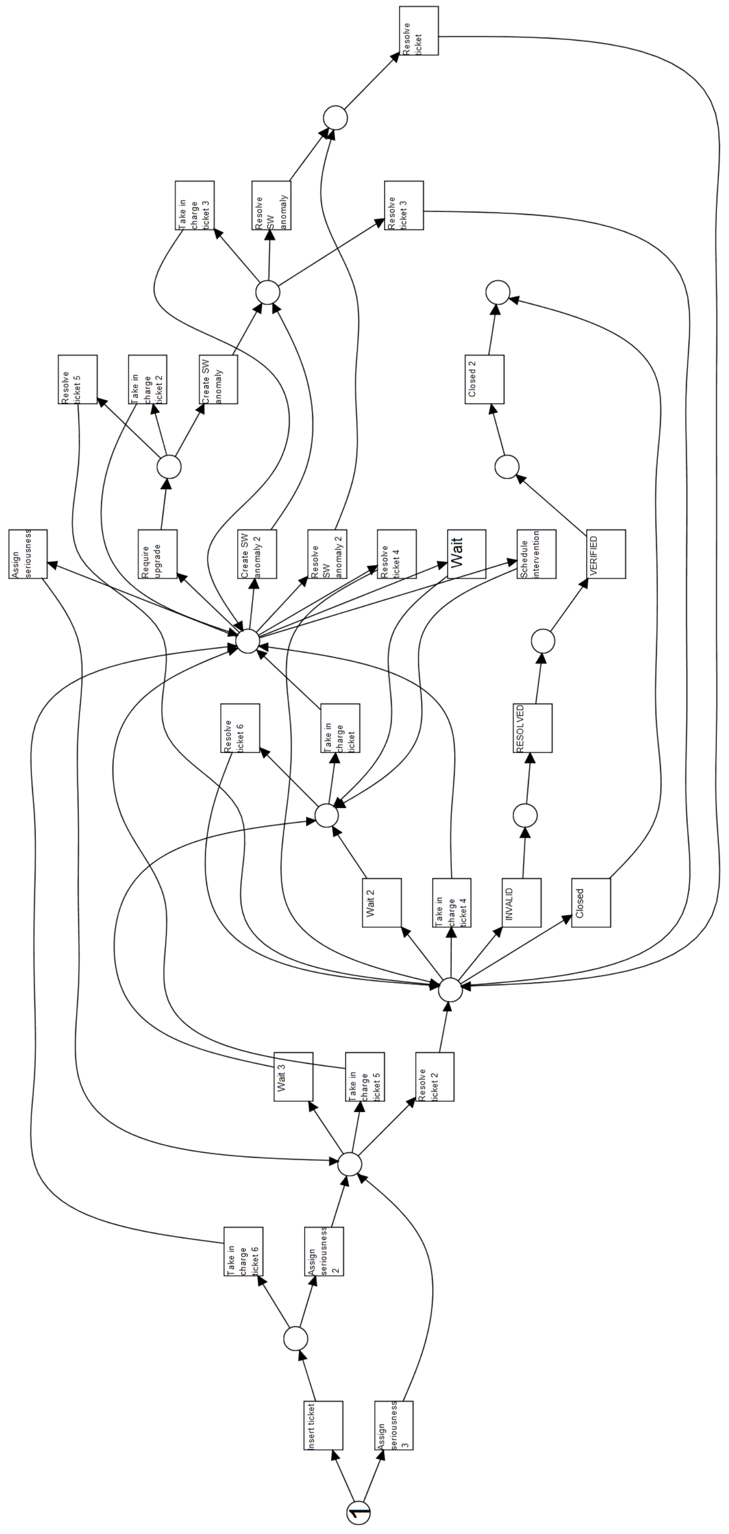


Figure 35: Petri net discovered using *log-pre-complete* and State-based Region Mining (default settings).

Process model M_{pre-r_2} (Exercise 2b)

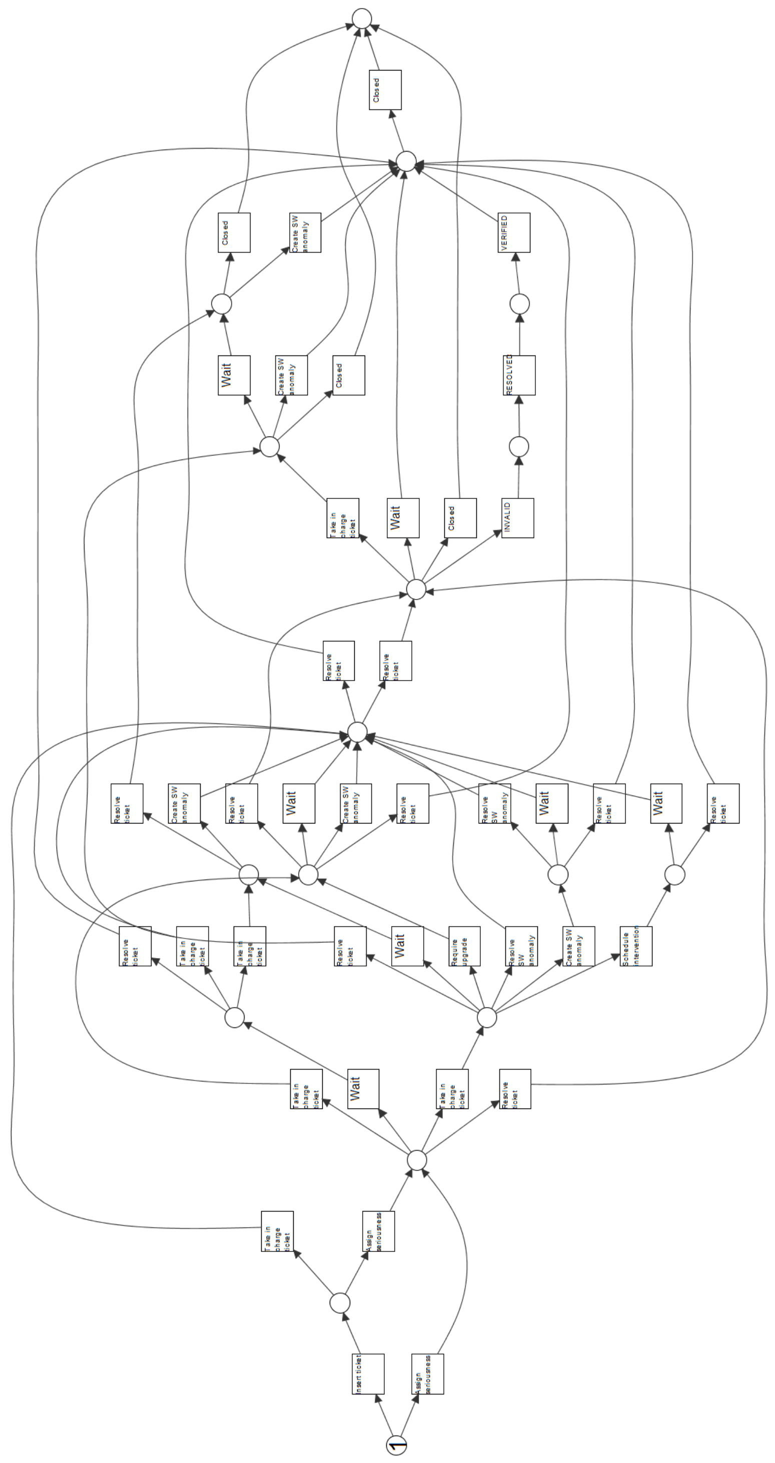


Figure 36: Petri net discovered using *log-pre-complete* and State-based Region Mining (transition system size limit: 100, collection size limit: 20, merge states with identical inflow and outflow).

Process model M_{pre-i_1} (Exercise 2b)

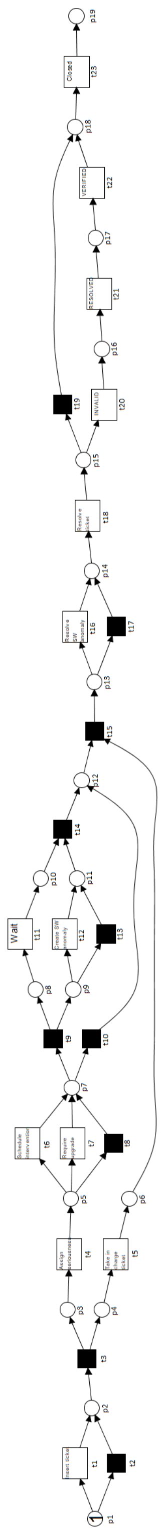


Figure 37: Petri net discovered using *log-pre-complete* and Inductive Miner (variant *IMf* with a noise threshold of 0.1) annotated with place and transition labels.

Process model M_{pre-i_2} (Exercise 2b)

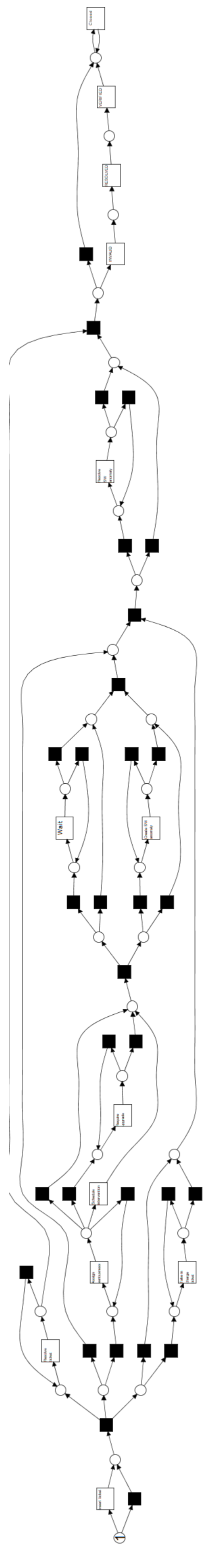


Figure 38: Petri net discovered using *log-pre-complete* and Inductive Miner (variant *IM*).

Process model M_{post-h_1} (Exercise 2b)

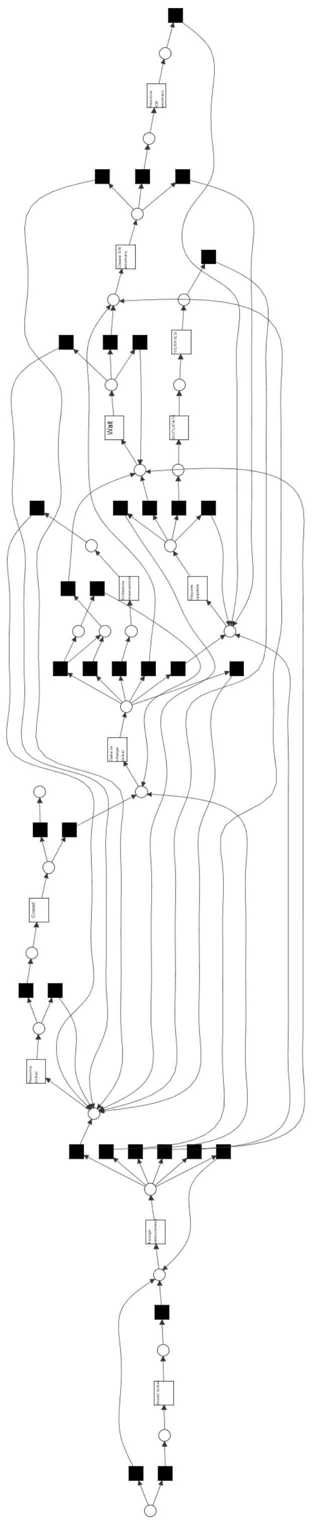


Figure 39: Petri net discovered using *log-post-complete* and Heuristic Miner (frequency of 0, dependency of 0.045).

Process model M_{post-h_2} (Exercise 2b)

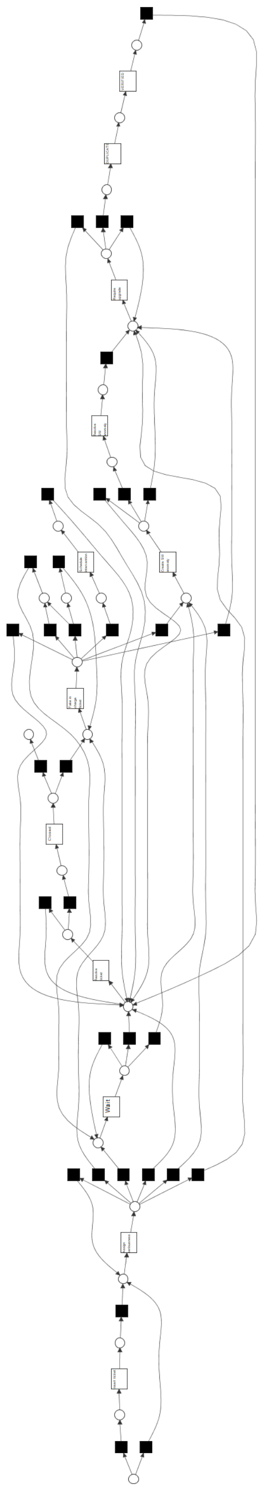


Figure 40: Petri net discovered using *log-post-complete* and Heuristic Miner (frequency of 0, dependency of 0.5).

Process model M_{post-r_1} (Exercise 2b)

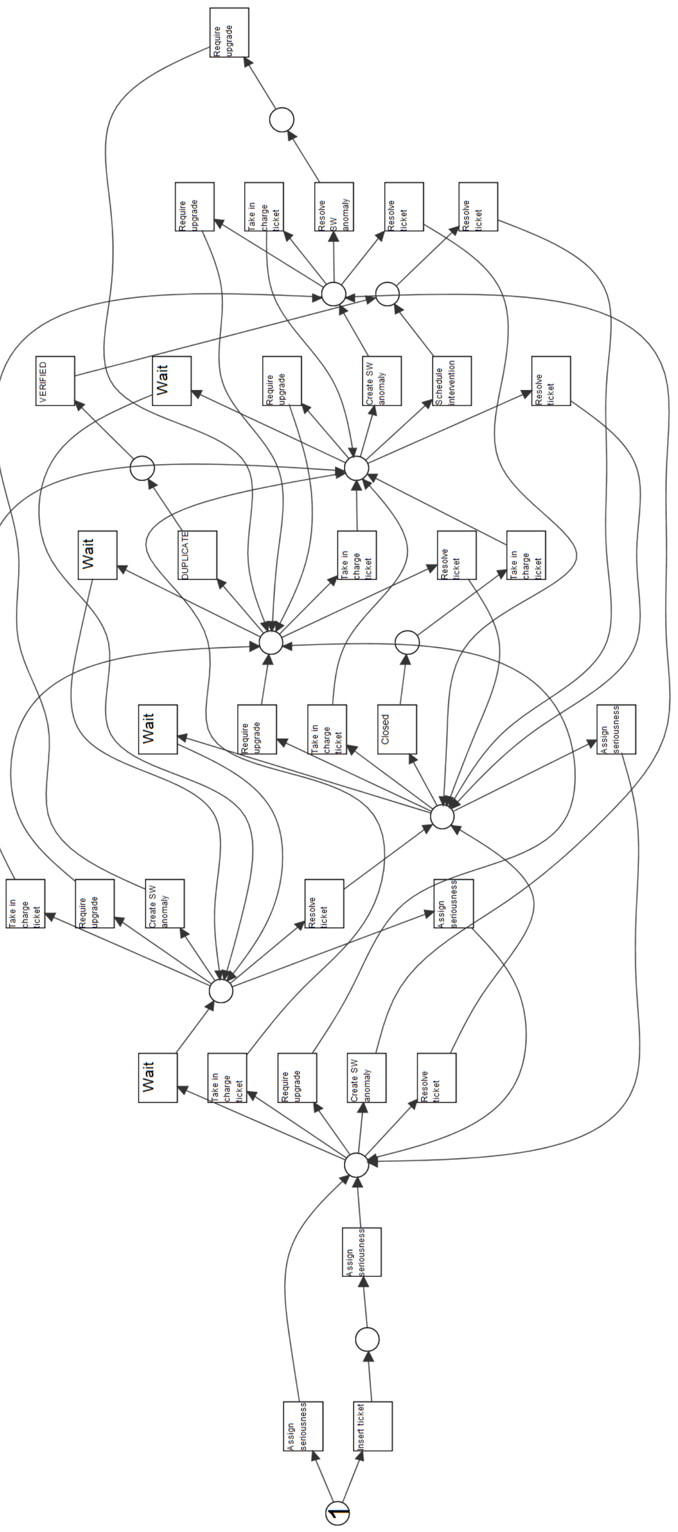


Figure 41: Petri net discovered using *log-post-complete* and State-based Region Mining (default settings).

Process model M_{post-r_2} (Exercise 2b)

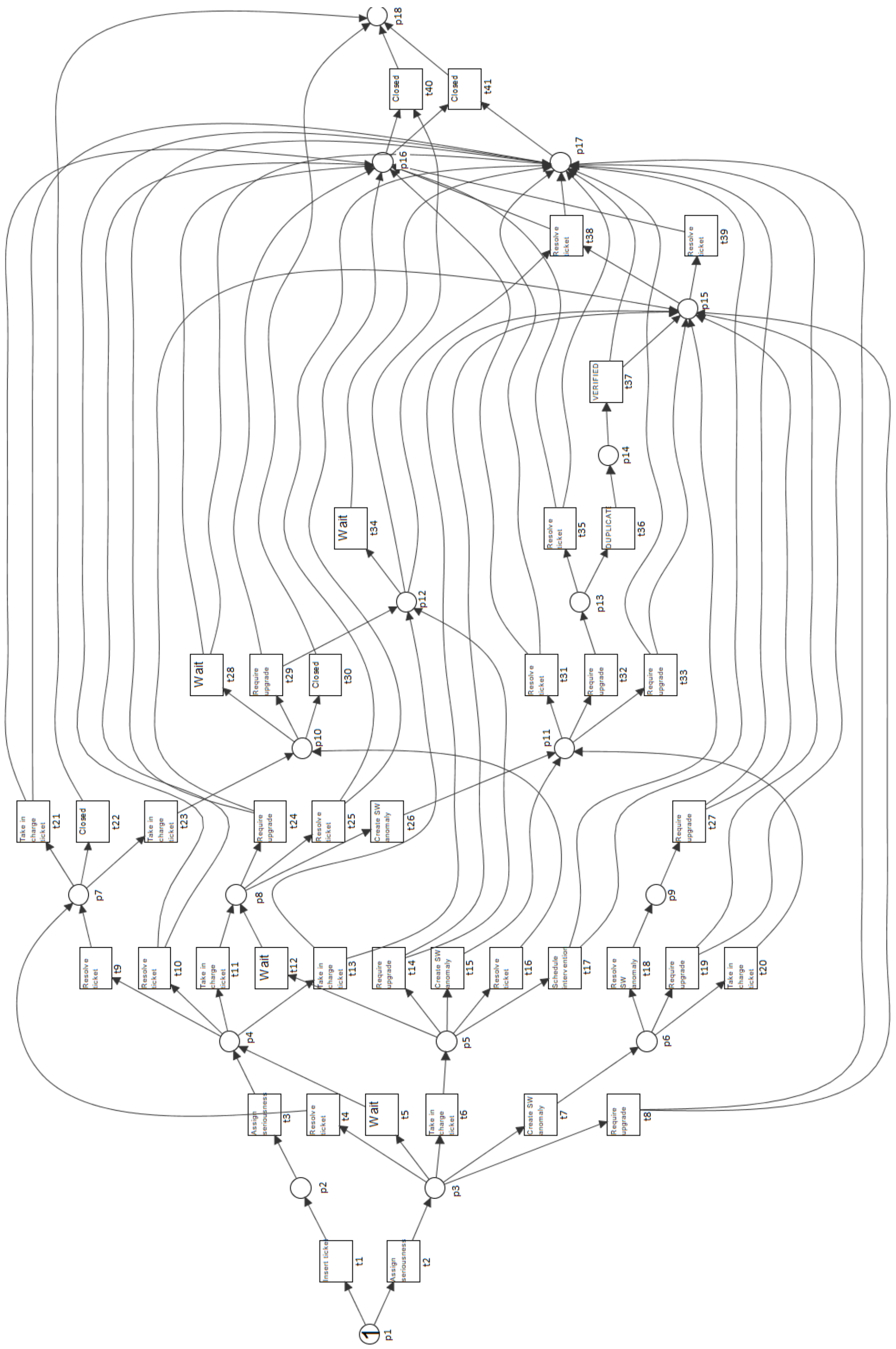


Figure 42: Petri net discovered using *log-post-complete* and State-based Region Mining (transition system size limit: 100, collection size limit: 20, merge states with identical inflow and outflow).

Process model M_{post-i_1} (Exercise 2b)

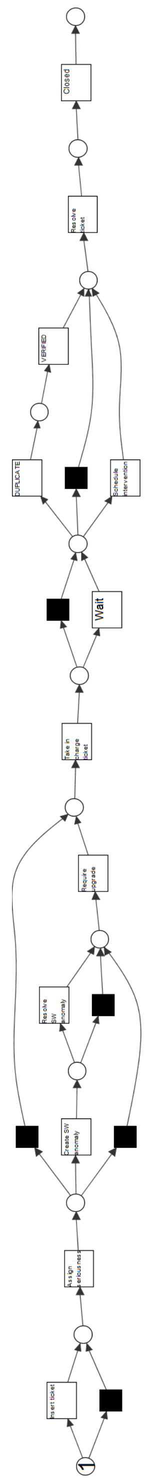


Figure 43: Petri net discovered using *log-post-complete* and Inductive Miner (variant *IMf* with a noise threshold of 0.2).

Process model M_{post-i_2} (Exercise 2b)

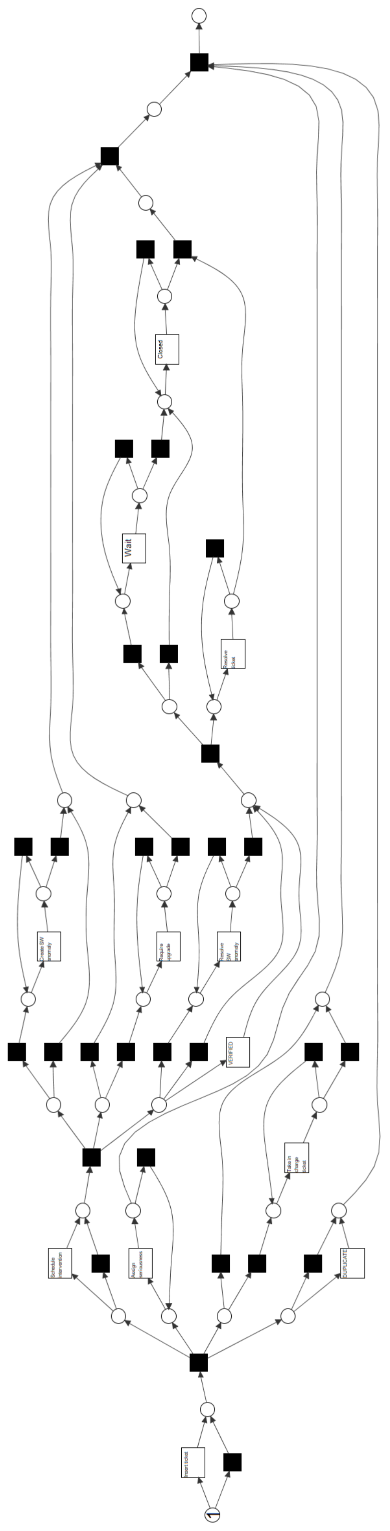


Figure 44: Petri net discovered using *log-post-complete* and Inductive Miner (variant *IM*).

Process model (Exercise 3a)

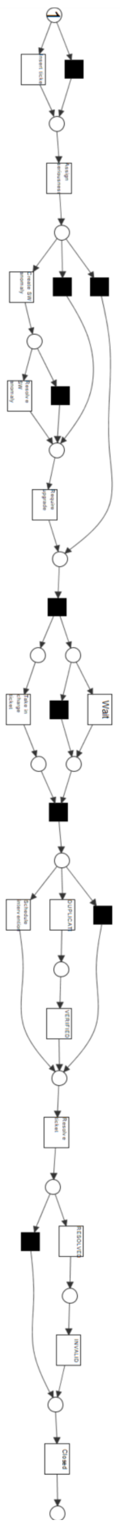


Figure 45: Petri net discovered using the filtered log and the Inductive Miner (variant *IMf*, noise threshold of 0.2).

Process model (Exercise 3a)

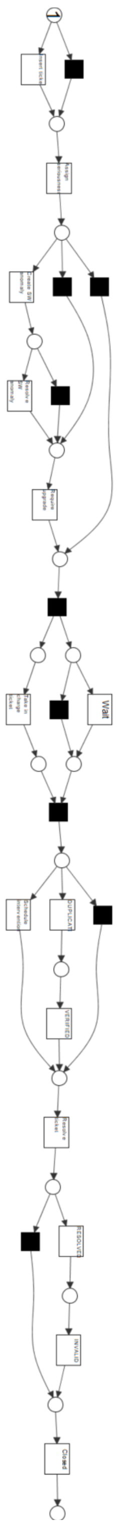


Figure 46: Petri net discovered using the filtered log and the Inductive Miner (variant *IMf*, noise threshold of 0.2).

Improved process model (Exercise 3b)

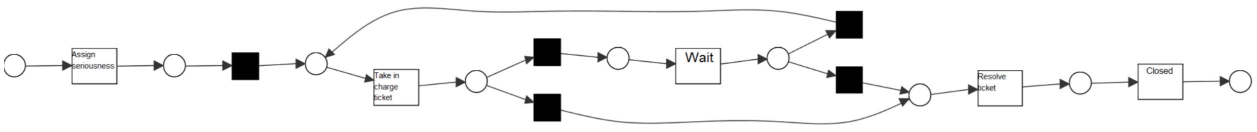


Figure 47: Petri net discovered using log "log-flat" and the Heuristic Miner.

Excerpt of variants containing a log move (Exercise 3c)

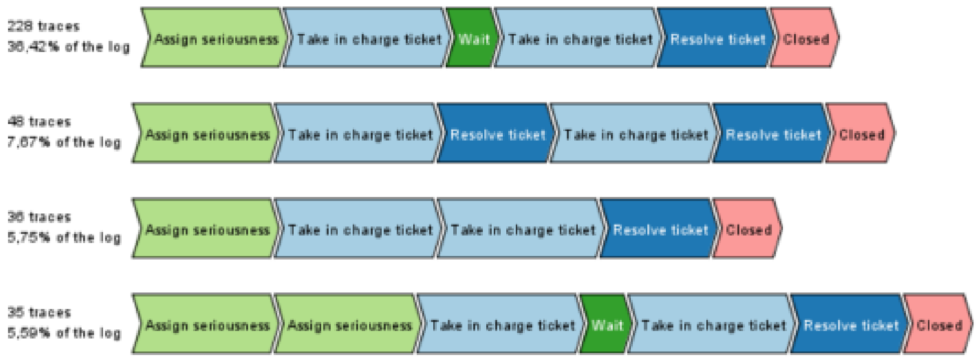


Figure 48: Excerpt of variants that contain a log move on *Take in charge ticket*

Decision Tree Workflow (Exercise 4b)

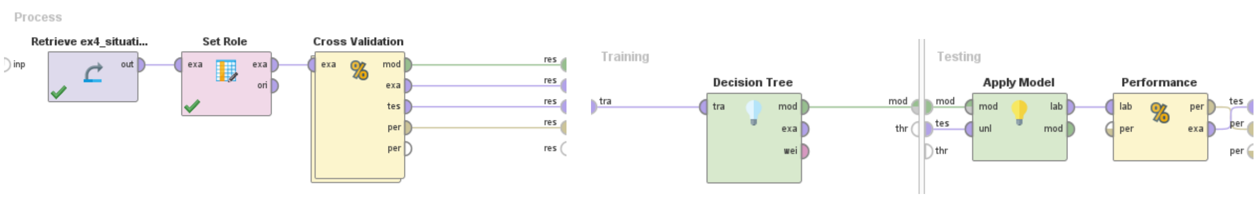
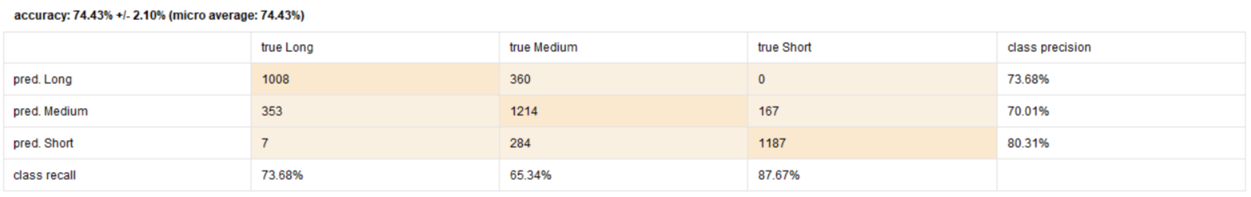


Figure 49: Screenshot of the process for training a Decision Tree in RapidMiner. The left sub-figure shows the overall process while the right sub-figure depicts the sub-process of cross validation.

Decision Tree Accuracy (Exercise 5d)



accuracy: 74.43% +/- 2.10% (micro average: 74.43%)

	true Long	true Medium	true Short	class precision
pred. Long	1008	360	0	73.68%
pred. Medium	353	1214	167	70.01%
pred. Short	7	284	1187	80.31%
class recall	73.68%	65.34%	87.67%	

Figure 50: Screenshot of the confusion matrix in RapidMiner showing information about precision, recall and accuracy of the trained decision tree.