UNIVERSITY OF MILAN

LM-18 - COMPUTER SCIENCE

BUSINESS INFORMATION SYSTEMS

BPI CHALLENGE 2020

Project supervisor: Prof. Ceravolo Paolo

Strudent:
Lorenzo Casati
Matr. Nr. 04413A

ACADEMIC YEAR 2022-2023

Contents

Chapter 1

Scenario

This project aims to develop the *BPI Challenge 2020*.
The challenge can be found at the related site: `https://www.tf-pm.org/competitions-awards/bpi-challenge/2020`

1.1 Description of the Challange

In many organizations, staff members travel for work. They travel to customers, to conferences or to project meetings and these travels are sometimes expensive. As an employee of an organization, you do not have to pay for your own travel expenses, but the company takes care of them.

At Eindhoven University of Technology (TU/e), this is no different. The TU/e staff travels a lot to conferences or to other universities for project meetings and/or to meet up with colleagues in the field. And, as many companies, they have procedures in place for arranging the travels as well as for the reimbursement of costs.

On a high level, we distinguish two types of trips, namely *domestic* and *international*:

- For *domestic trips*, no prior permission is needed, i.e. an employee can undertake these trips and ask for reimbursement of the costs afterwards.
- For *international trips*, permission is needed from the supervisor. This permission is obtained by filing a travel-permit and this travel permit should be approved before making any arrangements.

To get the costs for a travel reimbursed, a claim is filed. This can be done as soon as costs are actually payed (for example for flights or conference registration fees), or within two months after the trip (for example hotel and food costs which are usually payed on the spot).

1.2 The Data

For this *Business Process Intelligence Challenge*, the data is collected from the reimbursement process at TU/e. The files contain data from 2017 (only two departments) and 2018 the full TU/e.

The data is split into *travel permits* and several request types, namely *domestic declarations*, *international declarations*, *prepaid travel costs* and *requests for payment*, where the latter refers to expenses which should not be related to trips (think of representation costs, hardware purchased for work, etc.).

The data is anonymized and for all steps, only the role of the person executed the step is recorded. The resource recorded in the data is either the SYSTEM, a STAFF MEMBER or UNKNOWN, or, on occasion, the data is MISSING.
As said above, the collected data is split in five subsets:

- *Requests for Payment* (should not be travel related): 6,886 cases, 36,796 events;
- *Domestic Declarations*: 10,500 cases, 56,437 events;
- *Prepaid Travel Cost*: 2,099 cases, 18,246 events;
- *International Declarations*: 6,449 cases, 72151 events;
- *Travel Permits* (including all related events of relevant prepaid travel cost declarations and travel declarations): 7,065 cases, 86,581 events;

1.3 The Process Flow

The *domestic and international declarations*, *pre-paid travel costs* and *requests for payment* all follow a similar process flow. After submission by the employee, the request is sent for approval to the travel administration. If approved, the request is then forwarded to the budget owner and after that to the supervisor. If the budget owner and supervisor are the same person, then only one of the these steps it taken. In some cases, the director also needs to approve the request.

In all cases, a rejection leads to one of two outcomes. Either the employee resubmits the request, or the employee also rejects the request.

If the approval flow has a positive result, the payment is requested and made.

The travel permits follow a slightly different flow as there is no payment involved. Instead, after all approval steps a trip can take place, indicated with an estimated start and end date. These dates are not exact travel dates, but rather estimated by the employee when the permit request is submitted. The actual travel dates are not recorded in the data, but should be close to the given dates in most cases.

After the end of a trip, an employee receives several reminders to submit a travel declaration.

After a travel permit is approved, but before the trip starts, employees can ask for a reimbursement of pre-paid travel costs. Several requests can be submitted independently of each other. After the trip ends, an international declaration can be submitted, although sometimes multiple declarations are seen for specific cases.

It's important to realize that the process described above is the process for 2018. For 2017, there are some differences as this was a pilot year and the process changed slightly on several occasions.

Chapter 2

Goals and Challenges

The goal of this project is to analyze the data and to answer some questions of the challenge. The considered questions that have been answered, are:

- *What is the throughput of a travel declaration from submission (or closing) to paying?*
- *Is there are difference in throughput between national and international trips?*
- *Where are the bottlenecks in the process of a travel declaration?*
- *Where are the bottlenecks in the process of a travel permit (note that there can be multiple requests for payment and declarations per permit)?*
- *How many travel declarations get rejected in the various processing steps and how many are never approved?*
- *How many travel declarations are booked on projects?*
- *Are there any double payments?*

Afterwards, through the application of some miner models and the analysis of the comparisons of the results of those models, the most appropriate model that may be apply to this context has been generated.
The applied miner models used are:

- *Alpha Miner*
- *Inductive Miner*

- *Inductive Miner Infrequent*
- *Inductive Miner Directly-Follows*
- *Heuristic Miner*

The evaluation of the miner models has been done with the calculation of the following parameters: *fitness*, *precision*, *generalization* and *simplicity*.

Chapter 3

Knowledge Uplift Trial

The following table shows all the steps performed to reach the results. In each step is shown the input, the analysis and its type, and the output.

	Input	**Acquired Knowledge**		**Output**
		Analytics and Models	**Type**	
Step 1	5 Event Log	Read logs and analysis	Descriptive	Statistics
Step 2	Step 1	Filtering data from 2018	Prescriptive	Filtered event logs
Step 3	Step 2	Throughput of a travel declaration	Descriptive	Statistics
Step 4	Step 2 Step 3	Throughput difference between national and international trips	Descriptive	Statistics
Step 5	Step 2 Step 3	Bottlenecks in the process of a travel declaration	Descriptive	Statistics
Step 6	Step 2 Step 3	Bottlenecks in the process of a travel permit	Descriptive	Statistics
Step 7	Step 2	Rejected and never approved travel declarations	Descriptive	Statistics
Step 8	Step 2	Booked travel declarations	Descriptive	Statistics
Step 9	Step 2	Double payments	Descriptive	Statistics
Step 10	Step 2	Process Discovery	Prescriptive	Models and Nets
Step 11	Step 10	Quality evaluation	Descriptive	Statistics
Step 12	Step 2	Analysis on variants	Descriptive	Statistics
Step 13	Step 2 Step 12	Process Discovery with variants	Prescriptive	Models and Nets
Step 14	Step 13	Quality evaluation with variants	Descriptive	Statistics

Chapter 4

Materials and Methods

The tools used to perform the goals of this project are:

- *Google Colab*: based on the open source project Jupiter, it is a free platform that allows anyone to write and execute python code through a browser.
- *Pm4py*: it is an open source process mining platform written in Python.
- *Disco*: it is a process mining tool that allows an easy and flexible discovery of processes with its visualization and filtering capabilities. It is able to deal in an easy way event logs, complex model and data conversion and filtering.

Google Colab was used because it can be utilized through with any browser, and in this way no installation is required. The *pm4py* library was used to perform the mining process required to answer the questions of the challenge, while the *Disco* tool was useful to check and compare some results and get additional informations to achieve some tasks.

Chapter 5

Solution

This chapter illustrates every steps performed and the analysis to reach the intended results.

5.1 Data evaluation

As first thing to do, the given data has to be analyzed in order to understand if they are good enough to be used as they are, or it is necessary to do some manipulation or filtering. This evaluation is fundamental because if an analysis is performed with not properly adjusted data, the results that will be produced may be not correct.
The following table and graph show the number of events in the provided dataset.

Domestic declarations	International declarations	Permit log	Prepaid travel cost	Request for payment
10500	6449	7065	2099	6886

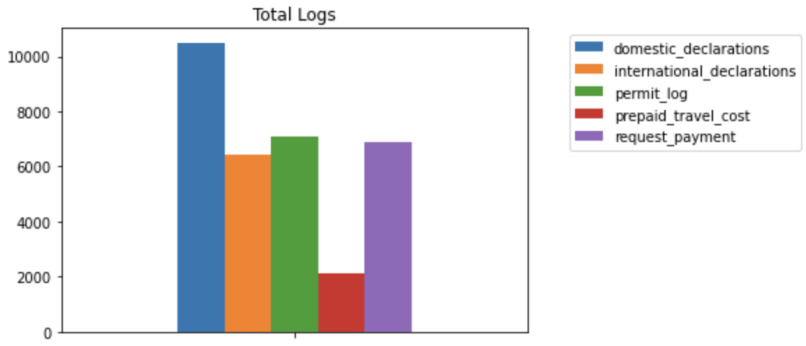


Figure 5.1: *Provided datasets*

The events of these datasets are logs from the year 2017. As said by the challenge and as reported in the section *1.2 The Data*, the data of the year 2017 are from only two departments. In order to used more coherent data, it is preferred to filter the partially data and utilize only the data start from 2018.

```
from pm4py.algo.filtering.log.timestamp import timestamp_filter
#
# Filtering data from 2018
xes_logs_from_2018 = {xes: timestamp_filter.filter_traces_contained(
        xes_logs[xes], "2018-01-01 00:00:00", "2022-12-31 23:59:59")
        for xes in xes_logs.keys()}
```

The above code uses the *pm4py* function *filter_traces_contained* to filter the events contained in the specified time interval. The events obtained are shown below.

Domestic declarations	International declarations	Permit log	Prepaid travel cost	Request for payment
8260	4952	5598	1776	5778

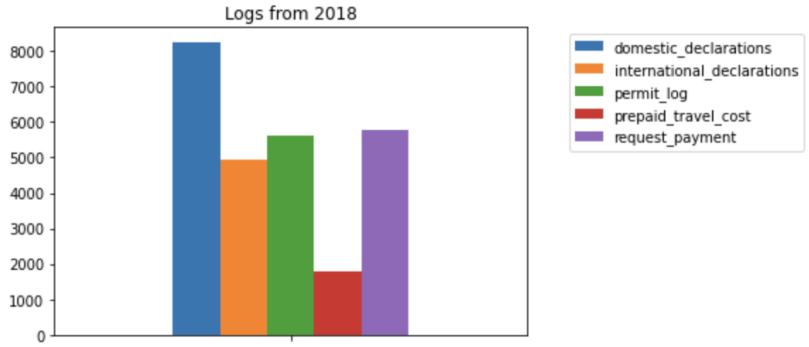


Figure 5.2: *Datasets starting from 2018*

So the data of 2017 are not considered.

Domestic declarations	International declarations	Permit log	Prepaid travel cost	Request for payment
2240	1497	1467	323	1108

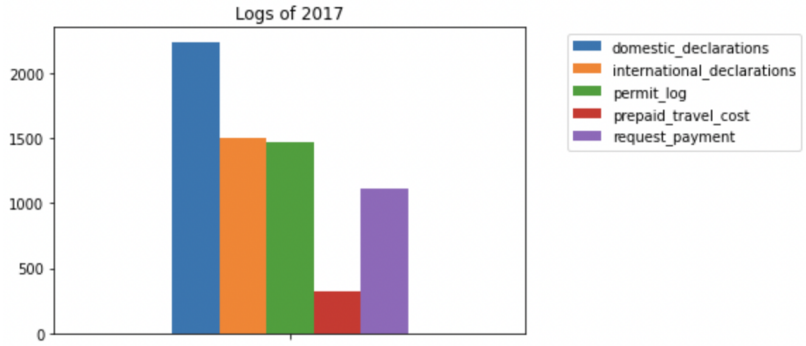


Figure 5.3: *Datasets of year 2017*

For more comprehension let's join the events together.

	Domestic declarations	International declarations	Permit log	Prepaid travel cost	Request for payment
Total cases	10500	6449	7065	2099	6886
Cases from 2018	8260	4952	5598	1776	5778
Cases of 2017	2240	1497	1467	323	1108

5.2 Questions answering

Now the data are ready to be analyzed, it is possibile to start answering some questions of the challenge.

5.2.1 Throughput

At first instance, the first answered questions are the ones related to the throughput. The throughput is meant in terms of time, so it refers to the total amount of time that it takes to run a particular process in its entirety from start to end.
The questions that are going to be answered now, are:

- *What is the throughput of a travel declaration from submission (or closing) to paying?*
- *Is there are difference in throughput between national and international trips?*

What is the throughput of a travel declaration from submission (or closing) to paying?

To calculate the throughput, first it was necessary to get the total duration of each cases for each subsets (*domestic declarations*, *international declarations*, etc...).

```
# Getting the duration from all cases
all_duration_cases = {xes: pm4py.get_all_case_durations(
        xes_logs_from_2018[xes]) for xes in xes_logs_from_2018.keys()}
```

Then for each subset the mean of the various durations was calculated.

```
# Getting calculate the mean from all the duration cases (in days)
mean_all_duration_cases = {xes: round(statistics.mean(
        all_duration_cases[xes])/(3600*24),2)
        for xes in all_duration_cases.keys()}
```

The mean is calculated in days because it is more clear than seconds to be understand.

	Mean throughput
Domestic declarations	11.56 days
International declarations	76.36 days
Permit log	84.15 days
Prepaid travel cost	36.60 days
Request for payment	12.35 days

Is there are difference in throughput between national and international trips?

Looking at the throughput of *domestic declarations* and the throughput of *international declarations*, there is a huge difference so, a deeper analysis is useful.

We're interest in find where the two type of declarations are different in throughput. So, the following function allows us to calculate the average throughput of the cases between the event *from_event* and the event *to_event*.

```
def throughtput_days_duration_between(logs, from_event, to_event):
        filtered_logs = pm4py.filter_between(logs, from_event, to_event)
        throughput_time_cases = pm4py.get_all_case_durations(filtered_logs)
        return statistics.mean(throughput_time_cases)/(24*3600)
```

Now, we try to calculate the throughput between the event *Declaration SUBMITTED by EMPLOYEE* and the event *Payment Handled*.

```
#
# Domestic Declarations
dd_throughput_days = throughtput_days_duration_between(
        xes_logs_from_2018["domestic_declarations"],
        "Declaration SUBMITTED by EMPLOYEE",
        "Payment Handled")
#
# International Declarations
id_throughput_days = throughtput_days_duration_between(
        xes_logs_from_2018["international_declarations"],
        "Declaration SUBMITTED by EMPLOYEE",
        "Payment Handled")
```

	"Declaration SUBMITTED by EMPLOYEE" ⇒ "Payment Handled"
Domestic declarations	11.62 days
International declarations	15.13 days

The difference is not so big, but it is possibile investigate a little bit more deeper in order to know exactly where the difference is. To do so, the steps from the *Declaration SUBMITTED by EMPLOYEE* to *Payment Handled*, were split in substeps:

- *Declaration SUBMITTED by EMPLOYEE* to *Declaration FINAL_APPROVED by SUPERVISOR*
- *Declaration FINAL_APPROVED by SUPERVISOR* to *Request Payment*
- *Request Payment* to *Payment Handled*

	Domestic declarations	**International declarations**
"Declaration SUBMITTED by EMPLOYEE" ⇒ "Payment Handled"	11.62 days	15.13 days
"Declaration SUBMITTED by EMPLOYEE" ⇒ "Declaration FINAL_APPROVED by SUPERVISOR"	5.27 days	8.58 days
"Declaration FINAL_APPROVED by SUPERVISOR" ⇒ "Request Payment"	2.84 days	2.86 days
"Request Payment" ⇒ "Payment Handled"	3.45 days	3.42 days

As we can see, the main difference in throughput between the *domestic declarations* and the *international declarations*, is how long it takes from the submission of the declaration by the employee to the final approval by the supervisor.

5.2.2 Bottleneck

In this section, we will find the answers of questions: *Where are the bottlenecks in the process of a travel declaration?* and *Where are the bottlenecks in the process of a travel permit?*

The bottlenecks are events, processes or tasks that slow down the entire process. In order to find the bottlenecks in the process of travel declaration, the *Disco* was used. Looking at the process map that visualizes the actual flow of the process based on the imported log data, using the performance mode in order to show the mean duration of the processes, we can understand where the bottlenecks are.

Domestic Declarations

The following figure shows the process map of the *domestic declarations* log data.

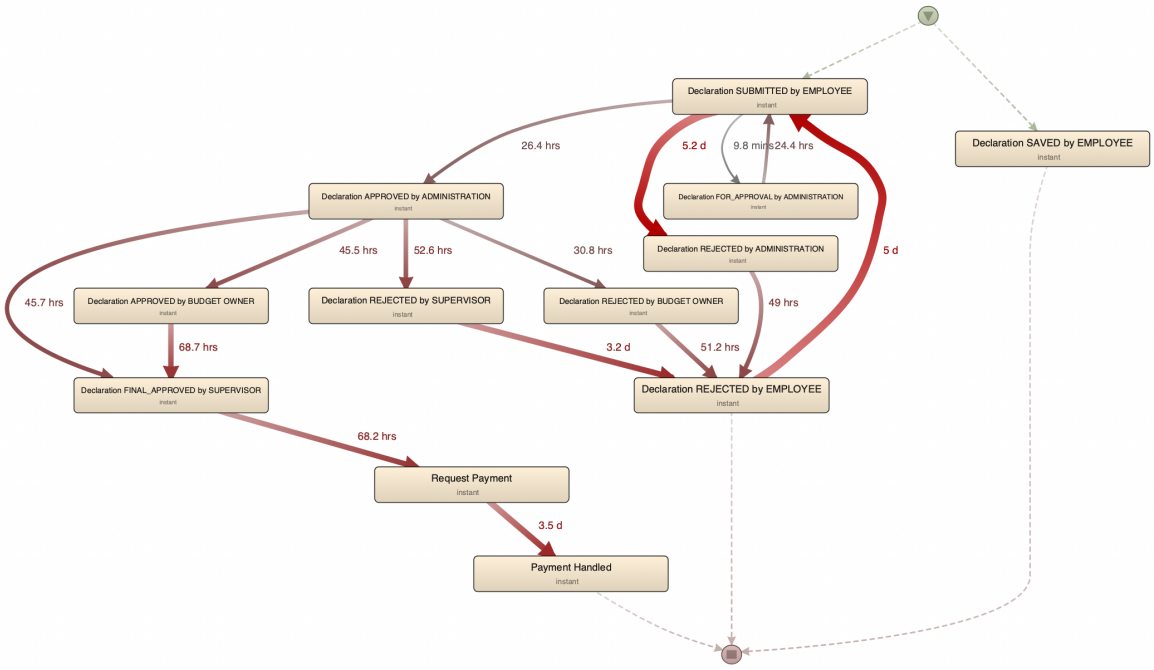


Figure 5.4: *Process map of Domestic Declarations*

During the calculation of the throughput, we found that the throughput is high between the *Declaration SUBMITTED by EMPLOYEE* and *Declaration FINAL_APPROVED by SUPERVISOR*. In fact, as we can see in the process map there are some bottlenecks between those events:

- *Declaration SUBMITTED by EMPLOYEE* to *Declaration REJECTED by ADMINISTRATION*
- *Declaration REJECTED by SUPERVISOR* to *Declaration REJECTED by EMPLOYEE*
- *Declaration REJECTED by EMPLOYEE* to *Declaration SUBMITTED by EMPLOYEE*

Another bottleneck is from *Request Payment* to *Payment Handled*, and the mean duration is exactly what we calculated before.

International Declarations

For the *international declarations* log data, we filter activities in order to get activities about declarations.

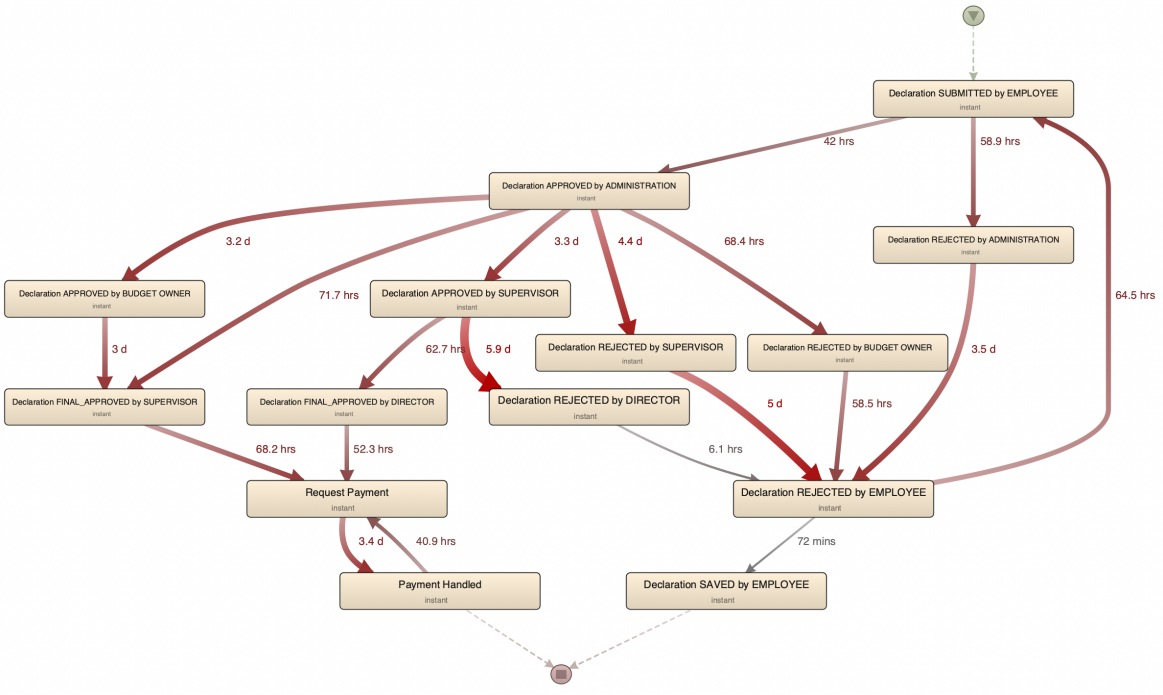


Figure 5.5: *Process map of International Declarations*

Looking at the process map, the bottlenecks are:

- *Declaration APPROVED by ADMINISTRATION* to *Declaration REJECTED by SUPERVISOR*
- *Declaration APPROVED by ADMINISTRATION* to *Declaration APPROVED by SUPERVISOR*
- *Declaration APPROVED by ADMINISTRATION* to *Declaration APPROVED by BUDGET OWNER*
- *Declaration REJECTED by SUPERVISOR* to *Declaration REJECTED by EMPLOYEE*
- *Declaration APPROVED by SUPERVISOR* to *Declaration REJECTED by DIRECTOR*
- *Declaration REJECT by ADMINISTRATION* to *Declaration REJECTED by EMPLOYEE*
- *Request Payment* to *Payment Handled*

We calculated that the main slowdown is between the *Declaration SUBMITTED by EMPLOYEE* and *Declaration FINAL_APPROVED by SUPERVISOR*, and as we can see, there are bottlenecks in there.

Permit Log

For the *permit log* log data, we filter activities in order to get activities about permits. From the following figure, the process map show us the bottlenecks in the process of a travel permit:

- *Declaration FINAL_APPROVED by SUPERVISOR* to *Request Payment*
- *Declaration FINAL_APPROVED by SUPERVISOR* to *Permit REJECTED by MISSING*
- *Declaration FINAL_APPROVED by DIRECTOR* to *Request Payment*
- *Payment Handled* to *Request Payment*

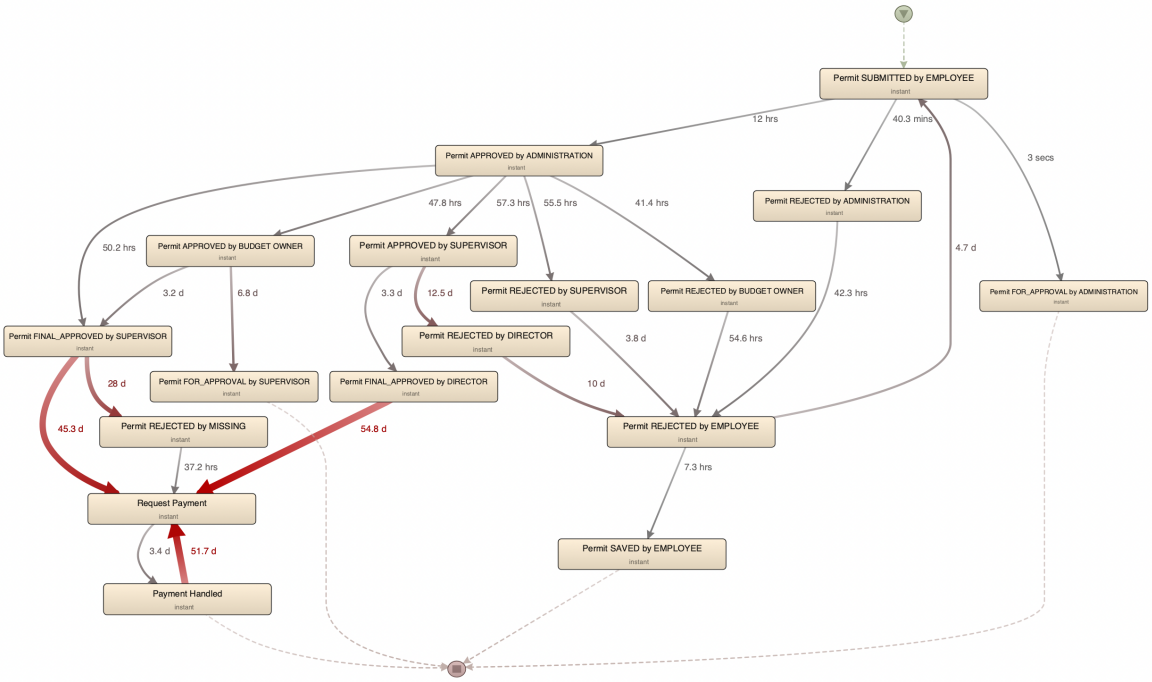


Figure 5.6: *Process map of Permit Log*

5.2.3 Rejected, never approved and booked declarations

The focus is on the questions:

- *How many travel declarations get rejected in the various processing steps and how many are never approved?*
- *How many travel declarations are booked on projects?*

How many travel declarations get rejected in the various processing steps and how many are never approved?

In order to get the number of the rejected travel declarations, the *pm4py* function *filter_event_attribute_values()* was used. This function allow us to filter the cases on a specified attribute that has a specific value. The value of the attribute can be specified as a set of value.

```
declarations = ['domestic_declarations', 'international_declarations']
rejections = ['Declaration REJECTED by SUPERVISOR',
              'Declaration REJECTED by BUDGET OWNER',
              'Declaration REJECTED by ADMINISTRATION',
              'Declaration REJECTED by EMPLOYEE']
```

```
#
# Rejected declarations
rejected_declarations_from_2018 = {xes: pm4py.filter_event_attribute_values(
        xes_logs_from_2018[xes], "concept:name", rejections,
        level="case", retain=True) for xes in declarations}
#
# Getting the count
rejected_declarations_from_2018_len = {xes: len(
        rejected_declarations_from_2018[xes]) for xes in declarations}
```

The results of the rejected declarations for *domestic declarations* and *international declarations* are displayed below:

	Domestic declarations	**International declarations**
Rejected declarations	1072	1351

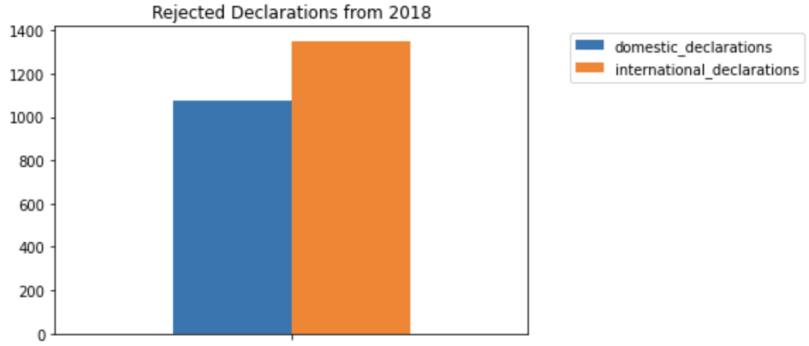


Figure 5.7: *Rejected declarations*

To find the never approved declarations, the approach is similar. Using the same function *filter_event_attribute_values()*, the never approved declarations were found. In this case the function was used to find the approved declaration, but setting the property *retain* as *false* in order to invert the search.

```
declarations = ['domestic_declarations', 'international_declarations']
rejections = ['Declaration REJECTED by SUPERVISOR',
        'Declaration REJECTED by BUDGET OWNER',
        'Declaration REJECTED by ADMINISTRATION',
        'Declaration REJECTED by EMPLOYEE']
#
# Never approved declarations
napproved_declarations_from_2018 = {xes: pm4py.filter_event_attribute_values(
        xes_logs_from_2018[xes],"concept:name", approved,
        level="case", retain=False) for xes in declarations}
```

```
#
# Getting the count
napproved_declarations_from_2018_len = {xes: len(
        napproved_declarations_from_2018[xes]) for xes in declarations}
```

	Domestic declarations	International declarations
Never approved declarations	357	16

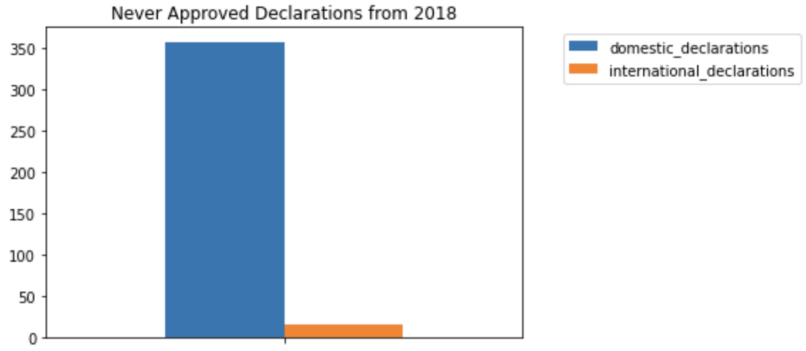


Figure 5.8: *Never approved declarations*

Same approach to find the booked declarations. Using the same function as before, the interested cases are the ones that have the event named *Payment Handled*.

```
declarations = ['domestic_declarations','international_declarations']
#
# Booked declaration
booked_declarations_from_2018 = {xes: pm4py.filter_event_attribute_values(
        xes_logs_from_2018[xes],"concept:name", "Payment Handled",
        level="case", retain=True) for xes in declarations}
#
# Getting the count
booked_declarations_from_2018_len = {xes: len(
        booked_declarations_from_2018[xes]) for xes in declarations}
```

	Domestic declarations	International declarations
Booked declarations	7903	4741

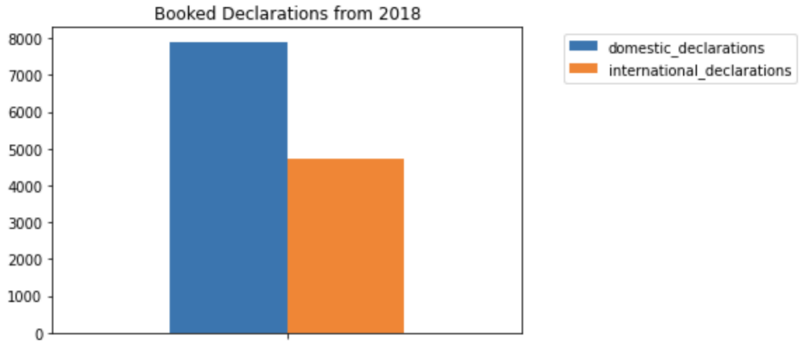


Figure 5.9: *Booked declarations*

5.2.4 Double payments

In this section, we're going to answer to the next question: *are there any double payments?*

To get the cases that have more than one payment, the function *filter_activities_rework()* of *pm4py* is very useful, because it returns the cases that have events with the specified attribute with the specified value, repeated for a certain number of times.

In this case, we search for cases that have the event *Payment Handled* repeated more than two times.

```
# Getting cases with double payments
double_payment_from_2018 = {xes: pm4py.filter_activities_rework(
        xes_logs_from_2018[xes], "Payment Handled", 2,
        activity_key='concept:name') for xes in xes_logs_from_2018.keys()}
#
# Counting the double payments cases
df_double_payment_from_2018 = pd.DataFrame({xes: len(
        double_payment_from_2018[xes])
        for xes in double_payment_from_2018.keys()}, index=[''])
```

	Double payments
Domestic declarations	0
International declarations	0
Permit log	1017
Prepaid travel cost	0
Request for payment	0

5.3 Process Discovery

The *Process Discovery* is a set of techniques that manually or automatically construct a representation of a business process. In process mining, the *Process Discovery* is performed applying inductive or selective algorithms to produces the *Process Model*. A *Process Model* may be a *direct-follows graph*, a *Petri Net* or a *Process Tree*.

5.3.1 Techniques

To perform the *Process Discovery*, the algorithms used are:

- *Alpha Miner*: is one of the most known algorithm in Process Discovery and it was the first algorithms that could adequately deal with concurrency. With an event log as the input, the α-algorithm produce a Petri net using the relations between the activities occurring in the event log.
- *Inductive Miner*: the basic idea is to use a divide-and-conquer approach and recursively cut the event log into sub-logs until a base case is found. Two process models can be derived a Petri Net and a Process Tree. We use three approaches:
  - *Inductive Miner*
  - *Inductive Miner Infrequent*
  - *Inductive Miner Directly-Follows*
- *Heuristic Miner*: The basic idea is that infrequent paths should not be incorporated into the model. It acts on the Directly-Follows Graph and this algorithm considers the frequencies of events and sequences when constructing a process model. The output of the Heuristics Miner is an Heuristics Net.

5.3.2 Quality Metrics

To evaluate the quality of the models, we use these following metrics:

- *Fitness*: aims to calculate how much of the behavior in the log is fitted by the process model.
- *Precision*: from the event log a model is discovered and this model should not allow for unrelated behavior than what it was seen in the log. The more the model differs from the log, the more the precision value is low.

- *Generalization*: is how much the discovered model generalize from the event log. A good generalization means all model subsets are frequently visited, whereas a bad generalization means some parts of the model are rarely used.
- *Simplicity*: based on Occam's Razor principle, a discovered model should not be unnecessary complex.
- *Token-Based Replay*: Token-based replay matches a trace and a Petri net model, starting from the initial place, in order to discover which transitions are executed and in which places we have remaining or missing tokens for the given process instance. Token-based replay is useful for Conformance Checking: indeed, a trace is fitting according to the model if, during its execution, the transitions can be fired without the need to insert any missing token.

5.3.3 The appropriate model

Now we want to find the most appropriate model applying the various techniques and compare them with the quality metrics. The generation of the various miner models was done with *pm4py* library.

Using all logs

```
from pm4py.algo.discovery.alpha import algorithm as alpha_miner
from pm4py.algo.discovery.inductive import algorithm as inductive_miner
from pm4py.algo.discovery.heuristics import algorithm as heuristics_miner
#
# Alpha miner
dd_alpha_miner = alpha_miner.apply(logs)
dd_am_statistics = get_statistics_model(logs, dd_alpha_miner)
#
# Inductive miner
dd_inductive_miner = pm4py.convert_to_petri_net(inductive_miner.apply(logs))
dd_im_statistics = get_statistics_model(logs, dd_inductive_miner)
#
# Inductive miner infrequent
dd_inductive_miner_f = pm4py.convert_to_petri_net(inductive_miner.apply(
        logs, variant=inductive_miner.Variants.IMf,
        parameters={inductive_miner.Variants.IMf:0.5}))
dd_imf_statistics = get_statistics_model(logs, dd_inductive_miner_f)
#
# Inductive miner direct-follows
dd_inductive_miner_d = pm4py.convert_to_petri_net(inductive_miner.apply(logs,
        variant = inductive_miner.Variants.IMd))
dd_imd_statistics = get_statistics_model(logs, dd_inductive_miner_d)
```

```
#
# Heuristic miner
dd_heuristic_miner = heuristics_miner.apply(logs)
dd_hm_statistics = get_statistics_model(logs, dd_heuristic_miner)
```

Here is the function to calculate the quality metrics.

```
# Function to get the statistics of a miner model
def get_statistics_model(log, miner_model):
    net, initial_marking, final_marking = miner_model
    replayed_traces = pm4py.conformance_diagnostics_token_based_replay(log, net,
    initial_marking, final_marking)
    return {
        'fitness': round(pm4py.fitness_token_based_replay(log, net,
            initial_marking, final_marking)['average_trace_fitness'], 2),
        'precision': round(pm4py.precision_token_based_replay(log, net,
            initial_marking, final_marking), 2),
        'generalization': round(generalization_evaluator.apply(log, net,
            initial_marking, final_marking), 2),
        'simplicity': round(simplicity_evaluator.apply(net), 2),
        'token_replay': round(statistics.mean([tr['trace_fitness']
             for tr in replayed_traces]),2)
        }
```

The following tables are the results of the quality metrics of the models.

	Domestic Declarations					International Declarations				
	AM	**IM**	**IMf**	**IMd**	**HM**	**AM**	**IM**	**IMf**	**IMd**	**HM**
Fitness	0.85	1.00	1.00	1.00	0.99	0.64	1.00	1.00	1.00	0.95
Precision	0.33	0.53	0.53	0.32	0.89	0.00	0.38	0.38	0.10	0.91
Generalization	0.88	0.89	0.89	0.91	0.88	0.85	0.87	0.87	0.86	0.79
Simplicity	0.56	0.67	0.67	0.60	0.67	0.41	0.64	0.64	0.39	0.54
Token replay	0.85	1.00	1.00	1.00	0.99	0.64	1.00	1.00	1.00	0.95

	Permit Log					Prepaid Travel Cost				
	AM	**IM**	**IMf**	**IMd**	**HM**	**AM**	**IM**	**IMf**	**IMd**	**HM**
Fitness	0.44	0.99	0.99	1.00	0.95	0.47	1.00	1.00	1.00	0.96
Precision	0.00	0.11	0.11	0.07	0.70	0.00	0.12	0.12	0.12	0.68
Generalization	0.83	0.89	0.89	0.84	0.60	0.89	0.87	0.87	0.90	0.65
Simplicity	0.60	0.60	0.60	0.37	0.49	0.21	0.57	0.57	0.41	0.54
Token replay	0.85	1.00	1.00	1.00	0.99	0.44	0.99	0.99	1.00	0.95

	Request for Payments				
	AM	**IM**	**IMf**	**IMd**	**HM**
Fitness	0.68	1.00	1.00	0.99	1.00
Precision	0.00	0.41	0.41	0.28	0.83
Generalization	0.76	0.81	0.81	0.82	0.73
Simplicity	0.43	0.60	0.60	0.57	0.60
Token replay	0.68	1.00	1.00	0.99	1.00

It's easy to see that the most appropriate model is the *Heuristic model* because according to the results of the metrics, in each subsets, it has the better results. In order to be sure that the *Heuristic model* is really the most appropriate model that fits the data, we're going to see its behavior using the variants of the logs.

Using variants

A variant is a set of cases that share the same control-flow perspective, so a set of cases that share the same classified events (activities) in the same order.
The *pm4py* library gives the method *get_variants(log)* to get all the variants in the event logs. In the following code, the function calculates the coverage of the top_k variants: it gets the variants from the logs, after it counts the occurrences for each types of variant and then it calculates the number of variants and the coverage.

```
def show_variants(logs, title, top_k):
        # Getting variants
        variants = pm4py.get_variants(logs)
        # Counting
        variants_count = {tuple_to_string(var): len(variants[var])
                for var in variants.keys()}
        df_variants_count = pd.DataFrame(variants_count, index=['count'])
                .transpose().sort_values(['count'], ascending=False)
        df_variants_count_k = df_variants_count.head(top_k)
        # Percentage
        total_variants = len(variants_count)
        total_cases = df_variants_count["count"].sum()
        k_cases = df_variants_count_k["count"].sum()
        # Printing
        print(f'Total cases: {total_cases}')
        print(f'Total number of variants: {total_variants}')
        print(f'Number of cases in top k={top_k} variants: {k_cases}')
        print(f'Percentage of cover: {round(k_cases*100/total_cases, 2)}%')
```

Knowing the coverage of the *top_k* variants, it is useful to understand how many

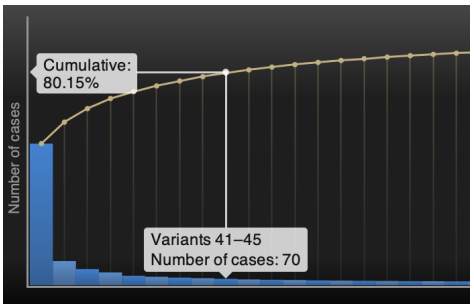
variants we should consider. We try to use a *top_k=5*.

	Total cases	Types of variants	Cases with k=5	Coverage
Domestic Declarations	8260	63	7786	94.26%
International Declarations	4952	523	2632	53.15%
Permit Log	5598	1115	2135	38.14%
Prepaid Travel Cost	1776	150	1199	67.51%
Request for Payment	5778	59	5432	94.01%

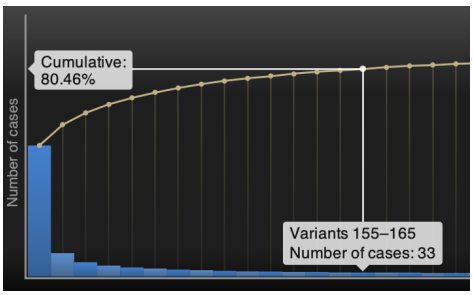
Using only 5 variants, the coverage for *International Declarations* and *Permit Log* is not so good, that is because they have many variants. Instead, using *top_k=10*, the coverage of *Permit Log* reaches almost 50%.

	Total cases	Types of variants	Cases with k=10	Coverage
Domestic Declarations	8260	63	7786	94.26%
International Declarations	4952	523	3038	61.35%
Permit Log	5598	1115	2750	49.12%
Prepaid Travel Cost	1776	150	1433	80.69%
Request for Payment	5778	59	5606	97.02%

As we can see, using Disco, to have a coverage of 80% for *International Declarations* we have to consider 41 or more variants, instead for *Permit Log* is necessary at least 155 variants.



(a) *International Declarations Variants*



(b) *Permit Log Variants*

Now using top_k=10 and the *pm4py* function *filter_variants_top_k(log, top_k)*, the top 10 variants are returned. On these variants we apply the Discovery Process and then we evaluate the quality of the models.

	Domestic Declarations					International Declarations				
	AM	**IM**	**IMf**	**IMd**	**HM**	**AM**	**IM**	**IMf**	**IMd**	**HM**
Fitness	0.85	1.00	1.00	0.94	0.99	0.76	1.00	1.00	1.00	1.00
Precision	1.00	0.78	0.78	0.35	0.99	1.00	0.29	0.29	0.09	0.87
Generalization	0.97	0.97	0.97	0.97	0.97	0.96	0.96	0.96	0.96	0.92
Simplicity	0.67	0.73	0.73	0.70	0.79	0.54	0.71	0.71	0.43	0.71
Token replay	0.85	1.00	1.00	0.94	0.99	0.76	1.00	1.00	1.00	1.00

	Permit Log					Prepaid Travel Cost				
	AM	**IM**	**IMf**	**IMd**	**HM**	**AM**	**IM**	**IMf**	**IMd**	**HM**
Fitness	0.73	1.00	1.00	1.00	1.00	0.76	1.00	1.00	0.98	0.97
Precision	0.00	0.28	0.28	0.08	0.86	1.00	0.51	0.51	0.25	0.60
Generalization	0.96	0.97	0.97	0.97	0.93	0.95	0.96	0.96	0.95	0.90
Simplicity	0.63	0.71	0.71	0.49	0.74	0.45	0.72	0.72	0.58	0.69
Token replay	0.73	1.00	1.00	1.00	1.00	0.76	1.00	1.00	0.98	0.97

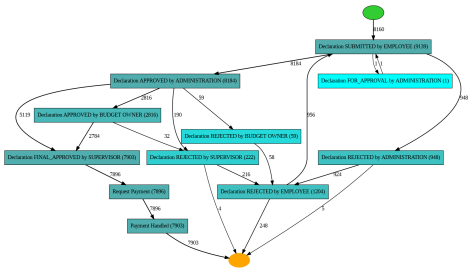
	Request for Payments				
	AM	**IM**	**IMf**	**IMd**	**HM**
Fitness	0.85	1.00	1.00	0.94	0.99
Precision	1.00	0.79	0.79	0.34	0.98
Generalization	0.94	0.96	0.96	0.95	0.95
Simplicity	0.60	0.70	0.70	0.67	0.75
Token replay	0.85	1.00	1.00	0.94	0.99

Evaluating the results we can say the performance are better now and, as before, the *Heuristic Miner* is the best fitted model.

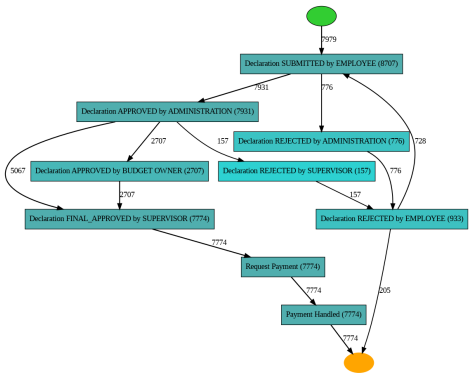
Graphical Models

This section will illustrate for each subsets the *Heuristic Net* and the *Petri Net* of the event logs and the variants.

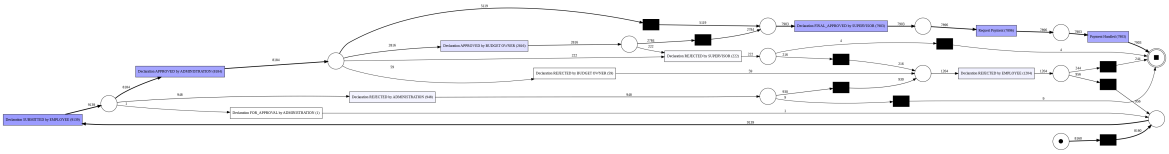
Domestic Declarations



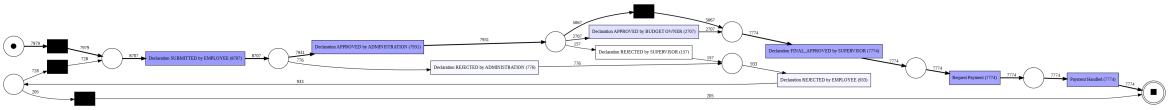
(c) *Heuristic Net of Domestic Declarations*



(d) *Heuristic Net of top_10 variants*

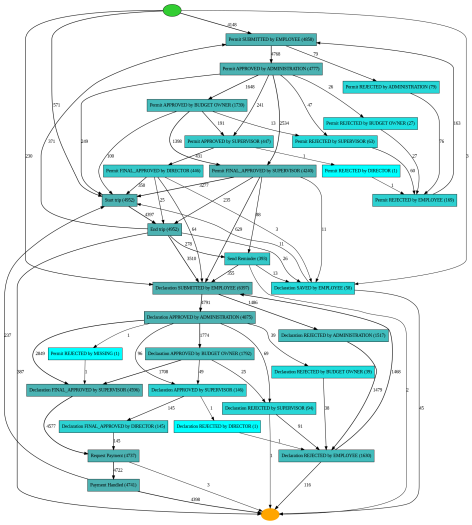


(e) *Petri Net of Domestic Declarations*

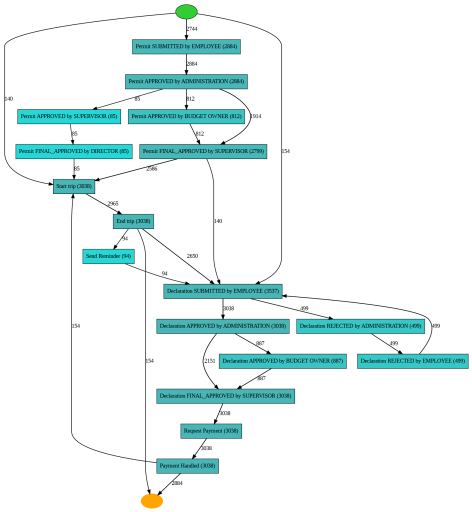


(f) *Petri Net of top_10 variants*

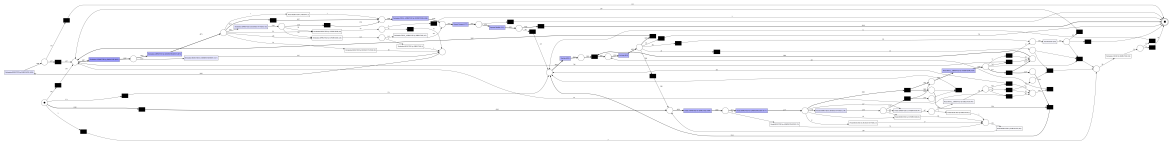
International Declarations



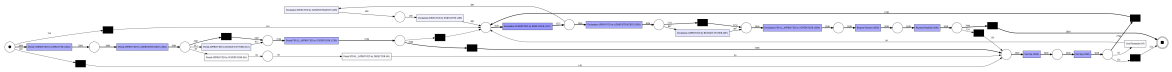
(g) *Heuristic Net of International Declarations*



(h) *Heuristic Net of top_10 variants*

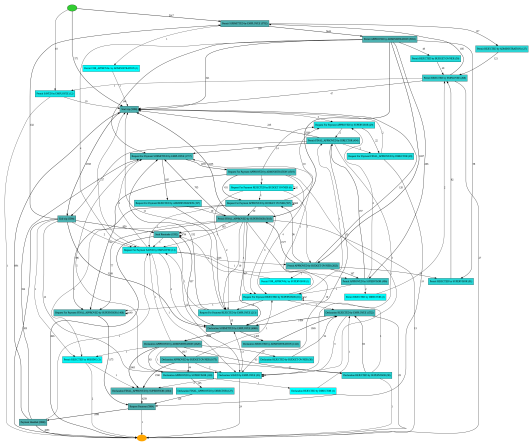


(i) *Heuristic Net of International Declarations*

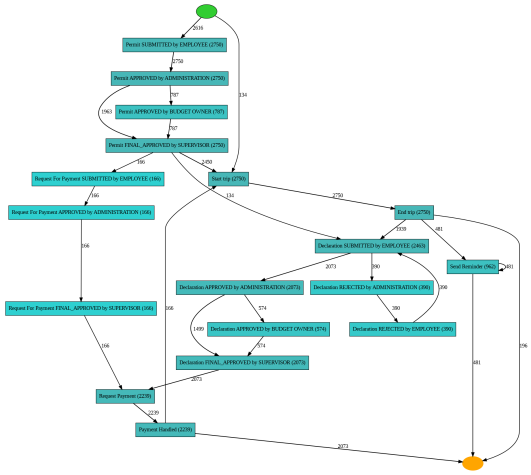


(j) *Petri Net of top_10 variants*

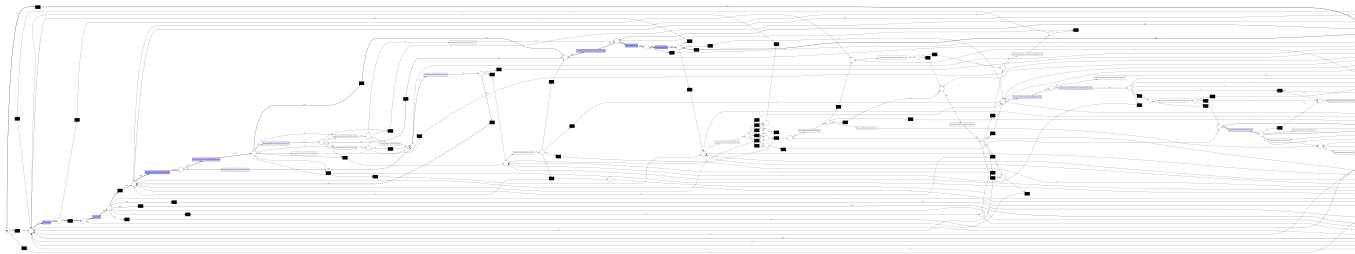
Permit Log



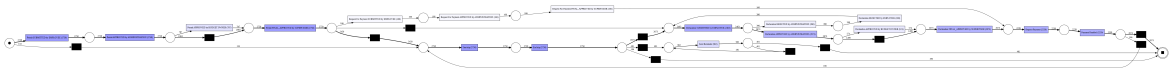
(k) *Heuristic Net of Permit Log*



(l) *Heuristic Net of top_10 variants*

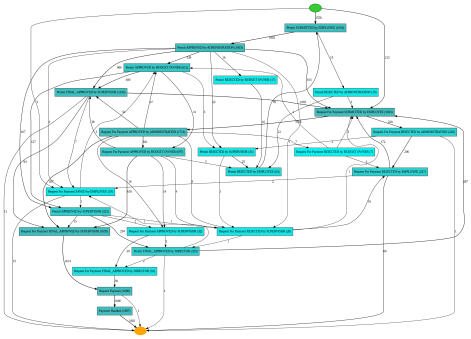


(m) *Heuristic Net of Permit Log*

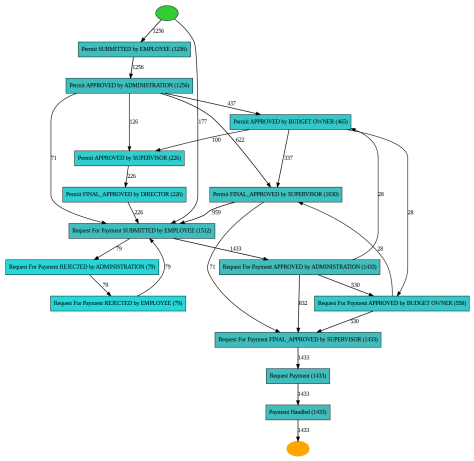


(n) *Petri Net of top_10 variants*

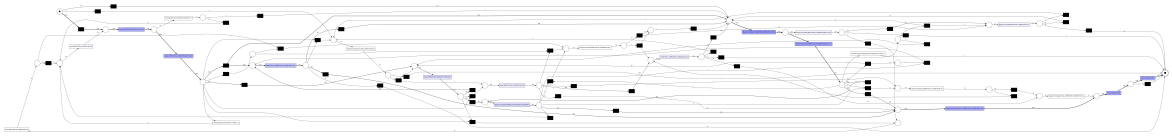
Prepaid Travel Cost



(o) *Heuristic Net of Prepaid Travel Cost*



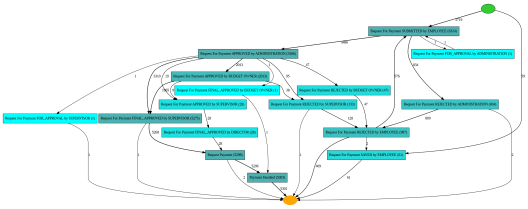
(p) *Heuristic Net of top_10 variants*



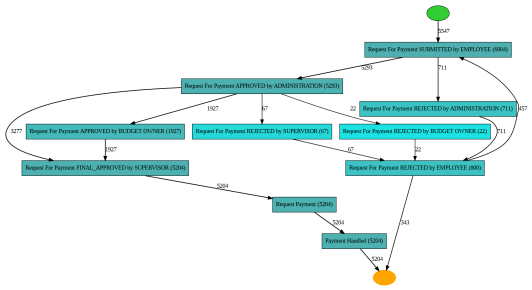
(q) *Heuristic Net of Prepaid Travel Cost*

(r) *Petri Net of top_10 variants*

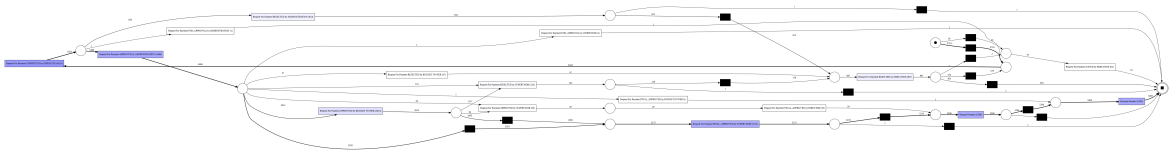
Request for Payment



(s) *Heuristic Net of Request for Payment*



(t) *Heuristic Net of top_10 variants*



(u) *Heuristic Net of Request for Payment*

(v) *Petri Net of top_10 variants*

Chapter 6

Conclusions

This report aims to analyze and to answer the questions chosen from the questions given by the challenge, and at the end, to find the best model could be applied to the context.

During the analysis, some problems were found. We found that a travel declaration may get stuck in some bottlenecks with even considerable loss of time, either to be rejected or approved. This may be because the many steps that the travel declaration has to go through. A possibile solution may be to use a simpler process with fewer events in a way to make the whole process more efficient and faster. It was also discovered that there is a waste of money due to multiple same payments. We saw that there are a thousand of double payments, and it is necessary a method to check if a declaration has been payed or not.

These problems found have to be solved both to avoid any kind of waste, especially of time and money, and also to improve the business process.