



UNIVERSITÀ DEGLI STUDI DI MILANO

## BIP CHALLENGE 2020

**Professor:** Paolo Ceravolo

Bocchino Daniele  
Matricola: 991031

July 2022

# Indice

<b>1</b>	<b>Description of the case study</b>	<b>1</b>
1.1	Description of Challenge . . . . .	1
1.2	The Data of Challenge . . . . .	2
1.3	The Process Flow . . . . .	2
<b>2</b>	<b>Organisational goals</b>	<b>3</b>
<b>3</b>	<b>Knowledge Uplift Trail</b>	<b>5</b>
<b>4</b>	<b>Project Results</b>	<b>6</b>
4.1	Data Visualization . . . . .	6
4.2	Temporal Filtering . . . . .	7
4.2.1	Temporal Filtering after 2018 . . . . .	7
4.2.2	Temporal Filtering before 2018 . . . . .	8
4.3	Throughput & Bottleneck . . . . .	8
4.3.1	Throughput & Bottleneck with PM4PY . . . . .	9
4.3.2	Throughput & Bottleneck with DISCO . . . . .	10
4.4	Declaration Rejected & Travel Booked . . . . .	14
4.4.1	Never Approved . . . . .	15
4.5	Double Payment . . . . .	15
4.6	Process Miner . . . . .	16
4.6.1	The Models . . . . .	16
4.6.2	The Evaluation . . . . .	16
4.6.3	Best Model . . . . .	17
4.6.4	Graphic Comparison Models . . . . .	20
<b>5</b>	<b>Conclusions</b>	<b>23</b>
5.1	Problems found . . . . .	23
5.2	Possible Solution . . . . .	23

# Capitolo 1

## Description of the case study

This project consists of the development of the BPI challenge 2020.  
you can view the code for this report at this github link:

[Bocchino Daniele BIS Project](#)

### 1.1 Description of Challenge

This challenge focuses on business travel and how organizations can schedule business travel. In fact, staff members in many organizations travel for business.

In general, 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. there are two types of travel: domestic and international.

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:

- **Domestic trips** are simple trips and no prior permission is needed. For example, the employees can undertake these trips and ask for reimbursement of the costs afterwards.
- **International trips** required a few complex procedure, for this kind of 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.

## 1.2 The Data of Challenge

<sup>1</sup> 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.). Within the dataset 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.

The datasets include :

- 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

*Domestic declarations, 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 these steps is taken. In some cases, the director also needs to approve the request.

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. <sup>2</sup>

---

<sup>1</sup>The dataset is available at this page : [Dataset BPI Challenge 2020](#)

<sup>2</sup>The challenge is available at this page : [BPI Challenge 2020](#)

# Capitolo 2

## Organisational goals

The purpose of this project was to analyze the data in detail and answer some questions on the challenge page. Within the challenge description it is indicated how the dataset is not complete for the year 2017. For this reason, I preferred to use only the year 2018 for the entire analysis.

At first I checked the entire dataset and tracked the number of logs present for each of the five files.

With the results obtained, I tried to answer some of the questions suggested in the challenge sheet. In particular, I answered the following questions:

- 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 ?
- 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?

For the completion of some of these questions, I used the Disco software and the python library *PM4PY*.

After answering these questions, I used process mining functions provided by *PM4PY* to generate and compare some models including :

- **Alpha Miner**
- **Inductive Miner**
- **Inductive Miner infrequent**
- **Inductive Miner directly-follows**
- **Heuristic Miner**

With the results obtained from these models, I generated a Petri net and calculated :

- **Fitness**
- **Precision**
- **Generalization**
- **Simplicity**

With this information I was able to deduce what was the best model to apply to this context.

For the development of this challenge I used :

- **PM4PY** : An open source platform for process mining written in python
- **DISCO** : A process mining tool capable of handling event logs, complex patterns and data filtering.

For the implementation of the code and functions of PM4PY, I used :

- **Google COLAB** : is a product of Google Research. Colab allows anyone to write and execute arbitrary python is particularly well suited for machine learning, data analysis and training.

## Capitolo 3

# Knowledge Uplift Trail

	Input	Analytics & Models	Type	Output
<b>Step 1</b>	5 Logs	Read file .xes and show plot with total event logs	Descriptive	Statistical result
<b>Step 2</b>	Step 1	Temporal Filtering after 2018	Prescriptive	Filtered event logs
<b>Step 3</b>	Step 2	Temporal Filtering before 2018	Prescriptive	Filtered event logs
<b>Step 4</b>	Step 3	Throughput of a travel declaration from submission to paying	Descriptive	Temporal statistical result
<b>Step 5</b>	Step 3	Difference in throughput between national and international trips	Descriptive	Temporal statistical result
<b>Step 6</b>	Step 2	The bottlenecks in the process of a travel declaration	Descriptive	Temporal statistical result
<b>Step 7</b>	Step 2	The bottlenecks in the process of a travel permit	Descriptive	Temporal statistical result
<b>Step 8</b>	Step 2 Step 3	Travel declarations rejected and never approved	Descriptive	Statistical result
<b>Step 9</b>	Step 2 Step 3	Travel Declarations are booked on projects	Descriptive	Statistical result
<b>Step 10</b>	Step 2 Step 3	Are there any double payments	Descriptive	Statistical result
<b>Step 11</b>	Step 2	Find models with Process Mining	Prescriptive	Petri net and other models
<b>Step 12</b>	Step 2	Filtering log by coverage percentage	Prescriptive	Statistical result
<b>Step 13</b>	Step 12	Find models with Process Mining using the variants	Prescriptive	Petri net and other models
<b>Step 14</b>	Step 11 Step 13	Model comparison and search for the best model	Descriptive	Models comparison and statistical result

**Tabella 1:** *Knowledge Uplift Trail*

In the table 1 I wanted to summarize and indicate all the steps taken in this analysis with their description and the inputs and outputs needed to implement them.

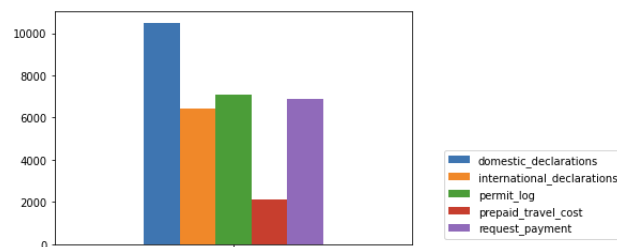
# Capitolo 4

## Project Results

In this chapter I will discuss the result of the analysis performed by integrating code images and graphs. Each section will refer to a step shown in the table in the previous chapter.

### 4.1 Data Visualization

Data visualization refers to the first step in the table in Figure 1.



DOMESTIC DECLARATION	INTERNATIONAL DECLARATION	PERMIT LOG	PREPAID TRAVEL COST	REQUEST PAYMENT
10500	6449	7065	2099	6886

**Figura 4.1:** *All logs in the dataset*

In Figure 4.1 I have shown the number of all logs and their events in the dataset. I wanted to report this data because the challenge emphasized that there was data for 2017. I only wanted to take and analyze the 2018 data.



## 4.2 Temporal Filtering

The temporal filter is the second and third step in the table in figure 1. This filter allows me to select events within a specific time range. I wanted to apply this filter in two variations:

1. To get the events after 2018
2. To get the events before 2018

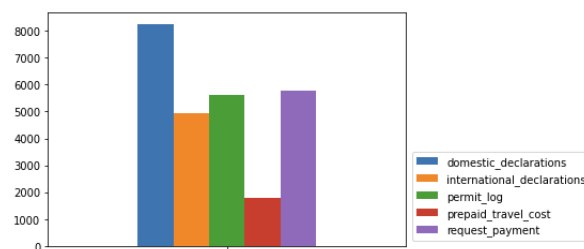
For both of these results, I used a function from the *PM4PY* library, in particular *filter\_time\_range*, which takes as input a dict of log, a time range, and a filtering method.

```
def temporal_filter_range(log, start_period, end_period):
    return pm4py.filter_time_range(log, start_period, end_period, mode='traces_contained')
```

Figura 4.2: Temporal Filter Function

Figure 4.2 shows the function I created to apply the temporal filter. This function allows me to apply the filter for logs before and after 2018. This way I can compare them and get the difference.

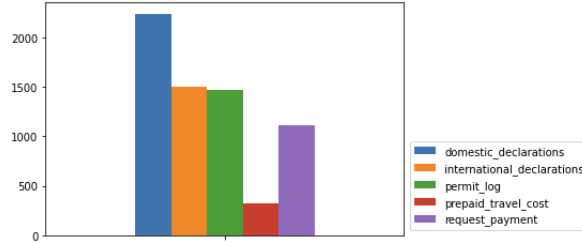
### 4.2.1 Temporal Filtering after 2018



DOMESTIC DECLARATION	INTERNATIONAL DECLARATION	PERMIT LOG	PREPAID TRAVEL COST	REQUEST PAYMENT
8260	4952	5598	1776	5778

Figura 4.3: Logs in the dataset after 2018.

### 4.2.2 Temporal Filtering before 2018



DOMESTIC DECLARATION	INTERNATIONAL DECLARATION	PERMIT LOG	PREPAID TRAVEL COST	REQUEST PAYMENT
2240	1497	1467	323	1108

**Figura 4.4:** Logs in the dataset before 2018.

By comparing Figures 4.3 and 4.4 we can see how the portion of the dataset I examined is the most consistent. From this point forward I will use only the portion of the dataset filtered from 2018 onward. In Figure 4.4, it is possible to see how the largest amount of data is concentrated starting in the year 2018.

	BEFORE 2018	AFTER 2018	TOTAL
DOMESTIC DECLARATION	2240	8260	10500
INTERNATIONAL DECLARATION	1497	4952	6449
PERMIT LOG	1467	5598	7065
PREPAID TRAVEL COST	323	1775	2099
REQUEST PAYMENT	1108	5778	6886

**Tabella 2:** Total log before and after 2018

## 4.3 Throughput & Bottleneck

In this section I will discuss steps : 4, 5, 6, 7 of the table 1. In particular, I will answer the challenge questions:

- *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)?

To answer these questions, I used the python library *PM4PY* and checked the results through the use of *DISCO*.

### 4.3.1 Throughput & Bottleneck with PM4PY

```
def get_throughput(log, from_, to):
    res = []

    for trace in log:
        timestamp = {'from': None, 'to': None }
        for e in trace:
            if e['concept:name'] == from_:
                timestamp.update({'from': e['time:timestamp'] })
            elif e['concept:name'] == to:
                timestamp.update({'to': e['time:timestamp'] })
                break

        if None not in timestamp.values():
            res.append(timestamp['to'] - timestamp['from'])

    return res
```

**Figure 4.5:** *function to calculate the throughput*

The function in figure 4.5 is a custom function for calculating throughput in the travel declaration process.

This function takes as input a log, and two strings corresponding to the event name. The purpose ti this function is to keep track of the event timestamp and then calculate the difference so as to compute the throughput.

I calculate the time it takes to go from Declaration SUBMITTED by EMPLOYEE to Payment Handled. I also wanted to calculate the time between the payment request and the completed payment, this is because that point is a bottleneck in *domestic declarations*, *international declarations* and *permit log*.

Min Throughput year 2018			
	domestic_declarations	international_declarations	permit_log
Declaration SUBMITTED by EMPLOYEE ↔ Request Payment	0 days 00:01:26	0 days 00:08:45	0 days 00:08:45
Declaration SUBMITTED by EMPLOYEE ↔ Payment Handled	1 days 01:33:30	1 days 00:55:52	0 days 00:15:35
Request Payment ↔ Payment Handled	0 days 02:19:06	0 days 00:39:32	0 days 01:15:38
Average Throughput year 2018			
	domestic_declarations	international_declarations	permit_log
Declaration SUBMITTED by EMPLOYEE ↔ Request Payment	6 days 20:02:17	8 days 20:08:45	8 days 15:09:06
Declaration SUBMITTED by EMPLOYEE ↔ Payment Handled	10 days 08:26:27	12 days 09:04:19	12 days 04:15:05
Request Payment ↔ Payment Handled	3 days 10:48:26	3 days 09:58:51	3 days 10:55:07
Median Throughput year 2018			
	domestic_declarations	international_declarations	permit_log
Declaration SUBMITTED by EMPLOYEE ↔ Request Payment	4 days 16:46:14	6 days 01:00:54	5 days 23:31:28
Declaration SUBMITTED by EMPLOYEE ↔ Payment Handled	7 days 07:48:38	9 days 06:27:53	9 days 05:20:25
Request Payment ↔ Payment Handled	3 days 05:07:54	3 days 05:38:01	3 days 05:42:01
Max Throughput year 2018			
	domestic_declarations	international_declarations	permit_log
Declaration SUBMITTED by EMPLOYEE ↔ Request Payment	287 days 20:50:27	427 days 22:19:35	349 days 15:46:59
Declaration SUBMITTED by EMPLOYEE ↔ Payment Handled	290 days 21:43:55	429 days 02:42:28	349 days 19:32:36
Request Payment ↔ Payment Handled	62 days 03:28:59	29 days 01:27:49	29 days 01:27:49

**Figura 4.6:** result of the min, average, median and max throughput in the specific event

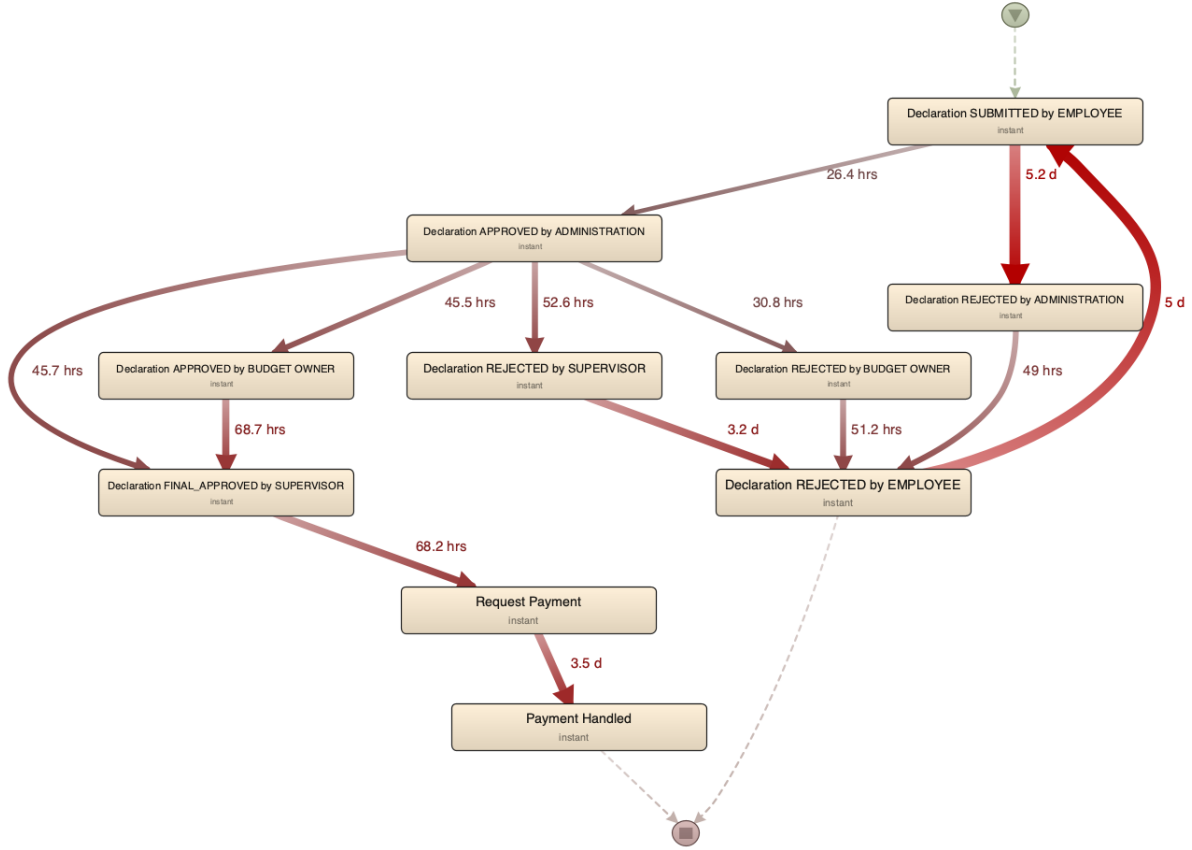
With the results shown in Figure 4.6 we can answer the first and second questions of the challenge. In fact, by comparing the results of the various tables, especially the middle table, we see the time it takes to go from a travel request to a completed payment. Furthermore, we can see how there is a difference, although not a very large one, between the throughput of the *domestic declarations* and that of the *international declarations*.

In the table I also wanted to show the difference between the time required between Declaration SUBMITTED by EMPLOYEE and Request Payment. The reason for this is due to the presence of a bottleneck. In fact, the time between request payment and its completion averages three and a half days, about 30% of the time required to accomplish the entire request. I used *DISCO* to confirm what is shown in the table in Figure 4.6.

### 4.3.2 Throughput & Bottleneck with DISCO

In this section I check the results obtained with PM4PY and identify all bottlenecks in each log

## 4.3.2.1 Domestic Declaration

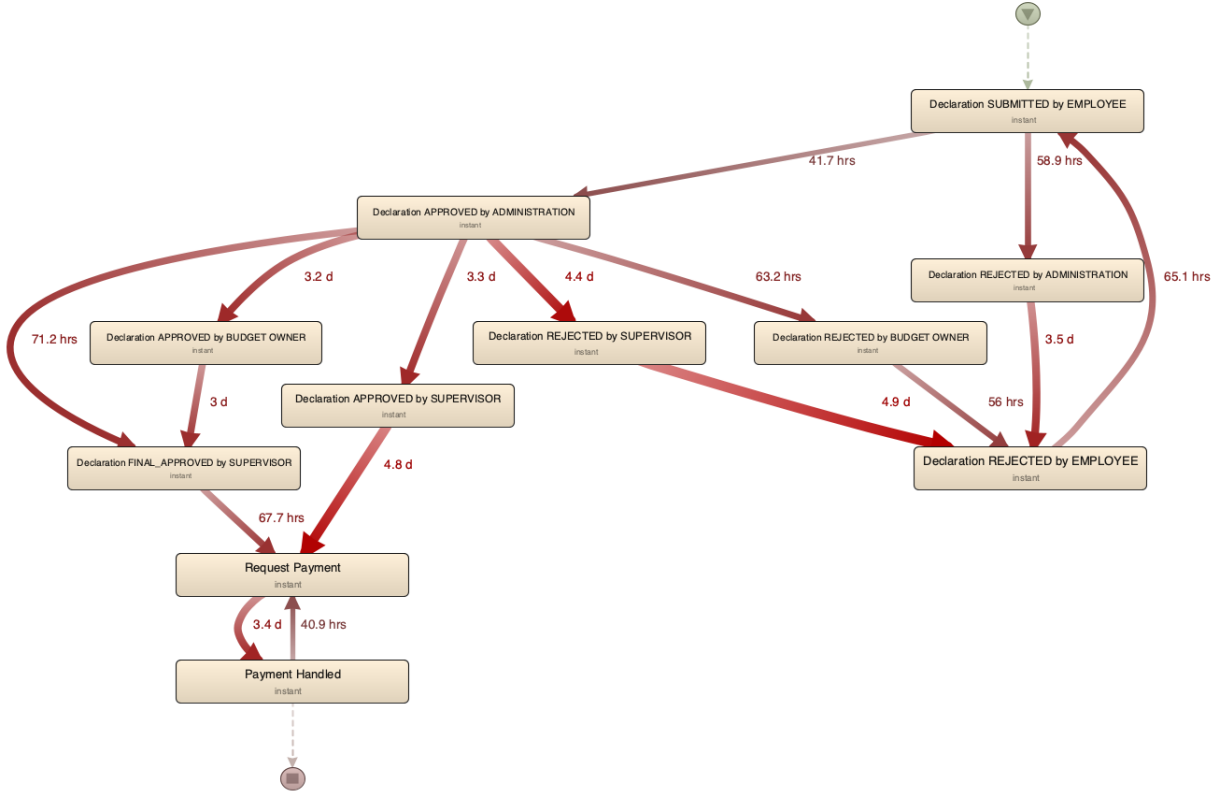


**Figure 4.7:** Throughput & bottleneck for domestic declaration

In Figure 4.7 we can see how the data obtained from *PM4PY* were correct and the average wait between *Request Payment* and *Payment Handled* were correct. Looking closely we can also see other bottlenecks in the declaration phase in the domestic declaration. In particular we have a slightly longer wait in the phases :

- Declaration SUBMITTED by EMPLOYEE & Declaration REJECTED by ADMINISTRATION
- Declaration REJECTED by EMPLOYEE & Declaration SUBMITTED by EMPLOYEE
- Declaration REJECTED by SUPERVISOR & Declaration REJECTED by EMPLOYEE

## 4.3.2.2 International Declaration

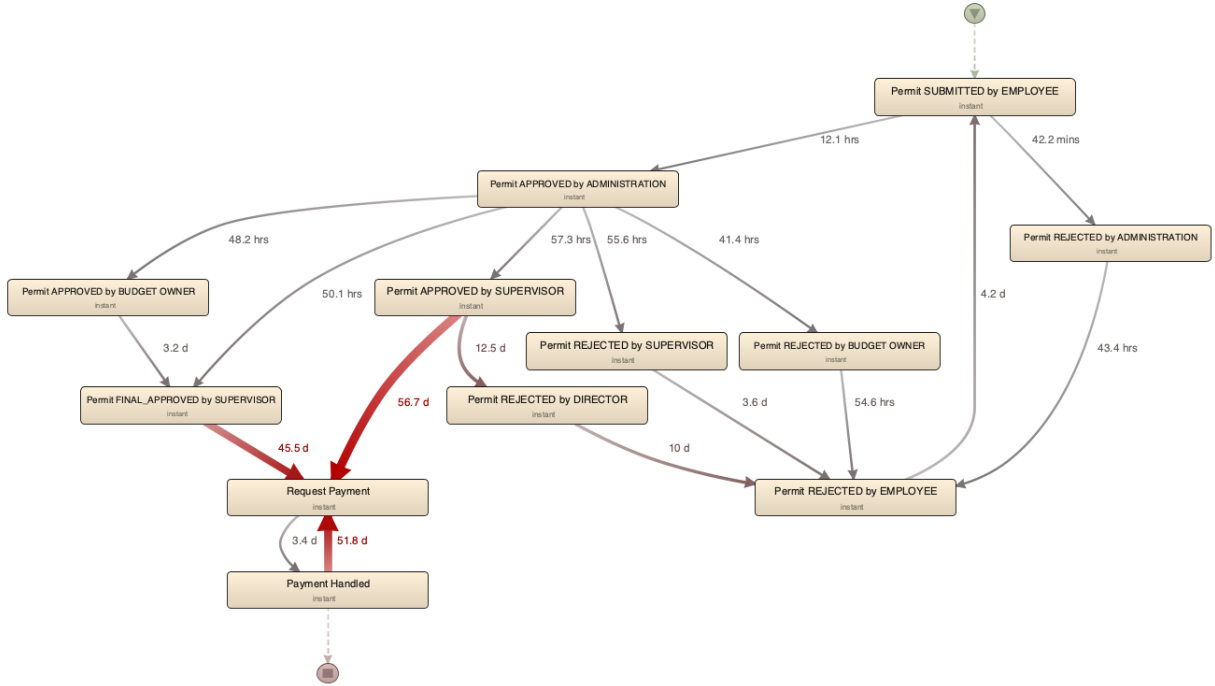


**Figura 4.8:** *Throughput & bottleneck for international declaration*

In Figure 4.8 we can see the results of *DISCO* for international declarations. Again they are in line with the data obtained from *PM4PY*. Compared to the domestic declaration we have an additional slowdown in the :

- Declaration APPROVED by ADMINISTRATION & Declaration APPROVED by BUDGET OWNER
- Declaration APPROVED by BUDGET OWNER & Declaration FINAL APPROVE by SUPERVISOR

### 4.3.2.3 Permit Log



**Figure 4.9:** Throughput & bottleneck for permit log

With *DISCO* we can also answer the question *Where are the bottlenecks in the process of a travel permit?*. In fact, as indicated by the table result in figure 4.6 the payment request is a bottleneck for the permit log, In addition to this the figure 4.9 shows us the presence of other bottle necks :

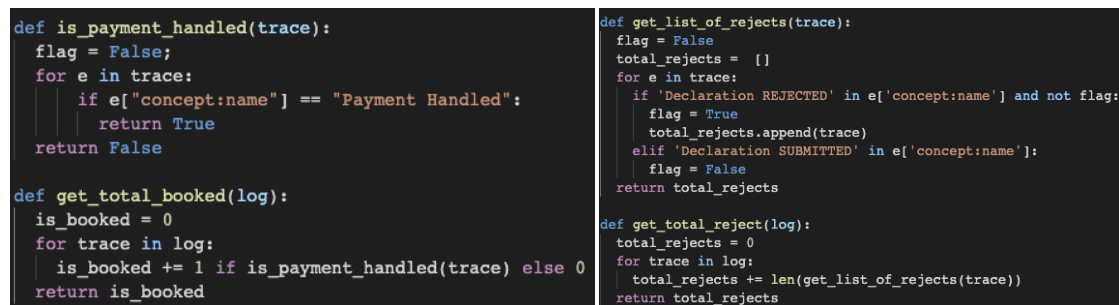
- Permit APPROVED by SUPERVISOR & Request Payment
- Permit FINAL APPROVE by SUPERVISOR & Request Payment
- Permit REJECTED by DIRECTOR & Permit REJECTED by EMPLOYEE
- Permit APPROVE by SUPERVISOR & Permit REJECTED by DIRECTOR

## 4.4 Declaration Rejected & Travel Booked

The steps 8 and 9 of the table in figure 1 answer 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?

To do this, I created some control functions that check whether an event contains the words "REJECTED", "FINAL APPROVED" or "Payment Handled" in its name, depending on what I want to achieve.



```
def is_payment_handled(trace):
    flag = False;
    for e in trace:
        if e["concept:name"] == "Payment Handled":
            return True
    return False

def get_total_booked(log):
    is_booked = 0
    for trace in log:
        is_booked += 1 if is_payment_handled(trace) else 0
    return is_booked

def get_list_of_rejects(trace):
    flag = False
    total_rejects = []
    for e in trace:
        if 'Declaration REJECTED' in e['concept:name'] and not flag:
            flag = True
            total_rejects.append(trace)
        elif 'Declaration SUBMITTED' in e['concept:name']:
            flag = False
    return total_rejects

def get_total_reject(log):
    total_rejects = 0
    for trace in log:
        total_rejects += len(get_list_of_rejects(trace))
    return total_rejects
```

Figura 4.10: Function for booked and rejected

These 4 functions in figure 4.4 I used to measure the number of booked and the number of rejected

	DOMESTIC DECLARATION	INTERNATIONAL DECLARATION
TOTAL LOG	8260	4952
TOTAL BOOKED	7903	4741
TOTAL REJECTED	1235	1656

Tabella 3: Result of Rejected declarations and Travel Booked



### 4.4.1 Never Approved

To check if there are any statements never to be approved, I created two functions very similar to the ones seen above and checked for an event with concept:name containing the word FINAL APPROVE, this way I kept track of all the travel declarations that have never been approved since 2018.

	DOMESTIC DECLARATION	INTERNATIONAL DECLARATION
TOTAL LOG	8260	4952
NEVER APPROVE	357	16

**Tabella 4:** *Function for calculated declarations never approved*

## 4.5 Double Payment

After calculating rejected statements, booked trips, and statements never approved, the next step in the table 1 includes the answer to the question about double payment. To do this again, I used *PM4PY*.

DOMESTIC DECLARATION	INTERNATIONAL DECLARATION	PERMIT LOG	PREPAID TRAVEL COST	REQUEST PAYMENT
0	0	1521	0	0

DOUBLE PAYMENT PERMIT LOG						
NUMBER OF DOUBLE PAYMENT	TOTAL DECLARATION SPENT	NUMBER OF OVERSPENT	OVERSPENT AMOUNT	NUMBER OF UNDERSPENT	UNDERSPENT AMOUNT	TOTAL OVERSPENT
1521	4478896.22 \$	693	1913662.91 \$	828	-745892.7 \$	1167770.22 \$

**Tabella 5:** *Result of double payment*

the results in table 5 show that double payment is present in the permit log and the amount of expenditure is very high. I also wanted to calculate the total overspending and non-overspending to better understand how much overspending is.

## 4.6 Process Miner

For the process mining phase, I used functions from the PM4PY library. The purpose of this step is to derive the best model that best fits the data.

### 4.6.1 The Models

**Alpha Miner:** The alpha miner is one of the most known Process Discovery algorithm and is able to find:

- A Petri net model where all the transitions are visible and unique and correspond to classified events (for example, to activities).
- An initial marking that describes the status of the Petri net model when a execution starts.
- A final marking that describes the status of the Petri net model when a execution ends.

**Inductive Miner:** The basic idea of Inductive Miner is about detecting a 'cut' in the log (e.g. sequential cut, parallel cut, concurrent cut and loop cut) and then recur on sublogs, which were found applying the cut, until a base case is found.

Inductive miner models usually make extensive use of hidden transitions, especially for skipping/looping on a portion on the model. Furthermore, each visible transition has a unique label (there are no transitions in the model that share the same label). There are three different implementation of inductive miner:

- Inductive Miner (IM)
- Inductive Miner infrequent (IMf)
- Inductive Miner directly-follows (IMd)

**Heuristic Miner:** Heuristics Miner is an algorithm that acts on the Directly-Follows Graph, providing way to handle with noise and to find common constructs (dependency between two activities, AND). The output of the Heuristics Miner is an Heuristics Net, so an object that contains the activities and the relationships between them. The Heuristics Net can be then converted into a Petri net.

### 4.6.2 The Evaluation

**Replay Fitness:** The calculation of the replay fitness aim to calculate how much of the behavior in the log is admitted by the process model. We propose two methods to calculate replay fitness, based on token-based replay and alignments respectively.

**Precision:** The idea underlying the two approaches is the same: the different prefixes of the log are replayed (whether possible) on the model. At the reached marking, the set of transitions that are enabled in the process model is compared with the set of activities that follow the prefix. The more the sets are different, the more the precision value is low. The more the sets are similar, the more the precision value is high.

**Generalization:** Generalization is the third dimension to analyse how the log and the process model match. Basically, a model is general whether the elements of the model are visited enough often during a replay operation (of the log on the model).

**Simplicity:** Simplicity is the fourth dimension for analyzing a process model. Simplicity considers only the Petri net model. The criterion to be used for simplicity is the inverse degree of the arc.

**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. If the reaching of the final marking is imposed, then a trace is fitting if it reaches the final marking without any missing or remaining tokens.

### 4.6.3 Best Model

To find the best-fitting model for the data, I first calculated the four evaluation dimensions, seen in section 4.6.2, for each model based on the temporally filtered data after 2018.

<i>DOMESTIC DECLARATION</i>						<i>INTERNATIONAL DECLARATION</i>					
	AM	HM	IM	IMf	IMd		AM	HM	IM	IMf	IMd
<b>FITNESS</b>	0.85	0.99	1.00	0.99	1.00	<b>FITNESS</b>	0.68	0.95	1.00	0.97	1.00
<b>PRECISION</b>	0.33	0.89	0.53	0.65	0.27	<b>PRECISION</b>	0.00	0.91	0.53	0.40	0.27
<b>GENERALIZATION</b>	0.88	0.88	0.89	0.88	0.86	<b>GENERALIZATION</b>	0.85	0.88	0.89	0.95	0.86
<b>SEMPPLICITY</b>	0.56	0.67	0.67	0.74	0.63	<b>SEMPPLICITY</b>	0.42	0.53	0.67	0.67	0.63
<b>TOKEN-BASED REPLAY</b>	0.85	0.99	1.00	0.99	1.00	<b>TOKEN-BASED REPLAY</b>	0.68	0.95	1.00	0.97	1.00

<i>PERMIT LOG</i>						<i>PREPAID TRAVEL COST</i>					
	AM	HM	IM	IMf	IMd		AM	HM	IM	IMf	IMd
FITNESS	0.45	0.95	1.00	0.98	1.00	FITNESS	0.63	0.96	1.00	0.94	1.00
PRECISION	0.00	0.76	0.08	0.17	0.07	PRECISION	0.00	0.68	0.12	0.135	0.12
GENERALIZATION	0.83	0.61	0.84	0.90	0.81	GENERALIZATION	0.89	0.65	0.90	0.90	0.86
SEMPPLICITY	0.59	0.49	0.48	0.61	0.46	SEMPPLICITY	0.12	0.54	0.53	0.63	0.50
TOKEN-BASED REPLAY	0.45	0.95	1.00	0.98	1.00	TOKEN-BASED REPLAY	0.63	0.96	1.00	0.94	1.00

<i>REQUEST PAYMENT</i>					
	AM	HM	IM	IMf	IMd
FITNESS	0.68	1.00	1.00	0.95	1.00
PRECISION	0.00	0.83	0.41	0.57	0.22
GENERALIZATION	0.76	0.73	0.81	0.87	0.81
SEMPPLICITY	0.43	0.60	0.60	0.67	0.53
TOKEN-BASED REPLAY	0.68	1.00	1.00	0.95	1.00

Looking at the tables we can see that the *Heuristic Miner* is apparently the best model but the results are not optimal. A possible solution to improve the result of this analysis may be to apply a filter on the variants. Using DISCO, I was able to obtain the leading variants that could enclose the most data. With the exception of Prepaid travel cost, using 5 variants I am able to enclose about 90% of the data

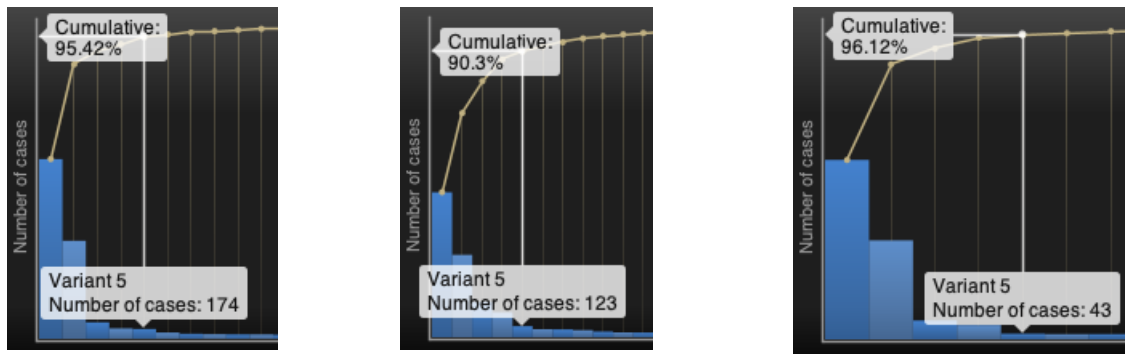
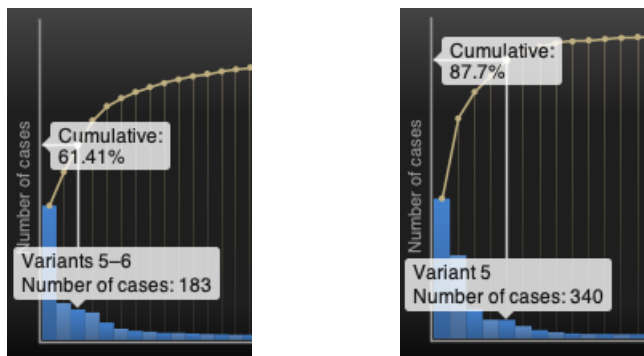


Figura 4.11: *Variant Domestic Declaration, International Declaration, Permit Log*



**Figura 4.12:** *Variant Prepaid Travel Cost, Request Payment*

By means of a PM4PY function, I filtered each log by taking the 5 variants with the highest number of traces, then performed a statistical analysis again to evaluate their performance.

<i>DOMESTIC DECLARATION</i>						<i>INTERNATIONAL DECLARATION</i>					
	AM	HM	IM	IMf	IMd		AM	HM	IM	IMf	IMd
FITNESS	0.85	1.00	1.00	1.00	0.88	FITNESS	0.84	1.00	1.00	1.00	0.89
PRECISION	1.00	0.99	0.86	0.75	0.24	PRECISION	1.00	0.81	0.52	0.65	0.30
GENERALIZATION	0.98	0.98	0.98	0.98	0.86	GENERALIZATION	0.97	0.97	0.98	0.98	0.81
SEMPPLICITY	0.82	0.85	0.81	0.79	0.67	SEMPPLICITY	0.74	0.80	0.77	0.82	0.70
TOKEN-BASED REPLAY	0.85	1.00	1.00	1.00	0.88	TOKEN-BASED REPLAY	0.84	1.00	1.00	1.00	0.89

<i>PERMIT LOG</i>						<i>PREPAID TRAVEL COST</i>					
	AM	HM	IM	IMf	IMd		AM	HM	IM	IMf	IMd
FITNESS	0.84	1.00	1.00	0.97	0.96	FITNESS	0.86	0.99	1.00	0.99	1.00
PRECISION	0.48	0.89	0.31	0.29	0.23	PRECISION	1.00	0.92	0.92	0.92	0.46
GENERALIZATION	0.96	0.96	0.96	0.93	0.84	GENERALIZATION	0.96	0.96	0.96	0.96	0.95
SEMPPLICITY	0.82	0.80	0.71	0.69	0.66	SEMPPLICITY	0.76	0.78	0.82	0.87	0.71
TOKEN-BASED REPLAY	0.84	1.00	1.00	0.97	0.96	TOKEN-BASED REPLAY	0.86	0.99	1.00	0.99	1.00

<i>REQUEST PAYMENT</i>					
	AM	HM	IM	IMf	IMd
FITNESS	0.85	1.00	1.00	0.99	0.89
PRECISION	1.00	0.99	0.86	0.75	0.23
GENERALIZATION	0.98	0.97	0.98	0.98	0.86
SEMPPLICITY	0.82	0.85	0.81	0.79	0.67
TOKEN-BASED REPLAY	0.85	1.00	1.00	0.99	0.89

Comparing the results, we can see that the Heuristic Miner proves to be the best. This solution is better than the previous one and we will generate a petri net on it

#### 4.6.4 Graphic Comparison Models

In this section I have reported graphical examples generated with PM4PY of the best model obtained with the unfiltered data and the best model obtained with the filtered data of the top 5 variants

##### 4.6.4.1 Domestic Declaration

##### Heuristic Miner Model

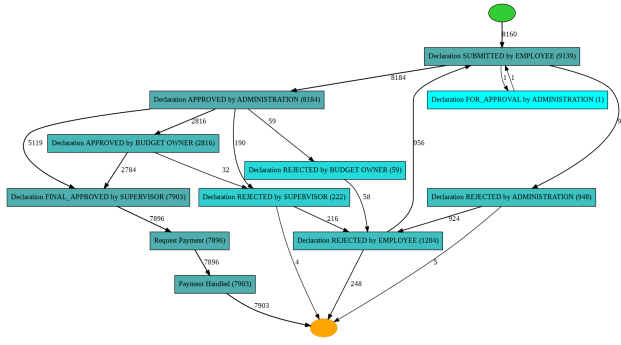


Figura 4.13: *HM without filter variants*

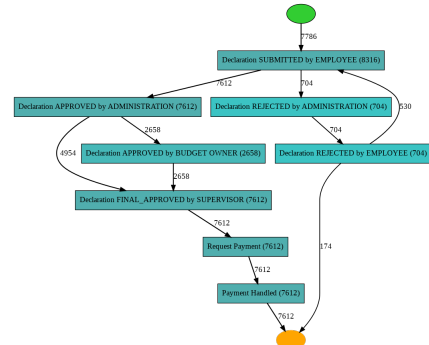


Figura 4.14: *HM with filter variants*

##### Petri Net

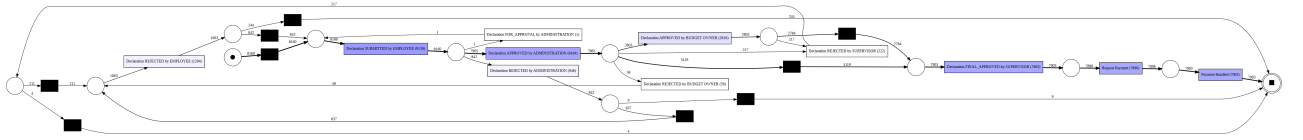


Figura 4.15: *Petri net without variants*

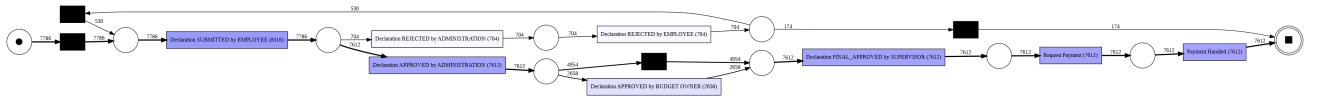


Figura 4.16: *Petri net with variants*

#### 4.6.4.2 International Declaration Heuristic Miner Model

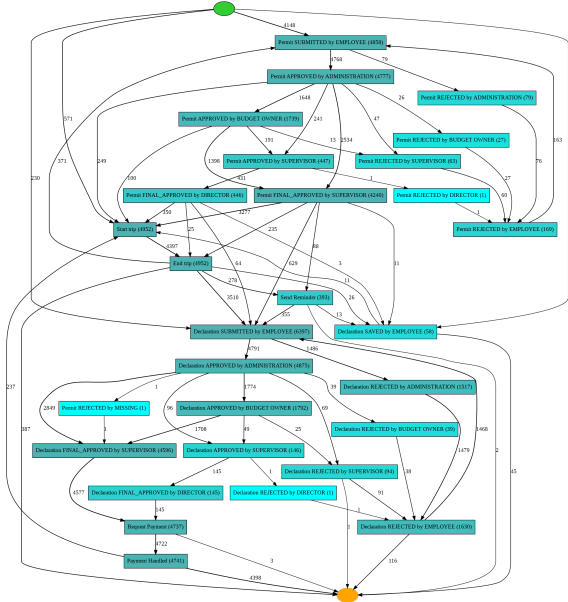


Figure 4.17: *HM without filter variants*

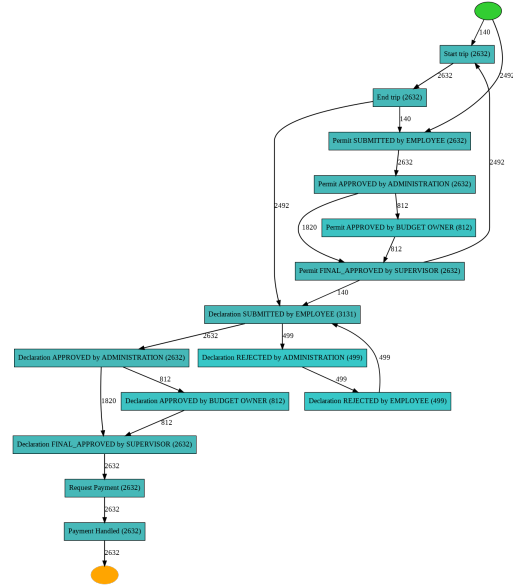


Figure 4.18: *HM with filter variants*

#### Petri Net

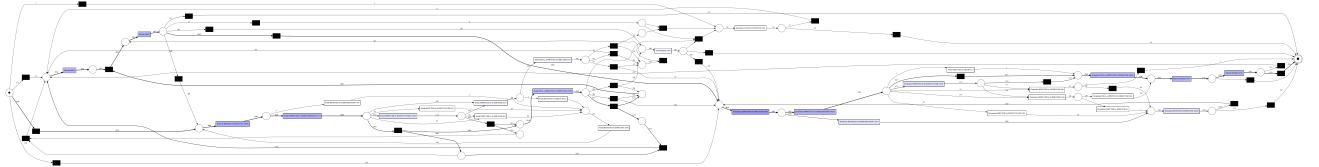


Figure 4.19: *Petri net without variants*

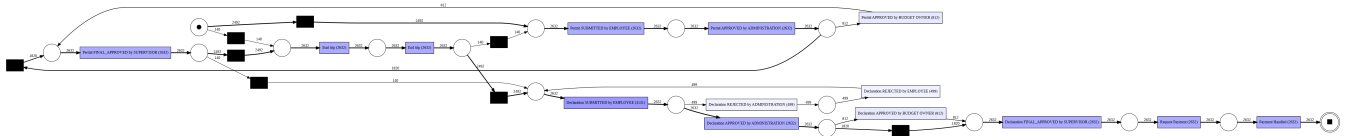


Figure 4.20: *Petri net with variants*

## 4.6.4.3 Request Payment

## Heuristic Miner Model

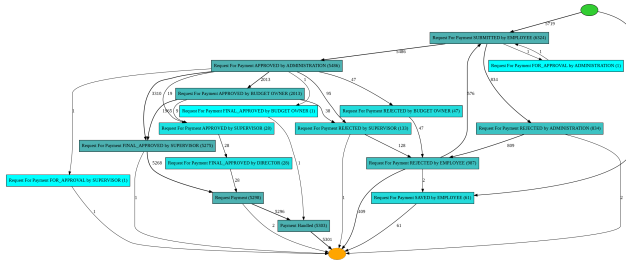


Figura 4.21: HM without filter variants

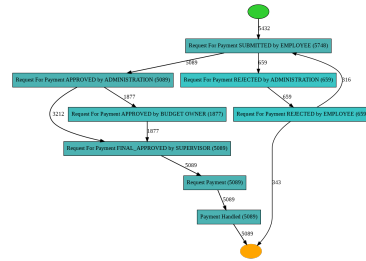


Figura 4.22: HM with filter variants

## Petri Net

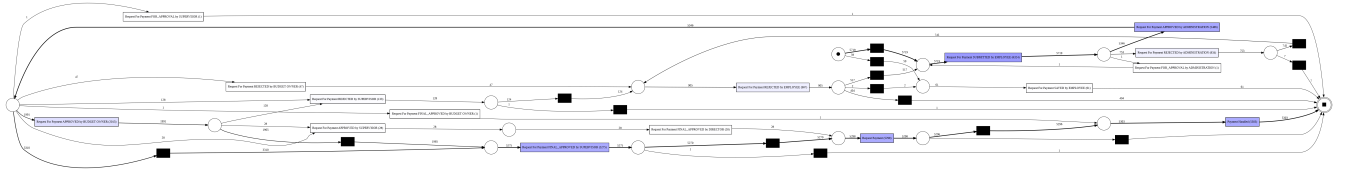


Figura 4.23: Petri net without variants

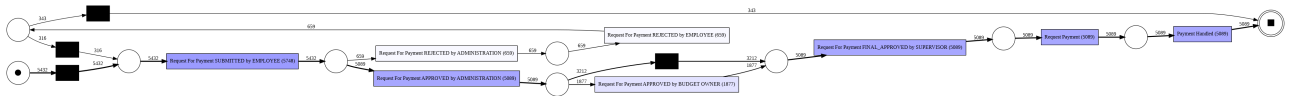


Figura 4.24: Petri net with variants



# Capitolo 5

## Conclusions

In this report, I focused on analyzing and answering the proposed questions of the BPI challenge 2020. In answering the questions, I have highlighted the results obtained and calculated the best model applicable to this context.

### 5.1 Problems found

The main problems I found in answering the challenge questions are related to the time from declaration of a trip to its approval and payment. The average time from trip request to completion of payment is about 10 days, in some borderline cases though it can be several months( results in figure 4.6). As shown in Figures 4.7 , 4.8 and 4.9, waiting times are particularly high in some cases of supervisor approval and in cases of permit rejection or declaration rejection.

Other issues found relate to the duplicate payments made, which are a substantial number as seen in table 5, and the high number of rejection which affects, as seen above, the time it takes to process a trip and its payment.

### 5.2 Possible Solution

One possible solution to the problems highlighted in the previous section is to use management software for travel requests and payment requests with different phases travel request or payment transits. In each false the requests should be ordered temporally so as to prioritize those that have been waiting the longest. In the payment section it would be necessary to highlight which requests have already received payment and prevent a second payment. Finally as seen from the templates presented in section 4.6.3 it would be desirable to simplify the process by removing some unnecessary statuses or those that adversely affect time