МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**Национальный исследовательский ядерный университет «МИФИ»**



Отчет по ЛР 2

по дисциплине “Нейронные сети и искусственный интеллект. Машинное глубокое обучение”.

Группа: М22-Ш04

Студенты: Смородинова С.С., Екимовская В.А.

# Москва 2023

1. Начало работы – установка и импорт необходимых библиотек:

**!**pip install pm4py

**import** pandas **as** pd

**from** pm4py.objects.conversion.log **import** converter **as** log\_converter

**from** pm4py.objects.log.importer.xes **import** importer **as** xes\_importer

from matplotlib import pyplot as plt

import seaborn as sns

1. Считываем данные

# Считывание файла в формате .xes

log = xes\_importer.apply('BPI\_Challenge\_2013\_incidents.xes')

1. Изучение набора данных, с которым предстоит работа

названия колонок

типы данных

пропущенные значения

и другое

1. После изучения преобразуем типы данных в нужные

log\_resalt['time:timestamp'] = pd.to\_datetime(log\_resalt['time:timestamp'])

В процессе изучения набора данных приходим к выводу, что исходном файле есть данные различных событий (инцидентов) в различных странах, а также имя сотрудника, время исполнения и жизненный цикл каждого события.

Размер файла: 65533 строк и 18 столбцов

Также стало понятно, что больше всего времени занимает процесс - “**В ходе выполнения**”. Совсем маленький процент отмененных заказов.

Наиболее встречаемая транзакция - **“В ходе выполнения”**, также часто встречаемой является **“ожидание назначения”**. Реже всего - **“отменен”**.

## Построение модели по журналу событий

**Alpha Miner**

# Импортируем алгоритм из библиотеки PM4PY

from pm4py.algo.discovery.alpha import algorithm as alpha\_miner

# Создаем алгоритм

net, initial\_marking, final\_marking = alpha\_miner.apply(event\_log)

from pm4py.visualization.petri\_net import visualizer as pn\_visualizer

parameters = {pn\_visualizer.Variants.FREQUENCY.value.Parameters.FORMAT: "png"}

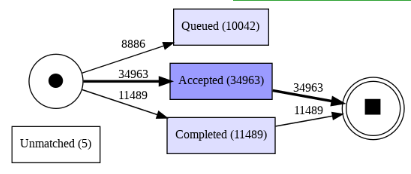
gviz = pn\_visualizer.apply(net, im, fm,

parameters=parameters,

variant=pn\_visualizer.Variants.FREQUENCY,

log=event\_log)

pn\_visualizer.view(gviz)



Простота: 1,00

Точность: 0,6

Обобщение: 0,6

**Inductive miner**

from pm4py.algo.discovery.inductive import algorithm as inductive\_miner

import pm4py

net, im, fm = pm4py.discover\_petri\_net\_inductive(event\_log)

from pm4py.visualization.petri\_net import visualizer as pn\_visualizer

parameters = {pn\_visualizer.Variants.FREQUENCY.value.Parameters.FORMAT: "png"}

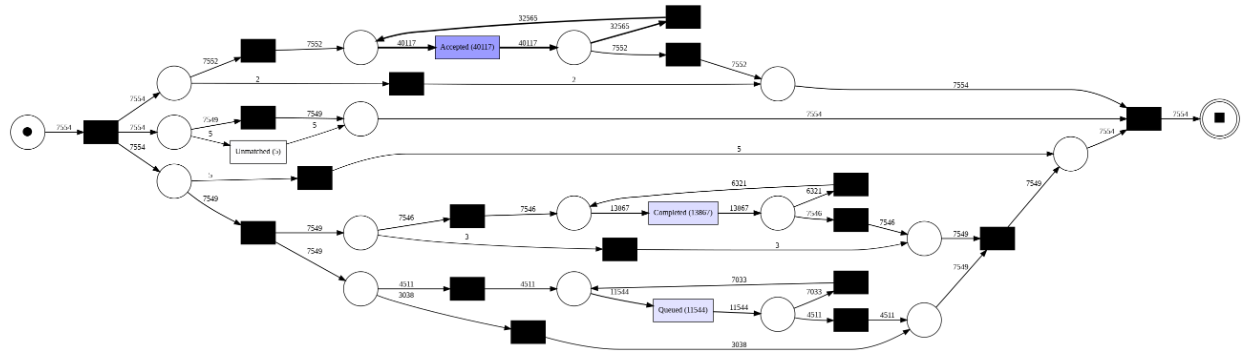
gviz = pn\_visualizer.apply(net, im, fm,

parameters=parameters,

variant=pn\_visualizer.Variants.FREQUENCY,

log=event\_log)

pn\_visualizer.view(gviz)

****

Простота: 0,66

Точность: 0,63

Обобщение: 0,63

**Heuristics miner**

from pm4py.algo.discovery.heuristics import algorithm as heuristics\_miner

net, im, fm = heuristics\_miner.apply(event\_log)

from pm4py.visualization.petri\_net import visualizer as pn\_visualizer

parameters = {pn\_visualizer.Variants.FREQUENCY.value.Parameters.FORMAT: "png"}

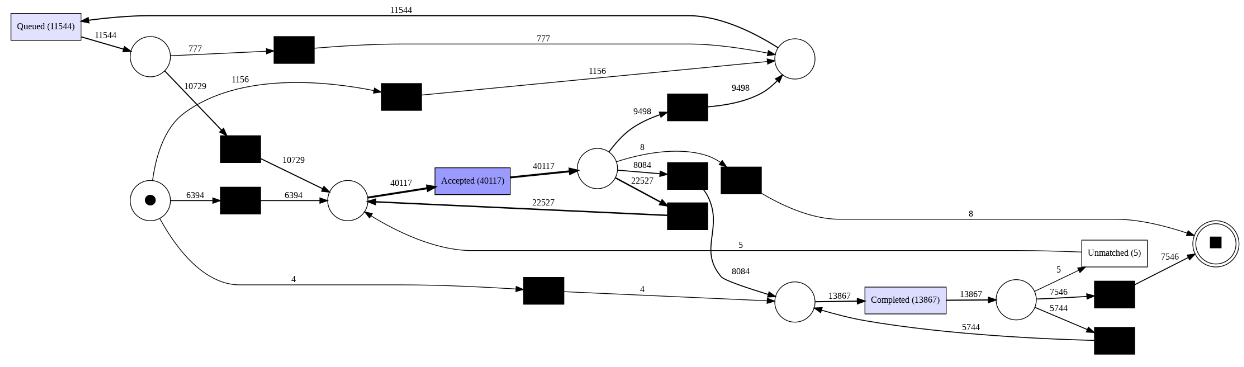
gviz = pn\_visualizer.apply(net, im, fm,

parameters=parameters,

variant=pn\_visualizer.Variants.FREQUENCY,

log=event\_log)

pn\_visualizer.view(gviz)



Простота: 0,62

Точность: 0,89

Обобщение: 0,89

**DFG**

from pm4py.algo.discovery.dfg import algorithm as dfg\_discovery

dfg = dfg\_discovery.apply(event\_log)

from pm4py.objects.conversion.dfg import converter as dfg\_mining

net, im, fm = dfg\_mining.apply(dfg)

from pm4py.visualization.petri\_net import visualizer as pn\_visualizer

parameters = {pn\_visualizer.Variants.FREQUENCY.value.Parameters.FORMAT: "png"}

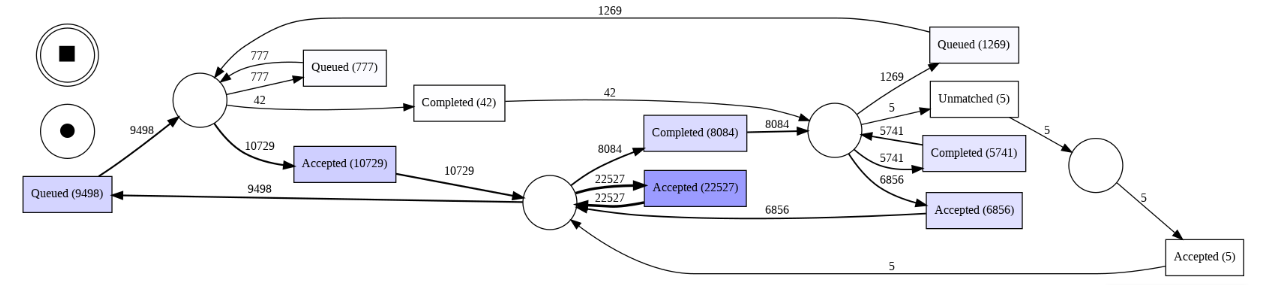
gviz = pn\_visualizer.apply(net, im, fm,

parameters=parameters,

variant=pn\_visualizer.Variants.FREQUENCY,

log=event\_log)

pn\_visualizer.view(gviz)



Простота: 0,63

Точность: 1,00

Обобщение: 1,00

1. Обобщаем полученные данные

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Простота** | **Точность** | **Обобщение** |
| **Inductive** | 0,66 | 0,63 | 0,63 |
| **Heuristic** | 0,62 | 0,89 | 0,89 |
| **Alpha** | 1,00 | 0,60 | 0,60 |
| **DFG** | 0,63 | 1,00 | 1,00 |

Лучше всего на этих данных показал себя DFG граф