



UNIVERSITÀ DELLA CALABRIA - ITALY

---

## Report of Process Mining Project

---

*Réalisé par :*

ELKHOUMSSI NOUR EL-HOUDA

CHERRAT MARWA

*Encadré par :*

Fionda VALERIA

Année Scolaire 2023/2024

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Realisation</b>	<b>2</b>
2.1	Preparing and importing data . . . . .	2
2.2	Understanding data and construction of ECKG and EKG. . . . .	3
2.3	Implementation of the algorithm . . . . .	5
<b>3</b>	<b>Results and Analysis</b>	<b>7</b>
<b>4</b>	<b>Conclusion</b>	<b>8</b>

# Chapitre 1

## Introduction

This project focuses on implementing and testing an algorithm designed to identify frequent directly-follows (df) paths in an Event Knowledge Graph (EKG). The initial phase involves preparing and importing data into a Neo4j graph database. Subsequently, we embark on constructing the Event Class Knowledge Graph (ECKG) and visually examining the Event Knowledge Graph (EKG) to comprehend the data structure, relations, and attributes.

Following this exploration, the focus shifts towards implementing the algorithm as per the project requirements. Throughout this stage, continuous testing is conducted, iterating and refining the algorithm based on the given specifications. This report will present what was mentioned as following :

1. Preparing and importing data .
2. Understanding data and construction of ECKG and EKG.
3. Implementation of the algorithm
4. Analysing the results

# Chapitre 2

## Realisation

### 2.1 Preparing and importing data

We utilized a dataset from <https://zenodo.org/records/470811>, provided in CSV format, to construct and work with our data. This stage proved interesting but challenging due to issues encountered during import and file preparation. Despite facing errors in the code, we persevered. The general process involved preparing the data using a code, and then using the generated file to successfully import the data into Neo4j.

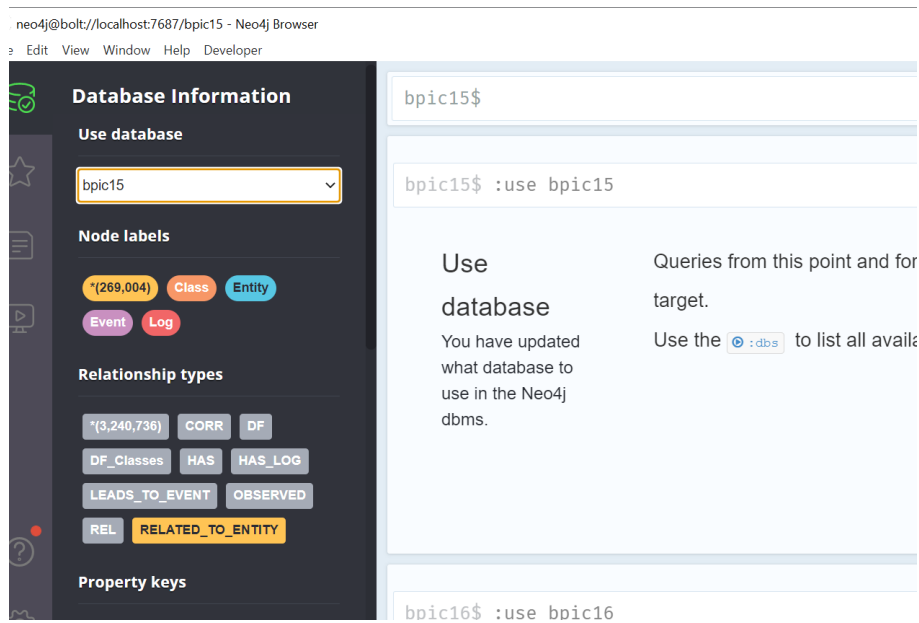


FIGURE 2.1 – data bpic15 in Neo4j

After this step , we moved to visualise our data in order to understand it more and then we moved to the creation of the ECKG and EKG.

## 2.2 Understanding data and construction of ECKG and EKG.

we started with Visualising Our data. So the circle in blue are the entities ,the purple circles are the events and the orange is the classs .

There is a relation between classes and events that is "Observed", and events

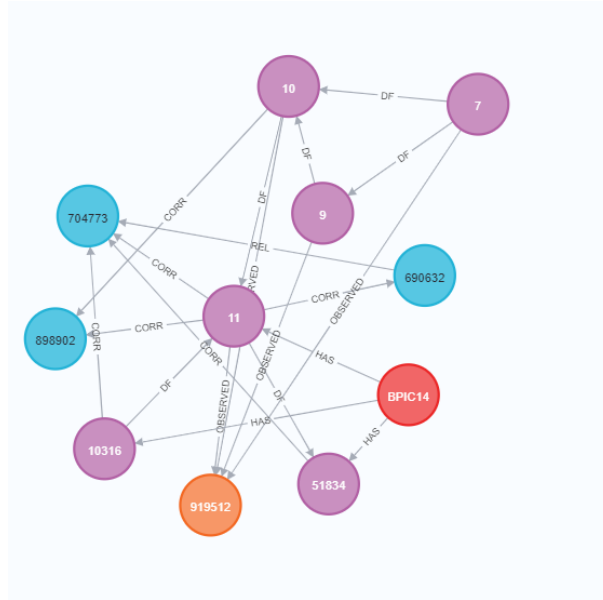


FIGURE 2.2 – Visualisation of a part of our data

are related with the relation "DF" relation.

The figure bellow will show different attributes of classes and events ,that was principally used in the implementation of the algorithm.

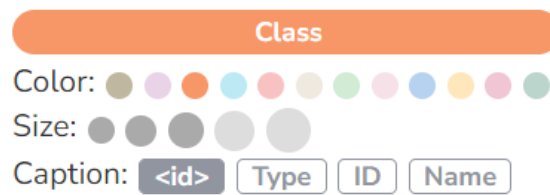


FIGURE 2.3 – Attributes of Classes

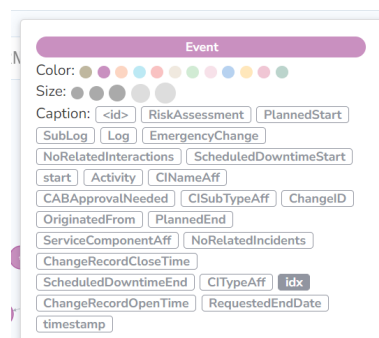


FIGURE 2.4 – Attributes of Event

we moved to create the ECKG. the code bellow create a relation r1 between c1 and C2 if there is a relation between an event 1 observed by c1 and an event 2 observed by c2 ,with the label "DfCount".

```
MATCH (c1:Class)-[:OBSERVED]-(e1:Event)-[dfRel:DF]->(e2:Event)-[:OBSERVED]-(c2:Class)
WHERE c1 <> c2
WITH c1, c2, e1, e2, COUNT(dfRel) AS dfCount, COLLECT(dfRel) AS dfs
LIMIT 10 // Set the limit as needed

// Create DF_Classes relationship for each pair of events
FOREACH (ignore IN CASE WHEN dfCount > 0 THEN [1] ELSE [] END |
  CREATE (c1)-[:DF_Classes] (
    DF_count: dfCount,
    start_event_id: id(e1),
    end_event_id: id(e2),
    df_ids: [df IN dfs | id(df)]
  )->(c2)
);|
```

FIGURE 2.5 – Creation of ECKG

We moved to visualise the graphs, to see if we got what we desired.(We used limits to reduce the number of elements displayed in order to see clearly the results)

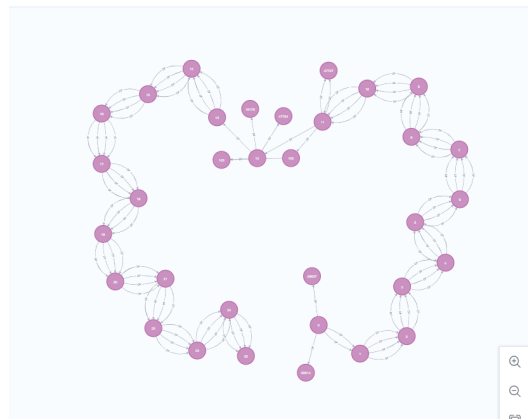


FIGURE 2.6 – Creation of EKG

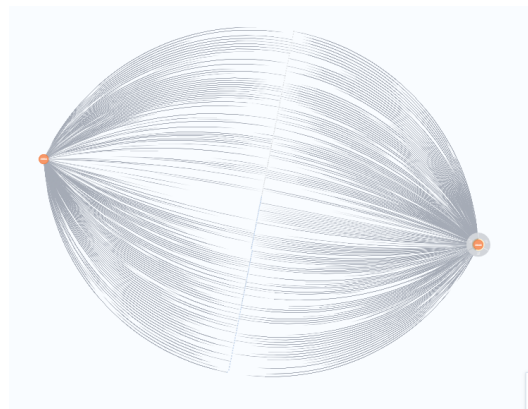


FIGURE 2.7 – Creation of ECKG

After creating the ECKG and understanding the data we moved to the implementation of the algorithm based on the description.

## 2.3 Implementation of the algorithm

We started by understanding the description of the algorithm ,then we moved to the implementation.

1. **Algorithm1** : Find Maximal Significant DF Paths.

```

16
17 # Algorithm 1: Find Maximal Significant DF Paths
18 def find_maximal_significant_df_paths(ECKG, frequency_threshold, significance_threshold):
19
20     significant_paths = [] # Start with an empty list of significant paths.
21     for i in range(len(ECKG)): #For each edge(relation) in ECKG
22
23         # print("HELLO ")
24         path = [ECKG[i]] # Initialize a path with that edge.
25         expand_and_check_maximal_path(path, ECKG, frequency_threshold, significance_threshold, significant_paths)
26
27     return significant_paths
28
29 """significant_paths"""

```

FIGURE 2.8 – Algorithm 1

## 2. ALgorithm2 : Expand And Check Maximal Path

[illegible]

FIGURE 2.9 – Algorithm 2

### 3. ALgorithm3 : bsolute Fequency

```

50
51 #Calculate Absolute frequency of a path
52
53 def absolute_frequency(path):
54
55     frequencyRelation = float('inf')
56     frequencyPath = 1
57     for i in range(len(path)):# Iterate over each directly-follows relation in the path
58
59         frequency_of_relation_In_ECKG = path[i][4] # Get the frequency of the Current relation in ECKG
60         frequencyRelation = min(frequencyRelation, frequency_of_relation_In_ECKG)
61         #frequencyPath = frequencyPath * frequencyRelation
62
63     return frequencyPath # Return the calculated absolute frequency of the path in ECKG
64 #####

```

FIGURE 2.10 – Algorithm 3

#### 4. **ALgorithm4** : Calculate Relative Frequency of a Path.

```
# Algorithm 4: Calculate Relative Frequency of a Path
def relative_frequency(path, starting_class):

    total_count = 1
    pathCount = 0
    for i in range(len(path)):
        TotalOfEvents = get_Events_In_StartingClass(path[i][1])
        #for j in range(len(TotalOfEvents)):
            total_count = total_count * len(TotalOfEvents)
        TotalOfEvents = get_Events_In_StartingClass(path[len(path)-1][2])
        #for k in range(len(TotalOfEvents)):
            total_count = total_count * len(TotalOfEvents)

    allEvents_In_StartingClass = get_Events_In_StartingClass(starting_class)
    for i in range(len(allEvents_In_StartingClass)):
        pathCount += Count_Path_In_EKG(allEvents_In_StartingClass[i])

    if total_count > 0 :
        relative_frequency = pathCount / total_count

    return relative_frequency
```

FIGURE 2.11 – Algorithm 4

We created also some other functions such as :

1. Function get-Events-In-StartingClass to get all the events observed by a specific class.
2. Function Count-Path-In-EKG to Get total count of Paths that start with a specific Event.
3. Function get-eckg-from-neo4j() to Get Event Class Knowledge Graph (ECKG) from Neo4j.

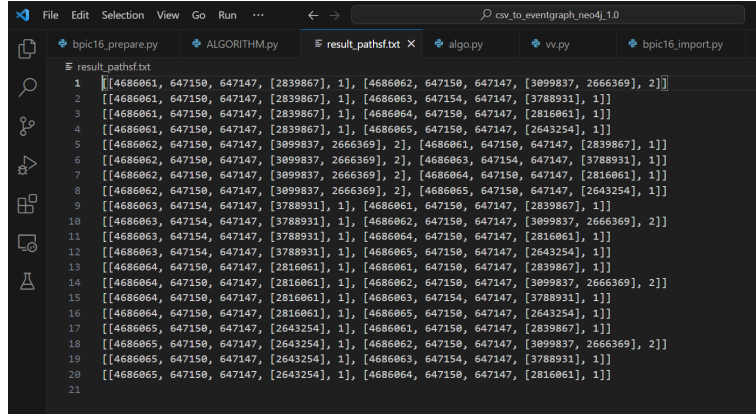


## Chapitre 3

# Results and Analysis

Because of the big dimension of our data we used the limit to reduce the size in order to see clearly the result .

So bellow this the results that we got after running the code with frequency-threshold=1, significance-threshold=0.3. So we can see that we got 20 path with



```
1 [[4686061, 647150, 647147, [2839867], 1], [4686062, 647150, 647147, [3099837, 2666369], 2]]
2 [[4686061, 647150, 647147, [2839867], 1], [4686063, 647154, 647147, [3788931], 1]]
3 [[4686061, 647150, 647147, [2839867], 1], [4686064, 647150, 647147, [2816061], 1]]
4 [[4686061, 647150, 647147, [2839867], 1], [4686065, 647150, 647147, [2643254], 1]]
5 [[4686062, 647150, 647147, [3099837, 2666369], 2], [4686061, 647150, 647147, [2839867], 1]]
6 [[4686062, 647150, 647147, [3099837, 2666369], 2], [4686063, 647154, 647147, [3788931], 1]]
7 [[4686062, 647150, 647147, [3099837, 2666369], 2], [4686064, 647150, 647147, [2816061], 1]]
8 [[4686062, 647150, 647147, [3099837, 2666369], 2], [4686065, 647150, 647147, [2643254], 1]]
9 [[4686063, 647154, 647147, [3788931], 1], [4686061, 647150, 647147, [2839867], 1]]
10 [[4686063, 647154, 647147, [3788931], 1], [4686062, 647150, 647147, [3099837, 2666369], 2]]
11 [[4686063, 647154, 647147, [3788931], 1], [4686064, 647150, 647147, [2816061], 1]]
12 [[4686063, 647154, 647147, [3788931], 1], [4686065, 647150, 647147, [2643254], 1]]
13 [[4686064, 647150, 647147, [2816061], 1], [4686061, 647150, 647147, [2839867], 1]]
14 [[4686064, 647150, 647147, [2816061], 1], [4686062, 647150, 647147, [3099837, 2666369], 2]]
15 [[4686064, 647150, 647147, [2816061], 1], [4686063, 647154, 647147, [3788931], 1]]
16 [[4686064, 647150, 647147, [2816061], 1], [4686065, 647150, 647147, [2643254], 1]]
17 [[4686065, 647150, 647147, [2643254], 1], [4686061, 647150, 647147, [2839867], 1]]
18 [[4686065, 647150, 647147, [2643254], 1], [4686062, 647150, 647147, [3099837, 2666369], 2]]
19 [[4686065, 647150, 647147, [2643254], 1], [4686063, 647154, 647147, [3788931], 1]]
20 [[4686065, 647150, 647147, [2643254], 1], [4686064, 647150, 647147, [2816061], 1]]
21
```

FIGURE 3.1 – Results

two relation, for each relation the first value represent the id of df-relation, then the starting class and ending class, for the third element is ids of df between events of the class , and the last one is the df count.

So those paths are the frequent directly-follows (df) paths in an Event Knowledge Graph (EKG) that was found based on the ECKG and with respecting the values of the treasholds. Nevertheless, without imposing a limit, the code takes an extended amount of time to execute. This underscores the necessity for optimizing the code promptly to reduce the execution time.

## Chapitre 4

# Conclusion

In summary, this project aimed to implement and test an algorithm for identifying frequent directly-follows (df) paths in an Event Knowledge Graph (EKG). The process involved meticulous data preparation, importation into Neo4j, and the creation of Event Class Knowledge Graphs (ECKG) and EKG. The implemented algorithm, comprising four main components, was thoroughly tested on the dataset, yielding 20 frequent df paths. The report concludes with insights into the algorithm's performance and its implications for understanding event-based systems.