UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Refactorization and extension of a data analysis tool based on graph decomposition

Master in Innovation and Research in Informatics

Specialization: Data Science

Thesis supervisor: José Luis Balcázar
Author: Nejada Karriqi

Barcelona, October 26th, 2021

Acknowledgments

Firstly, I want to express my deepest appreciation and gratitude to the person who was always next to me during the development of my thesis, not only as a mentor but also as a friend, professor Jośe Luis Balćazar. I can not express by words how much I appreciate him and how thrilled i am to having worked with him. He are an amazing human being.

Secondly, I want to thank Ely for agreeing on allowing me to use her research, for handing me all the documents and software versions needed to progress and for being available when I needed help or additional information.

Thirdly, I want to thank everyone who helped me become the person I am today. Everyone who shared his/her knowledge with me or who gave me a hand when I needed. Thank you all my friends and UPC academic staff for helping me successfully overcome all the difficulties I faced during my studies.

And finally, I want to thank my family for always supporting me, trusting me and always being by my side. Moreover, I want to thank someone special because without his support nothing I have today would had been possible.

Thanks to all of you, I am a better person and my future is brighter.

Abstract

Graph decomposition problems are one of the most widely studied topics of graph theory with considerable applications in data mining. When using graph decomposition you are decomposing the graph into sets of edge-disjoint subgraphs in which each edge belongs to exactly one of them. In our project, we are going to use Gaifman graphs and the terms modular decomposition of graphs and clan decomposition of 2-structures, which means we are decomposing the vertices into subsets. Those subsets are respectively called modules and clans. The vertices of the Gaifman graphs will represent the problem variables and the edges the connection of two variables if they happen to co-occur together. As base of our project will serve a PhD successfully obtained by Maria Ely Piceno. The focus of this PhD are the Gaifman graphs and its decomposition variants. Apart from the theoretical study, her project includes a software to practically demonstrate the results.

Given the fact that the PhD revealed interesting results and influenced by the difficulty faced when trying to use the PhD's software, we decided to introduce a novel way of generating the decomposition graphs through PyGame. Pygame is nothing more than a cross-platform set of python modules designed for game development. Since nowadays everyone talks about "gamification", we have decided to use one of the most recent modules of Pygame, the Pygame GUI. This module strives to create a contemporary interface with elements such as buttons, file dialog, sliders, scroll bars and drop down menus. Through this project, we are going to try how easy and feasible is to implement game elements and make the studying process more easy and attractive.

Contents

1. Introduction

Graphs are frequently utilized as one of the most common methods for visualizing the relationships between data. The goal of a graph is to represent large amounts of data in a little amount of space. Of course, in order to call the visualization process a success, the graph must be understandable and clear [9], which is why we must be cautious while selecting the graph type. Graphs are used in data mining to analyze real-world data features, forecast how those properties would effect a given application, and construct models that can generate realistic graphs that match the patterns found in real-world graphs.

A recent PhD thesis at UPC has explored the data analysis through graph decomposition [7]. More specifically, they have used the Gaifman graphs decomposition and explored their variants. For this reason, they also developed a software but the related software is not appropriate for external users as it requires prior knowledge on data mining and internal implementation.

This is the reason why we considered it necessary to create an interactive environment for data analysis between Gaifman graph decomposition and Pygame. Our master project is to achieve the main goal by creating an interface that will make the tool easy to use even by non experts. The users will need to define the edges parameter to get an intuitive visual support from our system. Therefore no technical knowledge is needed to use it because all the decomposition algorithms will run on the background.

As Ely has used GraphViz as visualization tool, here are some of the best alternatives of 2021 that can be used instead of it:

- Diagrams.net: free and open source
- yEd Graph editor: free
- LibreOffice-Draw: free and open source
- PlantUML: free and open source, and
- Microsoft Office Visio: paid

While many alternatives are available, we have to decide which one fits best to our needs. In our case, we decided to go with *PyGraphViz* even though it was a bit more complex that GraphViz because pip needs to compile C bindings and find all libraries.

The main aim of this project is to improve the software designed by Ely by introducing the **"gamification"** concept due to the fact that the learning process has proven to be much effective and easy when mixed with gaming. It might be also possible to apply this idea in many areas of data mining. Of course, this would require to create some interfacing between game creation tools, such as PyGame, and data analysis tools.

1.1 Context of the study

This project is based on the research work done by Maria Ely Piceno during her PhD studies. Her work consists in the search for relationships between items on data to identify co-occurrence patterns through the construction and decomposition of graphs, the so-called Gaifman graph and its variations. Throughout this thesis, they argue about the usefulness of Gaifman graphs on first-order relational structures. As exploratory data analysis tool, they provide the theory that supports our work and explain the algorithmics implemented [7].

The thesis was submitted on *July, 2020* to the Computer Science Department of the *Polytechnic University of Catalunya*. It was funded by **European Research Council (ERC)**, Grant/Award Number: ERC-2014-CoG 648276; **The Ministry of Economy, Industry and Competitiveness**, Grant/Award Number: TIN2017-89244-R; **AGAUR** (Generalitat de Catalunya), Grant/Award Number: 2017SGR-856. The thesis supervisor was **José Luis Balcázar**.

1.2 Objectives

The main objectives to be achieved with this thesis development are:

- Making the current software easy to be used by introducing the gaming concept,
- Creating an interface between Pygame and Gaifman graph decomposition,
- Creating graph animations with Pygame and,
- Exploring how difficult this may become.

1.3 Overview of the thesis

This master thesis is organized on five chapters. On the chapter 1, we introduce the project and give some alternatives to be used instead of GraphViz/PyGraphViz for data visualization. On chapter 2, we explain some basic concept related to graphs and graph decomposition and we give a brief explanation of the PhD project which will serve as the base for the development of this master thesis. Chapter 3 talks about "gamification" concept, Pygame, their advantages and the things that can be done using Pygame. Chapter 4 gives the details of our approach and chapter 5 includes some of the main difficulties of this project while revealing the results.

2. Cutting edge

This chapter will include all the details about the project where have based our work and research. First of all, we are going to explain some key concepts which will be used during the development of our projects and are also used on the project we are using as the base of our research. Then, we will explain in detail the datasets used, the key concepts and the logic of the work done by Marie Ely Piceno[6]. Finally, we will show and explain the results achieved by her in order to see the significance and importance of the study.

2.1 Graphs and data mining techniques

In general, data mining techniques are based on finding a common path or identifying data items that appear together very frequently. When dealing with huge datasets, the result of the mining techniques can be a huge amount of textual lists which are very difficult to process and understand. This is why graphs are very handy to help us understand and discover the most significant and useful information. Graphs are nothing more than a formal way to represent a collection of interconnected items. The formal definition of a graph in mathematics is: Graphs are ordered pair of a set of vertices and of a set of edges. Mathematically written as:

$$G = (V, E)$$

where V is a set of nodes also known as vertices and E is a set of edges. Graphs can be directed and undirected, depending if the definition of the origin and destination node matters. In our case, we will only consider undirected graphs.

2.1.1 Problems solved by graphs

The areas where graphs make a good solution are many. Here are some of the most common problems that are better represented or solved by graphs:

- Maps and pathing: to find the longest or the shortest path graphs are used.
- Data structures: using pointers to link data signifies that you are using graphs.
- Constraint satisfaction: sometimes the best way to find the solution which satisfies some constraints is to use graphs.
- Network problems: by using graphs it is easier to find out which is the best way to connect two pieces, if there is a way to build a connection or how to reach from one place to another.
- Compiler optimization problems: graphs are used for these kind of problems too.

- Web and state machines: both of them are represented by graphs.

As we have seen many applications or problems solved by graphs, let's focus on the way our problem is solved: using Gaifman graphs.

2.2 Gaifman graphs

In order to understand properly the result of this research, it is important to give some general information about Gaifman graphs and the key notions used. Gaifman proposed a so-called theory where some technical developments were based on the graphs. [4]. In modern terms, different from the original ones but equivalent to them, these graphs represent co-occurrences among items in dataset transactions. Given a first-order relational structure where the values appearing in the tuples of the relations R_i come from a fixed universe U, its corresponding Gaifman graph has the elements of U as vertices, and the edges (x, y) for $x \neq y$ are determined exactly when x and y appear together in some tuple $t \in \mathrm{R}_i$ for some R_i.
To make it more clear lets see an example:
Consider the relational dataset formed on the universe $U = \{\mathrm{a, b, c, d, e}\}$ by the tuples:

$$t1 : abc$$

$$t2 : ae$$

$$t3 : acd$$

$$t4 : de$$

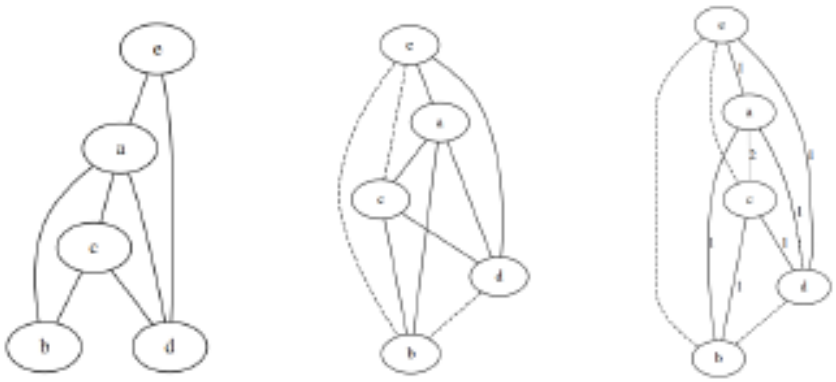

Figure 2.1: Gaifman graph models

The left graph is the Gaifman graph that represents the relational dataset information. The nodes are the items of the universe U and the edges between them represent the co-occurrence of the items. For example: item e is found together with the items a *and* d but not with b *and* c, a co-occurs with items b, c, d *and* e, item b co-occurs with a *and* c but not with d *and* e, and d co-occurs with a, c, e but not with b. The graph in the middle is the representation of the natural completion Gaifman graph. The solid lines represent two items that appear together frequently, while the broken lines represent the items that never co-occur together. The graph on the right is a representation of the labeled Gaifman graph. In the labeled Gaifman graphs, the labels of the edges show how many times these items appear together.

2.2.1 Gaifman graph generalization

The standard Graifman graphs use the modular decomposition. Modular decomposition is the process of decomposing the vertices into sets named modules which can be subgraphs of the original graph. It is very important to state that in our case only the strong modules are considered because the aim is to obtain tree-like decomposition and to avoid overlapping.
Ely's study is based on Gaifman theory. The novelty of their approach relies on the application of Gaifman graphs and their variant decompositions to provide a visualization of the behavior of the data which can be very useful when used in statistical approaches. The discretization methods are many but the ones used in this study are: label discretization, linear and exponential discretization and the variant based on the shortest path. After discretization process, the usage of graph decomposition makes possible the visual representation of the data co-occurrence as a fundamental step to continue then with other statistical analysis procedures. The resulting graph is hierarchical and increases the interpretability of the output in terms of data analysis.
The idea of adjusting the analysis is connected to putting some constraints when generating the graph. The first thing to do is to define a threshold k. The resulting Gaifman graph will be a graph where each labeled edge will be classified based on its co-occurrence. In cases when the label indicator will be greater than the k, the edge will belong to the equivalence class one; if it will be smaller or equal to k, the edge will belong to the equivalence class two. Therefore, we will have two equivalence classes. As mentioned above, there are also more variations of Gaifman graph generalizations such as label discretization, linear and exponential discretization and another one based on the shortest paths. All of them will be illustrated with an example below.
Since the notion of modular decomposition is not sufficient to handle properly the generalizations of Gaifman graphs, they introduce the notion of 2-structures and their clans. Here are some very important definitions:
Definition . A 2-structure is complete graph on some universe with an equivalence relation among its edges. For a 2-structure, we say that a subset C is a clan, if all the members of C are indistinguishable among them by nonmembers.
Definition: Singletons are called all the clans which contain only one value below them [2].
Definition . A clan C is a strong clan of a graph if it does not overlap with any other clan.
Notes: Strong modules are represented by black dots and singletons are the modules which only have one value below them. Nontrivial modules are represented by a box linked to the strong module (black dot). When two strong modules are connected to each other, we understand that each value below strong module one co-occurs with each value of the strong module two. The opposite means that the strong modules are not connected together [8]. Concrete examples will be included on the following chapters.

2.2.2 Datasets used

In her study, she has used two main datasets: the simplified version of the famous Titanic dataset which contains the information about 2201 people on board. The records include four attributes: traveling class, sex, age and survival.
The second dataset contains hospitalization records. The transactions include diagnostics, medical treatments and related information. All the hospitalizations between 2015-2016 are part of this dataset which was provided by the Hospital de la Creu i Sant Pau under the collaboration agreement with UPC. Some additional information, part of the dataset, has been discarded from the analysis. The data used for analysis includes 79 534 rows.

Important note: All the information that can identify or deanonymize the patients has been removed from the dataset.

2.2.3 Results and interpretation

Lets see the example they use to make it more understandable. In this case the dataset contains data about the patient hospitalizations and each transaction is a set of diagnostics, medical treatments and other related information. The first thing they do is to construct the 2-structures using one of the Gaifman graphs variants and then they apply clan decomposition method. The constructed 2-structures has as vertices all the attribute values and a setup of edges based on the Gaifman graph chosen variant. To limit the amount vertices displayed, a threshold is determined. Thus, only the vertices that appear more frequently then the threshold will be drawn.

2.2.3.1 Exponential Gaifman graph

This picture is the output obtained when exponential Gaifman graphs are used. The graph on the left is the graph where only the diagnostics attribute with frequency threshold of 100 are considered. The co-occurrence threshold is set to eight, which means the transactions must co-occur at least 8 times to be considered, otherwise they will be dropped. To understand the graph lets specify what the numbers mean:

650 *Normal delivery*

632 *Missed abortion*

305.1 *Tobacco use disorder*

401.9 *Unspecified essential hypertension*

272.4 *Other and unspecified hyperlipidemia*

As it can be seen, *Normal delivery* is connected with a dashed line with the rest of diagnostics. This means that it doesn't frequently co-occur with them. We have a big clan created by the rest of the diagnostics (the big box) and below it a smaller clan created by *Other and unspecified hyperlipidemia* and *Unspecified essential hypertension*. This clan was named as *Hypertension clan*. Both items inside it, are found in co-occurrence with the other diagnostics the same number of times, while the clan items are very highly connected together. The dashed line between *Missed abortion* and *Hypertension clan* signifies that they co-occur together less than the threshold.

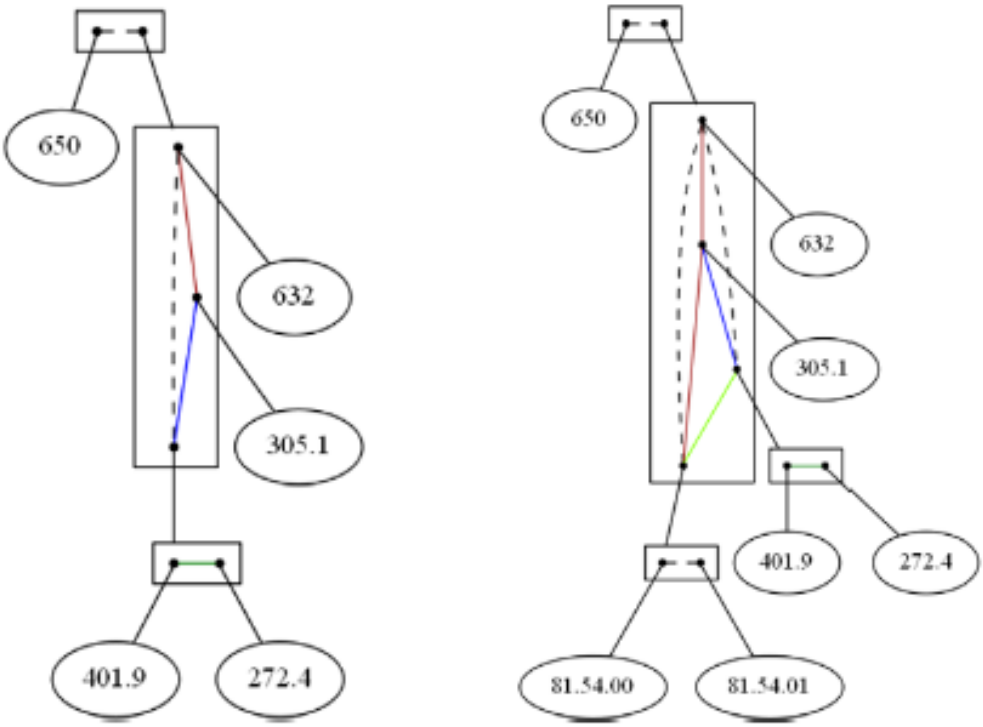

Figure 2.2: Gaifman graph exponential variant

The picture on the right, apart from diagnostics, takes into consideration the treatments. The co-occurrence threshold is maintained at eight and only the pairs *(diagnostics, treatment)* found at least 100 times are considered. Similarly to the left graph, *650* remains apart. There are two small clans now, the *hypertension clan* and *Total knee replacement clan*.

81.54.00 $Totalkneereplacement(left)$

81.54.01 $Totalkneereplacement(right)$

. The times, when a patient has had to replace both knees at the same time are few or zero. That's why the items are connected by a dashed line. Other very unlikely treatments to happen are between *Knee replacement clan* and *Normal delivery* and the one we saw on the previous graph between *Normal delivery* and *Hypertension clan*

2.2.3.2 Linear Gaifman graphs

In this example they apply the Linear variant of the Gaifman graphs where the data is split in intervals of size 1000. The diagnostics considered must appear more than 80 times. Here is the list:

650 $Normal\ delivery$

250.00 $Diabetes\ mellitus$

401.9 $Unspecified\ essential\ hypertension$

305.1 $Tobacco\ use\ disorder$

278.00 *Obesity*

272.4 *Other and unspecified hyperlipidemia*

634.92 *Spontaneous abortion without complications*

632 *Missed abortion*

Same as before, on the left graph, we can spot that *Normal delivery* doesn't co-occur frequently with any other diagnostics. We, can also see that a small clan has been created by *Tobacco use disorder* and *Obesity*. This means that they co-occur the same number of times. Same as before, the full lines mean co-occurrence above the threshold and the ones below are connected by dashed lines.
The graph on the right is constructed by considering the patient conditions that appear at least 250 times and co-occurrence threshold of 500. The interval size is kept the same as on the left graph. The list of items included on the graph is:

632 *Missed abortion*

272.4 *Other and unspecified hyperlipidemia*

*V*15.82 *Personal history of tobacco use*

*V*58.55 *Long − term use of aspirin*

*V*58.61 *Long − term use of anticoagulants*

The first clan, the one formed by *Long-term use of aspirin* and *Long term use if anticoagulants*. The items do not co-occur very often together because they are connected by dashed line. From the colors of the graph, we can understand that the items of the "Long-term" clan are found together with *272.4* more frequently than with *V15.82*, same as we can say that *272.4* co-occurs mostly with *V15.82*. This means that, It is very likely that if someone has history of using tobacco, they will also have hyperlipidemia and it is more likely to take anticoagulants when diagnosed with hyperlipidemia then when you have been smoking tobacco.

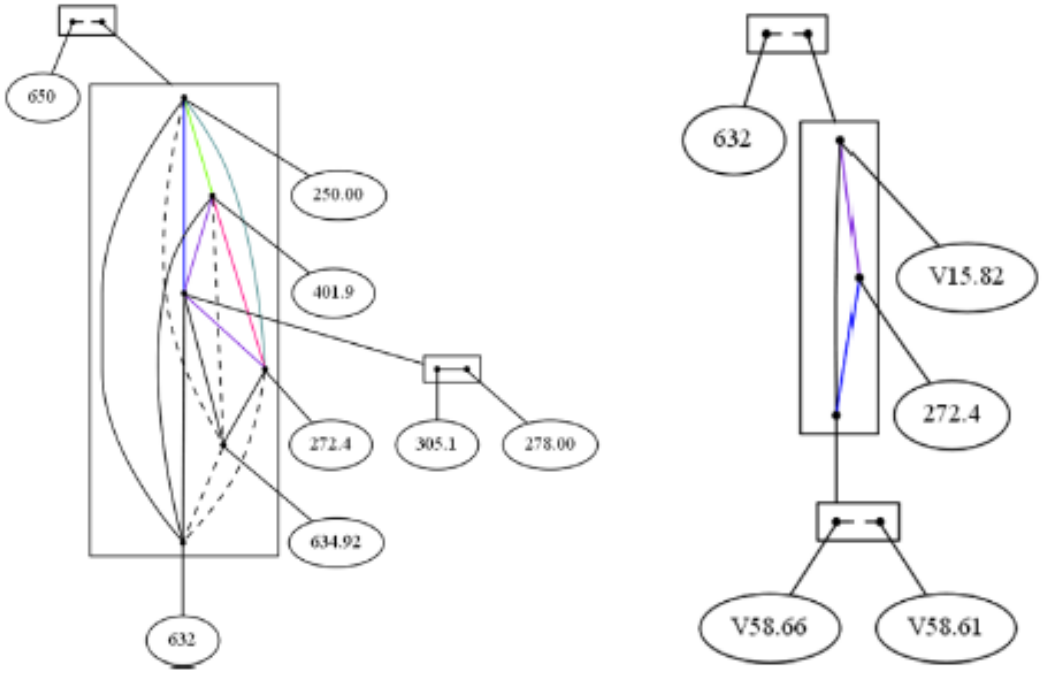

Figure 2.3: Gaifman graph linear variant

2.2.3.3 Shortest path Gaifman graphs

This variant can provide additional information about the data behaviour. The graph on the left includes all the items found at least 100 times and co-occur at least 8 times. As seen on the other variants, *Normal delivery* is not connected with the other diagnostics while *Tobacco use disorder* is connected with all of them. The so called *Hypertension clan* (the bottom one) is not connected with *Missed abortion* given the fact that they are connected with dotted line.

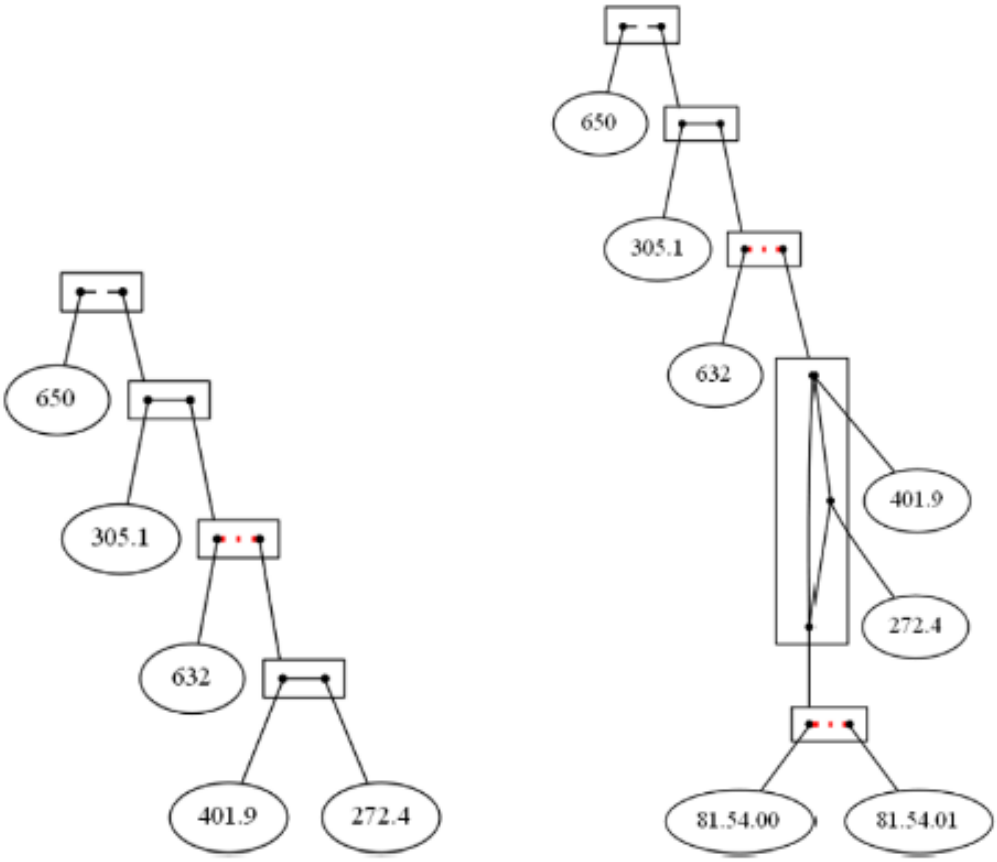

Figure 2.4: Gaifman graphs shortest path variant

The right graph includes all the diagnostics and the treatments that appear more than 100 times with the same threshold as before. The difference with the graph on the left, is the *Knee replacement clan*. As stated before, it is not very likely that the patient undergoes both knee replacements at the same time but the knee replacement seems to be connected with hypertension problems.
Apart from the theoretical study, an open source based on the algorithmic implementation was created to demonstrate their approach. Even though the created open source tool gives the wanted result, using it is quite difficult. Therefore, creating an interface to increase its usability is a necessity. For this reason, on the following chapter we will talk about "gamification" and Pygame.

3. Gamification through Pygame

"Gamification" is what everyone is talking about nowadays. It might seem surprising but even the big corporates are implementing it as part of their strategy. This is connected to the fact that when games or gaming elements and competitiveness are incorporated together in the daily tasks, not only the tasks are more lively but even the learner engagement is enhanced. There are plenty of studies that have shown the evolution of the learning process when games are included [3, 5].
Day by day, the gaming industry influence has increased, the data generated is huge and the improvements coming from the gaming data analysis are immense [1]. This brings us to the conclusion that games are designed to attract more people's attention and take most of their time. Given these facts, why not to try and use it to help people learn faster, improve their knowledge and make their work more exciting?

Gamification comes with many advantages:

- Motivation: Gamification makes learning fun and interactive not by turning work into a game but by making use of the psychology that handles the human motivation. Everyone has experienced at least once the feeling of competitiveness, the desire to improve and challenge others and sometimes even ourselves. People accept better certain intellectually effortful tasks if they are in a context of gaming
- Addiction: As we all know, games tend to be addictive. The same thing can happen with learning if gamification becomes part of it but do we retain everything we learn?Apart from addiction, Gamification helps on memorizing the knowledge as it can provide natural high. This "high" has a huge impact on knowledge acquisition.
- Learning experience enhancement: It is proven that learners who are enthusiastic about learning are more likely to remember what they have learned.

For all the above, we decided to use Pygame.

Figure 3.1

Apart from everything we have mentioned till now, I have decided to use PyGame because the main goal is to improve the previous open-source tool while making it more interesting and usable. PyGame is a set of Python modules which can be used almost in every platform and operating system, which makes it very portable. It is designed mostly for creating video games and it can be very easily installed by using pip tool and doesn't need setup tools.

3.1 Pygame advantages

Pygame is very time efficient since it uses optimized C and Assembly code for core functions, which are 10 to 100 times faster than python code. The best features of Pygame is the simplicity and the fact that it is the user who controls the main loop, they call pygame functions. In order to use all the functions, the user doesn't need a GUI because Pygame can also be used the command line. Given the fact that extra libraries and effects are developed separately, we can state that the amount of code needed is pretty small in pygame. The GUI module, is one of the extra modules, was introduced in 2019 and offers a modern appearance of all GUI elements and has no dependencies except Pygame. Some of the advantages of this module are:

- It offers extensive theming options for all the elements such as changing elements colors and appearance while the program is running, draw different shapes ect
- Allows rich text display (HTML can be used for styling)
- Allows full screen of GUI windows inside the pygame application
- Uses pygame structures and classes
- It is open source
- Still under active development

3.2 What can you do with pygame

It is very easy to add functionalities on the top of Simple DirectMedia Layer cross platform library which provides access to mouse, keyboard, audio, joystick and graphics hardware. This makes possible

the creation of fully featured games and multimedia programs using python programming language. In order to be able to use pygame, you first have to initialize its functionalities by using the command:

$$pygame.init()$$

After initialization, you will need to create a display for our game. To initialize it we will use:

$$pygame.display.set\_mode(width, height)$$

Since you are trying to create a game, you for sure will have object movements on the game display. For example, if you create animations, you are not doing something more than just creating changes between frames over time. To show these changes we need to update the screen after each change. There are two ways to update the screen in pygame. The first one is:

$$pygame.display.flip()$$

This function makes possible the update of the entire display. The second option is"

$$pygame.display.update()$$

Differently from the flip() function, this function can update only portions of the screen and it will save memory.
These are the main things one need to do to set up pygame and start creating the game environment. Now, its time to move to game logic by creating loops. Each game needs a main loop which is usually a *while* loop. Inside this loop, we can create all the logical loops we need for our game and iterate through the different types of events. To call the user events *pygame.event.get()* function needs to be called.
To quit the game, is as easy as it can be, just with one line:

$$pygame.quit()$$

This event is sent to the queue event when "x" sign is pressed. Everything mentioned till now is very basic. However, the amount of things one can do using pygame is huge. With pygame you can:

- Draw images using the command *pygame.image.load('image.png')* and *blit()* function to show the image on the game display
- Move images using their *(x, y)* coordinates
- Add screen boundaries (crashing games)
- Draw objects of different shapes using *pygame.draw()* function
- Display text using *blit()* function
- Create menus
- Keep scores
- Create buttons and make them interactive

And many more which can be explored depending on the game necessities.

4. Our Approach

As mentioned in the chapters 1 and 2, we want to reuse the graph decomposition software that Ely and her team has built by making it useful for everyone. For this, we have decided to use the best features of Pygame GUI. We do not choose any "game design" nor propose any specific game but that can be easily done once pygame works well together with the graph decomposition software

4.1 Our PyGame GUI

The very first thing to do is to create a window where we can add gaming elements such as button, sliders, text fields and where we are going to display the graph. This window will be the graphical user interface (GUI) provided by Pygame. Here how our GUI loks like:

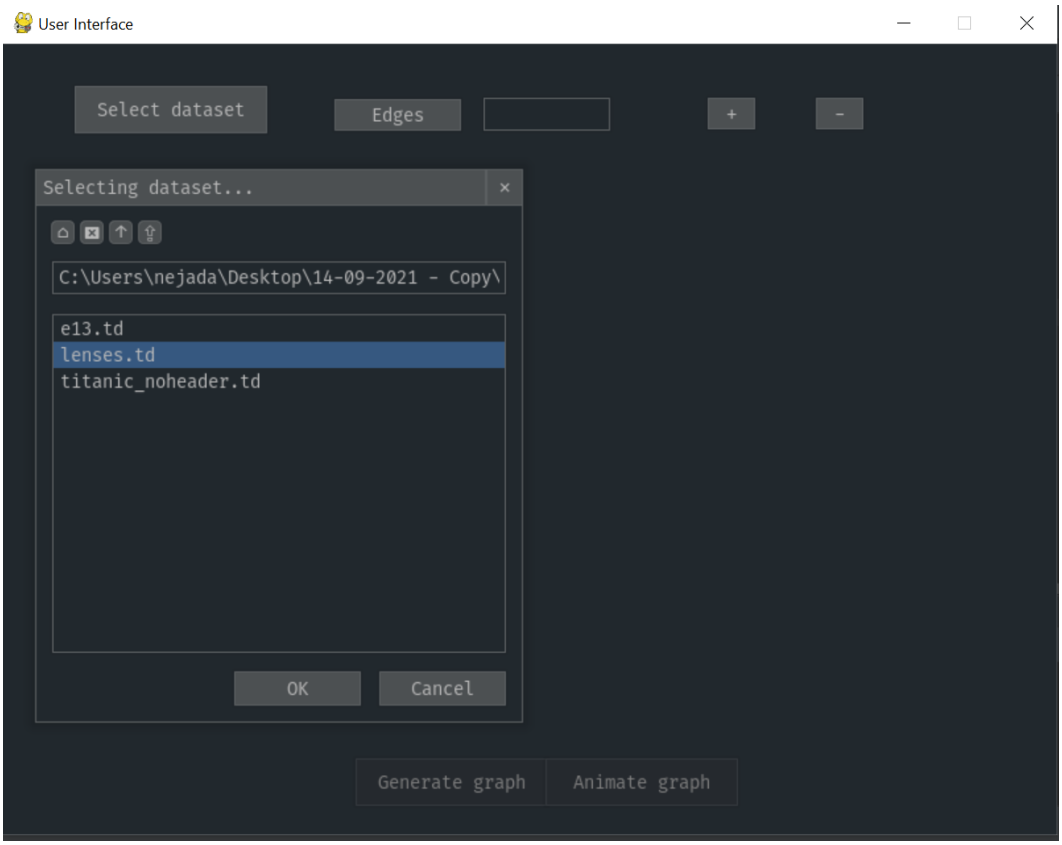

Figure 4.1: Our interface

The elements included in our GUI are:

- UIButtons
- UILabel
- UITextEntryLine
- UIFileDialog

We have many different buttons for different purposes. Starting with the first one at the top left corner of our window:

- Select Dataset button.

When this button is pressed, a file dialog is opened. Through this file dialog we can choose the name of the dataset we want to use for our analysis. In our case, the path where the datasets are stored inside our project is selected but this doesn't mean that you can not change it. You can put the datasets on a different location and use the file dialog to change or refresh the directory. You can also delete a dataset you select. It is very important to mention that the datasets we are using are transactional datasets. These types of datasets contain transactional data like the place, the time and all the related information about transactions.
Once the dataset is selected, the "OK" button will be activated. So, the first step is done. Now we need to specify the number of edges we want to visualize, otherwise the program will generate an error.
The number of edges can be specified in two ways:

- using the text entry line and then pressing enter so the program understands that we are done inserting values on the text field or by
- clicking the + and - buttons

No matter which option we choose, the number of edges is replicated in both of them. We will need to specify this number in order to define the size of the graph, smaller the number, smaller the graph. We are going to call it the edge's threshold. As we have specified the threshold, a so-called dot file with only the relevant items and information will be generated. Relevant items means the dot file will contain decomposition information only for the number of edges we have specified. This process is repeated every time we change the number of edges. **Note:** More detailed information about the implementation and the so-called dot files will be included below

Now, we basically have everything we need to generate the graph. At the bottom of the window, we can see two buttons:

- Generate graph button and
- Animate graph button

Generate graph button only generates the graph taking into consideration the number of edges the user specifies. Note that, if the number of edges is not specified, an error will be generated. As previously said, once the number of edges is specified, the corresponding dot file is generated. This dot file is accessed by the program and the graph is generated and drawn on the "game" display. Animate graph button works slightly different since it doesn't care if you specify the number of edges

or not. When clicked, this button will start generating the dot files from 1 to the max number of edges and you will be able to see on the screen the transition of the graph every time an edge is added automatically. When the maximum is reached, an pop up window saying that you have reached the maximum number of edges will show up. As the maximum number of edges, in our case, will serve a number specified by us. We have chosen 4 because we did not want to make the graph very complicated due to the fact that the result is not as clear and understandable as we wanted. In an ideal case, the max number of edges would be the sum of all the edges inside the dot file and would be different for each file. Somehow, this is a superficial description of what we see. On the next section, we are going to check in details what happens behind each selection or mouse click.

4.2 Implementation details

The very first thing to do is to import all the libraries and files we need for our project. Then we have to define the size of our game display using the command mentioned on chapter 3. To draw the graphs, we are going to use the dot files generated after the decomposition. I should mention here that the decomposition algorithms have been implemented by Maria Ely Piceno and her team, so, we are using her code to obtain the result. The study of the algorithms and the analysis of the output is out of this project scope. We are going to use the dot files obtained the way they are, without changing anything. All the auxiliary files needed to generate the decomposition graphs in dot format were given to me by my thesis supervisor. Two of those files were modified by me, in order to be used within my project, specifically **td2dot.py** and **seq.py**. The first file is used to read the graph in through a specific method. Inside this method, a threshold to consider only relevant information is specified. The file used the *read_graph_in()* method from *td2dot.py* file and some methods from Ely's decomposition files and generates the dot files with the information after the decomposition considering as well the number of edges we specify through our.

To be able to draw with Pygame, we need the coordinates of the different shapes we plan to show on display. The details of how we obtain a dot file that contains the *(x, y)* coordinates for each edge and node will be given below. Now, We will loop through all the nodes of the dot file corresponding to the dataset we have selected to extract their *(x, y)* coordinates. The same procedure will be repeated for the edges to get their start and end position. Each edge of the dot files contains more than one start-end coordinate, so, we have to split them into pairs and use those pairs to identify the connection points between items. Given the *(x, y)* coordinates for both, nodes and edges, we are ready to draw them on the screen on their exact location using Pygame *draw()* function. It is very important to mention that after each draw() we have to show the changes on the screen using blit() and then update() the display, otherwise the changes will not be visible for the user. This is how Pygame works.

Since after the decomposition, clans might be created, it is very important to identify the items of a clan and the singletons. For this reason, we have chosen to use colors. *Singletons*, which are the single nodes, will be circles (nodes) of bigger size while the clan items will be smaller circles colored in blue.

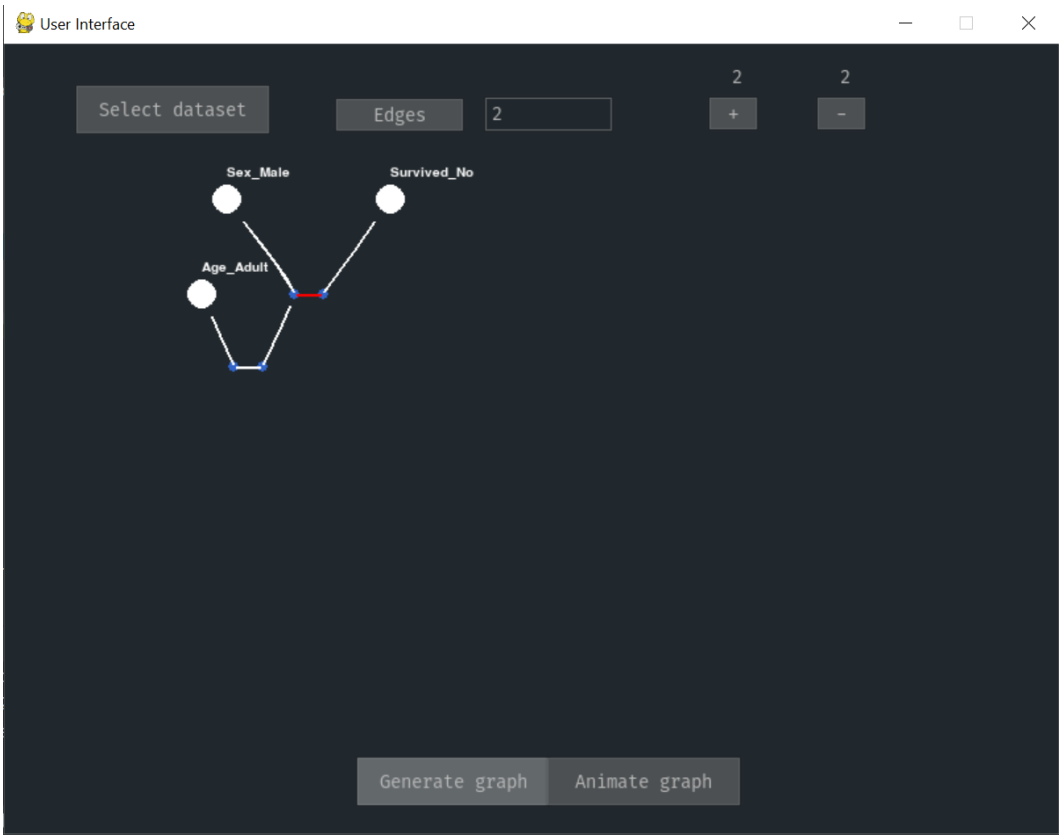

Figure 4.2: Titanic decomposition graph drawn with Pygame with edge threshold = 2

As you may have observed from the pictures of chapter 2 or remember from the description, there are two types of connections between graph items. The solid lines identify highly correlated items which co-occur together frequently while the dashed lines identify the items that are not frequently found together with other elements. For the time being, Pygame does not provide an option to draw the dashed lines. For this reason, we have used again colors to point out which elements co-occur frequently together and which not. The white lines of our displayed graph are used to identify highly frequent items, while the red lines are used instead of the dashed lines to identify the items that do not co-occur very often.

Every time we press a button, some method gets executed on the background. What I mean is that after each selection or button click, an event is fired. For example: when select dataset button is pressed, a file dialog is opened. The button click starts the execution of the methods (code) that are specified inside the *process_events()* function. This function includes all the user events of the projects and executes them when they are triggered. Some of the events we have are:

- QUIT fired when we press the "x" to close the game window
- KEYDOWN fired when a button is pressed
- HORIZONTAL SLIDER MOVED fired when we want to change the edge number using the slider
- TEXT ENTRY FINISHED fired when the edge number is defined by the text entry line.

- BUTTON PRESSED fired when we want to increase the edge number using the +/- buttons or when any other button is pressed.
- MOUSE BUTTON DOWN fired when we click on a node

Till now, we know how we extract the *(x, y)* coordinates, we know how to select the dataset and how to define the number of edges but we do not know how to obtain a dot file that contains the information about the node and edge coordinates. Here is how it is done:
After the event for the edge number is fired, the decomposition algorithms are executed and a dot file is created. This dot file does not contain the *(x, y)* coordinates. For this reason, we use the AGraph class of PyGraphViz. We need the AGraph to add all the graph items which satisfy the constraints. In order to have a dot file that contains the *(x, y)* coordinates we need to call the *layout()* method and put inside the brackets the parameter *progr='dot'*. The command would look like this:

$$graph.layout(progr =' dot')$$

The above process might be repeated k- times if the edge number is changed using one of the options we mentioned before. Of course, in case of +/- buttons, the - button will be disabled for k=1 because to have a graph we need at least one edge, and when the maximal number of edges is reached and we still try to increase it, a warning message will appear saying: *"Maximum Number of edges Reached"* and the + button is disabled as shown in the picture:

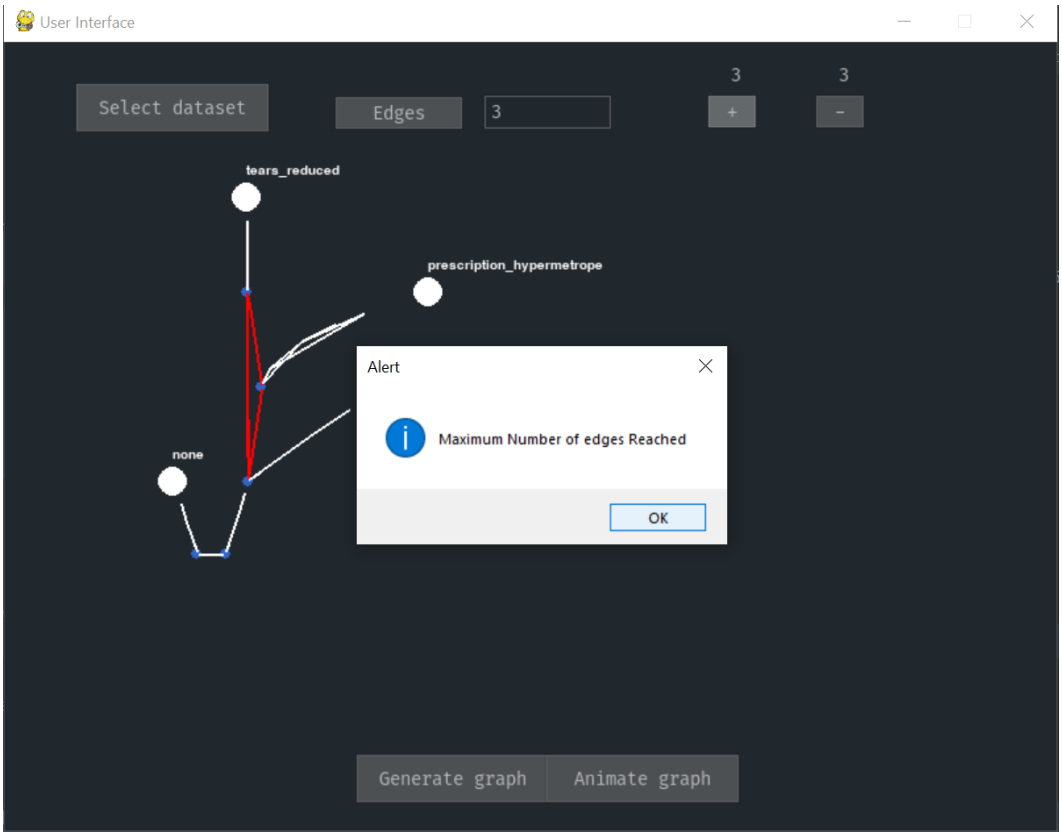

Figure 4.3: Maximum number of edges reached

In our case, the maximum number of edges is set to 4. The picture shows 3 because we clicked on the + button to increase the edge number and the message window popped up. After clicking *OK*, the edge number is updated and the graph with 4 edges is generated automatically.

4.3 Future work

As part of our future vision, we have implemented a method which identifies if we have placed the mouse over one of our graph items and at the moment we click over it a message window pops up saying: "Selected node *node_name*".

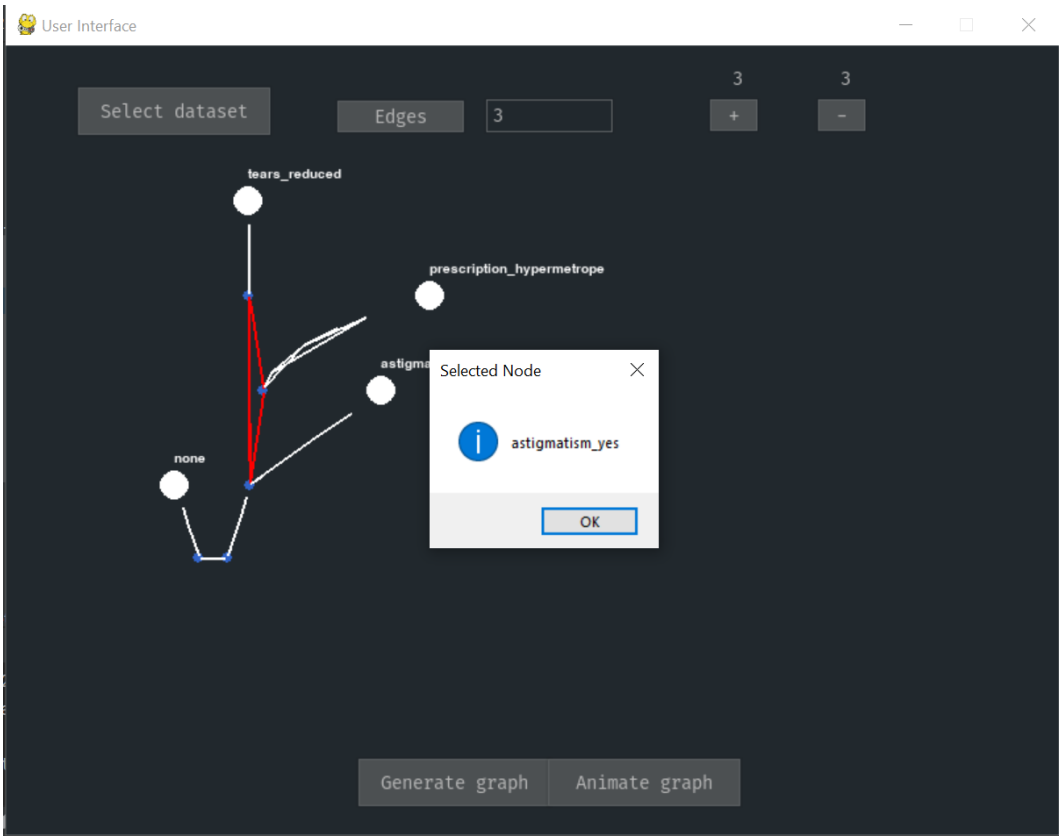

Figure 4.4: A node is selected process event

Of course, right now this event doesn't tell much about the node but the reason why we thought on implementing it was because its a feature that might be mastered in the future. Here is a list of potential improvements:

- Clarification of the cases when the graph gets more complicated than expected.
- Drawing the boxes. Being able to draw the boxes to identify the clans would make the graph more readable.
- Dashed lines. The representation of non frequent items for the moment is done by using different colors but it would be a much better solution if we could directly draw them as dashed lines.
- Design real "games" on top of all this to make the experience more impressive. Imagine having a graph which you can collapse or expand. By clicking over one of the nodes, you can either collapse the graph by grouping all the nodes with similar or correlated information together or do the opposite, expand the graph and identify all the items that were grouped together. This is the reason we have implemented the pop up window shown on the above picture.
- Obtain node information. Another thing to improve is the pop up window when a node is selected. It might be possible to be able to see the node information when hovering the mouse over it. This information might be the number of nodes collapsed or just general information about the item.

5. Implementation difficulties and Results

As part of the result we can say that the integration process between Pygame GUI and the decomposition algorithms was not as easy as we thought. Even though Pygame has improved significantly overall, Pygame GUI is still a new module and does not provide all the necessary features needed for data mining. We can mention as an example the dashed lines.

In the beginning the project started with png files. From the dot files, we had to draw the .png files and display them on the screen. This was a simple task but the things got more complicated when the drawing needed to be done by Pygame. For this reason we needed to create the dot files using *layout()*. The transition between displaying png files and drawing with Pygame was one of the difficult tasks of the project due to the fact that it required the integration of decomposition algorithms and the extraction of the *(x, y)* coordinates for both vertices and edges.

Also the coordinate extraction had his own difficulties because we had to loop through all the elements of the graph, identify the *clans* and *singletons*, connect the different *(x, y)* coordinates for all the edges and decide which vertex information was relevant to be shown on the graph and which not.

Something else missing in our project are the boxes which identify the clans. Drawing them was more difficult than expected so we decided to simply identify the elements of the clans by using different vertex size and color.

On chapter 4, we have shown the result drawn with Pygame and it looks clear and readable but we also mentioned that the maximum number of edges we decided to draw are 4. The reason was stated, because the graph would be very big and not very clear so lets see how it might look like.

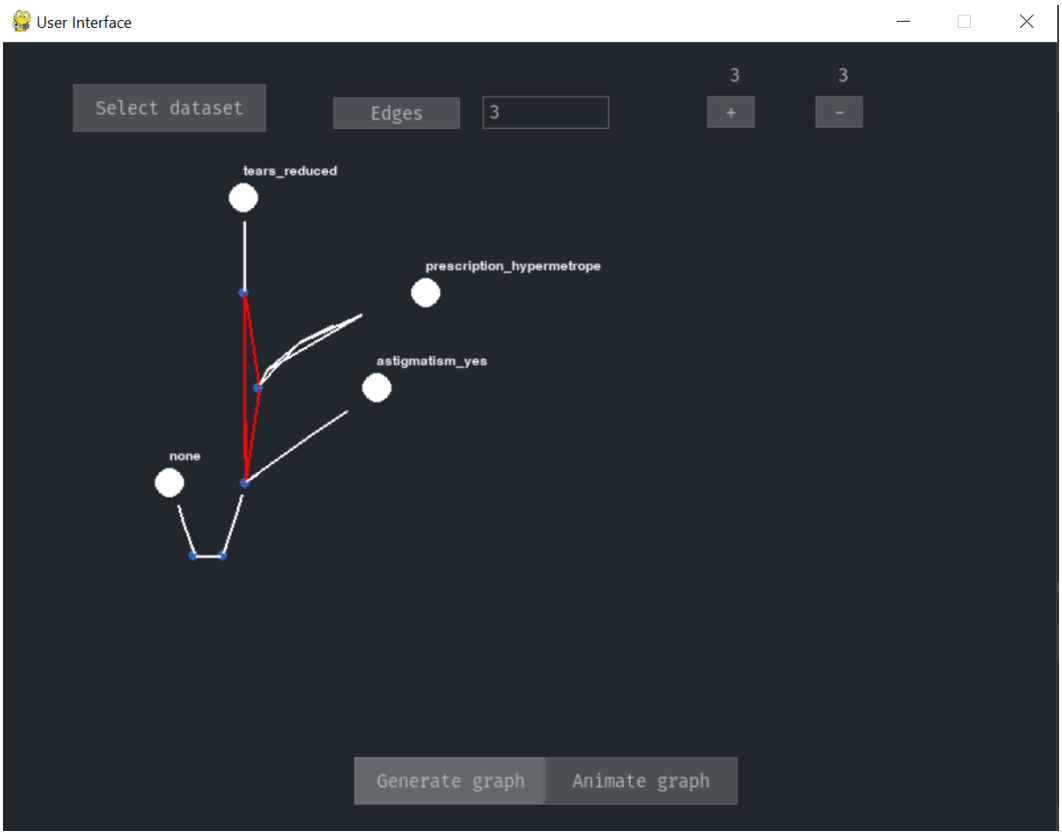

Figure 5.1: Lenses dataset decomposition graph drawn with Pygame with edge threshold = 3

The formed clan is very clear but if we increase the number of edges to our maximum (4) the result will not be as clear as it has been till now. For unknown reasons we get many "dashed lines" which we are not sure if all needed nor drawn on the right place. As a result the graph and the clans are no longer readable nor understandable.

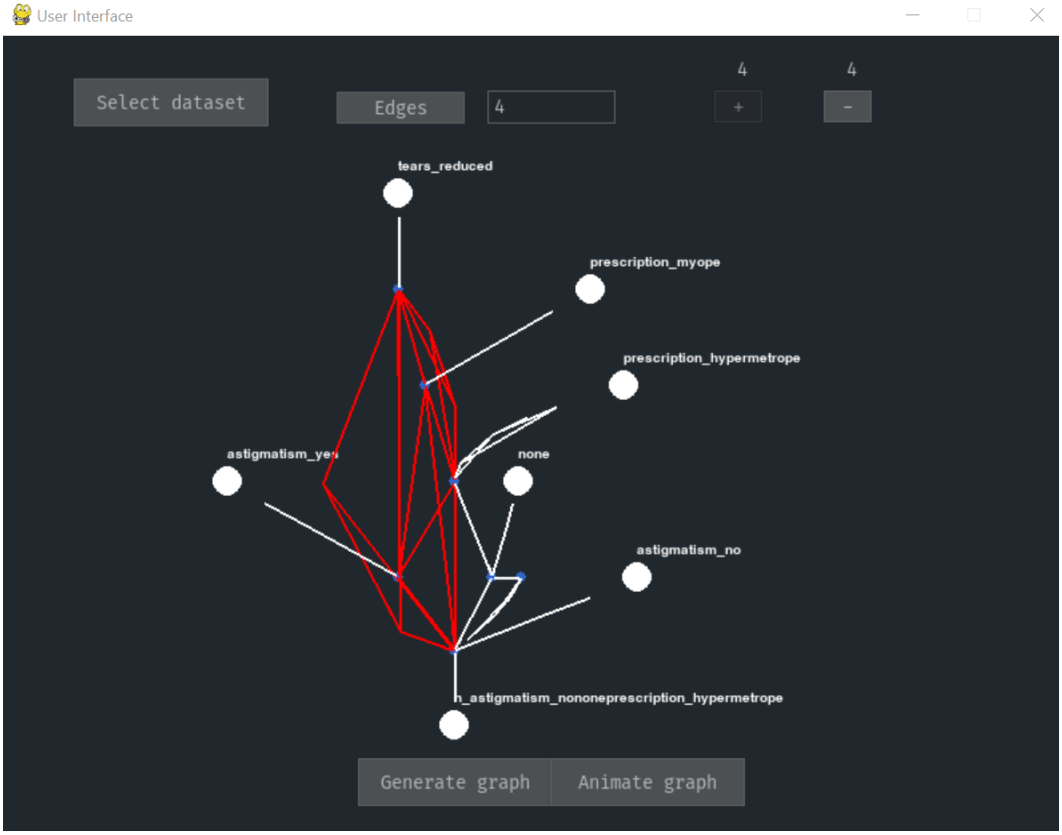

Figure 5.2: Lenses dataset decomposition graph with maximal number of edges

Finally, we can say that the idea is doable and likely to be successful but much harder than than anticipated. An implementation to which "gaming" ideas could be added as the whole power of Pygame is there [10]. It could also be possible, as a future work or development, to sell data mining activities in general to end users as some sort of game. For the time being if offers a window from Pygame into the outcome of the decomposition algorithm.

Bibliography

[1] Hycinta Andrat and Nazneen Ansari. "Integrating data mining with computer games". In: IEEE, Apr. 2016. DOI: 10.1109/ccaa.2016.7813717. URL: https://doi.org/10.1109/ccaa.2016.7813717.

[2] A Ehrenfeucht, T Harju, and G Rozenberg. *The Theory of 2-Structures*. WORLD SCIENTIFIC, 1999. DOI: 10.1142/4197. eprint: https://www.worldscientific.com/doi/pdf/10.1142/4197. URL: https://www.worldscientific.com/doi/abs/10.1142/4197.

[3] Carlo Fabricatore and Ximena Lopez. "Sustainability Learning through Gaming: An Exploratory Study". In: *Electronic Journal of e-Learning* 10 (July 2012), pp. 209–222.

[4] Haim Gaifman. "A Theory of Higher Order Probabilities". In: *Proceedings of the 1st Conference on Theoretical Aspects of Reasoning about Knowledge, Monterey, CA, USA, March 1986*. Ed. by Joseph Y. Halpern. Morgan Kaufmann, 1986, pp. 275–292.

[5] Katie Larsen McClarty et al. "A Literature Review of Gaming in Education. Research Report." In: 2012.

[6] Marie Piceno, Laura Rodríguez-Navas, and José Balcázar. "Co-occurrence patterns in diagnostic data: NA". In: *Computational Intelligence* (Apr. 2020). DOI: 10.1111/coin.12317.

[7] Marie Ely Piceno. *Data analysis through graph decomposition*. 2020. URL: https://upcommons.upc.edu/handle/2117/330388?show=full.

[8] Marie Ely Piceno and José Luis Balcázar. "Analysis of Co-Occurrence Patterns in Data through Modular and Clan Decompositions of Gaifman Graphs". In: *CoRR* abs/1910.05146 (2019). arXiv: 1910.05146. URL: http://arxiv.org/abs/1910.05146.

[9] David Slutsky. "The Effective Use of Graphs". In: 03.02 (May 2014), pp. 067–068. DOI: 10.1055/s-0034-1375704. URL: https://doi.org/10.1055/s-0034-1375704.

[10] Al Sweigart. *Making Games with Python & Pygame*. 2012.