Final year's internship

ComplexCity
Shanghai University Laboratory of Data Science

Clustering of cities using graph kernels and graph coarsening

上海大學中歐工程技術學院數據分析實驗室

Student
William REVERCHON
Ms in Mechanical Engineering
Belfort-Montbéliard
University of Technology

Supervisor
Dr. Fabien PFAENDER
Data Science researcher
Shanghai University

February - July 2021

List of Figures

Contents

Abstract

Cities can be modelled as complex systems defined by their socio-cultural-economical indicators as well as their topology. Here, we follow the trend in the literature of representing cities as graph objects so that we can manipulate them using algorithms from the field of graph theory. Our goal is to cluster cities accordingly to their topological similarities. We use method known as machine learning techniques from the family of unsupervised and supervised learning. These learning techniques permit to treat massive amounts of data, to spot the similarities between them. The algorithms require the data to be express as an object belonging to an euclidean space. Yet, graphs are mathematical objects not belonging to an euclidean space. The mathematical nature by which they are defined raises a challenge from their modelling's standpoint. They require a preprocessing before we can *feed* it to a machine learning pipeline. Graphlets are small graph motifs, and a known way of characterising networks. From here arise our problematic that is to find a method to describe our graph, using a novel coarsening technique based on graphlet and then another method to cluster them.

Keywords— graphlet, data science, urban, cities, graph, kernel methods, graph embedding, graph coarsening, unsupervised learning, machine learning, statistics

Acknowledgements

I was supposed to start this internship in Shanghai the 1th february 2020. Nevertheless, the sudden event of the coronavirus pandemic made it difficult to start the internship. So I would like to thank Ms. Alison BARBE, Mr. HORLACHER Jérôme from my engineering school Belfort-Montbéliard University of Technology (UTBM) for keeping up with my administrative situation and making it possible to delay my internship start to 1st February 2021. I first would like to thank my lab tutor Mr. Fabien PFAENDER from the laboratory also for the time he dedicated for finding the best conditions to do my internship. My work wouldn't have been possible without my tutor providing resources such as previous works, courses on which I could learn concepts of data science, machine learning. Meanwhile, I'm thankful for the users of stack-overflow for unblocking any theoretical questions or code issues. I thank the administration of Brest Faculty of Medecine, and Brest University's Faculty of literature and Brest's Department of Literature for letting me work in their amphitheater, using their huge blackboards. Their infrastructure were vital for letting me expose ideas on it the blackboards and understand the research papers. As well, I thank my lab tutor for answering my questions, solving technical problems, guiding me through the tremendously wide research process during the internship. That way, he transmitted to me his enthusiasm about graphs, data science and machine learning.

Chapter 1

Introduction

1.1 Working environment

1.1.1 Laboratory

I worked in the laboratory of the Sino-European University of Technology of Shanghai University (UTSEUS) in China. The lab take care of projects of data analysis for urban planning, complex systems for smart cities and sustainable cities of the future[1]. In February the lab received a project from the government of Java Island, lying between Sumatra and Bali, in Indonesia. The goal was to study the paths that can link the north to the south of the island for a urban planning project. One of the problematic was to choose which was the optimize path to build roads, considering the preservation of local villages and other urban parameters. Those projects take advantage of the modelling of cities as huge geometrical objects and as graphs. These processes are part of what is so called "big data". I worked under the supervision of Mr. Fabien PFAENDER, a urban data scientist in the laboratory.

1.1.2 Project

The goal of the internship is to make clusters of cities according to their similarity. Define a method to describe the structure of a graph based on a graph coarsening technique of graphlets. As well, find methods to analyse a set of cities road graphs to see if we can find possible underlying inner structure orders in the data set. This subject has direct application in urban science. It is part of a complex network scientific project lead by Fabien PFAENDER from Compiègne University of Technology-UTSEUS and Egon OSTROSI (UTBM).

1.1.3 Planning

The internship was decomposed into main parts with different objective in each. To organize myself, I have a set of goals for each week. We set up update reunions with Fabien PFAENDER to set of the next goals, so that he explains the ideas he has for the project. The first month objective is the install the working environment of python, jupyter notebook, graph-tool library, scikit-learn, numpy, pandas, geopandas etc. Also, I had to being to code with these libraries to get myself familiar with them. For example, creating a graph, manipulating the GeoDataFrame to plot the geometry of a city, and create the graph using graph-tool. My lab tutor gave me the task of computing some graph indicators such as betweenness centrality, compute the shortest path between a north point and a south point on a graph of the city of Java, located in Indonesia. As well, on the side, I

[1] https://utseus.shu.edu.cn/en/Research/Research_at_UTSEUS.htm

read blogs, articles, codes, courses on graph theory, graph clustering ways, graph embedding, graph convolutionnal network to get familiar of what techniques exist in the literature. The goal of the first month was to set up the environment and read about the state of the art of graph coarsening, graph embedding, graph convolutionnal networks and get a feeling of what all this is. The installation of the programming environment was difficult. I passed hours to solve the dependencies issues between graph-tool and the other libraries. Graph-tool is by the way known for being delicate to install, having many dependencies.

From about the second month half, I had the objectives of continuing to use the libraries to get familiar with them. On the other side, I have to read articles, blogs (any viable source) about graphlet, what are they, why are they usefull for describing a graph's structure, and especially, how to count them in a graph. The counting of graphlets in a graph is a problem that very treated subject in the research literature of graph analysis. I also read many courses on set theory Lefebvre-Lepot [2010–2011], first order logic, because most of the paper where using these fields to describe what it was they were doing with the graphs. As well, these are the languages I will have to use to describe the coarsening algorithm for capturing a graph's structural information that Fabien had in mind.

From the third month's, I read more articles literature, about graph coarsening, graph reduction and slightly about graph partitioning, and local graphlet counting. These articles helped me a lot to view how coarsening was described in the literature, formally. From there, I began to think more about ways to coarsen a graph, which nodes to combine together, combine them according to which rule, what information to collect during the contraction. Although I didn't know what the coarsening method will exactly look like, I began to be want to try to combine nodes according to their graphlet degree vector. So I searched for codes, libraries, or algorithms permitting us to do that. I found a paper describing how to count the local graphlet that an edge belongs to, from Ahmed u. a. [2016]. I also started to code using numpy and graph-tool the algorithm 2 LOCALGRAPHLET from this paper. The code is available in my github[2]. After months of theoretical ideas, it felt really good to have a grasp of a practical code that we will use.

The next month, I found a paper describing how to coarsen a graph Loukas [2019], given a partition P. I passed weeks to try to formalize the algorithm to coarsen a graph given graphlets in the graph. Nevertheless, I couldn't find a way to partition of graph into a set of contraction sets (given their graphlet degree vector). So Fabien suggested that we abandoned the idea of graph coarsening for my internship. Indeed, at that time I couldn't think of a way to formalise the rule to combine nodes together. Fabien suggested that I think of creating new graph kernels based on graphlet. The previous work of coding the LOCALGRAPHLET can also be used to create graph kernels.

From may, I read about graph kernel in other to rain myself to reproduce the examples I could find, so as to get myself familiar with what are kernels and how to design them. I searched and tested simple graph kernels (degree kernel, graphlet kernel) on a small dataset of simple graphs (lattice and circular graph) that already existed in literature. I also aimed to reproduce the results of previous intern Dupré [2016], performing classification of lattice and circular graph using support vector machine.[3]. Then, I proposed a kernel derived from LOCALGRAPHLET. Meanwhile, I found another code of counting orbits for each node, that is to be short, a more specific graphlet degree counting algorithm. I also worked on making a kernel from this.

Later in June, I found out a way to combine nodes together and create a algorithms that seems to converge to the same partition given a graph G. That partitioning algorithm is described later in this report.

During the last month, I extracted and choosed a dataset of cities using OSMnx. Fabien adviced to run the kernels I coded in a dataset of French cities. The last month was basically made to

[2] https://github.com/endingalaporte/alg2-Exact-and-Estimation-of-Local-Edge-centric-Graphlet-Counts/blob/main/_graph_tool%20LOCALGRAPHLET.ipynb

[3] https://github.com/endingalaporte/Shortest-path-graph-kernel/blob/main/_gt%20shortest%20path%20kernel%20and%20degree%20kernel.ipynb

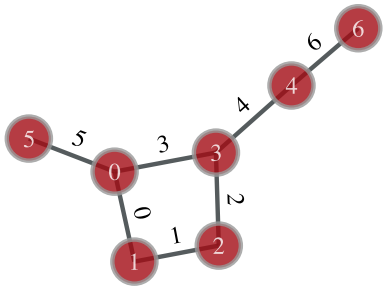


Figure 1.1: Graph G created using graph-tool library from the sets E and V

finishing writing the kernels and running them on a dataset of cities.

1.2 Preliminaries

1.2.1 Notations

A graph is a mathematical object composed of a set of vertices and a set of edges. Nodes can be linked together in pairwise relation forming edges. Vertices are interchangeably be called nodes. Two vertices linked together forms an edge. Let $G = (V, E)$ be a graph with $V = \{v_0, v_1, \ldots, v_a\}$ and $E = \{e_1, e_2, \ldots, e_b\}$ respectively the sets of vertices and edges. We denote the number of elements in a set by it cardinal $Card(.)$ or $|.|$. That way, $Card(V) = |V| = a + 1$. Each edge is composed of two nodes so that $\forall e \in E$, $\forall (v, u) \in V^2, e = (u, v)$. The degree $d(v)$ of a vertex $v \in V$ is the number of vertices to which the vertex is connected. $\forall v \in V$, let $\Gamma(v)$ be the set of neighbors of v so that $\Gamma(v) = \{w \mid (w, v) \in E\}$. The degree $d(v)$ of the node v is its number of neighbors $d(v) = |\Gamma(v)|$. I invite the reader to get an insight of what some basic graph manipulation such as graph creation, iteration in nodes, getting neighbors of a node looks like using the library graph-tool. You will find the code to create the graph of Figure 1.1 available in my github.[4]. Graph-tool has for example, the functionality of plotting the graph with the index of each edge and vertex attached to them. Graph-tool library provides high-level tools for the manipulation of graphs, which makes it handy. Graphs are only defined by the bi-sets V and E. So manipulating them and working with them is manipulating these two sets. In the literature, all paper related to the manipulation of graph comes back essentially to doing operation on this sets ; remove, add node, check if an edge or a node exists in the set, check if the node has neighbors, check if the nodes have edges in common and more. The manipulation of these sets relies strongly on notions of set theory and first order logic.

1.2.2 Graph are everywhere

Graphs are used in various domains. Each, we represent city road network as a graph. In social sciences, it is used to represent social networks where a node is a person, and the edge connecting them the social relationship they have. In Biology, proteins can be modeled as a network of amino acids. Or it can be used for disease spread modelling of the coronavirus Khorshidi u. a. [2020]. The flight search engines represent each destination by a node, and the edges connecting flights between destinations. In other words, graphs are widely used to represent topology of an object or relationships in a network. The algorithms we develop here can be used for other graphs representing other things. That adds weight to the utility of that research project, makes its applications wide and multidisciplinary.

[4] https://github.com/endingalaporte/Identify-graphlets-in-subgraph/blob/main/_graph-tool%20routines.ipynb

1.2.3 Notion of invariance

We talk here about the notion of invariant indicators, also called graph invariants in literature Washio und Motoda [2003]. A graph $G = (V, E)$ is defined by its sets of edges, and vertices. We could think that we could directly feed these sets into a learning machine, in order to describe the global structure of the graph. But, it is not relevant. Indeed (V, E) sets are dependant on node numeration. We can change the number of each node, while preserving the exact graph structure and the sets E and V would change. The same way, graphs can also be stored in an adjacency matrix $A \in \mathscr{M}_{|V|}([0, 1])$ space of symmetric matrices with each of its element that takes values 0 or 1. $V = \{v_0, v_1, \ldots, v_a\}$. A defines if a node v_i is connected to node v_j by the presence of 1 or 0 in the case of no edges. $A = \{A_{ij}\}^2_{(i,j)\in[0,1,\ldots,a]}$. $A_{ij} = 1$ if v_i and v_j are connected by an edge, if not $A_{ij} = 0$. We could think that this is a graph graph representation technic but in fact, it is highly depend on node numerotation. A and G are objects that are *variant* regarding the permutation of nodes. Node numerotation is subjective. And there is no given objective order to organize the node numerotation. If we permute node names, A and G would change. We don't want to use A or G directly to perform clustering or classification.

In this project, we focus on finding indicators that are invariant of node permutation. We don't explain here why the indicators are invariant to node permutation. Because, it is inherent to their mathematical definition. For example, the betweenness centrality, the maximum and average degree, the number of graphlets are invariant indicators. In general, these invariant indicators are expensive to get, we need to compute them, thus there is a problematic in the literature of computing invariant indicators that describes well the graph while being cheap to compute. In the next section 1.3.4, we compute some invariant indicators such as minimum degree, average degree, graph density.

1.3 Cities as graph

1.3.1 Data extraction

To extract our cities road networks, we will take advantage of the open source global scale cartography project OpenStreetMap (OSM). OSM is a collaborative project to create a free editable map of the world. Users can trace lines lines of buildings shapes, roads, railroads, bridges. As well, attributes are attached to the lines and surface to indicate if it is water, a highway, or even the speed of the highway, if a building is a shop or not. To put it simply, we can roughly say that is an open source version of the popular consumer application offered by Google : Google maps. Although it is a participative project, its data is free of Open Data Commons Open Database License, reliable. OSM is used by over the world – researchers, governments, organizations or even companies such as Baidu Maps or the GAFAs Amazon and Facebook. The data can be extracted in .xml format where each line is described by its cartesian coordinates in the corresponding geographical projection, as well with attributes attached to it. It would require tremendous effort to process the raw data, to clean it in order to get the corresponding geometry assigned with the different attributes. That's exactly what Geoff Boeing from Department of City and Regional Planning, University of California, Berkeley did. His framework Boeing [2017] permits to extract geometries in OSM delimited by given administrative borders. There is even a tool for extracting simplified undirected graph from that geometry. That's the function of OSMnx we will use.

1.3.2 Dataset

So, by city's topology, we are talking about how the streets, roads, paths are arranged. Everyday, pedestrians, transports, merchandise, information are flow that travel through these paths. And how these flow are carried through the city is specific to how a city is organized. Since centuries, cities are organized in particular ways. These reflects the culture, the history or sometimes just

randomness. With the emergence of huge quantity of data from satellites imagery, the so called "Big Data" revolutionized cartography and how humans we navigate. We can get a world-scale view of how the world is. As well, we can start to model city on a large scale to see their similarities, their properties, as complex systems. Cities are such mirror of modern civilisation, their organisation reflect how we live, how we decide to produce. For sure, they might contain some information about these informations Boeing [2021]. Can we guess, from a city's topology if its economy rely mainly on finance, automobile manufacturing, or tourism? Can we link the Gross domestic product of a country to the topology of their cities? The question of an urban planner would be "Is there an optimal topology for a city of the 21th century?". As well, can we guess, according to a city's topology, some socio-economical indicators such as a country's Gross domestic product (GDP), demography etc. Cities of the United States of America are build upon a urban model called hippodamian grid. For short, their city are grid organized. The grid plans are straight streets, cutting one another at right angles. It has be shown previously by Dupré [2016] that that we can differentiate USA and French cities only by analyzing their city's graph. The way a city is organized reflects the macro organization of how people in this city live, play, work. So maybe there is some structure underlying all these cities and we can group them and even find structure and new discoveries. My lab tutor proposed to me to first gather a set of 100 French metropolitan "communes" regions to see test my code and to see if we can find some underlying structure in the data. I also choosed to extract 33 capitals of main regions in Mainland China, Hong Kong, Taiwan, Macao. Let

$$\mathscr{D} = (G_0, G_1, \ldots, G_n) \tag{1.1}$$

be our dataset of graphs with $(n+1) = 103 + 33$. OSMnx library has a function that extract graph and store it as a networkx object. So export the edgelist, that way I create a graph-tool graph from it. It took around 1 hour to extract all 103 french cities graphs using OSMnx. The code for extracting the 136 cities is available in my github[5]. Extracting the geometry of Paris' roads returns a GeoDataFrame object in Figure 1.2. We associate a label to each graph that is its country France or China, 0 for France, 1 for China as in

$$\mathscr{Y} = \{Y_i\}_{i\in\{0,\ldots,n\}} \qquad Y_i \in \{0, 1\} \tag{1.2}$$

The python library Graph-tool is significantly faster then the library NetworkX to manipulate graphs. Plotting the graph of Paris' road network took 60 seconds to compute on a laptop configuration.[6]. Here's what the road graph of Paris looks like in Figure 1.3.

By comparing the two representations of Figure 1.2 and Figure 1.3, we can see similarities such as a circular shape and lines of edges that seems to represent the "seine" river. However other interpretations by visualizing the graph should be thought with care. Indeed, by the nature of graph object, nodes can move around while staying the same object. That's why a visual of the graph should only be taken as an intuition of what the data looks like. The graph can be drawn around in different ways. If we plot multiple times, each plot would shows a different layer configuration of the nodes of the graph.

Now that we have our data set $\mathscr{D}$ of graphs, we can begin to analyze it. The representation of our cities as graphs makes it handy for analysis. Now, we have a wide range of algorithms based on graph theory that we can use. We don't have to manipulate the geometry cartesian coordinates points. As well, although the graph nodes have degree of freedom, its pairwis relational nature (edges of nodes) makes it a good candidate for retaining the topological information of how the flux of people circulate through a city, by its roads, paths.

[5] https://github.com/endingalaporte/Extract-cities/blob/main/_osmnx%20french.ipynb
[6] Intel Core i5-3210M, 10GB, Mac OS 10.11.6

Figure 1.2: Plot of Paris' geometry using geopandas

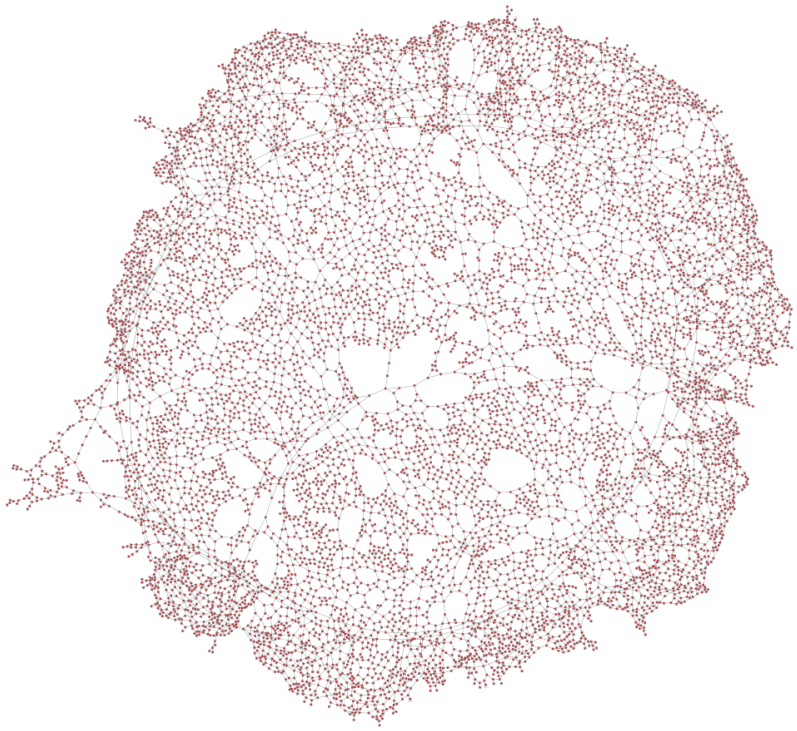


Figure 1.3: Plot of Paris' graph representation using graph-tool

1.3.3 Tools

During the internship, I used different libraries for manipulating the cites and analyzing them. OSMnx Boeing [2017] is a python library build on top of python for extracting data from OSM. Pandas is to manipulate huge data tablesMcKinney u. a. [2011]. It is a must have. GeoPandas is a python library built on top of Pandas. It adds the functionality of manipulating geometries, geospatial dataJordahl [2014]. While graph is our main object of study, we always keep GeoPandas at hand in for visualizing how the city looks like, without the deformation of graph's nodes. As well, when querying the data using OSMnx, we also store the attributes of each road in the GeoDataFrame such as speed, address of the street. From those, we can pass them into the graph using graph-tool. As well, we really rely on GeoPandas in order to visualize some properties of the graph. Let's say we have a graph of Hong Kong to Shenzhen, China. We want to visualize the shortest path between the Central district of Hong Kong Island and Shenzhen, China borders. Graph-tool will returns a list of edges that is the shortest path. Then, we pass the values to the GeoDataFrame. And we plot it while highligthing the lines corresponding to the shortest path. That's a good example of how GeoPandas and graph-tool are complementary tools.

Graph-tool is the most important tool here, as it permits us to manipulate, create, perform statistical analysis on our graphsPeixoto [2014]. There are other network analysis python libraries such as the popular networkx. Nevertheless graph-tool has a the benefit of being user-friendly used with python syntax and written with underlying packages written in C++ (mainly based on Boost Graph Library), a much lower level language then python. Indeed, it is pretty fast ! Performances are comparable as if graph-tool were a pure C++ library. Graph algorithms such as graphlet, betweenness centrality, shortest path ... have the reputation of being heavy computationally. Often, their range of use is limited by their time complexity. We are obliged of making approximation , by sampling data instead of analyzing it entirely. So, graph-tool, by its speed and relatively accessible handling, opens us a new wide range of applications and coding possibilities for processing our graphs ! Only one drawback remains, graph-tool is pretty hard to install. It relies on many libraries dependencies. It took me a non negligible amount time just to install it in my Anaconda environment.

Next library I used a lot is Numpy Van Der Walt u. a. [2011]. Numpy, build on top of C language permits to do fast operations on multidimensional arrays of numbers or objects while offering handy functions for scientific computing. It has the advantage of having a syntax similar to mathematical operations on matrix, vectors, tensors. I used numpy arrays to store the vertex properties of each graph. As well, I used the numpy arrays to store the $(n+1)$ graphs as graph-tool objects.

Last is scikit-learn. Scikit-learn offers a wide range of unsupervised and supervised learning algorithms so called "machine learning" implemented in pythonPedregosa u. a. [2011].

I use the integrated development environment jupyter notebook to edit and run code. Each of the codes in this report, will be showed as notebooks.

1.3.4 Statistical description of our cities

Each city is represented by a graph $G = (V, E)$. A city's road network is composed of streets, roads, intersections. As well, some roads are not straight but curved. The graph represented each intersection between two lines by a vertex. A straight road described by two consecutive nodes and their edge. When a road is curved, its curvature is described by a set of consecutive nodes of degree 2.

Each graph forms one component, $|C| = 1$, there is no isolated node or isolate graphs. Each city is represented by one graph. each pair of nodes can only have one edge between them, no multi-edges. Let χ be the space of undirected, simple (no multi-edge), one component graphs.

We have 103 french cities. We also choose to extract 33 other cities, capital of each province of China and its autonomous regions. And each of these graphs belong to χ so that $(G_0, G_1, \ldots, G_n) \in \chi^{n+1}$. That makes a total of $(n+1) = 103 + 33 = 136$ cities in our data set $\mathscr{D}$.

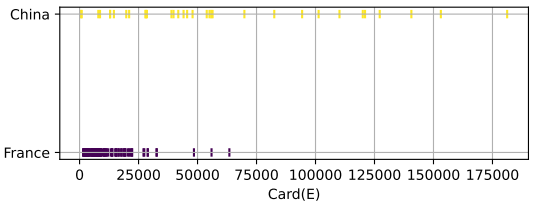


Figure 1.4: Distribution of graph size, for cities of China and France

The size of a graph is $|E| = Card(E)$. The Figure 1.4 is scatter plot of the the size of the cities for French and Chinese cities ouf our dataset. We immediately see that most of the cities in our dataset that are Chinese can be from $|E| = 1000$ edges to more then 175000 edges. Whereas, the most of french cities are smaller and the majority of their size are concentrated between 1000 and 25000. Cluster cities according to their size is trivial. We will concentrate in this report on ways to describe their topology and to group them by this criteria, regardless of their size. We will reduce the influence of size $|E|$ for example, in the case of the kernels, by normalizing the vector ϕ so tat only its shape is taken into account and not its size.

In each graph $G \in \mathscr{D}$, we compute the degree of each node. Then, we identify the minimum degree, maximum degree, and the average degree of a graph as

$$\min_{v \in V} d(v), \qquad \max_{v \in V} d(v), \qquad \frac{1}{|V|} \sum_{v \in V} d(v) \tag{1.3}$$

We do that for all of our cities in $\mathscr{D}$ and visualize the distribution using boxplot. In a boxplot the values are segmented the into boxs that gives more information on the distribution such as average, lower or upper quartile. All graphs have the same minimum degree of one. This is normal because our graphs have no isolated node. So each vertex of $G \in \mathscr{D}$ has at least one neighbor. In Figure 1.5,

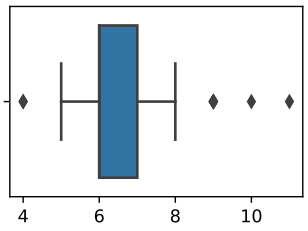


Figure 1.5: Boxplot of the maximum degree of each graph

the maximum degree found in $\mathscr{D}$ is 11 whereas one of the graphs have a maximum degree of 4. These results seem logical. An intersection connected to 11 roads is possible. By curiosity I found out that the city with the degree 11 node is Paris. These intersections are not usual in the city. We could imagine that this intersection be the "Square of the star" of Place Charles de Gaulle. It is large road junction a the meeting point of twelve straight avenues, including Champs-Élysées. Aside from that, a degree 4 node correspond to a four-way intersection, pretty common for a city. So our graphs have nodes with a degree ranging from 1 to 8. According to Figure 1.6, most cities has roughly an average degree between 2 and 4. The average of the average degree values is roughly ~ 2.7. A city is composed of intersection. We can also imagine that the long curved lines (of consecutive degree two nodes) lower the average degree value. For having more insight, we can also compute the density of each graph.

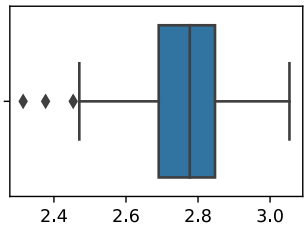


Figure 1.6: Boxplot of the average degree of each graph

The graph density of a simple graph is defined to be the ratio of the number of edges with respect to the maximum possible edges. For undirected graph we compute it by $\frac{2|E|}{|V|(|V|-1)}$. In the highest connected graph, a complete graph, where all the nodes are connected to each others, the density is equal to 1. We compute the density for each graph and plot all the densities in Figure 1.7. The

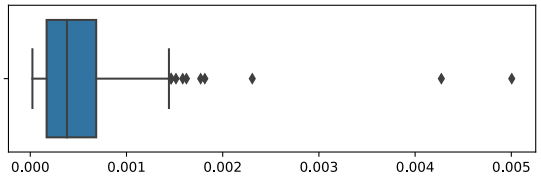


Figure 1.7: Boxplot of the density of each graph

densities are inferior to 0.005 meaning that the graphs are sparse and definitely not dense. The number of edges out-stand the number of edges. It isn't a surprise that we get sparse graph since their degrees values are low. These indications of low density could matche that our graphs have a great possibility to be planar, like cities. That matches the intuition of our graphs. They are composed of intersections and also long curved roads of degree two nodes. The code to reproduce these results is available in my github repository[7].

[7] https://github.com/endingalaporte/Stats-of-cities-road-graphs/blob/main/src/_gt%20stats.ipynb

Chapter 2

State of the art

Considering the length of my internship, it is very ambitious to say this section constitutes a sum up of the state of the art of the methods used in literature to describe the topology of a graph : graph embedding, graph coarsening for graph embedding, graph kernels, feature extraction using convolution neural networks, graph convolutional networks etc. That's why I thank my supervisor Fabien for giving me directions as in what articles read, what documentation to search and especially if the methods I want to use are interesting regarding what's have been done priorly in the literature. During these 6 months, I intensively read many articles and topics in order to get a strong insight of commonly used methods to describe the topological information of graph and cluster them. This section, even if named state of the art, is rather named like this for the scholar purpose of this report. It does not pretend to be a complete one. This section is roughly a summarizing of all the articles I read about the subject, what I understood of them and in which way they are usefull for solving our problematic. Each of the subjects I talk about is worthy of deep technical details, that I don't pretend to fully understand. Here, we propose an insight and a simple description of what I understood from the diverse methods used in literature. For more in depth state of the art on the subjects. I refer the reader of this report to papers of specialized laboratories that have a state of art much more exhaustive and representative then the one here.

2.1 Cities topology linked with to its socio-cultural-economical indicators

Around the world, street network have similar patterns. They can be clustered using their topological similarity. The topology of a city can be linked to its socio-cultural-economical characteristics Boeing [2021]. Worth mentioning that Geoff Boeing is from the author of the framework OSMnx we used to extract our cities. Other statistical indicators are computed in Boeing [2019] such as the orientation of a city, accross 100 cities around the world. His work helps us to define our dataset, which cities are we going to consider in our dataset in order to cluster them.

2.2 Challenge of machine learning from graph

The study of graphs is a wide subject and a dynamic prolific field in research, in various fields such as computer science, artificial intelligence, Medicine, Social studies, Urban modelling etc. Here, we center the research of what's being done, purposely in the goal of solving our main problematic that is to cluster cities according to their topological similarities.

Machine learning techniques from the family of unsupervised and supervised learning permit

to treat massive amounts of data, to spot the similarities between them. The algorithms require the data to be express as an object belonging to an euclidean space. This object is usually a vector. It can be a matrix (popular case of pictures) or higher dimensional objects. Yet, graphs are mathematical objects not belonging to an euclidean space. The mathematical nature by which they are defined raises a challenge from their modelling's standpoint. They require a preprocessing before we can *feed* it to a machine learning pipeline. This challenge is mostly roughly caused of the fact that, we can permute nodes, edge names, while still conserving the relational structure of the graph. Whereas, an euclidean object is defined by its fixed organized set elements. That can be for example a vector of integers, a matrix where the elements forms the picture of a face, or the wavelength of a sound in function of time in a timeserie. From, here we talk about two families of methods. The first methods are those who take a graph directly as a graph object and learns the structure from it. The second methods are those who compute some graph invariant indicators and put them in an euclidean object (vector, matrix, tensors ..etc) so that it can be fed to the machine learning pipeline.

2.3 Graph convolutional neural networks

We first talk about the first method in the following section, that takes a graph directly as input. A graph convolutional neural network (refer to section 2.9 for reminders on convolutional neural networks or CNN) or GCN is a neural network that takes a graph object as input. It is a function that takes a graph as input and output labels.

In Kipf und Welling [2016], the GCN takes a graph as input, with partially labeled nodes (some nodes are labeled while others are not labeled), by which we mean the laplacian matrix $L = A - D$ (see section 2.5) and the labels of each node, roughly with some other algebraic manipulations. From which, it can deduce the label of unlabeled nodes from the labeled nodes. To put it simply, using the intuition of CNN. It is like a CNN but taking a modified laplacian matrix L. This method works relatively well for semi supervised classification of nodes. Since it was cited more then 9139 times, it is a major piece of work in learning from graph. Even if the paper falls outside of our project (node classification while we aim to find graph clustering technics), I included this paper to get an insight of how structural information is extracted from the graph. In that case, it is by the laplacian matrix L.

The paper Zhang u. a. [2018] offers a way to classify graph, directly from their graph form. It presents a supervised learning technic where a specific Deep learning model they call Deep Graph Convolutional Neural Network is used (specific architecture of CNN, we will not talk about technical details of Deep Learning neural networks here, please refer to specific papers). It takes the adjacency matrix A, the degree matrix D, and the node labels as input. To tackle the issue of the fact that the adjacency matrix is not an invariant indicator, they define a new method that sort the nodes a consistent order. Then, since adjacency matrix depends on the graph's size. Their paper solve the issue by using a method so that all the graphs are embeded in the same size input (a matrix). Then, perform the traditional CNN routines, as in GCN until the dense classification layers. To cite the paper, "graph convolution layers extract vertices' local substructure features and definea consistent vertex ordering; 2) a SortPooling layer sorts thevertex features under the previously defined order and unifies input sizes" Zhang u. a. [2018].

2.4 Graph to euclidean space

From here, we're going to describe the second methods, that first transform a graph into an euclidean object such as vector, matrix. There exists already a wide range of graph indicators on the local scale of a node such as degree, betweenness centrality, clustering coefficients etc. Or, in the scale of the graph, the average degree in the graph (since it is a mean of graph invariants, it is itself a

graph invariant, pretty intuitive), diameter, density (as in section 1.3.4. As other graph scale graph invariants, there is the popular histograms of local graph invariant indicators, degree distribution, between centrality distribution. However these *routine* graph theory indicators are too weak to be used alone to describe sufficiently our cities graphs topology.

A challenge exist in literature of how we can define the graph invariant indicators, and seeing if they are reliable on describing the graph's topology. There is also the major issue of making the computing of these indicators possible in a polynomial time, so that the method scales up for bigger graphs and bigger datasets.

2.5 Graph embedding

There are two types of graph embedding. The first type is to project each node in the graph, to a surface (2d euclidean space), called node embedding. The second type of graph embedding is to project a graph, into a euclidean object, such as a vector, a matrix etc.

2.5.1 Node embedding

We first describe the first type.From a graph, build a set of points, where each point represent a node. The points are placed, *embeded* on a surface according to where they are in the graph. Usually, the distance between the points in the surface is analogous to the shortest paths between the nodes in the original graph. However, most of the relational information of the edges are lost. We can suppose, in the embedding that neighbor nodes are linked together. However there is no information of if two distant nodes are linked together or not in the embeding.

2.5.2 Graph embedding

Then, for the second type of graph embedding. Usually, the graph is transformed into a vector, a matrix. And the relational information of the edges is conserved. While in node embedding, this information is destroyed. In node embedding, the relative location of the nodes in the graph matters, but not to which node the node is connected matters. Back to graph embedding, it is basically a way to desribe a graph's structure using graph invariant indicators. A kernel is a type of graph 1D-embedding where each graph projected in $\mathscr{H}$ euclidean space. But the differences here with graph embedding is that it is a general characterisation of all methods to represent graph using euclidean objects. Let's take the example of a matrix, as analogously to representing an image. In a matrix, if the graph is embedded in a matrix (using the specific method), the local elements, and how they are arranged relatively to each other matter, and this arrangement describes localy the topology of the graph. In this case, a CNN could be a good tool to learn from it (the pooling layer takes advantage of learning local feature hierarhely). While a kernel is simply a list of graph invariant indicators that are independ to each others, graph embedding takes into account the local arrangement of elements in the euclidean object. I think that's the key difference between kernels and graph embedding (more the 1D, since kernel is a 1D embedding tehnic) technics that makes it relevant to use CNN to lar from them.

Back to graph embedding. For example, in spectral graph theory, we can represent the graph a matrix called Laplacian of the graph $L = A - D$, where A is the adjacency matix and D its degree matrix. L relies heavily on node numerotation. L embedding is an object used for finding partitions in the graph, but also as an object for graph classification using Graph convolutional networks (GCN). However, due to its definition, since A is not invariant, thus L is not invariant. That makes it a poor choice as an object to represent our cities.

(a) Graph G

(b) Coarse graph G_c

Figure 2.1: Illustration of the coarsening process. Each contraction set, underlined in grey area, is contracted into one node. We observe as well the process for nodes that are not contracted. They are in a single node contraction set and is contracted to itself, remaining the same.

2.6 Graph coarsening for graph representation

Graph coarsening is the process to reduce a graph by grouping (also called contracting, combining, collapsing) groups of nodes together in the goal of simplification of the graph, while making a compromise on conserving the graph strucuture. The implies to choose which nodes to combine together. This paper Graph Reduction with Spectral and Cut Guarantees Loukas [2019] offers a range of coarsening methods based on the manipulation of the laplacian matrix. As well, it describes a mathematical formulation of the coarsening process that I got my inspiration from for writing section 3.2. The paper describes the coarsening process. We're going to describe roughly the coarsening process. For more a more formal description, I refer the reader to the paper, or to the section 3.2. The tricky procedure is that, given a graph we want to contract, once we identified which nodes we want to combine together (called contracted set in the paper), we can begin to construct the coarsen graph. As for the node that are not combined. Each of them is placed in a single node contraction set (the contraction set of one node is combined to one node, thus remaining the same in the coarsened graph). The coarsen graph is composed of vertices of the previous graph, except that each contract set becomes a node, see Figure 2.1. Now that we have the vertices of the coarsen graph notion is the paper is which nodes of the coarsen graph we take to form edges. The paper defines a function φ that maps each contraction set's node, to its respective node in the coarsen graph. We remind that at this point, the graph is entirely partitioned, each node is part of a contraction set. That means that each node, if at the border of the contraction set (interchangeably, partition), it will have a neighbor part of another contraction set. This brings us to another notion defined in the paper that is border nodes. A border node is a node part of a partition that has at least one of its neighbor part of another partition. Now that we have defined the notions of contraction set, φ and a border node, we can construct the edges of the coarsen graph.

I didn't find the formulation of how edges are constructed. However I could reproduce by hand the coarsening with a rule for forming the edges, I got my inspiration from the paper. So the rule for forming edges in the coarsen graph is roughly to connect the contracted node (from previous contracted set) to another node from contracted node (from previous contracted set, even if it was a single node contraction set). The rule is that for each pair of node, check if they are neighbors (mutually border nodes), if they are part of different contraction set. If so, then create an edge between them. The compact description of this process is in section 3.2.

This paper describes the coarsening process but then, not a way to capture the information during the coarsening, nor how to embed the graph into a final representation of it.

The next paper, I draw most of my inspiration from is Hiearchical Stochastic Graphlet Embedding Dutta u. a. [2020]. In this paper, Dutta et Al desribe a method to embed a graph into a representation

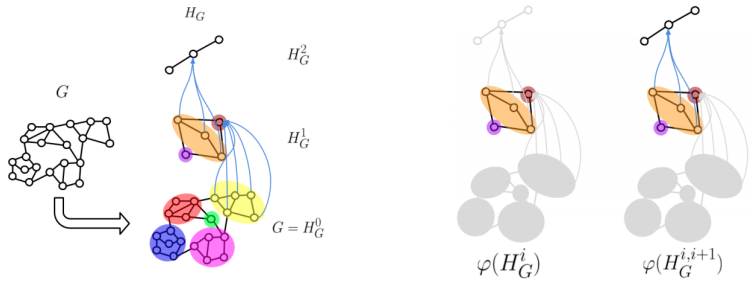


Figure 2.2: Illustration of the coarsening method, the construction of hierarchical graphs, image courtesy of the paper. We observe the partitioning of the graph into contraction sets in (a), and coarsened into a derived set of graphs. In (b), we see how the hierarhical graphs are constructed, and the notation of the graph of each level

.

(a long vector $\Phi(G)$) using coarsening. The process is roughly to use a function ϕ from Stochastic Graphlet Embedding Dutta und Sahbi [2018] to embed a graph into a vector of graphlet counts of different order (unlimited order they say in their paper). To avoid a computational explosion, their method is not counting all the graphlets but rather sampling graphlet graphlet of various orders.

In Dutta u. a. [2020], they don't directly embed the original graph using ϕ, because there is a loss of information of the topology. What they propose is to coarsen the orignal graph many times until one node remains. Each of the coarsening from of level to the next level graph is made using a partitioning algorithm based on betweenness centrality, the Girvan-Newmann algorithm. This wikipedia page offers a nice sum up of how it works[1]. How it works is that it computes the betweenness centrality of each edge, then breaks the edge, and repeat that process, until the graph is *broken* into two components. Do that until there is a certain number of components. Each of these components are the contraction set. In the illustration of Figure 2.2, we see each of the nodes in the same color zone, belonging to the same contraction set. After contraction each of the nodes, into one, we coarsen the graphs until one graph of one node remains. Then, we have a set of graphs derived from the original one. That's where it becomes a bit complicated. Additionally to the coarsen graphs, a new set of graphs is created, linking each contraction set of the previous graph, to the contracted node of the coarsen graph, in each level. These graphs that link the nodes of two different graphs between two consecutive levels are called Hierarchical graph and noted $H_G^{l_1,l_2}$ between level l_1 and l_2 in the paper. The process is illustrated in the paper, here's the Figure 2.2 of their paper. Then, after constructing all these graphs, hierarchical graphs (we call them derived graphs) from the original graph G. The paper describe a process where each of the derived graphs is passed through the Stochastic Graphlet Embedding function, and that makes the final embedding of the graph, a vector $\Phi(G)$ as expressed in the paper

$$\Phi(H_G) = [\varphi(H_G^0), \ldots, \varphi(H_G^K), \phi_1^1(H_G), \ldots, \phi_1^{k_1}(H_G), \phi_2^1(H_G), \ldots, \phi_2^{k_2}(H_G)] \tag{2.1}$$

$$\phi_1^k(H_G) = [\varphi(H_G^{0,k}, \ldots, \varphi_G^{K-k,K}] \tag{2.2}$$

$$\phi_2^k(H_G) = [\varphi(H_G^0 \cup \cdots \cup H_G^k), \ldots, \varphi(H_G^{K-k} \cup \cdots \cup H_G^K)] \tag{2.3}$$

Where $H_G^i \cup H_G^j$ is the union of the two graphs. It is basically the merging of the two graphs H_G^i and H_G^j. This graph manipulation operation is well defined in the literature so we choose to not explain here for a more concision.

[1] https://en.wikipedia.org/wiki/Girvan\OT1\textendashNewman_algorithm

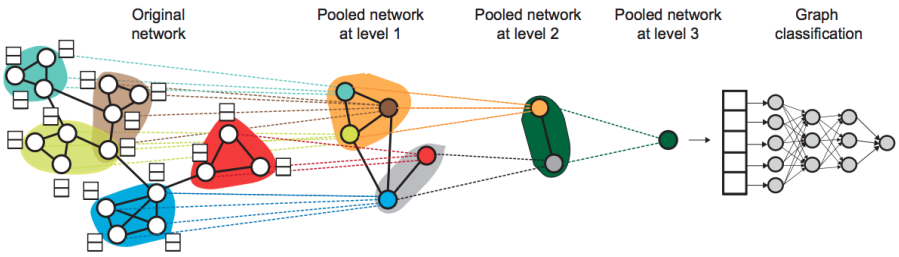


Figure 2.3: Illustration taken from their paper of the embedding processing using combination of nodes, in each hierarchical level, using graph convolutional network, in order to obtain a final vector representing the graph

The reason why I choosed to show Eq. 2.1 here is that because our coarsening embedding function ϕ_{embed} Eq. 3.11 is hugely inspired by them. Our embedding takes a smilar form, a list of graphlet counts. And our final representation of our propose graphlet coarsening technics uses the same process of grahlet counts of each of the derived graphs in the end in order to embed our graph. Our coarsening embedding method is hugely inspired from them. In section 3.2, we use a different partition method then Girvan–Newman, and instead introduced a method where partitions (contraction sets) are constructed with nodes that have a similar topological local environnement, described by their graphlet degree vector (see our section 3.2.5). Then, we use a similar function from Stochastic Graphlet Embedding, except that we take into advantage the library graph-tool. We count the $2, 3, 4, 5$ nodes graphlets (section 3.2).

In Ying u. a. [2018], Ying et Al in their paper Hierarchical Graph Representation Learning with Differentiable Pooling propose a method called DIFFPOOL to transform a graph into a vector using convolutional neural networks GCN to compress the hierarchical representation of the graph. The process is fairly complex, I didn't understand it all. Nevertheless, it permits to embed graph, while retaining its structural information, into a vector, that can be fed in a machine learning pipeline. Illustration of the DIFFPOOL process to convert graph to a vector in Figure 2.3. The model in the paper is tested in the situation of graph classification. The github page offers a great explanation of how DIFFPOOL works[2].

2.7 Graph kernels

Graph kernel is the kernel method applied to graph objects. The formulation of it is detailed in section 3.3. The idea is to determine a vector ϕ that represent a graph, for each graph. That vector is composed of invariant indicators. One of the ideas to capture the topology of a graph in various fields is to count small graph motifs, called graphlets in it (see section 3.2.2). Then, we basically compute the inner product between all vectors ϕ and their distance and obtain two matrices that we can feed to many kernelized machine learning techniques for cluster identification, dimentionality reduction, classification such as support vector machines, k-means, principal component analysis, t-distributed stochastic neighbor embedding, hierarchical agglomerative clustering dendrogram, to cite the most popular ones.

There are various paper that rely on the count of graphlets to construct a kernel and analyse their graphs with it.

As well, another technique, to build kernels, is to compute the graphlet degree vector for each node, then to build a kernel out of it Milenković und Pržulj [2008]. In Pržulj [2007], the orbits degree vector of 3,4,5 node graphlets are counted. Then, a histogram of each of the graphlet degree

[2] https://asail.gitbook.io/hogwarts/graph/diffpool

is created, and a graph is described by its list of graphlet degree histograms. They used it for comparing cellular networks. Counting graphlet degree vectors is not a trivial task. It is a complex problem where the time of computation (often too great) is an issue. We cite here the two algorithms used in this report for graphlet counting. In the next section, we talk about other graphlet counting technics found in literature. Algorithms that counts graphlets and graphlet degree vectors efficiently are a challenge. For counting graphlet degree vector of 3,4 nodes graphlets for each edge efficiently in parallel, the research team of Intel wrote Ahmed u. a. [2016]. I coded that algorithm 2 entirely for the section 3.3.2. The algorithm 2 Ahmed u. a. [2016] needs T and S_v that are not defined in the paper. But, they are defined in Rossi und Zhou [2016]. So I only used the second paper for finding a formal definition of how the sets S and T are defined. The code of the graphlet degree counting algorithm is available in my github [3]. For counting 3,4,5 node graphlets in a graph, Shervashidze u. a. [2009].

Here, since we concentrate on graph characterization using graphlets, we won't talk about other kernels. There exist other graph kernels such as shortest path kernel, Weisfeiler-Lehman kernel. In Wale u. a. [2008], each molecule is described using ϕ, called *descriptor*. The molecules, represented by their ϕ are the classified using support vector machine.

2.8 Graphlet counting

In Liu u. a. [2020], Liu et Al use Graph Convolutional Network on a set of labeled graphs. The labels are the Graphlet counts (Graphlet frequency distribution GFD), taking a graph as input. Their model make use of deep learning technic to predict the number of graphlets in a graph, without the computational expensive task of graphlet counting using naive, combinatorial, or sampling methods.

Until now, we saw methods to count graphlets in a graph, from scratch. I had the intuition that our coarsening algorithm will have to change the graph structure locally, hence, making the graphlet count change, and have to count them all over again. Cannoodt u. a. [2018] propose a method of counting graphlet in a graph that locally changed. Their method permits to obtain the new count, after the local change, without counting the graphlet all over the graph again.

Ahmed u. a. [2015] propose an algorithm, we can run in parralel, to count 3,4 node graphlets. They also implemented their code in C++ [4], very fast. The year after, they publish Ahmed u. a. [2016] to count the graphlet degree vector for each edge. When digging into the algorithms of the two papers, the equations from which the graphlets are get (combinatorial approach) are relatively similar.

Even if I didn't use these papers' code or methods of counts, they were very usefull for me to see what are the common notations for graphlets, graph manipulation. For example by which mathematical expression or Greek letter denote the set of neighbors, degree is commonly expressed. This was very useful for the creation of the coarsening method defined in section 3.2.

2.9 Which features the model learns from

During the internship, there was the idea to visualize the features of a city to which correspond a label (label can interchangeably be called class) by using Class Activation Mapping. That way, we can see why a graph belongs to a certain class, let's say of poor countries. Let's say we have our graph expressed as a matrix (as we did in section 3.2 or 3.3.2) We would see which of the elements in the matrix are highlighted. Let's say this element in the matrix, after further study, reveal a sparse organization of the city (access to each point in the city is hard). That way, we would

[3] https://github.com/endingalaporte/alg2-Exact-and-Estimation-of-Local-Edge-centric-Graphlet-Counts/blob/main/_graph_tool%20LOCALGRAPHLET.ipynb

[4] https://github.com/nkahmed/PGD

Figure 2.4: Visualization of the pixels (element of the matrix) using a palette of red-yellow range colors, the pixels that contributes the most to the activation of the label, name of the animal elephant, using Class Activation Mapping method

demonstrate empiracally from our data that sparse cities, or cities organized in a *bad* manner can lead to lower GDP. That would be the best goal. Let's describe the process of how we wil use Class Activation Mapping. We have our graphs expressed as a vector, matrix let's say, there is the idea we had to take a dataset of graph cities, and their labels such as Gross Domestic Product (GDP), or population. Then, to train a CNN. Then, to visualize the Class Activation Mapping using a library such as Keras, Tensorflow[5]. First, let's remind what is a convolutional neural network. A convolutional neural network (CNN) is a supervised learning method where the model *learns* to associate a datapoints matrices to its given label. It is a estimator function that given datapoint, returns its classification label. We will not explain the specific process by which it operates here and refer the reader to specificpapers. How it roughly works is that the CNN is composed of hierarchically imbricated individual functions (called neurons) that activate or not (value ranging from 0 to 1) accordingly to the neurons of previous layer. Weights are associate in the connection of each neurons so that each neuron contributes more or less to the activation of the neurons from next layer. The training process can be seen as an optimization problem where the model (CNN) weights are free to change, in order to fit the dataset matrices and labels. Considering the error between predictions of the model and the ground truth labels, during the training process, weights of each neuron of the model are updated so that the model parameters fits the labeled data. Each neuron weight's value is updated in order to minimize the error between the predicted labels and the ground truth labels. Each neuron weight is updated, by being added a small correction value, expressed in a relation combining the error variation according the the weight's variation (it's a partial derivative). The method is called gradient descent. For further details we refer the reader on documentation regarding backpropagation. This type of neural network has the specificity in the arrangement of the connectivity of each of its neurons of a layer, relatively to the previous layer. As well, two types of layers called filter and pooling layers, as opposed to multilayer perceptron that is composed only of fully connected layers compose the CNN. Class Activation Mapping is basically a process to highlight which part of the datapoint (matrix) contributes the most to the categorisation of it as a certain label (certain class). For more intuition visualization of the process, we see in Figure 2.4.

[5] https://keras.io/examples/vision/grad_cam/

Chapter 3

Graph to euclidean spaces

3.1 Two approaches

The first method is to find a way to capture the topological information of a graph using a coarsening technic based on graphlet. Once done, put all this in a mathematical object. Then, use a method of clustering of these mathematical objects. The second method is to build kernels that capture the topological information of each graph. Then, use of the wide range of unsupervised learning techniques for clustering.

3.2 FOLD : a graph embedding method using graphlet coarsening

3.2.1 Intuition

We inspire our method from Dutta u. a. [2020] and use notations and concepts from Loukas [2019] to formulate our graphlet coarsening method, see section 2.6. We call this method FOLD, subjectively inspired from the poetic analogy of folding a paper into itself while retaining structural information.In our method, instead of using the Girvan-Newman algorithm to partition our graph. We use our own method based on grouping nodes that have same local topological environnement, characterized by graphlet degree vector. As well, instead of using their Stochastic Graphlet Embedding function that embeds a graph into a vector of counts of graphlets of various orders, we use our own embedding function $\phi_{\mathscr{E}}$ defined in section 3.2.3. We also add the capture of one additional information in the coarsening that is which nodes environments have been coarsened and what portion of the graph of this environment have been coarsened. The intuition of this method we call "FOLD", is like having an original graph, we see which of the nodes lies in a similar local topological environment (let's say a lattice grid graph, in the case of the streets of New York). Then we combine all the nodes belonging to the redundant topological space. While doing so, we retain the information of which types of nodes have been combined together (using in section 3.2.5) and how much of the graph have been coarsened. The topological information would be collected in each contraction. That makes one coarsening from original graph at level l to level $(l+1)$. We repeat the same process until the graph can not be compressed any more (arrived at level $(c+1)$-th of the coarsened graph), that is all the nodes lies in a different topological space, the optimal compressed representation of the original graph. We have then a set of graph, we pass them through our embedding function $\phi_{\mathscr{E}}$ as described in the end of section 3.2.3. We have our graph embedded in a vector, while retaining the structural information of it.

3.2.2 Graphlet

A Graphlet is a specific graph identified and named. They can be seen by families of k nodes. For example, for $k = 5$ it is possible to generate 21 different graphlets, see Figure 3.1. In the literature, graphlets of 2,3,4 and 5 nodes are named commonly $g_0, g_1, \ldots, g_{29}$. Since our cities belong to χ, we will talk in this section only about graphlet of one component $|C| = 1$, with no isolated nodes, all connected, undirected, which by themselves also belong to χ Khorshidi u. a. [2020].

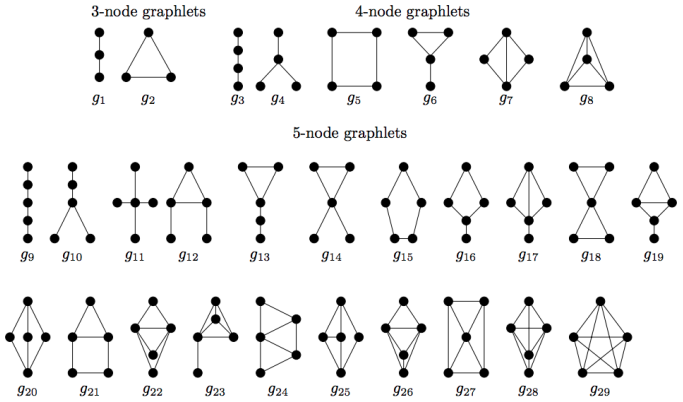


Figure 3.1: Names of the 29 undirected graphlets of 3, 4 and 5 nodes

Global structure

Here, the idea is that graphets g_i are like building blocks with which we can build any graph $G \in \chi$. A common way to describe a graph using an object of euclidean space (vector, matrix ...) is to count the number of graphlet g_i that G has. Graphlets are used many times in literature to describe a graph global structure.

Nevertheless, this method doesn't consider the information of how the graphlet are built upon each others, how they are connected is not taken into account. Even if graphlet enumerating methods have success for differentiating relative graph between each others, and classifying them, we think there is still some topological information that can be extracted.

Euclidean constraints on a graph space

We formulate here the graph representation vector problem as the search of invariant indicators (see section 1.2.3), that constraint the space of possible graphs defined that respect these constraints. To put it formally, let $\phi = (n(g_0), n(g_1), \ldots, n(g_{29}))$ be the number of 2,3,4,5 graphlets in $G \in \chi$. $G = (V, E)$ is a *huge* graph of more then 1000 edges (case of our cities). Let's say that our problem is that we want to do the reverse process. Given ϕ, we want to rebuild the graph G. If we try to enumerate all graphs that have the count ϕ. The Let χ_s be the finite space (space of solutions) of graph that respect the condition ϕ, χ_s will contain more then one graph. $|\chi_s| > 1$. In other words, to one ϕ doesn't correspond necessarily only a G.

Practically, this isn't an issue because in the literature, the goal of graph classification is to find enough characteristics to differentiate graphs relatively between them. So their classification methods still have pretty good results.

We can see ϕ as a list of constraints, that constrains the space of graphs solutions χ_s satisfying the constraints. In the following sections, we're going to find constraints ϕ' that reduce the size of the space $|\chi_s|$. Ideally we can think of our intermediary step to search for constraints ϕ' of $G \in \chi$

that reduces the space of χ_s to $|\chi_s| = 1$. In that case, one graph G would correspond to one ϕ'. It would be a bijective application of χ to an euclidean space. Let $\mathscr{F}$ be the space of functions that compute invariant indicators of graphs. We can formulate the problem as constructing a ϕ of the possible functions in space $\mathscr{F}$ in the goal of minimizing the solution space χ_s of graphs G that respect the constraints ϕ. Compactly, this writes beautifully like an optimization problem as

$$\min_{\phi(G)\in\mathscr{F}} |\chi_s| \tag{3.1}$$

while trying to make ϕ not too long to compute for each of our cities. In this report, the constraints we add will be based on graphlet degree distribution (kernels), a convergent coarsening technic based on graphlet degree vectors (graphlet coarsening). In this report, we're going to describe a method called *graphlet coarsening* that goes in that direction of minimizing $|\chi_s|$.

In this report, we will construct ϕ_{degree}, $\phi_{3,4,5-graphlet}$, ϕ_{orbit}, ϕ_{orbit_hist} and $\phi_{\mathscr{E}}$ that describe increasingly more structure of the the graph G. If we denote by $\chi_s(\phi)$ the space of graphs that respect and it constrained by ϕ as in Eq. 3.2

$$|\chi_s(\phi_{degree})| > |\chi_s(\phi_{graphlet})| > |\chi_s(\phi_{orbit})| > |\phi_{graphletdegree}| > |\chi_s(\phi_{orbit\_degree})| \gg |\chi_s(\phi_{\mathscr{E}})| \tag{3.2}$$

Local structure

Graphlet are also used extensively, in the literature Milenković und Pržulj [2008] and Milenković u. a. [2010][1], to describe the surrounding local topology of a node $v \in V$. A commonly used method is to count the number of times a node v touches a graphlet g_0, the graphlet $g_1, \ldots, g_{29}$. That associate a vector of counts that is called graphlet degree vector, or graphlet degree signature, or simply signature. As we saw in the section 1.3.4 that our cities have at most nodes of degree 7, and on average nodes of degree 2.7, we're going to consider graphlets of order $k = 5$ at most.

3.2.3 Formulation

We're going to define the method we use to transform a graph in a vector space using a coarsening technique based on graphlet. First we're going to explain the main idea. So we have a graph $G = (V, E)$. Let's consider a function $\mathscr{T}$ called *VertexPropertyMap* that given a graph, returns a matrix W (that is the vertex property map of that graph) where each row W^j is an array associate to each node v of G.

$$\begin{aligned} \mathscr{T} : \chi &\to \mathscr{M}_{|V|,s}(\mathbb{R}) \\ G &\mapsto W \end{aligned}$$

In our case, VertexPropertyMap function is the graphlet degree vector counting algorithm (of orbits) for each node $v \in V$ called "OrbitalStrike"[2]. We have a matrix of $\mathscr{M}_{|V|,s}(\mathbb{N})$ where each row contains the number of times the node v belongs to the i-th orbit. That way, since we can count 67 orbits, $s = 67$. If we count only if the node corresponding to one of the 29 graphlets, then $s = 29$. If we choose to associate the degree of a node $d(v)$ to the node, then $s = 1$. And each i-th component of the row corresponds to the orbit count for the i-th orbit. For each j-th node $v \in V$, we associate a vector $W^j \in \mathbb{R}^s$ and that vector is equal to the j-th row of the previous matrix containing the counts. That way, each node has a vector associated to it that is its respective orbit counts.

$$W = \{W^j\}_{j\in[0,1,\ldots,|V|-1]} \tag{3.3}$$

[1] cited approximately 400 and 200 times

[2] https://github.com/benedekrozemberczki/OrbitalFeatures

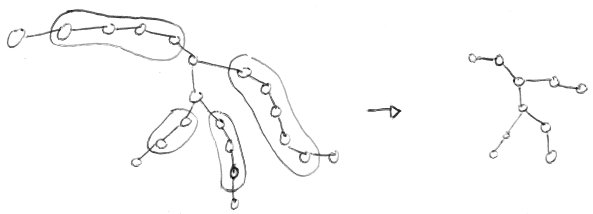


Figure 3.2: A graph of containing multiple redundant g_0 graphlets, each group of nodes forms a contraction set, each group of nodes is combined into one node, that forms the coarsened graph

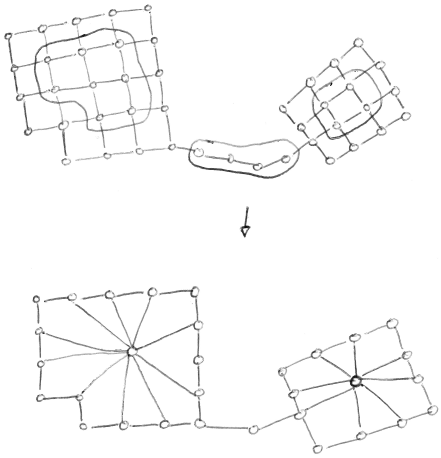


Figure 3.3: A graph of containing multiple redundant g_0 "rectangles" graphlets

and $|V| = |W|$. We called this vector W^j graphlet degree vector of the $j-th$ node v. In the next section we will also call W^j the *graphlet environment*. We will call the matrix W the vertex property map for two reasons. One is that the name is pretty much self explanatory. The second reasons is that the graph-tool library uses the same name the matrix W that associate a row vector of properties[3] (in our case a vector of numbers counts) to each node v.

Since W^j contains the information of the graphlets (orbits) to which the node belongs, it is a great measure of the local topological information of the node. It describes well the local street structure around the node as said in Milenković und Pržulj [2008]. We propose here to combine the vertex that have a similar Let's consider a city's graph. In a city's graph there are some redundant graphlet patterns. For example, long curved roads are defined as sequence of let's say $m \in \mathbb{N}$ times the g_0 graphlets (simple edge) put the one after another, thus that are neighbor to each others as in Figure 3.2. The idea would be to combine all the nodes of that long curved road into one g_0 and collect an information such as there is long curved road that is defined by m graphlets of type g_0. Another redundant topological structure in city's graphs are grids. A grid is composed of g_5 graphlets. Let's say in a city that we have streets organized in a grid manner, that is essentially let's say $m \cdot n$ times the g_5 graphlets assembled one with another to form a rectangle district of shape $m \cdot n$ as in Figure 3.3. The idea here would be the combine the nodes in the grid neighbor together. And to say that the neighbor is composed of a rectangle of $m \cdot n$ times g_5 graphlets. That way, we hope that the district's topology would be composed of non redundant structure that would be the core of the topology of the city. Let $G = (V, E)$ be a graph with its respective vertices weights W^j

[3] https://graph-tool.skewed.de/static/doc/graph_tool.html#graph_tool.VertexPropertyMap

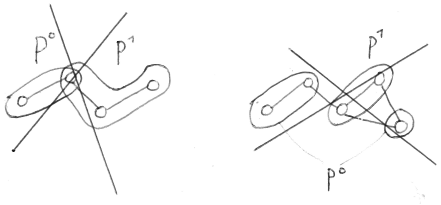


Figure 3.4: Contraction sets are disjoint sets and their union makes the set of vertices V

for each $j-th$ node so that $|W| = |V|$. Let P be a partition of graph G. A partition P of a graph is a set of group of nodes P^i. Each group of nodes is called a contraction set P^i. P^i contains at least one node, and at most $|V|$ nodes. P^i as Eq. 3.4 A partition is a set of contraction sets P^i as in Eq. 3.5.

$$P^i \in \bigcup_{t=1}^{|V|} (\bigtimes_{k=1}^{t} V) = V \bigcup V \times V \bigcup \ldots \bigcup \underbrace{V \times V \times \cdots \times V}_{|V| \text{ times}} \tag{3.4}$$

$$P = \{P^0, P^1, \ldots, P^r\} = \{\ldots, \{v_a, v_b, \ldots, v_c\}, \ldots, \{v_d\}, \ldots, \{v_e, \ldots, v_f\}, \ldots\} \tag{3.5}$$

The sets constrained by the following rules in order to be a viable partition P. Each of these group of nodes form one component $|C| = 1$ and is called a contraction set P^i as in [Loukas, 2019, p. 6]. That means that an isolated node from the group can't be part of this contracted set P^i. Each node of G can only be part of one contraction set P^i so that Eq. 3.6. And each node must be part of a contraction set, even if the contraction set contains only one node, so that Eq. 3.7.

$$P^i \bigcap P^j = \varnothing \tag{3.6}$$

$$V = \bigcup_{i=0}^{r} P^i \tag{3.7}$$

These two previous rules are illustrated in Figure 3.4. One contraction set contains at least one node and at most all the nodes in the graph. Now that we defined what a partition P of our graph G is, let's define the partitioning function $\mathscr{R}$. Let $\mathscr{P}$ be the space of possible viable, regarding the previous rules we said on P, partitions of for a graph $G \in \chi$. $\mathscr{R}$ takes a graph $G = (V, E)$ and its vertex property map W as input. $\mathscr{R}$ returns a partition $P \in \mathscr{P}$ of G.

$$\begin{aligned} \mathscr{R} \colon \chi \times \mathscr{M}_{|V|,s}(\mathbb{N}) &\to \mathscr{P} \\ (G, W) &\mapsto P \end{aligned}$$

Let $\mathscr{G}$ the function that given a graph G_{l-1} and its partition P_{l-1} of level $(l-1)$, returns the coarsen graph G_l of level l. The coarsen graph G_l is smaller then the graph G_{l-1}, $|E_l| \geqslant |E_{l-1}|$ and $|V_l| \geqslant |V_{l-1}|$.

$$\begin{aligned} \mathscr{G} \colon \chi \times \mathscr{P} &\to \chi \\ (G_{l-1}, P_{l-1}) &\mapsto G_l \end{aligned}$$

Let $\phi(G)$ be the normalized 3,4,5-graphlet counts of G defined as in Eq. 3.8.

$$\phi(G)_e = \frac{1}{\sum_{i=1}^{29} n(g_i)} (n(g_1), n(g_2), \ldots, n(g_{29})) \tag{3.8}$$

Let $r(G, P)$ be the ratio of contracted nodes. It is simply the number of nodes belonging to the contraction sets P^i containing more then one node, divided by the total number of nodes in G, that is $|V|$ as in Eq. 3.9.

$$r(G, P) = \frac{1}{|V|} \sum_{\substack{P^i \in P \\ |P^i|>1}} |P^i| \tag{3.9}$$

r describes how much of the graph has been contracted reduced. The more the graph is coarsened, the more groups of nodes are combined in one node, the closer r is to 1. On the contrary, the less node are grouped together, the less the graph coarsened, and r is closer to 0. $0 < r(G, P) < 1$. Now that we have the information of how much of the graph has been contracted, we want to extract the information of which graphlets have been contracted. We do so by considering the graphlet degree vector W_j of each node $v \in \{w \in P^i \mid |P^i| > 1\}$ belonging to the contracted sets of more then one node. Let $W(v) \in \mathbb{R}^s$ be the vertex property of the vertex v. Let $\phi(W, P)_{contracted} \in \mathbb{R}^s$ be the mean of all graphlet degree vector of of the nodes of the contraction set of more then one node and

$$q = \sum_{\substack{P^i \in P \\ |P^i|>1}} |P^i| \qquad \phi(W, P)_{contracted} = \frac{1}{q} \sum_{\substack{v \in P^i \\ |P^i|>1}} W(v) \tag{3.10}$$

ϕ_{embed} describes the structure of the graph by counts of 3,4,5-graphlets (normalized) ϕ_e. As well, r describes how much of the graph is contracted. Finally, $\phi_{contracted}$ of Eq. 3.11 describes which type of nodes and by extension, which graphlet environnements have been contracted.

$$\phi_{embed}(G, P, W) = (\phi(G)_e, r(G, P), \phi(W, P)_{contracted}) = F \tag{3.11}$$

So, long story short, by having a graph G_{l-1}, we can get the representation ϕ_{embed} of it where $\phi_{embed}(G_{l-1}, P_{l-1}, W_{l-1})$, and the coarsen graph G_l by passing by these functions in the following order. We have a graph G_{l-1}, we get the vertex property map by $\mathscr{T}(G_{l-1}) = W_{l-1}$. Then, we get the partition with $\mathscr{R}(G_{l-1}, W_{l-1}) = P_{l-1}$. With the partition and the graph, we combine the nodes (of each contraction set) of the partition in order to get the coarsen graph with $\mathscr{G}(G_{l-1}, P_{l-1}) = G_l$. We transform the uncoarsen graph, and the informations of how it is coarsened in a vector F using Eq. 3.12.

$$\phi_{embed}(G_{l-1}, P_{l-1}, W_{l-1}) = F \in \mathbb{R}^{29+1+s} \tag{3.12}$$

We decomposed the coarsening process in various functions. Now we're going to simplify notations by creating a last function $\mathscr{A}$ that given a graph G_{l-1}, returns its representation vector $F \in \mathbb{R}^{29+1+s}$ and the coarsen graph G_l. Its output are expressed by imbricated precedently defined coarsening functions in this section like Eq. 3.13.

$$\begin{aligned} \mathscr{A} &: \chi \to \chi \times \mathbb{R}^{29+1+s} \\ G_{l-1} &\mapsto (\mathscr{A}_0(G_{l-1}), \mathscr{A}_1(G_{l-1})) = (G_l, F) \end{aligned}$$

$$(G_l, F) = (\mathscr{G}(G_{l-1}, P_{l-1}), \phi_{embed}(G_{l-1}, \mathscr{R}(G_{l-1}, \mathscr{G}(G_{l-1})), \mathscr{T}(G_{l-1}))) \tag{3.13}$$

The coarsening process can be visualized as in Figure 3.5. We coarsen the graph c times. We obtain a set of $(c + 1)$ graphs contracted from the original $G = G_{l=0}$. Finally, our method is resumed as a function $\phi_{\mathscr{E}}$ that transform a graph in a euclidean space, given a chosen predefined number of coarsening $c \in \mathbb{N}$.

$$\begin{aligned} \phi_{\mathscr{E}} &: \chi \times \mathbb{N} \to \mathscr{M}_{c+1,29+1+s}(\mathbb{R}) \\ (G, c) &\mapsto M \end{aligned}$$

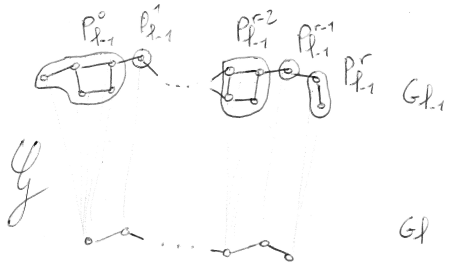


Figure 3.5: Illustration of the coarsening, each contraction set of G_{l-1} becomes one node in the graph G_l

$$M = \begin{bmatrix} \mathscr{A}_1(G_{l=0}) \\ \mathscr{A}_1(G_{l=1}) \\ \vdots \\ \mathscr{A}_1(G_{l=c}) \end{bmatrix} \tag{3.14}$$

M is the represents our original graph G by a set of its contracted decomposition. Here, the number of times G is coarsen is chosen by the user. However, it will be interesting to make the number of coarsening a choice depending on the graph's topology, like a condition to meet so that the coarsening stops. We only suppose here a condition where the graph contraction from level l to $l+1$ is too small so stop the coarsening. That could be a a condition to check like $\frac{|V_{l+1}|}{|V_l|}$

3.2.4 Vertex property map

In this section, we will define which method we used to compute the vertex property map. It is important since it will influence the way which node will be combined together (see that in next section). The goal here is to associate a vector to a node, for each node, that captures well its local topology around its neighborhood.

$$\begin{aligned} \mathscr{T} : \chi &\to \mathscr{M}_{|V|,s}(\mathbb{R}) \\ G &\mapsto W \end{aligned}$$

Vertex GDV from Edge GDV

So first, I found an algorithm described in paper Ahmed u. a. [2016] called LOCALGRPAHLET that can count the graphlet degree vector (GDV) for $s = 8$ graphlets of $2, 3$ and 4 nodes for each edge $e \in E$. I coded this algorithm entirely using numpy and graph-tool. The code of it is available in my github[4]. But the issue is that I couldn't manage to make a coarsening rule using edge GDV. So I thought of getting a kind of GDV from the ones of each edge. The idea will be that each node get the GDV of each edge that it touches, and make an average. $W(v)$ and $W(e)$ is the GDV of node v and edge e. We get the GDV of each node, from the GDV of each edge by

$$W(v) = \frac{1}{|\Gamma(v)|} \sum_{u \in \Gamma(v)} W((v,u)) \tag{3.15}$$

[4] https://github.com/endingalaporte/alg2-Exact-and-Estimation-of-Local-Edge-centric-Graphlet-Counts/blob/main/_graph_tool%20LOCALGRAPHLET.ipynb

Orbit GDV

The second method I found to count graphlets to which a node belongs is a code counting the $s = 67$ orbits. I didn't code any of the algorithm, I just used the made code. The code is an implementation of paper graphlet counting technic Pržulj [2007] by Benedek Rozemberczki from University of Edinburgh using NetworkX[5].

Degree

To each node v, we associate to it the degree of this node $d(v)$ so that the property of this vertex is its degree $W(v) = d(v)$, $s = 1$ and $W^j \in \mathbb{R}$.

3.2.5 Partitioning

The goal of this section is to define the partitioning process. We have the vertex property map $W \in \mathscr{M}_{|V|,s}(\mathbb{N})$. We're going to compare the similarity of the environment of each neighbor pair of nodes. To put it simply, we compute the distance between node u and v of G, where $e = (u, v)$, $\mathscr{B}(e) = \|W(u) - W(v)\|_1$. $\|.\|_1$ is the Manhattan distance. We choose the Manhattan distance because it easier to trace back the variation of the norm from each of its components since they are linearly linked. The Manhattan norm of vector is defined as a sum of all of its components, each raised to its absolute value by $|.|$. We compute the pairwise distance between each pair of nodes from each edge, and we associate each distance $\mathscr{B}(e)$ for each edge. We store these distances in $\mathscr{B}$. $\mathscr{B} = \{\mathscr{B}(e)\}_{e \in E}$. We make an histogram of $\mathscr{B}$ using H_{cv} defined in section 3.3.4. The idea here is to consider the set $S(\mathscr{B})$ of edges e that have a distance $d(e)$ inferior or equal to a threshold value $T_s \in \mathbb{R}^+$, $d(e) \leqslant T_s$. Then, in this set of edges, we're going to construct the derived set of corresponding nodes $v \in S(\mathscr{B})$, we're going to put in the same partitions, the nodes that are neighbors. The number of one component $|C| = 1$ in this set will be the number of partitions P^i.

For example, let's say we want to partition the line graph located in upper Figure 3.7. We see in the histogram of distance values that the graph has 11 edges of distance $\mathscr{B}(e) = 0$ colored in purple and 8 edges of distance $\mathscr{B}(e) = 1$ colored in yellow. By the way, the yellow edges indicates well the changing of the topology of the graph, which is what we want here. Here, we fix the treshold value T_s to be 0. We consider the set of edges $S(\mathscr{B})$, in Figure 3.7 colored in purple. We then consider the nodes $v \in S(\mathscr{B})$. Now, we put in the same partitions the nodes $v \in S(\mathscr{B})$ that are neighbors. That way, that generates 4 partitions P^i, $i \in \{0, 1, 2, 3\}$ of respectively 2, 3, 4 and 5 nodes.

Similarly, in the lattice graph of Figure 3.7, for the same threshold, we get 5 partitions $P^i, i \in \{0, 1, 2, 3, 4\}$ of 3, 3, 5, 5, and 15 nodes. The partition of 15 nodes P^4 is the center part of the lattice graph. We wanted to contract nodes that have similar environment. Well, we observe, in the case of the degree similarity, for these simple graphs, that the nodes can successfully be partitioned together according to the local redundant topology of the graph. This is what we wanted ! In this case, the partitioning is satisfying. We need to code this model and check if the partitioning rule works well. Since we have P^i, we then P. Using the previous rules we just defined, we can partition the graph G into P, using W.

$$\begin{aligned} \mathscr{R} : \chi \times \mathscr{M}_{|V|,s}(\mathbb{N}) &\to \mathscr{P} \\ (G, W) &\mapsto P \end{aligned}$$

We propose to build the threshold in relation to the distribution of edges similarity so that 20% of the nodes that are the most similar. In other words, that means to combine the nodes that have a distance $\mathscr{B}(e)$ inferior to the threshold $\mathscr{B}(e) < T_s$. Let Z be the *similarity ratio* so that $0 \leqslant Z < 1$.

[5] https://github.com/benedekrozemberczki/OrbitalFeatures/blob/master/README.md

And we impose here arbitrary, by intuition that the default value is $Z = 0.2$. This translates as 20% of most similar nodes will be combined together.

$$T_s(Z) = Z|H_{cv}(\mathscr{B})|$$

What it is that it does is that we consider the biggest distance $|H_{cv}(\mathscr{B})|$, multiply it by $0.2 = \frac{1}{4}$. That way, we get the distance threshold $T_s(Z)$ 4 times smaller then the biggest distance. Our partitioning rule based on graphlet environment similarity is now formally build. We describe their construction in the following pseudo-Algorithm 1. We define three actions A_i. A_0 : Put each node v that has neighbors in the same partition. A_1 : Put every remaining node don't respecting the condition in a single node partition. A_2 : Put remaining nodes not in partition in a single node partition. We proposed a partitioning rule, however, it can still be *tuned*. This includes how to choose the

Algorithm 1 Implementation of our mapping $\mathscr{R}$. Given a graph object G and its matrix of vertex property W, returns the partition P.

procedure $\mathscr{R}(G, W)$
  $S(\mathscr{B}) = \{0\}$
  $Z = 0.2$ ▷ Initialize similarity ratio
  $P = \{0\}$ ▷ Initialize an empty partition
  for For each $v \in B$ **do**
    if $\mathscr{B}(e) \leqslant T_s(Z)$ **then**
      for For each $v \in e$ **do**
        $S(\mathscr{B}) \leftarrow S(\mathscr{B}) + \{v\}$
  for For each $v \in V$ **do**
    for For each $v \in S(\mathscr{B})$ **do** ▷ Build partitions P^i of most similar nodes
      A_0
      A_1
    for For each $v \in V$ **do**
      for For each $v \in V \backslash S(\mathscr{B})$ **do** ▷ Put each of the nodes that are not similar in a single node partition
        A_2 ▷ At this point, we have $(r + 1)$ partitions P^i
  for For each $i \in \{0, 1, \ldots, r\}$ **do**
    $P \leftarrow P + \{P^i\}$
  return P ▷ the partition is P

threshold value T_s, to *tune* the expression of $T_s(Z)$ by running more empirical experiences. As it is for now, it can produce partitions. I propose that for a future work, we define formally A_0, A_1, A_2, and work on a relation that express the number of coarsening $(c + 1)$ in function of the histogram of distances distributions $H_{cv}(\{\mathscr{B}(e)\}_{e \in E})$. Because, for now $(c + 1)$ is a parameter we choose base on intuition. It should be based on specific characteristics of the coarsen graph.

orbit GDV

Figure 3.6. Code I wrote for this visualization is available in my github[6].

degree

Figure 3.7. The I wrote for the visualization of the partitioning is available in my github[7].

[6] https://github.com/endingalaporte/Graph-partitioning/blob/main/GDV_orbit/_gt%20gdv%20orbit.ipynb
[7] https://github.com/endingalaporte/Graph-partitioning/blob/main/GDV_degree/_gt%20gdv%20degree.ipynb

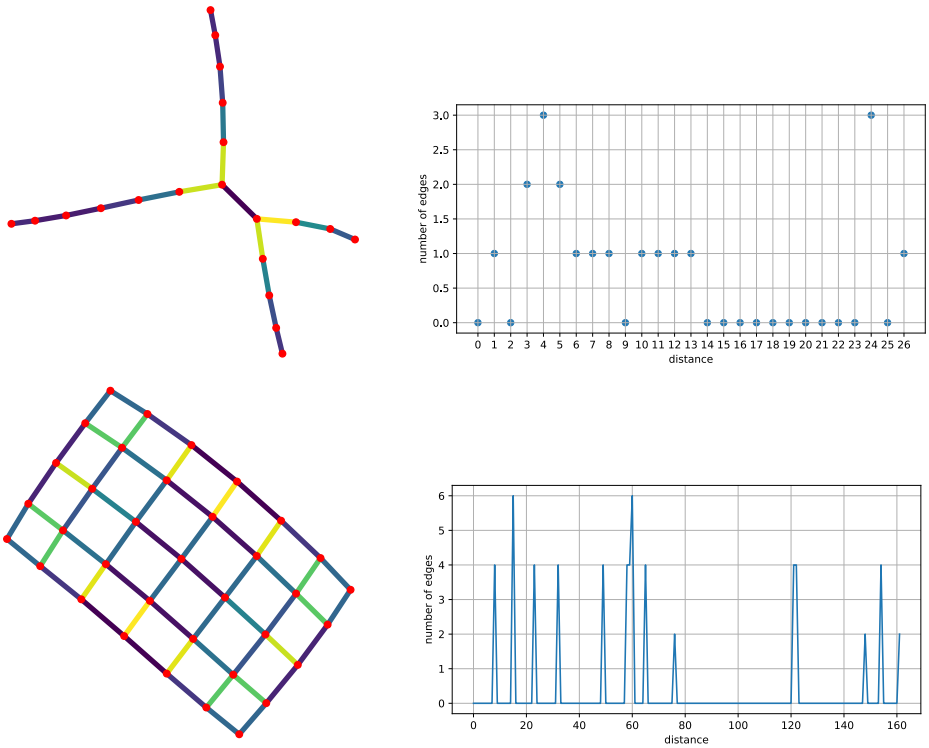


Figure 3.6: Similarity between pair of nodes of each edge according to their orbit GDV

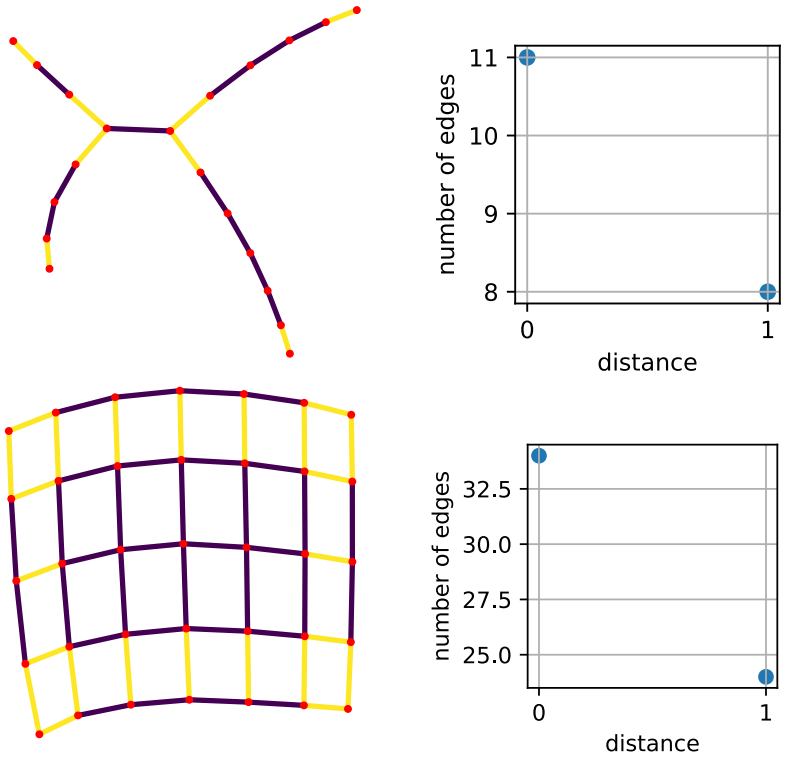


Figure 3.7: Similarity between pair of nodes of each edge according to their degree

3.2.6 Coarsening

Here, we're going to define the function $\mathscr{C}$ of how the coarsening is made from G_{l-1} to G_l. Let $G_l = (V_l, E_l)$ be the coarsen graph. P_{l-1}^i is the $i-th$ contraction set of G_{l-1}. Let G_{l-1} be the graph we want to coarsen, partitioned in $(r+1)$ contractions sets $P_{l-1}^i \subseteq V_{l-1}$. So here, each node of a contraction set are combined in one node, in the next graph G_l. To describe this process, we introduced a function φ that given a node $v \in P_{l-1}^i$, returns the combined node $v_i \in V_l$ in the coarsen graph. $\forall v \in P_{l-1}^i$, $\varphi(v) = v_i'$ where

$$\begin{aligned} \varphi \colon V_{l-1} &\to V_l \\ v &\mapsto v_i' \end{aligned}$$

The coarsen graph G_l has as many vertices v_i' as G_{l-1} has contraction sets. G_l has the set of vertices

$$V_l = \{v_0', \ldots, v_r'\} \tag{3.16}$$

The way the edges are created in the coarsen graph is a bit tricky. Basically, the nodes $v_i' \in V_l$ conserve the edges they had from their respective contraction set P_{l-1}^i while multi edges become one edge. To model that idea, we going to introduce the *neighbor testing function* ψ. ϕ takes two nodes as input, and returns 1 if they are neighbor, and 0 otherwise.

$$\begin{aligned} \psi \colon V \times V &\to \{0,1\} \\ (a,b) &\mapsto \begin{cases} 1 & \text{if } a \in \Gamma(b) \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

P_{l-1}^i and P_{l-1}^j respectively contracted to v_i' and v_j'. We define each edge $e_{v_i'v_j'}^l \in E_l$, that connects a pair of nodes $(v_i', v_j') \in V_l^2$ exists if P_{l-1}^i and P_{l-1}^i are connected by an edge $e \in E_{l-1}$. We use the neighbor testing function, φ and the partitions to define each edge. If the edge satisfies the following conditions, then it exists. For all $(a, b, v_i', v_j') \in (V_{l-1})^2 \times (V_l)^2$ with $a \neq b$ and $v_i' \neq v_j'$, an edge of the coarsen graph satisfies the conditions of Eq. 3.17.

$$e_{v_i'v_j'}^l = \{(u,v) \in V_l \times V_l \mid (a,b) \in P_{l-1}^i \times P_{l-1}^j \wedge \psi(a,b) = 1 \wedge \varphi(a) = u \wedge \varphi(b) = v\} \tag{3.17}$$

The coarsen graph G_l has the set of edges

$$E_l = \{e_{v_i'v_j'}^l\}_{(v_i', v_j') \in V_l} \tag{3.18}$$

$$\begin{aligned} \mathscr{G} \colon \chi \times \mathscr{P} &\to \chi \\ (G_{l-1}, P_{l-1}) &\mapsto G_l \end{aligned}$$

3.3 Graph kernels

A kernel is a function that measure the similarity between two mathematical objects. In our case, these are graph spaces χ. Let $\phi(G)$ be the function that takes a graph from χ as input and output a vector where each of its components is a feature of G. Let $\mathscr{H} = \mathbb{R}^n$ be the feature space of finite dimension n.

$$\begin{aligned} \phi \colon \chi &\to \mathscr{H} \\ G &\mapsto \phi(G) \end{aligned}$$

Let $\langle \cdot, \cdot \rangle_{\mathscr{H}}$ be the inner product defined in the feature space $\mathscr{H}$. Let K be a kernel graph function that maps two graphs to a real number. So, for all graph G_i and G_j belonging to the space χ,

$$k : \chi \times \chi \to \mathbb{R}$$
$$(G_i, G_j) \mapsto K_{ij}$$

The measure of similarity between G_i and G_j is the inner product of their respective feature vectors as $K_{ij} = \langle \phi(G_i), \phi(G_j) \rangle_{\mathscr{H}}$. Let $\|\cdot\|_{\mathscr{H}}$ be a norm defined in the feature space $\mathscr{H}$. K is symmetric so that $K_{ij} = K_{ji}$. The distance between two feature vectors is the norm of their difference in the feature space

$$D_{ij} = \|\phi(G_i) - \phi(G_j)\|_{\mathscr{H}}$$

. The unsurpervised and supervised learning algorithms require the Gram matrix K to be a positive, semi definite matrix. That translates into a condition imposed on $K_{ij} = k(G_i, G_j)$ that is to respect the Mercer Theorem : K is said to be semi definite positive if and only if $\sum_{i=0}^{n} \sum_{j=0}^{n} k(G_i, G_j) c_i c_j \geqslant 0$ for $\forall (G_i, G_j) \in \chi^2$ and $(c_0, \ldots, c_n) \in \mathbb{R}^{n+1}$. But practically, we assure our matrices K respect this condition by computing their eigen values using scipy library. Let K and D be respectively the Gram matrix and the distance matrix, $(K, D) \in \mathscr{M}_n(\mathbb{R})^2$. $K = \{K_{ij}\}_{(i,j)\in[0,\ldots,n]^2}$ and $D = \{D_{ij}\}_{(i,j)\in[0,\ldots,n]^2}$. Now that we have D and K, we can use unsupervised and supervised learning algorithms that are based on pairwise inner product or distance between two cities G_i and G_j of our dataset $\mathscr{D}$. In the following sections, we define a kernel, by defining each time a new ϕ function here the graph is embedded. As well, other operations such as normalizing, standardizing $\phi(G_i)$ relatively to other $\phi(G_j)$ are part of the computing of a kernel. At the end of the process, we obtain n vectors $\phi(G)$. We compute the Gram and the distance matrix K and D. Then, enter these matrices in various unsupervised learning algorithms, coded already in the handy library scikit-learn. In each of the following kernels, we normalize the kernel in order to reduce the influence of size on the results. We standardize the kernel so that each feature of ϕ as an equal weight influence on K and D.

3.3.1 Degree kernel

The degree of a node d(v) is by how many roads an intersection is connected too. As well, it defines if the end of a road if it is equal to 1, or if it is in a line road if its degree is equal to 2.So we can compute the degree of each node in a graph, then make a histogram of consecutive value with holes. We will define this histogram H_{cv} in this section.

This vector of feature will describe the information of the graph. The goal here is to choose a vector that can capture the topological information of the graph. Knowing degrees, it will be intuitive to compute all the degrees in a graph, then make a histogram of the degree values. In the vector, the first component is the number of nodes having a degree equal one $d(v) = 1$. The second component is the number of nodes having a degree equal to two $d(v) = 2$ until the maximum degree found across all graphs 11. Counting if a node has degree $d(v) = k$ is equal of counting if a node belong to the center (orbit) of a k-star graphlet. We remind the name of the 1-star, 2-star, 3-star, 4-star graphlets in the Figure 3.1 of page 21. So $n(g_0), n(g_1), n(g_4), n(g_1 1)$ are respectively the counts of degree 1,2,3,4 nodes in G. Let

$$d_{max} = \max_{\substack{G \in \mathscr{D} \\ v \in V}} d(v) \tag{3.19}$$

be the maximum degree node accross all our cities. We count all k-star graphlets for $k \in \{1, 2, \ldots, d_{max}\}$. Let $\phi(G)'^d$ be our feature degree vector of size d_{max} so that $|\phi(G)_d| = d_{max}$, defined as $\phi(G)'^d = (n(g_0), n(g_1), n(g_4), n(g_{11}), \ldots, n(g'))$. The bigger the city graph, the more nodes it will have, and the bigger the sample of degrees we get. To neglect the influence of the graph size so that ϕ'^d only

takes into account the shape of distribution, we normalize $\phi^{'d}(G)$ by dividing it by the sum of its components. We call $\phi_i^{'d}$ the $i-th$ component of $\phi^{'d}(G)$.

$$\phi^d(G) = \frac{1}{\sum_i \phi_i^{'d}} \phi^{'d}(G) \tag{3.20}$$

We compute ϕ^d for our cities of our dataset $G \in \mathscr{D}$. We store them. Then, compute the Gram matrix K. From there, we can import a PCA model using scikit-learn library. We compute the three components from K using .fit method of scikit-learn. The time to compute the kernel matrix

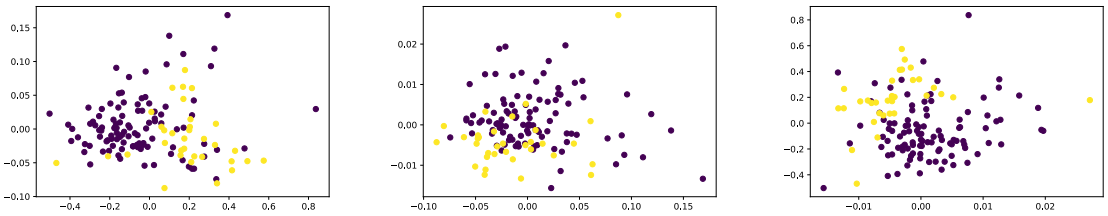


Figure 3.8: Visualization of three components of kernelized PCA using the degree kernel

K for $n = 136$ cities is less then 1 second. The code is available in my github.[8] We can see the french and cities being gathered together in Figure 3.8. However, we think that this kernel is too superficial and doesn't capture enough topological information in the graphs. The degree of a node describes if the node is an intersection $d(v) > 2$ or part of a line road $d(v) = 1$ or the end of a road $d(v) = 1$. We can suppose that since the cities of the same country (China) tend to be together, they seem be similar in the way that their distribution of intersection type is relatively similar. We want to capture if the cities contains triangle, shape of their intersections and more macro structure patterns. We can capture these informations by counting graphlets in G, as done in Dupré [2016].

3.3.2 3,4,5-graphlet normalized standardized kernel

We base this kernel on the work of the previous intern Hugo Dupré in the laboratory Dupré [2016]. He made a sampled normalized standardized 3,4,5-graphlet kernel using paper Shervashidze u. a. [2009]. At that time he was limited by networkx speed, so instead of counting all the graphlets in the graph, he had to estimate the real distrbution of graphlets by counting partially the graph, sampling graphlets. Our goal is to represent our graph using ϕ, the graphlet frequency distribution (GFD) as defined in Liu u. a. [2020].

Here we propose to use graph-tool motif count to get the total count of 3,4,5 graphlets without sampling. We take advantage that the library graph-tool has a counting motif graphlets functionality. The function takes the graphlets $g_1, g_2, \ldots, g_{29}$ (motifs), and a graph G, then count each graphlet g_i in G. $\phi_g(G)^{''} = (n(g_1), n(g_2), \ldots, n(g_{29}))$. The bigger size a graph $|E|$, the more graphlets it will have. However, we want to compare city's topology regardless of their size. If a huge city and a small city are organized as grid, they should considered similar. So, in order to counter balance the influence of the graph size on $\phi_g^{'}$, we normalize $\phi_g^{'}$ by the total number of graphlet in it. That way, each graph is embedded in the same space. $\phi_g(G)^{'}$ are defined by the proportion of each graphlet they have

$$\phi_g(G)^{'} = \frac{1}{\sum_{i=1}^{29} n(g_i)} (n(g_1), n(g_2), \ldots, n(g_{29}))$$

[8] https://github.com/endingalaporte/Graph-kernel/blob/main/_gt%20french%20china%20cities%20degree%20kernel.ipynb

We standardize each component, so that they have an equal contribution to the K_{ij} and D_{ij}. To do so, let's consider each $i-th$ component of $\phi_g(G)^{'}$ as $\phi_g(G)^{i'}$, being a random variable X^i, where X^i takes a value for each G. We compute the empirical standard deviation and average of X^i respectively as σ^i and μ^i. And we standardized each component $\phi_g(G)^{i'}$ by $\phi_g(G)^i = \frac{\phi_g(G)^{i'} - \mu^i}{\sigma^i}$. That we obtain our final $\phi_g(G) = (\phi_g(G)^1, \phi_g(G)^2, \ldots, \phi_g(G)^{29})$, explained in Dupré [2016]. In this

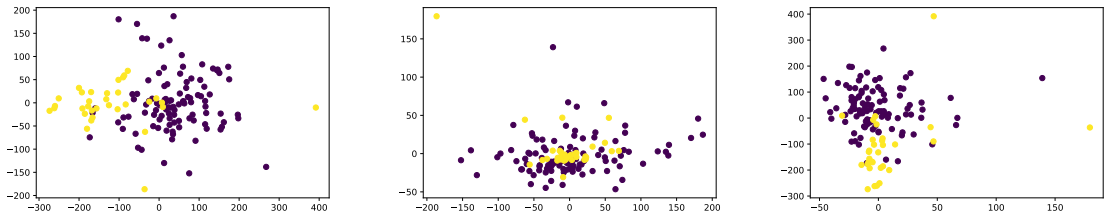


Figure 3.9: Visualization of the three components of kernalized PCA, using the 3,4,5-graphlet kernel normalized standardized

kernel we embeded each graph G in a size 29 vector $\phi_g(G)$, where each component is the count of a graphlet of 3, 4 or 5 nodes. When we gather all the $\phi_g(G)$ vector where cities are embedded and compute the standard deviation accross them all, we can see by which graphlet g_i the cities have a variance. In order words, we can observe by which graphlet we can differentiate cities between them. Let's $N(g_i)$ be size $(n+1)$ vector $|N(g_i)| = n$ where each component is the counts of the $i-th$ graphlet g_i in the city's graph. $N(g_i) = (n(g_i)_0, n(g_i)_1, \ldots, n(g_i)_n$. We can compute the standard deviation σ of the count of the i-th graphlet $N(g_i)$ in Figure 3.10. The time to compute the kernel for $n = 136$ cities is 15 minutes. The code of this kernel is available in my github[9].

In Figure 3.10, we see for example that the 6 graphlet to have the highest standard deviation σ are the graphlets n°10, 1, 4, 11, 15. In that case, the cities's topology vary the most by their count of graphlets tailed-3-star, 2-star, 3-star, 4-star, 5-cycle. We thought about counting higher order graphlet in a graph but since I didn't find any paper in the literrature defining enumerating specifically 6-node-graphlets, 7-node-graphlets, I didn't pursuit in this direction of counting higher number graphlets. In another hand, it seems that the higher the graphlet order (number of nodes in the graphlet), the smaller the standard deviation. So we don't bother so count higher order graphlets. In the next sections, we concentrate of how we can gather more information using 3,4,5-node-graphlets. One way is to count orbits in 3,4,5-graphlets.

[9] https://github.com/endingalaporte/3-4-5-graphlet-kernel/blob/main/src/_345graphlet%20kernel.ipynb

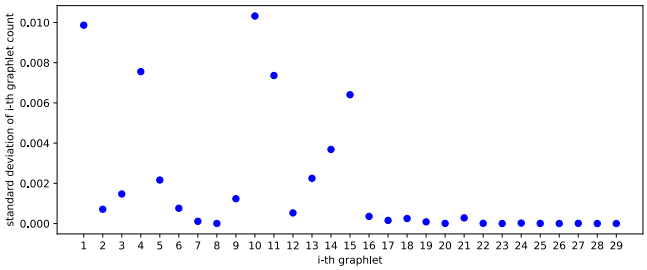


Figure 3.10: Standard deviation of the count of each graphlet accross n+1 cities

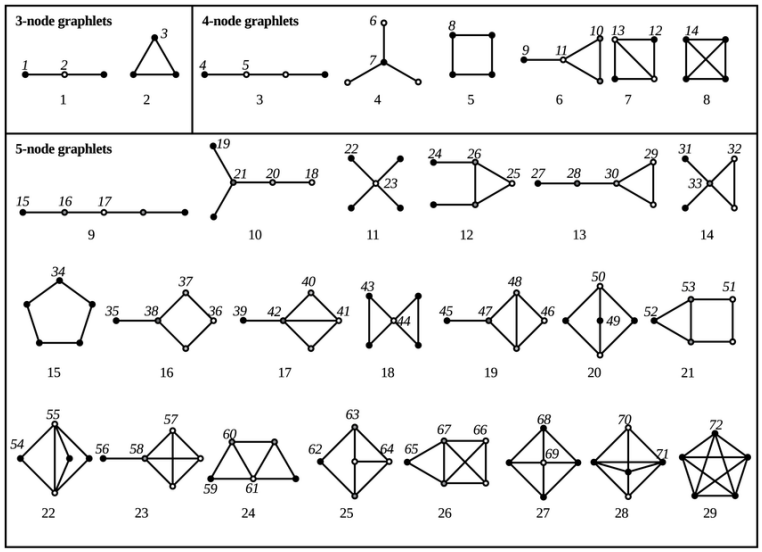


Figure 3.11: Names of orbits across 29 3,4,5 node graphlets

3.3.3 Orbit normalized standardized kernel

We inspire our method of comparing networks using orbit from the work of Pržulj [2007]. In their paper, they use the count of orbits to compare graphs. In their case, their graphs represent biological networks (molecule interactions in cellular systems). In the previous kernel, we saw that using 3,4,5-graphlets to compare cities is efficient, so we can use the work on Pržulj [2007], based on orbits of 3,4,5 graphlets as well. Similarly to the 3,4,5-graphlet kernel, intead of counting how mane nodes belong to a graphlet, we count how many nodes belong to a localised node in the graphlet. Each of these localisations are called orbits. For the 29 3,4,5-node graphlets, there are 72 orbits as we can see in Figure 3.11. I didn't find an algorithm of the orbit counting model. Nevertheless, we use an implementation of the code made by Benedek Rozemberczki, from Unviersity of Edinburgh called OrbitalStrike[10]. His code is in full python so isn't as fast as if we used a graph-tool implementation. OrbitalStrike takes a graph edgelist as input and returns a matrix of $\mathscr{M}_{|V|,67}(\mathbb{N})$ where each row is a node's orbits. It counts 67 orbits from the graphlet g_0(single edge graphlet), to graphlet $g_2 6$ from Figure 3.11 (see Figure 3.1 for graphlet names) and omit the three 5-graphlets. Let $\phi(G)_{orbit} \in \mathbb{R}^{67}$ be the function that transform G in a vector space using orbit counts.

For a graph $G \in \mathscr{D}$, for $k \in \{0, 1, \ldots, |V| - 1\}$ and $i \in \{0, 1, \ldots, 66\}$, let $n(o_i)_k$ be the number of time of that the k-th node in the city G belongs to the i-th orbit o_i. Let

$$\phi(G)^a = \sum_{k=0}^{|V|} (n(o_0)_k, n(o_1)_k, \ldots, n(o_{66})_k) \tag{3.21}$$

Then, same process in order to reduce the influence of city size $|E|$. We denote by $\phi(G)_i^a$ the i-th component of vector $\phi(G)^a$. We normalize the vector by dividing it by the number of orbit counts.

$$\phi(G)^b = \sum_{i=0}^{66} \phi(G)_i^a \tag{3.22}$$

The last step is to normalize each component of $\phi(G_i)^b$ relatively to the other $\phi(G_j)^b$. That way,

[10] https://github.com/benedekrozemberczki/OrbitalFeatures

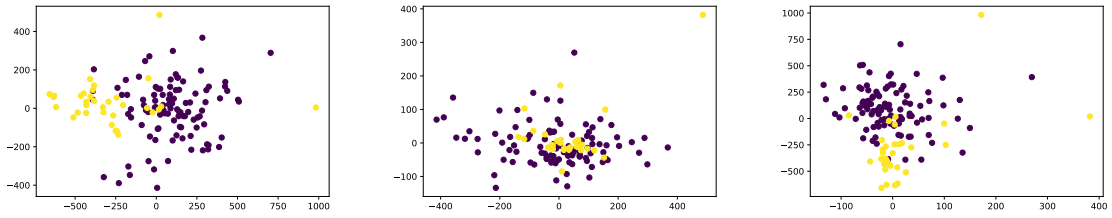


Figure 3.12: Visualization of our cities projected in three components using PCA

each component of all $\phi(G)$ will vary in the same range of value. We have $\phi(G)^b$, let

$$C_i^b = (\phi(G_0)_i^b, \phi(G_1)_i^b, \ldots, \phi(G_n)_i^b)$$

be the set i-th components of $\phi(G)_i^b$ of the other graphs. Let $\sigma(.)$ and $\mu(.)$ be the standard deviation and mean values of a set.

$$\phi(G)_{orbit} = (\frac{\phi(G)_0^b - \sigma(C_0^b)}{\mu(C_0^b)}, \frac{\phi(G)_1^b - \sigma(C_1^b)}{\mu(C_1^b)}, \ldots, \frac{\phi(G)_n^b - \sigma(C_n^b)}{\mu(C_n^b)}) \tag{3.23}$$

We compute $\phi(G)_{orbit}$ for each or our graphs, compute the K and D matrices. Once we projected each G using $\phi(G)_{orbit}$ in $\mathscr{H}$, we use PCA method to visualize it on three components. The downside of this kernel is that we rely on the external code OrbitalStrike to compute the orbits for each graph so it makes it harder to implement. As well, since I didn't code OrbitalStrike, I don't have an insight of how exactly the code work. I have two codes to execute instead of one. As well, since OrbitalStrike is coded in full python and NetworkX library, the code is slow.. It takes about 2 hours to compute K and D for $(n+1) = 136$ cities. The counting orbits process is the longuest, it takes around 7 minutes to computes the orbits of each node in a city G. The code to compute this kernel is available in my github[11].

3.3.4 Edge-centric 3,4-graphlet degree distribution kernel

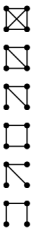
Let $B \in \mathbb{N}^{+b}$ be an array of positive integer values. b is the maximum value in B, $b = \max(B)$. We define here a function H_{cv} that we call cumulative values histogram. H_{cv} takes an array B as input and output a count of cumulatives values of 1 to d. We denote by $c(i)$ be the number of times the value i appears in B, $C = (c(1), c(2), \ldots, c(d))$

$$H_{cv} \colon (\mathbb{N}^+)^{|B|} \to (\mathbb{N}^+)^b$$
$$B \mapsto C$$

We define the histogram H_{cv} that way so that all we express all graphs in the same space. There are 8 all connected undirected graphlets of 3 and 4 nodes. The algorithm LOCALGRAPHLET from Ahmed u. a. [2016] gives for each edge $e \in E$, its graphlet degree vector for $2, 3, 4$ nodes graphlets. In the graphlet degree vector, each component is the count of the graphlet triangle, 2-star, 4-clique, chordal-cycle, tailed-triangle, 4-cycle, 3-star, 4-path, Figure 3.13. The algorithm is a map that given a graph $G \in \chi$, returns what will call the matrix of counts $Q \in \mathscr{M}_{|E|,8}$. $Q = \{Q_{ij}\}_{(i,j)\in\{1,2,\ldots,|E|\}\times\{1,2,\ldots,8\}}$. Let $d_{gc} = d_{graphletcounts}$ as

$$d_{gc} = \max_{\substack{G\in\mathscr{D} \\ e\in E \\ i\in\{1,2,\ldots,|E|\} \\ j\in\{1,2,\ldots,8\}}} Q_{ij} \tag{3.24}$$

[11] https://github.com/endingalaporte/Graph-partitioning/blob/main/GDV_orbit/_gt%20gdv%20orbit.ipynb

		CONNECTED		
GRAPHLETS	$\mathbf{k=3}$		G_1	triangle
			G_2	2-star
	$\mathbf{k=4}$		G_5	4-clique
			G_6	chordal-cycle†
			G_7	tailed-triangle‡
			G_8	4-cycle
			G_9	3-star
			G_{10}	4-path

† Diamond and chordal-cycle are interchangeable.
‡ Paw and tailed-triangle are interchangeable.

Figure 3.13: All connected k ∈ {3, 4} nodes graphlets

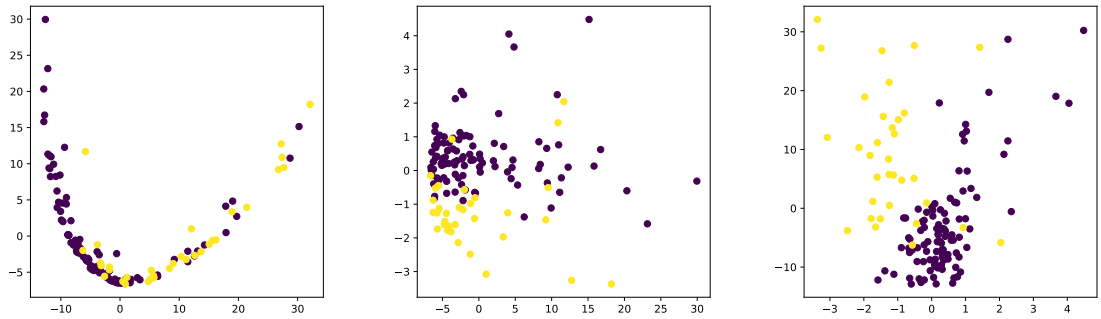


Figure 3.14: Visualisation of our cities projected in three components using the kernelized PCA, of the graphlet degree kernel

Let Q^j the $j-th$ column of the 8 columns of Q. We're going to compute the consecutive value histogram of Q^j for each column using H_{cv} with $b = d_{gc}$. H_{cv} takes values from $(\mathbb{N}^+)^{|B|}$ and returns its histogram of consecutive values in $(\mathbb{N}^+)^b$. We transform each distribution Q^j in a set of size $b = d_{gc}$. We obtain the vector of distribution histogram $H_{cv}(Q^j) \in (\mathbb{N}^+)^b$ with which we can plot an histogram. As well, we're going to normalize each histogram by dividing each of its components by the sum of all its components. Let $H^{\Sigma}_{cv}(Q^j)$ be the total sum of all the components of $H_{cv}(Q^j)$. Then our graph is represented as set of histograms of each graphlet degree.

$$\phi_{graphletdegree} = \left(\frac{H_{cv}(Q^1)}{H^{\Sigma}_{cv}(Q^1)}, \ldots, \frac{H_{cv}(Q^8)}{H^{\Sigma}_{cv}(Q^8)}\right) \tag{3.25}$$

We delete the components of $\phi_{graphletdegree}$ if they are filled with a zero for all $G \in \mathscr{D}$. That, way, we reduce the number of our features, and reduce also the artificial similarity between our cities. For example, if all cities, all don't have a a degree 2 of graphlet triangle, they will all contain a zero. However, we choose to keep only the features that differentiate them.[12]. We visualize the three components using PCA, Figure 3.14. We visualize them in a 3D plot in Figure 3.15. The French

[12] https://github.com/endingalaporte/Graphlet-degree-kernel/blob/main/src/_np%20graphlet%20degree%20kernel.ipynb

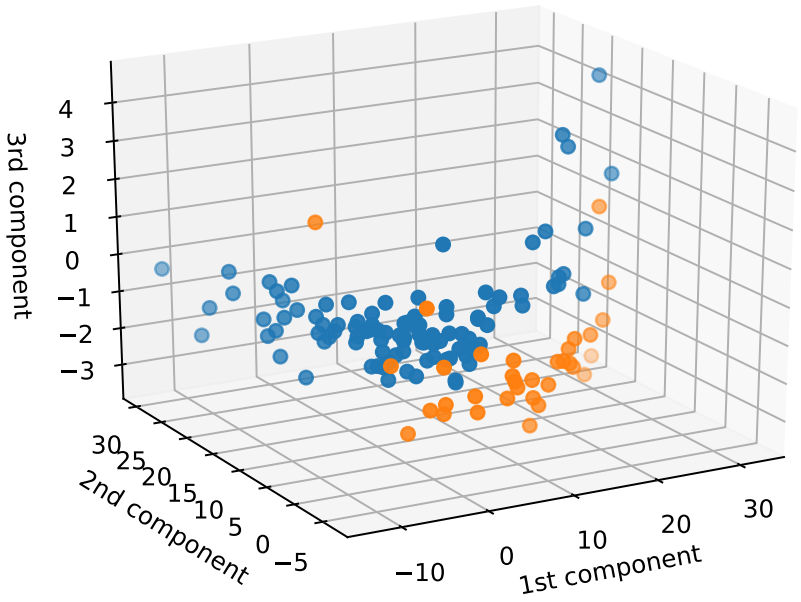


Figure 3.15: 3D visualization of three PCA projected components using the graphlet degree kernel

and Chinese cities look indeed to be separated from certain components while their separation is not obvious. In the clustering section, we will compute the silhouette score of to quantify the quality of the separation. The computing time of K and D is 1 hour.

3.3.5 Orbit degree distribution (GDD) kernel

In the orbit degree distribution kernel, we count the orbit of Figure 3.11 degree vector to which each node belongs to, using the code OrbitalStrike. The way this kernel is defined is similar to the graphlet degree kernel except that the feature vector of each node is of size $s = 67$ instead of $s = 29$. We visualize our cities projected in three components of PCA in Figure 3.12. We visualize our cities representation vector in Figure 3.17. It's nice to see that in the vector in which each graph is embedded, we can still observe the distribution of the degree of orbits. in the final embedding, after the removal of columns full of zeros (to increase the dissimilarity between datapoints), each graph is projected in the space $\mathscr{H}$ of 2037 dimensions. Computing orbits, instead of graphlet conts, adds a depth in the information we capture.Pržulj [2007]. The code of this kernel is available in my github[13]. However this kernel is long to compute.

[13] https://github.com/endingalaporte/Orbit-degree-kernel/blob/main/src/_np%20orbit%20degree%20kernel.ipynb

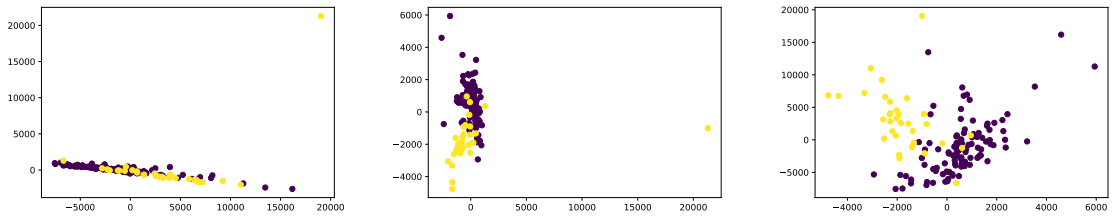


Figure 3.16: Visualization of our cities projected using kernelized PCA, with the orbit kernel

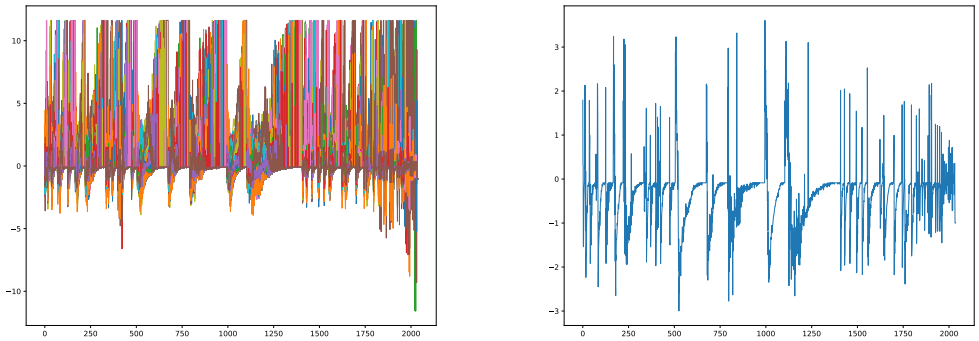


Figure 3.17: Each city is embedded in $\mathscr{H}$, the space of 2037 dimensions, and the representation of the city of Ajaccio in France

Chapter 4

Clustering

4.1 Silhouette score of each cluster of the same country

The silhouette is a tool to measure the quality of a clustering. Here's a little reminder of how the silhouette is computed, as defined in Rousseeuw [1987]. A cluster C_i is a set of datapoints. For all $i \in C_i$, point i in the cluster C_i, how well i is assigned to its cluster C_i is defined by $a(i)$, where the smaller $a(i)$ the better.

$$a(i) = \frac{1}{|C| - 1} \sum_{j \in C_i, i \neq j} d(i, j) \tag{4.1}$$

Dissimilarity $b(i)$ of i to other clusters C_k is the distance from i to all other clusters C_k, where $C_i \neq C_k$. For all $i \in C_i$, the smallest average distance to of i to other points of other clusters C_k is $b(i)$.

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i, j) \tag{4.2}$$

In a data set where clusters are identified, the silhouette $s(i)$ of a point i belonging to cluster C_i is an indicator of how well clustered the data point i is. We define the silhouette of a point i, so that $-1 \leqslant s(i) \leqslant 1$,

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{4.3}$$

if $|C_i| > 1$, and $s(i) = 0$ if $|C_i| \leqslant 1$. As an indicator, if all the points of a clusters are the same, case of hypothetical perfect cluster where all points are in the same exact location, $s(i)$ would be equal to 1. $s(i) = 0$ means that clusters cannot be differentiated in the data, this would be the case if two clusters have their points all equals they can not be distinguished from each other.

To measure the quality of our clustering for each kernel, we're going to compute the $s(i)$ silhouette of each data point, then compute the average of all data points. Each of our $(n + 1) = 136$ city graphs for a kernel, represented like $\phi(G)$. We're going to compute the silhouette score $\tilde{s}$ where

$$\tilde{s} = \frac{1}{n + 1} \sum_{G \in \mathscr{D}} s(\phi(G)) \tag{4.4}$$

In our dataset of cities $\mathscr{D}$, we have two clusters C_0 and C_1 corresponding respectively to the labeled French and Chinese cities. Each city is projected in $\mathscr{H}$ using ϕ. That way, $|C_0| = 103$ and $|C_1| = 33$. The library scikit-learn offers an implementation of the silhouette score[1] that we use to compute it for each kernel. The silhouette score $\tilde{s}$ for each kernel is in Table 4.1. Degree, Graphlet, Orbit,

[1] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html

Graphlet degree, Orbit degree are respectively the name of the kernels in section 3.3.1, 3.3.2, 3.3.4, 3.3.3, and 3.3.5. In Table 4.1, we arranged from left to right the kernel that constrain the most

Table 4.1: Silhouette score for each kernel

Kernel	Degree	Graphlet	Orbit	Graphlet degree	Orbit degree
Silhouette score	0.216	0.266	0.230	0.236	0.194

the space of graph solutions as in Eq. **??**. The more complex our kernel, the more topological information it takes from the graph. We supposed here, labeling cities according to the country they are in, that the topology of a city depends on which country they are. Another assumption is that the cities in a country have similar topology. If this was the case, the silhouette score would increase from left to right of the Table 4.1. But it isn't the case. From the degree kernel to the graphlet kernel, we see an improvement of the clustering. The degree kernel is too simple to describe the graph, thus not being able to different them using this kernel. When we look at the 3,4-graphlet degree kernel, or the orbit degree kernel, we see a fall of the silhouette score relatively to the graphlet kernel. I think that it is because the kernel finds differences between the city inside the same cluster, as their topology is even more captured by these two kernels. That way, I want to say that it is not relevant to judge a kernel's efficiency to cluster cities, by evaluating the clustering according to labels we are not even sure have a link the topology of the city. It would be more interesting to project the cities in that space, unlabeled and see where clusters seem to form. That's what we're going to do in the next section using an unsupervised learning method called DBSCAN. DBSCAN, given density parameter we choose, basically identify the high density zones of the spaces (density of datapoints) and deduce that there is a cluster in it. Moving right along to the next section.

4.2 DBSCAN reveals clusters of predetermined density

In this section, we describe how we make clusters out of our small dataset $\mathscr{D}$. We choose heuristically a density parameter in order to divide our data in three densed classes. We consider three dense zone as cluster T_1, T_2 and T_3. Then, we remove the points falling outside of that dense zones (noise points given by DBSCAN we will define here as well). Finally, we visualize each of the 3 dense zones, using a projection of our data using a kernel PCA of the graphlet degree kernel of section 3.3.2. Here, we remind what DBSCAN is. Density-Based Spatial Clustering of Applications with Noise, or of initials DBSCAN is a unsupervised learning technic. DBSCAN is a method that given a dataset and a density, defined by two parameters ϵ and m_s the minimum of samples, returns labels or clusters, and label of points classified as noise.

So basically, the algorithms begins to compute the density of each point p_i. To do so, the density is the number of points m in the hyper-circle of radius ϵ. If the number of points m is greater then the one we chose m_s then the point p_i is considered as a core point. The algorithms computes the density for each point, and deduce the core points. Then, from each core points, the cluster spreads until the points falls out of the chosen density (ϵ and m_s), after these points at the limit of the clusters, the remaining points are classified as noise. Here's an illustration of the process in Figure 4.1 from DiFrancesco u. a. [2020]. This method is great when we suppose that our data is composed of clusters of similar density, so that each cluster can be detected, using that density condition we first set. The advantage of this method is that it can find clusters even if they are dipersed non linearly in the space $\mathscr{H}$. In Figure 4.2, image courtesy of Wikipedia[2], we see the example on a dataset that has nothing to do with our study, just so that the reader gets a good intuition of how DBSCAN behaves. In our case, from the symmetric distance matrix $D \in \mathscr{M}_{(n+1)}(\mathbb{R}^+)$ we computed

[2] https://en.wikipedia.org/wiki/DBSCAN

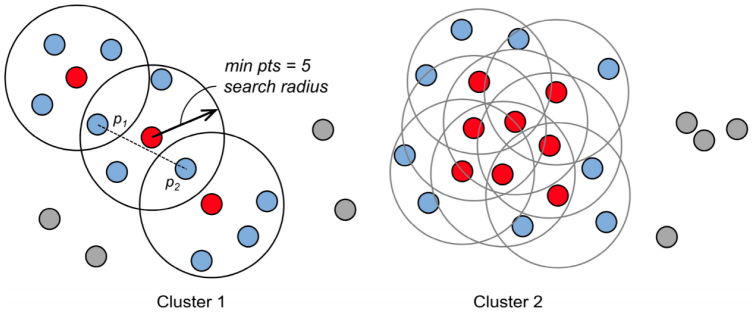


Figure 4.1: Illustration of cluster construction using DBSCAN. m_s is the min pts of the image. Core points are colored in red. The points in grey are those falling outside of the density, classified as noise

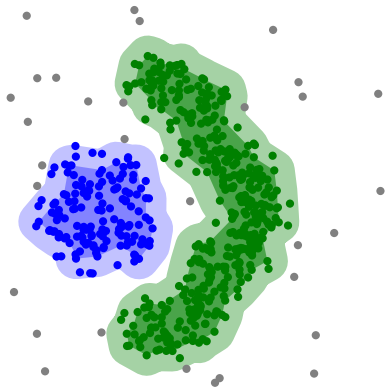


Figure 4.2: Here's a set of data points that has nothing to do with our study. It is an example of DBSCAN in action. Non linearly separable clusters are detected using DBSCAN. Here, the algorithm detected two main clusters, from the original unlabeled data points.

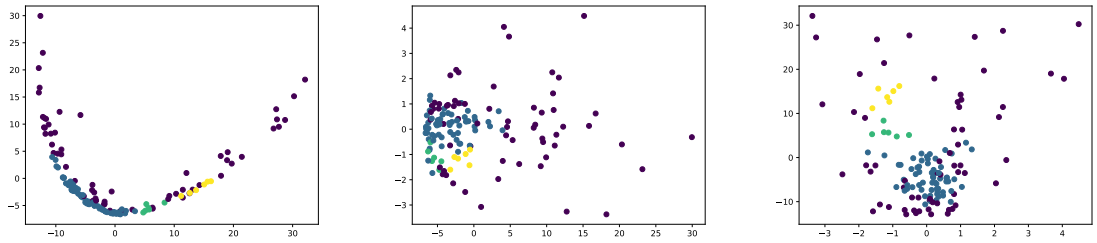


Figure 4.3: Visualization of our 3 clusters deduced by DBSCAN, with noise points, on our three PCA components of graphle degree kernel

using the edge degree kernel, the parameters $\epsilon \in \mathbb{R}^+$ and minimum samples $m_s \in \mathbb{R}^+$, DBSCAN returns an array of $(n+1) = 136$ labels $\tilde{\mathscr{Y}} \in (\mathbb{N}^+)^{n+1}$ where $\tilde{\mathscr{Y}} = \{\tilde{Y}_i\}_{i\in\{0,1,\dots,n\}}$. Since we want to define zones of topological similarity for practical use, there are no right answer to how to how many clusters we should find here. We could tweak the parameters in order to get, one, two, three, four clusters. So we rely on practicality to choose the number of clusters we want. The clusters we form here using DBSCAN depend on the parameters ϵ and m_s we choose. These parameters define the density. Given our chosen density, the data points will be labeled in clusters or in noise. Before choosing randomly ϵ and m_s, I first used method based on an assumption. The assumption is that the datapoints, each of the cities are part of a topological group. From here, we suppose so that there is little noise in our dataset. From that, we propose to choose ϵ and m_s so as to maximise the number of labeled datapoints. It is the equivalent to choose ϵ and m_s so that the number of datapoints labeled as noise ($Y_i = -1$) is minimized. But the result was not great. Majority the datapoints were labeled as one cluster, with little to no noise. That's what I asked, but the result is not satisfying. I choosed after that to test different values of ϵ and m_s in order to get the least points categorised as noise, and two or three or four (that is subjective here) more or less equally distributed clusters. Found out that $\epsilon = 1.6$ and $m_s = 3$ works well and produces 3 clusters, while other 50% of the nodes are categorized as noise. We see in the Figure 4.3 the three clusters in blue, green, yellow color. The noise points are colored in purple. We delete the datapoints that were classified as noise by DBSCAN. 70 of the $(n+1) = 136$ datapoints are removed. We plot the remaining nodes part of the three clusters. In Figure 4.4, we see each of the cluster of cities. We name each of the cluster T_1, T_2, T_3, three spaces were lies the cities of the three particular topology groups. For example, if we included American cities that are known for being organized in a grid like manner Dupré [2016], this configuration know as hippodamian plan, we would visualize a new cluster $T_{hippodamian}$ of cities being organized like this.

We can name each cluster accordingly to its topological features of ϕ. To do that, we could compute the centroid point for each cluster, then, compare (difference) each of the components ϕ of each centroid of each cluster. That way, we can trace back if a cluster is composed of cities that have (in the case of our graphlet degree kernel) let's say many nodes that are connected to 3 triangles, many nodes that are connected to 5 3-star graphlets. We could call this family the cities of topology $T_{intersection}$ for example. Here, I didn't trace back why citie are part of the same cluster, because of a lack of time. However it is possible, since we defined formally how each of our kernels are constructed. The code I wrote to reproduce these results is available in my github[3].

[3] https://github.com/endingalaporte/DBSCAN-/blob/main/src/_np%20graphlet%20degree%20kernel.ipynb

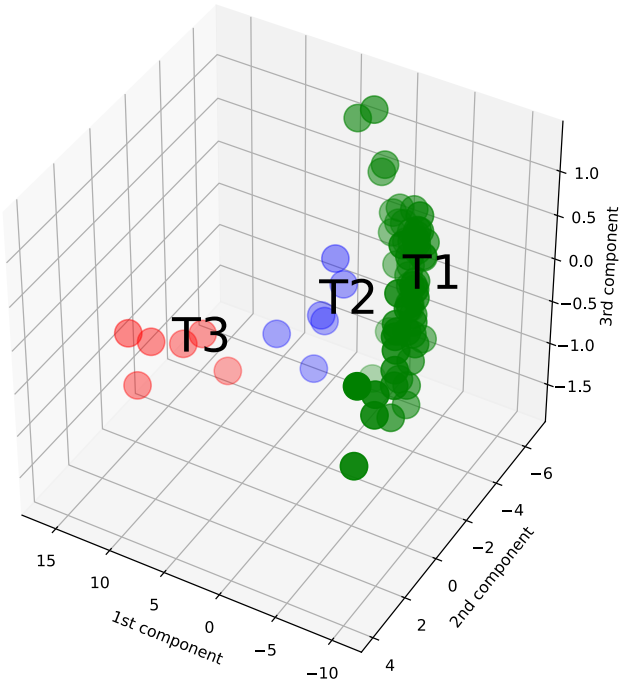


Figure 4.4: Three topological zones T_1, T_2 and T_3. Their names are placed in the centroid of each cluster

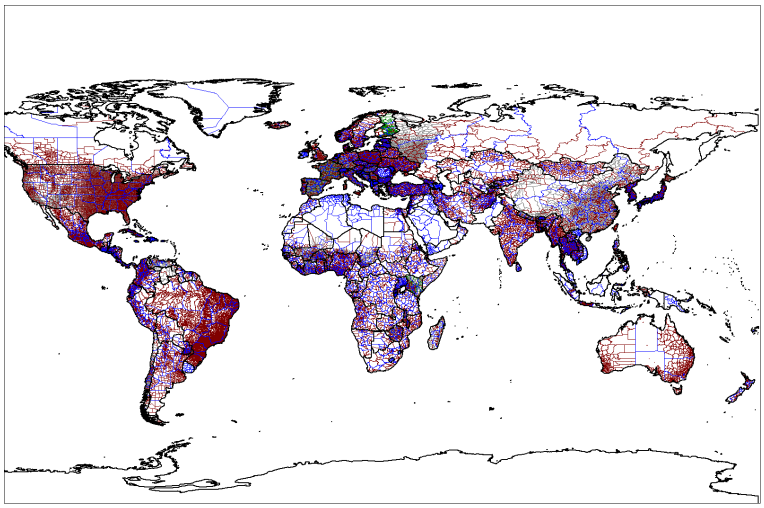


Figure 4.5: Visualization of how the earth's surface is divided in partitions, where each chunk is delimited by administrative limits lines

4.3 Proposition of a way to characterize cities

4.3.1 ISO 3166 : Existing spatial partitioning of cities

In this section, we describe the way cities limits in the surface of the earth are defined using a standardize norm ISO 3166 from the International Organization for Standardization. The iso 3166 defines how the surface of the earth is cut into chunks where each chunk is a place delimited by its administrative limits, Figure 4.5. Then, we talk about how the work of DBSCAN of the previous section, using cities of around the world (or the 200000 cities of Europe) can constitute similarly a new standardized way to classify cities. The idea is that, by projecting a huge dataset of cities around the world in a space $\mathscr{H}$ and by considering the densest zones, we can define different zones in $\mathscr{H}$, to which corresponds a particular city topology. As well, we could partition this space into chunks, where each chunk (and its limits, analogously to how cities limits are defined by their administrative limits) defines a family of city topological type.

4.3.2 Partitioning of the spaces of city topological types

Here, we used the method to identify clusters by their density. Our dataset is well ... very *small* and not equilibriate between classes (labels). For future improvement, I suggest that we take an equal number of cities from each label (here the label are countries), for example, taking 100 and 100 French and Chinese cities instead of around 100 and 30. Since DBSCAN suppose that each cluster has similar density (of points in the space), our clustering is biaised by the fact that the zone where lies the chinese cities has a smaller density, due by the smaller number of cities.

I suggest that for further study, we project in $\mathscr{H}$ using for example all 200_000 cities of Europe that my supervisor extracted, use the $3, 4, 5$-graphlet kernel of section 3.3.2 coded using graph-tool library, because it is fast (15 mins for 133 cities) thus scalable, and is a compromise to the graphlet degree kernel. As well, it is extensively used in the literature. During this study, I've been limited to the hardware I worked with, limited computing power.

I suggest that for future works, we project the 200000 cities of Europe in $\mathscr{H}$, choose a density to which we classify our cities of Europe. Then, define zones of city of same topology, as we did with T_1, T_2, T_3. I think this is great because it will be new in literature (according to what I read) to

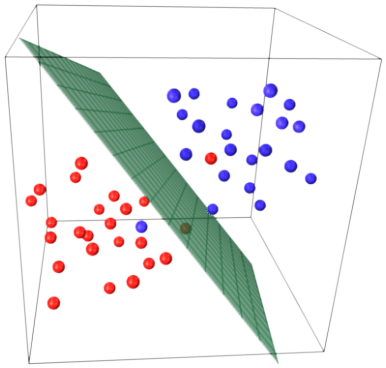


Figure 4.6: Illustration of an hyperplane lying in a 3D representation of $\mathscr{H}$. The hyperplan cuts $\mathscr{H}$ in two parts, where a cluster of cities lies in each chunk. Each chunk represent a type of cities of same topological property. We could name the left side and the right side of the space T_a, T_b let's say

classify city according to new topological types (additionally of the space of cities of hippodamian grid). So after all the cities are projected in the space $\mathscr{H}$, we can partition the space in various chunks, were each chunk is a topological zone. A method to do that would be to identify clusters, then to cut the space using hyperplanes in $\mathscr{H}$. This can be done by considering each cluster having a label, and computing the optimal hyperplanes using support vector machine. Then, we get the coordinates of the hyperplanes. That way, we can partition the space $\mathscr{H}$ into chunks and define each chunk correspond to which topology of cities T_k, as illustrated in Figure 4.6. This would be a additional way to classify cities. Such a visualization of the cities of the world would be ... awesome. It would be a novel (I didn't read such classification in the literature) way to characterize urban structure in urban science. It will have a philosophical and technical value and opens new possibilities to find correlation a city's topology and its socio-cultural-economical characteristics. In short, opening new paths for the understanding of cities as complex systems, in the context of urban study.

Chapter 5

Conclusion

We proposed a way to cluster graphs using kernel constructed from graphlets, and constructed from graphlet degree vectors. We can see patterns in the small dataset we have. There exist a topological similarity that seperate the cities of China from the ones in France. We formally defined a method to embed a graph into a vector space using a coarsening technique based on the grouping of nodes of similar topological environnement, that awaits to be implemented. We proposed a method to find clusters of cities in their topological space for any given number of cites. We showed our method using our small dataset of cities, clustered in three densed zones using DBSCAN. We also identified the limits of the clustering using DBSCAN due to the unbalanced classes in the labeled data.

5.1 Future work

For future work, we propose in this paragraph ways to improve of our algorithms, and also suggest to use a bigger more balanced dataset, with same quantity of datapoint in each class. I didn't have time to implement our coarsening algorithm. The next step would be to use code the coarsening algorithm, using graph-tool or any other frameworks. After embedding the cities using the method FOLD we defined, we could feed them to a standard VGG16 Convolutional Neural Network, using a predefined label (such as population, GDP). Then, we can visualize the Class Activation Maps to see which part the graph triggers the class it activates. Graph-tool library, additionaly to offers a motif count function, also has the functionnality of locating where the motif was found in the graph. We could write our graphlet degree function using graph-tool (backend in C++) so that it is faster then the one we wrote in python. That way, the kernel would be more faster and scalable for bigger datasets. During this internship, I was limited by the computing power I had. Further study include to use the fastest methods in this report (graphlet, graphlet degree kernel) with a bigger dataset, in order to define pertinent topological clusters group T_k of cities.

5.2 Personnal take away

In a personal standpoint, this internship was tremendously rich in technical knowledge. I will use the tools I learned personal to save time in my personal projects. As well, I got to learn to be computer literate, how to automate redundant processes. I got to learn to impose myself a schedule to work autonomously, organize my working time, so that my work fits my optimum productivity level. Nowadays, there are more remote based works in the job market. Trendy terms such as *digital nomads* appears and I wanted to get an insight of how that kind of work looks like. In this first experience, I earned skills to to work remotely efficiently. Regarding the subject, I got to fulfill a

strong erudite appetite I have concerning the wide subject of data science and machine learning, reading papers articles about the subject for months.

Appendices

Appendix A

Link to the codes I wrote

Here are the GitHub links to the main codes I wrote during the internship.

- Algorithm 2 of the Edge-centric graphlet count paper
  https://github.com/endingalaporte/alg2-Exact-and-Estimation-of-Local-Edge-centric-Graphlet-Cou
  blob/main/_graph_tool%20LOCALGRAPHLET.ipynb
- Clustering of lattice and circular graphs using shortest path and degree kernel
  https://github.com/endingalaporte/Shortest-path-graph-kernel/blob/main/_gt%20shortest%
  20path%20kernel%20and%20degree%20kernel.ipynb
- Routine graph manipulation using graph-tool library
  https://github.com/endingalaporte/Identify-graphlets-in-subgraph/blob/main/_graph-tool%
  20routines.ipynb
  https://github.com/endingalaporte/Extract-cities/blob/main/_osmnx%20china.ipynb
- Statistical description of French and Chinese graphs
  https://github.com/endingalaporte/Stats-of-cities-road-graphs/blob/main/src/_gt%
  20stats.ipynb
- Partitioning using orbit GDV similarity
  https://github.com/endingalaporte/Graph-partitioning/blob/main/GDV_orbit/_gt%20gdv%
  20orbit.ipynb
- Partitioning using degree similarity
  https://github.com/endingalaporte/Graph-partitioning/blob/main/GDV_degree/_gt%
  20gdv%20degree.ipynb
- Orbit kernel
  https://github.com/endingalaporte/Graph-partitioning/blob/main/GDV_orbit/_gt%20gdv%
  20orbit.ipynb
- 3,4,5-graphlet kernel
  https://github.com/endingalaporte/3-4-5-graphlet-kernel/blob/main/src/_345graphlet%
  20kernel.ipynb
- Orbit degree kernel
  https://github.com/endingalaporte/Orbit-degree-kernel/blob/main/src/_np%20orbit%
  20degree%20kernel.ipynb

- Making clusters using DBSCAN with graphlet degree kernel
https://github.com/endingalaporte/DBSCAN-/blob/main/src/_np%20graphlet%20degree%20kernel.ipynb

Bibliography

[Ahmed u. a. 2015] Ahmed, Nesreen K. ; Neville, Jennifer ; Rossi, Ryan A. ; Duffield, Nick: Efficient graphlet counting for large networks. In: *2015 IEEE International Conference on Data Mining* IEEE (Veranst.), 2015, S. 1–10

[Ahmed u. a. 2016] Ahmed, Nesreen K. ; Willke, Theodore L. ; Rossi, Ryan A.: Exact and estimation of local edge-centric graphlet counts. In: *Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications* PMLR (Veranst.), URL http://proceedings.mlr.press/v53/ahmed16.pdf, 2016, S. 1–17

[Boeing 2017] Boeing, Geoff: OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. In: *Computers, Environment and Urban Systems* 65 (2017), S. 126–139

[Boeing 2019] Boeing, Geoff: Urban spatial order: Street network orientation, configuration, and entropy. In: *Applied Network Science* 4 (2019), Nr. 1, S. 1–19

[Boeing 2021] Boeing, Geoff: Street network models and indicators for every urban area in the world. In: *Geographical Analysis* (2021)

[Cannoodt u. a. 2018] Cannoodt, Robrecht ; Ruyssinck, Joeri ; Ramon, Jan ; De Preter, Katleen ; Saeys, Yvan: IncGraph: Incremental graphlet counting for topology optimisation. In: *PloS one* 13 (2018), Nr. 4, S. e0195997

[DiFrancesco u. a. 2020] DiFrancesco, Paul-Mark ; Bonneau, David ; Hutchinson, D J.: The implications of M3C2 projection diameter on 3D semi-automated rockfall extraction from sequential terrestrial laser scanning point clouds. In: *Remote Sensing* 12 (2020), Nr. 11, S. 1885

[Dupré 2016] Dupré, Hugo: Etude de la topologie des villes à l'aide de kernels sur les graphes / ENSAE ParisTech National School of Statistics and Economic Administration. 2016. – Forschungsbericht

[Dutta u. a. 2020] Dutta, Anjan ; Riba, Pau ; Lladós, Josep ; Fornés, Alicia: Hierarchical stochastic graphlet embedding for graph-based pattern recognition. In: *Neural Computing and Applications* 32 (2020), Nr. 15, S. 11579–11596. – URL https://arxiv.org/pdf/1807.02839.pdf

[Dutta und Sahbi 2018] Dutta, Anjan ; Sahbi, Hichem: Stochastic graphlet embedding. In: *IEEE transactions on neural networks and learning systems* 30 (2018), Nr. 8, S. 2369–2382

[Jordahl 2014] Jordahl, K: GeoPandas: Python tools for geographic data. In: *URL: https://github. com/geopandas/geopandas* (2014)

[Khorshidi u. a. 2020] Khorshidi, Samira ; Al Hasan, Mohammad ; Mohler, George ; Short, Martin B.: The role of graphlets in viral processes on networks. In: *Journal of Nonlinear Science* 30 (2020), Nr. 5, S. 2309–2324

[Kipf und Welling 2016] Kipf, Thomas N. ; Welling, Max: Semi-supervised classification with graph convolutional networks. In: *arXiv preprint arXiv:1609.02907* (2016)

[Lefebvre-Lepot 2010–2011] Lefebvre-Lepot, Aline: *Quelques rappels sur le langage ensembliste*. 2010-2011

[Liu u. a. 2020] Liu, Xutong ; Chen, Yu-Zhen J. ; Lui, John C. ; Avrachenkov, Konstantin: Learning to count: A deep learning framework for graphlet count estimation. In: *Network Science* (2020), S. 1–38

[Loukas 2019] Loukas, Andreas: Graph Reduction with Spectral and Cut Guarantees. In: *J. Mach. Learn. Res.* 20 (2019), Nr. 116

[McKinney u. a. 2011] McKinney, Wes u. a.: pandas: a foundational Python library for data analysis and statistics. In: *Python for high performance and scientific computing* 14 (2011), Nr. 9, S. 1–9

[Milenković u. a. 2010] Milenković, Tijana ; Ng, Weng L. ; Hayes, Wayne ; Pržulj, Nataša: Optimal network alignment with graphlet degree vectors. In: *Cancer informatics* 9 (2010), S. CIN–S4744

[Milenković und Pržulj 2008] Milenković, Tijana ; Pržulj, Nataša: Uncovering biological network function via graphlet degree signatures. In: *Cancer informatics* 6 (2008), S. CIN–S680

[Pedregosa u. a. 2011] Pedregosa, Fabian ; Varoquaux, Gaël ; Gramfort, Alexandre ; Michel, Vincent ; Thirion, Bertrand ; Grisel, Olivier ; Blondel, Mathieu ; Prettenhofer, Peter ; Weiss, Ron ; Dubourg, Vincent u. a.: Scikit-learn: Machine learning in Python. In: *the Journal of machine Learning research* 12 (2011), S. 2825–2830

[Peixoto 2014] Peixoto, Tiago P.: The graph-tool python library. (2014)

[Pržulj 2007] Pržulj, Nataša: Biological network comparison using graphlet degree distribution. In: *Bioinformatics* 23 (2007), Nr. 2, S. e177–e183

[Rossi und Zhou 2016] Rossi, Ryan A. ; Zhou, Rong: Hybrid CPU-GPU framework for network motifs. In: *arXiv preprint arXiv:1608.05138* (2016)

[Rousseeuw 1987] Rousseeuw, Peter J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. In: *Journal of computational and applied mathematics* 20 (1987), S. 53–65

[Shervashidze u. a. 2009] Shervashidze, Nino ; Vishwanathan, SVN ; Petri, Tobias ; Mehlhorn, Kurt ; Borgwardt, Karsten: Efficient graphlet kernels for large graph comparison. In: Dyk, David van (Hrsg.) ; Welling, Max (Hrsg.): *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics* Bd. 5, URL http://proceedings.mlr.press/v5/shervashidze09a.html, 16–18 Apr 2009, S. 488–495

[Van Der Walt u. a. 2011] Van Der Walt, Stefan ; Colbert, S C. ; Varoquaux, Gael: The NumPy array: a structure for efficient numerical computation. In: *Computing in science & engineering* 13 (2011), Nr. 2, S. 22–30

[Wale u. a. 2008] Wale, Nikil ; Watson, Ian A. ; Karypis, George: Comparison of descriptor spaces for chemical compound retrieval and classification. In: *Knowledge and Information Systems* 14 (2008), Nr. 3, S. 347–375

[Washio und Motoda 2003] Washio, Takashi ; Motoda, Hiroshi: State of the art of graph-based data mining. In: *Acm Sigkdd Explorations Newsletter* 5 (2003), Nr. 1, S. 59–68

[Ying u. a. 2018] Ying, Rex ; You, Jiaxuan ; Morris, Christopher ; Ren, Xiang ; Hamilton, William L. ; Leskovec, Jure: Hierarchical graph representation learning with differentiable pooling. In: *arXiv preprint arXiv:1806.08804* (2018)

[Zhang u. a. 2018] Zhang, Muhan ; Cui, Zhicheng ; Neumann, Marion ; Chen, Yixin: An end-to-end deep learning architecture for graph classification. In: *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018