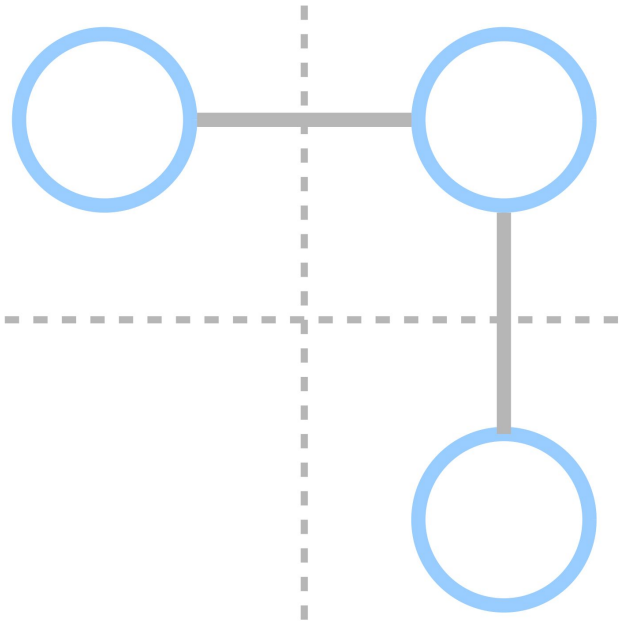


Subgraph Patterns in Spatial Networks

José Carlos Freitas Ferreira
Mestrado em Ciência de Computadores
Departamento de Ciência de Computadores
2022

Orientador
Pedro Ribeiro, Professor Auxiliar,
Faculdade de Ciências da Universidade do Porto


U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Sworn Statement

I, José Carlos Freitas Ferreira, enrolled in the Master Degree in Computer Science at the Faculty of Sciences of the University of Porto hereby declare, in accordance with the provisions of paragraph a) of Article 14 of the Code of Ethical Conduct of the University of Porto, that the content of this dissertation reflects perspectives, research work and my own interpretations at the time of its submission.

By submitting this dissertation, I also declare that it contains the results of my own research work and contributions that have not been previously submitted to this or any other institution.

I further declare that all references to other authors fully comply with the rules of attribution and are referenced in the text by citation and identified in the bibliographic references section. This dissertation does not include any content whose reproduction is protected by copyright laws.

I am aware that the practice of plagiarism and self-plagiarism constitute a form of academic offense.

José Carlos Freitas Ferreira

30/09/2022

Acknowledgements

"Not all those who wander are lost." As someones who wanders a lot, if I am not lost, it is thanks to the amazing people who have been part of my life for the past five years and even before that.

First of all, I would like to thank my family, in particular my parents, Carla and Vitor, and my brother, Miguel, who had to deal with me for the longest time. Thank you for being so understanding of the many evenings I did not spend with you because I was working on this thesis. Secondly, I want to thank all my friends, namely Fábio, Zé, Leão, Mesquita and Yander, I will forever cherish all the years of fun, adventures and friendship we spent together, you guys made and still make everything a lot easier.

I want to thank my supervisors, Pedro Ribeiro and Alberto Barbosa, for their guidance and support throughout this work, I still cannot believe I published a paper, and it was all thanks to you! You kept my head up during the most stressful moments and never doubted me, even when I did it myself.

Last, but most certainly not least, I want to thank Mafalda. Everyone close to me knows that this thesis would not exist if it was not for you. Thank you, for everything.

Thank you everyone, I know how lucky I am to have you all by my side.

Abstract

Numerous domains can be represented as networks. The analysis of those networks allows for the extraction of several conclusions about the systems they represent, and there is a lot of work done with the goal of refining the already existing techniques and introducing new ones so that even more insights can be taken from the same network. One of those techniques is known as Counting Subgraphs, a problem tied to the Subgraph Isomorphism problem, which is known to be NP-Complete. Later, this methodology branched to the discovery of Network Motifs, over-represented subgraphs in a graph.
In this work, we introduce a richer definition for Motifs, spatial Motifs, which in their essence correspond to Motifs with a spatial component. We present a novel concept of how a Spatial Motif can be represented, using a partitioned bounding box of its nodes and assigning each node to the correspondent partition, thus taking into account not only the absolute location of the nodes but their relative position in reference to each other as well.
After that, we tested our concept on networks corresponding to the street layout of cities, where each node is a junction of two streets, and the edges are the streets themselves. Those experiments have the purpose of showing that our methodology is able to distinguish between "grid-like" and "non grid-like" cities, but the concept can be further extended to different kinds of networks and is general enough that different boxes and partitions can be defined. Finally, we make several experiences to further understand the potential of our definition and provide a thorough analysis of the obtained results.

Keywords: Graph Algorithms, Subgraph Counting, Motifs, Spatial Motifs

Resumo

Inúmeros domínios podem ser representados como redes. A análise dessas redes permite a extração de várias conclusões sobre o sistema que estas representam, e muito trabalho tem sido desenvolvido com o objetivo de refinar as técnicas já existentes e introduzir técnicas novas de forma a obter ainda mais informações dessa mesma rede. Uma dessas técnicas é conhecida como Contagem de Subgrafos, e está diretamente ligada ao problema do Isomorfismo de Grafos, um problema NP-Completo conhecido. Esta metodologia ramificou-se para a procura de Network Motifs, subgrafos que estão sobre-representados numa rede.

Neste trabalho, introduzimos uma definição mais rica para Motifs, Motifs Espaciais, que na sua essência correspondem a Motifs com uma componente espacial. Apresentamos um novo conceito para a representação de Motifs Espaciais, usando a delimitação dos seus nós repartida em partições e atribuindo a cada nó a partição correspondente, de forma a ter em conta não só a posição absoluta de cada um dos nós, mas também a sua posição relativa entre os restantes.

Depois, testamos o nosso conceito em redes correspondentes ao esquema das estradas de cidades, em que a junção de duas estradas corresponde a um nó na rede e as arestas correspondem às estradas em si. Esses testes foram desenvolvidos com o intuito de mostrar que a nossa metodologia é capaz de distinguir entre cidades com esquemas em grelha e cidades sem esse tipo de esquema, mas o conceito pode ser extendido para diferentes tipos de redes e é suficientemente geral para que possam ser usadas delimitações e partições diferentes. Por fim, realizamos várias experiências para perceber melhor o potencial da nossa definição, e também fazemos uma análise minuciosa dos resultados obtidos.

Palavras-chave: Algoritmos de Grafos, Contagem de Subgrafos, Motifs, Motifs Espaciais

Contents

List of Tables

List of Figures

Acronyms

SPASM SPatial Arrangements of Side-chains and Main-chain

ESU Enumerate SUbgraph

OSM OpenStreetMap

API Application Programming Interface

HTML Hypertext Markup Language

JSON JavaScript Object Notation

Chapter 1

Introduction

In an ever-evolving world, where technology gets developed by the day, everything around us is more and more connected. When talking about connections, the prominent data structure are *graphs*, that despite being simple models, are capable of storing a lot of information regarding the system they are representing, especially their topology. With this, when a new knowledge extraction method is implemented, many new things can be discovered. It is reported that the first usage of graphs was by Leonhard Euler, in order to solve the Königsberg Bridge Problem [1]. This problem consists of trying to know if it is possible to cross the seven bridges of Königsberg, over the river of Preger, in a single trip. The bridges are depicted in Figure 1.1.

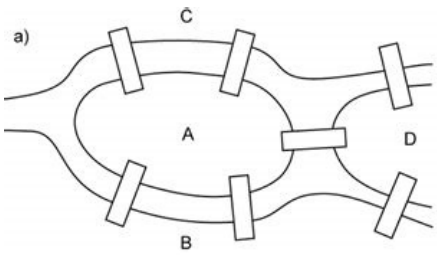


Figure 1.1: Königsberg's Bridges, adapted from [2].

This problem can be solved by creating a multigraph (graph where two nodes can be connected by multiple edges) where the nodes correspond to the letters A, B, C and D in Figure 1.1 and the edges correspond to the bridges, found in Figure 1.2.

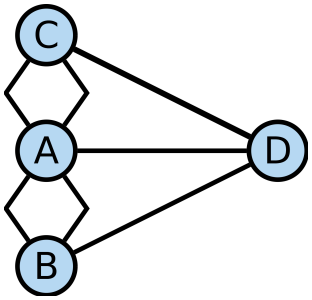


Figure 1.2: Multigraph originated from the Königsberg's Bridges.

Since then, the field of *graph theory* has been in constant progress. More recently, *network science* has emerged as multidisciplinary academic field combining theories and methods from graph theory with areas such as statistics mechanics, data mining and information visualization, aiming to study networks from several domains with non trivial topological characteristics [3]. In fact, *complex networks* are a very powerful abstraction of real-world systems that allow us to analyze their underlying interactions [4]. Many of these systems have a correspondence to the physical world, such as transportation networks (e.g. road, train or subway), power grids or brain networks. Their components are therefore embedded in space and topology alone does not capture all the relevant information [5]. Being able to understand and analyze these spatial networks is therefore a crucial task with multidisciplinary applicability [6, 7].

Subgraphs can be seen as the building blocks of networks and they are the core of rich characterization concepts such as network motifs [8] or graphlet degree distributions [9]. Despite extensions to incorporate dimensions such as weight [10], time [11], color [12] or multiple layers [13], to the best of our knowledge there is no general and widespread subgraph abstraction that incorporates the spatial dimension. We should note that for specific domains there has been some related work, such as in football, where passing networks between different regions of the playing field have been created [14], but these remain specialized and restricted to their own field of study.

1.1 Goals and Contributions

In this thesis, our goal is to aim towards a general concept of spatial motifs able to characterize networks from any domain.

Our first contribution (Section 3.1) is a novel subgraph abstraction that incorporates spatial information in a way that is general enough to incorporate several spatial dimensions (e.g. 2D or 3D) and granularities (e.g. macroscale vs microscale regions). The key idea is to automatically create a spatial partition of the subgraph bounding and to color the nodes according to the region they are in.

Our second contribution (Section 3.2) is an initial methodology and fully functional framework to detect and count these spatial motifs, based on enumerating subgraph occurrences and then computing their spatial and topological type.

Our third and last contribution (Chapter 4) is a thorough experimental section. We first provide a proof of concept analysis of several real-world road networks, showing that unlike purely topological motifs, we can distinguish between grid and non grid-like layouts. Furthermore, we showcase a generalization of our partition method and we use synthetic data to further explore and detail our methodology.

The initial core work of this thesis, including the spatial subgraph abstraction and initial proof of concept with 2×2 partition in road networks, was accepted as a full paper with oral

presentation at the 11th International Conference on Complex Networks and their Applications, held in November 2022 [15].

1.2 Thesis Outline

This thesis structure is as follows:

- **Chapter 2 - Preliminaries** defines the main terminology and notation to be used throughout this document and discusses work previously done that relates to this thesis or that serves as background for what will be discussed in further chapters;
- **Chapter 3 - Development** thoroughly explains the details of our contributions and how the produced program works;
- **Chapter 4 - Results and Analysis** shows the results obtained by our method with multiple datasets of different types as well as their analysis;
- **Chapter 5 - Conclusions and Future Work** states the conclusions we obtained from this work and mentions improvements that could possibly be done in the future that would allow for an extension of what we did in this thesis.

Chapter 2

Preliminaries

In this chapter, we present all the fundamental and necessary concepts so that the reader can have a complete understanding of the entire document. We start by defining the main notation and terminology used throughout this thesis and, after this, we thoroughly explain the concept of spatial networks, colored networks and motifs (the basis of this work), with examples to illustrate and thus allow for a better understanding of said concepts. In particular, we go through the definitions of motif finding, colored motifs and spatial motifs, all of which are frequently used throughout this thesis. Finally, we present *g-tries*, a data structure that we intended to use in the earlier stages of our work. Even though it ended up not being part of our program, we still feel like it is worth referencing as their usage might represent a future improvement to the code.

2.1 Notation and terminology

A **set** S corresponds to a collection of distinct elements. The number of elements of a set, its **cardinality**, is represented as $|S|$.

A **graph** G can be defined as a pair $(V(G), E(G))$, where $V(G)$ is the set of **vertices** of G and $E(G)$ its set of **edges**. Each edge of G is represented by a pair of connected vertices $e = (i, j)$, where $i, j \in V(G)$, and the **size** of G corresponds to its number of vertices, $|V(G)|$. If $|V(G)| = k$, then we call G a k-graph.

Graphs can be either **directed** or **undirected**, and we define a graph as **directed** if the set of edges $E(G)$ is comprised of ordered pairs, that is, $\forall i, j \in V(G), (i, j) \in E(G) \implies (j, i) \in E(G)$, and **undirected** otherwise. In an ordered pair of vertices, we name the first node **origin** and the second **target** (see Figure 2.1).

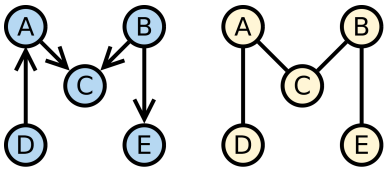


Figure 2.1: Directed graph M and its undirected counterpart.

In a directed graph, the **out-degree** of a node n corresponds to the number of edges with source in n, the **in-degree** to the number of edges whose target is n, and the **degree** to the summation of the in-degree and out-degree. Since in undirected graphs there isn't a notion of source and target, one can only refer to the **degree** of a node.

A **subgraph** S of G is a graph with $V(S) \subseteq V(G)$ and $E(S) \subseteq E(G)$, as seen in Figure 2.2. Note that this subgraph can also be a k-graph.

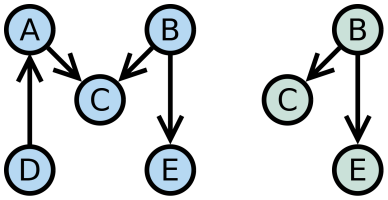


Figure 2.2: Graph M and one of its size three subgraphs.

A **labeled graph** corresponds to an assignment of values to the vertices, edges, or both, subject to certain conditions [16]. There has been thorough research in g*raph labeling* for several years [17] and there are many methods to generate it [18], each with its own characteristics. In Figure 2.3 we can see an example of four graphs, the first one is unlabeled, and each of the others represents a possible labeling of said graph, the last one using colors.

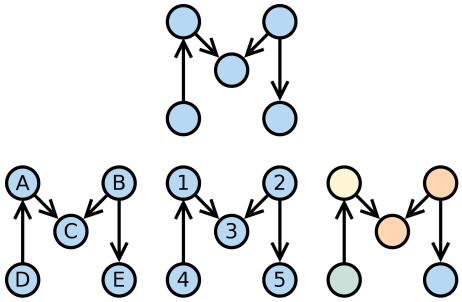


Figure 2.3: Graph M and three of its possible labelings.

As can be seen in the previous figure, labeling is not restricted to strings or integers. Adding this property to graphs also allows us to retrieve more information about its characteristics, like assigning two different nodes to the same category, as happened with the two orange nodes in Figure 2.3.

Two graphs G and G' are said to be **isomorphic**, written $G \equiv G'$, if there is a match between their vertices such that two vertices are connected by an edge in G if and only if their corresponding vertices in G' must also be connected by an edge, depicted in Figure 2.4. It should also be noted that if $G \equiv G'$ then $|V(G)| = |V(G')|$ and $|E(G)| = |E(G')|$. If the mapping that creates the isomorphism also takes into account the nodes' labels, then this definition can be further extended to labeled graphs. Two vertices i and j are said to be **structurally indistinguishable**, written $i \sim j$, if there is an automorphism (an isomorphism from a graph to itself) that maps i to j.

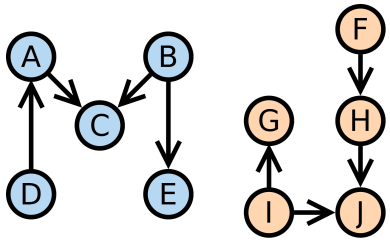


Figure 2.4: Two isomorphic graphs, M and J, with mapping $A \sim H, B \sim I, C \sim J, D \sim F$ and $E \sim G$.

A subgraph S_g is said to be an **induced subgraph** of G if it is formed from a subset of vertices of G and all of the edges that connect pairs of vertices in that subset. Saying that S_g is an induced subgraph of G is interchangeable with saying that G induces S_g. In Figure 2.5, we can see a graph G and two subgraphs, S_1 that is induced by G and S_2 that is not. Note that subgraph S_2 is not induced by G because the edge that connects nodes E and C is not part of S_2, and both S_1 and S_2 are made up of nodes B, C and E.

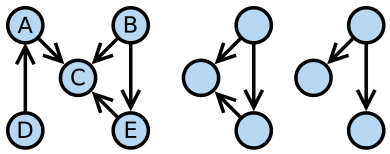


Figure 2.5: Graph G (on the left) and two of its subgraphs, S_1 (in the middle, induced from G) and S_2 (in the right, not induced by G).

The **canonical labeling** of G, also known as the canonical form of G, corresponds to a graph G' isomorphic to G and that represents the whole isomorphism class of G. This definition was described by Piperno [19], but, in our case, the canonical labeling does not correspond to a graph, but to the concatenation of a set of colors with a graph. A further explanation of this adaptation can be found in Section 3.2.3.

The **fingerprint** of a graph is a set of characteristics that allows us to identify a given network. Much like the fingerprints of a person distinguish them from other people, it makes it possible to individually tell apart two graphs, but it can also be used to create some kind of grouping among networks. This concept is used in many areas, such as biology [20] or even the internet, with digital fingerprinting becoming more common by the day [21]. For example, if

we define the fingerprint of a graph to be the number of nodes and the number of edges, then graphs M and J have the same fingerprint, whilst graph C does not, as is shown in Figure 2.6. Here, the fingerprint consists of a set of features, in this case an array with two values, number of nodes and number of edges. Using that definition, graph M's fingerprint is $[5, 4]$, graph J's is $[5, 4]$ and graph C's is $[4, 3]$.

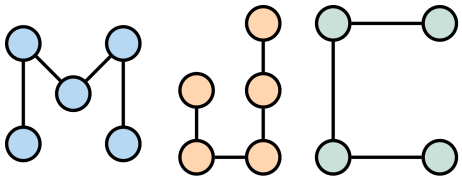


Figure 2.6: Two graphs with the same fingerprint (considering only the number of nodes and edges), M and J, and another with a different one, C.

The **subgraph frequency** essentially corresponds to the number of times a given subgraph occurs in a network. To know if a subgraph has more than one occurrence, we must first define how to identify a match between subgraphs. When there is a set of nodes from a graph G that induce a subgraph G', then G' is said to have a match in G, that is, there is a subgraph S in G that is isomorphic to G'. With this, the frequency of G' is the number of different subgraphs S in G that induce G'. If no node or edge is shared, then two matches are considered different [22]. In an undirected network, there are only two possible connected subgraphs of size 3 to induce, triangle and chain, which means that either the nodes are all connected between themselves or that two of them are not connected. In Figure 2.7 we see subgraph G and the two possible subgraphs, triangle and chain. There is one triangle (formed by nodes B, C and E) and six chains (formed by nodes $\{A, B, C\}$, $\{A, C, D\}$, $\{C, D, E\}$, $\{A, C, E\}$, $\{A, D, E\}$ and $\{B, E, D\}$).

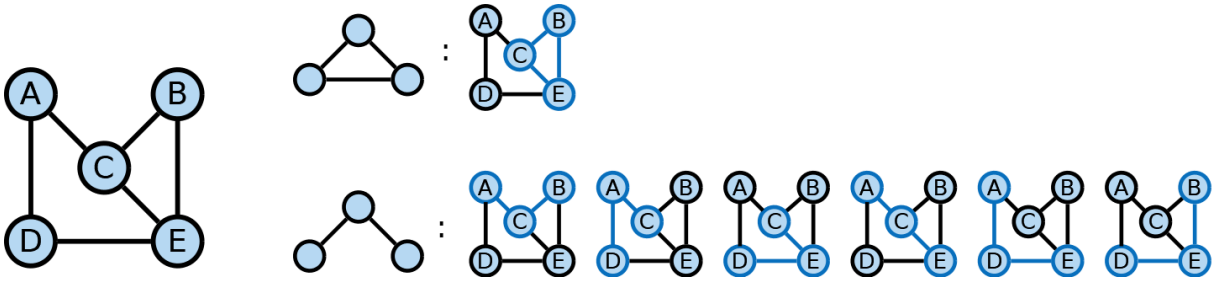


Figure 2.7: Subgraph G and the frequency of its size 3 subgraphs.

A **subgraph census** consists in finding the frequency of all or of a subset of subgraphs of a network [23–25]. This is closely tied to the *Subgraph Isomorphism Problem* (which is known to be NP-complete [26]), which consists on, given two graphs G and H, trying to determine if there exists a subgraph in G isomorphic to H. A lot of work has been developed regarding this problem, including the usage of constraint programming [27] and dynamic programming [28]. In the case of subgraph census, we are not only trying to find if an isomorphism exists, we are also counting its occurrences, which makes this problem even harder.

2.2 Spatial Networks

A typical graph is defined by its vertices and edges, but other factors, such as their absolute locations or their relative location towards each other, might also be relevant, as is the example of power grids or neural networks [5]. A lot of work concerning *urban street patterns* has been done using the typical relational networks [29], but it was proven that by using spatial networks, that is, ones with a spatial property, some features like the differentiation between planned and self-organized cities is much more evident [30]. The importance of the spatial component in a network is illustrated in Figure 2.8.

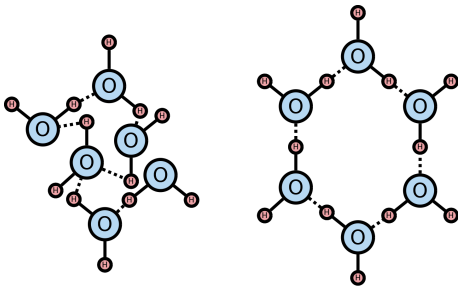


Figure 2.8: Simplified water (left) and ice (right) molecular structure.

We can see that in order to distinguish water from ice on a molecular level, the topology alone is not enough, and we need to also take into account the location of the atoms, as ice is known to have a hexagonal molecular structure, which is responsible for the higher volume with the same mass as water. Note that in reality this molecule has a third dimension, but for the sake of simplicity we presented it here in 2D as an approximation illustrating an idea.

The spatial component of a network might be interpreted in different ways. In some cases, the positioning might be considered absolute, such as for instance on a power grid or road network, where the location of the nodes might be represented by its earth coordinates. However, in other cases, such as in a molecule or a brain network, the location of a node is relative to the locations of other nodes, as there is no clear external origin reference.

2.3 Colored Networks

Colored networks simply correspond to networks where either the nodes, the edges or both of them are labeled with a color. There are many applications to this type of network, and one of the most well known is *graph coloring problem*. It is one of Karp's original 21 NP-complete problems [31], having many algorithmic contributions that try to attenuate its hardness [32–34]. For a given network, the problem consists in trying to find the least number of colors needed so that no two adjacent nodes share the same color. This is for instance the concept used behind register allocation in compilers. Figure 2.9 shows an example of a graph , M, colored in different ways.

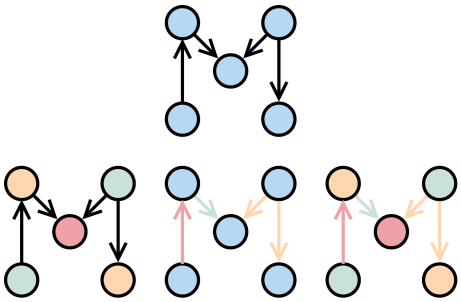


Figure 2.9: Graph M and some of its colored counterparts.

Colored networks allow us to represent categorical properties of the nodes or edges of a network. Note that the color does not necessarily mean a "real" color, and it is just a representation of a label that we use to aggregate groups of nodes or edges. If we have a college network, we might want to distinguish nodes that represent students, teaching and non-teaching staff, thus coloring the nodes accordingly, or maybe we have a money transaction network and want to distinguish if a transaction between two nodes is a payment or an income, coloring the edges as needed.

2.4 Network Motifs

First introduced by R. Milo *et al.* [8], *network motifs* correspond to subgraphs that are over-represented in a network. A subgraph is considered to be over-represented if it is far more frequent in a given network than it would be in a randomized similar one. Figure 2.10 shows an example of a motif (for the sake of simplicity, for the rest of this work we will use the term *motif* to denote a *network motif*).

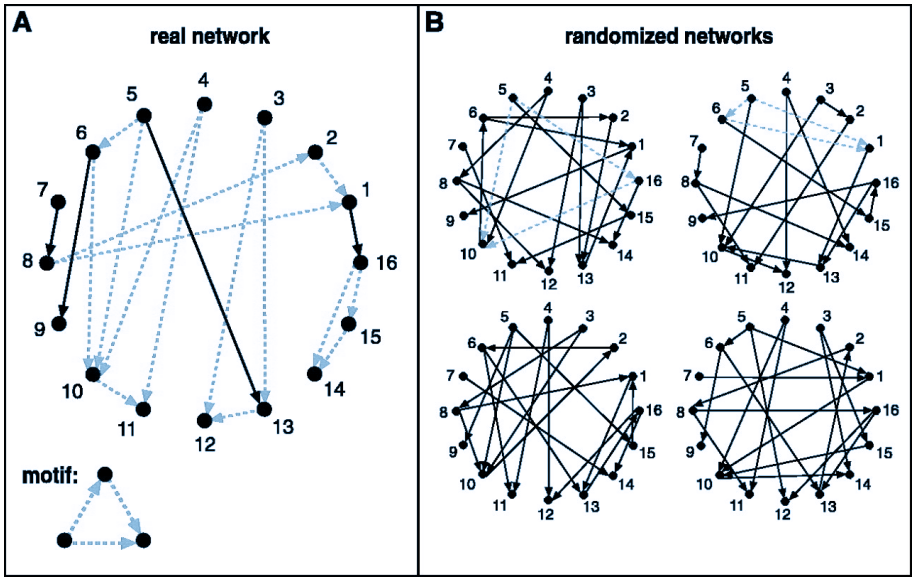


Figure 2.10: Schematic view of network motif detection, adapted from [8].

In the case of Figure 2.10, the motif is represented by the small triangle in the bottom left corner of the image. This particular directed subgraph is known as *feed forward loop*, and consists of three nodes connected in a chain, one of them having an in-degree of two, another in-degree one and out-degree one and the third one out-degree two. In the right image, we can observe (painted in blue) five occurrences of that subgraph. When we compare that frequency with the four randomized networks on the left, we notice that each of them contains at most one occurrence of the subgraph, and the two bottom ones don't have any occurrence of it. Since the four randomized networks present a similar structure to the real network, namely the degree distribution, we could conclude that in the real network that subgraph is over-represented, thus corresponding to a motif. Here, the randomized networks work as null models, allowing us to verify that the subgraph is indeed over-represented in the real network, and not simply common in similar networks.

The motif distribution can be used as a fingerprint or to define classes of networks, since it was proven that networks that shared some kind of hidden structure also shared the same over-representation of motifs [35]. This kind of structure is known to have been very helpful in many different areas such as biology [36, 37], electronics [38], the web [39], social networks [40] and even neurology [41]. Typically, motifs are used in unweighted networks, but they can also be applied to weighted ones. Applying an approach that uses weighted networks to find motifs, using the geometric mean of edge weights and the ratio of their geometric to their arithmetic mean, changes the results obtained for some financial and metabolic networks [42]. A method was proposed to mine motifs in weighted networks without the need to generate random graphs (which makes it more efficient), whilst having improved accuracy [10].

2.4.1 Motif Finding

In order to understand if a subgraph is over-represented, we must first define what the expected frequency is, and for that a null model of random graphs is chosen, and the frequency of the given subgraph is then calculated for this set of graphs. A null model consists of a model of a network that allows for the generation of data under the null hypothesis (that is, all associations are random), in order to help in the identification of nonrandom patterns on other networks [43]. The default approach for the generation of the null model is to keep the degree sequence [44, 45], but a model that generates it by maintaining the frequency of subgraphs of size $k-1$, where k is the size of the motifs to be found, has been proven to have significantly better results [46]. As stated before, the frequency of a subgraph is needed to determine if a subgraph is a motif or not. In order to obtain the frequency, one must first find the subgraph, that is, we have to perform a subgraph census on the network. There are many subgraph census algorithms, most of them already adapted for the counting of subgraphs, and they can be divided essentially into three categories: exact subgraph counting, approximate subgraph counting and parallel subgraph counting. More on those algorithms can be found in [47]. Another kind of motif discovery that has been gaining traction over the past few years is the search for temporal motifs, that is, motifs that incorporate a chronological component in their definition [11].

To the best of our knowledge, there are seven main exact subgraph counting algorithms that rely on their enumeration [48], which will be crucial for our implementation as will be seen later. In this work, our first approach was to use **G-tries** (more on this data structure in Subsection 2.5), but ended up using **FanMod**, also known as **ESU** (more information can be found in Subsection 3.2.2). Nonetheless, we believe it is pertinent to mention the other algorithms since even though they were not directly used in our implementation, they represent progress in the motif finding field. With that in mind, below we can see a quick overview of the algorithms not mentioned in further sections.

- **Grochow**: This algorithm was first introduced in 2007 by Grochow [49] and is a motif-centric approach. This means that the frequency of a subgraph is obtained by searching all possible mappings of that individual subgraph on the network. Grochow named those subgraphs *query graphs*.
- **MFinder**: In contrast, MFinder [50] samples a k-subgraph simply by randomly appending edges until the subgraph has size k.
- **MODA**: Just like Grochow, MODA [51] is motif-centric. It uses an *expansion tree* to extract motifs of size k. This is a hierarchical structure that allows for the enumeration of induced and non-induced subgraphs simultaneously.
- **QuateXelero**: QuateXelero [52] is very similar to ESU, but it uses a quaternary tree (a tree where each node has four children at most) in the enumeration process. The goal of the algorithm was to simplify the *nauty* process (details regarding *nauty* can be found in Subsection 3.2.3).
- **Kavosh**: Finally, we have Kavosh [53], a network-centric approach. The process consists of four main steps: discovery of all k-subgraphs of a network, classification into isomorphic groups, generation of random networks using the input network, and motif finding identification.

Due to the complexity of the problem at hand, there have been recent approaches that incorporate machine learning techniques in motif discovery. In particular, deep learning has been applied to the problem of subgraph counting with promising results [54], which might lead the way to further improvements on this topic.

2.4.2 Colored Motifs

Most of the research on network motifs was focused only on the structure of the motif, that is, every node and edge have the same type and no distinguishing is done between them. Without the use of a label, like colors, there is the possibility that some valuable information is lost. Even though there was already some work done with colored nodes [55–57] and there was proof that also coloring edges could provide a lot of insight [58], there were no attempts of implementing

the discovery of fully colored motifs, and with this in mind, Ribeiro *et al.* [12] proposed an algorithm that was able to do so, with the use of g-tries. As the name suggests, *a colored motif* is the combination of two previously defined concepts: network motifs and colored networks. Essentially, by adding a color property to the default definition of motif, some new information can be retrieved. In Figure 2.11, we can observe the same network depicted in Figure 2.10 but with colored nodes. On the top two networks, we can see the same number of occurrences of the *feed forward loop* motif. If we now say that the motif must contain an orange node with in-degree two, a green node with in-degree one and out-degree one and a yellow node with in-degree two, we verify that the number of occurrences is very different, as the two bottom images show.

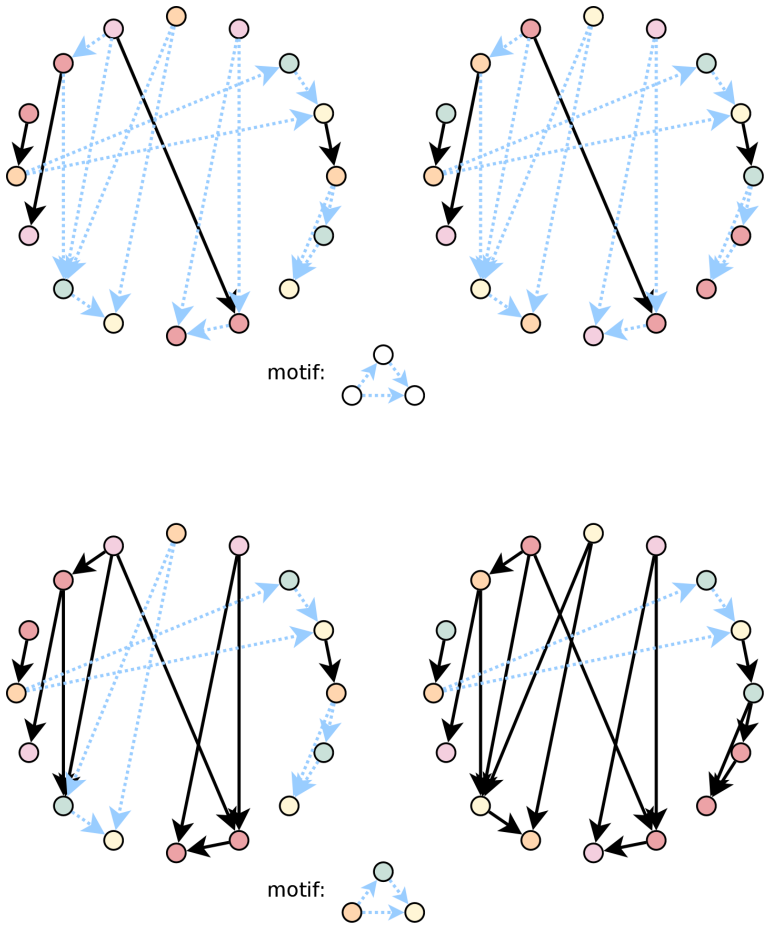


Figure 2.11: Difference between regular motifs and colored motifs.

2.4.3 Spatial Motifs

Spatial motifs are motifs present in spatial networks, which means that the spatial component of the network will also be a defining feature for the motifs, and they have been applied in the recognition of protein structures [59]. When it comes to the search of motifs in this kind of structure, the most widely known software is SPatial Arrangements of Side-chains and Main-chain (SPASM), a program that searches for subgraph matches with a user-defined motif in a given structural database, where the structures are represented as networks. This algorithm uses

depth-first search with pruning as its main component to find subgraphs, and the nodes in the motif are represented as the C^{α} atom (the first carbon atom that attaches to a functional group, what causes the molecule's characteristic chemical reaction) of the correspondent residue and the centre of gravity of its side-chain atoms, and the spatial component derives from the calculation of the distances between those atoms. Another well known program used in protein structure motifs is RIGOR, and unlike SPASM, it searches for the occurrences of a single protein structure in a database of pre-defined motifs. To the extent of our knowledge, there are no other implicit applications of spatial motifs in different fields. To better illustrate the difference between spatial motifs and colored motifs, Figure 2.12 recreates Figure 2.11 for that purpose.

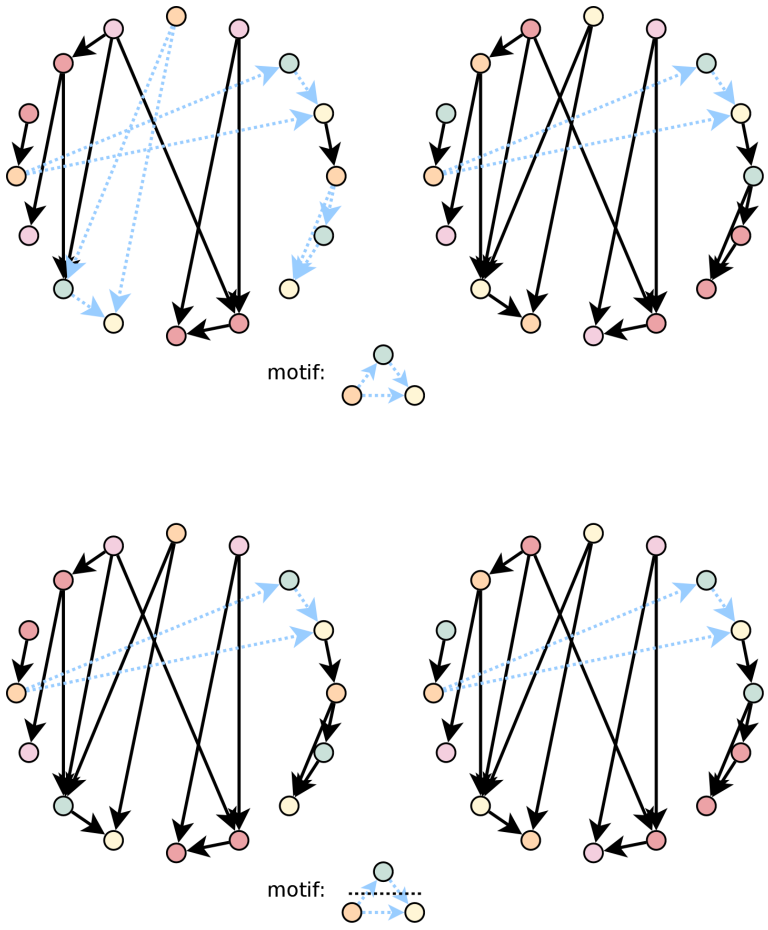


Figure 2.12: Difference between colored motifs and spatial motifs.

In Figure 2.12, in the bottom part of the image we add a new property to the subgraph we are trying to find. Besides requiring that the subgraphs contains an orange node with in-degree two, a green node with in-degree one and out-degree one and a yellow node with in-degree two, we also take into account the position of the green node: it must have a higher y-axis value compared to the other two, considering a Cartesian Coordinate System. With this, we see that once again, the subgraphs contain the same amount of subgraphs that follow those characteristics. This is relevant since if it was known to be a motif, its counting would depend on the spatial component.

2.5 G-tries

G-tries are a data structure designed to find and count subgraphs [60, 61]. They share the same core to their implementation as prefix trees [62] (or tries), as they use a common topology to create a tree where nodes with the same ancestor share a substructure. An example of a prefix tree can be found in Figure 2.13.

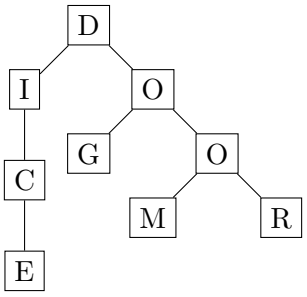


Figure 2.13: Prefix tree for the words dice, dog, doom and door.

When referring to a prefix tree, a path from the root to a leaf node defines a substring, with g-tries that path defines a subgraph. Below, in Figure 2.14, an example g-trie for six graphs of size four can be seen.

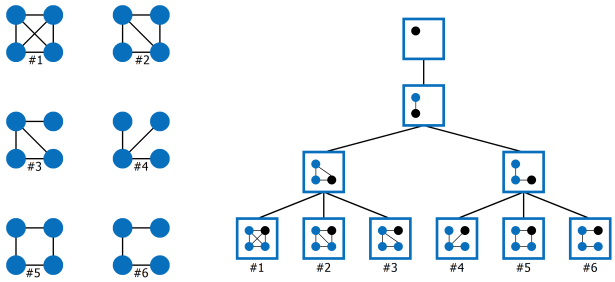


Figure 2.14: G-trie example, adapted from [63].

Unlike network-centric [53, 64, 65] or subgraph-centric approaches [66], g-tries implement what is called a set-centric approach, this means that the structure will search for a set of sub-graphs of fixed size, instead of trying to find all the subgraphs of the network or only one. When we first started developing our work, the idea was to use g-tries in the subgraph enumeration portion, but the complexity of adapting the structure to our concept ended up being too high, and so we resorted to another method, described in Section 3.2.

Chapter 3

Development

In this chapter we introduce a novel concept of *spatial motifs*, that allows us to take into account not only the absolute position of nodes but also their relative positions in reference to one another, with the use of a bounding box of the graph and its subsequent partitioning. We also present a fully functional framework that uses an algorithm that performs subgraph enumeration using that same approach, thoroughly explaining each of its steps and external tools used.

3.1 A Novel Concept of Spatial Motifs

There are several possible ways to express spatial properties. For instance, the distance between nodes can be used as edge weight [67], but this would not take into account the relative position of the nodes. Another option would be to rely on angles between nodes, hence losing the distance information. Our approach relies on first creating a *bounding box* around the found subgraph using the nodes spatial location, and then partitioning this box into regular-sized regions, thus taking into account both the relative position and the distance between nodes.

For the sake of simplicity and given the space constraints, we will mainly focus on a two dimensional example divided into 2×2 and later on 3×3 partitions, but as explained later, our approach is general and extends naturally to higher dimensions. The creation of the bounding box is straight-forward: we require a set of coordinates for each node on the input, and for each found subgraph, we calculate the maximum and the minimum of both the x and the y values, which gives us the limits of our box, as depicted in Figure 3.1.

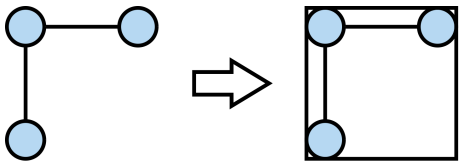


Figure 3.1: Bounding box creation: calculation of the box limits.

Then, we simply calculate the relative position of each node when referring to the center of

the bounding box, as seen in Figure 3.2, assigning the node to a partition of the box on the form of a color. In the case of a 2×2 partitioning, each partition will correspond to a quadrant of the bounding box.

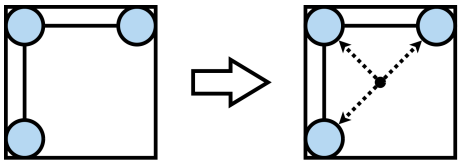


Figure 3.2: Bounding box creation: assignment of nodes to partitions.

The division of the bounding box into equally sized partitions can easily be mathematically defined. Let min_x, min_y be the minimum for both the x and y values of the nodes respectively and max_x, max_y the maximum of those values. To divide an axis in k points and to get the point in position n, that is, of coordinates (x_n, y_n), $\forall n : 0 \leq n \leq k$ we can apply Equation 3.1.

$$(x_n, y_n) = (min_x + \frac{n}{k}(max_x - min_x), min_y + \frac{n}{k}(max_y - min_y)) \tag{3.1}$$

Using Equation 3.1, we calculate all points in each line of the bounding box that will serve as the endpoints of each of the line segments forming the partition of said bounding box. After that, we just connect the dots in non-adjacent lines in a way that the resulting line segments together form a grid-like partition of the bounding box.

Below, in Figure 3.3, we can see an example of how to divide the plane in a 3×3 manner, with $min_x = 2$, $max_x = 20$, $min_y = 4$ and $max_y = 16$.

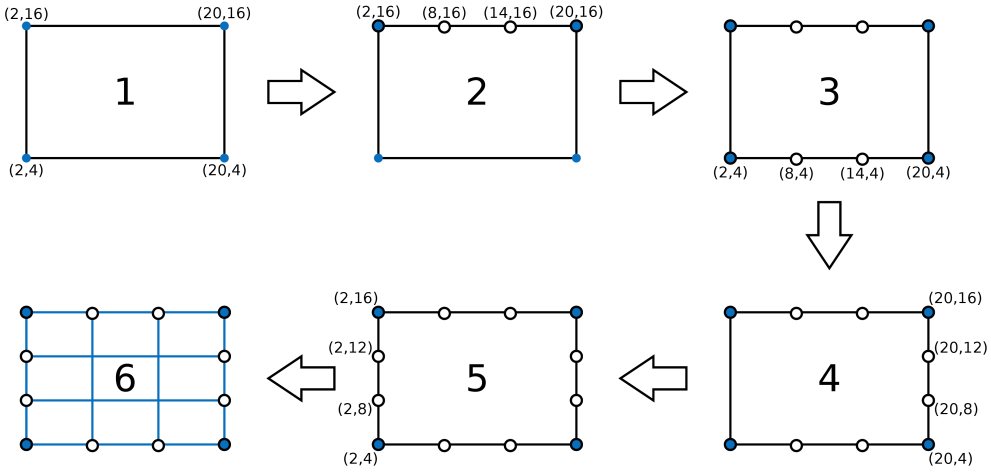


Figure 3.3: Division of the plane in a three by three manner.

- In the first box we can see the four extreme points of the box, formed by min_x, max_x, min_y and max_y.
- In box two, we start by calculating the division point in the upper axis of the box. By applying Equation 3.1, with the parameters $min_x = 2$, $max_x = 20$, $min_y = 16$, $max_y = 16$

and $k = 3$, we get the following points:

$$(x_0, y_0) = (2 + \frac{0}{3}(20 - 2), 16 + \frac{0}{3}(16 - 16)) = (2, 16)$$
$$(x_1, y_1) = (2 + \frac{1}{3}(20 - 2), 16 + \frac{1}{3}(16 - 16)) = (8, 16)$$
$$(x_2, y_2) = (2 + \frac{2}{3}(20 - 2), 16 + \frac{2}{3}(16 - 16)) = (14, 16)$$
$$(x_3, y_3) = (2 + \frac{3}{3}(20 - 2), 16 + \frac{3}{3}(16 - 16)) = (20, 16)$$

- In the third box we calculate the points in the lower axis of the box, using the parameters $min_x = 2$, $max_x = 20$, $min_y = 4$, $max_y = 4$ and $k = 3$, getting the points:

$$(x_0, y_0) = (2 + \frac{0}{3}(20 - 2), 4 + \frac{0}{3}(4 - 4)) = (2, 4)$$
$$(x_1, y_1) = (2 + \frac{1}{3}(20 - 2), 4 + \frac{1}{3}(4 - 4)) = (8, 4)$$
$$(x_2, y_2) = (2 + \frac{2}{3}(20 - 2), 4 + \frac{2}{3}(4 - 4)) = (14, 4)$$
$$(x_3, y_3) = (2 + \frac{3}{3}(20 - 2), 4 + \frac{3}{3}(4 - 4)) = (20, 4)$$

- In the fourth box, for the right axis, we use the parameters $min_x = 20$, $max_x = 20$, $min_y = 4$, $max_y = 16$ and $k = 3$, thus getting:

$$(x_0, y_0) = (20 + \frac{0}{3}(20 - 20), 4 + \frac{0}{3}(16 - 4)) = (20, 4)$$
$$(x_1, y_1) = (20 + \frac{1}{3}(20 - 20), 4 + \frac{1}{3}(16 - 4)) = (20, 8)$$
$$(x_2, y_2) = (20 + \frac{2}{3}(20 - 20), 4 + \frac{2}{3}(16 - 4)) = (20, 12)$$
$$(x_3, y_3) = (20 + \frac{3}{3}(20 - 20), 4 + \frac{3}{3}(16 - 4)) = (20, 16)$$

- The forth box corresponds to the discovery of the left axis points, with the parameters $min_x = 2$, $max_x = 2$, $min_y = 4$, $max_y = 16$ and $k = 3$, that gives us the points:

$$(x_0, y_0) = (2 + \frac{0}{3}(2 - 2), 4 + \frac{0}{3}(16 - 4)) = (2, 4)$$
$$(x_1, y_1) = (2 + \frac{1}{3}(2 - 2), 4 + \frac{1}{3}(16 - 4)) = (2, 8)$$
$$(x_2, y_2) = (2 + \frac{2}{3}(2 - 2), 4 + \frac{2}{3}(16 - 4)) = (2, 12)$$
$$(x_3, y_3) = (2 + \frac{3}{3}(2 - 2), 4 + \frac{3}{3}(16 - 4)) = (2, 16)$$

- Finally, the last box connects the corresponding points, giving us the 9 partitions of the bounding box.

With this, it is trivial to infer what partition a node belongs to, simply by looking at the

nodes coordinates and the partitions edges.

An example of a graph and all the 3-subgraphs it contains, each in their respective 2 by 2 bounding box, can be seen in Figure 3.4, where the original spatial network is given above, in the blue nodes, and all its three node spatial subgraph occurrences are given below.

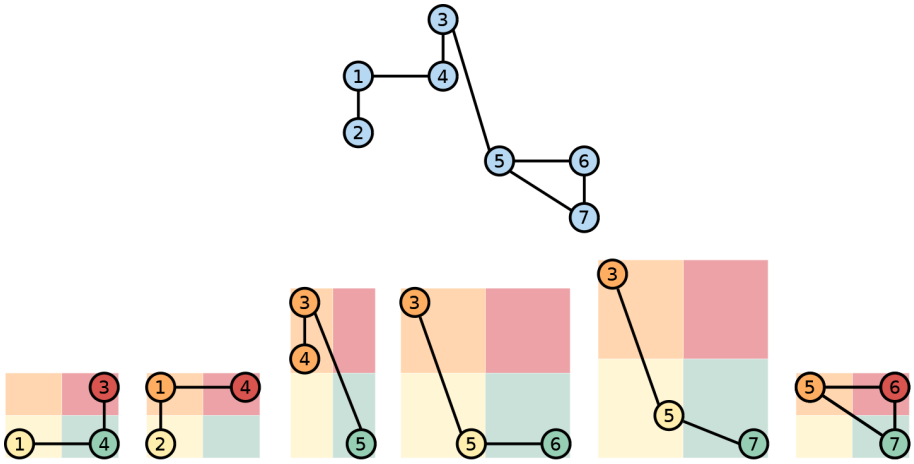


Figure 3.4: Example of subgraphs with spatial coloring by 2D quadrants.

Taking into account the spatial dimension of the nodes in the previous figure (Figure 3.4), we can enumerate five different subgraph types, being the fourth ($\{3, 5, 6\}$) and the fifth occurrence ($\{3, 5, 7\}$) of the same type: they both have three nodes in the same quadrants (one orange, one yellow and one green) and the same connections (one orange-yellow edge and another yellow-green edge).

By contrast, if only purely topological properties were used, there would exist only two subgraph types, as depicted in Figure 3.5, with the first five occurrences being a chain of three nodes and the last one ($\{5, 6, 7\}$) being a triangle.

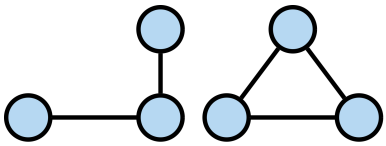


Figure 3.5: Chain (type A) and Triangle (type B) topological subgraphs.

The above example already illustrates how much richer our spatial representation is, but we would like to emphasize how general our conceptual approach is. From a scale point of view, it naturally extends to higher numbers of nodes (just consider more nodes in each subgraph). From a topological point of view, it is also able to organically integrate features such as direction (just consider that when distinguishing between different isomorphic types). From a granularity point of view, we can also consider any regular division. Here we exemplified with 2×2 and 3×3 partitions, but we could use any $n \times n$ partition, depending on what we want to measure (and moreover we could even use on the same analysis subgraph occurrences at different n sizes to create a richer set of features). Finally, our approach also naturally extends to higher dimensions (for instance, in a 3D space one could use $2 \times 2 \times 2$ octants as the equivalent of 2D quadrants).

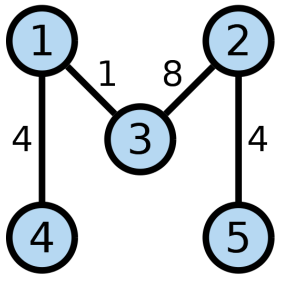
3.2 Finding and Counting Spatial Motifs

In this section we explain our methodology for finding and counting the occurrences of spatial motifs as defined in the previous section. The motivation for counting will become clearer in Section 4.2, but essentially by computing subgraph frequencies we are able to obtain numerical features characterizing the underlying network. Counting subgraphs is therefore a core network analysis primitive. A fully detailed survey on how to count purely topological motifs can be found in [47], including approximate and parallel approaches.

Our proposed initial approach has two steps: (i) we first enumerate all subgraph occurrences of a given size k, obtaining sets of k connected nodes; (ii) for each occurrence we identify its spatial type by producing a canonical labeling that is unique to each colored isomorphic type.

3.2.1 The Data Structure

In order to store the data, we created our own Graph implementation. It contains four fields: A Boolean value that indicates whether the graph is directed or undirected (the first option is represented by a 1 and the latter by a 0), a matrix corresponding to the graphs adjacency matrix and two numbers that store the number of nodes and the number of edges in the Graph. The adjacency matrix is a two dimensional array that in each cell holds a value that represents a connection (or the lack of it) between two nodes identified by the row and column values. We wanted to have a future proof program, and so instead of in each cell simply storing a Boolean to check if two nodes are connected, we store a number that corresponds to the weight of that connection. Figure 3.6 shows a graph, M, and its adjacency matrix according to our representation.



	1	2	3	4	5
1	\\	0	1	4	0
2	0	\\	8	0	4
3	1	8	\\	0	0
4	4	0	0	\\	0
5	0	4	0	0	\\

Figure 3.6: Graph M and its corresponding adjacency matrix.

Note that since we are working with an undirected graph, the adjacency matrix is symmetric, and because we do not currently allow self loops, the main diagonal is invalid. A value of 0 represents the lack of a connection.

3.2.2 Enumerating Subgraph Occurrences

In order to enumerate the occurrences of subgraphs with k nodes, we opted to use Enumerate SUbgraph (ESU) [68], a general purpose subgraph enumeration algorithm capable of finding each occurrence only once, avoiding symmetries. The choice of using ESU was based essentially on how well known the algorithm is and how simple it would be to adapt it to our specific needs. In short, this is done by starting from a single node and expanding from there, using only vertices that have an index (label at the original graph) greater than that of the original node and that can be neighbors of a newly added vertex but not of any other one previously added. In Figure 3.7 we illustrate this process with a small example for $k = 3$ and an original network of six nodes. Inside each tree node box we indicate two node sets: first the current subgraph being enumerated ($V_{subgraph}$) and secondly the set of nodes which can expand it ($V_{extension}$).

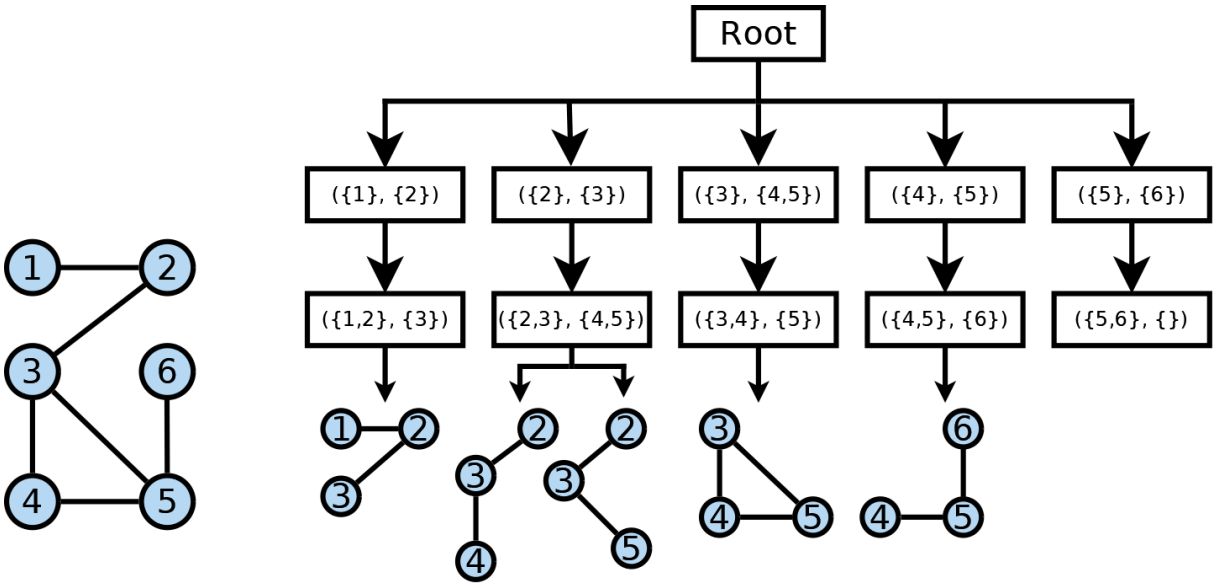


Figure 3.7: Example of an ESU search tree for k-subgraph enumeration with $k = 3$.

The root of the search tree is a starting point to evaluate the subgraph. Its children, on the second level, correspond essentially to one branch per node, with the extension sets being their immediate neighbors with a larger index than the node itself. For instance, the second branch contains $V_{subgraph} = \{2\}$ and $V_{extension} = \{3\}$ (3 is a neighbor of 2 and 1 is not considered since $1 < 2$ and the subgraph with $\{1, 2\}$ would be already considered in the first branch). This process continues in the following tree levels: we add 3 to $V_{subgraph}$ and since it has two neighbors that meet the requirements, those are added to $V_{extension}$, resulting in $V_{subgraph} = \{2, 3\}$ and $V_{extension} = \{4, 5\}$. Now we have two possible branches, $V_{subgraph} = 2, 3, 4$ and $V_{subgraph} = 2, 3, 5$. In both these cases we have $|V_{subgraph}| = 3$ and we have reach the desired node set size.

After doing this to every single node we end up with the subgraphs of size 3 represented on the leafs of the tree. The required conditions for a node to be added to $V_{extension}$ make sure that no subgraph is found twice.

Below we can see the pseudocode for the ESU algorithm[68].

Algorithm 1 ESU algorithm

Input A graph $G = (V, E)$ and an integer $1 \leq k \leq |V|$.
Output All size-k subgraphs

1: **procedure** STARTESU
2: **for** each vertex $v \in V$ **do**
3: $V_{extension} \leftarrow u \in N(\{v\}) : u > v$
4: EXTENDSUBGRAPH$(\{v\}, V_{extension}, v)$
5: **end for**
6: **end procedure**
7: **procedure** EXTENDSUBGRAPH$(V_{subgraph}, V_{extension}, v)$
8: **if** $|V_{subgraph}| = k$ **then**
9: **return** $G[V_{subgraph}]$
10: **end if**
11: **while** $|V_{subgraph}| \neq 0$ **do**
12: Remove an arbitratly chosen vertex w from $V_{extension}$
13: $V'_{extension} \leftarrow V_{extension} \cup \{u \in N_{excl}(w, V_{subgraph}) : u > v\}$
14: EXTENDSUBGRAPH$(V_{subgraph} \cup \{w\}, V'_{extension}, v)$
15: **end while**
16: **return**
17: **end procedure**

3.2.3 Subgraph Types and Canonical Labeling

After having the node sets that correspond to each subgraph occurrence, we still need to discover the spatial type of each one, so that we can increment its frequency. For instance, as we could observe in Figure 3.4, the fourth and fifth subgraphs belong to the same type.

In our approach, we first determine the bounding box of each occurrence by computing the minimum and maximum values of each spatial dimension. We then partition the box into the desired number of regions and we "color" the nodes according to the region in which each one lies, effectively obtaining what could be considered a colored subgraph [12]. Afterwards, we compute a canonical labeling such that two subgraph occurrences will have the same labeling if and only if they correspond to the same (colored) isomorphic type.

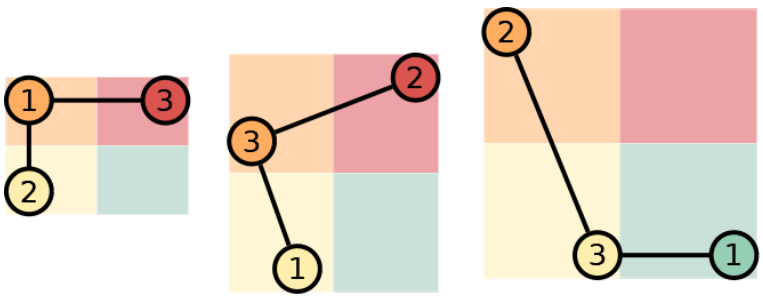


Figure 3.8: The first two subgraphs are of the same type and should have the same canonical labeling; the third one should have a different labeling.

Figure 3.8 illustrates the need for a canonical labeling that takes node colors into account. From a purely topological point of view, all three subgraphs are chains and therefore indistinguishable.

However, when incorporating spatial information, this is not the case. We want the first and second subgraphs to have the same label, as they have the same colored topological properties: one node in each quadrant except the fourth one (represented by the colors orange, red and yellow), and two connections (an orange-red edge and an orange-yellow edge). The third subgraph has different spatial properties that correspond to different colored nodes and edges.

In general, even without colors, computing canonical labelings is a very hard computational task, closely related to the graph isomorphism problem [69]. We therefore resorted to `nauty` [70], a third-party and very efficient set of procedures to determine the automorphism group of a vertex-colored graph. Since `nauty` has built-in support for colored nodes, a call to the default method with the required arguments and the quadrant as color is enough to give us the canonical label. To achieve a labeling using colors, `nauty` requires the colors to be given in some order, and the edge labels will be returned in the order the colors were provided, that is, first the edges with the first color, then the ones with the second color, and so on. There are several parameters that the default method requires, which we will now explain. A standard call takes 12 arguments: *g*, *lab*, *ptn*, *active*, *orbits*, *options*, *stats*, *workspace*, *worksize*, *m*, *n* and finally *canong*. To better understand how we use this program, each of these parameteres is described below:

- *g*: The input graph, that is, the one we are labeling;
- *lab* and *ptn*: We are grouping these two parameters as they are dependent on each other to work. They are both of size*n* and their usage depends on the options set in the *options* parameter. In our case, they define the initial coloring of the graph. After the call, the value of *lab* will contain the vertices of *g* in the order that they need to be relabelled so that *canong* is achieved;
- *active*: This is an array of *m* setwords specifying the initially active colors, but `nauty` does not actually require this argument, and so we pass it the value `NULL`;
- *orbits*: An array that holds the automorphism group of the graph. This is a write-only parameter and since we do not require the automorphism group of the graph, we do not use it;
- *options*: This constitutes the default options to use. Besides the default options, we override some options that are needed in order to work with colors and to obtain the canonical labeling. We set `getcanon` to true in order to retrieve the canonical labeling, `writeautoms` to false as we don't need the automorphism group generators and `defaultptn` also set to false and we intend to define different colors to each node, and by letting this value be true, `nauty` will assume every node has the same color;
- *stats*: This structure provides some statistics regarding the operations performed by `nauty` under the hood;
- *workspace* and *worksize*: These two values correspond to the address and length of working storage array used by `nauty`;

- m: The number of setwords in sets;
- n: The number of vertices;
- *canong*: This corresponds to the canonically labelled g.

As the other values are somewhat constant, independently of what subgraph we are analyzing, let us focus on the generation of g, *lab* and *ptn*. Since our implementation of the graph is different from the one required by `nauty`, we must first parse ours into *nauty*'s. We did this by traversing our adjacency matrix and adding the graph rows and elements using functions provided by `nauty`. Let `adjm` be our adjacency matrix, `n` the number of vertices of the graph, `m` the setwords needed and g the final graph form. The parsing can be seen below in Algorithm 2.

Algorithm 2 Graph Parsing Algorithm

```
1: procedure PARSE
2:     adjm ← GETADJENCYMATRIX()
3:     n ← GETVERTICES()
4:     m ← GETSETWORDS()
5:     for i ← 0 to n do
6:         gv ← GRAPHROW(g, i, m)
7:         EMPTYSET(gv, m)
8:         for j ← 0 to n do
9:             if adjm[i][j] then
10:                ADDELEMENT(gv, j)
11:            end if
12:        end for
13:    end for
14: end procedure
```

With this, we have the required graph, that is, our g. The generation of *lab* and *ptn* is not as trivial. In order to obtain the canonical labeling with colors, *lab* must contain the vertices in some order and *ptn* the division into colors. If `ptn[i]=0`, that means a color class ends at that vertex. For instance, if we have the following color partition: $[\{0, 8\}, \{1\}, \{2, 3, 4, 5, 6\}, \{7, 9, 10\}]$ in which nodes in the same set have the same color, then one possible definition of *lab* and *ptn* would be `lab=[0,8,1,2,3,4,5,6,7,9,10]` and `ptn=[1,0,0,1,1,1,1,0,1,1,0]`. Because we can not define what the colors are exactly, the canonical labelling we use is not only defined by the one returned by `nauty`. Even though we provide the colors always in the same order, that does not mean that two different graphs will not have the same canonical labeling if the only thing they differ in is the colors themselves, and not the number of colors or connections. As an example, take the two graphs in Figure 3.9. Even if we try to sort the nodes by the same color order (Red, Orange, Yellow, Green), we still end up with the same labeling. This is because, to `nauty`, both subgraphs have three nodes, of three different colors, having two with one connection and another with two connections.

In order to mitigate this unwanted behaviour, we always append the colors of the nodes, (with a consistent order throughout all the subgraphs) in the beginning of the label.

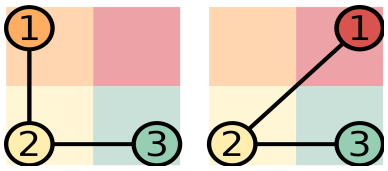


Figure 3.9: Example of two subgraphs that would generate the same canonical labelling, even using `nauty`.

Using the example in Figure 3.9, `nauty` would generate the label 010101010 for both subgraphs. Since in the left images of Figure 3.9 the nodes are in quadrants 2, 3 and 4 and in the right image the nodes are in quadrants 1, 3 and 4, then by adding the colors at the beginning of the original label will generate the labels $234 \cup 010101010 = 234010101010$ and $134 \cup 010101010 = 134010101010$ (where the symbol $\cup$ represents concatenation), thus distinguishing the subgraphs. It should be noted that `nauty` produces the same labeling because the adjacency matrix of the three nodes is the same. In reality, the label corresponds to the different rows of the adjacency matrix concatenated. Both subgraphs in Figure 3.9 have the adjacency matrix represented in Figure 3.10.

	1	2	3
1	0	1	0
2	1	0	1
3	0	1	0

Figure 3.10: Adjacency matrix of subgraphs in Figure 3.9.

3.2.4 Subgraph Output

Once our program finishes the subgraph enumeration process, the output is sent to a file in a JavaScript Object Notation (JSON) format, to be read by a motif visualization program, described in Subsection 3.2.5. It consists of an object with three attributes: the name of the network, an indication of whether the network is directed or not and an array of subgraph structures. Those structures are, again, an object, but have more complexity. They contain the subgraphs label, their nodes, edges, number of occurrences and the relative frequency of their count. Each node stores information regarding the ID of the node, the color it should be painted with and its position in a Cartesian Coordinate System. Finally, the edges simply consist of a pair of values that represent the connections between nodes.

As an example, for the undirected graph M shown in Figure 2.1, the subgraph $\{A, B, C\}$ the following code is generated:


```
"subgraphs": [
  {
     "label": 138001001110,
     "nodes": [
        {
           "id": 0,
           "color": 1,
           "pos": [3,19]
        },
        {
           "id": 1,
           "color": 3,
           "pos": [19,19]
        },
        {
           "id": 2,
           "color": 8,
           "pos": [11,3]
        }
     ],
     "edges": [
        [0,2],
        [1,2],
        [2,0],
        [2,1]
     ],
     "occurrences": 1,
     "percentage": 0.333333
  }
```


3.2.5 Motif Visualization

In order to better visualize the results obtained in the previous sections, we developed a simple script that generates a visualization of the subgraphs found, along with a label (to allow for an easier comparison between outputs of different cities), the number of occurrences of said subgraph and its relative frequency. The input of this script is the output defined in Subsection 3.2.4.

We keep a database of all previously found subgraphs and assign a numeric label to each subgraph that is found: if the subgraph was already found before, it keeps the same number, otherwise a new entry is created. With this approach, even if the size of the subgraphs we are trying to find or the number of partitions increases, the labeling is always possible. This is of the utmost importance, as manually labeling each possible subgraph would be very complex when those values increase. For a better visualization, let us consider a size 3, a size 4 and a size 5 subgraph without taking into account our approach (that further increases the number of

possible permutations). If all the graphs are directed, the total number of possible subgraphs of size 3 is 13, but when we increase that number to 4, then the total is 199, and with 5 nodes we already have 9364 possible subgraphs [71], and using our proposed coloring, this number increases even more drastically.

Every time we run our program, an Hypertext Markup Language (HTML) page is created with a button to switch the visualization order. The subgraphs found can either be sorted by label or by number of occurrences, as shown in Figures 3.11 and 3.12 that represent the first 8 subgraphs found in Espinho sorted by occurrences and label respectively. It should be noted that the depiction of the subgraph is not based on their actual spatial arrangement, but on the generic class representative subgraph. Depending on the type of bounding box, the representative subgraphs of the main classes are described in Figure 4.5 for a 2×2 bounding box or in Figures 4.12 and 4.13 for the 3×3 case. More on this can be found in Sections 4.2 and 4.3.

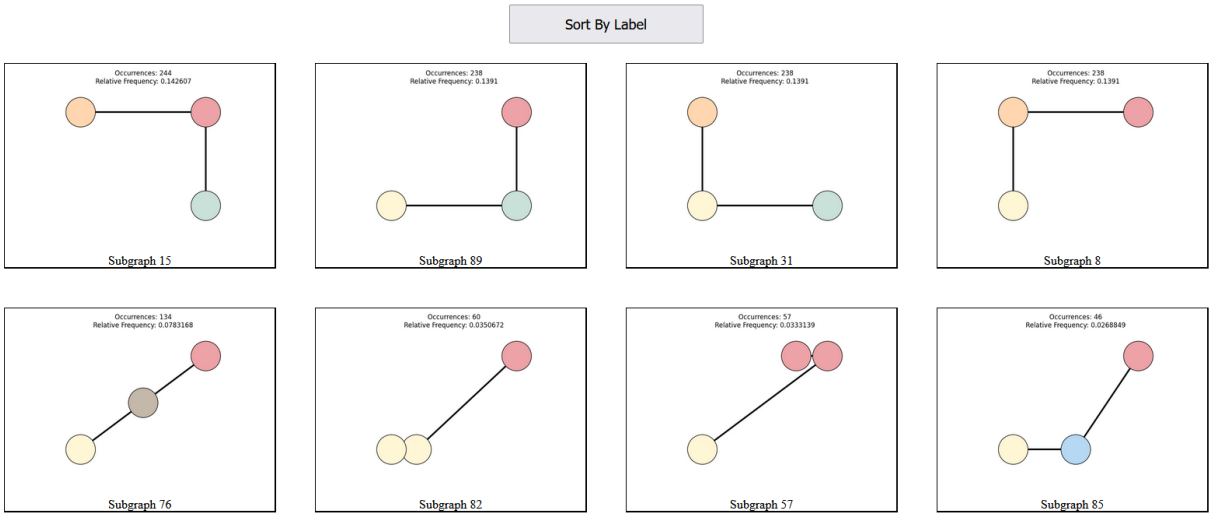


Figure 3.11: Visualization of subgraphs found sorted by number of occurrences.

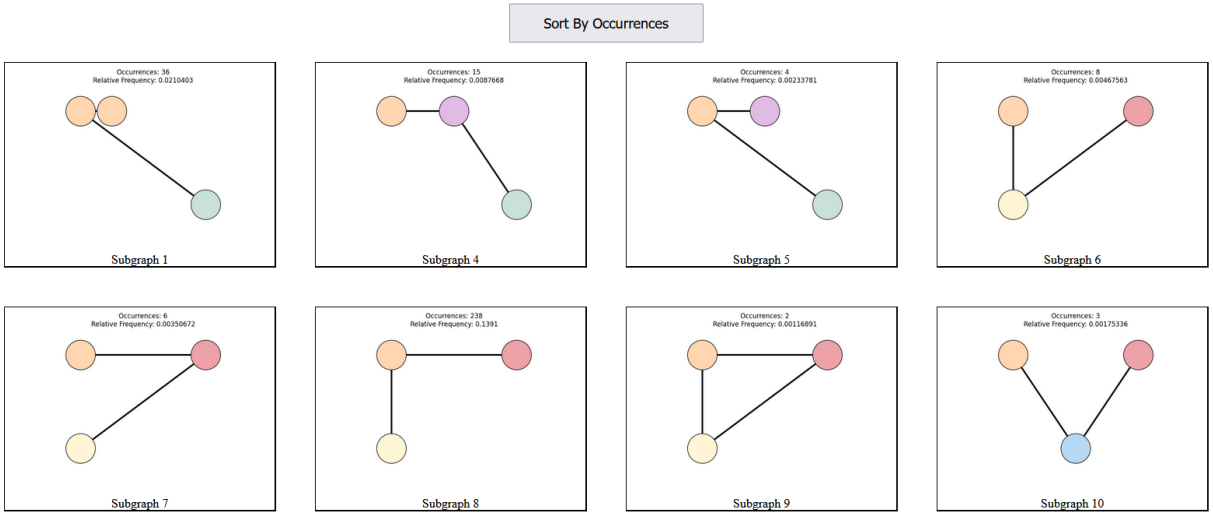


Figure 3.12: Visualization of subgraphs found sorted by label.

Chapter 4

Results and Analysis

In this chapter, we describe the experiences we performed in order to validate our concept, as well as the results obtained from said experiences. First, in Section 4.1, we start by showing how we obtained the main datasets used in the experiences. After this, in Section 4.2 we describe experiments performed using a 2×2 bounding box, followed by experiments with a bigger box, of size 3×3 in Section 4.3. In Section 4.4 we perform some additional tests in different cities with different layouts, in order to take further conclusions from our work. Section 4.5 shows a stress test we performed on our program using different portions of Tokyo as input. Finally, Section 4.7 shows experiments using synthetic data. All the experiments performed in this section used the same computer, with an Intel Core i7-8750H processor, an NVIDIA GeForce GTX 1060 with 6GB of memory size and 16GB of RAM.

4.1 Dataset

In order to test our implementation and showcase the applicability of our proposed subgraph abstraction, we now provide a proof of concept. We use real world road networks, which can be considered one of the quintessential spatial network examples to which everyone can relate to. We selected several cities with very different layouts, but we will be focusing on the analysis of two cities with "grid-like" street layouts (Espinho, Portugal and Detroit, USA) and two with "non grid-like" layouts (Porto, Portugal and Oxford, UK), aiming to distinguish between these two layout groups through the usage of our proposed approach. It is worth noting that we did not use the entirety of the cities as input, since that would bring around a lot of noise to the input. Instead, we selected parts of the city that represent what we want to show in the results.

Our original source of raw street data was OpenStreetMap (OSM) [72], which is a collaborative project that aims to provide a free editable geographical database of the entire world. The process of collecting the data is straight forward: we do a query to get all the points that belong to streets with two sets of coordinates, corresponding to the top left and the bottom right corners of the area we wish to analyze but, as with any real dataset, we had to clean the input

before using it in our program, this includes fixing incorrect, duplicate and even incomplete data, which in our case translated to removing nodes that have degree 0, as those would never form a subgraph of size more than 1, and removing duplicate nodes. The input our code requires consists of a Boolean that indicates if we want an undirected (value 0) or a directed (value 1) graph. After this, an integer v (the number of nodes in the graph) must be provided, followed by v lines containing the coordinates of said nodes. Then, an integer e, the number of edges and e consecutive lines containing three integers (ID from the origin node, ID from the destination node and weight of the connection) should be given. It is worth noting that we do not currently support weighted edges, but this part of the feature was already implemented for future work. The way we retrieved data from OSM gives us a list of a data structure known as `ways`. Each `way` has `nodes` that represent points where a street either connects to another or has a change in direction, and each `node` has two properties, `lat` and `lon` that correspond to its latitude and longitude respectively. Figure 4.1 shows an example of what would be the `nodes` of a `way` in OSM, each of them being represented with a blue circle.

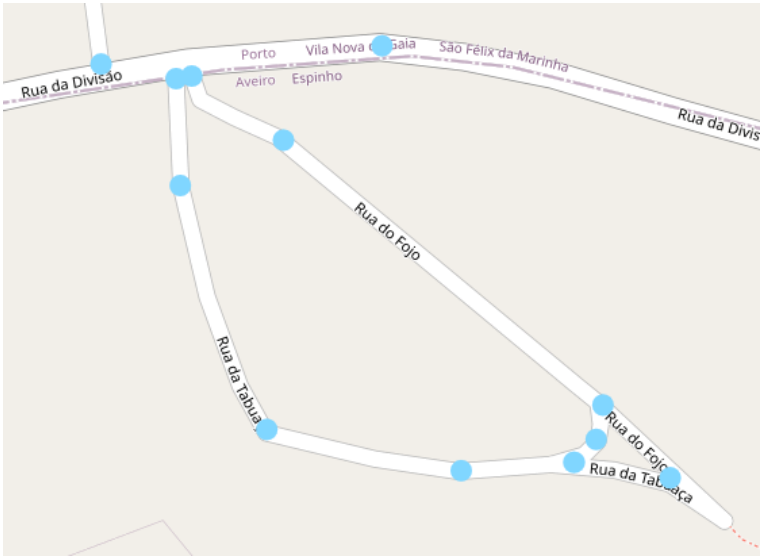


Figure 4.1: Representation of `nodes` in OSM.

As we can see from Figure 4.1, if we take as an example Rua da Tabuaça, only three `nodes` are, in fact, needed, and the total number of nodes found is six. To mitigate this, we created a function that traverses the `nodes` of all the `ways` found and discards the ones that are simple turning points in the streets. We always consider the first and last `nodes`, as they are always an endpoint of the street, and thus must correspond to an edge in the graph. Our next step is to traverse all of them and remove the ones that only appear once, since if they are not a limit of the `way`, in order to appear only once they must be a turning point of the street. This can be seen in Algorithm 3. Note that the set *valid* represents the nodes that will be used in our code as input, as those are the only ones that actually represent a junction between streets (or one of their ending points).

Algorithm 3 Get Valid Nodes Algorithm

```
1: procedure GetValidNodes
2:     counter ← {}
3:     valid ← {}
4:     for way in result.ways do
5:         for id,node in way.nodes do
6:             if id = 0 or id =getSize(way.nodes) then
7:                 valid ← addNode(node.lat, node.lon)
8:             else
9:                 counter[(node.lat, node.lon)] + +
10:                if counter[(node.lat, node.lon)] > 0 then
11:                    valid ← addNode(node.lat, node.lon)
12:                end if
13:            end if
14:        end for
15:    end for
16: end procedure
```

To generate the edges, we traverse the `results` object once again, but this time we connect consecutive valid `nodes`, as is shown in Algorithm 4, always checking that the `nodes` to evaluate are valid.

Algorithm 4 Get Edges Algorithm.

```
1: procedure GetEdges
2:     edges ← {}
3:     for way in result.ways do
4:         for id,node in way.nodes do
5:             if isValid((node.lat, node.lon)) then
6:                 for i ← id + 1 to getSize(way.nodes) do
7:                     newNode ← (way.nodes[i].lat, way.nodes[i].lon)
8:                     if isValid(newNode) then
9:                         edges ← addEdge((node.lat, node.lon), newNode)
10:                    end if
11:                end for
12:            end if
13:        end for
14:    end for
15: end procedure
```

In Figure 4.2, we show an image of the approximate area used for each of the cities mentioned above, taken from OSM, which clearly indicates the nature of the road layouts that are being analyzed.

(a) Espinho

(b) (Northwest) Detroit

(c) Porto

(d) Oxford

Figure 4.2: Layout of the four cities used as input, taken from OSM and accessed in August 31st, 2022.

In Table 4.1 we can see the characteristics of the four networks, as well as the time the program took to compute the results (without taking into consideration input and output operations).

City	Nodes	Edges	Processing Time (s)
Espinho	520	788	0.0684531
Detroit	16029	24773	33.5919
Porto	2040	3206	0.476797
Oxford	1711	2213	0.303629

Table 4.1: Network characteristics of Espinho, Detroit, Porto and Oxford

4.2 Experimental results with a 2x2 Box

From the raw data we create a network in which the nodes are true road intersections and edges represent roads between them (this implied the creation of an automated script that given a geographical bounding box will extract all OSM features from it, which are further processed and simplified to create the desired intersection network).

In this network, we fix the subgraph size to $k = 3$ and count the number of occurrences of each spatial subgraph. Figure 4.3 exemplifies this process for a small portion of a map, illustrating the extracted network and all its occurring subgraph occurrences, some of them belonging to the same spatial type. For instance, $\{1, 2, 3\}$ and $\{3, 4, 5\}$ are of the same type, and the same can be said for $\{1, 2, 7\}$ and $\{3, 4, 6\}$.

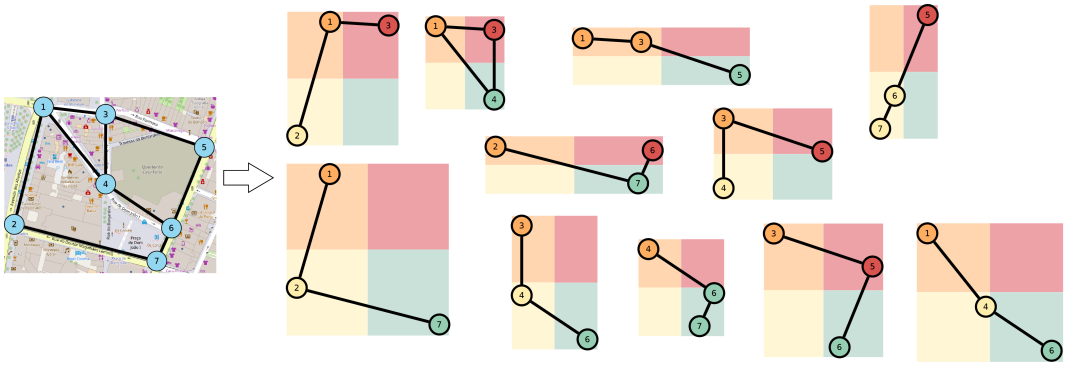


Figure 4.3: The network corresponding to a map and its subgraph enumeration.

To better understand and visualize the differences in subgraph occurrences, we opted to further divide spatial types into *classes* (families of subgraphs), that corresponds to the four 90 degrees rotations of the same simple type. Figure 4.4 illustrates this concept and one class of subgraphs.

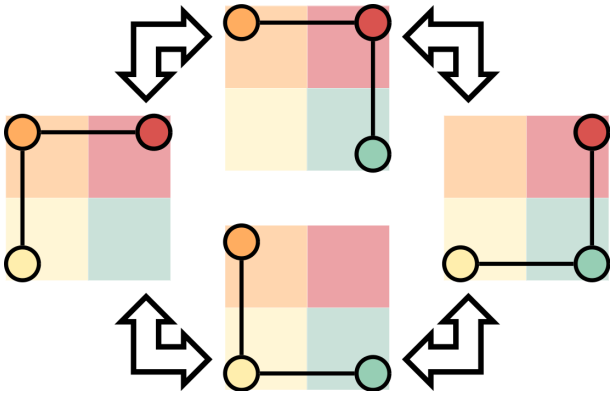
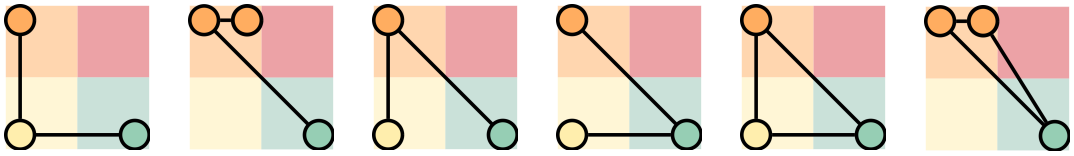


Figure 4.4: Subgraph class 1 and its four spatial subgraph types, corresponding to 90 degree rotations.

Figure 4.5 illustrates representatives of the six most frequent classes of subgraphs that we found in the studied road networks (the frequency of the other possible classes is residual and

their very low relative frequency does not impact the conclusions of our analysis).



(a) Class 1 (b) Class 2 (c) Class 3 (d) Class 4 (e) Class 5 (f) Class 6

Figure 4.5: Representatives of the most frequent classes of subgraphs considered.

It should be noted that not all permutations of colors are possible. If we take the example in Figure 4.6, we can see that if a bounding box were to have two green nodes and one red, then it must be wrong, as the limits of the bounding box would imply the existence of a fourth node *stretching* it outside of the box generated by nodes $1, 2$ and 3 in the figure.

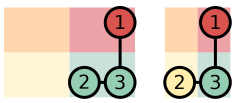
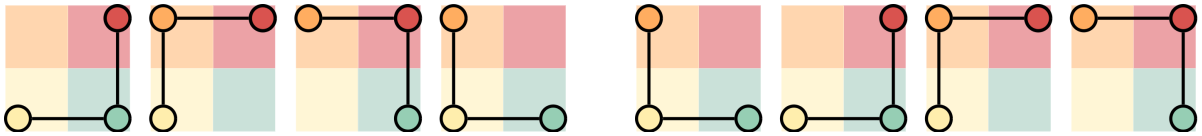


Figure 4.6: Incorrect bounding box (left) and its correct counterpart (right).

4.2.1 Results for "grid-like" street layouts

Figure 4.7 represents the top-4 (in order, from left to right) of the subgraphs with most occurrences in Espinho and Detroit. The results are as expected and capture the grid-like nature of the layout. Even if the exact order of simple types is not exactly the same, this top-4 corresponds to class 1 subgraphs, whose representation resembles a right angle, that is, where each node is in a different quadrant and the subgraph is a chain that connects nodes in consecutive quadrants.



(a) Espinho (b) Detroit

Figure 4.7: Top-4 of subgraphs with most occurrences in the two grid-like cities. Note that all the subgraphs are of class 1 (as defined in Figure 4.5).

The difference in frequency from the 5th to the 4th most common subgraph is noticeable, particularly in the case of Espinho. In both cases, subgraph types from the 5th to the 8th positions are of class 2. Tables 4.2 and 4.3 give more detail on the results, showing the relative frequency (percentage of total occurrences) of the 10 most common subgraph types and their associated class. In total, there were 22 different subgraph types found for Espinho and 32 for Detroit.

Subgraph Rank	Subgraph Type	Relative Frequency
1	1	0.178258
2	1	0.167738
3	1	0.161894
4	1	0.160140
5	2	0.089421
6	2	0.080070
7	2	0.046756
8	2	0.044418
9	4	0.010520
10	3	0.009351
Top-10 total	—	0.948567

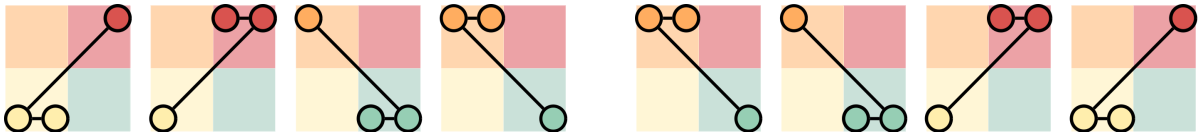
Table 4.2: Spatial subgraph frequencies in Espinho

Subgraph Rank	Subgraph Type	Relative Frequency
1	1	0.139369
2	1	0.138028
3	1	0.135329
4	1	0.134110
5	2	0.108859
6	2	0.097870
7	2	0.086585
8	2	0.083102
9	3	0.019452
10	3	0.019208
Top-10 total	—	0.961913

Table 4.3: Spatial subgraph frequencies in Detroit

4.2.2 Results for "non grid-like" street layouts

On the other hand, if the city does not have a well defined grid layout, we can observe that the most frequent subgraphs are very different, as can be seen in Figure 4.8. In fact, the top-4 for these two cities does not have any subgraph type in common with the grid-like cities. Here, the topological chain type of subgraph is still the most common, which means that if only the topological information of the subgraphs was used, conclusions with this level of detail would not be possible, but in this case instead of having each node in a different quadrant, two nodes share a quadrant, there is a connection between them, and one of them connects to a node in the opposite quadrant, with the entire top-4 of most frequent subgraphs being of the same class.



(a) Porto (b) Oxford

Figure 4.8: Top-4 of subgraphs with most occurrences in the two non grid-like cities. Note that all the subgraphs are of class 2 (as defined in Figure 4.5).

As in the previous section, we give detailed results of the top-10 most common subgraph types in Tables 4.4 and 4.5. Again there is a noticeable increase in frequency from the 5th to the 4th most common subgraph, and the same class appears from rank 5 to rank 8. In total, there were 28 different subgraphs found for both Porto and Oxford.

Subgraph Rank	Subgraph Type	Relative Frequency
1	2	0.133904
2	2	0.131815
3	2	0.118655
4	2	0.118446
5	1	0.081888
6	1	0.080426
7	1	0.079590
8	1	0.075621
9	3	0.023605
10	3	0.022143
Top-10 total	—	0.866095

Table 4.4: Spatial subgraph frequencies in Porto

Subgraph Rank	Subgraph Type	Relative Frequency
1	2	0.149768
2	2	0.148675
3	2	0.133916
4	2	0.121618
5	1	0.074064
6	1	0.071331
7	1	0.070511
8	1	0.066684
9	4	0.024050
10	4	0.022683
Top-10 total	—	0.883302

Table 4.5: Spatial Subgraph frequencies in Oxford

4.2.3 Comparison between cities

In Figure 4.9 we can observe a bar chart of the relative frequency of each subgraph class per city mentioned in the previous sections, with the usage of their spatial properties.

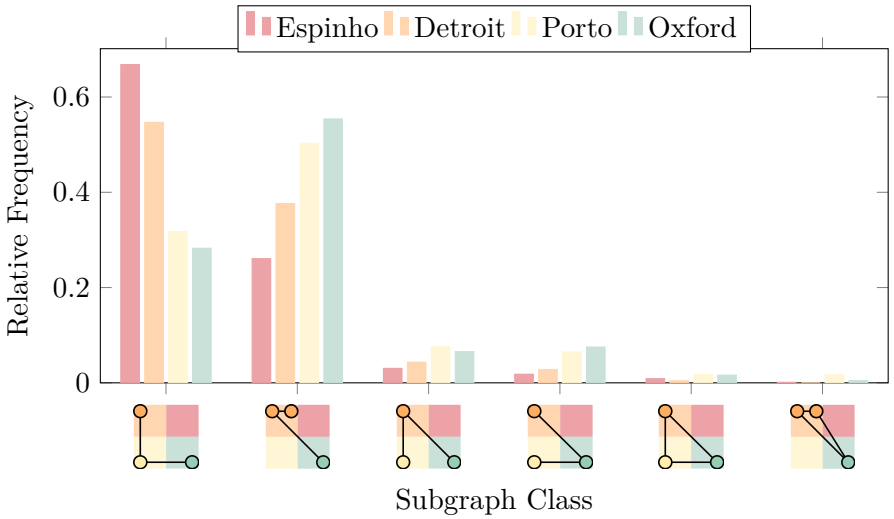


Figure 4.9: Spatial subgraph fingerprint of each of the studied cities.

If we remove the spatial component from the subgraphs, we are left with only two types of subgraphs: chains and triangles. This means that spatial classes 1 to 4 will be of the single topological type A (chain), and classes 5 and 6 will be of the topological type B (triangle). Using the same data as the previous plot but ignoring the spatial properties results in the bar chart depicted in Figure 4.10.

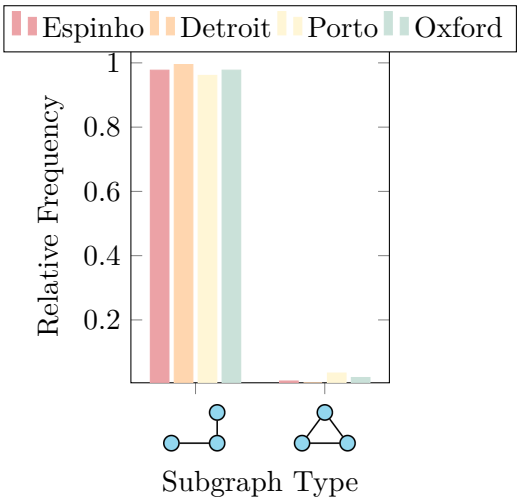


Figure 4.10: Purely topological subgraph fingerprint of each of the studied studies.

We can observe from Figure 4.9 that the distribution of the subgraph classes clearly shows a predominance of class 1 in the two grid-like cities, whilst class 2 is more common in the two cities without this layout, which allows us to easily distinguish them, using only this distribution. Conversely, using the data from Figure 4.10 it is not possible to make that distinction, as all cities show a clear dominance of type A subgraphs with around the same difference in frequency when compared to type B, which is really uncommon in street networks. It is also interesting to note that cities without a grid layout have a bigger frequency of other types of subgraphs other than classes 1 and 2, even though those types are still by far the most relevant.

As a final note, we would like to remark that using a normal laptop the subgraph counting and labeling phase takes less than a minute to compute even in the largest considered network (Detroit, with 16 029 nodes and 24 773 edges). A potential drawback of our proposed strategy is that increasing the size k of the subgraph will inevitably lead to an exponential growth of the number of subgraph occurrences and hence on the execution time. However, in this paper we were mainly concerned with proving that the concept could be useful and there are still many improvements that can be made regarding efficiency.

4.3 Using a Bounding Box with a different number of partitions

In order to get more results, we extended our code to be capable of handling different partition numbers for the bounding box. After this, we ran the same input as the previous section but for a 3×3 bounding box. In Figure 4.11 we can see the difference in the resulting subgraphs between using a 2×2 and a 3×3 bounding box. Note that only the first, second and sixth maintain the same properties.

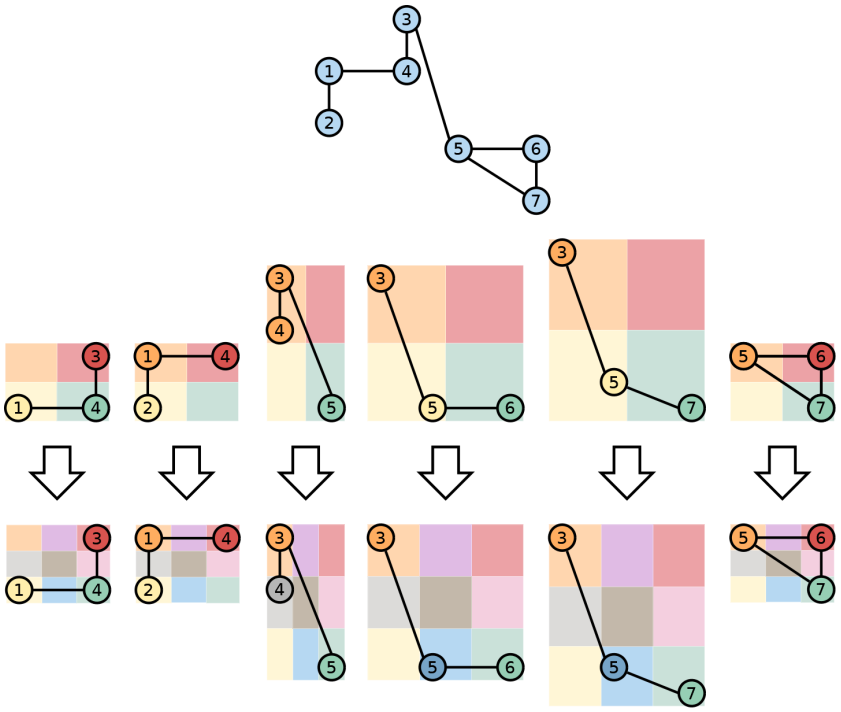
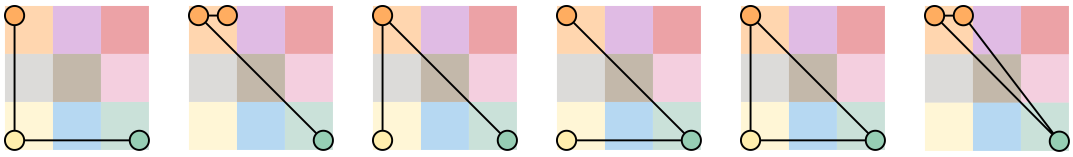


Figure 4.11: Resulting subgraphs from using a 2×2 and a 3×3 bounding box.

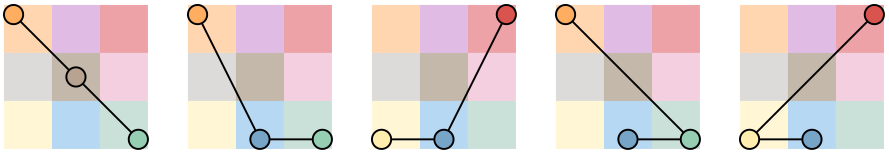
With the use of more partitions, comes the need to define more classes for the subgraphs found. With that in mind, we made an extension to the classes defined in Figure 4.5. Before creating new classes, there was the need to adapt the 2×2 classes to the new model. Figure 4.12 reflects that adaptation.



(a) Class 1 (b) Class 2 (c) Class 3 (d) Class 4 (e) Class 5 (f) Class 6

Figure 4.12: Classes of Figure 4.5 adapted to a 3×3 bounding box.

In Figure 4.13, we can see the new representatives of the new classes. Note that once again the elements of a class correspond to a 90^{o} rotations of another member of that class.



(a) Class 7 (b) Class 8 (c) Class 9 (d) Class 10 (e) Class 11

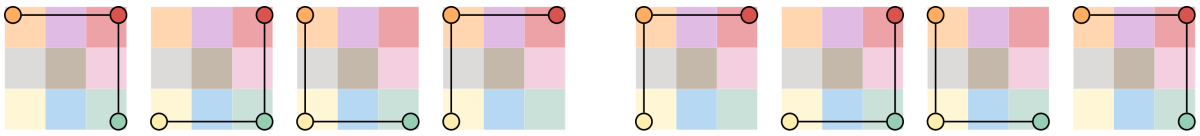
Figure 4.13: Newly defined classes of subgraphs.

As stated in Section 4.2, not all permutations of colors are possible. From this point, every time subgraph classes are mentioned, we are referring to the ones in Figures 4.12 and 4.13, unless

stated otherwise.

4.3.1 Results for "grid-like" street layouts

Following the same logic as in the figures of Section 4.2, we will be now showing the top four subgraphs found for both Espinho and Detroit, ordered from left to right with the rightmost being the most frequent subgraph.



(a) Espinho (b) Detroit

Figure 4.14: Top-4 of subgraphs with most occurrences in the two grid-like cities. Note that all the subgraphs are of class 1 (as defined in Figure 4.12).

As we can see in Figure 4.14, all the top four subgraphs are still of class 1, as was verified in Figure 4.7. Since there is a direct mapping from the class 1 presented in Figure 4.5 to the one shown in Figure 4.12 and all the properties are kept between the two of them, we can say that the top subgraphs maintained the same class. The relative frequency of those subgraphs can be found in Tables 4.6 and 4.7.

Subgraph Rank	Subgraph Type	Relative Frequency
1	1	0.142607
2	1	0.139100
3	1	0.139100
4	1	0.139100
Top-4 total	—	0.559907

Table 4.6: Spatial subgraph frequencies in Espinho (3×3 bounding box).

Subgraph Rank	Subgraph Type	Relative Frequency
1	1	0.112707
2	1	0.111436
3	1	0.109364
4	1	0.109242
Top-4 total	—	0.442749

Table 4.7: Spatial subgraph frequencies in Detroit (3×3 bounding box).

It is worth noting that the ordering amongst the ranking of each city varied when using a 2×2 and a 3×3 bounding box, but this is due to the increase in the granularity of the results. In Figure 4.15, we can see an example of a subgraph that did not preserve the colors of its nodes when the bounding box with more partitions was used, and thus now begins to a distinct class of subgraph.

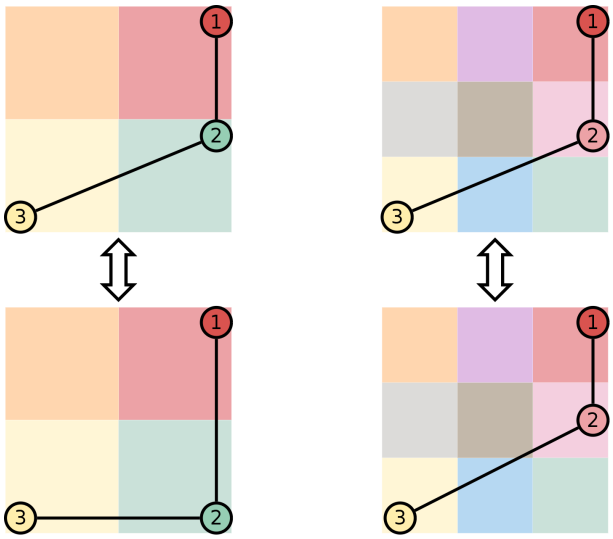
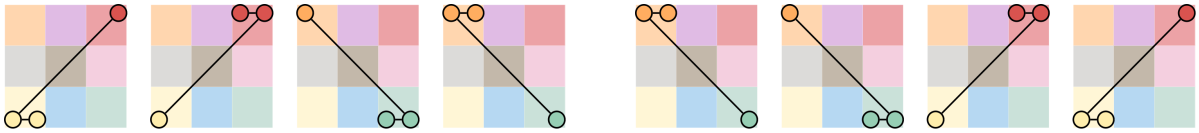


Figure 4.15: Class change upon usage of a 3×3 Bounding Box.

In the top images, we can see the actual placement of the nodes in their respective bounding box, and in the bottom images their class representation. As node 2 was very close to the *red-green* edge when compared to node 3 in the real node location inside the 2×2 box, when the number of divisions is increased, node 2 ends up in the pink partition instead of the green one, thus changing the subgraph to a class other than 1.

4.3.2 Results for "non grid-like" street layouts

Once again we resorted to the same principle we used in Section 4.2, now showing the four subgraphs with the most occurrences of the cities of Porto and Oxford.



(a) Porto (b) Oxford

Figure 4.16: Top-4 of subgraphs with most occurrences in the two non grid-like cities. Note that all the subgraphs are of class 2 (as defined in Figure 4.12).

In Figure 4.16, we can observe the same behaviour stated in Section 4.3.1. The top four subgraphs maintain the same equivalent class, but this time there was no shifting in the ranking of the subgraphs, they kept the same ordering they had in the 2×2 Bounding Box. The relative frequency of those subgraphs can be found in Tables 4.8 and 4.9.

Subgraph Rank	Subgraph Type	Relative Frequency
1	2	0.087947
2	2	0.082724
3	2	0.078755
4	2	0.069981
Top-4 total	—	0.319407

Table 4.8: Spatial subgraph frequencies in Porto (3×3 bounding box).

Subgraph Rank	Subgraph Type	Relative Frequency
1	2	0.096201
2	2	0.091555
3	2	0.081443
4	2	0.075704
Top-4 total	—	0.344903

Table 4.9: Spatial subgraph frequencies in Oxford (3×3 bounding box).

4.4 Additional Results

In this section, we will be showing more general results obtained by running our code (using both a 2×2 and a 3×3 box) with five more cities and special locations inside cities. Like before, each city is categorized as being "grid-like", "non grid-like". For the "grid-like" cities, we used Kyoto in Japan and Barcelona in Spain, and for the "non grid-like" we used Gent in Belgium, Brøndby Haveby in Denmark and Plaza del Ejecutivo, in Ciudad de México in Mexico. These last two places have a circular layout, which is very different from the other cities we analyzed, but we assessed that the results were very similar to those of "non grid-like" cities, and as they don't have a grid layout, we considered them to be of that type. Below, in Figure 4.17 we can see how Plaza del Ejecutivo looks like from above.

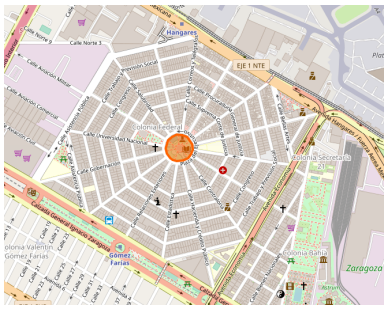


Figure 4.17: Plaza del Ejecutivo, taken from OSM and accessed in 5$^{\text{th}}$ September, 2022.

Using a 2×2 bounding box, the results were the ones we expected: Subgraphs of Class 1 dominate in Kyoto and Barcelona, while the other cities have Class 2 as the most frequent. The frequency of Classes 3 and 4 is relevant in all test cases, even though they appear to be more common in "non grid-like" cities, as they tend to have a more balanced subgraph distribution, that is, the difference in frequency between different classes is not as relevant as it is in the ones with a "grid-like" layout. In the case of Classes 5 and 6, the frequency is negligible in both cases. This information can be seen below, in Figure 4.18.

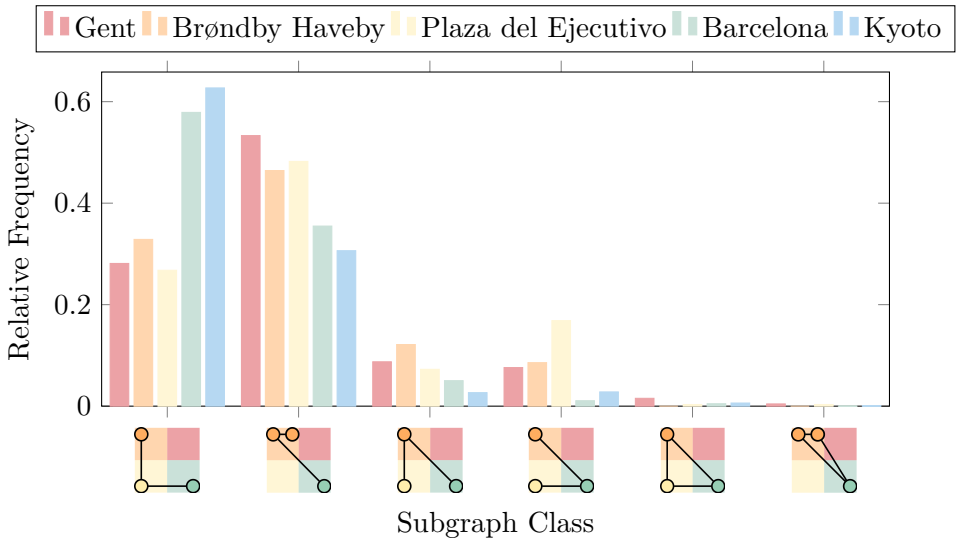


Figure 4.18: Spatial subgraph fingerprint, 2×2 bounding box.

In Table 4.10 we observe the number of nodes, edges and processing time of the five networks.

City	Nodes	Edges	Average Node Degree	Processing Time (s)
Gent	2405	3353	2.7884	0.713091
Brøndby Haveby	97	97	2.0000	0.0033111
Plaza del Ejecutivo	369	558	3.0244	0.0378826
Barcelona	2323	3512	3.0237	0.796847
Kyoto	4914	7303	2.9723	3.3063

Table 4.10: Network characteristics and run-times for Gent, Brøndby Haveby, Plaza del Ejecutivo, Barcelona and Kyoto, 2×2 box.

Using a 3×3 box the same behaviour occurred, which further supports the claim that our work is able to differentiate between these two types of cities, as is shown in Figure 4.19. Using this variation, the distribution of the subgraph classes is even more distinct: the relative frequency of the Class 1 subgraphs in Barcelona and Kyoto is overwhelmingly superior to that of Gent, Brøndby Haveby or Plaza del Ejecutivo.

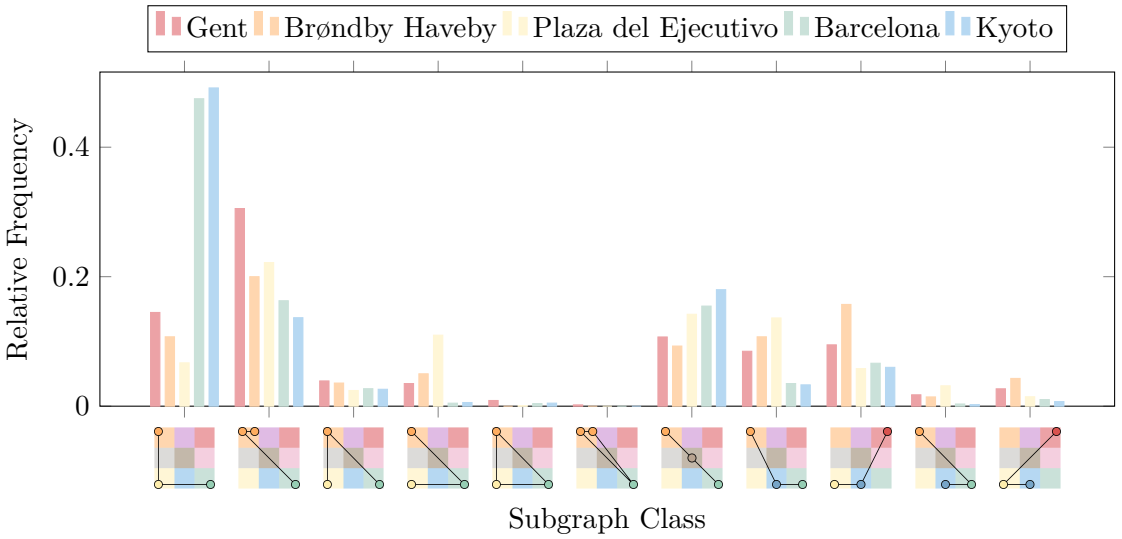


Figure 4.19: Spatial subgraph fingerprint, 3×3 bounding box.

Using this number of partitions allows us to draw further conclusions. In Figure 4.19 we can observe that Classes 7, 8 and 9 are also very relevant. When we divided the box into four quadrants, the most common Classes were 1 and 2. If we look at Class 2 in particular, when we add the extra granularity of using more partitions, one of the two nodes in the same quadrant can either stay in the same partition as the other one or "move" to one of its three adjacent partitions. In the latter case, we obtain either Class 7, 8 or 9. Classes 10 and 11 have a relatively big number of occurrences but they do not allow us to make any conclusion about the networks, although they appear to be more common in cities without a grid pattern.

Figure 4.20 represents the execution times of the program, comparing the usage of a 2×2 bounding box with a 3×3. By observing the figure, we can conclude that for small scale networks, the difference in execution time between the box sizes is not very big, but it appears that the disparity increases as networks get bigger and more dense. This is the case in Barcelona and Kyoto, where we can see a clear lower execution time with the 2×2 box, represented by the blue squares, but Figures 4.26 and 4.28 contradict this statement. We further explain why the first premise is incorrect in Subsection 4.7.

The locations are sorted by average node degree, as that was observed to be the main factor responsible for the increase in execution time. This is due to the fact that a higher average node degree directly translates to a denser network, and consequently to a bigger number of possible subgraphs, and while this is also true simply applied to the number of nodes, Enumerate SUbgraph (ESU) works by searching for possible edges starting from a node, thus making a more dense network have more impact in the performance. It is interesting to note that the average degree of all the cities tends to be of around 3, meaning that each intersection point is, in average, connected to 3 different roads.

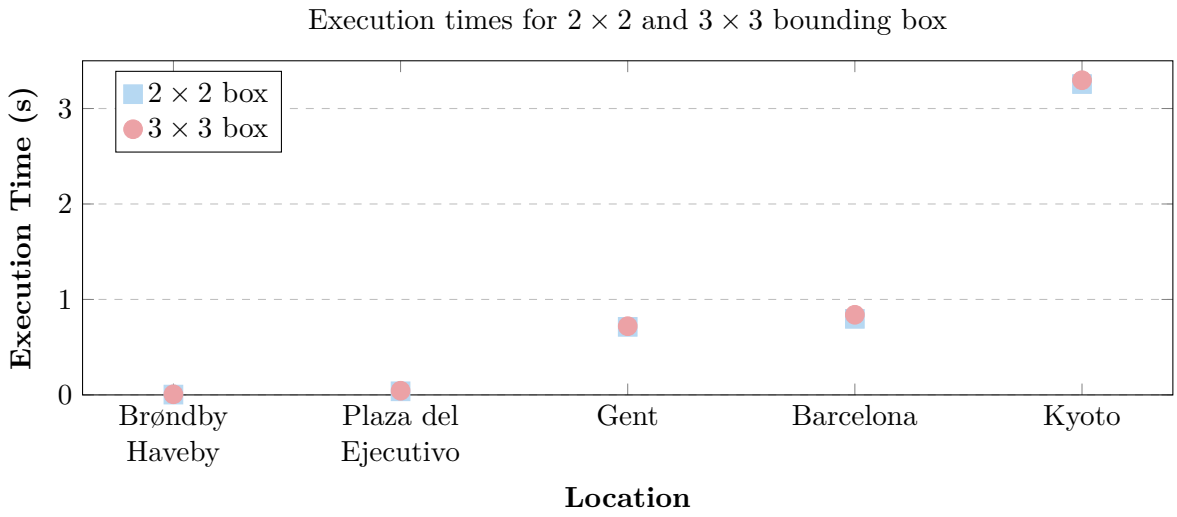


Figure 4.20: Execution times for 2×2 and 3×3 real networks.

4.4.1 A different dataset

We want to extend our work to other domains, but there is a substantial lack of networks with spatial information to be found online. We were able to retrieve the network of Central Chile's power grid system and applied our concept to that network. It contains a total of 347 nodes and 444 edges, with an average node degree of 2.55. The distribution of the classes of subgraphs found resembles that of "non grid-like" cities, as there is a very high concentration of Class 2 when compared to others. The results can be found in Figure 4.21.

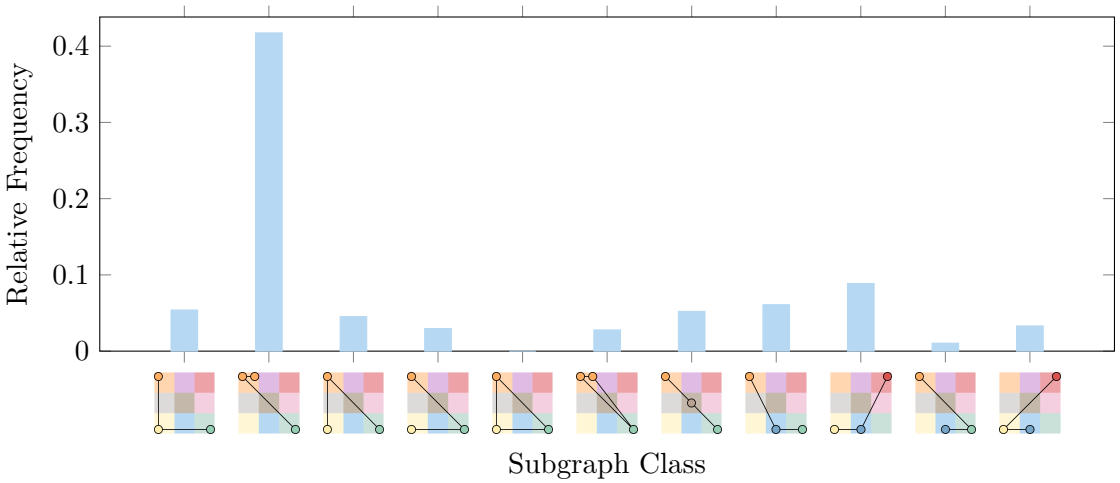


Figure 4.21: Subgraph class distribution in Central Chile's power grid system.

4.5 Real Network Stress testing

Upon analyzing the results shown in the previous sections, we decided to test the limists of our program. To do that, we used increasingly wider portions of the city of Tokyo as input. In Table 4.11 we can see the points used in each iteration (represented as a set of coordinates where each value corresponds to the latitude and longitude of the point respectively), as well as the number of nodes and edges that resulted in. All the tests performed in this section used the 3×3 bounding box.

Point 1	Point 2	Nodes	Edges	Total Subgraphs Found
(35.7005,139.6822)	(35.6816,139.7293)	8644	12577	26260
(35.7005,139.6822)	(35.6799,139.7451)	11131	16147	33704
(35.7016,139.6664)	(35.6799,139.7451)	14079	20443	42869
(35.7092,139.6548)	(35.6799,139.7451)	22671	33237	68620
(35.7136,139.6345)	(35.6799,139.7451)	31382	46075	95595

Table 4.11: Characteristics of each iteration (Real Network Stress Test).

Figure 4.22 represents the execution time in function of the number of edges of the network. As expected, the bigger the network, the longer the execution time. We can observe that this growth is exponential, but this is to be expected, since as stated before, the problem of finding a subgraph (to which our task is directly related) is a known NP-complete problem.

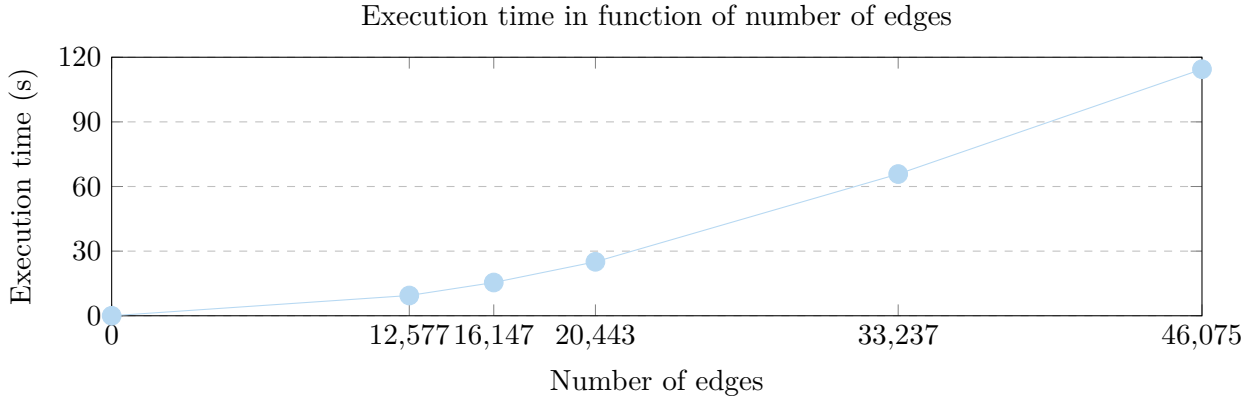


Figure 4.22: Execution times for real network stress testing.

In order to show that the complexity of the program is, in fact, exponential, Figure 4.23 presents the same plot as Figure 4.22 but on a logarithmic scale. As we can observe, the functions grows linearly, thus allowing us to conclude that there is an exponential factor in the complexity of the code.

We noticed from the data in Table 4.11 that the total number of subgraphs found is around the double of the number of edges of the network. This is due to the fact that Tokyo has a very "grid-like" layout, which means that there are a lot of cycles on the roads, thus generating more subgraphs as most of the times each road is part of at least two subgraphs, each one forming around each of the edges of that road. An example of why that happens can be seen

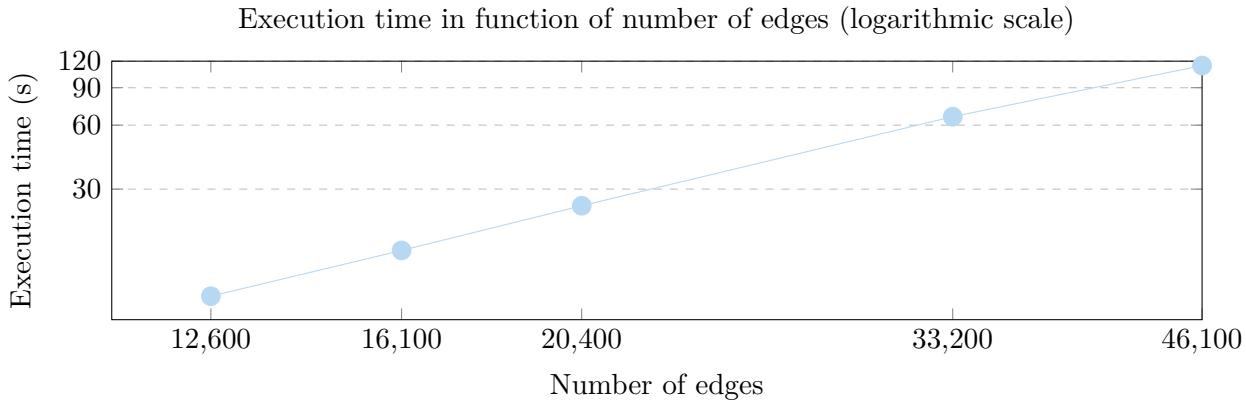


Figure 4.23: Execution times for real network stress testing (logarithmic scale).

in Figure 4.24. Using only a nine node cyclic graph with twelve edges, we ended up finding 22 subgraphs, and the difference increases if we try to increase the number of nodes of the network whilst keeping the same characteristics. This behaviour is shown in Table 4.11, as the difference between the total subgraphs found and the double of the number of edges increases as the number of edges grows.

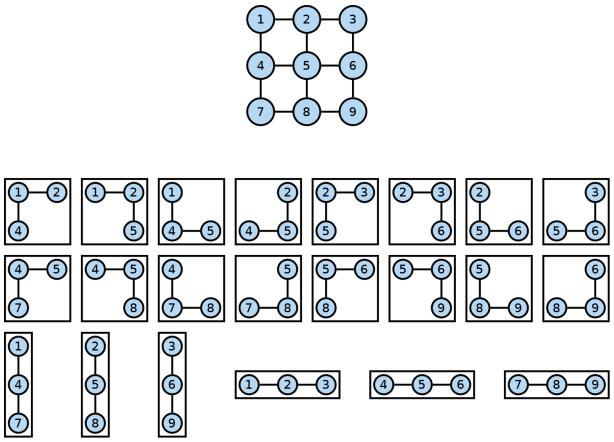


Figure 4.24: Cyclic "grid-like" network.

4.6 Overview

To summarize, Figure 4.25 shows a diagram of the workflow of our code.

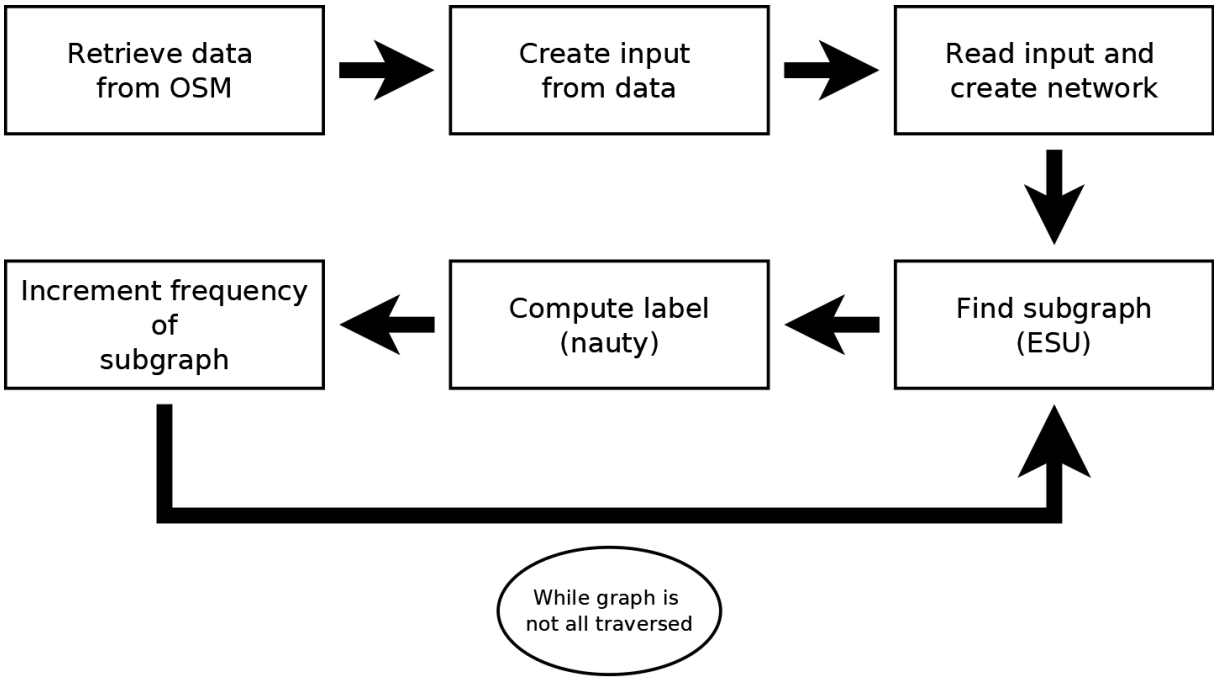


Figure 4.25: Program workflow.

Below can be found a description of the steps in more detail:

- In step 1, we retrieve the data from OSM using the *overpass* Application Programming Interface (API);
- In step 2, we transform that data into the input that our program requires;
- In step 3, we create a graph from the input;
- In step 4, we use ESU to traverse the graph and find subgraphs. When one is found, we calculate its corresponding bounding box, in order to assign colors to each node, and then proceed to step 5;
- In step 5, we parse the subgraph to one suitable for *nauty* and generate its canonical label;
- In step 6, we increment the frequency of subgraphs with the received label. We go back to step 4 when the graph is not fully traversed.

4.7 Experiences with synthetic data

When using real networks, there is always a lot of *noise* in the data. With the usage of synthetic data, we are able to show the true potential of our work. With that in mind, we implemented

methods capable of generating certain types of networks, with special characteristics useful to showcase insightful results.

4.7.1 Grid networks

We started by creating pure "grid-like" networks. For the sake of simplicity, we created a program that is only able to generate networks with the same number of nodes on the x and the y axis and where the space between two adjacent nodes is always the same. In Algorithm 5 we can see the pseudo-code for our function, where `s` represents the total number of edges in the network, that is, its size. Since we are creating networks with a square configuration, we take the square root of its size to assert the limits of each row and column, represented by the function *sqrt* in the code. The first procedure generates the set of coordinates of each node and the second procedure calculates which nodes are connected, first producing horizontal connections, followed by vertical ones. Essentially, we traverse the nodes from left to right, top to bottom and connect nodes in the same row and nodes in the same column. We do this efficiently by labeling the nodes in that same order, so that we can easily know what position a certain node belongs to.

Algorithm 5 Grid Network Generation Algorithm

```
1:  procedure GENERATE NODES(s)
2:      sq ←SQRT(s)
3:      for i ← 0 to sq do
4:          for j ← 0 to sq do
5:              PRINT(i,j)
6:          end for
7:      end for
8:  end procedure
9:  procedure GENERATE EDGES(s)
10:     sq ←SQRT(s)
11:     for k ← 0 to sq do
12:         for i ← k × sq + 1 to (k + 1) × sq do
13:             PRINT(i,i + 1)
14:         end for
15:     end for
16:     for k ← 1 to (sq − 1) ∗ (sq + 1) do
17:         for i ← k × sq + 1 to (k + 1) × sq by sq do
18:             PRINT(i,i + sq)
19:         end for
20:     end for
21: end procedure
```

We used in total 5 networks, of size 2500 to 62500 with steps, increasing the side of the square by 50 in each iteration. It should be noted that these networks are very dense, in the sense that each node is connected to potentially four other nodes, and to at least two. In Table 4.12 we can see that the number of edges (and of subgraphs found) is overwhelmingly superior to those of

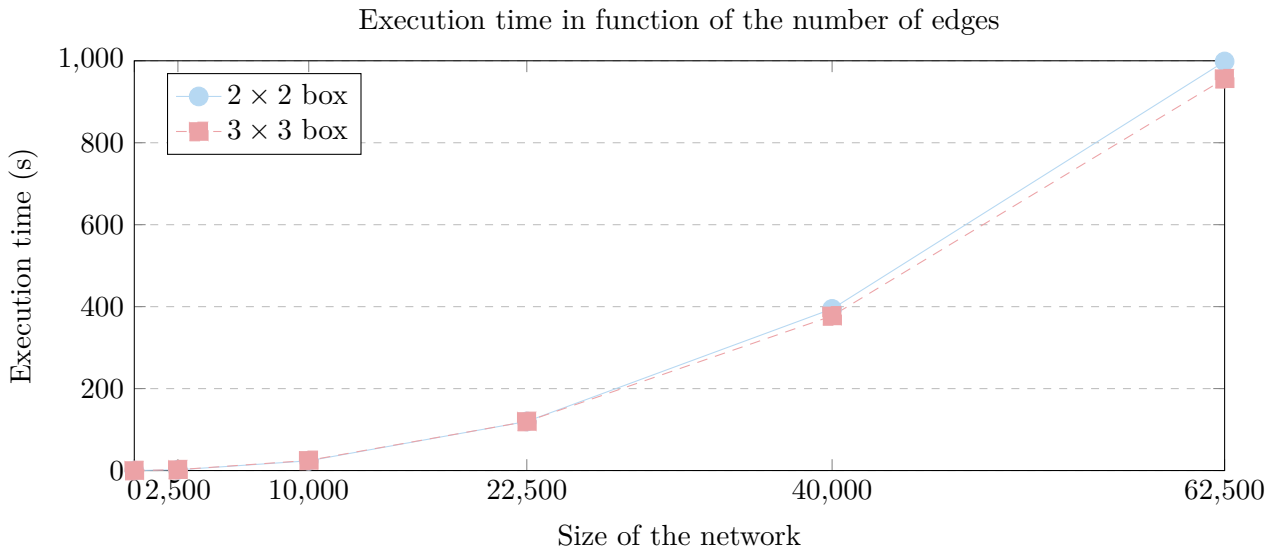


Figure 4.26: Execution times for synthetic grid networks.

real networks of similar size. For instance, if we compare the third network, with 22500 nodes to the fourth portion taken from Tokyo, with 22671 nodes, we can see that there is a big difference in the number of edges, with 44700 in the pure grid network compared to only 33237 in Tokyo, which is further justified if we consider the average node degree for each network: Tokyo's portion has an average degree of 2.93 whilst the random network has an average degree of 3.973. This difference increases as the number of edges gets higher.

Nodes	Edges	Total Subgraphs Found
2500	4900	14404
10000	19800	58804
22500	44700	133204
40000	79600	237604
62500	124500	372004

Table 4.12: Characteristics of each iteration (Synthetic Grid Network).

We can have a better perception of how the algorithm scales by using equivalent networks of varying sizes.

At first glance, one might think that all the subgraphs found are of class 1, as the network is purely "grid-like", but this is not the case, as a vertical and horizontal line that connects three nodes is also a subgraph, as we can see in Figure 4.27. For the sake of simplicity, we named each labeled each network with a number: The network with 2500 nodes with the number 1, the one with $10,000$ nodes with 2, the one with $22,500$ nodes with 3, the one with $40,000$ nodes with 4 and the one with $62,500$ with 5. The name of the networks will be $network_k$, where k corresponds to the label of the network.

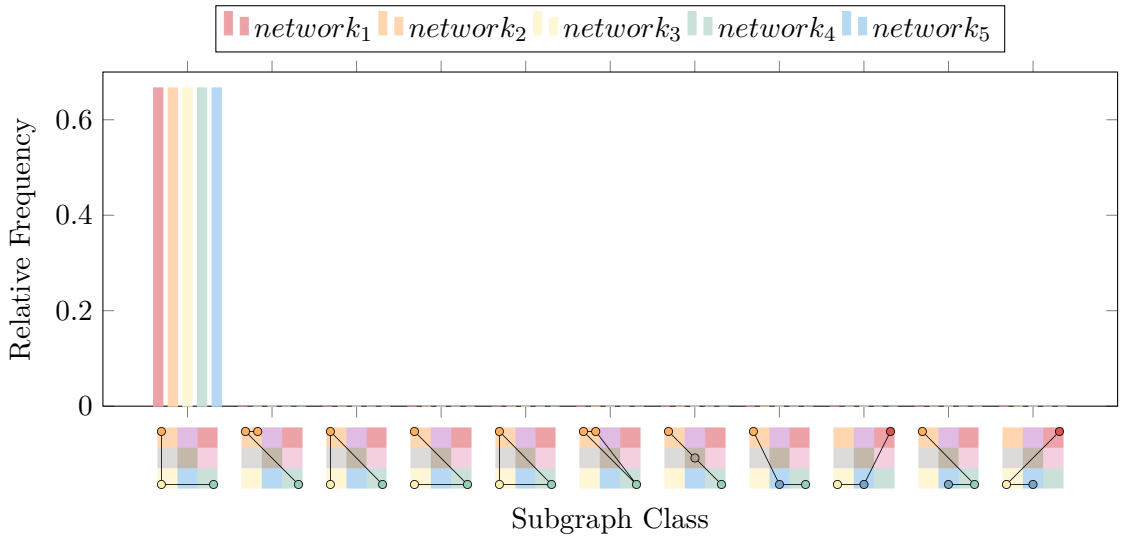


Figure 4.27: Spatial subgraph fingerprint, 3×3 bounding box on grid networks.

With this, two thirds of the subgraphs found are of class 1, and the others are of an undefined class, as they were not common in the real networks, but consist of straight lines (both vertical and horizontal). Figure 4.24 better illustrates this characteristic.

4.7.2 Random Networks

After the grid ones, we generated completely random networks. We pass as input the number of nodes we want the network to have, and from there the program connects each of them to an arbitrary number of edges that are at least the size of the network and at most double its size. Each node has a set of coordinates associated, with each one corresponding to an integer value between $-100,000$ and $100,000$, which allows for a total of $200,000 \times 200,000 = 40,000,000,000$ possible sets of coordinates. There are no loops, and the graph is, again, undirected. The pseudo-code for the function can be seen in Algorithm 6. As was the case with the grid network generator in Algorithm 5, s represents the network's size, the first procedure generates the nodes and the second one the edges.

Algorithm 6 Random Network Generation Algorithm

```
1:  procedure GENERATE NODES(s)
2:      for i ← 0 to s do
3:          x ←RAND(-100000,100000)
4:          y ←RAND(-100000,100000)
5:          while IS_USED((x, y)) do
6:              x ←RAND(-100000,100000)
7:              y ←RAND(-100000,100000)
8:          end while
9:          PRINT(x,y)
10:         MARK_USED((x, y))
11:     end for
12: end procedure
13: procedure GENERATE EDGES(s)
14:     e ←RAND(s,s × 2)
15:     for i ← 0 to e do
16:         a ←RAND(1,s)
17:         b ←RAND(1,s)
18:         while IS_USED((a, b)) or a = b do
19:             a ←RAND(1,s)
20:             b ←RAND(1,s)
21:         end while
22:         PRINT(a,b)
23:         MARK_USED((a, b))
24:     end for
25: end procedure
```

In Algorithm 6, lines 5 and 10 make sure that no two nodes have the same set of coordinates, whilst lines 18 and 23 do not allow for edges to repeat or for self loops to happen. With this, we have a set of random networks to serve as a base case, in order to allow for comparisons with other types of networks. For the sake of consistency, we created random networks with the same amount of nodes as the ones represented in Table 4.12. The details for these random networks can be found in Table 4.13. The same labeling used in Figure 4.27 for the networks will also be used in further figures.

Nodes	Edges	Average Node Degree	Total Subgraphs Found
2500	2604	2.0832	5357
10000	11217	2.2434	25012
22500	42226	3.7534	158136
40000	69480	3.4740	240628
62500	114018	3.6485	416690

Table 4.13: Characteristics of each iteration (Synthetic Random Network).

It is clear from Table 4.13 that the first two random networks have a relatively low number of edges compared to the number of nodes, whilst the last three are very dense, as shown by the average node degree suggests.

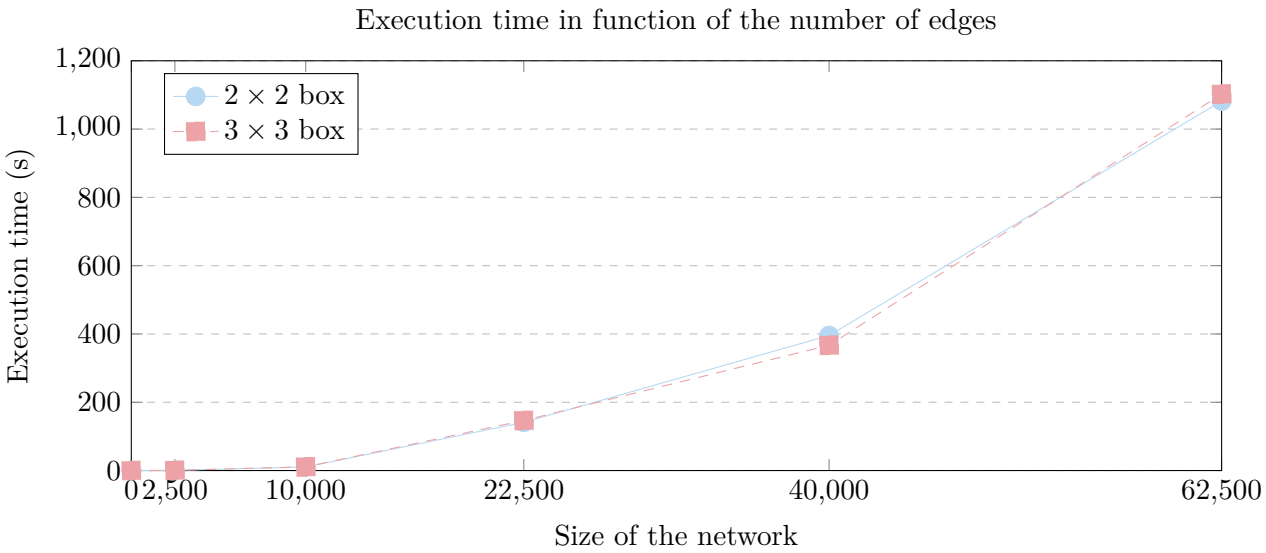


Figure 4.28: Execution times for synthetic random networks.

Figure 4.28 shows the execution times for these networks in function of the number of edges.

As stated in Subsection 4.5, Figures 4.26 and 4.28 argue against the idea that increasing the size or density of the graph also increases the difference in execution times when comparing the usage of a 2×2 and a 3×3 bounding box. In fact, there are some cases where the 2×2 box was even slower. It is expected that the bounding box will not affect the execution times since the complexity of the program is dominated by the ESU algorithm.

Since we are now working with random networks, the classes of subgraphs found have very different distributions when compared to previous scenarios. Now, there is almost an even distribution of the frequencies of the subgraphs amongst the first four classes, and they keep being the most relevant in all networks. Figure 4.29 allows for a better visualization of these results.

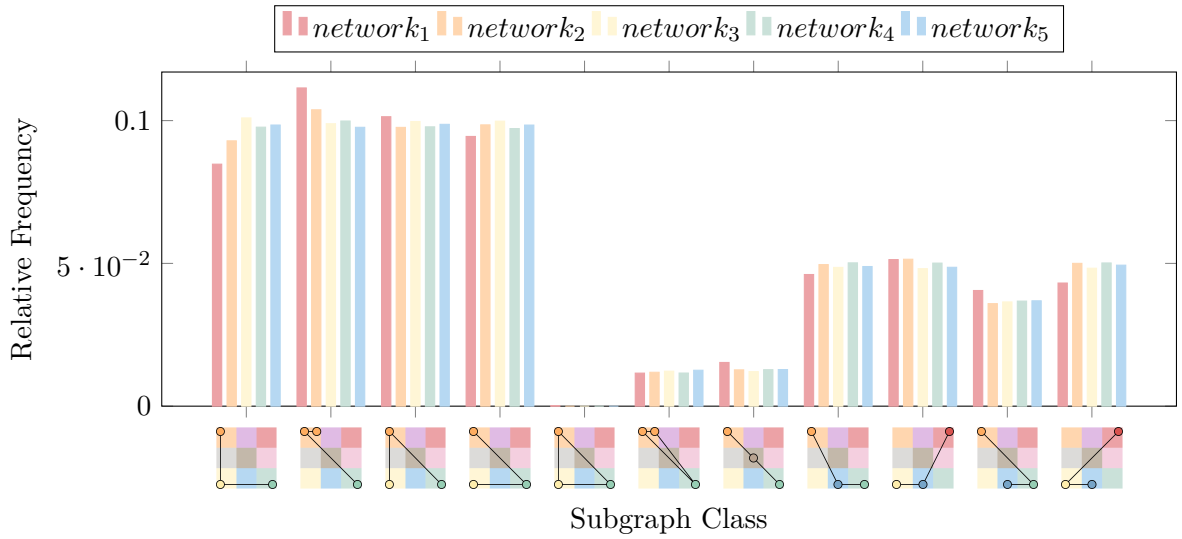


Figure 4.29: Spatial subgraph fingerprint, 3×3 bounding box on random networks.

It is interesting to notice that even though the networks were generated completely at random, they have similar distribution of subgraph classes among themselves. One could use that distribution as a null model, that is, as the basis on which comparisons are made to distinguish types of networks. Because of the more balanced distribution, Classes 10 and 11 end up being more relevant in the overall analysis of subgraphs found compared to the cities, where they were dominated by the first two classes.

4.7.3 Fingerprint analysis

As previously stated, network fingerprints allow us to have a representation of the graph, that can be used in graph mining tasks, like classification. We will be deriving a subgraph fingerprint for two graphs, to show that a fingerprint with a spatial component is able to differentiate two otherwise similar subgraphs. In Figure 4.30 we can see two subgraphs, R and P, that will be used to test our previous assumption.

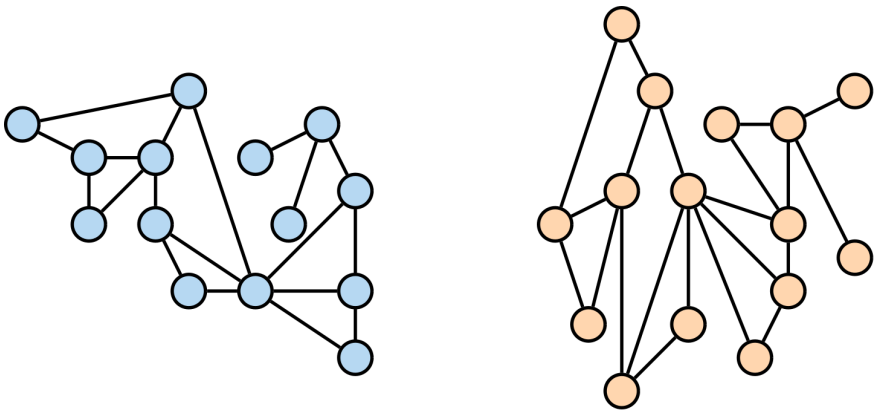


Figure 4.30: Graphs R and P.

As seen before, if we don't consider any spatial component or coloring of the nodes and edges in undirected networks, there are only two possible subgraph types, triangles and chains. Figure 4.31 shows the fingerprints of R and P considering only if a subgraph is either a triangle or a chain.

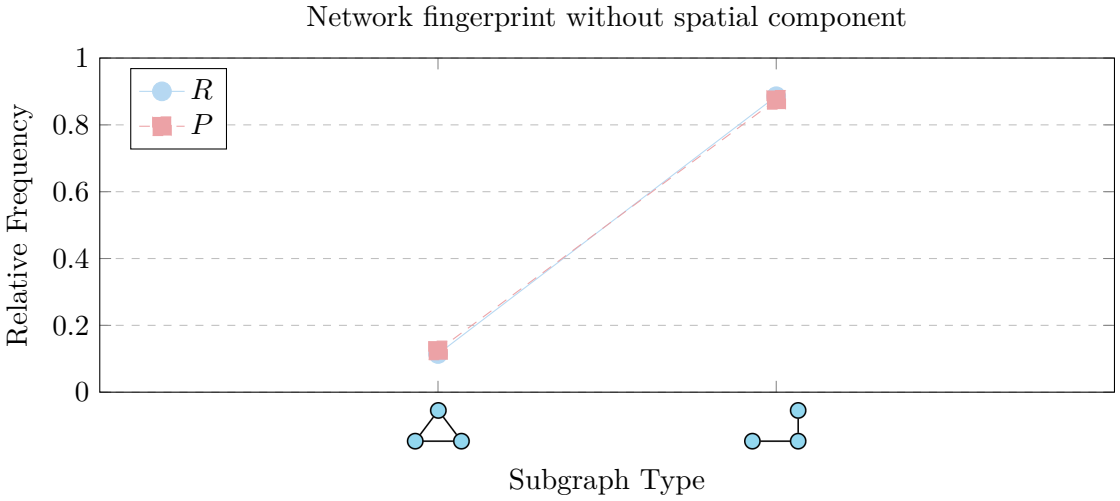


Figure 4.31: Network fingerprint of R and P without spatial component.

As can be seen in Figure 4.31, both networks have a very similar fingerprint, and would then be considered similar. When we take into account the spatial component, that is no longer the case. Figure 4.32 shows the fingerprint using the subgraph classes defined in Figures 4.12 and 4.13, extracted using the 3×3 box.

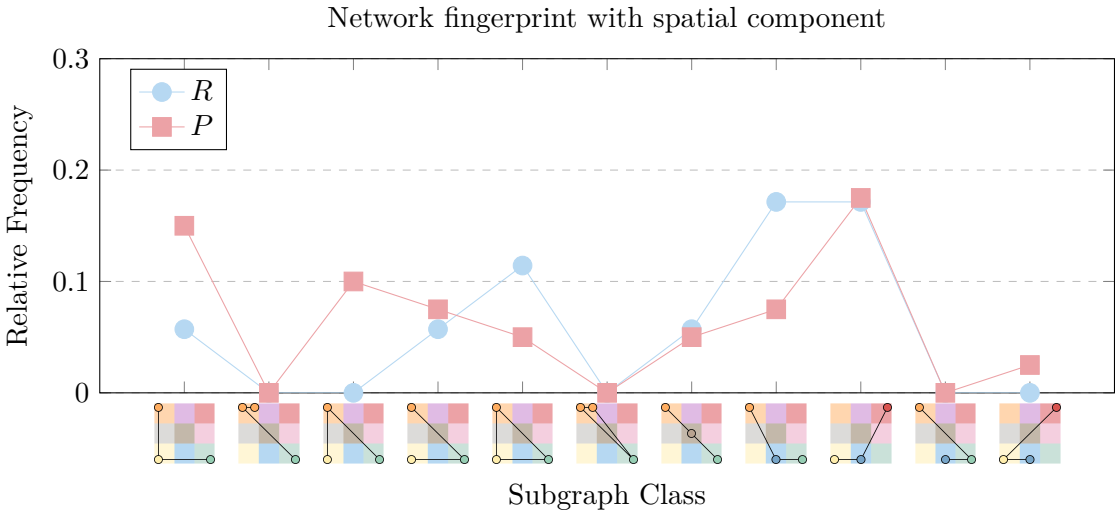


Figure 4.32: Network fingerprint of R and P with spatial component, 3×3 box.

Even though there are similarities, the networks no longer have the same fingerprint, as there is a very noticeable difference in subgraphs of Classes 1, 3, 5 and 8.

Even if the simpler bounding box is used, the same conclusion can be extracted, as can be seen in Figure 4.33. The results are not as evident, but there is a clear difference between the frequency of Classes 2, 3 and 4 in networks R and P, further proving our point.

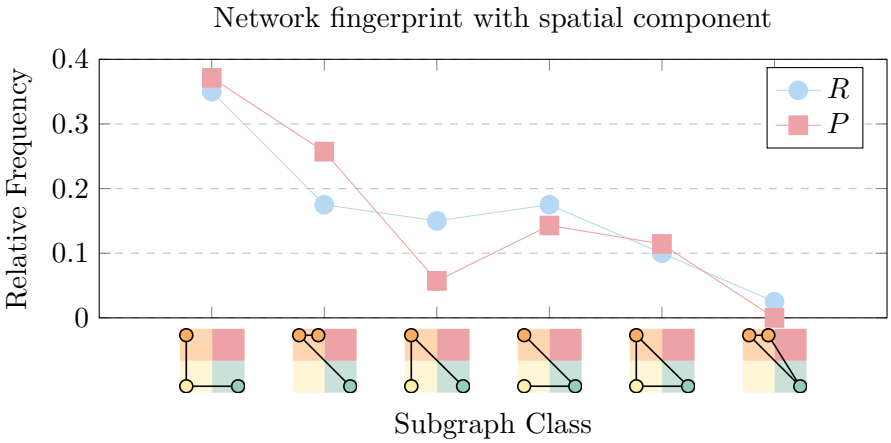


Figure 4.33: Network fingerprint of R and P with spatial component 2×2 box.

Chapter 5

Conclusions and Future Work

With this, we reach the end of this thesis. In our work, we present a set of contributions aiming to incorporate spatial properties into subgraph analysis. We first offer a novel abstraction that relies on a bounding box and regular spatial partitions to attribute node colors that describe the relative position of nodes within the subgraph. We defined a set of classes to be used to characterize the subgraphs found, thus allowing for a higher level of grouping when compared to individually classify a subgraph occurrence. We then describe an implementation of a framework capable of discovering and counting these spatial subgraphs, based on enumerating occurrences and then discovering their type using a specialized canonical labeling mechanism that assigns nodes to their respective partition of a bounding box, coloring them appropriately and allowing for their visualization, with the usage of Enumerate SUbgraph (ESU) and *nauty*. We also provide a proof of concept experiment using real-life data in which we show that our approach is able to go beyond classical topological motifs, capturing enough information to distinguish between different road network layouts. Finally, we performed some extra tests with synthetic networks that allow for the extraction of even more conclusions about our concept, such as the computational limits and patterns of networks that are harder to grasp when using real data.

We believe these are promising results that could lead to new insights on the characterization and comparison of networks with spatial information. Our end goal is to be able to provide a universal spatial concept of network motifs that can be generally applicable to networks of any domain and a definition as general as the one proposed in this work is a big step towards that goal.

5.1 Future Work

In order to further extend our work, we intend to study the incorporation of higher dimensional data, such as 3D brain networks (as depicted in Figure 5.1), and we want to make an extensive evaluation of the role of the granularity in the information gained, by carefully analysing what happens when we use more or less and bigger or smaller spatial partitions. Furthermore, we want

to study how changing the point of reference would impact the results (e.g. what happens to the patterns when we make arbitrary rotations?) and we intend to explore different symmetries and subgraph families that could provide classes that are invariant to spatial transformations (e.g. mirror symmetry).

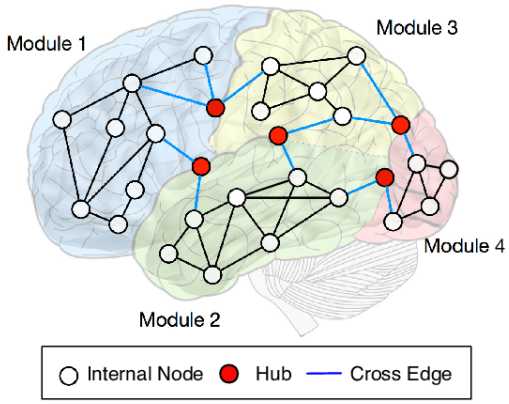


Figure 5.1: Brain network, taken from [73].

The idea is also general enough that it can be applied to a variety bounding box geometries and with different sized partitions. For instance, one could use an octagonal format to define the box, assigning three different partitions that correspond to the top, middle and bottom of the octagon, as is shown in Figure 5.2.

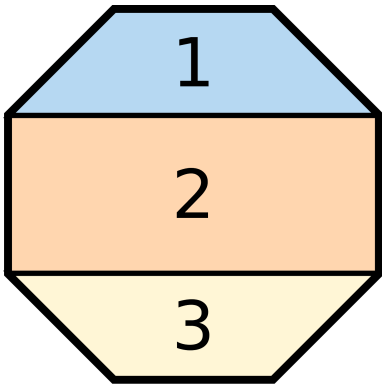


Figure 5.2: Octagon box with different sized partitions.

It should be noted that partitions can represent many different things. If we think of a Football field, partitions could be the attack, middle and defense sections of the field, in the case of a brain network, partitions could represent the brain's different parts, and if we have a flight network, the partitions could be the countries an airport belongs to. With that said, one should not think of them as being simple divisions of the space, but as divisions that do in fact aggregate nodes (or even edges) into groups.

Besides changing the box and the partitions, we also intend to use different sources of data. In the case of the cities, we were able to assert the layout its streets have, and so some

interesting conclusions might be derived from applying that same methodology to brain networks, molecular structures or even transportation networks. The amount of real-life systems that can be represented as networks is vast and we believe our concept might be useful to provide new knowledge in some of those systems, as long as there is a spatial component attached to the network.

We also want to understand how we can assess statistical significance of the subgraph frequencies by studying what could be appropriate spatial null models. Having a defined null model, one might be able to characterize and assign a certain network to a group without needing to compare it with other examples, unlike what was necessary in the case of the cities.

Finally, we want to improve the efficiency, not only by improving the exact counting computation, but also by trading accuracy for speed (e.g. using sampling) or using parallelism (e.g. using several threads in multicore machines).

Bibliography

[1] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii Academiae Scientiarum Imperialis Petropolitanae*, 8:128–140, 1736.

[2] Pawel Boguslawski. *Modelling and analysing 3D building interiors with the dual half-edge data structure*. PhD thesis, University of South Wales (United Kingdom), January 2011.

[3] Ré ka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, January 2002.

[4] Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2): 167–256, 2003.

[5] Marc Barthélemy. Spatial networks. *Physics Reports*, 285:1–101, February 2011.

[6] Marc Barthélemy. *Morphogenesis of spatial networks*. Springer, 2018.

[7] Marián Boguñá, Dmitri Krioukov, Pedro Almagro, and M. Ángeles Serrano. Small worlds and clustering in spatial networks. *Physical Review Research*, 2(2):023040, 2020.

[8] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298:824–827, October 2002.

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

[10] Sarvenaz Choobdar, Pedro Ribeiro, and Fernando Silva. Motif mining in weighted networks. *Data Mining Workshops (ICDMW), 2012 IEEE 12th International Conference on.IEEE*, pages 210–217, December 2012.

[11] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. Motifs in temporal networks. In *Proceedings of the tenth ACM international conference on web search and data mining*, pages 601–610. Association for Computing Machinery, 2017.

[12] Pedro Ribeiro and Fernando Silva. Discovering colored network motifs. In Pierluigi Contucci, Ronaldo Menezes, Andrea Omicini, and Julia Poncela-Casasnovas, editors, *Complex Networks V. Studies in Computational Intelligence*, volume 549, pages 107–118. Springer, March 2014.

[13] Sallamari Sallmen, Tarmo Nurmi, and Mikko Kivelä. Graphlets in multilayer networks. *Journal of Complex Networks*, 10(2):cnac005, 2022.

[14] Takuma Narizuka, Ken Yamamoto, and Yoshihiro Yamazaki. Statistical properties of position-dependent ball-passing networks in football games. *Physica A: Statistical Mechanics and its Applications*, 412:157–168, 2014.

[15] José Ferreira, Pedro Ribeiro, and Alberto Barbosa. Towards the concept of spatial network motifs. *Complex Networks and their Applications*, November 2022.

[16] Joseph Gallian. A dynamic survey of graph labeling. *The Electronic Journal of Combinatorics*, 19, 2000.

[17] Alexander Rosa. On certain valuations of the vertices of a graph. In *Theory of Graphs (International Symposium, Rome)*, pages 349–355. Dunod Gordon & Breach Science Publishers, 1966.

[18] R. L. Grahams and N. Sloane. On additive bases and harmonious graphs. *Siam Journal on Algebraic and Discrete Methods*, 1, 1980.

[19] Adolfo Piperno. Search space contraction in canonical labeling of graphs. *CoRR*, abs/0804.4881, 2008.

[20] Xiuliang Cui, Haochen He, Fuchu He, Shengqi Wang, Fei Li, and Xiaochen Bo. Network fingerprint: a knowledge-based characterization of biomedical networks. *Scientific Reports*, 5:13286, 2015.

[21] Aaron M. Corpstein. Browser fingerprinting and the importance of digital privacy. *Scholarly Horizons: University of Minnesota, Morris Undergraduate Journal*, 8:1, 2021.

[22] Pedro Ribeiro, Pedro Paredes, Miguel E. P. Silva, David Aparicio, and Fernando Silva. A survey on subgraph counting. *ACM Computing Surveys*, 54:1–36, 2022.

[23] Pedro Ribeiro and Pedro Paredes. Large scale graph representations for subgraph census. In *Advances in Network Science*, pages 186–194. Springer International Publishing, 2016.

[24] Pedro Paredes and Pedro Ribeiro. Rand-fase: fast approximate subgraph census. *Social Network Analysis and Mining*, 5(1):1–18, 2015.

[25] David Aparício, Pedro Paredes, and Pedro Ribeiro. A scalable parallel approach for subgraph census computation. In *European Conference on Parallel Processing*, pages 194–205, 2014.

[26] Stephen A. Cook. The complexity of theorem-proving procedures. In *IN STOC*, pages 151–158. ACM, 1971.

[27] Stéphane Zampelli, Yves Deville, and Christine Solnon. Solving subgraph isomorphism problems with constraint programming. *Constraints*, 15:327–353, July 2010.

[28] David Eppstein. Subgraph isomorphism in planar graphs and related problems. *Graph Algorithms and Applications*, 1:283–309, 2002.

[29] Bill Hillier and Julienne Hanson. *The Social Logic of Space*. Cambridge University Press, 1984.

[30] Paolo Crucitti, Vito Latora, and Sergio Porta. Centrality measures in spatial networks of urban streets. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 73:036125, March 2006.

[31] Richard Karp. Reducibility among combinatorial problems. In *The IBM Research Symposia Series*, volume 40, pages 85–103. Dunod Gordon & Breach Science Publishers, 1972.

[32] Walter Klotz. Graph coloring algorithms. *Mathematik-Bericht*, 5:1–9, January 2002.

[33] Leonid Barenboim and Michael Elkin. Distributed graph coloring: Fundamentals and recent developments. *Synthesis Lectures on Distributed Computing Theory*, 4(1):1–171, 2013.

[34] M. R. Garey and S. D. Johnson. The complexity of near-optimal graph coloring. *Journal of the ACM*, 23(1):43–49, January 1976.

[35] Hana Rizvić, Sanda Martincic-Ipsic, and Ana Meštrović. Network motifs analysis of croatian literature. *arXiv*, pages 1–36, 2014.

[36] Patrik D'haeseleer. What are dna sequence motifs? *Nature biotechnology*, 4:423–425, May 2006.

[37] Albert-Laszlo Barabasi and Zoltan Oltvai. Network biology: Understanding the cell's functional organization. *Nature reviews. Genetics*, 5:101–113, March 2004.

[38] S. Itzkovitz, R. Levitt, N. Kashtan, R. Milo, M. Itzkovitz, and U. Alon. Coarse-graining and self-dissimilarity of complex networks. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 71:016127, February 2005.

[39] Lei Wang, Jing Ren, Bo Xu, Jianxin Li, Wei Luo, and Feng Xia. Model: Motif-based deep feature learning for link prediction. *IEEE Transactions on Computational Social Systems*, 7: 1–14, February 2020.

[40] Guangyu Wu, Martin Harrigan, and Padraig Cunningham. Characterizing wikipedia pages using edit network motif profiles. *International Conference on Information and Knowledge Management, Proceedings*, 3:45–52, October 2011.

[41] Olaf Sporns and Rolf Kötter. Motifs in brain networks. *PLoS biology*, 2:e369, December 2004.

[42] Jukka-Pekka Onnela, Jari Saramäki, János Kertész, and Kimmo Kaski. Intensity and coherence of motifs in weighted complex networks. *Physical Review E*, 71(6):065103, July 2005.

[43] Lisa Röttjers, Doris Vandeputte, Karoline Jeroen, and Faust. Null-model-based network comparison reveals core associations. *ISME Communications*, 1:36, 2021.

[44] Ron Milo, Nadav Kashtan, Shalev Itzkovitz, M. Newman, and Uri Alon. On the uniform generation of random graphs with prescribed degree sequences. *Tech rep*, 21, January 2004.

[45] Ron Milo, Shalev Itzkovitz, Nadav Kashtan, Reuven Levitt, Shai Shen-Orr, Inbal Ayzenshtat, Michal Sheffer, and Uri Alon. Superfamilies of evolved and designed networks. *Science*, 303: 1538–1542, March 2004.

[46] Miguel Silva, Pedro Paredes, and Pedro Ribeiro. Network motifs detection using random networks with prescribed subgraph frequencies. In Bruno Gonçalves, Ronaldo Menezes, Roberta Sinatra, and Vinko" Zlatic, editors, *Complex Networks VIII*, pages 17–29. Springer International Publishing, February 2017.

[47] Pedro Ribeiro, Pedro Paredes, Miguel EP Silva, David Aparicio, and Fernando Silva. A survey on subgraph counting: concepts, algorithms, and applications to network motifs and graphlets. *ACM Computing Surveys (CSUR)*, 54(1):1–36, 2021.

[48] Shuo Yu, Yufan Feng, Da Zhang, Hayat Bedru, Bo Xu, and Feng Xia. Motif discovery in networks: A survey. *Computer Science Review*, 37:100267, August 2020.

[49] Joshua A. Grochow and Manolis Kellis. Network motif discovery using subgraph enumeration and symmetry-breaking. In *Research in Computational Molecular Biology*, pages 92–106. Springer Berlin Heidelberg, 2007.

[50] Nadav Kashtan, Shalev Itzkovitz, Ron Milo, and Uri Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics (Oxford, England)*, 20:1746–58, 2004.

[51] Saeed Omidi and Falk Schreiber. Moda: An efficient algorithm for network motif discovery in biological networks. *Genes & genetic systems*, 84:385–95, October 2009.

[52] Sahand Khakabimamaghani, Iman Sharafuddin, Norbert Dichter, and Ina Koch. Quatexelero: An accelerated exact network motif detection algorithm. *PloS one*, 8:e68073, July 2013.

[53] Zahra Kashani, Hayedeh Ahrabian, Elahe Elahi, Abbas Nowzari, Elnaz Saberi Ansari, Sahar Asadi, Shahin Mohammadi, and Falk Schreiber. Kavosh: A new algorithm for finding network motifs. *BMC bioinformatics*, 10:318, October 2009.

[54] Xutong Liu, Yu-Zhen Chen, John C. S. Lui, and K. Avrachenkov. Learning to count: A deep learning framework for graphlet count estimation. *Network Science*, 9, September 2020.

[55] Francesco Bruno, Luigi Palopoli, and Simona Rombo. New trends in graph mining: Structural and node-colored network motifs. *International Journal of Knowledge Discovery in Bioinformatics*, 1:81–99, July 2012.

[56] Christoph Adami, Jifeng Qian, Matthew Rupp, and Arend Hintze. Information content of colored motifs in complex networks. *Artificial Life*, 17(4):375–390, October 2011.

[57] Jifeng Qian, Arend Hintze, and Christoph Adami. Colored motifs reveal computational building blocks in the c. elegans brain. *PLoS ONE*, 6:e17013, March 2011.

[58] Esti Yeger-Lotem, Shmuel Sattath, Nadav Kashtan, Shalev Itzkovitz, Ron Milo, Ron Pinter, Uri Alon, and Hanah Margalit. Network motifs in integrated cellular networks of transcription-regulation and protein-protein interaction. *Proceedings of the National Academy of Sciences of the United States of America*, 101:5934–9, May 2004.

[59] Gerard J. Kleywegt. Recognition of spatial motifs in protein structures. *Journal of Molecular Biology*, 285:1887–1897, January 1999.

[60] Pedro Ribeiro and Fernando Silva. G-tries: An efficient data structure for discovering network motifs. *Proceedings of the ACM Symposium on Applied Computing*, pages 1559–1566, January 2010.

[61] Pedro Ribeiro and Fernando Silva. G-tries: a data structure for storing and finding subgraphs. *Data Mining and Knowledge Discovery*, 28:337–377, March 2014.

[62] Edward Fredkin. Trie memory. *Communications of The ACM*, 3:490–499, September 1960.

[63] Pedro Ribeiro. *Efficient and Scalable Algorithms for Network Motifs Discovery*. PhD thesis, University of Porto, June 2011.

[64] Sebastian Wernicke and Florian Rasche. FANMOD: a tool for fast network motif detection. *Bioinformatics*, 22(9):1152–3, June 2006.

[65] Pedro Paredes and Pedro Ribeiro. Towards a faster network-centric subgraph census. In *2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, page 264–271. Association for Computing Machinery, 2013.

[66] Joshua A. Grochow and Manolis Kellis. Network motif discovery using subgraph enumeration and symmetry-breaking. In *Proceedings of the 11th Annual International Conference on Research in Computational Molecular Biology*, RECOMB'07, page 92–106. Springer-Verlag, 2007.

[67] Samiul Hasan, Christian M Schneider, Satish V Ukkusuri, and Marta C González. Spatiotemporal patterns of urban human mobility. *Journal Statistical Physics*, 151(1): 304–318, 2013.

[68] Sebastian Wernicke. Efficient detection of network motifs. *IEEE/ACM transactions on computational biology and bioinformatics*, 4(4):347–359, 2006.

[69] Martin Grohe and Pascal Schweitzer. The graph isomorphism problem. *Communications of the ACM*, 11(11):128–134, 2020.

[70] Brendan D McKay and Adolfo Piperno. Practical graph isomorphism, ii. *Journal of symbolic computation*, 60:94–112, 2014.

[71] Frank Harary and Edgar M. Palmer. *Graphical enumeration*. Academic Press, 1973.

[72] Mordechai Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive computing*, 7:12–18, 2008.

[73] Guixiang Ma, Chun-Ta Lu, Lifang He, Philip Yu, and Ann Ragin. Multi-view graph embedding with hub detection for brain network analysis. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 967–972, September 2017.