UNIVERSITÀ DEGLI STUDI DI CATANIA
FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
CORSO DI LAUREA SPECIALISTICA IN FISICA

Alessio Vincenzo Cardillo

Structural Properties of Planar Graphs of Urban Street Patterns

---

TESI DI LAUREA SPECIALISTICA

Relatori:
Chiar.mo Prof. Vito Latora

ANNO ACCADEMICO 2009/2010

A Nino ed al Prof. Raciti

. . . un pezzetto di voi viaggia sempre con me.

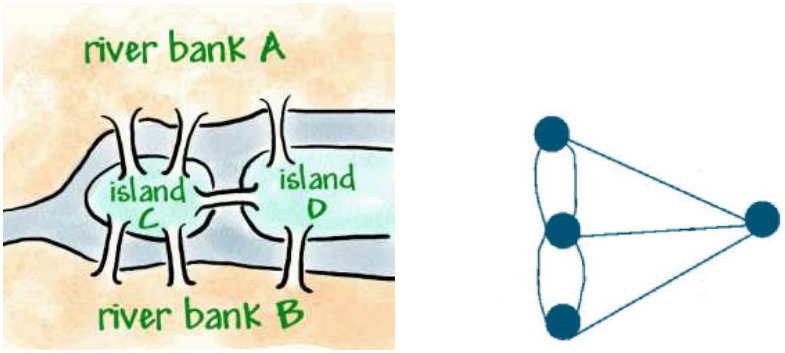
CONTENTS

INTRODUCTION

Networks are all around us, and we are ourselves, as individuals, the units of a network of social relationships of different kinds and, as biological systems, the delicate result of a network of biochemical reactions.

Historically, the study of networks started as the domain of a branch of discrete mathematics known as **graph theory**, initiated in 1736, when Leonard Euler published the solution to the Könisberg bridges problem (it consisted in finding a round trip that traversed each of the Könisberg's seven bridges exactly once; cf. Fig 1(a)). Since then, graph theory has developed and has provided answers to a series of practical questions such as: what is the maximum flow per unit time from source to sink in a network of pipes? How to assing n jobs to n people with maximum total utility? In addition, the study of networks over the years has seen important achievements in some specialized contexts such as social sciences, immunology and computer science to cite a few.

The last decade has known the birth of a new movement of interest and research in the study of **complex networks**, i.e. networks whose structure is irregular, complex and dynamically evolving in time, with a renewed attention to the properties of networks of dynamical units. This flurry of activity has been triggered by two seminar papers: one



(a) (b)

Figure 1: A schematic view of the Könisberg bridges: (a) map view and (b) corrisponding graph.

by Watts and Strogatz on *small-world* networks appeared in 1998 in "Nature" [1], and one by Barabàsi and Albert on *scale-free* networks appeared in 1999 in "Science" [2]. The main character of this activity was the physicists' community. They were induced by the possibility to study the properties of a plenty of large databases of real networks. These include transportation networks, phone call networks, the Internet and the World Wide Web, the actors' collaboration network in movie databases, neural or genetic networks, metabolic and protein networks, scientific coauthorship and citation networks from the Science Citation Index and so on. In fact, the growing availability of large databases togheter with the increasing computing powers, as well as the development of reliable data analysis tools, have constituted a better machinery to explore the topological properties of several complex networks from the real world [3–7].

The main outcome of this activity has been to reveal that, despite the inherent differences, most of the real networks are characterized by the same topological properties, as for instance relatively small **characteristic path lengths** and high **clustering coefficients** (the so called small-world property) [1], scale-free **degree distributions** [2], **degree**

correlations [8], and the presence of **motifs** [9] and **community structures** [10].

All such features make real networks radically different from regular lattices or random graphs, the standard topologies usually used in modeling and computer simulations. This has led to a noticeable attention towards the comprehension of the evolution mechanisms that have shaped the topology of a real network, and to the design of new models retaining the most significant properties observed empirically.

Spatial networks are a special class of complex networks whose nodes are embedded in a two or three-dimensional Euclidean space and whose edges do not define relations in an abstract space (such as in networks of acquaintances or collaborations between individuals [11–14]), but are real physical connections [6, 15]. Typical examples include neural networks [16], information/communication networks [17, 18], electric power grids [19] and transportation systems ranging from river [20], to airport [21, 22] and street [23] networks. Initially most of the works in the literature, with a few relevant exceptions [17, 24, 25], have focused on the characterization of the topological properties of spatial networks, while the spatial aspect has received less attention, when not neglected at all. However, it is not surprising that the topology of such systems is strongly constrained by their spatial embedding and influenced by the properties of the surrounding space. For instance, there is a cost to pay for the existence of long-range connections in a spatial network, this having important consequences on the possibility to observe a small-world behavior. Moreover, the number of edges that can be connected to a single node is often limited by the scarce availability of physical space, imposing constraints on the degree distributions. In few words, spatial networks are different from other complex networks and as such they need to be studied in a different way.

In this thesis we focus on a particular class of spatial networks: **net-**

works of urban street patterns (c.f. Fig 2). We consider a database of 1-square mile samples of different world cities and for each city we construct a spatial graph. By construction, the resulting graphs are **planar graphs**, i.e. graphs forming nodes whenever two edges cross. In this thesis we present a comparative study of the basic properties of spatial networks belonging to different city street patterns. The main results we have obtained are presented in Chapters 3 and 4. In particular, in Chapter 3, we evaluate the characteristics of the graphs both at a *global* and at a *local* scale evaluating various quantities such as: **efficiency** [26, 27], **cost** proportional to the sum of street lengths [27], number of different motifs [28, 29]. In order to enable the classification and comparison of different kinds of city layout we need to introduce a proper normalization of the results.

The common procedure in relational (non-spatial) complex networks is to compare the properties of the original graph derived from the real system with those of some randomized versions of it, i.e. of graphs with the same number of nodes and links as the original one, but where the links are distributed at random. This is, for instance, the standard way proposed by Watts and Strogatz in [1] to assess whether a real system is a small world or not. The main problem with spatial graphs is that, in most of the cases, random graphs are no more a good way to normalize the results. In a planar graph, as those describing urban street patterns, the randomized version of the graph is not significative because it is almost surely a non-planar graph due to the edge crossings induced by the random rewiring of the edges. Moreover, because of the presence of long-range links, a random graph corresponds to an extremely costly street pattern configuration.

The alternative is to compare urban street patterns with grid-like structures. Using the method proposed by Buhl et al. in [24] as a starting point, we have considered two particular kinds of planar graphs, i.e. the

Figure 2: The urban street pattern graph of the city of Catania (Italy). The colours represent the values of *betweenness centrality* for each "street" where red means high values and blue low values of betweenness.

minimum spanning tree and the **greedy triangulation**, induced by the real distribution of nodes in the plane. Spanning tree and greedy triangulation will serve as the two extreme cases to normalize the various measures we have computed.

In Chapter 4 we deal with the typical problem of city planners. A urban planner is intrested in how the land is used and on the distribution of services and goods in the system with the main purpose of building systems better than the previous ones. For this reason we have focused on the issue of node centrality. In particular, we have set up a tool to extract the *backbone* of a city by convoluting centrality with the city

structure [30].and we have looked for a correlation between centrality and density of retail activities to gain information about the localization of retail activities.

From the physicists point of view, the study of networks of urban street patterns is whorthful for at least two reasons: The possibility to study particular man made networks as the geographical ones and the opportunity to apply well known models and techniques (like percolation or flux models to cite a few) to systems which are very different from the ones studied usually. Part of the work presented in this thesis has been published in Physical Review E [31], and in European Physical Journal B [30] in 2006 and in Environment and and Planning B [32] in 2009.

The thesis is organized in the following way:

- In Chap. 1 we present an introduction to the fundamental concepts of graph theory and to all the main measures that we will use in the following.

- In Chap. 2 we focus on the representation of a city as a graph and on the techniques used to compare graphs of cities of different kind. Here, we will introduce the concepts of minimum spanning tree (MST) and greedy triangulation (GT).

- In Chap. 3 we present a list of all the analysis performed and of the results obtained focusing on the study of both global and local properties of the investigated graphs.

- Finally, in Chap. 4, after introducing various measures of centrality, we discuss about the use of centrality analysis in networks of urban patterns. In particular we discuss about possibility to enlight the skeleton of a city and the relation between centrality and retail. We introduce also a future development of all the work shown in terms of dynamical evolution of the system.

CHAPTER 1

INTRODUCTION TO GRAPH THEORY

Many real-world situations can conveniently be described by means of a diagram consisting of a set of points together with lines joining certain pairs of these points. For example, the points could represent people, with lines joining pairs of friends; or the points might be communication centres, with lines representing communication links. Notice that in such diagrams one is mainly interested in whether two given points are joined by a line; the manner in which they are joined is immaterial. A mathematical abstraction of situations of this type gives rise to the concept of **graph**.

Graphs are so named because they can be represented graphically, and it is this graphical representation which helps us understand many of their properties. Each vertex is indicated by a point, and each edge by a line joining the points representing its ends. There is no single correct way to draw a graph; the relative positions of points representing vertices and the shapes of lines representing edges usually have no significance.

However, we often draw a diagram of a graph and refer to it as the graph itself; in the same spirit, we call its points "*vertices*" and its lines "*edges*". Most of the definitions and concepts in graph theory are sug-

gested by this graphical representation. The ends of an edge are said to be incident with the edge, and vice versa. Two vertices which are incident with a common edge are *adjacent*, as are two edges which are incident with a common vertex, and two distinct adjacent vertices are *neighbours*.

However, sometimes drawing a graph is completely useless, especially when the number of vertices and edges is very large. In fact, the big number of connections and points could transform a brief clear sketch into a complete mess. In addition, since positions of vertices are meaningless the "properties" of such systems cannot be deduced only by a visual analysis. This facts seems to limitate the range of validity of graph theory as it has been described until now. Fortunately for us, mathematics comes in our aid and provide a rigorous formalism under which all the properties of a graph can be expressed. So, for a proper comprehension of the results, it is necessary to introduce this common language providing some definitions and notations.

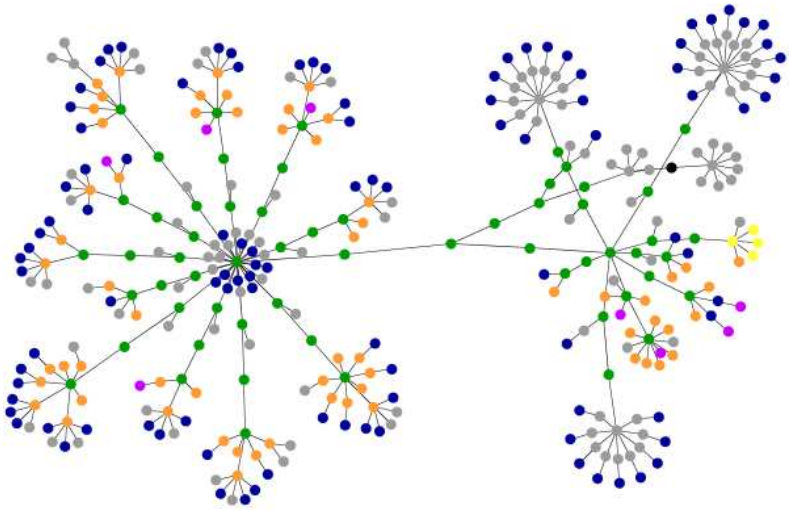


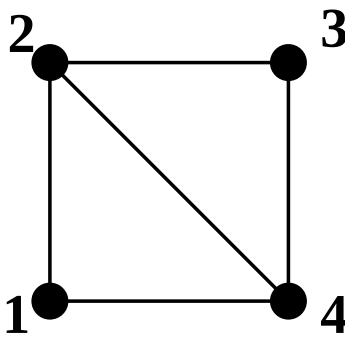
Figure 1.1: An example of graph graphical representation.

1.1 Fundamental Concepts

Let us start with the construction of the mathematical language used to describe graphs and in particular their structural properties. Graph theory [33–36] is the natural framework for the exact mathematical treatment of complex networks and, formally, a complex network can be represented as a graph. An **undirected** (*directed*) **graph** $G = (\mathcal{N}, \mathcal{L})$ is a mathematical object which consists of two sets: $\mathcal{N}$ and $\mathcal{L}$, such that $\mathcal{N} \neq \emptyset$ and $\mathcal{L}$ is a set of unordered (ordered) pairs of elements of $\mathcal{N}$. The elements of $\mathcal{N} \equiv \{n_1, n_2, \ldots, n_N\}$ are the **nodes** (or *vertices*, or *points*) of the graph G, while the elements of $\mathcal{L} \equiv \{l_1, l_2, \ldots, l_k\}$ are its **links** (or *edges*, or *lines*). The number of elements in $\mathcal{N}$ and $\mathcal{L}$ are denoted by N and K, respectively. Then, later, a graph will be indicated as $G(N, K) = (\mathcal{N}, \mathcal{L})$, or simply $G(N, K)$ or G_N, whenever it is necessary to emphasize the number of nodes and links in the graph.

A powerful way to represent a graph is the **adjacency** (or *connectivity*) **matrix** $\mathcal{A}$, a $N \times N$ square matrix whose entry a_{ij} $(i, j = 1, \ldots, N)$ is equal to:

$$a_{i,j} = \begin{cases} 1, & \text{if the edge between nodes } i \text{ and } j \text{ exists,} \\ 0, & \text{otherwise.} \end{cases}$$



$$\mathcal{A} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Figure 1.2: A graph and its corresponding adjacency matrix $\mathcal{A}$.

Certain types of graphs play prominent roles in graph theory. A **complete graph** is a graph in which any two vertices are adjacent so it has $\frac{N(N-1)}{2}$ edges.

However, many real networks display a large heterogeneity in the capacity and intensity of connections. Examples are the electrical resistance in resistors networks, passengers in airline networks, and the existence of weak ties between individuals in social networks [26, 37–40]. These systems can be better described in terms of **weighted networks**, i.e. networks in which each link carries a numerical value measuring the strength of the connection. In this sense a new set $\mathcal{W} \equiv \{w_1, w_2, \ldots, w_K\}$ must be considered. With this new position a graph G is defined as: $G^W = \{\mathcal{N}, \mathcal{L}, \mathcal{W}\}$.

When weighted networks are considered, it is useful to represent them using another matrix over the adjacency one. In these cases it is useful to consider a **weight matrix** $\mathcal{W}$, a $N \times N$ square matrix whose entry w_{ij} $(i, j = 1, \ldots, N)$ is equal to the link weight.

This thesis focuses on the **structural properties** of networks based on urban street patterns. This means that our networks are all *weighted* and *undirected*. Now it is time to define the quantities that will help us in the characterization of these properties.

1.1.1 Node degree, strength and their distributions

The **degree** (or *connectivity*) k_i of a node i is the number of edges incident with it. It is defined in terms of the adjacency matrix $\mathcal{A}$ as:

$$k_i = \sum_{j \in \mathcal{N}} a_{ij} \,. \tag{1.1}$$

In the case of weighted graphs it can also be defined a node **strenght** s_i which is the sum of the edges weights incident with the node i. In terms

of weight matrix $\mathcal{W}$ we could express it as:

$$s_i = \sum_{j \in \mathcal{N}} w_{ij} \,. \tag{1.2}$$

The most basic topological characterization of a graph G can be obtained in terms of the **degree distribution** $P(k)$, defined as the probability that a node uniformly chosen at random has degree k or, equivalently, as the fraction of nodes in the graph having degree k (analogously one can define a *strenght distribution* $P(s)$). Information on how the degree is distributed among the nodes of an undirected network can be obtained either by a plot of $P(k)$, or by the calculation of the moments of the distribution. The n-moment of $P(k)$ is defined as:

$$\langle k^n \rangle = \sum_k k^n \, P(k) \,. \tag{1.3}$$

The firs moment $\langle k \rangle$ is the **average degree** of G. The second moment measures the fluctuations of the connectivity distribution and the divergence of $\langle k^2 \rangle$ is responsable for the change of behaviour in dynamical processes acting on a net (in the limit of infinite systems) [41–43]. The degree distribution completely determines the statistical properties of uncorrelated networks as shown by Newman et al. in [44] using the approach based on *generating functions*.

In finite-size networks, fat tailed degree distributions have natural cut-offs [45]. When analyzing real networks, it may happen that the data have rather strong intrinsic noise due tu the finiteness of the sampling. Therefore, when the size of the system is small and the degree distribution $P(k)$ is heavy-tailed, it is sometimes advisable to measure the **cumulative** degree distribution $P_>(k)$, defined as:

$$P_>(k) = \sum_{k'=k}^{\infty} P(k') \,. \tag{1.4}$$

Indeed, when summing up the original distribution $P(k)$, the statistical fluctuations generally present in the tails are smoothed.

A large number of real networks are *correlated* in the sense that the probability that a node of degree k is connected to another node of degree, say k', depends on k. In these cases, it is necessary to introduce the ***conditional probability*** $P(k'|k)$, defined as the probability that a link from a node of degree k points to a node of degree k'. $P(k'|k)$ satisfies the degree detailed balance condition $kP(k'|k)P(k) = k'P(k'|k)P(k')$ [46, 47]. For uncorrelated graphs, the balance condition gives $P(k'|k)P(k) = k'P(k')/\langle k\rangle$.

Correlated graphs are classified as **assortative** if the average degree of nearest neighbour k_{nn} is an increasing function of k, whereas they are referred to as **disassortative** when k_{nn} is a decreasing function of k [48]. In other words, in assortative networks the nodes tend to connect to their connectivity peers, while in disassortative networks nodes with a lower degree are more likely connected with highly connected ones.

In general, assortative mixing, that is typical of many social systems, is not detected in urban networks. Such a result is probably related to a principle of hierarchy which drives rich streets to "order" the urban pattern at the local level: to have many rich streets intersecting each other would lead to a waste of land and financial resources, for one single "main street" can easily and rather successfully connect the most of an urban district [49].

1.1.2 Clustering

Clustering, also known as transitivity, is a typical property of acquaintance networks, where two individuals with a common friend are likely to know each other [10]. This can be quantified by defining the *transitivity* T of the graph as the relative number of triples, i.e. the fraction of connected nodes which also form triangles. An alternative possibility is to use the graph's **clustering coefficient** C, a measure introduced by Watts and Strogatz [1] defined as follows. A quantity c_i (local clustering

of node i) is introduced, expressing how likely $a_{jm} = 1$ for two neighbors j and m of node i. Its value is obtained counting the number of edges (denoted by e_i) in the subgraph G_i of node i neighbours (a local property). The local clustering coefficient is defined as the ratio between e_i and $k_i(k_i - 1)/2$, the maximum possible number of edges in G_i, so:

$$c_i = \frac{2e_i}{k_i(k_i - 1)} = \frac{\sum_{j,m} a_{ij}a_{jm}a_{mi}}{k_i(k_i - 1)}. \tag{1.5}$$

The clustering coefficient of the graph C is the average of c_i over all nodes in G:

$$C \equiv \langle c \rangle = \frac{1}{N} \sum_{j \in \mathcal{N}} c_i. \tag{1.6}$$

By definition, $0 \leqslant c_i \leqslant 1$, and $0 \leqslant C \leqslant 1$. It is also useful to consider $c(k)$, the clustering coefficient of a connectivity class k, which is defined as the average of c_i over all nodes with degree k.

1.1.3 Shortest path lengths and diameter

Graphs are often used to modelize systems in which something moves through them. Consider, for example, transportation or communication systems but also social systems have "things" moving through them (gossip news or infections to cite some). So it is very important to know which route is the best to ensure a fast and safe delivery as good as the extention and the robustness of the system in which we are moving through. In this section we focus on quantities that deal with such issues.

The shortest path plays an important role in transport and communication within a network. Suppose one needs to send a data packet from one computer to another through the Internet: the *geodesic* or **shortest path** is the shortest path connecting two nodes (in an unweighted network the minimum number of hops to reach node j from i). Geodesics provide optimal path way, since one would achieve a fast transfer and save of system resources. For such a reason, shortest paths have also

played an important role in the characterization of the internal structure of a graph [10]. It is useful to represent all the shortest path lengths of a graph G as a matrix $\mathcal{D}$ in which the entry d_{ij} is the length of the geodesic from node i to node j. The maximum value of d_{ij} is called the **diameter** of the graph. A measure of the typical separation between two nodes in the graph is given by the **average shortest path length** L (a global property), also known as the **characteristic path length**, defined as the mean of geodesic lengths over all couples of nodes [10, 50].

$$L = \frac{1}{N(N-1)} \sum_{i,j \in \mathcal{N}, i \neq j} d_{ij}. \tag{1.7}$$

A problem arising from the above definition is that L diverges if there are disconnected components in the graph. One possibility to avoid such a divergence is to limit the summation in Eq. (1.7) only to pairs of nodes belonging to the largest connected component, or **giant component** [51]. The other alternative, based on the so-called *efficiency* will be described in Sec. 3.3.1.

1.2 Planar Graphs

A graph is said to be *embeddable* in the plane, or **planar**, if it can be drawn in the plane so that its edges intersect only at their ends [35, 36]. Such a drawing is called a **planar embedding** of the graph. A planar embedding $\tilde{G}$ of a planar graph G can be regarded as a graph isomorphic to G; the vertex set of $\tilde{G}$ the edge set of $\tilde{G}$ is the set of points representing the vertices of G, the edge set of $\tilde{G}$ is the set of lines representing the edges of G.For this reason,we often refer to planar embedding $\tilde{G}$ of planar graph G as a **plane graph**, and we refer to its *points* as vertices and its *lines* as edges. However, when discussing both a planar graph G and a planar embedding $\tilde{G}$, in order to distinguish the two graphs, we call the vertices and edges of $\tilde{G}$ points and lines. An example of planar embedding of a graph is shown in Fig. 1.3.

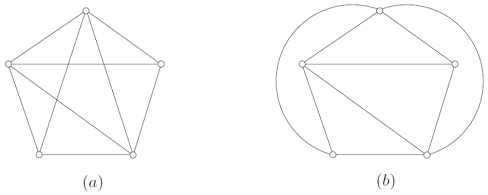


Figure 1.3: A graph (a) and its planar embedding (b).

1.2.1 The Jordan curve theorem

It is evident from the above definition that the study of planar graphs necessarily involves the topology of the plane. For further reading one could look at [52].

The results of topology that are especially relevant in the study of planar graphs are those which deals with simple curves. By **curve**, we mean a continuous image of a closed unit line segment. Analogously, a **closed curve** is a continuous image of a circle. A curve, or a closed curve, is **simple** if it does not intersect itself (in other word, if the mapping

is one-to-one). Properties of such curves come into play in the study of planar graphs because cycles in the plane graphs are simple closed curves.

A subset of the plane is **arcwise-connected** if any two of its points can be connected by a curve lying entirely within the subset. The basic result of topology that we need is the **Jordan curve theorem**.

Theorem 1.2.1 (The Jordan curve theorem)**.** *Any simple closed curve C in the plane partitions the rest of the plane in two disjoint arcwise-connected open sets.*

Although this theorem is intuitively obvious, giving a formal proof of it is quite trocky. The two open sets into which a simple closed curve C partitions the plane are called the **interior** and the **exterior** of C. We denote them by $\text{int}(C)$ and $\text{ext}(C)$, and their closures by $\text{Int}(C)$ and $\text{Ext}(C)$, respectively (thus $\text{Int}(C) \cap \text{Ext}(C) = C$). The Jordan curve theorem implies that every arc joining a point of $\text{int}(C)$ to a point of $\text{ext}(C)$ meets C in at least one point as shown in Fig. 1.4 The Jordan

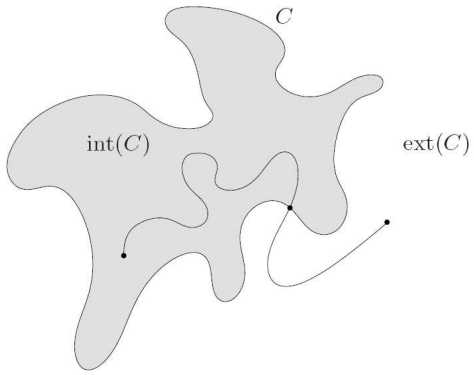


Figure 1.4: The Jordan curve theorem.

curve theorem is an useful tool that allows to declare if a graph is planar or not.

Theorem 1.2.2. *Given a complete graph with five nodes, such graph is nonplanar.*

Proof. By contradiction. Let G be a planar embedding of the complete graph K_5, with vertices v_1, v_2, v_3, v_4, v_5. Because G is complete, any two of its vertices are joined by an edge. Now the cycle $C \equiv v_1v_2v_3v_1$ is a simple closed curve in the plane, and the vertex v_4 must either lie in $\text{int}(C)$ or $\text{ext}(C)$. Without loss of generality, we may suppose that $v_4 \in \text{int}(C)$. Then the edges v_1v_4, v_4v_4, v_2v_4 all lie entirely in $\text{int}(C)$, too (apart from their ends v_1, v_2, v_3) as one can see in Fig. 1.5.

Consider the cycles $C_1 \equiv v_2v_3v_4v_2, C_2 \equiv v_3v_1v_4v_3,$ and $C_3 \equiv v_1v_2v_4v_1$. Observe that $v_i \in \text{ext}(C_i), i = 1, 2, 3$. Because v_iv_5 is an edge of G, and G is a plane graph, it follows from Jordan curve theorem that $v_5 \in \text{ext}(C_i), i = 1, 2, 3$ too. Thus $v_5 \in \text{ext}(C)$. But now the edge v_4v_5 crosses C, again by the Jordan curve theorem. This contraddicts the planarity of the embedding G. □

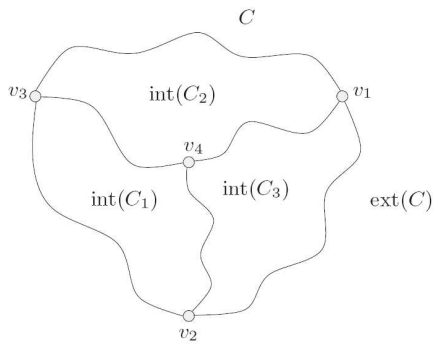


Figure 1.5: Proof of the non planarity of a complete graph with five nodes.

1.2.2 Subdivisions

Any graph derived from a graph G by a sequence of edge subdivisions is called a **subdivision** of G (or a *G-subdivision*).

Proposition 1.2.1. *A graph G is planar if and only if every subdivision of G is planar.*

Planar graph and graphs embeddable on the sphere are two faces of the same coin. To see this, we make use of a kind of mapping known as stereographic projection. A stereographic projection is made in this way: consider a sphere S resting on a plane P and denote by z the point that is diametrically opposite to the point of contact between S and P. The mapping between S and P made through z is called a *stereographic projection from* z and is illustrated in Fig. **??**.

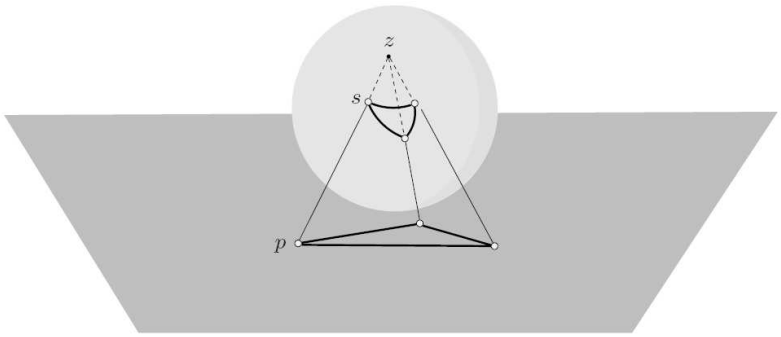


Figure 1.6: Stereographic projection.

Theorem 1.2.3. *A graph G is embeddable on the plane if and only if it is embeddable on the sphere.*

Proof. Suppose that G has an embedding $\tilde{G}$ on the sphere. Choose a point z of the sphere not in $\tilde{G}$. Then the image of $\tilde{G}$ under stereographic projection from z is an embedding of G on the plane. The converse is proved similarly. □

On many occasions, can be useful to consider embedding of graphs on the sphere; one example is proposed in [53].

1.3 Duality

1.3.1 Faces

A plane graph G partitions the rest of the plane into a number of arcwise-connected open sets. These sets are called the **faces** of G. An

example is shown in Fig. 1.7 in which the graph has five faces, namely, $f_1, f_2.f_3, f_4$ and f_5. Each plane graph has exactly one unbounded face called the **outer face**. In our example the outer face is f_1. We denote by $\mathcal{F}(G)$ and $f(G)$ the set of faces and the number of faces, respectively. The notion of faces applies also to embeddings of graphs on other surfaces.

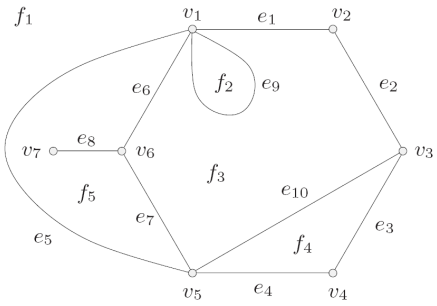


Figure 1.7: A plane graph with five faces.

The **boundary** of a face f is the boundary of the open set f in the usual topological sense. A face is said to be *incident* with the vertices and edges in its boundary, and two faces are **adjacent** if their boundaries have an edge in common. In Fig. 1.7, the face f_1 is incident with the vertices v_1, v_2, v_3, v_4, v_5 and the edges e_1, e_2, e_3, e_4, e_5; it is adjacent to the faces f_3, f_4, f_5.

We denote the boundary of a face with $\partial(f)$, the reason will be clear when we will discuss about duality. The boundary of a face may be regarded as a subgraph.

Proposition 1.3.1. *Let G be a planar graph, and let f be a face in some planar embedding of G. Then G admits a planar embedding whose outer face has the same boundary as f.*

By the Jordan curve theorem, a planar embedding of a cycle has exactly two faces. In the ensuing discussion of plane graphs, we assume, a number of other intuitively obvious statements concerning theri faces.

We assume, for example, that a planar embedding of a tree has just one face, and that, each face boundary in a connected plane graph is itself connected. Some of these facts rely on another basic result of plane topology, known as the Jordan-Schönfliess theorem.

Theorem 1.3.1 (The Jordan-Schönfliess theorem)**.** *Any homeomorphism of a simple closed curve in the plane onto another simple closed curve can be extended to a homeomorphism of the plane.*

One implication of this theorem is that any point p on a simple closed curve C can be connected to any point not on C by means of a simple curve which meets C only in p [52]. A cut edge in a plane graph has just one incident face, but we may think of the edge as being incident twice with the same face (once for each side); all other edges are incident with two distinct faces. We say that an edge *separate* the faces incident with it. The **degree**, $d(f)$ of a face f is the number of edges in its boundary $\partial(f)$, cut edges counted twice. In Fig. 1.7, the edge e_9 separates the faces f_2 and f_3 and the edge e_8 separate the face f_5 from itself; so $d(f_3)$ and $d(f_5)$ are 6 and 5 respectively.

Suppose that G is a connected plane graph. To **subdivide** a face f of G is to add a new edge e joining two vertices on its boundary in such a way that apart from its endpoints, e lies entirely in the interior of f. This operation results in a plane graph $G+ \equiv G + e$ with exactly one more face than G; all the faces of G except f are also faces of $G+$ and the face f is replaced by two new faces f_1 and f_2, which meet in the edge e as shown in Fig. 1.8

Theorem 1.3.2. *In a nonseparable plane graph with number of nodes greater or equal to three, each face is bounded by a cycle.*

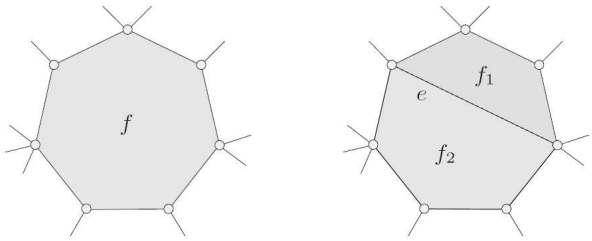


Figure 1.8: Subdivision of a face by adding an edge e. The right graph is called $G+$.

1.3.2 Duals

Given a plane graph G, one can define a second graph G^* as follows. Corresponding to each face f of G there is a vertex f^* of G^*, and corresponding to each edge e of G there is an edge e^* of G^*. Two vertices f^* and g^* are joined by the edge e^* in G^* if and only if their corresponding faces f and g are separated by the edge e in G. Observe that if e is a cut edge of G, then $f = g$, so e^* is a loop for G^*; conversely, if e is a loop of G, the edge e^* is a cut edge of G^*. The graph G^* is called the **dual** of G. The dual of plane graph in Fig. 1.3 is drawn in Fig. 1.9.

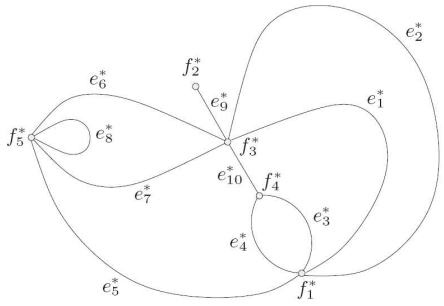


Figure 1.9: The dual of the plane graph shown in Fig. 1.7.

In the dual G^* of a plane graph G, the edges corresponding to those which lie in the boundary of a face f of G are just the edges incident with the corresponding vertex f^*- When G has no cut edges, G^* has no loops, and this set is precisely the trivial edge cut $\partial(f^*)$; that is $\partial(f^*) = \{e^* : e \in \partial(f)\}$.

It is easy to see that the dual G^* of a plane graph G is itself a planar graph; in fact, there is a natural embedding of G^* in the plane. We place each vertex f^* in the corresponding face f of G, and then draw each edge e^* in such a way that it crosses the corresponding edge e of G exactly once (and crosses no other edge of G) as shown in Fig. 1.10. It

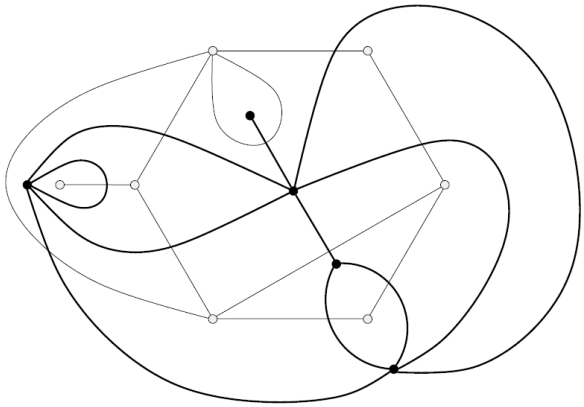


Figure 1.10: A plane graph and its plane dual. The dual is drawn using bold lines and each vertex is placed into its corresponding face.

is intuitively clear that we can always draw the dual as a plane graph in this way, but we do not prove this. We refer to such a drawing of the dual as a **plane dual** of the plane graph G.

Proposition 1.3.2. *The dual of any plane graph is connected.*

Proof. Let G be a plane graph and G^* a plane dual of G. Consider any two vertices of G^*. There is a curve in the plane connecting them which avoids all vertices of G. The sequence of faces and edges of G traversed by this curve corresponds in G^* to a walk connecting the two vertices. □

Although defined abstractly, it is often convenient to regard the dual G^* of a plane graph G as being itself a plane graph, embedded as described above. One may then consider G^{**} the dual of G^*. When G is connected, it is not difficult to prove that $G^{**} \cong G$ and a glance at Fig. 1.10 prove it.

The following relations are direct consequences of the definition of the dual G^*.

$$\mathcal{N}(G^*) = \mathcal{F}(G), \quad \mathcal{L}(G^*) = \mathcal{L}(G), \text{ and } d_{G^*}(f^*) = d_G(f)\ \forall f \in \mathcal{F}(G)\,. \tag{1.8}$$

Now, we can re-write a theorem in its dual version:

Theorem 1.3.3. *if G is a plane graph,*

$$\sum_{f\in\mathcal{F}} d(f) = 2K\,,$$

where K is the total number of edges in the graph.

Proof. Let G^* be the dual of G. By Eq. (1.8) and remembering that $\sum_{v\in\mathcal{N}} k_i = 2K$ we have:

$$\sum_{f\in\mathcal{F}(G)} d(f) = \sum_{f^*\in\mathcal{N}(G^*)} d(f^*) = 2\mathcal{L}(G^*) = 2\mathcal{L}(G) = 2K\,.$$

□

A simple connected plane graph in which all faces have degree three is called a **plane triangulation** or, simply **triangulation**. As a consequence of Eq. (1.8) we have:

Proposition 1.3.3. *A simple connected plane graph is a triangulation if and only if its dual is cubic.*

It is easy to show that every simple plane graph on three or more vertices is a spanning subgraph of a triangulation. On the other hand, no simple spanning supergraph of a triangulation is planar. For this reason, triangulations are also known as **maximal planar graphs**. They play an important role in the theory of planar graphs.

1.4 Euler's formula

There is a simple formula relating the number of vertices, edges and faces in a connected plane graph. It was first established for polyhedral graphs by Euler in 1752 and is known as *Euler's formula*.

Theorem 1.4.1 (Euler's formula)**.** *For a connected plane graph G the following relation holds:*

$$N(G) - K(G) + f(G) = 2\,. \tag{1.9}$$

Proof. By induction of $f(G)$, the number of faces of G. if $f(G) = 1$, each edge of G is a cut edge and so G, being connected, is a tree. In this case $K(G) = N(G) - 1$ and the assertion holds. Suppose that is true for all connected plane graph with fewer than f faces, where $f \geq 2$, and let G be a connected plane graph with f faces. Choose an edge e of G that is not a cut edge. Then $G^- \equiv G - e$ is a connected plane graph with $f - 1$ faces, because the two faces of G separated by e coalesce to form one face of G^-. By the induction hypotesis,

$$N(G^-) - K(G^-) + f(G^-) = 2\,.$$

Using the relations:

$$N(G^-) = N(G), \quad K(G^-) = K(G) - 1, \text{ and } f(G^-) = f(G) - 1\,,$$

we obtain

$$N(G) - K(G) + f(G) = 2\,.$$

The theorem follows by induction. □

Corollary 1.4.1. *All planar embeddings of a connected planar graph have the same number of faces.*

Proof. Let $\tilde{G}$ be a planar embedding of a planar graph G. By Euler's formula Eq. (1.9), we have:

$$f(\tilde{G}) = K(\tilde{G}) - N(\tilde{G}) + 2 = K(G) - N(G) + 2\,.$$

Thus the number of faces of $\tilde{G}$ depends only on the graph G, and not on its embedding. □

Corollary 1.4.2. *Let G be a simple planar graph on at least three vertices. Then $K \leq 3N - 6$. Furthermore, $K = 3N - 6$ if and only if every planar embedding of G is a triangulation.*

Proof. It clearly suffices to prove the corollary for connected graphs. Let G be a simple connected planar graph with $n \geq 3$. Consider any planar embedding $\tilde{G}$ of G. Because G is simple and connected, on at least three vertices, $d(f) \geq 3$ for all $f \in \mathcal{F}(\tilde{G})$. Therefore, remembering that $\sum_{f\in\mathcal{F}} d(f) = 2K$ and Euler's formula (1.9):

$$2K = \sum_{f\in\mathcal{F}(\tilde{G})} d(f) \geq 3f(\tilde{G}) = 3(K - N + 2)\,, \tag{1.10}$$

or, equivalently,

$$K \leq 3N - 6\,. \tag{1.11}$$

Equality holds in Eq. 1.11 if and only if it holds in Eq. 1.10, that is, if and only if $d(f) = 3$ for each $f \in \mathcal{F}(\tilde{G})$. □

Corollary 1.4.3. *Every simple planar graph has a vertex of degree at most equal to five.*

Proof. This is trivial for $N < 3$. If $N \geq 3$. Then remembering that $\sum_i k_i = 2K$ and the previous corollary we found:

$$\delta N \leq \sum_{v\in\mathcal{N}} d(v) = 2K \leq 6N - 12\,.$$

It follows that $\delta \leq 5$. □

The definitions and theorems presented in this chapter tell much about the importance of planar graphs. As we will see soon, a city can be mapped into a planar graph. However, cities are not the only examples of networks that can be embedded in a plane. Fig. 1.11 show some examples of planar graph that we can encounter in everyday life and in many differnt sectors of science.

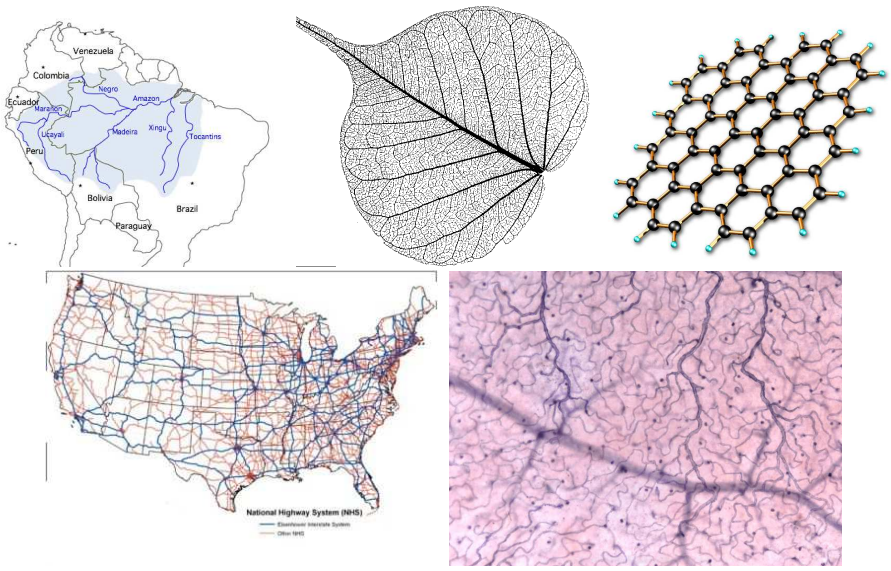


Figure 1.11: Some examples of plane graphs that can be found in everyday life. From top left in clockwise sense: Amazon river basin, a leaf venation system, a graphene sheet, human arterial venation system and the US highway system.

1.5 Fractal Box-Counting Dimension d_{box}

What is the "dimension" of a set of points? For familiar geometric objects, the answer is clear: lines and smooth curves are one-dimensional, planes and smooth surfaces are two-dimensional, solids are three-dimensional, and so on. If forced to give a definition, we could say that the dimension is the minimum number of coordinates needed to describe every point in the set. When we try to apply this definition to fractals, we quickly run into paradoxes. Consider, for example, the von Koch curve (Fig. 1.12): What is the dimension of von Koch curve? Since it is a curve you might be tempted to say that is one. But the trouble is that it has infinite length! To prove this, observe that if the lenght of E_0 is L_0, then the length of E_1 is $L_1 = \frac{4}{3}L_0$. The length increases by a factor $\frac{4}{3}$ at each stage, so $L_n = (\frac{4}{3})^n L_0$ and $L_n \to \infty$ as $n \to \infty$.

Moreover, the arc length between any two points on the curve is in-

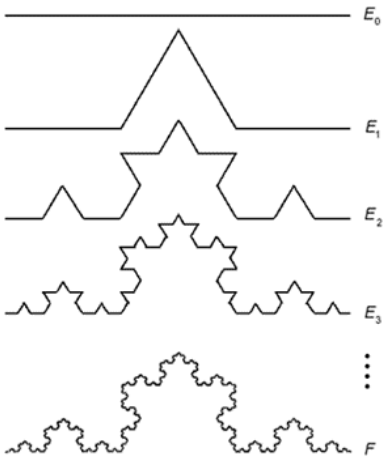


Figure 1.12: Iterative procedure to generate a von Koch curve. At each step, each segment is divided in three parts. Then the central part is removed and replaced with two segments of equal length. This procedure can be repeated as many times as one want. Here we report the first four step and the resulting curve when the number of iterations become big.

finite. Hence points on the curve are not determined by their arc length from a particular point, because every point is infinitely far from every other. This suggest that the curve is more than one-dimensional. However, would we really want to say that it is two-dimensional? It certainly does not seem to have any "area". So the dimension should be *between* 1 and 2, whatever that means.

Same discussion applies for the set of points which represents the nodes of a spatial graph. Since they are displaced over a surface, we can think to give dimension 2 to this set. Nevertheless, they do not have an "area" like the points of the von Koch curve and does not lay on a curve, suggesting that their dimension must be higher than 1.

Any object having a dimension which is not integer is called a **fractal** [54]. Now the question became: how can we calculate the **fractal dimension** of a set of points? To deal with fractals that are not **self-similar** (i.e. they are made of scaled-down copies of themselves, all the

way down to arbitrary small scales), various definitions have been proposed [55]. All the definitions share the idea of "measurement at a scale ε": we measure the set in a way that ignores irregularities of size less than ε, and then study how measurement vary as $\varepsilon \to 0$. Consider a

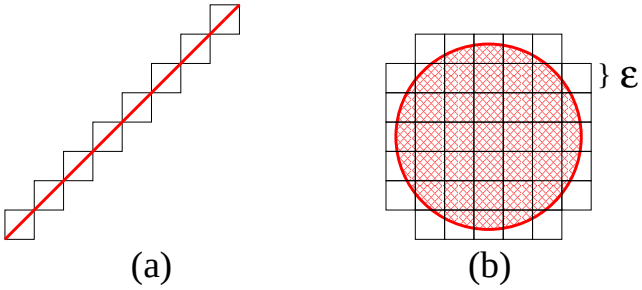


Figure 1.13: An example of square covering for a smooth curve (a) and for a planar region (b).

set of points in a D-dimensional Euclidean space. Let $N(\varepsilon)$ be the minimum number of D-dimensional cubes of side ε needed to cover the set. How does $N(\varepsilon)$ depends on ε? Consider, for example, the cases shown in Fig. 1.13. For a smooth curve of length L (Fig. 1.13 (a)), $N(\varepsilon) \propto L/\varepsilon$; for a planar region of surface A bounded by a smooth curve (Fig. 1.13 (b)), $N(\varepsilon) \propto A/\varepsilon^2$. The key observation is that the dimension of the sets equals the exponent d in the power law $N(\varepsilon) \propto 1/\varepsilon^d$. This relation holds also for fractals, in which the dimension d is no longer integer. The **box counting dimension** is defined as:

$$d = \lim_{\varepsilon \to 0} \frac{\ln N(\varepsilon)}{\ln(1/\varepsilon)}, \quad \text{if the limit exist.} \tag{1.12}$$

So if one plot the logarithm of $N(\varepsilon)$ versus the logarithm of $1/\varepsilon$ he would find a straight line of slope d which gives the estimate of the box dimension. As shown in Fig. 1.14 there are also points with a constant value of $\ln N(\varepsilon)$. These points represent the cases in which the box is too small or either too big and always contains the same number of points. However, the box dimension is rarely used in practice. Its computation requires too much storage space and computer time, compared to other types of

fractal dimension. Grassberger and Procaccia [56] in 1983 proposed a more efficient approach tha has become standard. Consider a point x in the Euclidean space. Let $N_x(\varepsilon)$ denote the number of points of the system inside a ball of radius ε centered on x. Now vary ε. As ε increases, the number of points in the ball tipically grows and if one averages over many x we found that this quantity scales as a power law:

$$\langle N_x(\varepsilon) \rangle \propto \varepsilon^d \,,$$

where the average is meant over the points x. d is called **correlation dimension**. In general, $d_{\text{correlation}} \leq d_{\text{box}}$, although they are usually very close [56].

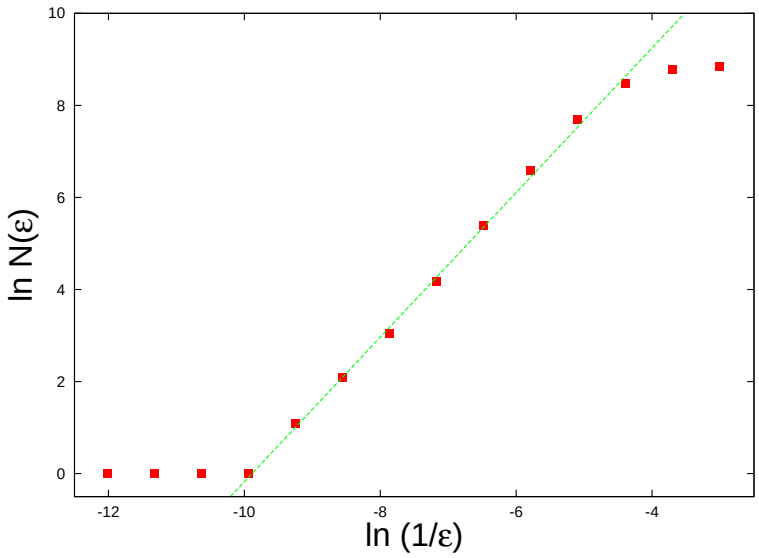


Figure 1.14: Logarithm of number of squares needed to cover the set $N(\varepsilon)$ as a function of the logarithm of the inverse of the square size ε. The data (red squares) refers to the whole city of Leicester (UK) while the dashed line is the linear fit which gives the box dimension ($d_{\text{box}} = 1.57$). (*data courtesy of the City Form project*).

CHAPTER 2

GRAPHS OF URBAN STREET PATTERNS

After the brief theoretical introduction of the previous chapter on planar graphs, we are now ready to discuss the issue of extracting and converting an urban street pattern into a plane graph. To do so, we need to introduce the basic notions on the GIS, the geographical information system. Moreover, as we have seen before we have to deal with the possibility of choosing a plane graph or its dual, and with the definition of a good methodology to compare the characteristics of graphs of different cities. The comparison requires the use of two particular plane graphs and the algorithms used to extrapolate those graphs from the original one are here explained. The result of the analysis will be presented in the next chapter.

2.1 Geographical Information Systems

A **geographic information system**, or **geographical information system** (GIS), is any system that integrates hardware, software, and data to captures, stores, analyzes, manages, and presents data that are linked to location [57, 58]. In the simplest terms, GIS is the merging of cartography, statistical analysis, and database technology. GIS

systems are used in cartography, remote sensing, land surveying, utility management, natural resource management, photogrammetry, geography, urban planning, emergency management, navigation, and localized search engines. GIS allows us to view, understand, question, interpret, and visualize data in many ways that reveal relationships, patterns, and trends in the form of maps, globes, reports, and charts. In a more generic sense, GIS applications are tools that allow users to create interactive *queries* (user-created searches), analyze spatial information, edit data, maps, and present the results of all these operations.

The use of GIS establish its foundation in 1854 when John Snow depicted a cholera outbreak in London using points to represent the locations of some individual cases, possibly the earliest use of the geographic method [59]. His study of the distribution of cholera led to the source of the disease, a contaminated water pump (the Broad Street Pump, whose handle he had disconnected, thus terminating the outbreak) within the heart of the cholera outbreak.

Figure 2.1: A reconstruction of John Snow map's. The points represent the cholera cases.

While the basic elements of topography existed previously in cartography, the John Snow map was unique, using cartographic methods not

only to depict but also to analyze clusters of geographically dependent phenomena for the first time.

The year 1962 saw the development of the world's first true operational GIS in Ottawa, Ontario, Canada by the federal Department of Forestry and Rural Development. Developed by Dr. Roger Tomlinson, it was called the "Canada Geographic Information System" (CGIS) and was used to store, analyze, and manipulate data collected for the Canada Land Inventory (CLI) – an effort to determine the land capability for rural Canada by mapping information about soils, agriculture, recreation, wildlife, waterfowl, forestry, and land use at a scale of 1:50,000. A rating classification factor was also added to permit analysis.

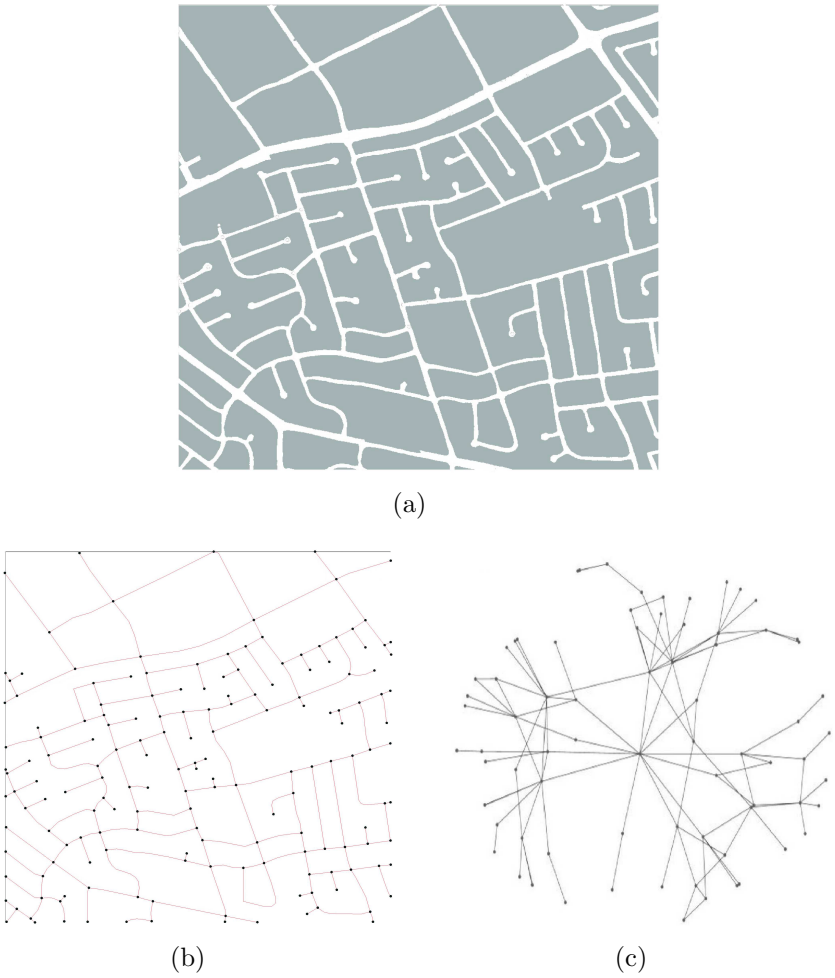
GIS is a powerful tool for urban planners since it allows to conjugate the network approach with the availability of informations which are spatially distributed like retail or population densities [32]. As we will see below, Using GIS systems is possible to convert city maps into graphs.

2.2 The Primal Approach and The Dual Approach

How can we convert the usual representation of a city as a map printed on a sheet of paper into a plane graph dataset to be used in computer simulation? The theory itself suggests that different approaches could be used [35, 36], but here we focus on only two of them which are the most used in urban planning [23, 49, 60, 61]. From now on, we will refer to them as the **primal** and the **dual** approach.

- In the *primal approach* one focus on intersections mapping road crossings as nodes and streets as edges. The advantage of this approach are that it is very intuitive and that the outcome of this mapping is unique [25, 62, 63].

- Conversely, the *dual approach* mapps streets as nodes and intersections between streets as links. This approach has the advantage to focus on streets (useful in route finding with GPS navigator [64]). Unfortunately, focusing on the concept of street the outcome of this kind of mapping is not unique (dependent on street naming, line of sight, etc.) [61, 65, 66].



(a) (b) (c)

Figure 2.2: Different approaches to map a city into a graph. The original city map of Walnut Creek (USA) (a), the primal graph (b) and the dual one (c).

An example of the outcome of the two approaches is showed in Fig. 2.2. In this work we decided to adopt the primal approach to map the cities we are going to study. In particular, we have imported twenty maps into a GIS environment and constructed the correspondent spatial graphs of street networks by using a road-centerline-between-nodes format [62]. Namely, each urban street pattern is trasformed into an undirected, weighted graph $G = (\mathcal{N}, \mathcal{L})$, embedded in the 2-dimensional unit square. $\mathcal{N}$ is the set of N nodes representing street intersections and characterized by their positions $\{x_i, y_i\}_{i=1,...,N}$ in the square. $\mathcal{L}$ is the set of K links representing streets. The links follow the footprints of real streets and are associated a set of real positive numbers representing the street lengths, $\{l_k\}_{k=1,...,K}$. The graph is then described by the adjacency $N \times N$ matrix $\mathcal{A}$, and by a $N \times N$ matrix $\mathcal{W}$, whose entry w_{ij} is equal to the length of the street connecting node i and node j. In this way both the topology and the geography (metric distances) of the system will be taken into account.

2.3 Greedy Triangulation and Minimum Spanning Tree

We have previously commented on the fact that in order to compare different graphs or in order to say if a certain feature is statically significative one has to "normalize" the results with respect to a *randomized version* of the graph under study [2, 9, 33, 35, 50]. We have also found that, planar graphs are those graphs forming vertices whenever two edges cross, whereas non-planar graphs can have edge crossings that do not form vertices [35]. So the tecnique of studying graphs with the same number of nodes and edges but with edges randomly rewired is no longer good since does not guarantee planarity as one can see in Fig. 2.3.

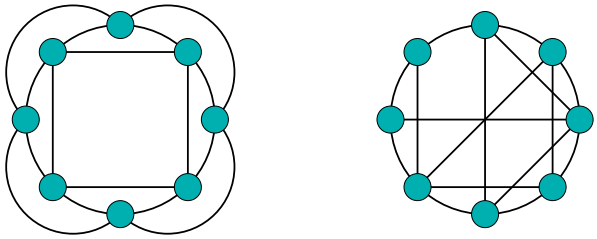


Figure 2.3: A typical example of randomization. The graph edges are rewired randomly while preserving the number of nodes N and edges K.

The graphs representing urban street patterns are, by construction, planar, and we will then compare their structural properties with those of minimally connected and maximally connected planar graphs. In particular, following Buhl et al. [24], we consider the **Minimum Spanning Tree** (MST) and the **Greedy Triangulation** (GT) induced by the distribution of nodes (representing street intersections) in the square.

Spanning trees are the planar graphs with the minimum number of links in order to assure connectedness, while *greedy triangulations* are graphs with the maximum number of links compatible with the planarity. So, these graphs represent the lower and upper bound to the number of edges that a plane graph can have. However, we do not have only to deal with the problem of planarity but also with limited resources. In fact, there is a cost associated to the creation of a road. This may depend on many factors (type of street, its width, materials used, to cite a few) but the simplest dependece is on mere street length. We define the **cost** as the sum of the street lengths. In formula:

$$W = \sum_{i,j} a_{ij}\, w_{ij}\,, \tag{2.1}$$

where:

$$w_{ij} = d_{ij}^{\mathrm{Eucl}} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{2.2}$$

is here taken to be equal to the Euclidean distance d_{ij}^{Eucl}. Now it is time to give a proper definition of these two special graphs.

The **Minimum Spanning Tree** (MST) is the shortest tree which connects every nodes into a single connected component. By definition the MST is an acyclic graph that contains $K_{\min} = N - 1$ links. This is the minimum number of links in order to have all the nodes belonging to a single connected component [35, 36]. At the other extreme, as proved in the previous chapter, the maximum number of links, $K_{\max}$, that can be accomodated in a planar graph with N nodes (without breaking the planarity) is equal to $K_{\max} = 3N - 6$ [36]. The natural reference graph should be then the **Minimum Weight Triangulation** (MWT), which is the planar graph with the highest number of edges $K_{\max}$, and that minimize the total length.

Since no polynomial time algorithm is known to compute the MWT, we thus consider the **Greedy Triangulation** (GT), that is based on connecting couples of nodes in ascending order of their distance provided that no edge crossing is introduced [67]. The GT is easily computable and leads to a maximal connected planar graph, while minimizing as far as possible the total length of edges considered. In Fig. 2.4 we report an example of the graphs that we have extracted from the city maps that we have studied in [31].

2.4 Algorithms for calculating MST and GT

We have seen that a weighted undirected graph could be represented using two matrices: the adjacency $\mathcal{A}$ and the weight $\mathcal{W}$ ones. However, such matrices are sparse i.e. contains a lot of null elements. So, mapping a graph on a computer using its adjacency matrix could become very expensive in terms of memory consumption (especially when the number of nodes N becomes big). A way to overcome this problem is to store the matrix using one of the sparse matrix structure [68]. The most common

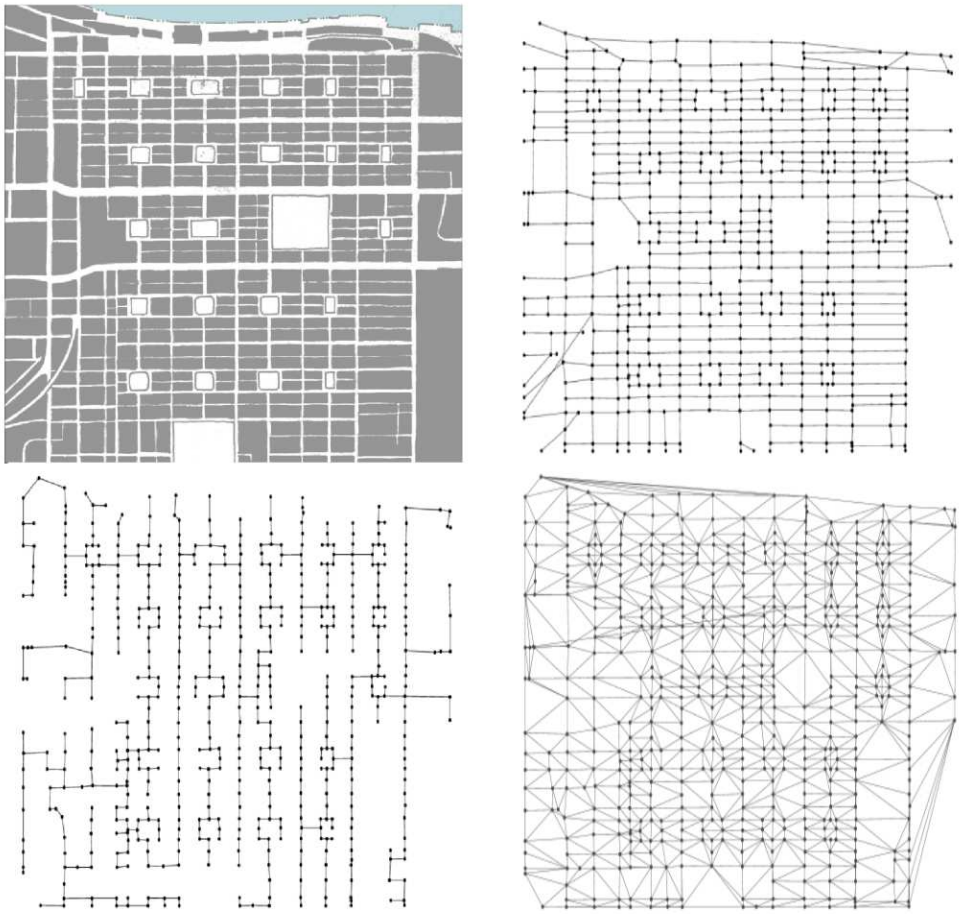


Figure 2.4: The urban pattern of Savannah (USA) as it appears in the original map (top left), and the correspondent spatial graph (top right). The resulting MST (bottom left) and GT (bottom right).

are: *compressed sparse column* (CSC), *compressed sparse row* (CSR) and *coordinate list* (COO). Let us describe the **coordinate list**, which is the one we choose to adopt in our simulations. A sparse matrix in coordinate format, also known as *triplet*, is a structure whose element stores a list of (`row, column, value`) tuples. Ideally, the entries are sorted (by row index, then column index) to improve random access times. Let us explain the role of each field of the tuples, considering an element $a_{ij} = c$ of a sparse matrix $\mathcal{A}$ its COO representation will be:

`row` the row index of the element. In our case: i;

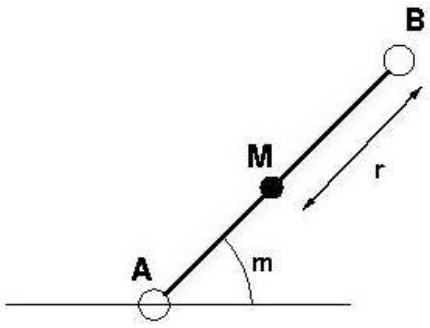
`column` the column index of the element. In our case: j;

`value` the value of the element. In our case: c.

In the case of graphs, for each link `row` and `column` represent the id of the nodes incident with that link and `value` it is the weight of the link itself (in the case of unweighted graphs this information can be omitted allowing further memory saving). The advantages of COO representation lay in its easy convertibility to and from other sparse matrix formats, while its disadvantages are relatet with the fact that it does not allow direct mathematic operations on it [69].

In our dataset we added even more entries in the COO tuples in order to store more informations. A tipical entry looks like this:

N_{from}	X_{from}	Y_{from}	X_{mean}	Y_{mean}	N_{to}	X_{to}	Y_{to}	r	m
28	42.0	38.0	42.5	38.0	44	42.0	38.0	3.50	4.18



where:

$N_{\textbf{from}}$**:** ID of the source node;

$X_{\textbf{from}}$**:** x coordinate of the source node;

$Y_{\textbf{from}}$**:** y coordinate of the source node;

$X_{\textbf{mean}}$**:** x coordinate of the mean point M of segment AB;

$Y_{\mathbf{mean}}$**:** y coordinate of the mean point M of segment AB;

$N_{\mathbf{to}}$**:** ID of the destination node;

$X_{\mathbf{to}}$**:** x coordinate of the destination node;

$Y_{\mathbf{to}}$**:** y coordinate of the destination node;

r**:** radius $\frac{d_{AB}}{2}$;

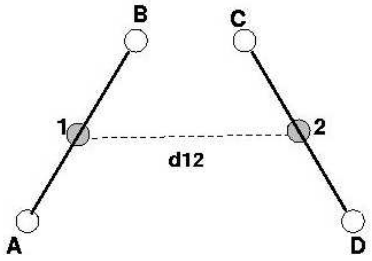
m**:** angular coefficient of segment AB.

These informations are used by both the algorithms that compute the GT and the MST that we are going to explain. To construct both the MST and the GT induced by the spatial distribution of points (nodes) $\{x_i, y_i\}_{i=1,\dots,N}$ in the unit square, we sorted out all the couples of nodes, representing all the possible edges of a complete graph, by ascending order of their length.

2.4.1 The greedy triangulation algorithm

The algorithm we adopted for computing the GT is a *brute force* one based on some peculiar characteristics of planar GT [67], i.e. it performs a series of checks to say if an edge could be placed in the GT or not. The algorithm browses the ordered list of edges in ascending order of length and, for each edge, it checks whether adding the edge produces any intersections with any other edge that have been already added. The checks are divided into two levels. First level checks are easy to implement and act as an high level filter over the list of edges to select only the "worthful" cases avoiding an explosion of the computational complexity. The second level checks, instead, are much more complex and allow to individuate without mistakes the edges belonging to the GT.

Taking two edges, AB (belonging to GT) and CD (to check if it belongs to GT) and say m_1, m_2 the angular coefficients; $1, 2$ the middle points and r_1, r_2 the radius of the segments, we have:



Step 1 If d_{12} is the distance between point 1 and 2

$$d_{12} \begin{cases} > (r_1 + r_2) & \text{CD does not cross AB and can be added;} \\ \leq (r_1 + r_2) & \text{we must go to Step 2.} \end{cases}$$

Step 2 Supposing that the vertex of an edge are labeled such that $x_A < x_B$ and $x_C < x_D$, consider the linear system associated with the equations of the straight lines which the two segments AB and CD belong to.

$$\begin{cases} Y - y_1 &= m_1(X - x_1) \qquad X \in [X_A, X_B] \\ Y - y_2 &= m_2(X - x_2) \qquad X \in [X_C, X_D] \end{cases}$$

From this we can extract the coordinates of the hypotetic intersection point X and Y:

$$\boxed{X = \frac{y_1 - m_1 x_1 - (y_2 - m_2 x_2)}{m_2 - m_1}.}$$

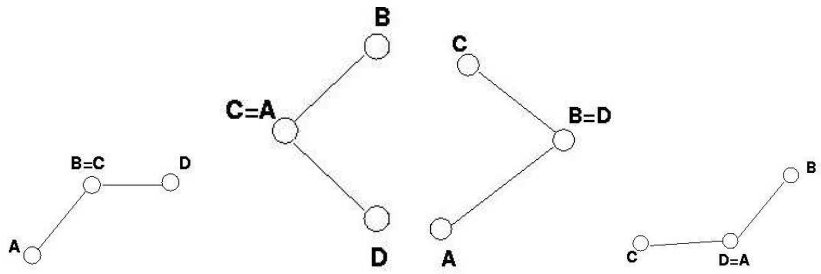
Analogously one can calculate:

$$\boxed{Y = \frac{m_2 y_1 - m_1 y_2 + m_1 m_2 (x_2 - x_1)}{m_2 - m_1}.}$$

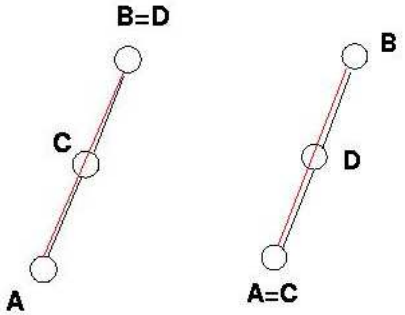
Step 3 If $X \in ([X_A, X_B] \wedge [X_C, X_D])$, then the two edges will surely cross each other and the edge CD has to be rejected from the GT list because, since we are on a planar graph, two edges cannot cross themselves without forming a node.

The first level check system illustrated above is very powerful but substantialy WRONG since if one of the two end points of an edge coincide with one of the other two the edge will be purged from the list even if it could belong to GT. To avoid that, during the implementation of the algorithm further checks have been added to take into account those cases.

Step 3.1 Accept all the edges that have $X \in ([X_A, X_B] \wedge [X_C, X_D])$ but with one extreme in common. The possible configurations could be divided into four classes which are showed below.

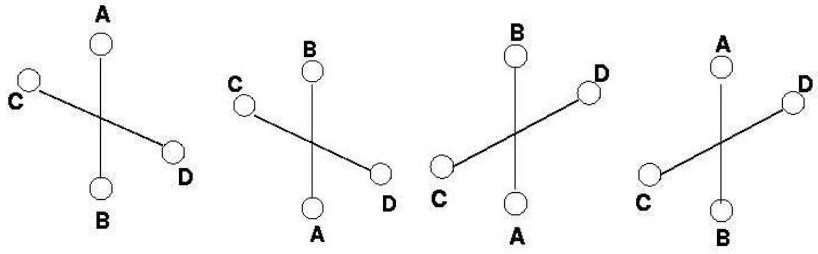


Step 3.1.1 Moreover, we have to purge all those edges that have an extreme in common but that are "parallel", i.e. edges which superpose over existing ones as shown below (the red edge is the superposing one).

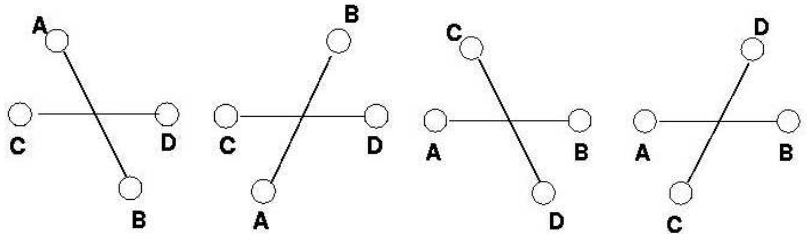


Unfortunately, the checks based on the analysis of parameter X (or eventually Y) are not enough to eliminate all the edges that does not belong to GT. In particular, it has been observed that when one or more edges are parallel to an axis the algorithm shows some bugs, related to the angular coefficient m, that can be eliminated adding another check. This check is of the same kind of the one used on X but, this time, applicated on Y. In this way we can avoid certain peculiar configurations that can be encountered during simulations.

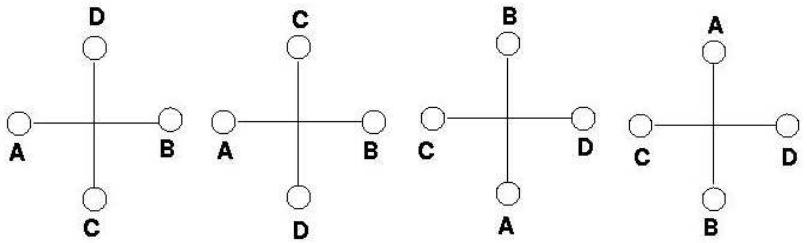
Step 4 Purge all the edges being in one of the following configurations:



Edges parallel to the y axis.



Edges parallel to the x axis.



Orthogonal edges.

In the computed GT, the graphs we have obtained have less than

$3N-6$ edges. This is due to the fact that we have considered all the edges as being straight-lines connecting the two end-nodes. For such a reason some of edges connecting the external nodes (the nodes on the border of the unit square) cannot be placed without causing edge crossings. However, the number of these edges is of the order of $\sqrt{N}$ and this issue has a minor effect on the results.

2.4.2 The Kruskal algorithm

To compute the MST we have used the Kruskal algorithm [70, 71]. The algorithm consists in browsing the ordered list of edges, starting from the shortest edge and progressing toward the longer ones. Each edge of the list is added if and only if the graph obtained after the edge insertion is still a *forest* or it is a *tree*. A **forest** is a disconnected graph in which any two elements are connected by at most one path, i.e., a disconnected ensemble of **trees**. (In practice, one checks whether the two end-nodes of the edge are belonging or not to the same component). With this procedure, the graph obtained after all the links of the ordered list are considered is the MST. In fact, when the last link is included in the graph, the forest reduces to a single tree. Since in the Kruskal algorithm an edge producing a crossing would also produce a cycle, following this procedure prevents for creating edge crossings. Using an algorithmic design it looks like this:

$A \leftarrow \emptyset$
for each edge $(u, v) \in \mathcal{L}(G)$, taken in increasing order of weight w **do**
  if $u, v \notin \mathcal{N}(A)$ **then**
    $A \leftarrow A \cup \{(u, v)\}$
  end if
end for
return A

Where G is the original graph and A is the MST. An example of how the Kruskal algorithm works is shown in Fig. 2.5.

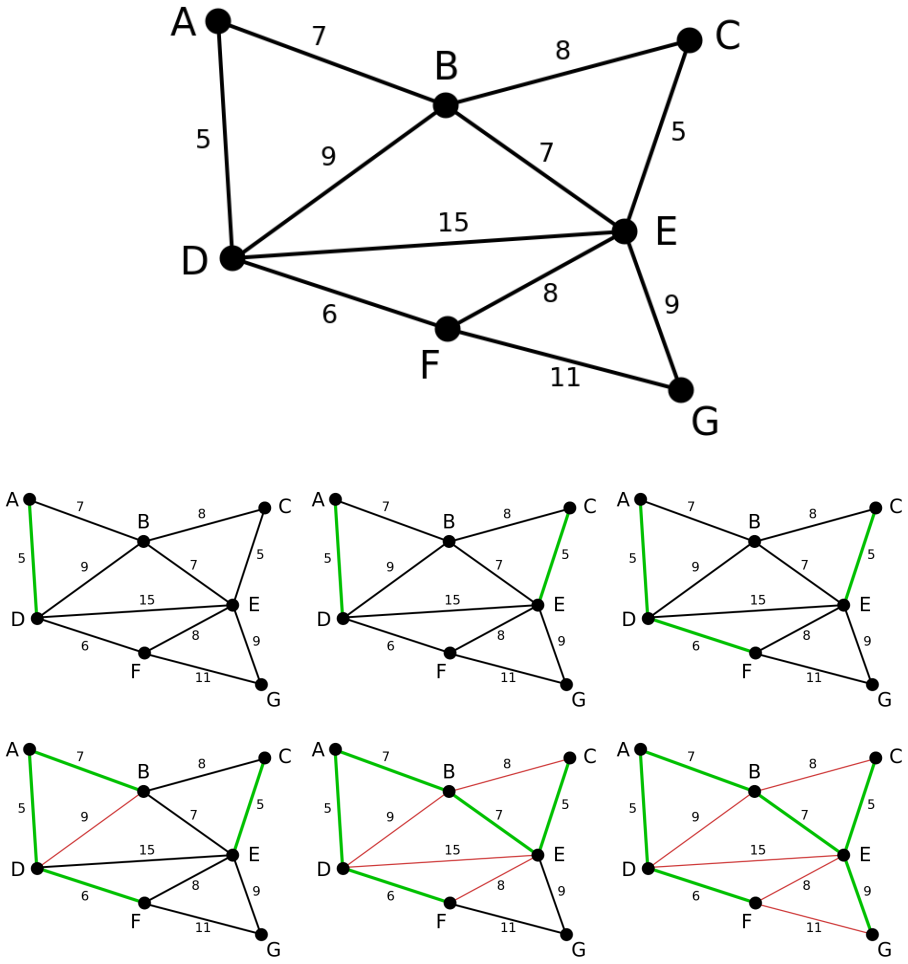


Figure 2.5: An example of how the Kruskal algorithm works. Top picture represents the original graph from which we have to extract the MST. The six pictures represent in clockwise sense choosing and adding the six edges that form the MST. The green edges are those belonging to the MST while the red ones are discarded because either they form loops or have an higher weight w.

CHAPTER 3

STRUCTURAL PROPERTIES OF PLANAR GRAPHS OF URBAN STREET PATTERNS

In this chapter we present the dataset used and the results obtained on both its *local* and *global* properties. Namely, we try to to establish if there are some peculiar features that allows to discriminate the different species of urban patterns. After a first characterization based on a topological analysis, we pass to investigate the structural properties and finally we try to understand how the resources are used in our cities analyzing the behaviour of the cost versus efficiency.

For each city we have constructed the respective MST and GT. These two bounds make also sense as regards as the possible evolution of a city: the most primitive forms are close to trees, while more complex forms involve the presence of cycles. We can then compare the structural properties of the original graphs representing the city with those of the two limiting cases represented by MST and GT.

3.1 Dataset and its basic properties

The database we have studied consists of twenty 1-square mile samples of different world cities, selected from the book by Allan Jacobs

[72]. A list of the considered cities is reported in Tab. 3.1, together with the basic properties of the derived graphs. The considered cases exhibit striking differences in terms of cultural, social, economic, religious and geographic context. In particular, they can be roughly divided into two large classes:

1. Patterns grown throughout a largely self-organized, fine-grained historical process, out of the control of any central agency;

2. Patterns realized over a short period of time as the result of a single plan, and usually exhibiting a regular grid-like, structure.

Ahmedabad (India), Cairo (Egypt) and Venice (Italy) are the most representative examples of self-organized patterns, while Los Angeles (USA), Richmond (USA), and San Francisco (USA) are typical examples of mostly-planned patterns. We have selected two different parts of the city of Irvine, CA (USA), (named Irvine 1 and Irvine 2) for two highly diverse kinds of urban fabrics: the first is a sample of an industrial area showing enormous blocks with few intersections while the second is a typical residential early sixties "lollipop" low density suburb based on a tree-like layout with a lot of dead-end streets.

The differences between cities are already evident from the basic properties of the derived graphs. In fact, the number of nodes N, the number of links K, and the cost of the wiring W, measured in meters, assume widely different values, notwithstanding the fact we have considered the same amount of land (Tab. 3.1). Notice that Ahmedabad has 2870 street intersections and 4387 streets in a surface of 1-square mile, while Irvine 1 has only 32 intersections and 37 streets. Ahmedabad and Cairo are the cities with the largest cost, while the cost is very small (less than 40000 meters) in Barcelona, Brasilia, Irvine, Los Angeles, New Delhi, New York, San Francisco, Washington and Walnut Creek. A large difference is also present in the average edge (street) length $\langle l \rangle$, that assumes

	CITY	N	K	W	$\langle l \rangle$	D_{box}
1	Ahmedabad	2870	4387	121037	27.59	1.92
2	Barcelona	210	323	36179	112.01	1.99
3	Bologna	541	773	51219	66.26	1.95
4	Brasilia	179	230	30910	134.39	1.83
5	Cairo	1496	2255	84395	37.47	1.82
6	Irvine 1	32	36	11234	312.07	–
7	Irvine 2	217	227	28473	128.26	1.81
8	Los Angeles	240	340	38716	113.87	1.90
9	London	488	730	52800	72.33	1.94
10	New Delhi	252	334	32281	96.56	1.85
11	New York	248	419	36172	86.33	1.72
12	Paris	335	494	44109	89.29	1.88
13	Richmond	697	1086	62608	57.65	1.78
14	Savannah	584	958	62050	64.77	1.85
15	Seoul	869	1307	68121	52.12	1.87
16	San Francisco	169	271	38187	140.91	1.90
17	Venice	1840	2407	75219	31.25	1.81
18	Vienna	467	692	49935	72.16	1.88
19	Washington	192	303	36342	119.94	1.93
20	Walnut Creek	169	197	25131	127.57	1.80

Table 3.1: Basic properties of the planar graphs obtained from the twenty city samples considered. N is the number of nodes, K is the number of edges, W and $\langle l \rangle$ are respectively the total length of edges and the average edge length (both expressed in meters), D_{box} is the box-counting fractal dimension.

the smallest values in cities as Ahmedabad, Cairo and Venice, and the largest value in San Francisco, Brasilia, Walnut Creek and Los Angeles. So street *density* and average street length are two quantities to look at if one wants to "distinguish" between self-organized and planned cities. In Ref. [73, 74] Crucitti et al. have studied the edges length distribution $P(l)$ for the two different classes of cities, showing that self-organized cities show single peak distributions, while mostly planned cities exhibit a multimodal distribution, due to their grid pattern.

Except for the case of Irvine 1 where the number of points is too low to calculate the box counting dimension, all the values of D_{box} for the

twenty cities are in the range between 1.72 and 1.99 which suggest us to conclude that the fractal dimension of the set of nodes $\mathcal{N}$ is close to 2 but not equal. Anyway, in our analysis we decided to use the box dimension, instead of the correlation one, since the number of points is quite small hence the calculations did not last too long. If we look at FIg. 3.1

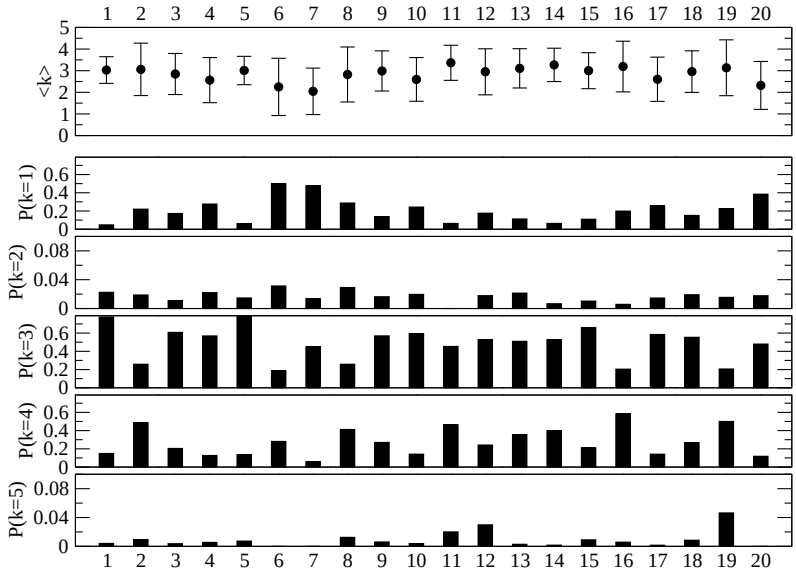


Figure 3.1: Average degree $\langle k \rangle$ and probability of having nodes with degree respectively equal to 1,2,3,4 and 5 for the twenty cities considered. The cites are labeled from 1 to 20 as reported in Tab. 3.1.

we could infer informations on the structure of the patterns. Focusing on the *average degree* $\langle k \rangle$ (upper panel) we see that it assumes values between 2 and 3. As one can expect, cities with less nodes exhibit larger fluctuations. Instead, if we look at the degree distribution $P(k)$ (black bars panels) we can try to guess some informations about the structure of road intersections. For example, the cities of Irvine (1 and 2) and Walnut Creek (columns 6,7 and 20) have a lot of nodes with degree 1 due to their lollipop structure [66, 75]. Instead, Ahmedabad and Cairo (columns 1 and 5) have about the 80% of the nodes with degree 3 (Seoul (15), Venice (17) and Vienna (18) are just below this percentage) denoting a self-organized structure typical of a long time unplanned evolution.

Finally, observing the $P(k = 4)$ is "easy" to identify the grid-like cities like Barcelona (2), Los Angeles (8), New York (11), San Francisco (16) and Washington (19).

It is not the aim of this work to discuss the meaning of such differences in terms of their possible impacts on crucial factors for urban life, such as pedestrian movements, way finding, land uses, or other cognitive or behavioral matters. However, it is worth noting that, for instance, 3-arms and 4-arms street junctions are expected to perform very differently in human orienteering within an urban complex system due to differences in the angle widths involved in each turn [65, 76].

In order to help the interpretation, and give better readability and emphasis to the results, we have used an *a priori* classification of our twenty cities. The classes are:

Medieval (arabic and european) ($\bigcirc$): Ahmedabad, Cairo, Bologna, London, Venice and Vienna;

Grid-iron ($\Diamond$): Barcelona, Los Angeles, New York, Richmond, Savannah and San Francisco;

Modernist ($\square$): Brasilia and Irvine 1;

Baroque ($*$): New Delhi and Washington:

Mixed ($+$): Paris and Seoul;

Lollipop ($\triangle$): Irvine 2 and Walnut Creek.

3.2 Local Properties

Let us now start to study the **local properties** of our cities. In particular we focus on two quantities: the **meshedness coefficient** and **number of cycles** of different species. The former tell us if our graphs resemble most to a GT or a MST, the latter, instead, will tell us what

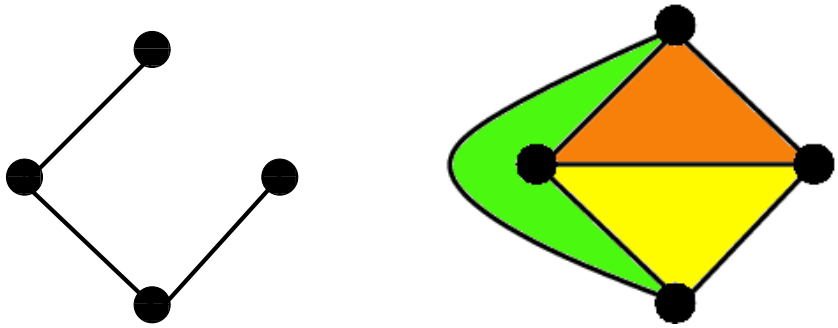
is the abbundance of a certain kind of motif. These are local properties since does not involte the whole structure of the network but only small glimpses of it. Starting from now, we will use the method based on normalization with respect to MST and GT to discuss the results found.

3.2.1 The Meshedness Coefficient

Many complex networks contain a large number of short cycles or specific motifs [3, 4, 6]. For instance, the so called *local clustering*, also known as *transitivity*, is a typical property of acquaintance networks, where two individuals with a common friend are likely to know each other [10]. The degree of *clustering* is usually quantified by the calculation of the clustering coefficient C, introduced in [1]. Such a quantity is not suited to characterize the local properties of a planar graph, since by a simple counting of the number of triangles present in the graph it is not possible to discriminate between different topologies. For instance, there are cases as diverse as trees, square-meshes and honey-comb meshes, all having the same clustering coefficient equal to zero. Buhl et al. have proposed a more general measure of the structure of cycles (not restricted to cycles of length 3) in planar graphs, the so called **meshedness coefficient** M [24]. The meshedness coefficient is defined as:

$$M = \frac{F}{F_{\max}}\,, \tag{3.1}$$

where F is the number of faces (excluding the external ones) associated with a planar graph with N nodes and K edges, and expressed by the Euler formula (1.9). $F_{\max}$ is the maximum possible number of faces that is obtained in the maximally connected planar graph i.e. in a graph with N nodes and $K_{\max} = 3N - 6$ edges, thus $F_{\max} = 2N - 5$. To better understand the meaning of this coefficient let us consider an example shown in Fig. 3.2: If we consider the graph of Fig. 3.2(a) (a tree) we have that $F = K - N + 1 = 3 - 4 + 1 = 0$ such that $M = \frac{0}{2*4-5} = \frac{0}{3} = 0$.



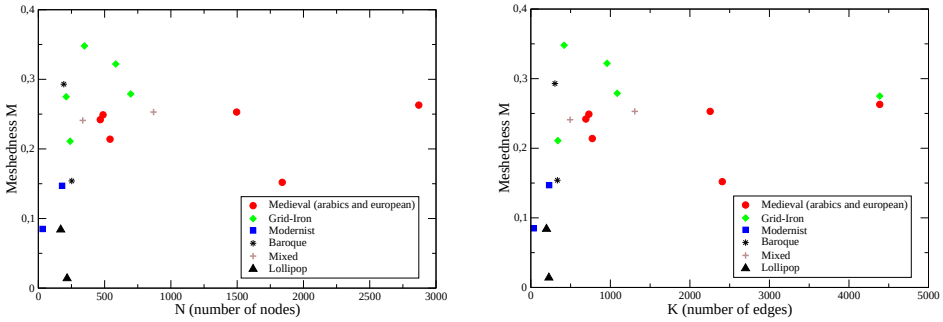
(a) (b)

Figure 3.2: Extreme cases for the meshedness coefficient: A tree (a) ($M = 0$), and a complete plane graph (b) ($M = 1$) where each face is filled with a different color.

Instead, if we look at the graph in Fig. 3.2(b) (a complete plane graph) we found that $F = K - N + 1 = 6 - 4 + 1 = 3$ such that $M = \frac{3}{2*4-5} = \frac{3}{3} = 1$. We are then able to say that:

$$M(G) = \begin{cases} 0 & \text{if } G \text{ is a tree,} \\ 1 & \text{if } G \text{ is a complete plane graph.} \end{cases}$$

In our case, since our GT does not contain $3N - 6$ edges the value of meshedness is not exactly equal to 1 but close to it. The values of M for the twenty cities are reported in Tab. 3.2. Values span from 0.014 for Irvine 2 to 0.348 for New York demonstrating that our cities (even the most dense) are far away from a complete planar graph. An additional thing to look at is how the meshedness coefficient varies with respect to both number of nodes N and of edges K. In Fig. 3.3 we report the behaviour of M versus N and K. As one could notice at first glance there are no big differences. Furthermore, excluding cities of medieval and grid-iron class, cities tends to clusterize showing similar meshedness values. Medieval cities instead, seems to have a "quasi-constant" value of M regardless of their number of nodes or edges. This confirms the idea that meshedness does not depend on the street density nor on the number of their intersections but only on HOW these are made.



(a) (b)

Figure 3.3: Meshedness coefficient M as a function of: number of nodes N (a), number of edges K (b).

3.2.2 Triangles VS Squares

A **motif** $\tilde{G}$ is a pattern of interconnections occurring either in a undirected or in a directed graph G at a number of times significantly higher than in randomized version of the graph. As a pattern of interconnections, a motif $\tilde{G}$ is usually meant as a connected (directed or not) n-node graph which is a *subgraph* of G. The concept of motifs was originally introduced by Alon and coworkers [9, 28, 29, 77, 78], who studied small n motifs in biological networks but that can be also found in other kind of networks. The research of significant motifs in a graph G is based on matching algorithms counting the total number of occurrences of each n-node subgraph $\tilde{G}$ in the original graph and in its randomized version considering it statistically significative if the frequence is much higher than in the random case. In our case we are interested in counting a particular kind of motifs known as **cycles**. A *cycle* on three or more vertices is a simple graph whose vertices can be arranged in a cyclic sequence in such a way that two vertices are adjacent if they are consecutive in the sequence, and are nonadjacent otherwise. Cycles can have different length k (meant as the number of hops one has to do in order to return to the starting point). In particular, we focus on cycles of length three (*triangles*), four (*squares*) and five (*pentagons*).

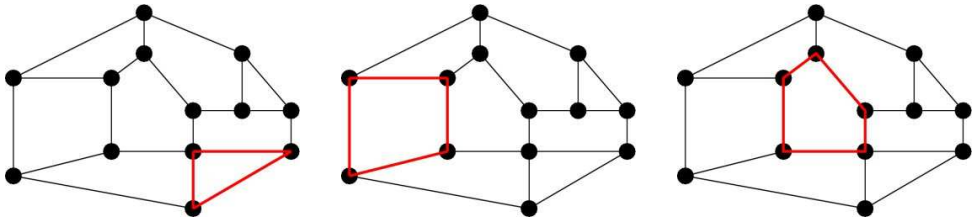


Figure 3.4: An example of cycles of length three (a), four (b) and five (c).

To count the number of cycles of a given length k, following the paper of Alon et al. [79], we use the properties of powers of the adjacency matrix $\mathcal{A}$. For example, the number of cycles of length three $n(C_3)$ is simply equal to:

$$n(C_3) = \frac{1}{6} \operatorname{Tr}\left(\mathcal{A}^3\right) . \tag{3.2}$$

In general, let $G = (\mathcal{N}, \mathcal{L})$ be a simple undirected graph and let $\mathcal{A}$ be its adjacency matrix. Assume for simplicity that $\mathcal{N} = \{1, \ldots, N\}$ and denote by $a_{ij}^{(k)} = \left(\mathcal{A}^k\right)_{ij}$ the elements of the k-th power of $\mathcal{A}$. The **trace** $\operatorname{Tr}\left(\mathcal{A}^k\right)$ of $\mathcal{A}^k$, which is the sum of the entries along the diagonal of $\mathcal{A}^k$, gives us the number of closed walks of length k in G. If we could also compute the number of *nonsimple* closed walks of length k in G we would easily obtain the number of **simple** closed paths of length k in G. This number is just $2k$ times the number of cycles of length k, $n(C_k)$, in G.

Before describing a way of counting the number of nonsimple closed walks of length k, where $k \leq 7$, in a graph G in $O(N^\omega)$ time, we need to introduce few concepts.

Let $G_1 = (\mathcal{N}_1, \mathcal{L}_1)$ and $G_2 = (\mathcal{N}_2, \mathcal{L}_2)$ be two simple graphs. A mapping $f : \mathcal{N}_1 \cup \mathcal{L}_1 \to \mathcal{N}_2 \cup \mathcal{L}_2$ is a **homomorphism** if for every $v \in \mathcal{N}_1$ we have $f(v) \in \mathcal{N}_2$ and for every $e = (u, v) \in \mathcal{L}_1$ we have $f(e) = (f(u), f(v)) \in \mathcal{L}_2$. If f is onto $\mathcal{N}_2 \cup \mathcal{L}_2$, we say that G_2 is a **homomorphic image** of G_1.

A graph $H = (\mathcal{N}_H, \mathcal{L}_H)$ is said to be k**-cyclic**, for $k \geq 3$, if it is

a homomorphic image of the cycle C_k. The number of different homomorphisms from C_k to H is denoted by $c_k(H)$. Clearly, H is k-cyclic if and only if $c_k(H) > 0$. It is not so difficult to check that, in general, $c_k(C_k) = 2k$ for every $k \geq 3$. The k-cyclic graphs for $3 \leq k \leq 6$ are given in Fig. 3.5. Let $n_G(H)$ denote the number of subgraphs of G isomorphic

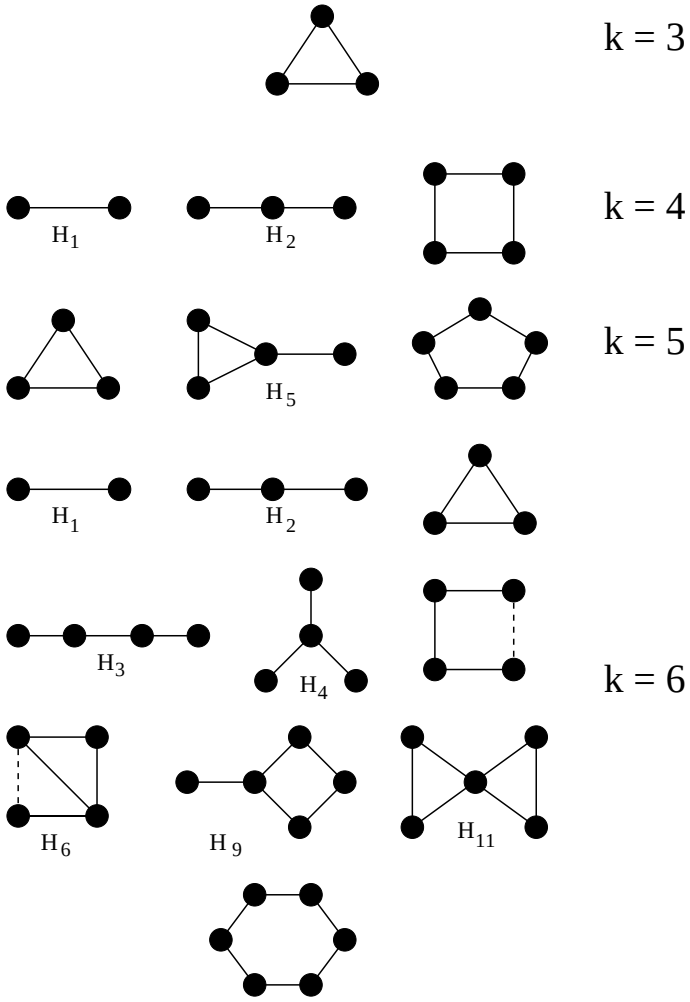


Figure 3.5: All the possible k-cycles for $3 \leq k \leq 6$. Graphs labeled as H_l represent the nonsimple ones where l is the number of edges.

to H. Clearly, the total number of closed walks of length k in G is:

$$\mathrm{Tr}\left(\mathcal{A}^k\right) = \sum_H c_k(H)\, n_G(H)\,.$$

If $c_k(H) > 0$, then H is connected and has at most k edges. Also, H cannot be a tree on $k+1$ vertices as each edge leading to a leaf must be the image of at least two edges in C_k. Hence, $|\mathcal{N}_H| \leq k$ and in fact, $|\mathcal{N}_H| < k$ unless $H = C_k$. We therefore obtain, for an undirected graph $G = (\mathcal{N}, \mathcal{L})$:

$$n_G(C_k) = \frac{1}{2k}\left[\mathrm{Tr}\left(\mathcal{A}^k\right) - \sum_{|\mathcal{N}_H|<k} c_k(H)\, n_G(H)\right] . \tag{3.3}$$

We obtain the following theorem:

Theorem 3.2.1. *The number of C_k's, for $3 \leq k \leq 7$ in an undirected (or directed) graph $G = (\mathcal{N}, \mathcal{L})$, can be found in $O(N^\omega)$ time (with $\omega = 2.376$ as in the Strassen's algorithm [68]).*

Here we do not give the proof of the theorem that can be found in [79]. Now, in order to normalize the results we need to divide the number of cycles in the city graph by the number of cycles in the corresponding GT. We have denoted by C_k the number of cycles of length k in a given city, and by C_k^{GT} the same number in the corresponding GT. The results are reported i

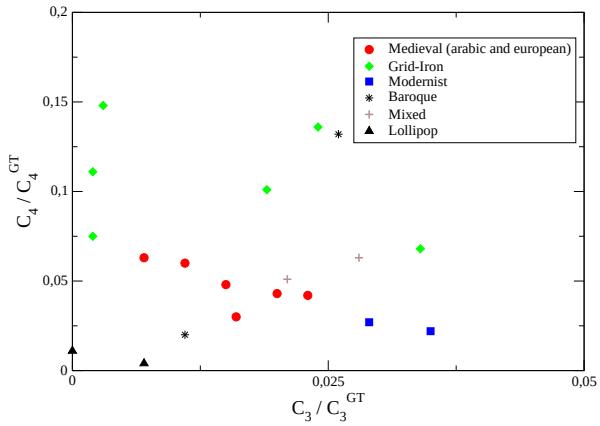


Figure 3.6: Normalized number of squares versus normalized number of triangles for all the considered cities. Values are taken from Tab. 3.2.

Considering the normalized values of triangles, squares and pentagons (we choose to not get further to limit the computation time during simulations) in Tab. 3.2 we have plotted, for each city, the normalized number of triangles versus the same quantity but for squares. The outcome is

	CITY	M	C_3/C_3^{GT}	C_4/C_4^{GT}	C_5/C_5^{GT}
1	Ahmedabad	0.262	0.023	0.042	0.020
2	Barcelona	0.275	0.019	0.101	0.019
3	Bologna	0.214	0.015	0.048	0.013
4	Brasilia	0.147	0.029	0.027	0.012
5	Cairo	0.253	0.020	0.043	0.019
6	Irvine 1	0.085	0.035	0.022	0.005
7	Irvine 2	0.014	0.007	0.004	0.001
8	Los Angeles	0.211	0.002	0.075	0.011
9	London	0.249	0.011	0.060	0.020
10	New Delhi	0.154	0.011	0.020	0.011
11	New York	0.348	0.024	0.136	0.028
12	Paris	0.241	0.028	0.063	0.016
13	Richmond	0.279	0.034	0.068	0.022
14	Savannah	0.322	0.002	0.111	0.026
15	Seoul	0.253	0.021	0.051	0.021
16	San Francisco	0.309	0.003	0.148	0.003
17	Venice	0.152	0.016	0.030	0.010
18	Vienna	0.242	0.007	0.063	0.018
19	Washington	0.293	0.026	0.132	0.022
20	Walnut Creek	0.084	0.000	0.011	0.003

Table 3.2: Local properties of the graphs of urban street patterns. We report the meshedness coefficient M, and the number C_k of cycles of length $k = 3, 4, 5$ normalized to the number of cycles in the GT, C_k^{GT}.

shown in Fig. 3.6. In most of the samples, we have found a small value of C_3/C_3^{GT} (as compared, for instance, to C_4/C_4^{GT}), denoting that triangles are not common in urban city patterns. This is another proof that the clustering coefficient C alone is not a good measure to characterize the local properties of such networks. Walnut Creek, Los Angeles and Savannah are the cities with the smallest value of C_3/C_3^{GT}, while Irvine

1, Richmond, Brasilia and Paris are the cities with the largest value of C_3/C_3^{GT}. In 17 samples out of 20 we have found $C_4/C_4^{GT} > C_3/C_3^{GT}$: Brasilia, Irvine 1 and Irvine 2 are the only cities having a prevalence of triangles with respect to squares. San Francisco, New York, Washington, Savannah and Barcelona are the cities with the largest value of C_4/C_4^{GT} (larger than 0.1). In particular, we found that grid-iron cities, in general, have a lot more squares than triangles (as one would expect). However, half of them have a number of triangles which is far superior even if the number of squares remains "constant". The modernists ones, instead, have a ratio between squares and triangles which is close to 1, while in mixed cities the ratio climbs up to about 2.

Finally, concerning C_5/C_5^{GT}, we have found three classes of cities. Samples such as Ahmedabad, Cairo, Seoul and Venice having $C_3/C_3^{GT} \cong C_5/C_5^{GT}$. Samples such as Brasilia, Irvine and Paris with $C_3/C_3^{GT} > C_5/C_5^{GT}$, and samples as Los Angeles, Savannah and Vienna with $C_3/C_3^{GT} < C_5/C_5^{GT}$.

3.3 Global Properties

After discussing the local properties of our city patterns, we deal now with the **global properties**, i.e. properties related to the whole system not with its particular (microscopic) structure. These quantities are the most suited to be used when one wants to compare graphs of different categories (for example a biological network with a man made one).

One of the possible mechanisms ruling the growth of an urban system is the achievement of efficient pedestrian and vehicular movements on a global scale. This has important consequences on a number of relevant factors affecting the economic, environmental and social performances of cities, ranging from accessibility to microcriminality and land uses [61]. The global efficiency of an urban pattern in exchanging goods, people and ideas should be considered a reference when the capacity of that city

to support its internal relational potential is questioned. It is especially important to develop a measure that allows the cases of different form and size, which poses a problem of normalization [60]. We focus our attention on two of such quantities: *efficiency* and *cost*.

3.3.1 Efficiency

The global structural properties of a graph can be evaluated by the analysis of the shortest paths between all pairs of nodes. In a relational (unweighted) network the shortest path length between two nodes i and j is the minumum number of edges to traverse to go from i to j. In a spatial (weighted) graph, instead we define the shortest path length d_{ij} as the smallest sum of the edge lengths throughout all the possible paths in the graph from i to j [26, 27, 36]. In this way both the topology and the geography of the system are taken into account.

A problem arising from the definition of characteristic path length in Eq. (1.7) is that L diverges if there are disconnected components in the graph. We already spoke about the possibility to limit the summation to the nodes belonging to the giant component. Another possible way to solve this problem is to consider the harmonic mean of the geodesic lengths, and to define the so-called **efficiency** of a graph [26, 27]. As a measure of the efficiency in the communication between the nodes of a spatial graph, we use the following definition of **efficiency** E of a graph G, a measure defined in [26] as:

$$E = \frac{1}{N(N-1)} \sum_{i,j \in \mathcal{N}, i \neq j} \frac{d_{ij}^{\text{Eucl}}}{d_{ij}} \,. \tag{3.4}$$

Where $d_{[ij}$ is the shortest path length between i and j, and d_{ij}^{Eucl} is the distance between nodes i and j along a straight line (Eq. (2.2)). Such quantity is an indicator of the traffic capacity of a network, and avoids the divergence in Eq. (1.7), since any pairs of nodes belonging to disconnected component of the graph yields a contribution equal to zero to

the summation in Eq. (3.4) (if two nodes have no paths connecting them then we set $d_{ij} = \infty$). Here we have used a normalization for geographic networks proposed by Vragovic et al. in [80]. Such a normalization captures to which extent the connecting route between i and j deviates from a virtual straight line.

	CITY	E	E^{MST}	E^{GT}
1	Ahmedabad	0.818	0.351	0.944
2	Barcelona	0.814	0.452	0.930
3	Bologna	0.799	0.473	0.936
4	Brasilia	0.695	0.503	0.931
5	Cairo	0.809	0.385	0.943
6	Irvine 1	0.755	0.604	0.943
7	Irvine 2	0.374	0.533	0.932
8	Los Angeles	0.782	0.460	0.930
9	London	0.803	0.475	0.936
10	New Delhi	0.766	0.490	0.930
11	New York	0.835	0.433	0.931
12	Paris	0.838	0.473	0.938
13	Richmond	0.800	0.502	0.939
14	Savannah	0.793	0.341	0.922
15	Seoul	0.814	0.444	0.941
16	San Francisco	0.792	0.448	0.893
17	Venice	0.673	0.386	0.943
18	Vienna	0.811	0.423	0.937
19	Washington	0.837	0.452	0.930
20	Walnut Creek	0.688	0.481	0.938

Table 3.3: The efficiency E of each city is compared to the minimum and maximum values of the efficiency obtained respectively for the MST and the GT. The cities are labeled from 1 to 20 as in Table 3.1.

In Tab. 3.3 we report the values of efficiency obtained for each city and for the respective MST and GT. The values of E^{MST} and E^{GT} serve to normalize the results, being respectively the minimum and the maximum value of efficiency that can be obtained in a planar graph having the same number of nodes as in the original graph of the city. One could argue

that, given a set of nodes, the MST is not *unique* (this is true especially in grid-like cities). Notice that, if we change by a little amount the location of some of the nodes of a graph, we get a MST different from the previous one. Nevertheless, we have checked that the two MSTs have a very similar value of efficiency. Notice that Irvine 2 is the only case in which $E < E^{\mathrm{MST}}$. This is simply due to the fact that Irvine 2 is the only city whose corresponding graph is not connected. Consequently, the MST has a smaller number of edges but a higher value of efficiency because it is, by definition, a connected graph. The main result is that the cities considered, despite their inherent differences, achieve a relatively high value of efficiency, which is in most of the cases about 80% of the maximum value of the efficiency in a planar graph, E^{GT}. Following Buhl et al. [24] we define the **relative efficiency** E_{rel} as:

$$E_{\mathrm{rel}} = \frac{E - E^{\mathrm{MST}}}{E^{\mathrm{GT}} - E^{\mathrm{MST}}} \,. \tag{3.5}$$

3.3.2 Cost

Of course, the counterpart of an increase in efficiency is an increase in the **cost** of construction, i.e. an increase in the number and in the length of the edges. The simplest way to quantify the cost is through Eq. (2.1). Given a set of N nodes, the shortest (minimal cost) planar graph that connects all nodes correspond to the MST, while a good approximation for the maximum cost planar graph is given by the GT. We thus define a normalized cost measure W_{rel}. as:

$$W_{\mathrm{rel}} = \frac{W - W^{\mathrm{MST}}}{W^{\mathrm{GT}} - W^{\mathrm{MST}}} \,. \tag{3.6}$$

Looking at the results of Tab. 3.4 we find, as expected $W^{\mathrm{MST}} < W < W^{\mathrm{GT}}$. Ahmedabad is the most "expensive" city, while Irvine 1 is the "cheapest". If we look at the ratios of $\frac{W}{W^{\mathrm{MST}}}$, $\frac{W}{W^{\mathrm{GT}}}$ and $\frac{W^{\mathrm{GT}}}{W^{\mathrm{MST}}}$ we obtain

	CITY	W	W^{MST}	W^{GT}
1	Ahmedabad	121037	54295	281033
2	Barcelona	36179	18600	91843
3	Bologna	51219	25336	140475
4	Brasilia	30910	14748	80328
5	Cairo	84395	37438	200981
6	Irvine 1	11234	7339	34341
7	Irvine 2	28473	15847	92051
8	Los Angeles	38716	17816	96870
9	London	52800	24835	130737
10	New Delhi	32281	15864	94415
11	New York	36172	14885	76441
12	Paris	44109	19894	110428
13	Richmond	62608	24915	147817
14	Savannah	62050	25529	128584
15	Seoul	68121	32016	166520
16	San Francisco	38187	13914	77257
17	Venice	75219	38222	204333
18	Vienna	49935	24404	128584
19	Washington	36342	17639	84042
20	Walnut Creek	25131	14698	76744

Table 3.4: Cost W (expressed in meters) of each city is compared with the corresponding values of the same quantity in the MST (W^{MST}) and the GT (W^{MST}). The cities are labeled from 1 to 20 as in Table 3.1.

as average values of them 2.12, 0.39 and 5.37. This means that, on average, the cost of a city is about 2 times that of the MST and less than half (approx 2/5) of its GT. Instead, if we look at the ratio of the cost of GT and that of MST we can say that a GT approximateli costs more than five times a MST. In particular, in the case of New Delhi, New York and Richmond the ratio skims the value of 6.

3.3.3 Results

After discussing all the properties of a planar graph based of an urban street pattern we want to define a methodology to compare different cities

from a global point of view. If one asks a driver or a pedestrian to tell which quantity differentiates a city from another probably he will talk about something related with the capability to easly reach any point in the city from any other. On the other side, if one asks an urban planner which quantity distinguishes a city from another he will, instead, probably focus more on the smart use of resources. The former opinion can be more likely referred to the concept of *efficiency* while the latter may be related more with the *cost*. An interesting approach, used also in many other context [24, 81, 82], is to study the relation between relative efficiency and cost expressed by Eqs. (3.5), (3.6).

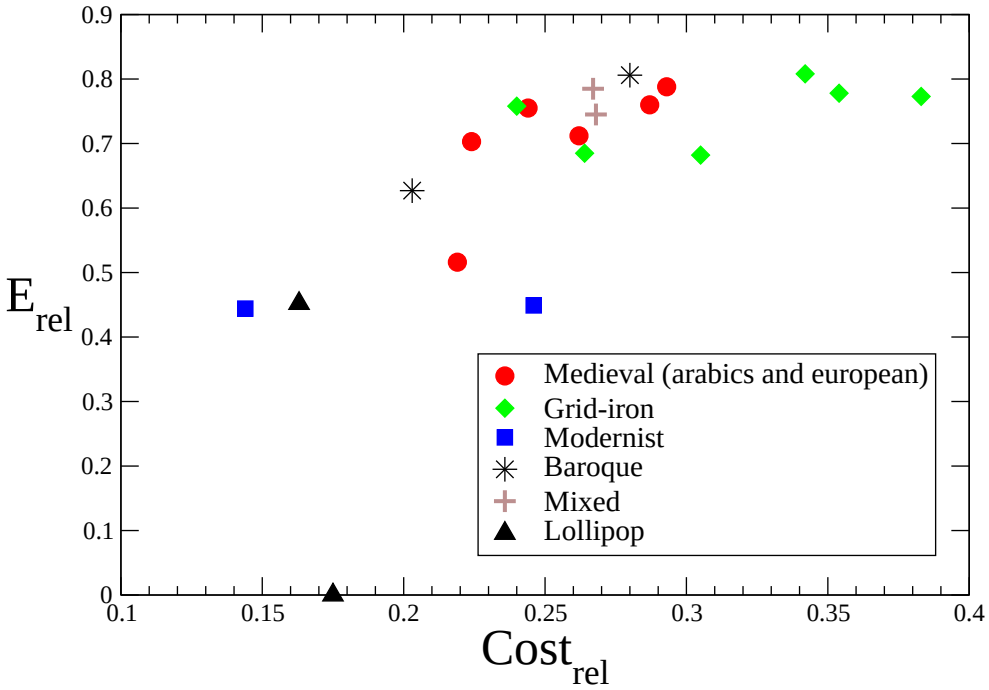


Figure 3.7: A plot of the relative efficiency E_{rel} as a function of the relative cost W_{rel} of a city.The plot indicate a correlation between structural properties and *a priori* known classes of cities. The point of coordinates (0,0) would correspond to the cost/efficiency of the MST while the point (1,1) would correspond to the GT network. The city of Irvine 2, having coordinates (0.175,-0.398), i.e., e negative value of relative efficiency, has been plotted instead as having coordinates (0.175,0).

By definition the MST has both relative cost and efficiency $W_{\text{rel}} = E_{\text{rel}} = 0$, while GT has $W_{\text{rel}} = E_{\text{rel}} = 1$. An interesting charachterization of different city patterns can be obtained by a plot of E_{rel} as a function of W_{rel} which is reported in Fig. 3.7. Using the classification made in Sec. 3.1, the plot E_{rel} vs W_{rel} has a certain capacity to characterize the different classes of cities. The plot indicates an overall increasing behaviour of E_{rel} as a function of W_{rel}, with a saturation at $E_{\text{rel}} \sim 0.8$ for values of $W_{\text{rel}} > 0.3$. Grid-iron patterns exhibit a high value of relative efficiency, about 70–80% of the efficiency of the GT, with a relative cost which oscillate between 0.24 and 0.4. The three grid-iron cities (New York, Savannah and San Francisco) with the largest value of efficiency, $E_{\text{rel}} \sim 0.8$, have respectively a relative cost equal to 0.342. 0.354 and 0.383. Medieval patterns have in general a lower cost and efficiency than grid-iron patterns although, in some cases as Ahmedabad and Cairo (the two medieval cities with the largest efficiency), they can also reach a value of $E_{\text{rel}} \sim 0.8$ with a smaller relative cost equal to 0.29. Modernist and lollipop layouts are those with the smallest value of W_{rel} but of relative efficiency too.

CHAPTER 4

CENTRALITY MEASURES IN NETWORKS OF URBAN STREET PATTERNS

The idea of **centrality** was first applied to human communication by Bavelas [83] who was interested in the characterization of the communication in small groups of people and assumed a relation between *structural centrality* and influence in group processes. Since then, various measures of centrality have been proposed over the years to quantify the importance of an individual in a social network [84]. More recently, the issue of structural centrality has attracted the attention of physicists [3, 4, 6], who have extended its applications to the realm of biological and technological networks (see for instance [5, 85, 86]). The standard centrality measures can be divided into two classes:

- measures based on the idea that the centrality of a node in a network is related to *how it is near* to other nodes;

- measures based on the idea that central nodes *stand between others*, playing the role of intermediary.

Measures such as **degree** [87] and **closeness** [88] are examples of measures of the first kind, while *shortest-path* [89] or *flow* [90] **betweenness**

are measures of the second kind. Latora et al. proposed a new class of centrality measures, the so-called *delta centrality* (or Δ centralities), which is a combination of the two main classes of centrality mentioned above [91].

The concept of centrality can be exported also to networks of urban street patterns. In this chapter we will discuss some applications of centrality on city networks made [30, 32, 73, 74, 92]. At the end of the chapter we discuss some possible applications related with the time evolution of both structure and centrality in a city [93, 94].

4.1 Centrality in cities and the Multiple Centrality Assessment

Imagine to be a tourist visiting a foreign city. You are in the downtown on a sidewalk of a street crossing, stop a person passing by and ask him/her information about the most important places and route of the city. After few moments, he/she will probably tell you a list of "important" places and routes. A natural question to ask is: "Can we *detect* those points without asking people in the city"? A method named **Multiple Centrality Assessment** (MCA) developed by Porta, Crucitti et al. [30, 73, 74] as a methodology for investigation of spatial systems as complex networks can be of much help.

In this methodology, the evaluation of the importance of a node is based on the study of a set of different node centrality measures, namely **degree** C^D, **closeness** C^C, **betweenness** C^B, **straightness** C^S and **information** C^I.

4.1.1 Classic measures of centrality

Centrality measures will be illustrated and framed in their natural historical context, that of social networks. A *social network* is here rep-

resented as a undirected, non-weighted graph G, consisting of a set of N nodes and a set of K edges connecting pairs of nodes. The nodes of the graph are the *individuals*, the actors of a social group, and the lines represent the *social relations*. We will describe all the relations in terms of the adjacency matrix $\mathcal{A}$.

The **degree centrality** is based on the idea that important nodes are those with the largest number of ties to other nodes in the graph. The degree centrality of a node i is defined as [87]:

$$C_i^D = \frac{k_i}{N-1} = \frac{1}{N-1} \sum_{j \in G} a_{ij} \,, \tag{4.1}$$

where k_i is the degree of node i defined in Eq. (1.1). An example is shown in Fig. 4.1(a).

The **closeness centrality** of a node i is based on the concept of minimum distance or geodesic d_{ij}, i.e. the minimum number of edges traversed to get from i to j [6] and is defined as [84, 88]:

$$C_i^C = \frac{1}{L_i} = \frac{N-1}{\sum\limits_{j \in G} d_{ij}} \,, \tag{4.2}$$

where L_i is the average distance from i to all the other nodes. An example is shown in Fig. 4.1(b).

The communication of two non-adjacent nodes, say j and k, depends on the nodes belonging to the paths connecting j and k. Consequently, assuming that the communication travels just along the geodesic, a measure of the relevance of a given node can be obtained by counting the number of geodesics going through it, and defining the so called **node betweenness**. More precisely, the betweenness of a node i is defined as [10, 89, 95]:

$$C_i^B = \frac{1}{(N-1)(N-2)} \sum_{\substack{j,k \in \mathcal{N} \\ i \neq k \,, j \neq k}} \frac{n_{jk}(i)}{n_{jk}} \,, \tag{4.3}$$

where n_{jk} is the number of shortest paths connecting i and k, while $n_{jk}(i)$ is the number of shortest paths connecting i and k and passing through

i. An example is shown in Fig. 4.1(c). The concept of betweenness can be extended also to edges. The **edge betweenness** is defined as the number of shortest paths between pairs of nodes which run through that edge [38].

Straightness centrality, C^S, originates from the idea that the efficiency in the communication between two nodes i and j is equal to the inverse of the shortest path length d_{ij} [26]. In the case of a spatial network embedded into a Euclidean space, the straightness centrality of node i is defined as:

$$C_i^S = \frac{1}{(N-1)} \sum_{j \in \mathcal{N}\, j \neq i} \frac{d_{ij}^{\text{Eucl.}}}{d_{ij}}\,, \tag{4.4}$$

where d^{Eucl} is the Euclidean distance between nodes i and j along a straight line, and we have adopted a normalization proposed for geographic networks [80]. This measure captures to which extent the connecting route between nodes i and j deviates from the virtual straight route. An example is shown in Fig. 4.1(d). Obviously, in a non-Euclidean metric space d^{Eucl} will be substituted with the distance defined in that space.

4.1.2 Delta centralities

Delta centrality measures are based on the following idea: the importance of a node (group of nodes) is related to the ability of the network to respond to the deactivation of the node (group of nodes) from the network. If G is the graph representing the network, the delta centrality (or Δ centrality), of node i, C_i^Δ, is defined as:

$$C_i^\Delta = \frac{(\Delta P)_i}{P} = \frac{P[G] - P[G']}{P[G]}\,, \tag{4.5}$$

where P is a generic quantity measuring the cohesiveness of the graph, and $(\Delta P)_i$ is the variation of P under the deactivation (isolation) of node i, i.e. the removal from the graph of the edges incident with node i. By

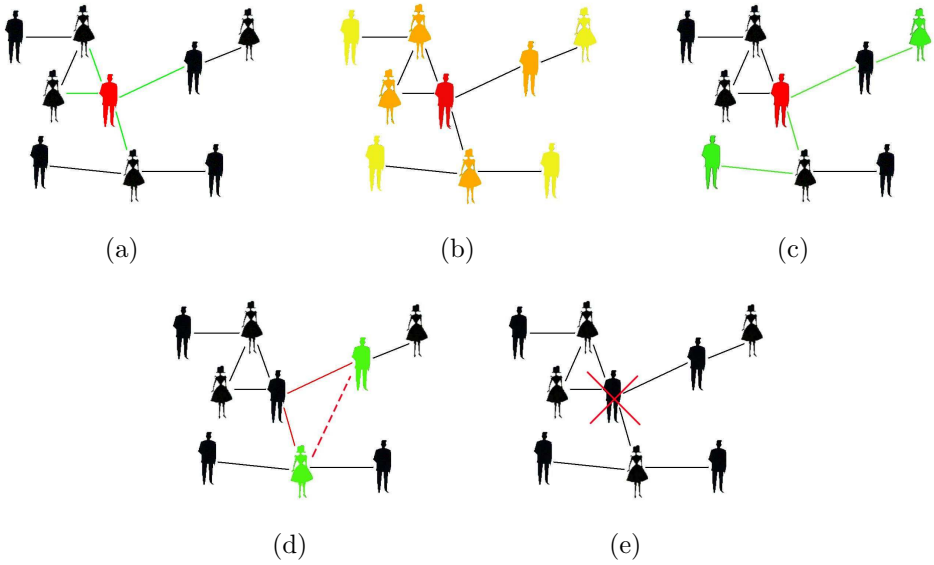


Figure 4.1: Examples of centrality measures in a social network: degree centrality C^D (a), closeness centrality C^C (b), betweenness centrality C^B (c), straightness centrality C^S (d) and information centrality C^I (e).

G' we indicate the graph obtained by removing from G the edges incident with node i. The only general restrictions on the choice of the quantity P in the above choice is that $(\Delta P)_i \geq 0$ for any node i of the graph. Of course, the meaning and effectiveness of the centrality measure C^Δ will depend on the choice of P. The simplest possibility is to take $P[G] = K$, where K is the number of edges in the graph G. In this case, $(\Delta P)_i$ will be proportional to the degree and the degree centrality, C_i^D, of i. A more interesting example is to take as P the efficiency E of the graph. This quantity is based on the assumption that the information/communication in a network travels along the shortest routes, and that the efficiency in the communication between two nodes i and j is equal to $1/d_{ij}$. The efficiency of a graph G, $E[G]$, has been defined in Eq. (3.4) and measures the mean flow-rate of information over G [26, 27]. In such case, the delta centrality of a node i, that we name **information centrality** C_i^I, is defined as the relative drop in the network efficiency caused by the

deactivation of i from the graph G:

$$C_i^I = \frac{(\Delta E)}{E} = \frac{E[G] - E[G']}{E[G]} \,. \tag{4.6}$$

An example is shown in Fig. 4.1(e). The removal of some of the edges affects the communication between various nodes of the graph increasing the length of the shortest paths, consequently $E[G'] \leq E[G]$ ensuring a positive difference. It is immediately seen that C^I is somehow correlated to the three standard measures C^D, C^C and C^B. In fact, the information centrality of node i depends on k_i, since the efficiency $E[G']$ is smaller if the number of edges removed from the original graph is larger. C_i^I is correlated to C_i^C since the efficiency of a graph is connected to $(\sum_i L_i)^{-1}$. Finally C_i^I, similarly to C_i^B, depends on the number of geodesics passing by i, but it also depends on the lengths of the new geodesics, the alternative paths that are used as communication channels, once the node i is deactivated. No information about the new shortest paths is contained in C_i^B, and in the other two standard measures.

All the centrality measures defined above can also be extended to quantify the importance of a group of nodes. Nevertheless such an extension (and the relative normalization) is not unique and a series of conventions must be adopted [96]. An example which illustrates the meaning of each centrality measure introduced until now, in the case of a social network, is shown in Fig. 4.1. Now, we are ready to discuss about the Multiple Centrality Assessment and its application to the city graphs.

4.1.3 Mutiple Centrality Assessment (MCA)

The MCA relies on the idea that, in order to identify the key role playing subject in a network, many different *centrality indexes* and *measures* must be taken into account. These quantities allow to estabilish some criteria for comparison analysis of network, and thus making their

study easier for the researchers. After centrality measures are calculated

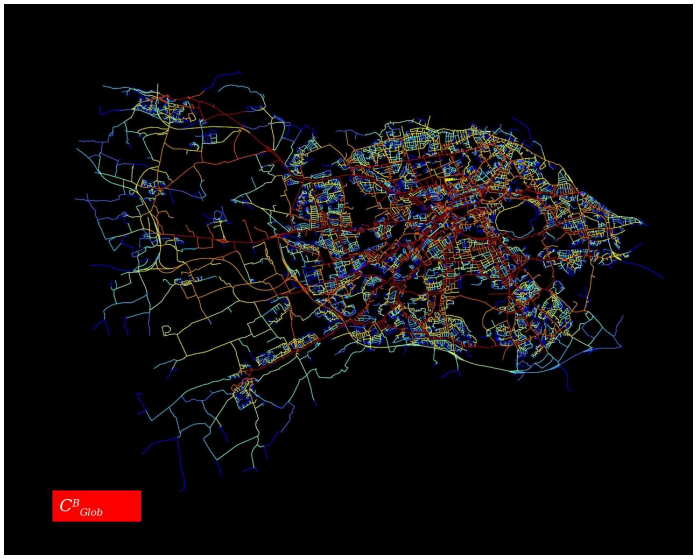


Figure 4.2: An example of Multiple Centrality Assessment. Here we report the values of betweenneess for the city of Edinburg (UK). The color of a street represent the value of betweenneess calculated on its endpoints. Blue represent low values of betweenneess while red correspond to higher value of it.

for each node of the graph, color/coded values are reported on them, giving rise to figures like Figs. 4.2 and 4.3. The final layout can either mapped on nodes as well as edge. In this latter case, the centrality of an edge is calculated as the average of its couple of endnodes. This simple procedure highlights a deep character of spatial networks when represented in such way: one edge exchanges with the system only at nodes, so its relational properties as a component of the system entirely depends on its endnodes importance.

The spatial distributions of node centralities can be graphically illustrated by means of GIS supported color-coded maps, in which one of eight different colors is plotted on each node of the graph. In Fig. 4.3 is shown the case of Cairo. The colors represent eight classes of nodes with different values of the centrality index C. The classes, defined in terms of multiples of the standard deviations σ from the average, are:

$[-\infty, -3\sigma]$, $[-3\sigma, -2\sigma]$, $[-2\sigma, -\sigma]$, $[-\sigma, 0]$, $[0, \sigma]$, $[\sigma, 2\sigma]$, $[2\sigma, 3\sigma]$, $[3\sigma, \infty]$, and the corresponding colors are reported in the figure legend. Looking

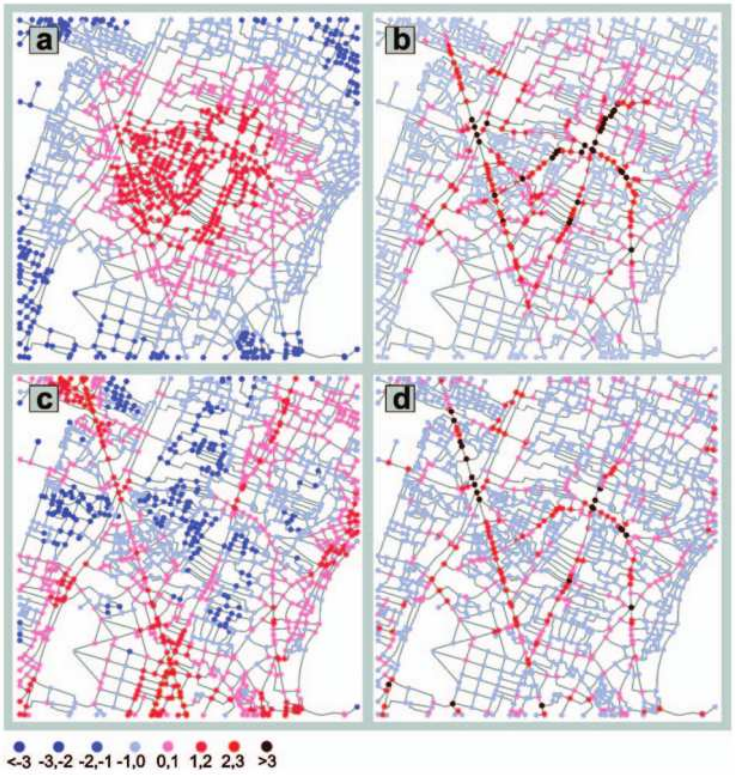


Figure 4.3: Thematic color map representing the spatial distributions of centrality in Cairo. The four indices of node centrality, (a) closeness C^C, (b) betweenness C^B, (c) straightness C^S and (d) information C^I, used in the MCA, are visually compared. Different colors represent classes of nodes with different values of the centrality index. The classes are defined in terms of multiples of standard deviations from the average, as reported in the color legend.

at Fig. 4.3 panel a, C^C exhibits a strong trend to group higher scores at the center of the image. This is due to the artificial boundaries imposed by the 1-square-mile maps representation and to the same nature of the closeness centrality. Edge effects are also present, although less relevant, in all tht other centrality measures except betweenness (see for instance the contour nodes in Fig. 4.3, panel a,c and d. The spatial distribution

of C^B nicely captures the continuity of prominent urban routes across a number of intersections, changes in direction and focal urban spots. This is visible both in Cairo, Fig. 4.3 panel b, and in Edinburg Fig. 4.2. In some cases, C^B clearly identifies the primary structure of movement channels as different to that of secondary, local routes. This is the case of Ahmedabad, Seoul, Richmond and Venice.

The spatial distribution of C^S depicts both linear routes and focal areas in the urban system Fig. 4.3 panel c, C^S takes high values along the main axes, even higher at their intersections. Finally C^I, although based on a different concept of centrality, exhibits a spatial distribution that is in many cases similar to that of C^B. This is especially evident in Cairo (Fig. 4.3 panel d), as well as in Ahmedabad and Venice.

In conclusion, we have shown how the Multiple Centrality Assessment can represent a powerful tool for an urban planner. In this way, in fact, one could say if the building of a new road gives benefits to the neighbourhood or if the removal of an esistent street affects the connectivity of it. MCA has been used also to characterize the structure of many cities [97, 98] which does not belong to our dataset like Edinburg (Fig. 4.2), Worcester to cite a few.

4.2 Centrality and Minimum Spanning Trees: the backbone of a city

In economic geography and in regional planning centrality has been dominating the scene especially since the Sixties and Seventies. In the field of urban design, a long-term effort has been spent in order to understand what urban streets and routes would constitute the "*skeleton*" of a city. By using this term, we mean the chains of urban spaces that are most important for the connectedness, liveability and safety at the local scale [99, 100], and its legibility in terms of human wayfinding [101].

Scellato et al. provided a tool for the analysis of the backbone of a complex urban system represented as a spatial (planar) graph [30]. Such a tool is based on the concept of spanning trees, and on the efficiency of centrality measures in capturing the essential edges of a graph. To construct spanning trees based on edge centrality, first one has to localize high centrality edges, namely the streets that are structurally made to be traversed (*betweenness centrality*) or the streets whose deactivation affects the global properties of the system (*information centrality*). Of course other definitions of edge centrality (as for instance closeness or straightness) can be used as well.

The spanning trees based on edge centrality are called **Maximum Centrality Spanning Trees** (MCSTs), i.e. maximum weight spanning trees where the edge weight is defined as the centrality of the edge. The reliability of this tool has been tested on the cities of Bologna and San Francisco as examples of self-organized and planned fabrics.

A single graph can have many different spanning trees. To construct MCSTs we assign a weight w_α to each edge α, which is usually a number representing how favorable (for instance how central) the edge is, and assign a weight to a spanning tree by computing the sum of the weights of the edges in that spanning tree. A **maximum weight spanning tree** is then a spanning tree with weight larger than or equal to the weight of every other spanning tree of the graph. It appears evident that it is possible to define appropriate edge weights with the aim of finding particular structures capable of connecting every single node of the graph while minimizing the corresponding total weight. In particular, for each city we have computed two different MCSTs, respectively based on betweenness and information. The two cases are obtained by respectively fixing $w_\alpha = C_\alpha^B$ and $w_\alpha = C_\alpha^I$, with $\alpha = 1, \ldots, K$. Since the two centrality measures focus on different properties of the network, using both of them allows us to enforce our analysis.

In Fig. 4.4 we compare graphically the two MCSTs with the **Minimum Length Spanning Trees** (MLST). We named it MLST instead of MST to emphasize the use of length as weight function. We have studied the "similarities" between MCST and MLST and summarized them in Tab. 4.1.

	Bologna		**San Francisco**	
	Bet	Info	Bet	Info
% of common links with MLST	82	75	70	76
% of total centrality in MCST	86	84	82	95

Table 4.1: Comparison between MCST and MLST calculated for the cities of Bologna and San Francisco. In the first row we report the % of common links between MLST and MCST calculated using betweennes (Bet) or information (Info) centrality. In the second row we report the % of total centrality hold by MCST in the case of betweenness (Bet) and information (Info) centrality.

The graphical visualization of the maximum centrality trees of Fig. 4.4 is of interest for urban planners since the trees express the uninterrupted chain of urban spaces that serves the whole system while maximizing centrality over all edges involved. This method identifies **the backbone of a city** as the sub-network of spaces that are most likely to offer the highest potential for the life of the urban community in terms of popularity, safety and services locations, all factors geographically related with central places. This is evident in Fig. 4.4, where the comparison between the trees in the two cities clearly indicates that the spatial subsystem that keeps together a city in terms of the shortest trip length is not the same spatial sub-system that does it in terms of the highest centrality.

Another thing is that while the shortest length backbone performs effectively when applied to planned urban fabrics like San Francisco, in self-organized evolutionary cases like that of Bologna it does not find continuous routes nor clearly distinguishes a hierarchy of sub-systems in

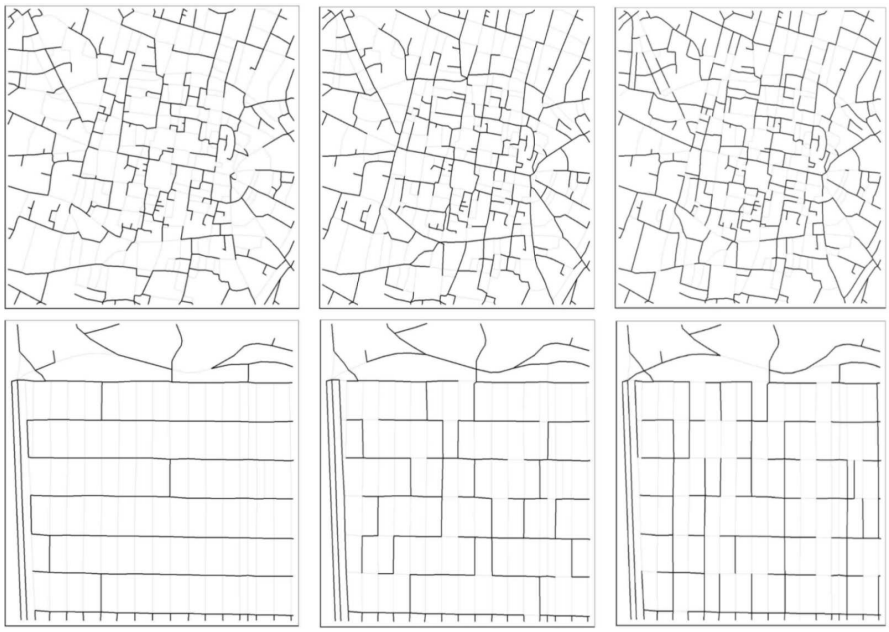


Figure 4.4: Spanning trees of Bologna (above) and San Francisco (below). From left to right, MLSTs, betweenness-based and information-based MCSTs.

the network, while the highest information and especially the highest betweenness backbones do. This is confirmed by results in Tab. 4.1 In a way, we would say that organic patterns are more oriented to put things and people together in public space than to shorten the trips from any origin to any destination in the system, this latter character being more typical of planned cities.

4.3 Correlation Between Centrality and Commercial Activities

Suppose to ask your grocer (butcher, fiorist, or whatever kind of salesman) why he put his shop in that place instead of another. He will probably answer that before making a choice he has spent some time around the streets looking for where was the highest flux of people. "The first thing, he could say, is where do the people walk. You have to be central,

people are where centrality is". Looking at the location of the commercial activities in a city a question pops out quasi immediately: *is there an empirical evidence of correlation between street centrality and economic activities*?

At a glance, one istinctively answers yes to the above question, but can we give a response based on more solids grounds? In order to study the correlation between centrality and density of commercial activities we have to convert discrete information (location of shops) into a continuous one. This can be done dividing the city surface into square cells and computing, for each one of them, the **Kernel Density Estimator** (KDE) which operate a so-called *spatial smoothing* on the datas. A **kernel function** is a non-negative real-valued integrable function K satisfying the following two requirements [102]:

$$\int_{-\infty}^{+\infty} K(u)\,du = 1\,,$$
$$K(u) = K(-u) \qquad \forall u\,.$$

The KDE uses the density within a range (window) of each observation to represent the value at the centre of the window itself. Within the window, the KDE weighs nearby objects more than distant ones, on the basis of a kernel function. This is necessary in order to respect Tobler's first law of geography – that is, "*everything is related to everything else, but near things are more related than distant things*" [103]. This property of distance decay for spatial interaction gave birth to a family of gravity models following the same notion with strong theoretical foundations which have many successful applications in urban and regional studies.

By doing spatial smoothing, the KDE generates a density of the events (discrete points) as a continuous field, and therefore converts the two datasets and permits the analysis of relationships between them. An example of this method is shown in Fig. 4.5 which shows the case of Bologna (Italy) which has been analyzed by Porta et al. in [32].

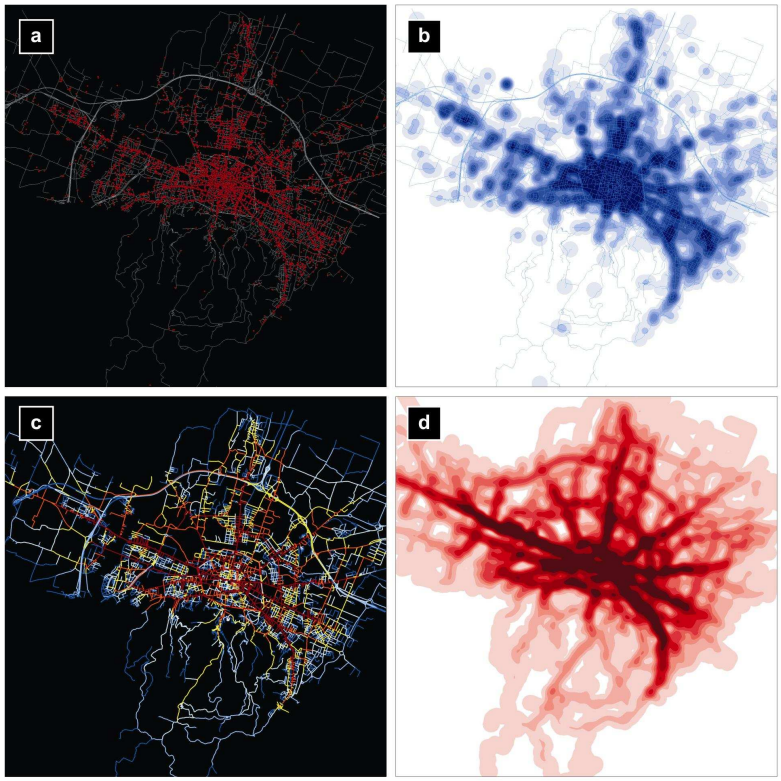


Figure 4.5: Density of activity and street centrality: (a) location of commercial and service activities (red dots); (b) kernel density estimation (KDE) of commercial and service activities; (c) street betweenness C^B (blue for lower values and red for higher); (d) KDE of C^B.

Looking at the pictures in Fig. 4.5 one could deduce, at first glance, that a correlation between retail and centrality exist. However, such relation is not completely linear and depends strongly on the kind of commercial activity considered. In fact, as found by Jensen et al. [104, 105], commercial activities tends to aggregate according to some economic factors in some cases (luxury goods), while in other cases they tend to place themselves away from activities of the same category. This kind of studies are very important in the so-called *geomarketing*, a discipline which studies methods to find the best location for opening a shop according with both economical and geographical information on the system. Now-

days, such kind of services are offered more and more often to people who wants to open a new shop by, for example. real estate agency.

4.4 A future development: Time Evolving Graphs

Up to now we have looked at graphs which are *static*, i.e. not changing in time. However, a city is far away from a static object. New roads are built and old ones are modified continuously. Obviously, the typical times to produce a tangible effect are longer than a day or a week. Nevertheless, if one could monitor the system for long enough we will be able to see changes in the structure of the city. The inclusion of the time dimension, is not a prerogative of graph of urban street patterns. Other sistems change their structure during time. For example *social networks* like Facebook, Twitter and Linked-in show the same behaviour.

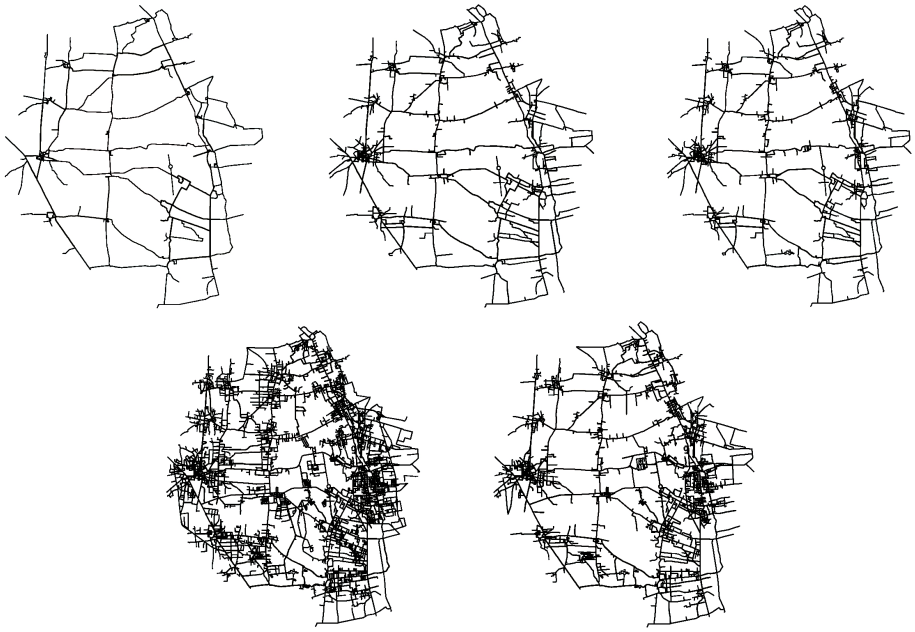


Figure 4.6: Time evolution of the urban pattern of the district named Groane Park near Milan (Italy). From top left in clockwise sense, the city structure as it was in: 1833, 1914, 1933, 1955 and 1980.

The study of dynamical evolution of the topology is a hot topic in the field of complex networks. In the last year, together with Sergio Porta, Vito Latora and Emanuele Strano, we have started to study the time evolution of networks based on urban street patterns. The pictures in Fig. 4.6 represent the street pattern of the urban area of Groane Park, a district in the region of Lombardia near Milan (Italy). As one can see, the network does not only grows towards its outer part but also towards its inner ones. Such growth is not uniform neither isotropic. It would be very interesting to see if the growth is correlated with some factors like population or retail densities. This can be done using a biased percolative growth model like the ones used to interpret leaf growth [106].

Moreover, we are planning to study some structural features which are relevant from the urbanistic point of view like the evolution of T-shaped three edges street crossings into four edge ones, and the role of such points as growth nucleus for other streets. A final aspect to look at is the *evolution of centrality in time*. We have found that there are cases in which a new street inherits the centrality of its parents. These studies are still in progress and could represent the natural evolution of the characterization proposed in this thesis.

CONCLUSIONS

In this thesis we have presented a characterization of the main topological features of twenty different samples of world cities. These samples were chosen to take into account different classes of urban fabrics from self-organized cities to single plan cities.

After introducing the fundamental concepts of graph theory and the main characteristics of plane graphs, we have shown the methods used to map a city into a graph. In order to compare the characteristics of the twenty cities a normalization of results is needed. Since the usual normalization method cannot be applied because of planarity breaking a new method has been used. This new methodology introduced by Buhl et al. [24] requires the introduction of two particular kinds of plane graph: the minimum spanning tree (MST) and the greedy triangulation (GT). All the results were normalized with respect to the same quantities calculated in such graphs.

We have studied first the topological properties of the considered graphs like number of nodes N, of edges K and average degree $\langle k \rangle$. Then we focused on the local properties such as the meshedness coefficient M and the number of cycles and on global properties like the cost W and the efficiency E. In particular, the relation between efficiency and cost

unveils that: *it is not necessary to spend a lot of resources in order to achieve a high efficiency.*

We then focused on centrality measures such as betweenness and information. The use of these and other quantities to spotlight the important routes of a city is the main purpose of the multiple centrality assessment (MCA), a metodology to extract information about a city through the study of the spatial distribution of centrality. In addition to this, the same quantities can be also used to extract the skeleton of a city. We have also used an empirical approach to study the correlation between centrality and retail densities. Last, but not the least, we discussed about the time evolution of urban pattern graphs. in particular, represent the new frontier towards which the physicists are moving in their studies on complex networks. These new studies represent the forefront of research in the field of complex networks and will bring soon a flurry of results and novelty to analyze and comprehend also in the realmm of networks of urban street patterns.

APPENDIX A

SOURCE CODE FOR COMPUTING GREEDY TRIANGULATION

```
/* programma per la realizzazione delle Greedy Triangulation da inserire all'interno
   dell'interfaccia
CLI realizzata da Salvo Scellato per i conti del progetto city-form */

/* Realizzato da Alessio Cardillo aka Spariggio */

/* Questo programma sostanzialmente e' l'unione di tre programmi Fortran
   precedentemente realizzati
 e compie le seguenti operazioni:

1) Ricava le coordinate dei nodi a partire dalla tabella di connettivita' ;

2) Crea un grafo completamente connesso a partire dalle coordinate dei nodi ;

3) Ordina i lati in maniera crescente di lunghezza ;

4) Calcola a partire dal grafo completamente connesso la GT ;

*/


/* @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@                              @@
@@         IMPORTANTISSIMISSIMO: gli ID dei nodi vanno da UNO a NODI   @@
@@                              @@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*/
```

```
26 /* Includo gli header necessari */

28 #include <stdio.h>
29 #include <stdlib.h>
30 #include <math.h>
31 #include <string.h>

33 /* definisco alcune quantita' fondamentali */

35 #define COLONNE  8

37 /* definisco anche una chiave per i debug */

39 /* #define DEBUG */


42 /*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
43 C
44 C  DICHIARAZIONE DELLE VARIABILI DA USARE NEL PROGRAMMA
45 C
46 CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/

48 /* variabili costanti */

50 int NODI, LATI ; /* numero di nodi e numero di lati */

52 /* e' piu' conveniente ai fini del sorting e di altre cose immagazzinare i lati all'
```

```
    interno di un
53 array di tipo struct il cui elemento contiene tutte le nformazioni relative a ciascun
       lato */

55 typedef struct {
56    int nfrom ; /* indice del nodo from */
57    float xa ; /* coordinata x del nodo from */
58    float ya ; /* coordinata y del nodo from */
59    float xm ; /* coordinata x media del lato */
60    float ym ; /* coordinata y media del lato */
61    int nto ; /* indice del noto to */
62    float xb ; /* coordinata x del nodo to */
63    float yb ; /* coordinata y del nodo to */
64    float r ; /* lunghezza del lato divisa per due */
65    float m ; /* coefficiente angolare del lato */
66 }lato ; /* ho dichiarato la struct ed anche i suoi componenti */

68 /* puntatori alle strutture dati */

70 float** A ; /* puntatore all'array su cui va la tabella di connettivita' */
71 float** B ; /* puntatore all'array su cui vanno le coordinate dei nodi */
72 lato* tconn ; /* puntatore all'array contenente il grafo completamente connesso */
73 lato* gt ; /* puntatore all'array contenente la greedy triangulation */


76 int NFROM, NTO ; /* variabili temporanee che contengono gli indici dei nodi from e to
       */
```

```
77  float XM, YM, R, M ; /* variabili temporanee che contengono dei valori parziali */

79  int i,j,k ; /* variabili contatore */

81  int lati_add ; /* contatore per vedere quanti lati malloc(NODI * sizeof(float*))) ;
82  ho aggiunto alla GT */

84  int rigl ; /* indice della riga del lato aggiunto alla GT */
85  int l,t ; /* nform & nto sono i numeri dei nodi */
86  int dummy ; /* variabile che non serva ad un cazzo */
87  float dummy2 ; /* variabile che non serva ad un cazzo */
88  int linecount, nodicount ; /* variabili contatore per nodi e lati */


91  /* veriabili per i nomi dei files */

93  char* file_in ; /* variabile contenente il nome del file */
94  char file_out[64] ; /* array contenente il nome del file in I/O */
95  int len ; /* variabile contenente la lunghezza (in caratteri) del nome del file in
        input */


98  /* variabili usate dall'algoritmo brute force per il calcolo della GT */

100 float D12 ; /* distanza tra i due punti medi */
101 float X ; /* coordinata x dell'eventuale intersezione */
102 float Y ; /* coordinata x dell'eventuale intersezione */
```

```
float AX,BX,CX,DX ; /* le x dei vertici da usare nello step 2 dell'algoritmo brute
    force */
float AY,BY,CY,DY ; /* le y dei vertici da usare nello step 2 dell'algoritmo brute
    force */


/*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@
@  DICHIARAZIONE DELLE FUNZIONI
@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*/

/* @@@@@@    FUNZIONE 1    @@@@@@@
questa funzione serve per fare il confronto tra due elementi di un
array di struct (in particolare tra le lunghezze) */

int compfunc_int(const void *x, const void *y)
{

    /* ho dovuto modificare questa funzione perche' non riusciva a fare il confronto
    tra quantita' che differivano tra loro per meno di 1 */

    float diff ; /* variabile che contiene la differenza tra i due valori da esaminare
         */

    diff = ((lato *)x)->r - ((lato *)y)->r ;
```

```
127     /* metto un if che si basa su una tolleranza */
128     if( diff == 0. )
129     { return 0 ;}
130     else if( (diff >= 0.000000001) && (diff > 0.) )
131     { return 1 ;}
132     else if( (diff <= -0.000000001) && (diff < 0.) )
133     {return -1 ;}
134     else
135     {return 0 ;}

137     /* return ((lato *)x)->r - ((lato *)y)->r ; */
138     /*return (int) (*x -*y);*/
139 }


142 //int main() /* dichiarazione del main "classica" */
143 int main( int argc , char** argv) /* mi permette di inserire parametri insieme al
        nome del file eseguibile */
144 {

146     /* if che controlla che il numero di parametri inseriti sia quello corretto */

148     if( argc > 1)
149     {
150         file_in = argv[1] ; /* assegno il nome del file in ingresso */
151     }
152     else
```

```
{
    printf("\n\nATTENZIONE SI E' VERIFICATO UN ERRORE NELL'INSERIMENTO DEL NOME DEL FILE DA ESAMINARE\n\n");
    printf("LA SINTASSI CORRETTA E' ./gt.exe <nome file da esaminare>\n");
    printf("\n Adesso esco dal programma ...... \n");
    exit(1); /* causo l'uscita dal programma */
}


/*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   DETERMINAZIONE DEL NUMERO DI NODI E LATI DEL GRAFO DA ESAMINARE
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/

/* apertura file di input */

FILE *pfile_in ;

#ifdef DEBUG
    printf("sto leggendo il file %s \n", file_in); /* verifica quale file sto leggendo */
#endif

/* Aggiungo un if per controllare se il puntatore al file e' stato assegnato */

if ((pfile_in = fopen( file_in , "r")) == NULL )
```

```
178     {
179         printf("Il file %s non puo' essere aperto \n", file_in);
180         printf("\n Adesso esco dal programma ..... \n");
181         exit(1); /* causo l'uscita dal programma */
182     }

184     /* metto un ciclo while in modo da leggere fino alla fine del file */

186     /* dato che i valori letti vanno messi in una matrice debbo usare un indice
187     per "indirizzare" i dati nel posto "giusto" */

189     /* inizializzo i contatori */

191     linecount = nodicount = 0 ;

193     while (!feof(pfile_in))
194     {
195         for(j=0 ; j < COLONNE ; j++)
196         {
197             fscanf(pfile_in , "%f ", &dummy2);

199             /* metto un if per ricavare il numero di nodi */

201             if( ( (j==1) || (j==4) ) && ( dummy2 > nodicount ) )
202             {
203                 nodicount = (int) dummy2 ;
204             }
```

```
205         }
206             fscanf(pfile_in, "\n"); /* mi serve per andare a capo */
207             linecount++ ; /* incremento il numero di linee contate */
208     }

210     /* Una volta letti i valori necessari chiudo il file */

212     fclose(pfile_in);

214     /*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
215     C
216     C  ALLOCAZIONE DELLE STRUTTURE DATI
217     C
218     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/

220     /* inizializzo i valori del numero di nodi e lati */

222     NODI = nodicount ;
223     LATI = linecount ;

225     /* metto un if per controllare che il numero di nodi non sia troppo grande */

227     if(NODI > 10000)
228     {
229         printf("\n\nATTENZIONE IL NUMERO DI NODI E' MAGGIORE DI 10000 LA MEMORIA
              POTREBBE RISULTARE INSUFFICIENTE ....\n\n");
230     }
```

```
232     /* creo gli array in maniera dinamica */

234     A = (float**) (malloc(LATI * sizeof(float*))) ;

236     for (i = 0 ; i < LATI ; i++)
237     {
238        A[i] = (float*) (malloc(COLONNE* sizeof(float))) ;
239     }


242     B = (float**) (malloc(NODI * sizeof(float*))) ;

244     for (i = 0 ; i < NODI ; i++)
245     {
246        B[i] = (float*) (malloc(3* sizeof(float))) ;
247     }


250     tconn = (lato*) (malloc( ((NODI*(NODI-1))/2) * sizeof(lato))) ;

252     gt = (lato*) (malloc( ((3*NODI)-6) * sizeof(lato))) ;



256     /*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
257     C
```

```
258     C   LETTURA DEL FILE IN INGRESSO
259     C
260     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/


263     /* apertura file di input */

265     #ifdef DEBUG
266        printf("sto leggendo il file %s \n", file_in); /* verifica quale file sto
              leggendo */
267     #endif

269     /* Aggiungo un if per controllare se il puntatore al file e' stato assegnato */

271     //if ((pfile_in = fopen( file1 , "r")) == NULL )
272     if ((pfile_in = fopen( file_in , "r")) == NULL )
273     {
274        printf("Il file %s non puo' essere aperto \n", file_in);
275        printf("\n Adesso esco dal programma ..... \n");
276        exit(1); /* causo l'uscita dal programma */
277     }

279     /* metto un ciclo while in modo da leggere fino alla fine del file */

281     /* dato che i valori letti vanno messi in una matrice debbo usare un indice
282     per "indirizzare" i dati nel posto "giusto" */
```

```
284     i = 0 ; /* inizializzo l'indice */

286     while (!feof(pfile_in))
287     {
288         for(j=0 ; j < COLONNE ; j++)
289         {
290             fscanf(pfile_in , "%f ", &A[i][j]);

292         }
293             fscanf(pfile_in , "\n"); /* mi serve per andare a capo */
294             i++ ; /* incremento i */
295     }

297     /* metto un if per vedere quante righe ho letto */

299     if ( i != LATI)
300     {
301         printf("HO LETTO %d righe , ne dovevo leggere %d \n", i, LATI);
302     }

304     /* Una volta letti i valori necessari chiudo il file */

306     fclose(pfile_in);


309 /*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
310 @@           @@
```

```
311 @@ FASE 1 DEL PROGRAMMA     @@
312 @@                  @@
313 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*/

315     #ifdef DEBUG
316        /* Controllo della corretta lettura del file in ingresso
317        mi faccio stampare le prime 5 righe cosi' sono apposto */

319        printf("\nControllo della corretta lettura file di input \n\n");

321        for(i= 0 ; i < 5 ; i++)
322        {
323           for(j= 0 ; j < COLONNE ; j++)
324           {
325              printf("%f \t", A[i][j] );
326           }

328           printf("\n"); /* vado a capo */
329        }
330     #endif


333     /* inizializzo l'array in cui vanno a finire le coordinate dei nodi */

335     for(i=0; i<NODI; i++)
336     {
337        B[i][0] = B[i][1] = B[i][2] = 0. ; /* inizializzo l'array */
```

```
338     }
340     #ifdef DEBUG
341        /* Stampa del vettore B per controllare l'avvenuta inizializzazione */
342        printf("\n\nStampa del vettore B per controllo inizializzazione\n\n");
343        for(i=0; i<5; i++)
344        {
345           printf("%f \t %f \t %f\n", B[i][0], B[i][1],  B[i][2]);
346        }
347        printf("\n\n");
348     #endif
350     /*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
351     C
352     C  RICERCA DELLE COORDINATE DEI NODI
353     C
354     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/
356     printf("\nSto cercando le coordinate dei nodi ...\n\n");
358     for(i = 1 ; i <= NODI ; i++)
359     {
360        for(j=0 ; j < LATI ; j++)
361        {
362           if( (int) (A[j][1]) == i )
363           {
364              B[(i-1)][0] = (float) i ;
```

```
365             B[(i-1)][1] = A[j][2] ;
366             B[(i-1)][2] = A[j][3] ;
367          }
368          else if ( (int) (A[j][4]) == i)
369          {
370             B[(i-1)][0] = (float) i ;
371             B[(i-1)][1] = A[j][5] ;
372             B[(i-1)][2] = A[j][6] ;
373          }
374       }

376       /* verifica della sostituzione */
377       #ifdef DEBUG
378          printf("\nle coordinate di %d sono: \t x= %f \t y= %f \n", (int) B[(i-1)][0]
                ,B[(i-1)][1] , B[(i-1)][2] );
379       #endif
380    }

382    /* conclusa la fase 1 avviso che inizio a fare la fase 2 */
383    #ifdef DEBUG
384       printf("\nHO CONCLUSO LA FASE 1 DEL PROGRAMMA ..... INIZIO LA FASE 2 \n\n");
385    #endif

387 /*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
388 @@           @@
389 @@ FASE 2 DEL PROGRAMMA @@
390 @@                 @@
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*/

   printf("\nInizio i conti per fare il grafo totalmente connesso .... abbi pazienza\
      n\n");

   /* inizializzo il contatore */

   k = 0 ;

   /* metto un doppio ciclo for per creare tutti i lati necessari */

   for(i = 0 ; i < NODI ; i++)
   {
      for(j= (i+1) ; j < NODI ; j++)
      {
         NFROM = (i+1) ;
         NTO = (j+1) ;
         R  =  sqrt( (pow(( B[j][1] - B[i][1] ) , 2) ) + ( pow( ( B[j][2] - B[i][2] )
            , 2) ) ) / 2. ;
         XM = ( ( B[i][1] + B[j][1] )  / 2. ) ;
         YM = ( ( B[i][2] + B[j][2] )  / 2. ) ;

         /* calcolo il coefficiente angolare ma ci metto un if per evitare gli
            infiniti */
         if( ((( B[j][1] - B[i][1] ) <= 0.000001) && (( B[j][2] - B[i][2] ) >= 1)) ||
            (( B[j][1] - B[i][1] ) == 0.) )
         {
```

```
414             M = 99999. ; /* se e' infinito ci metto questo valore */
415          }
416          else
417          {
418             M = ( ( B[j][2] - B[i][2] ) / ( B[j][1] - B[i][1] ) ) ;
419          }


422          /* Trasferimento dei risultati sulla lista finale */

424          tconn[k].nfrom = NFROM ;
425          tconn[k].xa = B[i][1] ;
426          tconn[k].ya = B[i][2] ;
427          tconn[k].xm = XM ;
428          tconn[k].ym = YM ;
429          tconn[k].nto = NTO ;
430          tconn[k].xb = B[j][1] ;
431          tconn[k].yb = B[j][2] ;
432          tconn[k].r = R ;
433          tconn[k].m = M ;

435          k++ ; /* aggiorno il valore di k */
436       }
437    }


440    #ifdef DEBUG
```

```
441         /* per verificare se ho trasferito correttamente i dati sull'array di struct */
442         printf("\n\n@@@@@ CHECK TRASFERIMENTO DATI SU STRUCT @@@@\n\n");
443         for(i=0 ; i < 5 ; i++)
444         {
445             printf("tconn[%d] = \t %d \t %f \t %f \t %f \t %f \t %d \t %f \t %f \t %f \t
                    %f \n", i, tconn[i].nfrom , tconn[i].xa , tconn[i].ya , tconn[i].xm ,
                   tconn[i].ym , tconn[i].nto , tconn[i].xb , tconn[i].yb , tconn[i].r ,
                   tconn[i].m );
446         }
447     #endif

449     printf("\n\nHO FINITO DI CALCOLARE TUTTI I LATI DEL GRAFO COMPLETAMENTE CONNESSO\n
        ");
450     printf("ADESSO MI APPRESTO AD ORDINARLI IN MODO CRESCENTE\n\n");


453 /*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
454 @@                                    @@
455 @@ FASE 3 DEL PROGRAMMA               @@
456 @@                                    @@
457 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*/

459     /* Inizio della procedura di ordinamento dei lati trovati */

461     /* Occorre disporre i lati in una struttura di tipo array di struct e nella struct
462      metto tutte le   quantita' caratteristiche del singolo lato (ovvero le colonne
            del
```

```
463       file di partenza)

465     Una volta fatto questo, lancio il sorting di questo array usando la routine qsort

467     la routine qsort vuole il puntatore ad una funzione di comparazione e' necessario
468     quindi scrivere la funzione di comparazione tale che questa compari solo un campo
469     della struct in particolare la lunghezza del lato. */

471     /* chiamata della funzione qsort  */

473     qsort( tconn , (((NODI) * (NODI-1)) /2) , sizeof(lato) , compfunc_int ) ;

475     #ifdef DEBUG
476        /* verifica se ho trasferito correttamente i dati sull'array di struct */
477        printf("\n\n@@@ HO FINITO DI ORDINARE LA STRUCT CONTROLLO I CAMBIAENTI @@@@\n\n
              ");
478        for(i=0 ; i < 5 ; i++)
479        {
480           printf("tconn[%d] = \t %d \t %f \t %f \t %f \t %f \t %d \t %f \t %f \t %f \t
                  %f \n",i , tconn[i].nfrom , tconn[i].xa , tconn[i].ya , tconn[i].xm ,
                  tconn[i].ym , tconn[i].nto , tconn[i].xb , tconn[i].yb , tconn[i].r ,
                  tconn[i].m );
481        }
482     #endif


485  /*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
@@                    @@
@@ FASE 4 DEL PROGRAMMA      @@
@@                    @@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*/

    printf("\nINIZIO IL CALCOLO DELLA GT ABBI PAZIENZA ....\n\n");


    /* inizio del calcolo della GT mediante una procedura di tipo brute force */

    /* ALGORITMO BRUTE FORCE PER GT

    c---------------------------------------------------------------------------

    c Inizio dell'algoritmo brute force by Vito Lator. Per velocizzare il tutto
    e per dare un senso alla Greedy Triangulation bisogna avere i dati ordinati
    per lunghezza crescente del raggio!!!! */


    /* il primo lato va sicuramente preso gli altri poi seguono */

    #ifdef DEBUG
       printf("\n\nAggiungo il primo lato \n\n");
    #endif

    gt[0].nfrom =  tconn[0].nfrom ;
    gt[0].xa     =    tconn[0].xa ;
```

```
513     gt[0].ya      =     tconn[0].ya ;
514     gt[0].xm      =     tconn[0].xm ;
515     gt[0].ym      =     tconn[0].ym ;
516     gt[0].nto     =     tconn[0].nto ;
517     gt[0].xb      =     tconn[0].xb ;
518     gt[0].yb      =     tconn[0].yb ;
519     gt[0].r    =     tconn[0].r ;
520     gt[0].m    =     tconn[0].m ;

522     #ifdef DEBUG
523        printf("\n\nProcedo con l'aggiunta degli altri lati .... \n\n");
524     #endif

526     /* inserisco un ciclo while che itera finche' il numero di lati aggiunti non
527     e' pari a 3N - 6 per farlo uso un contatore che mi dice a che punto sono
528     arrivato nell'aggiungere lati */

530     /* inizializzo il contatore */

532     lati_add = 1 ;

534     /* inizializzo anche l'indice che mi dice a quale riga dell'array corrisponde il
           lato aggiunto */
535     rigl = 0 ;

537     while( lati_add < ((3*NODI) - 6) ) /* lo metto minore e non diverso cosi' sono
           sicuro */
```

```
538     {
540         /* inserisco un doppio ciclo for per fare il parsing dei lati "elegibili" */
542         for( j = (rigl+1) ; j < ((NODI * (NODI - 1)) / 2) ; j++)
543         {
544             for( i = 0 ; i < lati_add ; i++) /* controlla bene i limiti di iterazione di
                    questo ciclo */
545             {
546                 /*Calcolo la distanza tra i punti medii d12 e la x dell'eventuale
547                 intersezione anche se computazionalmente non conviene*/
549                 AX = gt[i].xa ;
550                 AY = gt[i].ya ;
551                 BX = gt[i].xb ;
552                 BY = gt[i].yb ;
554                 CX = tconn[j].xa ;
555                 CY = tconn[j].ya ;
556                 DX = tconn[j].xb ;
557                 DY = tconn[j].yb ;
559                 /*L'algoritmo usa la posizione che Xa<Xb e Xc<Xd
560                 Occorre un if che verifichi tale condizione ed eventualmente la realizzi
                       */
562                 if (BX < AX)
```

```
563             {
564                AX = gt[i].xb ;
565                AY = gt[i].yb ;
566                BX = gt[i].xa ;
567                BY = gt[i].ya ;
568             }

570             if (DX < CX)
571             {
572                CX = tconn[j].xb ;
573                CY = tconn[j].yb ;
574                DX = tconn[j].xa ;
575                DY = tconn[j].ya ;
576             }

578             /* Calcolo della distanza tra i due punti medii dei segmenti */

580             D12 = (sqrt( pow(( gt[i].xm - tconn[j].xm ),2) + pow(( gt[i].ym - tconn[j
                   ].ym ),2) )) ;


583             /* Calcolo delle coordinate X e Y dell'intersezione tra le rette basate
                   sui lati */

585             X = ( ( gt[i].ym - ( gt[i].m * gt[i].xm ) - ( tconn[j].ym - ( tconn[j].m
                   * tconn[j].xm ) ) ) / ( tconn[j].m - gt[i].m ) ) ;
```

```
587             Y = ( ( ( tconn[j].m * gt[i].ym ) - ( gt[i].m * tconn[j].ym ) + ( ( tconn
                    [j].m * gt[i].m ) * ( tconn[j].xm - gt[i].xm ) ) ) / ( tconn[j].m - gt
                    [i].m ) ) ;

589             /* Qui ci sono i check da fare su ogni candidato
590             c-------------------------------------------------------------------------
591             c  STEP1
592             c-------------------------------------------------------------------------*/

594             if ( D12 > ( ( gt[i].r + tconn[j].r ) ) )
595             {
596                /* Il candidato viene accettato */
597                #ifdef DEBUG
598                   printf("\ni lati sono distanti, passo avanti ...\t %d \t %d \t %d\n
                         \n", i, j, lati_add );
599                #endif

601                goto nove ;
602             }

604             /*-------------------------------------------------------------------------
605             c  STEP2
606             c-------------------------------------------------------------------------*/

608             /* Controllo per i lati "paralleli" */

610             else if ( ( gt[i].m == tconn[j].m ) && ( ( (BX == DX) && (BY == DY) ) ) )
```

```
611             {
612                 #ifdef DEBUG
613                     printf("\nil lato va scartato perche' parallelo\t %d \t %d \t %d\n\
                        n", i,j, lati_add );
614                 #endif
615
616                 goto sei ;
617             }
618             else if ( ( gt[i].m == tconn[j].m ) && ( ( (AX == CX) && (AY == CY) ) ) )
619             {
620                 #ifdef DEBUG
621                     printf("\nil lato va scartato perche' parallelo\t %d \t %d \t %d\n\
                        n", i,j, lati_add );
622                 #endif
623
624                 goto sei ;
625             }
626
627             /* Check per i lati con estremi coincidenti */
628
629             else if ( (BX == CX) && (BY == CY) )
630             {
631                 /* Se B coincide con C il candidato viene accettato */
632                 #ifdef DEBUG
633                     printf("\nB coincide con C, passo avanti \t %d \t %d \t %d\n\n", i,
                        j, lati_add );
634                 #endif
```

```
636                 goto nove ;
637             }
638             else if ( (AX == CX) && (AY == CY) )
639             {
640                 /* Se A coincide con C il candidato viene accettato */
641                 #ifdef DEBUG
642                     printf("\nA coincide con C, passo avanti \t %d \t %d \t %d\n\n", i,
                            j, lati_add );
643                 #endif
644
645                 goto nove ;
646             }
647             else if ( (AX == DX) && (AY == DY) )
648             {
649                 /* Se A coincide con D il candidato viene accettato */
650                 #ifdef DEBUG
651                     printf("\nA coincide con D, passo avanti \t %d \t %d \t %d\n\n", i,
                            j, lati_add );
652                 #endif
653
654                 goto nove ;
655             }
656             else if ( (BX == DX) && (BY == DY) )
657             {
658                 /* Se B coincide con D il candidato viene accettato */
659                 #ifdef DEBUG
```

```
660                 printf("\nB coincide con D, passo avanti \t %d \t %d \t %d\n\n", i,
                        j, lati_add );
661             #endif

663             goto nove ;
664         }
665         /* Check per i lati "strani"
666         c Quando uno dei due lati e o orizzontale o verticale bisogna fare
                scattare
667         c questo check. Bisogna pero tenere conto del fatto che i lati sono
                orientati
668         c rispetto alle coordinate x. Questo comporta uno sdoppiamento dei check
669         c a causa delle diverse coordinate y dei lati. Inoltre il check si basa
                su un doppio
670         c controllo sia su X che su Y che altrimenti creerebbe dei bug nella
                capacita
671         c del programma di rilevare lati con intersezioni strane. Questo ha
                consentito
672         c di eliminare anche il check sui lati a croce. */

674         /* @@@@@ Caso 1 @@@@@   */

676         else if ( ( ( (AX == BX) && (CY > DY) ) && (AY > BY) ) && ( ( (X >= CX)
               && (X <= DX) ) && ( ( (Y <= CY) && (Y >= DY) ) && ( (Y <= AY) && (Y >=
                BY) ) ) ) )
677         {
678             #ifdef DEBUG
```

```
679                 printf("\nl lati si intersecano in modo strano 1a \t %d \t %d \t %d
                        \n\n", i,j, lati_add );
680             #endif

682             goto sei ;
683         }
684         else if ( ( ( (AX == BX) && (CY > DY) ) && (AY < BY) ) && ( ( (X >= CX)
               && (X <= DX) ) && ( ( (Y <= CY) && (Y >= DY) ) && ( (Y >= AY) && (Y <=
                BY) ) ) ) )
685         {
686             #ifdef DEBUG
687                 printf("\nl lati si intersecano in modo strano 1b \t %d \t %d \t %d
                        \n\n", i,j, lati_add );
688             #endif

690             goto sei ;
691         }

693         /* @@@@@ Caso 2 @@@@@   */

695         else if ( ( ( (AX == BX) && (DY > CY) ) && (AY < BY) ) && ( ( (X >= CX)
               && (X <= DX) ) && ( ( (Y <= DY) && (Y >= CY) ) && ( (Y >= AY) && (Y <=
                BY) ) ) ) )
696         {
697             #ifdef DEBUG
698                 printf("\nl lati si intersecano in modo strano 2a \t %d \t %d \t %d
                        \n\n", i,j, lati_add );
```

```
699                 #endif

701                 goto sei ;
702             }
703             else if ( ( ( (AX == BX) && (DY > CY) ) && (AY > BY) ) && ( ( (X >= CX)
                    && (X <= DX) ) && ( ( (Y <= DY) && (Y >= CY) ) && ( (Y <= AY) && (Y >=
                     BY) ) ) ) )
704             {
705                 #ifdef DEBUG
706                     printf("\nl lati si intersecano in modo strano 2b \t %d \t %d \t %d
                        \n\n", i,j, lati_add );
707                 #endif

709                 goto sei ;
710             }

712             /* @@@@@ Caso 3 @@@@@    */

714             else if ( ( (CY == DY) && (AY > BY) ) && ( ( ( (X >= AX) && (X  <= BX) )
                    && ( (X >= CX) && (X <= DX) ) ) && ( (Y >= BY) && (Y <= AY) ) ) )
715             {
716                 #ifdef DEBUG
717                     printf("\nl lati si intersecano in modo strano 3a \t %d \t %d \t %d
                        \n\n", i,j, lati_add );
718                 #endif

720                 goto sei ;
```

```
721             }
722             else if ( ( (CY == DY) && (BY > AY) ) && ( ( ( (X >= CX) && (X <= DX) )
                   && ( (X >= AX) && (X <= BX) ) ) && ( (Y >= AY) && (Y <= BY) ) ) )
723             {
724                 #ifdef DEBUG
725                     printf("\nI lati si intersecano in modo strano 3b \t %d \t %d \t %d
                           \n\n", i,j, lati_add );
726                 #endif
728                 goto sei ;
729             }
731             /* @@@@@  Caso 4  @@@@@ */
733             else if ( ( (AY == BY) && (CY > DY) ) && ( ( ( (X >= AX) && (X <= BX) )
                   && ( (X >= CX) && (X <= DX) ) ) && ( (Y <= CY) && (Y >= DY) ) ) )
734             {
735                 #ifdef DEBUG
736                     printf("\nI lati si intersecano in modo strano 4a \t %d \t %d \t %d
                           \n\n", i,j, lati_add );
737                 #endif
739                 goto sei ;
740             }
741             else if ( ( (AY == BY) && (CY < DY) ) && ( ( ( (X >= CX) && (X <= DX) )
                   && ( (X >= AX) && (X <= BX) ) ) && ( (Y >= CY) && (Y <= DY) ) ) )
742             {
```

```
743                 #ifdef DEBUG
744                     printf("\nl lati si intersecano in modo strano 4b \t %d \t %d \t %d
                        \n\n", i,j, lati_add );
745                 #endif
747                 goto sei ;
748             }
750             /* @@@@@  Caso 5  @@@@@ */
752             else if ( ( ( (AY == BY) && (CX == DX) ) && ( (DY > CY) ) ) && ( ( (X >=
                    AX) && (X <= BX) ) && ( (Y <= DY) && (Y >= CY) ) ) )
753             {
754                 #ifdef DEBUG
755                     printf("\nl lati si intersecano in modo strano 5a \t %d \t %d \t %d
                        \n\n", i,j, lati_add );
756                 #endif
758                 goto sei ;
759             }
760             else if ( ( ( (AY == BY) && (CX == DX) ) && ( (DY < CY) ) ) && ( ( (X >=
                    AX) && (X <= BX) ) && ( (Y >= DY) && (Y <= CY) ) ) )
761             {
762                 #ifdef DEBUG
763                     printf("\nl lati si intersecano in modo strano 5b \t %d \t %d \t %d
                        \n\n", i,j, lati_add );
764                 #endif
```

```
766                 goto sei ;
767             }

769             /* @@@@@  Caso 6  @@@@@ */

771             else if ( ( ( (CY == DY) && (AX == BX) ) && ( (BY > AY) ) ) && ( ( (X >=
                    CX) && (X <= DX) ) && ( (Y <= BY) && (Y >= AY) ) ) )
772             {
773                 #ifdef DEBUG
774                     printf("\nl lati si intersecano in modo strano 6a \t %d \t %d \t %d
                            \n\n", i,j, lati_add );
775                 #endif

777                 goto sei ;
778             }
779             else if ( ( ( (CY == DY) && (AX == BX) ) && ( (BY < AY) ) ) && ( ( (X >=
                    CX) && (X <= DX) ) && ( (Y <= AY) && (Y >= BY) ) ) )
780             {
781                 #ifdef DEBUG
782                     printf("\nl lati si intersecano in modo strano 6b \t %d \t %d \t %d
                            \n\n", i,j, lati_add );
783                 #endif

785                 goto sei ;
786             }
```

```
788             /* @@@@@  Check per le intersezioni  @@@@@      */

790             else if ( ( (X >= AX) && (X <= BX) ) && ( (X >= CX) && (X <= DX) ) )
791             {
792                 /* Il candidato viene scartato perche' si interseca con altri lati */
793                 #ifdef DEBUG
794                     printf("\ni lati si intersecano scarto il lato \t %d \t %d \t %d\n\
                        n", i, j, lati_add );
795                 #endif

797                 goto sei ;
798             }

800             /* il lato in esame ha superato tutti i test */

802 nove:
803             dummy = dummy ; /* gli faccio fare una cosa che non serve ad un cazzo */

805             } /* fine del for su i */

807         /* il lato viene accettato */
808         #ifdef DEBUG
809             printf("\nIl lato e' definitivamente accettato \t %d \t %d \t %d\n\n", i,
                    j, lati_add );
810         #endif

812         /* Viene memorizzata la riga del lato accettato */
```

```
814             rigl = j ;

816             /* Trasferimento del lato dalla lista tconn a gt */

818                gt[lati_add].nfrom =    tconn[j].nfrom ;
819                gt[lati_add].xa    =    tconn[j].xa ;
820                gt[lati_add].ya    =    tconn[j].ya ;
821                gt[lati_add].xm    =    tconn[j].xm ;
822                gt[lati_add].ym    =    tconn[j].ym ;
823                gt[lati_add].nto   =    tconn[j].nto ;
824                gt[lati_add].xb    =    tconn[j].xb ;
825                gt[lati_add].yb    =    tconn[j].yb ;
826                gt[lati_add].r     =    tconn[j].r ;
827                gt[lati_add].m     =    tconn[j].m ;

829             goto uno ;

831  sei:
832          dummy = dummy ; /* gli faccio fare un'operazione che non serve ad un cazzo */

834          } /* fine del for su j */


837  uno:
838          /* prima di chiudere il ciclo aggiorno il contatore */
839          lati_add ++ ;
```

```
841         #ifdef DEBUG
842            printf("\n\nil numero di lati aggiunti e' %d\n\n", lati_add );
843         #endif

845      } /* fine del ciclo while */

847      /*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
848      C
849      C  SCRITTURA DEL FILE IN USCITA
850      C
851      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/

853      /* esecuzione di alcune operazioni sulle stringhe per creare il nome del file in
            uscita */

855      len=strlen(file_in) ; /* calcolo la lunghezza del nome del file e la metto in una
            variabile */

857      strncpy(file_out , file_in , ( len - 4 ) ) ; /* copio la stringa in in quella out
            */

859      strcat( file_out , "-greedy.dat") ; /* concateno la stringa ottenuta con la
            desinenza voluta */

861      /* Apertura del file in uscita */
862      FILE *pfile_out ;
```

```
864     #ifdef DEBUG
865        printf("sto scrivendo il file %s \n", file_out); /* verifica quale file sto
              leggendo */
866     #endif

868     /* if per controllare se il puntatore al file e' stato assegnato */

870     if ((pfile_out = fopen( file_out , "w")) == NULL )
871     {
872        printf("Il file %s non puo' essere aperto \n", file_out);
873        printf("\n Adesso esco dal programma ..... \n");
874        exit(1); /* causo l'uscita dal programma */
875     }
876     else
877     {
878        #ifdef DEBUG
879           printf("\nSto scrivendo la lista finale .....\n\n");
880        #endif

882        /* Trasferimento dei risultati sul file finale */

884        for(i=0 ; i < ( (3*NODI) - 6) ; i++)
885        {
886           /* Dato che non tutti i lati realmente appartenenti alla GT non possono
                 essere aggiunti, allora metto un if
887           per filtrare quelli completamente vuoti */
```

```
889             if( (gt[i].r) != 0.) /* mi serve a stampare su file solo quelli non nulli */
890             {
891                 fprintf(pfile_out, "%d \t %f \t %f \t %d \t %f \t %f \t %f \n", gt[i].
                        nfrom , gt[i].xa , gt[i].ya , gt[i].nto , gt[i].xb , gt[i].yb , (2*(gt
                        [i].r)) );
892             }

894             /* mi faccio stampare le prime cinque righe solo per controllo */
895             #ifdef DEBUG
896                 if(i < 5)
897                 {
898                     printf("%d \t %f \t %f \t %d \t %f \t %f \t %f \n", gt[i].nfrom , gt[i
                            ].xa , gt[i].ya , gt[i].nto , gt[i].xb , gt[i].yb , (2*(gt[i].r)) )
                            ;
899                 }
900             #endif
901         }
902     }

904     /* chiusura del file */

906     fclose(pfile_out) ;

908     printf("\nNO FINITO\n\n");

910 } /* fine del main */
```

RINGRAZIAMENTI

Una volta **un amico** mi disse: "i ringraziamenti vanno scritti bene perché sono l'unica cosa che la gente legge in una tesi" (chi doveva capire ha capito). Dietro questa semplice, quanto mai assoluta verità si nasconde un compito assai difficile (almeno per me).

Dall'ultima volta in cui ho dovuto scrivere dei ringraziamenti (ovvero quattro anni fa) di acqua sotto i ponti ne è passata davvero tanta: ho meno capelli; sono più vecchio (ma più bello); sono stato, anche se solo per una volta, capitano della mia squadra; sono ancora l'unico cretino tra i miei colleghi con cui sono entrato all'università a non essermi laureato (oggi ci ho messo finalmente una pezza a 'sta storia); ecc ecc ...

Di persone sulla mia strada ne ho incontrate molte ma alcune di queste hanno davvero *lasciato un segno* attraverso un gesto, una parola o anche solo semplicemente con il loro comportamento. Prima di passare alle "dediche personalizzate" volevo dire GRAZIE a tutte quelle persone che ho conosciuto e che mi hanno dato una mano, piccola o grande che sia, per arrivare fin dove sono arrivato.

Volevo anche ringraziare tutti quelli che invece hanno per così dire "remato contro" e che tramite il loro contributo hanno reso il mio percorso meno facile. Alla facciazza vostra ce l'ho fatta lo stesso (uno a zero e

palla al centro).

Ma bando alle ciance e veniamo a noi (capisco che siete curiosi ma dopo un pò uno si scoccia). Vorrei ringraziare:

- Vito Latora. Per avermi introdotto al mondo delle reti complesse e per tutte le cose che mi ha insegnato in questi cinque (o sono sei?) anni passati sotto la sua guida. Guardando indietro nel tempo posso dire di essere davvero fortunato ad aver avuto un "capo" come lui dal quale ho potuto apprendere un sacco di cose sia a livello scientifico ma soprattutto umano. Una volta mi disse che per lui io ero una "scommessa", spero che che l'abbia vinta la scommessa ed anche se così non fosse lo ringrazio lo stesso per aver creduto in me.

- Mia mamma e mio fratello. La prima per tutto l'amore che profonde nei suoi pranzetti ipercalorici e per il supporto che mi ha dato in questi anni con frasi del tipo:"Bravo, ma quand'è che fai il prossimo esame?" (detta, appena dopo aver dato un esame).
  Il secondo invece per l'amorevole modo con cui mi cazzia un giorno si e l'altro pure (a volte con più sessioni giornaliere) con il fine ultimo di rendermi appena appena più furbo.

- Giuseppe Angilella. Quest'uomo noto per essere il santo protettore dello studente incasinato, piombò nella vita del sottoscritto per ragioni di tipo "informatico" e da allora non se n'è più voluto andare. Occorrerebbe un intero libro per poter descrivere il rapporto che mi lega a lui, fatto di irruzioni nei rispettivi uffici al solo scopo di mandarsi reciprocamente a quel paese o di sguardi e pacche sulle spalle che sottendono intere discussioni. Una specie di fratello maggiore. (Vaff...).

- Jesus, Roberta, Enzo e Maria. Cosa sarebbe la vita se al lavoro fossi circondato da gente con cui non vai d'accordo? Un vero inferno. Invece, io ho sempre avuto la fortuna di lavorare con gente con cui vado d'accordo (in realtà sono io quello da sopportare). Un grazie quindi ad Jesus per tutte le cose che mi ha insegnato e per le mille avventure passate insieme in questi anni. A Roberta, per la pazienza con cui accetta il ruolo di "vittima" dei miei pesantissimi scherzi nonché per le consulenze sulle cose BIO. Enzo, per tutte quelle volte in cui resto a bocca aperta a vederlo letteralmente "dialogare" col computer e per il ruolo di spalla negli scherzi a Roberta ed infine a Mariuccia, per tutte le splendide serate e jam-sessions passate durante la sua permanenza qua a Catania. Lavorare con voi è davvero uno spasso.

- Secondo me una cosa bella dell'università è quel clima di spensieratezza che si respira quando fai le cose (almeno nei primi anni prima di cominciare ad "invecchiare"). Vorrei quindi rivolegere un ringraziamento speciale ai componenti del cosiddetto "quintetto etilo-lescano" coi quali ho trascorso momenti bellissimi (i nostri fegati ringraziano):

  – Gabriele: perché, nonostante sia emigrato nella terra dei tortellini, non perde occasione per farsi sentire e per il continuo e costante supporto che mi da. Grande appasionato di manga e carne di cavallo come me è uno dei pochi da cui accetto consigli e critiche (per altro sempre azzeccatissimi).

  – Danilo: c'è una canzone dei Punkreas che recita: "eeeehhhh ... prvaticamente mi sono pervso". Lui rappresenta l'unico caso al mondo di polentone convertito a terrone con successo ed in buona misura il (de)merito di ciò è mio. Mi piace ricordare le ore passate sui libri a cercare di risolvere esercizi, il

cui risultato veniva puntualmente "aggiustato" mediante l'applicazione del celeberrimo metodo Jaccarino, intervallati da una citazione cinematografica, letteraria o musicale che sia. Grazie, vecchio trapano!

- T*ħ*ænøo (ad Enna si scrive così): l'unico uomo ad avere più soprannomi dei granelli di sabbia della spiaggia di Palm Beach in California. Grandissimo compagno di bevute (e per questo conosciuto anche come *figlio illegittimo di Bacco*) e campione mondiale nei giochi Nomi, Città, Cose e Briscola in 5. Dotato di una pazienza infinita (non ci ha ancora ucciso nonostante abbiamo coniato per lui qualcosa come 126.000 epiteti diversi), non perde occasione per passarti a trovare quando torna in quel del centro del mondo (Enna) avendo scelto di truffare inconsapevoli risparmiatori con teorie economiche assurde in quel di Milano.
- Salvo: grandissimo fan del leggendario Pino Puggelli è quello che sembra non colparci nulla col suo atteggiamento "tranquillo", ma che sotto sotto ne combina di cotte e crude. Lui tra una canzone dei Led Zeppelin e un'altra degli Oasis ama sfoggiare citazioni di carattere sovietico. È anche quello che una volta a casa di T*ħ*ænøo propose di sfilare un cavatappi da una bottiglia di Lancers usando la termodinamica. Per me resterai sempre quello di A'VOUGLIA.

- Ma la vita non è solo lavoro (per fortuna). Quando penso ai miei amici posso tranquillamente affermare di essere una persona fortunata. Nel film The big Kahuna c'è una frase che dice: "Renditi conto che gli amici vanno e vengono. Ma alcuni, i più preziosi, rimarranno". Tra i miei amici preziosi vorrei ringraziare Paola per gli innumerevoli caffé e le piacevoli chiaccerate, Cristina per avermi

eletto quale unico beneficiario dei suoi "in bocca al lupo", Stefano con cui ho condiviso le fatiche degli esami degli ultimi anni e che non perde occasione per passarmi a trovare in ufficio, i miei compagni di classe Alex e Rauni che non hanno mai smesso di fare il tifo per me dal giorno in cui le nostre strade si sono divise. Una menzione speciale va ai miei compagni di squadra Antonio, Peppe puglia, Luca, Peppe spina ed al mio allenatore Michelangelo veri e propri *compagni d'armi* per i quali le parole non bastano.

- Un ringraziamento davvero particolare va ad Angela. Una sola parola: MI SOPPORTA. Sembra poco ma vi assicuro che non è da tutti riuscire a farlo. Questo perché oltre ad essere uno stronzo di prima categoria, ho anche qualcosa come 26 milioni di difetti. Nonostante tutto questo, lei è sempre là al mio fianco cercando ogni giorno, mollichina a mollichina, di farmi diventare una persona migliore. Grazie per tutto quello che fai.

- Per finire un ringraziamento per due persone che purtroppo non potranno leggerlo. Vorrei ringraziare Antonino Copia per avermi insegnato ad affrontare la vita sempre col sorriso sulle labbra. La sua spensieratezza, simpatia (non l'ho mai visto triste eccetto che una volta) ed il suo apparente non voler prendere sul serio nulla, anche quando la vita non è "tenera" con te (e non succede di rado), rappresentano ormai una parte integrante (ed importante) del mio carattere e di questo gli sarò eternamente grato. Vorrei ringraziare anche il Prof. Giovanni Raciti per avermi impartito quella che io ritengo la LEZIONE NUMERO 1 dal titolo: PASSIONE. La passione è quella cosa che ti permette di: stare 12 ore in ufficio senza sentirne il peso semplicemente perché stai facendo quello che ti piace; lavorare sabati, domeniche, giorno, notte e di essere comunque contento; percepire uno stipendio schifosamente basso o di non

percepirne affatto ma di essere comunque li perché non sapresti che altro fare. Lo ringrazio anche perché è stata una delle primissime persone (prima ancora di Vito) ad aver visto qualcosa in me che meritasse il suo appoggio e la sua fiducia. Se non ci fosse stato lui non sarei dove sono oggi, grazie grazie grazie.

Prima di lasciarvi, per premiarvi della vostra pazienza nella lettura di questo papello, volevo farvi dono di tre citazioni a cui sono particolarmente affezionato e che utilizzo spesso come luce guida nella vita di tutti i giorni.

> E così, nelle operazioni militari:
> Se conosci il nemico e conosci te stesso,
> Nemmeno in cento battaglie ti troverai in pericolo.
> Se non conosci il nemico ma conosci te stesso,
> Le tue possibilità di vittoria sono pari a quelle di sconfitta.
> Se non conosci né il nemico né te stesso,
> Ogni battaglia significherà per te sconfitta certa.
>
> Sun Tzu – L'arte della guerra

> The spotted hawk swoops by and accuses me, he complains of my gab and my loitering.
>
> I too am not a bit tamed, I too am untraslatable,
> I sound my barbaric yawp over the roofs of the world.
>
> Walt Whitman – Foglie d'erba

> Ogni ostacolo, ogni muro di mattoni, è li per un motivo preciso. Non è li per escluderci da qualcosa, ma per offrirci la possibilità di dimostrare in che misura ci teniamo. I muri di mattoni sono li per fermare le persone che non hanno abbastanza voglia di superarlo. Sono li per fermare gli altri.
>
> Randy Paush – The last lecture

BIBLIOGRAPHY

[1] D. J. Watts S. H. Strogatz, Nature (1998).

[2] A.L. Barabási R. Albert, Science **286**, 509 512 (1999).

[3] R. Albert and A.-L. Barabási, Rev. Mod. Phys. **74**, (2002).

[4] M.E.J. Newman, SIAM Review **45**, (2003).

[5] A. Vespignani R. Pastor-Satorras, *Evolution and Structure of the Internet: A Statistical Physics Approach* (Cambridge University Press, Cambridge, UK, 2004).

[6] S. Boccaletti V. Latora Y. Moreno M. Chavez and D.-U. Hwang, Phys. Rep. **424**, (2006).

[7] M. E. J. Newman, *Networks: An Introduction* (Oxford University Press, USA, 2010).

[8] R. Pastor-Satorras A. Vázquez and A. Vespignani, Phys. Rev. Lett. **87**, (2001).

[9] R. Milo S. Shen-Orr S. Itzkovitz N. Kashan D. Chklovskii and U. Alon, Science **298**, (2002).

[10] S. Wasserman and K. Faust, *Social Network Analysis* (Cambridge University Press, Cambridge, 1994).

[11] M. E. J. Newman, Phys. Rev. E **64**, (2001).

[12] M. E. J. Newman, Phys. Rev. E **64**, (2001).

[13] M. E. J. Newman, Proc. Natl. Acad. Soc. **98**, 404 (2001).

[14] A. Cardillo S. Scellato and V. Latora, Physica A **372**, 333 (2006).

[15] M. T. Gastner and M. E. J. Newman, J. Stat. Mech. (2006).

[16] O. Sporns, Complexity **8**, (2003).

[17] S.-H. Yook H. Jeong and A.-L. Barabási, Proc. Natl. Acad. Sci. U.S.A. **99**, (2002).

[18] V. Latora and M. Marchiori, Phys. Rev. E **71**, (2005).

[19] R. Albert R. Kinney, P. Crucitti and V. Latora, Eur. Phys. J. B **46**, (2005).

[20] F. Pitts, The Professional Geographer **17**, (1965).

[21] R. Guimerà S. Mossa A. Turtschi and L.A.N. Amaral, Proc. Natl. Acad. Sci. USA **102**, (2005).

[22] R. Pastor-Satorras A. Barrat, M. Barthélemy and A. Vespignani, Proc. Natl. Acad. Sci. USA **101**, (2004).

[23] P. Crucitti S. Porta and V. Latora, Environ. Plan. B **33**, (2006), physics/0506009.

[24] J. Buhl J. Gautrais R. V. Solé P. Kuntz S. Valverde J. L. Deneubourg and G. Theraulaz, Eur. Phys. J. B **42**, (2004).

[25] M. T. Gastner and M. E. J. Newman, Eur. Phys. J. B **49**, 247 (2006), cond-mat/0407680.

[26] M. Marchiori V. Latora, Phys. Rev. Lett. **87**, (2001).

[27] M. Marchiori V. Latora, Eur. Phys. J. B **32**, (2003).

[28] S. S. Shen-Orr R. Milo S. Mangan and U. Alon, Nature Genetics **31**, 64 (2002).

[29] N. Kashtan S. Itzkovitz R. Milo U. Alon, Bioinformatics **11**, (2004).

[30] V. Latora S. Scellato, A. Cardillo and S. Porta, Eur. Phys. J. B **50**, (2006).

[31] V. Latora A. Cardillo, S. Scellato and S. Porta, Phys. Rev. E **73**, (2006).

[32] S. Porta E. Strano V. Iacoviello R. Messora V. Latora A. Cardillo Fahui Wang and S. Scellato, Environ. Plan. B **36**, 450 (2009), physics/0701111.

[33] B. Bollobàs, *Random Graphs* (Academic Press, London, 1985).

[34] B. Bollobàs, *Modern Graph Theory*, *Graduate Text in Mathematics* (Springer, New York, 1998).

[35] D. B. West, *Introduction to Graph Theory* (Prentice-Hall, Englewood Cliffs, NJ, 1995).

[36] J.A. Bondy U.S.R. Murty, *Graph Theory*, Vol. 244 of *Graduated Texts in Mathematics* (Springer, London (UK), 2008).

[37] E. Katifori G. J. Szöllősi and M. O. Magnasco, Phys. Rev. Lett. **104**, (2010).

[38] M. Girvan M. E. J. Newman, Phys. Rev. E **69**, (2004).

[39] V. Latora M. Marchiori, Physica A **285**, (2000).

[40] M. Granovetter, American J. Sociology **78**, (1973).

[41] R. Cohen K. Erez D. ben Avraham and S. Havlin, Phys. Rev. Lett. **85**, (2000).

[42] R. Cohen K. Erez D. ben Avraham and S. Havlin, Phys. Rev. Lett. **86**, (2001).

[43] R. Cohen S. Havlin and D. ben Avraham, Phys. Rev. Lett. **91**, (2003).

[44] M.E.J. Newman S.H. Strogatz and D.J. Watts, Phys. Rev. E **64**, (2001).

[45] L.A.N. Amaral A. Scala M. Barthélemy H.E. Stanley, Proc. Natl. Acad. Sci. (USA) **97**, (2000).

[46] R. Pastor-Satorras M. Boguñà, Phys. Rev. E **66**, (2002).

[47] A. Vespignani M. Boguñà, R. Pastor-Satorra, Lect. Notes Phys. **625**, (2003).

[48] M. E. J. Newman, Phys. Rev. Lett. **89**, (2002).

[49] P. Crucitti S. Porta and V. Latora, Physica A **369**, (2006), cond-mat/0411241.

[50] D. J. Watts, *Small Worlds: The Dynamics of Networks between Order and Randomness* (Princeton University Press, Princeton, NJ, 1999).

[51] D. Stauffer and A. Aharony, *Introduction To Percolation Theory* (CRC Press, USA, 1992).

[52] B. Mohar and C. Thomassen, *Graphs on Surfaces* (Johns Hopkins Press, Baltimore, USA, 2001).

[53] M. Tumminello T. Aste T. Di Matteo and R. N. Mantegna, Proc. Natl. Acad. Soc. **102**, 10421 (2005).

[54] B. B. Mandelbrot, *The Fractal Geometry of Nature* (W. H. Freeman, USA, 1983).

[55] S. H. Strogatz, *Nonlinear Dynamics and Chaos with Applications to Physics, Biology, Chemistry and Engineering* (Perseus, Cambridge, 1994).

[56] P. Grassberger I Procaccia, Physica D **9**, 189 (1983).

[57] K. Chang, *Introduction to Geographic Information System*, 4th ed. (McGraw Hill, USA, 2007).

[58] Wikipedia the Free Encyclopedia, Geographic Information System, 2010.

[59] J. Summers, *Soho – A History of London's Most Colourful Neighborhood* (Bloomsbury, London, UK, 1989), pp. 113–117.

[60] H.J. Timmermans J.A. Teklemburg and A.F. Van Wagemberg, Environment and Planning B **20**, (1993).

[61] A. Penn, Environment and Behavior **35**, (2003).

[62] N. Dalton J. Peponis and R. Dalton, in *Proceedings of the 4th International Space Syntax Symposium* (PUBLISHER, London, UK, 2003).

[63] P. Crucitti S. Porta and V. Latora, Environ. Plan B **33**, (2006).

[64] M. Rosvall A. Trusina P. Minnhagen and K. Sneppen, Phys. Rev. Lett. **94**, (2005).

[65] R. Dalton, Environment and Behavior **35**, (2003).

[66] S. Marshall, *Streets and Patterns* (Routledge, London, UK, 2004).

[67] M.T. Dikerson R.L.S. Drysdale S.A. McElfresh and E.Welzl, Computational Geometry **8**, (1997).

[68] G. H. Golub C. F. Van Loan, *Matrix Computations*, 3rd ed. (Johns Hopkins, Baltimore, USA, 1996).

[69] *Numpy and Scipy Documentation – Sparse matrices*, 2010.

[70] T. H. Cormen C. E. Leierson R. L. Rivest and C. Stein, *Introduction to Algorithms* (MIT University Press, Cambridge, 2001).

[71] J. B. Kruskal, Proc. Am. Math. Soc. **2**, (1956).

[72] A. Jacobs, *Great streets* (MIT Press, Boston, MA, USA, 1993).

[73] V. Latora P. Crucitti and S. Porta, Chaos **16**, (2006).

[74] V. Latora P. Crucitti and S. Porta, Phys. Rev. E **73**, (2006), physics/0504163.

[75] M. Southworth and E. Ben-Joseph, *Streets and the shaping of town and cities* (Island Press, Washington D.C., USA, 2003).

[76] B. Hillier and I. Shinichi, in *COSIT 2005, Conference On Spatial Information Theory* (Springer, Ellicottville, New York, USA, 2005).

[77] M. Sheffer U. Alon R. Milo S. Itzkovitz N. Kashtan R. Levitt S. Shen-Orr I. Ayzenshtat, Science **303**, 1538 (2004).

[78] S. Mangan and U. Alon, Proc. Natl. Acad. Sci. USA **100**, 11980 (2003).

[79] R. Yuster N. Alon and U. Zwick, Algorithmica **17**, (1997).

[80] A. Diaz-Guilera I. Vragovic, E. Louis, Phys. Rev. E **71**, (2005).

[81] J. Buhl J. Gautrais N. Reeves R.V. Solé S. Valverde P. Kuntz G. Theraulaz, Eur. Phys. J. B **49**, 513 (2006).

[82] J. Buhl K. Hicks E.R. Miller S. Persley O. Alinvi D.J. Sumpter, Behavioral Ecology and Sociobiology **63**, 451 (2009).

[83] A. Bavelas, Human Organization **7**, 16 (1948).

[84] J. Scott, *Social Network Analysis: A Handbook*, 2nd ed. (Sage, London (UK), 2000).

[85] S.A. Wagner D.A. Fell, Proc. R. Soc. Lond. B **268**, (2001).

[86] H. Jeong S.P. Mason A-L Barabási Z.N. Oltvai, Nature **411**, (2001).

[87] J. Nieminem, Scand. J Psychol. **15**, (1974).

[88] G. Sabidussi, Psychometrika 581 (1966).

[89] C. L. Freeman, Social Networks **1**, (1979).

[90] C.L. Freeman S.P. Borgatti D.R. White, Social Netw. **13**, 141 (1991).

[91] V. Latora M. Marchiori, New J. Phys. **9**, (2007).

[92] S. Porta V. Latora F. Wang S. Rueda B. Cormenzana F. Càrdenas L. Latora E. Strano E. Belli A. Cardillo S. Scellato, submitted for publication (unpublished).

[93] M. Barthelemy A. Flammini, Phys. Rev. Lett. **100**, (2008).

[94] J. Tang S. Scellato M. Musolesi C. Mascolo V. Latora, Phys. Rev. E **81**, (2010).

[95] C. L. Freeman, Sociometry **40**, (1977).

[96] M.G. Everett S.P. Borgatti, J. Math. Soc. **23**, 181 (1999).

[97] S. Porta P. Crucitti V. Latora, Urban Design International **13**, 41 (2008).

[98] CityForm – The Sustainable Urban Form Consortium.

[99] B. Hillier J. Hanson, *The social logic of space* (Cambridge University Press, Cambridge (UK), 1984).

[100] B. Hillier, *Space is the machine: a configurational theory of architecture* (Cambridge University Press, Cambridge (UK), 1996).

[101] *The hippocampal and parietal foundations of spatial cognition*, edited by N. Burgess K.J. Jeffery J. O'Keefe (Oxford University Press, Oxford (UK), 1999).

[102] Li Qi Racine S. Jeffrey, *Nonparametric Econometrics: Theory and Practice* (Princeton University Press, Princeton (USA), 2007).

[103] W. Tobler, Economic Geography **46**, 234 (1970).

[104] P. Jensen J. Boisson H. Larralde, Phys. A **351**, 551 (2005).

[105] Pablo Jensen, Phys. Rev. E **74**, (2006).

[106] A. Runions M. Fuhrer B. Lane P. Federl A-G Rolland-Lagan P. Prusinkiewicz, ACM Trans. Graph. **24**, 702 (2005).