On the Analysis of Network Reliability and Spatial Networks with Point Patterns

Pol Llagostera Blasco

Universitat de Lleida

Ph.D. THESIS

On the Analysis of Network Reliability and Spatial Networks with Point Patterns

Doctoral Program in Engineering and Information Technology

by

Pol Llagostera Blasco

Thesis Director
Carles Comas Rodríguez
Nacho López Lorenzo

Thesis Supervisor
Nacho López Lorenzo

University of Lleida

December, 2022

On the Analysis of Network Reliability and Spatial Networks with Point Patterns

by

Pol Llagostera Blasco

Submitted to the University of Lleida on December, 2022

Abstract

It's the year 1736, in the city of Königsberg in Prussia there are seven bridges that connect two portions of land through two large islands, and the Swiss mathematician Leonhard Euler is thinking about one question: How to walk through the city crossing all the bridges only once? Euler concluded that this journey cannot be made. To demonstrate this conclusion, he represented the problem by drawing a simple graph, which became the precursor of a new way of modeling the world, this was the beginning of Graph Theory.

Nowadays networks are present in many areas and in many aspects of everyday life, from road networks to chemical formulations. This trend has increased in recent years due to the emergence of the internet and digitization. These factors have generated a new flow of data that has reaffirmed the need to create network models to comprehend and solve problems. This need has led to the development of Graph Theory which has become an important research branch of mathematics with applications in computer science, biology, chemistry, social science, and geography among others. In some cases, not only the network structure can be of interest, but also we can be interested in events occurring along these linear configurations. For instance, in geography, networks containing events (typically point patterns) are modeled into spatial (and often planar) graphs. This representation is useful to describe and predict the future behavior of such occurrences. Another important research line in Graph Theory is the study of graph reliability. This is an expanding area where there are many unknowns, for instance, in the reliability of hamiltonian graphs.

The aim of this thesis is to give more light on both of these areas, i.e. graph reliability and point patterns on networks, by providing new methodologies and practical tools accessible to all researchers. Therefore, this thesis is structured into two main parts covering the areas of network reliability and spatial networks respectively. The first part presents research focused on the design of uniformly most reliable hamiltonian networks and a study to calculate the all-terminal reliability for undirected networks. The second part provides new developments in optimal path algorithms based on weighted graphs and a paper that presents a new package based on the ***R*** computational language focused on the visualization and statistical analysis of point processes occurring over spatial networks.

The author of the thesis acknowledges the support of the MCIN/AEI/ 10.13039/ 501100011033 (Spanish Government) through Grants MTM 2017-86767-R and PID 2020-115442RB-I00.

Resumen

Es el año 1736, en la ciudad de Königsberg en Prusia hay siete puentes que conectan dos porciones de tierra a través de dos grandes islas y el matemático suizo Leonhard Euler está pensando en una pregunta: ¿Cómo caminar por la ciudad cruzando todos los puentes sólo una vez? Euler concluyó que este viaje no puede realizarse. Para demostrar esta conclusión, representó el problema dibujando un gráfico sencillo, el cual se convirtió en el precursor de una nueva forma de modelar el mundo, este fue el inicio de la Teoría de los Grafos.

Actualmente, las redes están presentes en muchos ámbitos y aspectos de la vida cotidiana, desde redes de carreteras hasta formulaciones químicas. Esta tendencia ha aumentado en los últimos años debido a la aparición de internet y la digitalización, estos factores han generado un nuevo flujo de datos que ha reafirmado la necesidad de crear modelos de red para entender y resolver nuevos problemas. Esta necesidad ha desembocado en el desarrollo de la Teoría de Grafos que se ha convertido en una importante rama de investigación de las matemáticas con aplicaciones en informática, biología, química, ciencias sociales y geografía, entre otros. En algunos casos, no solo puede ser de interés la estructura de la red, sino que también podemos estar interesados en los eventos que ocurren a lo largo de estas configuraciones lineales. Por ejemplo, en geografía, las redes que contienen eventos (normalmente patrones puntuales) se moldean en gráficos espaciales (y a menudo planares). Esta representación es útil para describir y predecir el comportamiento futuro de estos hechos. Otra línea de investigación relevante en teoría de grafos es el estudio de la fiabilidad de los grafos. Esta es un área en expansión donde hay muchas incógnitas, por ejemplo, en la fiabilidad de los grafos hamiltonianos.

El objetivo de esta tesis es dar más luz a estos dos ámbitos, es decir, la fiabilidad de grafos y los patrones de puntos en las redes, aportando nuevas metodologías y herramientas prácticas accesibles a todos los investigadores. Por tanto, esta tesis se estructura en dos partes principales que cubren las áreas de fiabilidad de redes y redes espaciales respectivamente. La primera parte presenta una investigación centrada en el diseño de redes hamiltonianas uniformemente más fiables y un estudio para calcular la fiabilidad (*all-terminal*) para redes no dirigidas. La segunda parte ofrece nuevos desarrollos en algoritmos de caminos óptimos basados en grafos ponderados y un artículo que presenta un nuevo paquete basado en el lenguaje computacional centrado ***R*** en la visualización y análisis estadístico de procesos puntuales que se producen en redes espaciales.

El autor de la tesis agradece el apoyo del MCIN/AEI/10.13039/501100011033 (Gobierno de España) a través de las Subvenciones MTM 2017-86767-R y PID 2020-115442RB-I00.

Resum

És l'any 1736, a la ciutat de Königsberg a Prússia hi ha set ponts que connecten dues porcions de terra a través de dues grans illes i el matemàtic suís Leonhard Euler està pensant en una pregunta: Com caminar per la ciutat travessant tots els ponts només una vegada? Euler va concloure que aquest viatge no es pot fer. Per demostrar aquesta conclusió, va representar el problema dibuixant un gràfic senzill, el qual es va convertir en el precursor d'una nova manera de modelar el món, aquest va ser l'inici de la Teoria dels Grafs.

Actualment, les xarxes estan presents en molts àmbits i en molts aspectes de la vida quotidiana, des de xarxes de carreteres fins a formulacions químiques. Aquesta tendència ha augmentat en els últims anys a causa de l'aparició d'internet i de la digitalització, aquests factors han generat un nou flux de dades el qual ha reafirmat la necessitat de crear models de xarxa per entendre i resoldre nous problemes. Aquesta necessitat ha desembocat al desenvolupament de la Teoria de Grafs que s'ha convertit en una important branca de recerca de les matemàtiques amb aplicacions en informàtica, biologia, química, ciències socials i geografia, entre d'altres. En alguns casos, no només pot ser interessant l'estructura de la xarxa, sinó que també podem estar interessats en els esdeveniments que tenen lloc al llarg d'aquestes configuracions lineals. Per exemple, en geografia, les xarxes que contenen esdeveniments (normalment patrons puntuals) es modelen en gràfics espacials (i sovint planars). Aquesta representació és útil per descriure i predir el comportament futur d'aquests fets. Una altra línia de recerca rellevant en teoria de grafs és l'estudi de la fiabilitat dels grafs. Aquesta és una àrea en expansió on hi ha moltes incògnites, per exemple, en la fiabilitat dels grafs hamiltonians.

L'objectiu d'aquesta tesi és donar més llum en aquests dos àmbits, és a dir, la fiabilitat de grafs i els patrons de punts a les xarxes, aportant noves metodologies i eines pràctiques accessibles a tots els investigadors. Per tant, aquesta tesi s'estructura en dues parts principals que cobreixen les àrees de fiabilitat de xarxes i xarxes espacials respectivament. La primera part presenta una investigació centrada en el disseny de xarxes hamiltonianes uniformement més fiables i un estudi per calcular la fiabilitat (*all-terminal*) per a xarxes no dirigides. La segona part ofereix nous desenvolupaments en algorismes de camins òptims basats en grafs ponderats i un article que presenta un nou paquet basat en el llenguatge computacional ***R*** centrat en la visualització i anàlisi estadístic de processos puntuals que es produeixen en xarxes espacials.

L'autor de la tesi agraeix el suport del MCIN/AEI/ 10.13039/ 501100011033 (Gobiern d'Espanya) a través de les Subvencions MTM 2017-86767-R i PID 2020-115442RB-I00.

Acknowledgments

Four years ago I was introduced to the exciting (and sometimes exasperating) world of research. During this journey I have been surrounded by wonderful people who have inspired me, shared their knowledge and helped me when I needed it. This section is dedicated to them.

I would like to express my deepest gratitude to my directors and tutors, Dr. Carles Comas and Dr. Nacho López, who proposed me this adventure and which guidance has made this thesis possible. I would also want to thank the members of the department of Mathematics of the University of Lleida for the warm welcome to the department and which kindness made me feel like home. Special thanks to the secretary of the department Àngel Herrero who helped me navigate the tricky bureaucracy associated with the doctorate and who gave me a hand whenever I needed it.

Words cannot express my gratitude to my parents Joan and Eva, my sister Gisela and my partner Noe for supporting me and always being by my side. They have given me the courage to pursue my doctorate and have made the stressful moments more bearable. I am also grateful to my friends in special to Dr. Sergi Vila and Dr. Daniel Gibert for sharing their experiences, motivating me, and having so much-needed great times outside the research.

Finally, I had the pleasure of working with Dr. Matthias Eckardt who welcome me to his research group from the Humboldt University of Berlin which collaboration has generated a very interesting work.

Contents

Chapter 1

Introduction

1.1 The world as a network

The problem of the seven bridges of Königsberg was the first approach to model the world as a network. Since then many fields have adopted network modeling as an effective way to solve problems through Graph Theory. These fields are really diverse and we can see graph theory applications in main areas. For example, in Computer Science, see for instance, in communication networks, the internet of things (IoT), topological networks, and image analysis [28]. Also, we can find examples in biology sciences, see for instance, drug characterization, protein interaction, biological and metabolical networks. We can also find applications in chemistry sciences, see for instance, molecular connectivity, drug design, chemical formulation, and biomacromolecules study [19], or electrical engineering to study electrical network dynamics, analyze its properties and design new networks [14].

Since the apparition of the internet, available data has grown exponentially, from more sources, better detailed, and with great accessibility. This has promoted the apparition of new fields of research such as Data Mining [55] or the boost of other fields closely related to the availability of data such as Machine Learning [57]. Graph Theory has demonstrated to be a powerful tool for these areas too and boosted its usage since then. If we think about our daily lives we can see how graphs are used all the time, from searching on the internet to calculating the best travel to our destination. We can also note that the increase in available information provides better results, such as accurate search findings [13], optimized traffic route options based on road events (transit density, accidents, etc.) [56], or personalized advertising [29].

Therefore, having reliable graphs and better or new methodologies to analyze such graphs affects positively many areas at once.

In this thesis, we will take a deeper look into network reliability and spatial networks where we will detail how to construct reliable networks and calculate their reliability and, how random events occurring over a spatial plane interrelates with networks.

1.1.1 Network Characteristics

As described in Graph Theory, a network or graph is mathematically modeled by an ordered pair $G = (V, E)$, where V is a non-empty set of *nodes* (also called *vertices*) $\{v_1, v_2, \ldots, v_n\}$ which can be interconnected using a set $\{e_1, e_2, \ldots, e_m\} \in E$ of *links* (or *edges*). Moreover, we define a spatial point process X on a linear network as a stochastic mechanism that generates a countable and finite set of events $x_i = 1, \ldots, n$, where n is not fixed in advance over a linear network. In addition, depending on the characteristics of such links, the graphs are categorized into different types:

- Multigraph: A ***multigraph*** is a generalization of a graph that allows multiple edges, therefore two nodes may be connected through more than one edge. In some definitions, a multigraph also can include loops, that is, an edge that connects a vertice to itself.
- Digraph: A graph is called ***directed*** or ***digraph*** when its edges have a definite direction (usually indicated with an arrow). For instance, if the vertex v_1 is connected to the vertex v_2 with one directed edge from v_1 to v_2 $\{v_1 \rightarrow v_1\}$, then from v_2 its impossible to reach v_1 without any additional vertices or edges. Formally defined, a directed graph is a graph $G = (V, E)$ where E is a set of ordered pairs of elements of V. On the opposite, a graph $G = (V, E)$ is ***undirected*** when the set of edges E is composed of unordered pairs of vertices, therefore, the graph doesn't contain any directed edge.
- Mixed: If a graph contains directed and undirected edges, then it's considered a ***mixed graph*** (also known as partially directed graph).
- Weighted: When a graph contains attributes (or weights) embedded into its edges or vertices, then its named ***weighted*** graph. For instance, in a road network where its roads represent edges, can be useful to attach its length as edge weights.
- Induced sub-graph: An induced sub-graph of a graph is another graph formed by a sub-set of the vertices of the original graph and all the edges incident to pairs of vertices of said sub-set.

By looking at the nodes and edges, we can observe particularities and structures that can help to comprehend the graph. For instance, some of them are:

- Walk: A pair of nodes connected by a sequence of nodes and edges is called a ***walk***. In this sequence, both nodes and edges can be repeated.
- Path: If a pair of nodes are connected by a sequence of non-repeated edges and nodes, this sequence is named a ***path***. Note that a path is also a walk but a walk may not be a path.
- Cycle: It is a ***cycle*** a path where its starting node it's also its ending node.
- Distance: The number of edges of a shortest path connecting a pair of nodes is named ***distance***.
- Eccentricity: A node ***eccentricity*** is the maximum distance from itself to any other node in the graph.
- Radius: The ***radius*** of a graph is the minimum eccentricity of its nodes.

- Diameter: Among all the shortest paths between each pair of nodes in a graph, the longest is called the ***diameter*** of the graph.
- Neighborhood: The ***neighborhood*** of a node v is referred to as the induced sub-graph composed of all the adjacent nodes of v. In other words, it's the sub-graph formed by all the nodes connected via an edge to v.
- Degree: The ***degree*** is the number of edges that are incident to a node.

The form in which the nodes are interconnected can result in several graph configurations. Such configurations can contain characteristics that describe how the graph is structured. Below we define some of the most common:

- Connected: A graph is categorized as ***connected*** when each of its nodes can be reachable from any other node. That is, exists a path between each pair of vertices. On the other hand, a graph that doesn't fulfill the previous requisite is considered ***disconnected***, and its connected parts are called ***components***.
- Planar: A graph is ***planar*** if it can be 'drawn' in the plane so that none of its edges cross each other.
- Bipartite: If the nodes of a graph can be divided into two disjoint and independent sets U and V so that the edges cannot relate vertices of the same set, then, the graph is called **bipartite**. If each node in the set U connects to a node in the set V, then the graph is called **complete bipartite**.
- Complete: A graph is considered ***complete*** if each of its nodes is linked to all the remaining nodes, in other words, if each pair of vertices is connected with an edge.
- Regular: When all the vertices of a graph have the same number of neighbors, the graph is ***regular***. It can also be called k***-regular*** where k is the degree of the nodes. Regarding a directed graph, it is regular if all the nodes have the same number of in and out edges.
- Hamiltonian: In a connected graph, if there exists a path that traverses all the nodes only once, it's called a ***hamiltonian*** path. If said path ends in the same starting node, then it is a ***hamiltonian*** cycle. Therefore, a graph that contains such a cycle is considered ***hamiltonian***.
- Tree: In an undirected graph if for each pair of nodes only exists one path connecting them, then the graph is categorized as a ***tree***.

Note that a graph can be defined by a combination of several types and structures since they are not exclusive. For example, we can have a planar regular mixed graph with multiple directed weighted edges as shown in Fig. 1.1.

1.1.2 Network Representation

As described in Graph Theory, network representation is a procedure that allows saving the network in various forms. This procedure is useful to store the network in computer memory and ease its analysis and computations. Similarly to language,

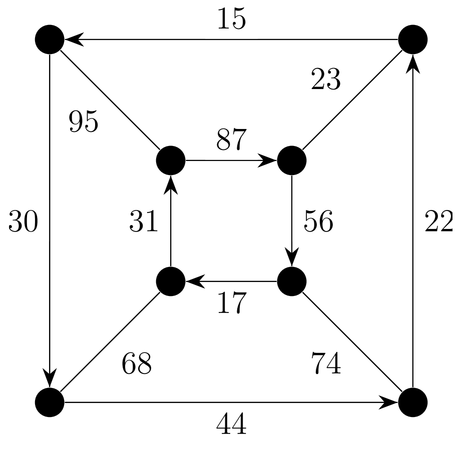


Figure 1.1: Example of a mixed graph that is planar, regular, and with weighted mixed edges (directed and undirected)

it is an encoding process that allows representing the same information in different ways. To this end, there are some points to take into consideration, for instance, the links or edges between each pair of nodes must be properly encoded as well as its direction and its weight (if exist). Some of the most common representations are in form of matrices which depending on the structure can be differentiated between ***adjacency*** and ***incidence*** matrices. There is another recent way to represent networks by using ASCII characters named ***Graph6***, although this encoding method is not intuitive it's notable for being memory-efficient.

Adjacency Matrix

As its name indicates, an adjacency matrix represents the adjacency between each pair of nodes, that is, the existence of an edge connecting these nodes. An ***adjacency matrix*** $A = (a_{ij})$ is composed by $n \times n$ elements where a value $a_{ij} = 1$ refers to the existence of an edge between the vertex i and the vertex j and $a_{ij} = 0$ otherwise. This type of representation can encode directed and undirected networks with or without weight. In the weighted case, instead of a value of $a_{ij} = 1$, the weight value is used. If the weight is $a_{ij} = 0$, this implies the nonexistence of an edge. When working with sparse graphs (with few edges), the adjacency matrix contains large quantities of 0 values, in this case, there are some techniques to reduce the space in memory designed to store the network, for instance, the usage of triples (Row, Column, value) to only store the non-zero values. To this end, there exists some packages that already provide this conversion, for example, the library ***Scipy*** for the ***Python*** computer language or the library ***Matrix*** for ***R***. Another method to optimize memory space is the usage of ***adjacency lists***. In this representation, each node of the network contains a list of its adjacent vertices. Although this method also allows checking whether two vertices are adjacent, it is slower than the adjacency matrix.

Incidence Matrix

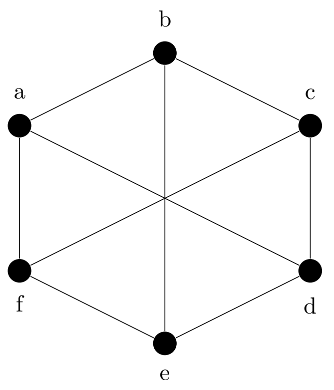
This matrix focuses on the edges associated with each node, in other words, the incident edges. An ***incident matrix*** $B = (b_{ik})$ is formed by $n \times m$ elements where the nodes are represented as rows and the edges as columns such that if e_k is an edge between the pair of nodes (i, j), then the elements b_{ik} and b_{jk} will contain the value 1 ($b_{ik} = b_{jk} = 1$) and 0 otherwise if the graph is undirected. In the case of directed graphs, the value can be 1 to represent an outgoing edge from the node, -1 for an incoming edge, or 0 if there is no edge. In terms of memory space, between incident or adjacent matrices, is better to use an incident matrix if the number of nodes is much larger than the number of edges.

Graph6

Presented by Brendan McKay, ***Graph6*** focuses on the representation of undirected graphs. It's a method that encodes the size and the upper part of the adjacency matrix in ASCII characters. To this end, this methodology first represents the graph in an adjacency matrix (which is symmetric), then, it creates a new matrix T with the upper triangle part of the adjacency matrix excluding its diagonal (since only contains zeroes). Next, it builds a bit vector of size $n(n-1)/2$ by traversing T row by row. Following, the vector is divided into chunks of 6 bits. These chunks are then converted to integers in the range of 63 to 126. Finally, each integer is transformed into its corresponding ASCII character. After the creation of this format, the author developed another graph representation for undirected sparse graphs named ***Sparce6*** and directed graphs with the name of ***Digraph6***.

Example

Graphical Network



Adjacency Matrix

$$\begin{array}{c c} & \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \end{array} & \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Incidence Matrix

$$\begin{array}{c c} \begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \end{array} & \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

Graph6

ElUg

Figure 1.2: Different representations of a network with 6 nodes and 8 edges

Chapter 2

Aim and Objectives of the Thesis

The world of networks as described in Graph theory have numerous branches and covers many areas, the aim of this thesis is to give more light on some of the less studied branches as well as to provide new methodologies and tools. To this end, the thesis is divided into two main parts. The first part is devoted to the study of network reliability, more specifically the all-terminal reliability polynomial. This polynomial gives the probability of the network remaining connected when all its edges have a certain fixed probability of failure. In this study, we take two approaches, the design of the uniformly most reliable graphs for certain families, and the efficient calculation of the reliability polynomial. All the reliability study has been implemented and tested in ***Python*** computer language in form of open-access repositories and libraries.

The second part focuses on the study of point processes occurring on spatial networks, for instance, traffic accidents in a road network, criminal activity in a city, or fires provoked by high-tension towers. All these events have a space-time dependence but the space where they occur is restricted to the roads, streets, or power lines. The presence of these events allows us to consider weighted networks, weights based on these events, and then analyze the resulting network structure accordingly. For instance, we studied how to optimally traverse a given network in terms of the characteristics of these events occuring on it, and how these events are related to the network structure and correlated with each other. This study has been implemented and tested in ***R*** the computer language, which can also be found in form of open-access repositories and libraries.

To summarize, the research objectives are divided into two:

All-terminal network reliability

- The design of uniformly most reliable graphs for certain families.
- The efficient calculation of the all-terminal reliability polynomial.
- The definition of new tools in form of ***Python*** repositories and libraries.

Occurrences in spatial networks

- The study of event-sensitive optimal paths that traverse networks.
- The definition of weighted network based on events occurring along networks.
- the definition of new tools in form of ***R*** repositories and libraries.

Chapter 3

Thesis Structure

This dissertation is structured in 7 chapters. Chapter 1 introduces network models as described in Graph Theory. It gives an overview of graph applications, characteristics, and representations. Chapter 2 details the aim and objectives of the thesis whereas chapter 3 presents its structure. The research articles are divided into two groups regarding the two main aims of this study, Network Reliability and, Spatial Networks with Point Processes. The first aim (Network Reliability) is represented in Chapter 4. The first part of this chapter introduces network reliability, gives an overview of its evolution from 1952 until recent years, and differentiates the two main approaches in network reliability. The first approach focuses on the design of optimal networks with high all-terminal reliability whereas the second approach focuses on the calculation of network reliability. Before each article, a short overview and the main takeaways of the article are given. The first research studies the design of Uniformly Most Reliable (UMR) hamiltonian networks and, the second research, provides tools and methodologies in form of a ***Python*** package to calculate the network all-terminal reliability. The second aim of this thesis (Spatial Networks with Point Processes) is presented in Chapter 5. Similarly to Chapter 4, this Chapter begins with an introduction to spatial points processes and gives an overview of the state of the art from 1995 to 2021. The following articles presented in this Chapter are also introduced with a brief overview and their main takeaways. The first research provides a methodology to calculate the safest path between any two points of the network based on weighted graphs. In particular, we focus our analysis on point patterns of wildlife-vehicle collisions, and other variables (the road type, speed limits, and vegetation density near the roads) to define the weight of each linear segment of the road network structure. To this end, the study tested the model using a real data set on a square area of 40×40 square km and the related road network structure around the city of Lleida, Spain. The second article presents an ***R*** package which provides tools and methodologies for undirected, directed, and mixed networks to manipulate network data, estimate intensities, compute local and global autocorrelation statistics, and data visualization. Chapter 6 presents other research activities such as conferences, posters, and, the abroad research stay. And finally, Chapter 7 presents a summary of all the research done in this thesis, the summary includes state of art, methodologies, and results. This chapter also discusses the development of the thesis, presents the limitations and difficulties, details the implications of the results, and finalizes with possible future research lines and upgrades, and some concluding remarks of this thesis.

Chapter 4

Network Reliability

4.1 Overview

The ability of a network $G = (V, E)$ to remain operational when some of its links are inoperative is named network reliability. A network is operational if there exists at least one walk between each pair of nodes. This is crucial in many areas, for instance, in road networks to maintain all areas connected, in distributed data systems to allow access to all the information, in electrical networks to provide electricity to all areas, or in supply chains to prevent a stock shortage. Our research is centered on the all-terminal network reliability problem. Given a probability p of an edge to be operational, this problem focuses on the probability of a network G to remain connected, in other words, that exist at least one operational path between all pairs of nodes. In our model, the network has perfect nodes, and the edges have the same independent probability $0 \leq p \leq 1$ to remain operational, moreover, no repair is allowed if an edge fails (see Fig. 4.1).

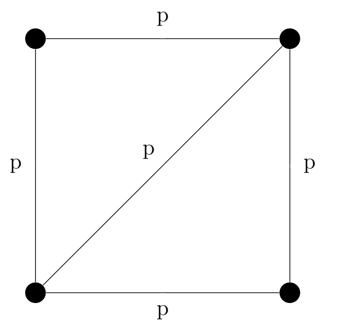


Figure 4.1: Network example with a reliability polynomial of: $p^5 + 5p^4(1-p) + 8p^3(1-p)^2$

4.1.1 State of Art

Although the problem of constructing reliable networks was presented by John von Neumann in 1952, the definition of network reliability as we understand nowadays was first coined in 1956 by E.F. Moore and C.E. Shannon in a paper called '*Reliable circuits using less reliable relays*' [31]. They proposed a probabilistic network model with perfect nodes and edges that could fail independently with a certain probability p. Since then, networks have become more present in our society and within many areas. This provided a fertile ground for research from which many studies have been done and with different variants of the original problem. For instance, one variant of the network reliability problem is to consider node failures in the model. This model is closer to real-life cases and was first presented in 1986 [17]. Other variants consider directed and mixed graphs. In general, there are three main problems related to network reliability: the two-terminal reliability problem, the all-terminal reliability problem, and the k-terminal reliability problem. Regarding the all-terminal problem for undirected networks (which this thesis focuses on), different methodologies had been purposed. For instance, the Inclusion-Exclusion principle and the Deletion-Contraction algorithm can calculate the all-terminal reliability polynomial while the Kruskal–Katona theorem [54] and the Edge-Packing methodology [12] obtain bounds for the reliability. Such principles and algorithms work with a wide range of network types. Still, there are other algorithms that calculate the reliability for specific graphs such as recursive methodologies for bipartite graphs [18], Series-Parallel reduction [47] or Monte Carlo algorithms [26] for Planar graphs, or Kruskal–Katona theorem for hypercubes and related networks [8].

Recently, part of the research is focused on; multivariate reliability polynomials where they consider two types of links with different failure probabilities [30], ternary networks where each component can be in one of three possible states [22], geographic networks which they consider dependent failures instead of independent [58] and, diameter-constrained reliability where the nodes have a probability of remain connected based on a path of length D or less [11].

To see a more detailed state of the art, the paper '*Sixty Years of Network Reliability*' from Hebert Pérez-Rosés [37] reviews the basic concepts of network reliability, provides results, analyses some recent developments and important research directions.

4.1.2 Two Approaches in Network Reliability

In the network reliability context, there are two main research lines; network design and, reliability calculation. The first approach is centered on the design of optimal networks with high reliability, for instance, the design of *uniformly most reliable* (UMR) networks. We say that a graph G is uniformly more reliable than a graph G' if for any probability p of an edge to be operational, G always has a greater probability to remain connected than G'. The second research line is focused on finding ways to efficiently calculate the reliability polynomial and reduce the computational time.

Next are presented two studies that approach both lines, the first is focused on the design of UMR networks related to the cake graph families whereas the second presents a new modular methodology to calculate the reliability polynomial.

4.2 Article: 'Network reliability in hamiltonian graphs'

This article has been published in the journal Discrete Optimization, belonging to the second quartile (Q2) in the area of applied mathematics as classified by Journal Citation Reports (JCR) and Scimago Journal Rank (SJR). See Table 4.1.

Table 4.1: Journal metrics corresponding to the journal Discrete Optimization for the year 2021

Journal Metric	Value
Citescore	1.9
Impact Factor (JCR)	1.509
SJR	0.646
SNIP	1.22

The purpose of the article *'Network reliability in hamiltonian graphs'* is to study the design of uniformly-most reliable (UMR) network configurations belonging to the hamiltonian family. The research focuses on one particular hamiltonian family named Cake networks. Such networks can be thought of as cycle networks with diametral chords that traverse the cycle and cross each other like cuts in a cake. With this configuration, the study characterizes the construction of UMR networks for n nodes and $m = n + 1$, $m = n + 2$, and provides a computational approach for $m = n + 3$ edges. The research also demonstrates the non-existence of UMR hamiltonian networks for some configurations. To summarize, the main points of this article are:

- The characterization of UMR hamiltonian graphs for $m = n+1$ and $m = n+2$ edges.
- Computational approach for the hamiltonian case with $m = n + 3$ edges.
- Analysis of Cake graphs and comparison with the UMR hamiltonian graphs.
- Demonstration of non existence of UMR hamiltonian graphs for $m = \binom{n}{2} - \frac{n+2}{2}$ for all $n \geq 6$ even and, $m = \binom{n}{2} - \frac{n+5}{2}$ for all $n \geq 7$ odd.

Discrete Optimization 41 (2021) 100645

Contents lists available at ScienceDirect

Discrete Optimization

www.elsevier.com/locate/disopt

Network reliability in hamiltonian graphs

Pol Llagostera [*], Nacho López, Carles Comas

Dep. of Mathematics, University of Lleida, Lleida, Spain

A R T I C L E   I N F O

Article history:
Received 24 April 2020
Received in revised form 23 April 2021
Accepted 23 April 2021
Available online xxxx

Keywords:
Network reliability
Network design
Reliability polynomial
Hamiltonian graph

A B S T R A C T

The reliability polynomial of a graph gives the probability that a graph remains operational when all its edges could fail independently with a certain fixed probability. In general, the problem of finding uniformly most reliable graphs inside a family of graphs, that is, one graph whose reliability is at least as large as any other graph inside the family, is very difficult. In this paper, we study this problem in the family of graphs containing a hamiltonian cycle.

1. Introduction

Notation and terminology

In the reliability context, networks are modeled by graphs. We recall that a graph is an ordered pair $G = (V, E)$, where V is a non empty set of *vertices* or *vertices*, and E is a set of unordered pairs of different elements of V, called *links* or *edges*. The degree of a vertex $v \in V$, denoted by $d(v)$, is the number of incident edges at v. A *walk* of length $\ell \geq 0$ from a vertex u to a vertex v is a sequence of $\ell+1$ vertices, $u_0u_1 \ldots u_{\ell-1}u_\ell$, such that $u = u_0$, $v = u_\ell$ and each pair $u_{i-1}u_i$, for $i = 1, \ldots, \ell$, is an edge of G. A *connected* graph has always a walk between any pair of vertices. Otherwise, the graph is *not connected*. Our model is a stochastic network with perfect vertices but with edge failures: each edge remains operational independently with probability $0 \leq p \leq 1$ (every edge has the same probability of being operational). Moreover, no repair is allowed after an edge fails.

Through this paper, we consider the problem of computing the probability that a network remains connected. More precisely, we focus on what is called, the *all-terminal network reliability problem*: given the probability p of an edge being operational in a network G, what is the probability that there exists an operational path between every pair of vertices u and v of G (see [1]).

[*] Corresponding author.
E-mail addresses: pol.llagostera@udl.cat (P. Llagostera), nacho.lopez@udl.cat (N. López), carles.comas@udl.cat (C. Comas).

https://doi.org/10.1016/j.disopt.2021.100645
1572-5286/© 2021 Elsevier B.V. All rights reserved.

Hamiltonian graphs

A graph G is a *hamiltonian graph* if it contains a spanning cycle, that is, a cycle passing through all the vertices of the graph. This cycle is called a *hamiltonian cycle*. The problem of finding a hamiltonian cycle takes back to the 1850s when Sir William Rowan Hamilton presented the problem of finding a hamiltonian cycle in a dodecahedron. The complexity of finding a hamiltonian cycle in a graph is in general NP-complete and much research in this area is devoted to finding necessary and/or sufficient conditions for a graph to be hamiltonian. One of such conditions is Ore's Theorem, which is a well-known result in the field.

Theorem 1.1 (*Ore*)**.** *Let G be a connected graph of order n such that $d(u) + d(v) \geq n$ for any two pair of non adjacent vertices u and v. Then G is a hamiltonian graph.*

Recent developments and many results regarding hamiltonian graphs can be found in [2].

The reliability polynomial

Let $|G|$ denote the number of edges of a network $G = (V, E)$. Let us consider the set $\mathcal{G}$ of connected spanning subgraphs of G. Then, the probability that G is connected, as a function of p, is

$$\sum_{G' \in \mathcal{G}} p^{|G'|}(1-p)^{m-|G'|} \qquad (1)$$

This formula is known as *the reliability polynomial of* G, and it is denoted as $\mathrm{Rel}(G,p)$. There are several methods for computing $\mathrm{Rel}(G,p)$, but in general this problem is NP-complete (see [3]). A *pathset* of a graph $G = (V, E)$ is a subset $N \subseteq E$ of edges that makes the graph (V, N) connected. Hence, an alternative definition for the reliability polynomial is,

$$\mathrm{Rel}(G,p) = \sum_{i=0}^{m} N_i p^i (1-p)^{m-i} \qquad (2)$$

where N_i denotes the number of pathsets of cardinality i. From this point of view, some of the coefficients of the reliability polynomial are 'easy' to compute. For instance, $N_i = 0$ for all $i < m-n+1$. Also $N_i = \binom{m}{m-i}$ for all $i > m - \lambda$, where λ denotes the edge-connectivity of G. The edge-connectivity can be found with a network flow algorithm in polynomial time (see [4]). Moreover, any spanning tree of G contain $m - n + 1$ edges, so $N_{m-n+1} = \tau$ where τ is the number of spanning trees of G (also known as the tree-number). Again, τ is computed in polynomial time (see [5]) using the Kirchoff's matrix tree theorem. Further, from [1], section 5.2, $N_{m-\lambda}$ can also be computed in polynomial time.

Uniformly most reliable graphs

A main problem in the reliability context is concerned with the design of networks with 'high' reliability. To this end, let $\mathcal{G}(n,m)$ be the set of all simple connected graphs with n vertices and m edges. Given two graphs $G, G' \in \mathcal{G}(n,m)$, we say that G is *uniformly more reliable* than G' if $\mathrm{Rel}(G,p) \geq \mathrm{Rel}(G',p)$ for all $p \in [0,1]$. This means that, for any value of an edge to being operational p, graph G has equal or higher probability to remain connected than graph G'. If there exists a graph G such that $\mathrm{Rel}(G,p) \geq \mathrm{Rel}(H,p)$ for all $p \in [0,1]$ and for all $H \in \mathcal{G}(n,m)$, then G is known as a *uniformly most reliable* graph in the set $\mathcal{G}(n,m)$. We point out that uniformly most reliable graphs have been also known as *uniformly optimally reliable* graphs (see [6–9]) and *uniformly optimal* digraphs for the directed case (see [10]).

However, the reliability polynomial does not define a total ordering in $\mathcal{G}(n,m)$, because there are graphs whose corresponding reliability polynomial have a crossing point in the interval $(0,1)$ (see [1] p.48). In order to prove that a graph is uniformly most reliable graph, the following observation is widely used.

Table 1
Uniformly most reliable graphs known in $\mathcal{G}(n,m)$, where n denotes the number of vertices and m the number of edges, for $m \leq n+3$ and other sporadic values.

(n,m)	UMR graphs for $m \leq n+3$ and other sporadic values	Reference
$(n,n-1)$	UMR$(n,n-1)$ (Tree graphs T_n)	
(n,n)	UMR(n,n) (Cycle graphs C_n)	
$(n,n+1)$	UMR$(n,n+1)$ (θ-graphs)	[11]
$(n,n+2)$	UMR$(n,n+2)$ (subdivision of K_4)	[14]
$(n,n+3)$	UMR$(n,n+3)$ (subdivision of $K_{3,3}$)	[15]
$(8,12)$	Wagner graph	[12]
$(10,15)$	Petersen graph	[16]
$(12,18)$	Yutsis 18j-symbol label F graph	[17]

Observation 1.1. *Let G, G' be graphs such that $N_i(G) \leq N_i(G')$, for all $0 \leq i \leq m$. Then Rel$(G,p) \leq$ Rel(G',p) for all $p \in [0,1]$.*

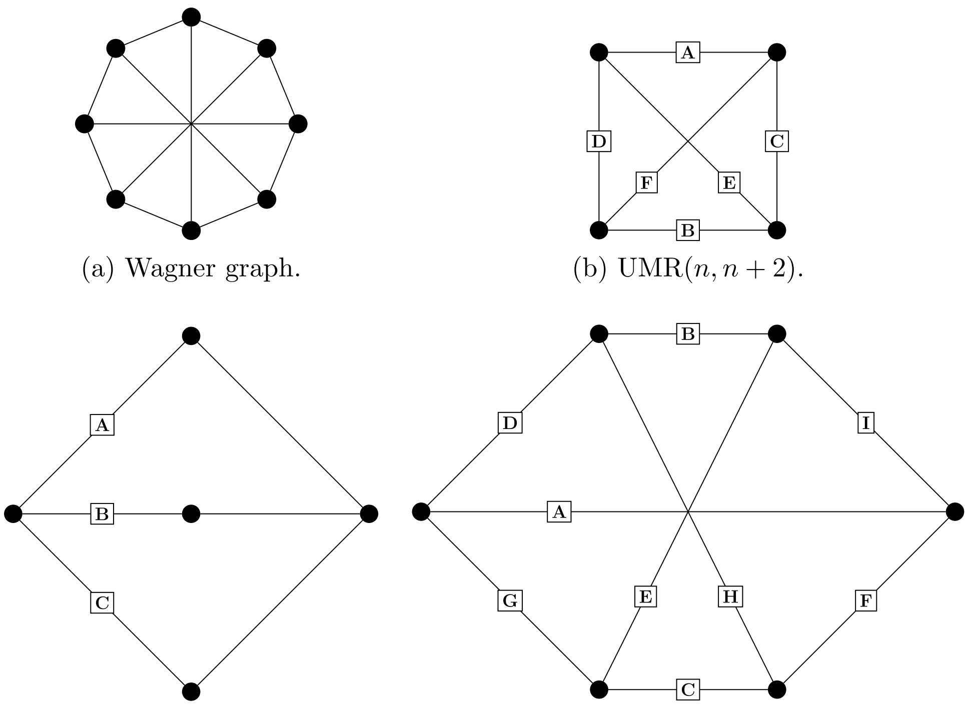
Hence, maximizing the number of pathsets N_i has been a classical method to obtain uniformly most reliable graphs. We point out that the converse assertion of Observation 1.1 is not known to be true.

There exist uniformly most reliable graphs for $m \leq n+3$, $m \geq \binom{n}{2} - n$, and other sporadic values (see Table 1). The uniformly most reliable graphs for $m \leq n+3$ are *subdivisions* of certain small graphs. The *subdivision* operation for an edge uv is the deletion of uv from the graph and the addition of two edges uw and wv along with the new vertex w. A graph which has been derived from G by a sequence of subdivision operations is called a *subdivision* of G. For instance, every uniformly most reliable graph in the case $m = n+1$ $(n \geq 5)$ is a particular *θ-graph*, constructed taking a graph of order 5 as a basis and subdividing edges and adding vertices one by one by following the sequence $A, B, C, A, B, C, \ldots$ (see Fig. 1). The characteristics of this construction are that the resulting graphs have nearly or equal path lengths. This graph is also called the *Monma graph* [11]. Uniformly most reliable graphs can be also obtained as subdivisions of small graphs for $m = n+2$ and $m = n+3$. We will denote these extremal graphs as UMR(n,m). It has been conjectured that UMR$(n,n+4)$ are particular subdivisions of the Wagner graph for $n \geq 8$ (see [12]). Despite these general constructions, uniformly most reliable graphs have been found only for $(n,m) = (8,12)$ (Wagner), $(10,15)$ (Petersen) and $(n,m) = (12,18)$ (Yutsis 18j-symbol label F). Besides, candidates for uniformly most reliable graphs have been found using heuristics in [13] for even orders between 14 and 20.

Besides, a graph G with $m \geq \binom{n}{2} - \frac{n}{2}$ edges is uniformly most reliable if its complement graph has a matching (see [18]). The remaining values in the range $\binom{n}{2} - n \leq m < \binom{n}{2} - \frac{n}{2}$ have also uniformly most reliable graphs characterized by their corresponding complementary graph (see [19,20]). On the other hand, there are infinitely many values of n and m where uniformly most reliable graphs do not exist (see [21,22]).

Main contributions of the paper

We study uniformly most optimal graphs inside the family of hamiltonian graphs with n vertices and m edges. First, in Section 2, we see that most uniformly most optimal graphs are not hamiltonian, so we perform a deep study in cases $m \leq n+2$. To this end, we compute the reliability polynomial in terms of what we call the chord type graph and the chord-path lengths vector, obtaining the reliability polynomial of any hamiltonian graph. We present an algorithmic approach for the construction of a family of graphs $FCG_{n,m}$, based in a 'fair cutting cake process', that provides uniformly most reliable hamiltonian graphs in some cases. In Section 2.3 we take advantage of the computational tools developed for the construction of $FCG_{n,m}$ graphs to give some light for the case $m = n+3$ and beyond. In particular, a modified version of the Factoring theorem has been developed and therefore we have been able to compute every uniformly most reliable hamiltonian graph with diametrical chords for all values of $n \leq 34$. We also obtain a few uniformly most reliable hamiltonian graphs for $m = n+4$ and $m = n+5$. Section 2.4 is devoted to prove the non-existence of uniformly most reliable hamiltonian graphs for infinitely many values of n and m.



(a) Wagner graph. (b) UMR$(n, n+2)$.

(c) UMR$(n, n+1)$. (d) UMR$(n, n+3)$.

Fig. 1. Uniformly most reliable graphs.

2. Uniformly most reliable hamiltonian graphs

Interconnection networks are usually modeled by graphs. In this context, the design of optimal topologies is based on finding graphs satisfying some requirements of their properties. The study of connectivity, fault tolerance and/or reliability is of utmost importance in this area (see [23]). Also hamiltonicity is a common requirement in the design of some network topologies (also called 'ring embedding' problem, see [24]). However the study of uniformly most reliable hamiltonian graphs is a challenging problem in designing suitable topologies under certain requirements.

Uniformly most reliable graphs have been classically studied on the set $\mathcal{G}(n, m)$ of (simple) graphs on n vertices and m edges. A restricted version of the problem is the study of these extremal graphs inside the family of hamiltonian graphs. Let us denote as $\mathcal{H}(n, m)$ the set of (non-isomorphic) hamiltonian graphs with n vertices and m edges. Of course, $\mathcal{H}(n, m)$ is a subset of $\mathcal{G}(n, m)$ and since every hamiltonian graph on n vertices and m edges contains a spanning cycle, then we may assume that $m \geq n$. If a particular graph G is uniformly most reliable in $\mathcal{G}(n, m)$ and it is in addition hamiltonian, then G is also uniformly most reliable in $\mathcal{H}(n, m)$. This happens trivially in case (n, n) for all $n \geq 3$, where C_n are uniformly most reliable graphs, and also in $(8, 12)$ and $(12, 18)$, where the Wagner and Yutsis graphs are uniformly most reliable graphs, respectively, and also hamiltonian. It is well known that the Petersen graph is not hamiltonian (see [2]) and in UMR$(n, m \leq n+3)$, when n is large enough, uniformly most reliable graphs are not hamiltonian, as next result shows.

Proposition 2.1. *The following uniformly most reliable graphs in $\mathcal{G}(n, m)$,*

(a) UMR$(n, n+1)$ *is hamiltonian if and only if* $n = 4$;

(b) $\mathrm{UMR}(n, n+2)$ *is hamiltonian if and only if* $n \leq 8$*;*
(c) $\mathrm{UMR}(n, n+3)$ *is hamiltonian if and only if* $n \leq 13$*;*

Proof. For any graph G containing a vertex v of degree $d(v) \geq 3$ and such that there are at least three different neighbors w_1, w_2, w_3 of v such that $d(w_1) = d(w_2) = d(w_3) = 2$, then G is not hamiltonian. Indeed, any vertex of G uses exactly two edges in any hamiltonian cycle, hence the only two incident edges of w_1, w_2 and w_3 belong to any hamiltonian cycle. But one of these two pairs of edges is incident also to v, and hence v would use three edges in any hamiltonian cycle, which is impossible. The graph $\mathrm{UMR}(n, n+1)$ $(n \geq 5)$ given in Fig. 1 has a vertex v of degree 3 and that all its neighbors have degree 2. Hence it is not hamiltonian. Besides $\mathrm{UMR}(4, 5)$ exists and is $K_4 - e$ (a complete graph of order 4 where an edge has been removed) which is a hamiltonian graph. It is easy to find a hamiltonian cycle in $\mathrm{UMR}(n, n+2)$ for $4 \leq n \leq 8$, when the four subdivisions A, B, C, D are performed at most. But for $n \geq 9$ there exists a vertex of degree 3, such that all its neighbors have degree 2, and hence it is no longer hamiltonian. The argument for case (c) is similar: the sequence of edges (subdivided or not) A, D, B, E, C, F gives a hamiltonian cycle for every $n \leq 12$ and the next subdivision (edge G) produces a vertex of degree 3 such that all its neighbors have degree 2. □

So the natural question about which hamiltonian graphs are uniformly most reliable (whenever they do exist) arises. We partially answer this question in the following sections.

2.1. The case $m = n + 1$

Let us consider G as a hamiltonian graph with n vertices $\{0, 1, \dots, n-1\}$ and $m = n + 1$ edges where the hamiltonian cycle is given by the sequence $0, 1, \dots, n-1, 0$. Without loss of generality, we assume that the remaining edge of the graph joins vertex 0 and vertex x_1. We may consider that $2 \leq x_1 \leq \lfloor \frac{n}{2} \rfloor$ (the remaining cases $\lfloor \frac{n}{2} \rfloor < x_1 < n - 2$ produce an isomorphic graph taking $x_1' = n - x_1$). Then, the reliability polynomial of G is given by

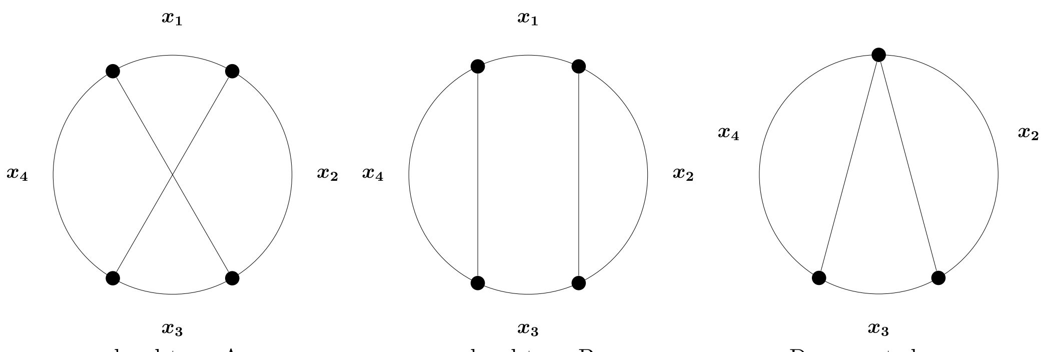
$$\mathrm{Rel}(G, p) = p^m + mp^{m-1}(1-p) + \tau p^{m-2}(1-p)^2, \tag{3}$$

where $\tau = n + x_1(n - x_1)$. Indeed, the removal of one edge or less guarantee the connectivity of the graph (the edge-connectivity of G is precisely $\lambda = 2$ for $n \geq 4$). Besides, deleting any set of three edges or more disconnects the graph ($N_i = 0$ for all $i < m - 2$). Hence, just the coefficient N_{m-2} is unknown. This is precisely the tree number τ. The number of spanning trees of G is $n + x_1(n - x_1)$ since we have two cases: if edge $(0, x_1)$ is removed, then we can remove any of the n remaining edges. Otherwise we may delete two edges belonging to the hamiltonian cycle. In order to guarantee the connection of the resulting graph, we must choose one edge from the cycle $0, 1, \dots, x_1, 0$ and the other from the cycle $0, x_1, x_1 + 1, \dots, n, 0$. There are x_1 edges in one cycle and $n - x_1$ in the other. This gives the total number $n + x_1(n - x_1)$.

Proposition 2.2. *The set of hamiltonian graphs* $\mathcal{H}(n, n+1)$*,* $n \geq 4$*, is totally ordered by the reliability polynomial and the uniformly most reliable graph is given by joining any of two vertices of* C_n *at maximum distance.*

Proof. For any graph G in $\mathcal{H}(n, n+1)$, there exist x_1, with $2 \leq x_1 \leq \lfloor \frac{n}{2} \rfloor$ such that G is isomorphic to a graph with hamiltonian cycle $0, 1, \dots, n-1, 0$ plus an edge $(0, x_1)$. According to (3), for any p, the value of $\tau(G) = -x_1^2 + nx_1 + n$ attains the maximum at $x_1 = \lfloor \frac{n}{2} \rfloor$. This graph G is isomorphic to the graph constructed from C_n by joining two vertices at maximum distance. □

Although the most reliable graph constructed from C_n and adding one single edge was previously known (see [12]) it is good to notice that $\mathcal{H}(n, n+1)$ is totally ordered by the reliability polynomial for all $n \geq 4$, since in general, this is not true for $m \geq n + 2$ where reliability polynomials can cross for $p \in (0, 1)$.



Degenerated case

Fig. 2. Different chord-types of hamiltonian graphs with two chords. The degenerated case is a chord-type A (or B) graph with $x_1 = 0$.

2.2. The case $m = n + 2$

Here we will present uniformly most reliable graphs in $\mathcal{H}(n, n+2)$. The reliability polynomial for any $G \in \mathcal{H}(n, n+2)$ is,

$$\mathrm{Rel}(G,p) = p^m + mp^{m-1}(1-p) + N_{m-2}p^{m-2}(1-p)^2 + \tau p^{m-3}(1-p)^3. \tag{4}$$

Any hamiltonian graph G of order n and size $n+2$ can be graphically depicted in a circular embedding where every vertex of G is in a hamiltonian cycle C of G together with 2 more edges (*chords*) through the hamiltonian cycle C. Then, the hamiltonian cycle is split by the chords in four paths of lengths x_1, x_2, x_3 and x_4, where $x_1 + x_2 + x_3 + x_4 = n$ and $x_i \in \mathbb{Z}^+$, for all $1 \le i \le 4$. In fact, we have two more situations regarding the relative position between the chords (see Fig. 2). When $x_i = 0$ for an specific $1 \le i \le 4$ then we fall in what we call a *degenerated* case. However, for any chord-type graph $G \in \mathcal{H}(n, n+2)$ we associate its corresponding vector (x_1, x_2, x_3, x_4). We will refer this vector as the vector of *chord-path lengths.* In the other way around, given any positive integer vector (x_1, x_2, x_3, x_4), such that $x_1 + x_2 + x_3 + x_4 = n$, we can construct any graph G of $\mathcal{H}(n, n+2)$ of any type. Moreover, since any cyclic permutation of (x_1, x_2, x_3, x_4) produces an isomorphic graph, we can just consider those vectors modulo cyclic permutations.

Next we will compute coefficients N_{m-2} and τ in Eq. (4) depending on (x_1, x_2, x_3, x_4).

Proposition 2.3. *Let G be a chord-type A graph in $\mathcal{H}(n, n+2)$ with vector of chord-path lengths (x_1, x_2, x_3, x_4). Then,*

$$\begin{aligned} N_{m-2} &= 1 + 2n + \sum_{1 \le i < j \le 4} x_i x_j. \\ \tau &= n + (x_1 + x_2)(x_3 + x_4) + (x_1 + x_4)(x_2 + x_3) + \sum_{1 \le i < j < k \le 4} x_i x_j x_k. \end{aligned} \tag{5}$$

Proof. The coefficient N_{m-2} counts the number of connected graphs after the deletion of two edges of G. If these two edges are the chords, then we have just one graph (the hamiltonian cycle). Besides, if the removed edges are one chord and one edge of the cycle, then, we can choose between 2 chords and n edges of the hamiltonian cycle. This gives $2n$ graphs. Finally, if the removed edges belong to the hamiltonian cycle, then both edges must be from different paths defined by the chords. This gives the number $\sum_{1 \le i < j \le 4} x_i x_j$. The computation of τ is similar, we need to remove three edges of G in order to obtain a spanning tree of G.

We proceed depending on which edges we are dealing with:

1. *Two chords plus one edge of the cycle*: Any edge of the cycle can be removed after the removal of the two chords, so we count n spanning trees of this type.
2. *One chord plus two edges of the cycle*: After the removal of one chord, say c_1, we must remove one edge of each path defined by the other chord. Since these paths have lengths x_1+x_2 and x_3+x_4, the number of spanning trees is $(x_1+x_2)(x_3+x_4)$. The same applies for the other chord, say c_2, where now the paths lengths are x_1+x_4 and x_2+x_3.
3. *Three edges of the cycle*: The graph remain connected after the deletion of three edges of the cycle only if these three edges belong to different paths defined by the chords. Hence the number of spanning trees in this case is given by the all different three products of x_1, x_2, x_3, x_4. □

Chord-type B graphs are not interesting for the study of uniformly most-reliable hamiltonian graphs, as next result states:

Proposition 2.4. *Let (x_1, x_2, x_3, x_4) be a chord-path lengths vector of G_A and G_B, which are chord-type graphs of types A and B, respectively. Then, $\tau(G_A) > \tau(G_B)$ and $N_{m-2}(G_A) \geq N_{m-2}(G_B)$.*

Proof. Following the ideas behind the proof of Proposition 2.3 one can see that

$$\begin{aligned} N_{m-2}(G_B) &= 1 + 2n + \Big(\sum_{1\leq i<j\leq 4} x_i x_j\Big) - x_1x_3, \\ \tau(G_B) &= n + (x_1+x_2+x_4)x_3 + (x_2+x_4+x_3)x_1 + \sum_{1\leq i<j<k\leq 4} x_ix_jx_k - (x_1x_2x_3 + x_1x_3x_4). \end{aligned} \tag{6}$$

A simple comparison of both formulas with the ones given in Proposition 2.3 gives the desired result,

$$\begin{aligned} &N_{m-2}(G_A) = N_{m-2}(G_B) + x_1x_3 \geq N_{m-2}(G_B) \text{ and} \\ &\tau(G_A) = \tau(G_B) + x_1x_3x_4 + (x_1x_2 + 2x_1)x_3 > \tau(G_B). \end{aligned} \qquad \square$$

Notice that Eqs. (5) and (6) coincide when $x_1 = 0$, since they represent the same 'degenerated' graph (the one depicted in Fig. 2). In order to obtain those integer vectors (x_1, x_2, x_3, x_4) such that produce uniformly most-reliable hamiltonian graphs, we should maximize both N_{m-2} and τ in Eq. (4). To this end, let us consider the set of chord-path length vectors $X_n = \{(x_1, x_2, x_3, x_4) \in \mathbb{Z}_+^4 \,|\, x_1+x_2+x_3+x_4 = n\}$ and the functions

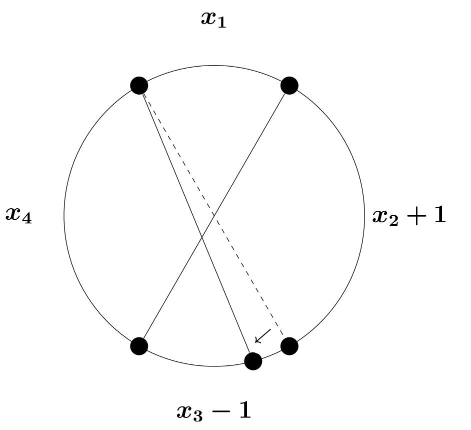
$$\begin{array}{llcl} f\colon & X_n & \longrightarrow & \mathbb{Z} \\ & (x_1,x_2,x_3,x_4) & \longmapsto & \tau \end{array} \quad \text{and} \quad \begin{array}{llcl} g\colon & X_n & \longrightarrow & \mathbb{Z} \\ & (x_1,x_2,x_3,x_4) & \longmapsto & N_{m-2} \end{array} \tag{7}$$

where τ and N_{m-2} are given in Prop. 2.3. Let $n = 4k + \alpha$, $\alpha \in \{0, 1, 2, 3\}$. For any $(x_1, x_2, x_3, x_4) \in X_n$ we define

$$D(x_1, x_2, x_3, x_4) = \sum_{i=1}^{4} |x_i - k|$$

as a measure of closeness to the constant vector (k, k, k, k). Then the elements of X_n can be measured according to its closeness to this constant vector. For instance, when $\alpha = 1$, there is just one vector in X_n with $D = 1$, which is $(k+1, k, k, k)$ (any cyclic permutation of this vector of chord-path lengths gives an isomorphic graph, so we do not take them into account). Notice that D must be odd when $\alpha = 1$, so next D is 3 and the set of vectors of chord-path lengths with $D = 3$ is $\{(k+1, k+1, k-1, k), (k, k+2, k-1, k), (k, k+2, k, k-1)\}$.

We also define the following graphical operators in X_n: given a graph with vector of chord-path lengths $(x_1, x_2, x_3, x_4) \in X_n$, the operator σ moves just one vertex of a chord in such a way that two contiguous path lengths are modified by one unit each (see Fig. 3), that is, $\sigma(x_1, x_2, x_3, x_4) = (x_1, x_2+1, x_3-1, x_4)$. Besides the operator r defined as $r(x_1, x_2, x_3, x_4) = (x_2, x_3, x_4, x_1)$ is simply a rotation of the graph. For instance, in



$\sigma(x_1, x_2, x_3, x_4) = (x_1, x_2 + 1, x_3 - 1, x_4)$

Fig. 3. Graphical representation of operator σ.

the case $\alpha = 1$, starting from the single vector with $D = 1$ one can obtain all the vectors of chord-path lengths for $D = 3$: $\sigma(k+1, k, k, k) = (k+1, k+1, k-1, k)$, $\sigma \circ r(k+1, k, k, k) = \sigma(k, k+1, k, k) = (k, k+2, k-1, k)$ and $r^{-1} \circ (\sigma \circ r)^2(k+1, k, k, k) = (k, k+2, k, k-1)$.

Clearly, functions f, g and D are invariants under operator r. This is no longer true for operator σ. In contrast, we have the following result.

Lemma 2.1. *Let* $\mathbf{x} = (x_1, x_2, x_3, x_4) \in X_n$ *and* $\mathbf{y} = (y_1, y_2, y_3, y_4) \in X_n$ *such that* $D(\mathbf{y}) = D(\mathbf{x}) + 2$ *and* $\mathbf{y} = \sigma(\mathbf{x})$. *Then* $f(\mathbf{x}) \geq f(\mathbf{y})$ *and* $g(\mathbf{x}) \geq g(\mathbf{y})$.

Proof. Suppose that $\mathbf{y} = \sigma(\mathbf{x})$, that is, $y_1 = x_1$, $y_2 = x_2+1$, $y_3 = x_3-1$ and $y_4 = x_4$. From $D(\mathbf{y}) = D(\mathbf{x})+2$ we have that

$$|(x_2 + 1) - k| - |x_2 - k| + |(x_3 - 1) - k| - |x_3 - k| = 2$$

which is equivalent to

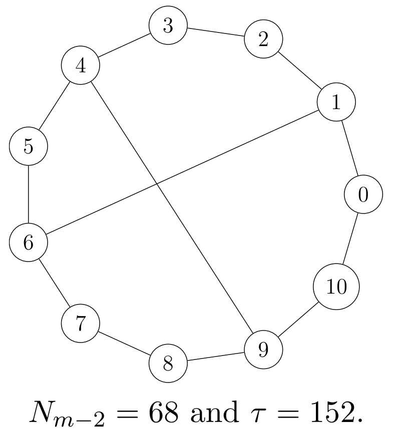
$$\big(|(x_2 - k) + 1| - |x_2 - k|\big) + \big(|(x_3 - k) - 1| - |x_3 - k|\big) = 2. \tag{8}$$

Taking into account that $|a + 1| - |a| \leq 1$ and $|b - 1| - |b| \leq 1$ and both equalities hold if and only if $a \geq 0$ and $b \leq 0$, respectively. Then Eq. (8) is equivalent to $x_2 \geq k$ and $x_3 \leq k$. Besides, from Eq. (5) we have $f(\mathbf{x}) - f(\mathbf{y}) = (x_2 - x_3)(x_1 + x_4 + 1)$. From $x_2 \geq k$ and $x_3 \leq k$ we have $(x_2 - x_3) \geq 0$ and hence $f(\mathbf{x}) - f(\mathbf{y}) = (x_2 - x_3)(x_1 + x_4 + 1) \geq 0$ since both factors are positive. Besides, again from Eq. (5), $g(\mathbf{x}) - g(\mathbf{y}) = x_2 - x_3 + 1$ which is also positive since $(x_2 - x_3) \geq 0$. $\square$

Theorem 2.1. *A uniformly most reliable graph* G *exists in* $\mathcal{H}(n, n+2)$. *Moreover,* G *is a chord-type A graph with vector* (x_1, x_2, x_3, x_4) *of chord-path lengths equal to*

- (k, k, k, k) *if* $n = 4k$ *for some positive integer* k,
- $(k+1, k, k, k)$ *if* $n = 4k+1$ *for some positive integer* k,
- $(k+1, k, k+1, k)$ *if* $n = 4k+2$ *for some positive integer* k,
- $(k+1, k+1, k+1, k)$ *if* $n = 4k+3$ *for some positive integer* k.

Proof. By Proposition 2.4 we have to take into account only graphs of type A. In order to maximize the coefficients of the reliability polynomial, let us consider the discrete functions $f : X_n \to \mathbb{Z}$ and $g : X_n \to \mathbb{Z}$



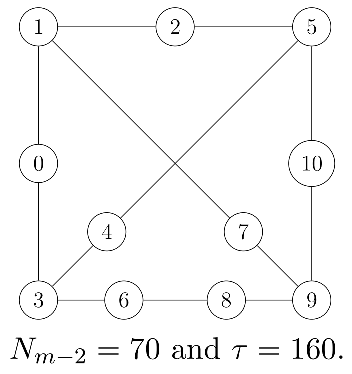


Fig. 4. Uniformly most reliable graphs in $\mathcal{H}(11, 13)$ and $\mathcal{G}(11, 13)$, respectively, and the corresponding coefficients N_{m-2} and τ of their reliability polynomials.

defined in Eq. (5). We want to find the maximum of the discrete functions f and g in X_n. By Lemma 2.1, the elements of X_n with minimum D are precisely the candidates for being a maximum of the functions f and g. Hence we just have to look at those chord path length vectors with minimum D. If $n = 4k$, then the constant vector $(k, k, k, k) \in X_n$ has minimum D and hence this is precisely the chord path length vector giving the maximum value for τ and N_{m-2} in (5). For the remaining cases, that is, when $n = 4k + \alpha$, $\alpha \in \{1, 2, 3\}$. Then $(\frac{n}{4}, \frac{n}{4}, \frac{n}{4}, \frac{n}{4}) = (k + \frac{\alpha}{4}, k + \frac{\alpha}{4}, k + \frac{\alpha}{4}, k + \frac{\alpha}{4}) \notin X_n$ and we want to find the chord path length vectors that give the maximum value of f and g in X_n. When $\alpha = 1$, there is only one vector of chord-path lengths with minimum D, which is $(k + 1, k, k, k)$.

For $\alpha = 2$, the set of vectors with minimum D is $\{(k + 2, k, k, k), (k + 1, k + 1, k, k), (k + 1, k, k + 1, k)\}$. According to Eq. (5), the last two vectors are maximum with the same value for g, which is $6k^2 + 14k + 6$, meanwhile for f we have that $(k+1, k, k+1, k)$ has maximum value since $f(k+1, k, k+1, k) = 4k^3 + 14k^2 + 14k + 4 = f(k + 2, k, k, k) + 2k + 2 = f(k + 1, k + 1, k, k) + 1$.

Finally, for $\alpha = 3$, the set of vectors $\{(k+3, k, k, k), (k+2, k+1, k, k), (k+2, k, k+1, k), (k+1, k+1, k+1, k)\}$ has minimum D. The vector of chord-path lengths $(k + 1, k + 1, k + 1, k)$ achieves the maximum both for g and f, where $g(k + 1, k + 1, k + 1, k) = 6k^2 + 17k + 10$ and $f(k + 1, k + 1, k + 1, k) = 4k^3 + 17k^2 + 22k + 8 = f(k + 3, k, k, k) + 6k + 5 = f(k + 2, k + 1, k, k) + 2k + 3 = f(k + 2, k, k + 1, k) + 2k + 1$. □

Example. For $n = 11$ and $m = 13$, the uniformly most reliable hamiltonian graph must have vector of chord-path lengths $(3, 3, 3, 2)$ and $N_{m-2} = 68$ and $\tau = 152$ (according to Theorem 2.1, case $k = 2$ and $\alpha = 3$). We also used *Nauty*[1] to generate all non-isomorphic graphs with 11 vertices and 13 edges. There are 33 851 of such graphs. Then we filtered the hamiltonian ones using a function from a *Python* library called *Graphx*. There are 56 hamiltonian graphs in this case and the uniformly most reliable hamiltonian graph is the expected one. We also computed the uniformly most reliable graph among the total set of graphs (which corresponds to a particular subdivision of K_4, as expected). We have depicted both graphs in Fig. 4. The computational method to obtain the reliability polynomial is explained in Section 2.3.

[1] http://users.cecs.anu.edu.au/bdm/nauty.

The fair cake-cutting graph $FCG_{n,c}$

In this section we present an algorithmic construction of a family of graphs $FCG_{n,c}$ that produces uniformly most reliable graphs in some cases. We introduce this family of graphs inspired by the results of Theorem 2.1, where uniformly most reliable graphs are those with chord-path length vectors with almost equal components. From a geometrical point of view, these optimal graphs can be seen as 'fair cuts of a cake', where every slice is as similar as possible to the other slices. As far as we know, there are different definitions for this *fair cake-cutting process* in the literature (see [12,25]). For our purposes, we have to cut a cake (modeled as a circular drawing of the cycle C_n) performing a given number of cuts c. The idea is to deliver each part as equal (fair) as possible for every guest. Each cut of the cake is represented in the graph by a diametrical chord, that is, one edge joining two opposite vertices. Given the order n of the hamiltonian cycle and the number of chords c ('cuts' in terms of cake cutting), we present an algorithm to construct what we call *The fair cake-cutting graph* $FCG_{n,c}$. Since the cuts can be performed only over the lines joining two opposite vertices of C_n, the idea behind this algorithm is that the slices have 'almost' equal areas in the plane.

```
input  : $n \leftarrow$ Number of desired vertices for the $FCG$
         $c \leftarrow$ Number of desired chords for the $FCG$
output: $FCG$
1  graph $fcg \leftarrow$ Cycle with edges $(0,1),(1,2),\ldots,(n-1,0)$.
2  float $separation = n/(2c)$
3  float $position = separation$
4  int $placedch = 0$
5  while $placedch \leq c$ do
6      Add edge $(int(position)-1, int(position+n/2)-1)$
7      //-1 due that the vertices begin with 0
8      $position = position + separation$ //note that the variable is still a float
9      $placedch = placedh + 1$
10 end
11 return graph
```
Algorithm 1: Fair Cake construction

A detailed explanation of the algorithm is the following: With the given number of vertices, the algorithm constructs its corresponding cycle. After, the vertices of its hamiltonian cycle are stored into a list. Then, the 'separation' between vertices of degree 3 (corresponding to the endpoints of the chords) is calculated. This 'separation' is also stored in the variable 'position' that in the first iteration of the loop will be one of the first chord endpoints. The loop in line 5 places all the cuts to the previously generated cycle. Line 6-9: Each edge chord is added to the graph using the vertices stored inside the hamiltonian cycle list in the corresponding positions. Each position is calculated by adding the previous position plus the separation. The last endpoint of each cut must have maximum distance from its first endpoint, this distance is equal to the half of the vertices of the graph. Notice that the positions of the list must be a natural number but its decimal part is not overlooked. This part is added when the next position is calculated. Finally, the cycle graph with all the chords placed is returned (see Table 2).

Example. For $n = 16$ and $c = 3$, we start from a cycle graph C_{16} with edge set $\{(0,1),(1,2),\ldots,(15,0)\}$. This table summarizes the procedure that the algorithm follows to calculate the positions (p_1, p_2) of the

Table 2
Algorithm of construction of $FCG_{16,3}$.

# cut	Separation	Accumulated	Total	p_1	p_2	(v_1, v_2)
1st	$16/6 = 2.\overline{6}$	0	$2.\overline{6}$	2	$2 + 8 = 10$	$(2 - 1, 10 - 1)$
2nd	$16/6 = 2.\overline{6}$	$2.\overline{6}$	$5.\overline{3}$	5	$5 + 8 = 13$	$(5 - 1, 13 - 1)$
3rd	$16/6 = 2.\overline{6}$	$5.\overline{3}$	8	8	$8 + 8 = 16$	$(8 - 1, 16 - 1)$

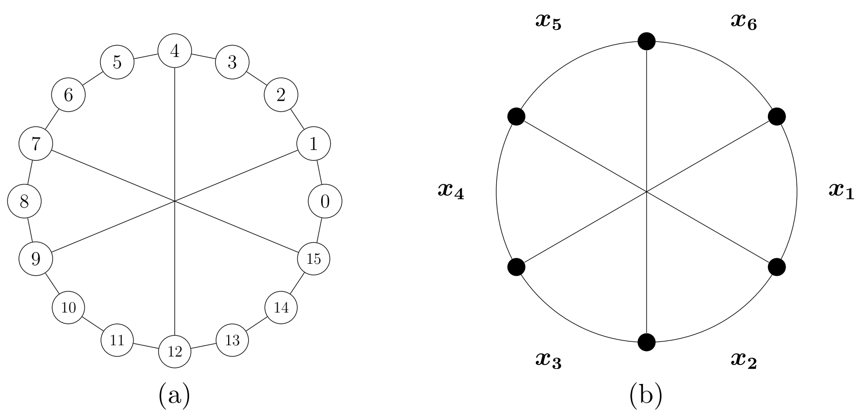


Fig. 5. (a) The fair cake-cutting graph $FCG_{16,3}$. It has vector of chord-path lengths $(2, 3, 3, 2, 3, 3)$. (b) A chord-type A graph in $\mathcal{H}(n, n + 3)$ with diametrical chords.

hamiltonian path list for each chord (v_1, v_2): First the separation of the cuts is calculated, $\frac{n}{2c} = \frac{16}{6} = 2.\overline{6}$ and then the accumulated part is added. This part is the summation of all previously calculated values (Totals). With this information, the positions of the chord endpoints can be obtained: p_1 is equal to the decimal part from the 'Total'. $p_2 = p_1 + \frac{n}{2}$. Finally, the vertices of each chord are extracted from the hamiltonian path list using the previous positions: For instance, since the vertices starts with 0, at position 5 we have the vertex 4, and the position 13 the vertex 12, therefore we add edge $(4, 12)$.

The fair cake-cutting graphs $FCG_{n,c}$ are similar to the family of graphs that are a solution of the following augmentation problem: Starting from the cycle graph C_n, add a single edge at each step, in order to maximize the reliability of the resulting graph. Romero (see [12,16]) finds the sequence of graphs $\{G^{(i)}\}_{i=0,\dots,\lfloor \frac{n}{2} \rfloor}$ with $G^{(0)} = C_n$ such that $G^{(i+1)} = G^{(i)} \cup \{e_{i+1}\}$ gives the best augmentation. This process (called also the fair cake-cutting process) ends with the circulant graph with steps 1 and $\frac{n}{2}$, that is, a cubic hamiltonian graph where every vertex is joined to its opposite vertex in the hamiltonian cycle. For $n = 8$, this is the Wagner graph depicted in Fig. 1, which is a uniformly most reliable graph for $(8, 12)$. The main difference between both families is that in our case the number of cuts c is previously known, meanwhile, in this other family, the cuts are performed as the guest arrives. In fact, both families differ when three or more cuts are performed. As a consequence of Theorem 2.1 and Proposition 2.2 we have that $FCG_{n,c}$ produces uniformly most reliable hamiltonian graphs for $c = 1$ and $c = 2$.

Corollary 2.1. *$FCG_{n,1}$ and $FCG_{n,2}$ are uniformly most reliable hamiltonian graphs for $m = n + 1$ and $m = n + 2$, respectively.*

2.3. Computational approach for $m = n + 3$ and beyond

Uniformly most reliable hamiltonian graphs have been totally characterized when $m \leq n+2$ in Section 2. We also give a construction of these optimal graphs in Section 2.2. Beyond this point, we have performed some computational tools in order to generate uniformly most reliable hamiltonian graphs for $m \geq n+3$. First, we discuss the computation of the reliability polynomial of a graph.

The computation of the reliability polynomial

The factoring theorem is a recursive method for computing $\text{Rel}(G,p)$ based on the combination of two graph operations: edge deletion $G-e$, and edge contraction G/e (for further details see, for instance, [3]):

$$\text{Rel}(G,p) = \begin{cases} \text{Rel}(G-e,p) & \text{if } e \text{ is a loop,} \\ p\text{Rel}(G/e,p) & \text{if } e \text{ is a cut-edge,} \\ (1-p)\text{Rel}(G-e,p) + p\text{Rel}(G/e,p) & \text{otherwise.} \end{cases} \tag{9}$$

The algorithm 2 shows the implementation of the theorem done in our code.

```
input  : $g \leftarrow$ Graph
output: Reliability Polynomial
// If the graph is not connected, then it has a reliability polynomial of 0
if  g is not connected then
    return 0
end
// if the number of edges > 0, then we perform the two sub-cases of the Factoring Theorem if
number of edges of g > 0 then
    edge e = g.random_edge(e)
    graph contracted = g.contract_edge(e)
    graph deleted = g.delete_edge(e)
    polynomial rec_contracted = recursion with the graph contracted
    polynomial rec_deleted = recursion with the graph deleted
    polynomial s = $p \cdot rec\_contracted + (1-p) \cdot rec\_deleted$
    return s
end
// Otherwise, we only have 0 edges and 1 vertex, which is connected, so we return 1.
return 1
```

Algorithm 2: Reliability Polynomial Factoring Theorem

We perform a modified version of this *Factoring Theorem* which works slightly different: In each recursion, if there exist some method that can directly retrieve the reliability polynomial or with less cost than another recursion, then, the method will retrieve it and the recursion will stop in that generated subgraph. In other words the main idea to improve this algorithm is to prevent it to 'dismantle' the graph to its very basic components (trivial graphs) by giving the reliability of the subgraphs before becoming basic components. We developed a series of fast formulas for specific families of graphs such as multi-tree, multi-cycle and glued cycles that when one of the subgraphs matches one of the families of our formulas then, the reliability is directly returned. With this modified method we have been able to compute the reliability polynomial of all graphs in $\mathcal{H}(n, n+3)$ for any $n \leq 11$. All (non-isomorphic) graphs have been generated first using *Nauty* and we get the hamiltonian ones using the library *Graphx* from *Python*. The drawback of this function is that, at the worst case, runs in linear time ($O(n)$). The coefficients list N_i of the reliability polynomial of those uniformly most reliable hamiltonian graphs is presented in Table 3.

Every graph listed in Table 3 is of type A (in the sense explained in Section 2.2) and it has some diametrical chords (chords joining two vertices at maximum distance in the hamiltonian cycle, see Fig. 5). Although we do not have a proof that uniformly most reliable hamiltonian graphs must be of type A with

Table 3
Coefficients list of uniformly most reliable hamiltonian graphs for $6 \leq n \leq 11$ and $m = n + 3$.

Study case	$Rel(H, p)$ coefficients vector (N_i)
$\mathcal{H}(6, 9)$	$[1, 9, 36, 78, 81]$
$\mathcal{H}(7, 10)$	$[1, 10, 44, 104, 117]$
$\mathcal{H}(8, 11)$	$[1, 11, 53, 137, 168]$
$\mathcal{H}(9, 12)$	$[1, 12, 63, 178, 240]$
$\mathcal{H}(10, 13)$	$[1, 13, 74, 226, 328]$
$\mathcal{H}(11, 14)$	$[1, 14, 86, 284, 445]$

diametrical chords for $m = n+3$, this experimental result encourage us to look optimal graphs in this subset of hamiltonian graphs that we denote as $H_D(n, m)$. So, we designed a direct way to construct all graphs in $H_D(n, n + 3)$ that lead us to go beyond $n = 11$ vertices: starting with a cycle, draws a diametrical chord dividing the cycle in two parts. Then, makes a set of 2-element combinations between each group of vertices from each part. Notice that all the edges in this set cross the first diametrical chord. With this set it makes another 2-element combinations, but this time with the elements inside the created edge set. Finally, with each combination of 2 edges creates a new graph by adding them into the graph with the diametrical chord. The algorithm 3 shows in detail this process.

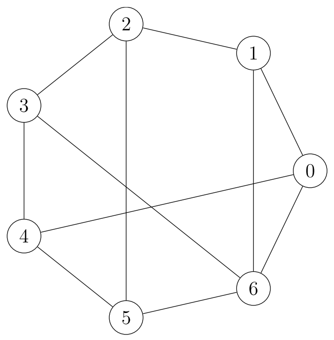
```
   input  : $n \leftarrow$ Number of vertices
   output: List of type $A$ hamiltonian graphs with at least one diametrical chord

 1 graph cycle $\leftarrow$ Cycle with edges $(0, 1), (1, 2), \ldots, (n - 1, 0)$.
 2 list vertices $\leftarrow$ cycle.vertices
 3 // Set the diametrical chord
 4 cycle $\leftarrow$ add edge $\{0,\ floor(n/2)\}$
 5 // Remove the vertices of the added chord from vertices
 6 vertices $\leftarrow$ remove $\{0,\ floor(n/2)\}$
 7 // Get possible chords
 8 list hvertices1 $\leftarrow$ 1st half of the list vertices
 9 list hvertices2 $\leftarrow$ 2nd half of the list vertices
10 list possibleVertices $\leftarrow$ all combinations of elements between hvertices1 and hvertices2
11 // From all non-existent possible edges, get combinations of 2 chords
12 // Notice that we already added 1 chord (the diametrical one)
13 list edgeCombinations $\leftarrow$ all combinations of 2 elements from the list possibleVertices
14 // For each combination create a graph and save it into a list of graphs
15 list hamiltonians
16 for combination in edgeCombinations do
17     graph tmp $\leftarrow$ copy(cycle)
18     tmp $\leftarrow$ add edges in combination
19     hamiltonians $\leftarrow$ add tmp
20 end
21 return hamiltonians
```
Algorithm 3: Generation of chord-type A hamiltonian graphs with three diametrical chords.

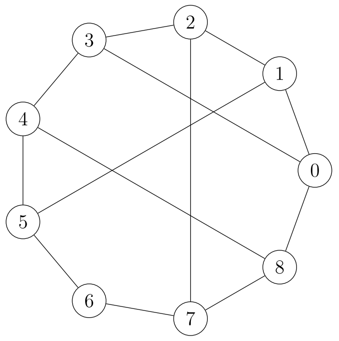
The generated list of graphs contains many isomorphic graphs. Then we use *nauty* to remove isomorphic graphs from the list. We compute the reliability polynomial of each graph of this shorter list with the method explained at the first part of this section. Table 4 shows the uniformly most reliable graphs of chord-type A with at least one diametrical chord for $n \leq 34$ and $m = n + 3$.

Table 4
Coefficients list of uniformly most reliable hamiltonian graphs of type A with at least one diametrical chord.

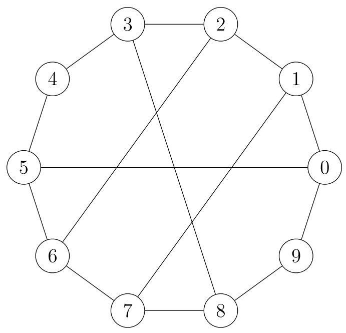
Study case	$Rel(H,p)$ coefficients vector (N_i)	Study case	$Rel(H,p)$ coefficients vector (N_i)
$\mathcal{H}_{\mathcal{D}}(12,15)$	$[1,15,99,353,600]$	$\mathcal{H}_{\mathcal{D}}(24,27)$	$[1,27,315,1977,5832]$
$\mathcal{H}_{\mathcal{D}}(13,16)$	$[1,16,112,422,755]$	$\mathcal{H}_{\mathcal{D}}(25,28)$	$[1,28,338,2192,6669]$
$\mathcal{H}_{\mathcal{D}}(14,17)$	$[1,17,126,502,948]$	$\mathcal{H}_{\mathcal{D}}(26,29)$	$[1,29,362,2426,7620]$
$\mathcal{H}_{\mathcal{D}}(15,18)$	$[1,18,141,594,1188]$	$\mathcal{H}_{\mathcal{D}}(27,30)$	$[1,30,387,2680,8700]$
$\mathcal{H}_{\mathcal{D}}(16,19)$	$[1,19,157,697,1464]$	$\mathcal{H}_{\mathcal{D}}(28,31)$	$[1,31,413,2953,9880]$
$\mathcal{H}_{\mathcal{D}}(17,20)$	$[1,20,174,814,1799]$	$\mathcal{H}_{\mathcal{D}}(29,32)$	$[1,32,440,3248,11209]$
$\mathcal{H}_{\mathcal{D}}(18,21)$	$[1,21,192,946,2205]$	$\mathcal{H}_{\mathcal{D}}(30,33)$	$[1,33,468,3566,12705]$
$\mathcal{H}_{\mathcal{D}}(19,22)$	$[1,22,210,1078,2611]$	$\mathcal{H}_{\mathcal{D}}(31,34)$	$[1,34,496,3884,14201]$
$\mathcal{H}_{\mathcal{D}}(20,23)$	$[1,23,229,1225,3088]$	$\mathcal{H}_{\mathcal{D}}(32,35)$	$[1,35,525,4225,15864]$
$\mathcal{H}_{\mathcal{D}}(21,24)$	$[1,24,249,1388,3648]$	$\mathcal{H}_{\mathcal{D}}(33,36)$	$[1,36,555,4590,17712]$
$\mathcal{H}_{\mathcal{D}}(22,25)$	$[1,25,270,1566,4272]$	$\mathcal{H}_{\mathcal{D}}(34,37)$	$[1,37,586,4978,19704]$
$\mathcal{H}_{\mathcal{D}}(23,26)$	$[1,26,292,1762,4995]$		



(a) $n=7, m=11$



(b) $n=9, m=13$



(c) $n=10, m=14$

Fig. 6. Some uniformly most reliable hamiltonian graphs for $m=n+4$.

Our conjecture about the uniformly most reliable hamiltonian graph in $H(n,m)$ must be in $H_D(n,m)$ is no longer true for $m \geq n+4$. In Fig. 6 we present some uniformly most reliable hamiltonian graphs for $m=n+4$ found by computer. Despite the case $n=8, m=12$ (see Fig. 1(a)), the remaining cases are not of chord-type A. The situation for $m=n+5$ is similar (see Fig. 7).

We have been able to find all uniformly most reliable hamiltonian graphs up to $n=11$ vertices and $m=16$ edges using the general method described at the beginning of the section: First we generate a list containing all non-isomorphic graphs of a given order and size using *Nauty*. Afterwards, the reliability polynomial of every graph in the list is computed by using our improved version of the *factoring theorem.*

A (simple) graph of order n has at most $m=\binom{n}{2}$ edges. There is only one of such graphs with maximum number of edges, which is the complete graph K_n and hence it is uniformly most reliable. There is also only one graph (up to isomorphisms) with $m=\binom{n}{2}-1$ edges, but for $m=\binom{n}{2}-2$ we have two different graphs: we can eliminate from K_n either a path of length two or two independent edges. This latter case gives the uniformly most reliable graph. More in general, it is already known that when $m \geq \binom{n}{2}-\frac{n}{2}$ then there exists a uniformly most reliable graph which is the graph whose complement graph has a set of independent edges (see [26]). Any graph with a number of edges large enough is hamiltonian, and hence uniformly most reliable graphs are hamiltonian in this case. For instance, it is well known that every graph satisfying $m \geq 1/2(n^2-3n+6)$ is hamiltonian (see [27]). Every graph with $m \geq \binom{n}{2}-\frac{n}{2}$ satisfies also $m \geq 1/2(n^2-3n+6)$ whenever $n \geq 6$ and hence uniformly most reliable graphs in this case are also uniformly most reliable hamiltonian.

Proposition 2.5. *Given an integer $n \geq 6$, there is a uniformly most reliable graph in $\mathcal{H}(n,m)$ for any $m \geq \binom{n}{2}-\frac{n}{2}$.*

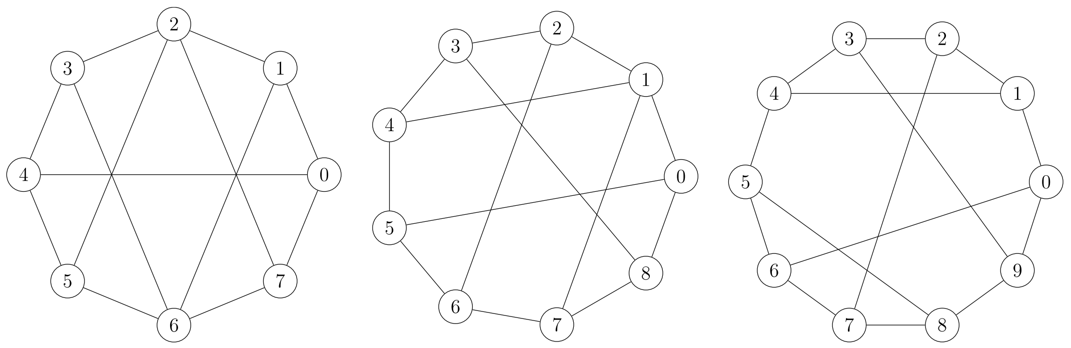

(a) $n = 8, m = 13$ (b) $n = 9, m = 14$ (c) $n = 10, m = 15$

Fig. 7. Some uniformly most reliable hamiltonian graphs for $m = n + 5$.

2.4. Non existence of uniformly most reliable hamiltonian graphs for some cases

There are some cases where uniformly most reliable graphs do not exist: it is shown in [21] that the graph G_2 of order $n \geq 6$ even, which is defined as the complement of the graph $P_4 \cup K_2 \cup \frac{n-6}{2} K_2$ satisfies $\mathrm{Rel}(G_2, p) > \mathrm{Rel}(G', p)$ for all $G' \in \mathcal{G}(n, m)$, $m = \binom{n}{2} - \frac{n+2}{2}$ and p sufficiently close to one. Besides, the complement of $2P_3 \cup \frac{n-6}{2} K_2$, defined as G_1, satisfies $\mathrm{Rel}(G_1, p) > \mathrm{Rel}(G_2, p)$ for p sufficiently close to zero. As a consequence, there are no uniformly most reliable graphs in $\mathcal{G}(n, m)$ for $m = \binom{n}{2} - \frac{n+2}{2}$. We use this result to prove that there are infinitely many pairs (n, m) where uniformly most reliable hamiltonian graphs do not exist.

Proposition 2.6. *There are no uniformly most reliable graphs in $\mathcal{H}(n, m)$ for*

- $m = \binom{n}{2} - \frac{n+2}{2}$ *for all $n \geq 6$ even;*
- $m = \binom{n}{2} - \frac{n+5}{2}$ *for all $n \geq 7$ odd.*

Proof. For the case $m = \binom{n}{2} - \frac{n+2}{2}$ it is suffice to show that the graphs G_1 and G_2 defined above as the complements of the graphs $2P_3 \cup \frac{n-6}{2} K_2$ and $P_4 \cup K_2 \cup \frac{n-6}{2} K_2$ for $n \geq 6$ even, respectively, are hamiltonian graphs. To this end notice that the minimum degree for a vertex in G_1 is $n - 3$. Hence for any two given vertices u and v, $d(u) + d(v) \geq 2n - 6 \geq n$ for all $n \geq 6$. Applying Theorem 1.1 we deduce that G_1 is hamiltonian. The same argument applies for G_2.

For $n \geq 7$ odd, define G_3 as the complement of $C_3 \cup P_4 \cup \frac{n-7}{2} K_2$ and G_4 as the complement of $C_5 \cup K_2 \cup \frac{n-7}{2} K_2$ as in [21] and apply again Theorem 1.1 to proof that they belong to $\mathcal{H}(n, m)$, where $m = \binom{n}{2} - \frac{n+5}{2}$. In [21] it is shown that $\mathrm{Rel}(G_4, p) > \mathrm{Rel}(G', p)$ for all $G' \in \mathcal{G}(n, m)$ for p sufficiently close to one. Besides $\mathrm{Rel}(G_3, p) > \mathrm{Rel}(G_4, p)$ for p sufficiently close to zero, and the proof is completed. □

3. Conclusions and open problems

In this paper uniformly most reliable hamiltonian graphs have been characterized for $m \leq n + 2$ and we give some light for the case $m = n+3$ which somehow follow the previous cases. The situation is different for $m \geq n + 4$, where the problem is totally open except for some small cases found by computer. Nevertheless, one can use these results to obtain uniformly most reliable hamiltonian graphs for a fixed value of n. For instance, when $n = 6$ we have nine cases to analyze ($6 \leq m \leq 15$): For $m = 6$, C_6 is the uniformly most reliable hamiltonian graph. For $7 \leq m \leq 9$ we have that $FCG_{6,c}$ are uniformly most reliable hamiltonian graphs for $c = 1, 2, 3$, respectively (also $K_{3,3}$ for $m = 9$, which is isomorphic to $FCG_{6,3}$). For $m = 11$ there is no uniformly most reliable hamiltonian graph (Proposition 2.6) and for $m \geq 12$ subgraphs of K_6 whose complement graph has a set of independent edges are uniformly most reliable (Proposition 2.5). It remains the case $m = 10$. We have found by computer that adding any edge to the graph $K_{3,3}$ produces the uniformly most reliable graph in this case. We point out that in every case, the uniformly most reliable graph found in this paper has the largest coefficient vector.

Problem 3.1. Characterize uniformly most reliable hamiltonian graphs for other values of n and m.

The fair cake-cutting graph $FCG_{n,c}$ presented in Section 2.2 is a uniformly most reliable graph for many values, but we believe that is also optimal for other values.

Conjecture 3.1. *Let $n \equiv 0 \pmod{2c}$. Then $FCG_{n,c}$ is a uniformly most reliable hamiltonian graph for $m = n + c$.*

The problem of finding uniformly most reliable graphs seems difficult, even for restricted versions of the problem, like the one we present here for hamiltonian graphs. It would be worth to study this problem for other classes of graphs.

Problem 3.2. Study the problem of finding uniformly most reliable graphs for a named class of graphs, such as bipartite graphs, Cayley graphs, etc.

Acknowledgments

The authors would like to thank the anonymous referees whose useful comments led to an improvement of the manuscript. Research of the authors was supported in part by grant MTM2017-86767-R (Spanish Ministerio de Ciencia e Innovación).

References

[1] C.J. Colbourn, The Combinatorics of Network Reliability, Oxford University Press, New York, NY, USA, 1987.
[2] R. J. Gould, Advances on the hamiltonian problem - A survey, Graphs Combin. 19 (2003) 7–52.
[3] H. Perez-Roses, Sixty years of network reliability, Math. Comput. Sci. 12 (3) (2018) 275–293.
[4] S. Evenand, R. Tarjan, Network flow and testing graph connectivity, SIAM J. Comput. 4 (1975) 507–518.
[5] D. Gross, J. Saccoman, C. Suffel, Spanning Tree Results for Graphs and Multigraphs: A Matrix-Theoretic Approach, World Scientific Publishing Company, 2014.
[6] F.T. Boesch, On the synthesis of optimally reliable networks having unreliable nodes but reliable edges, in: INFOCOM '88. Networks: Evolution or Revolution, Proceedings. Seventh Annual Joint Conference of the IEEE Computer and Communcations Societies, IEEE, 1988, pp. 829–834.
[7] F.T. Boesch, X. Li, C. Suffel, On the existence of uniformly optimally reliable networks, Networks 21 (2) (1991) 181–194.
[8] D. Gross, J.T. Saccoman, Uniformly optimally reliable graphs, Networks 31 (4) (1998) 217–225.
[9] D.H. Smith, L.L. Doty, On the construction of optimally reliable graphs, Networks 20 (6) (1990) 723–729.
[10] J. Brown, X. Li, Uniformly optimal digraphs for strongly connected reliability, Networks 49 (2) (2007) 145–151.
[11] J.F. Wang, M.H. Wu, Network reliability analysis: on maximizing the number of spanning trees, Proc. Natl. Sci. Counc. Repub. China A 11 (3) (1987) 193–196.
[12] P. Romero, Building uniformly most-reliable networks by iterative augmentation, in: 2017 9th International Workshop on Resilient Networks Design and Modeling (RNDM), 2017, pp. 1–7.
[13] M. Bourel, E. Canale, F. Robledo, P. Romero, L. Stábile, A hybrid GRASP/VND heuristic for the design of highly reliable networks, in: Hybrid Metaheuristics, Springer International Publishing, Cham, 2019, pp. 78–92.
[14] A. Satyanarayana, L. Schoppmann, C.L. Suffel, A reliability-improving graph transformation with applications to network reliability, Networks 22 (2) (1992) 209–216.
[15] G. Wang, A proof of Boesch's conjecture, Networks 24 (5) (1994) 277–284.
[16] G. Rela, F. Robledo, P. Romero, Petersen graph is uniformly most-reliable, in: Machine Learning, Optimization, and Big Data, Springer International Publishing, Cham, 2018, pp. 426–435.

[17] E. Canale, F. Robledo, P. Romero, J. Viera, Building reliability-improving network transformations, in: 2019 15th International Conference on the Design of Reliable Communication Networks (DRCN), 2019, pp. 107–113.
[18] A. Kelmans, V. Chelnokov, A certain polynomial of a graph and graphs with an extremal number of trees, J. Combin. Theory Ser. B 16 (3) (1974) 197–214.
[19] L. Petingi, F. Boesch, C. Suffel, On the characterization of graphs with maximum number of spanning trees, Discrete Math. 179 (1) (1998) 155–166.
[20] B. Gilbert, W. Myrvold, Maximizing spanning trees in almost complete graphs, Networks 30 (2) (1997) 97–104.
[21] W. Myrvold, K.H. Cheung, L.B. Page, J.E. Perry, Uniformly-most reliable networks do not always exist, Networks 21 (4) (1991) 417–419.
[22] J. Brown, D. Cox, R. Ehrenborg, The average reliability of a graph, Discrete Appl. Math. 177 (2014) 19–33.
[23] J. Rak, M. Pickavet, K. Trivedi, et al., Future research directions in design of reliable communication systems, Telecommun. Syst. 60 (2015) 423–450.
[24] B. Yener, Virtual embeddings on regular topology networks, in: Proceedings of SPDP '96: 8th IEEE Symposium on Parallel and Distributed Processing, 1996, pp. 562–565.
[25] F. Brandt, V. Conitzer, U. Endriss, J. Lang, A. Procaccia, Handbook of Computational Social Choice, Cambridge University Press, 2016.
[26] A.K. Kelmans, On graphs with randomly deleted edges, Acta Math. Acad. Sci. Hung. 37 (1) (1981) 77–88.
[27] G. Chartrand, L. Lesniak, P. Zhang, Graphs & Digraphs, sixth ed., Chapman & Hall, 2015.

4.3 Article: ‘Computing all-terminal network reliability: A new modular methodology’

The research presented in the article *‘Computing all-terminal network reliability: A new modular methodology’*, proposes a new modular methodology to calculate the reliability of undirected graphs. This methodology combines the generality of the Deletion-Contraction (DC) algorithm [38] with the computational speed of specific algorithms for certain families of graphs. For instance, the DC algorithm can compute the reliability of undirected graphs, but it has an exponential computational cost, while the specific algorithms for certain families of graphs are much faster but they are bound to its specific graph families. Both advantages (generality and speed) are combined in a new methodology in form of a ***Python*** package named ‘*atr*’ which can be found in the Python Package Index *PyPi* as well as in *GitHub* repositories. The methodology is based on the DC algorithm and integrates, in the form of modules, specific algorithms for certain families of graphs. Such modules can be activated or deactivated depending on the need of the user. Moreover, the package facilitates the addition of new modules without the need to modify the original code. The principal takeaways of this study are as follows:

- The development of algorithms to calculate the reliability for certain graph families.
- The development of a new methodology that combines the generality of the Deletion-Contraction algorithm with the computational speed of the specific algorithms.
- The implementation of the methodology in a ***Python*** package named ‘*atr*’ and can be found in *PyPi* and *GitHub* repositories..
- The design of the ‘*atr*’ package is modular, easily expandable, and adaptable to the given networks.

Where the package can be found following the next URLs:

- GitHub: `https://github.com/LlagosteraPol/ATR`
- PyPi: `https://pypi.org/project/atr/`

This paper has been submitted for publication and is not yet been published (changes might occur during the revision process).

Computing all-terminal network reliability: A new modular methodology

Pol Llagostera*†, Nacho López†, and Carles Comas†

***Abstract*—The reliability of a network is a measure of connectedness that can be loosely defined as the ability of the system to stand operational after the failure of some of its components. If the nodes of the network are considered to be perfectly reliable, and the links could fail independently with the same probability, this leads to the so-called reliability polynomial of the network. The coefficients of this polynomial provide an efficient tool to compare the reliability of different networks having the same number of nodes and links. Knowing this reliability is becoming increasingly important in modern society since networks become more present, e.g., in computing, road systems, social relations, etc. Therefore, among other things, we can use this knowledge to optimize the design of networks, make better decisions or mitigate possible damages. Throughout this paper, we present a new modular methodology to calculate the reliability polynomial of an undirected networks. It has been implemented in a Python package called *atr*. In form of 'modules', the methodology integrates close formulas to calculate the reliability, an improved algorithm specialized in cake networks and several methodologies to identify network families. At the end of the paper, we also detail how to easily expand the package with new modules and methodologies.**

***Index Terms*—network reliability, network design, reliability polynomial, reliability optimization, hamiltonian network**

I. Introduction

ONE of the main concerns of the early designers of computer circuits was the reliability. The problem of constructing reliable circuits was first studied in the fifties [1], when a primal probabilistic model of network reliability was introduced. In this initial model, only the links between nodes could stop working with a certain probability. Since then, networks have become ubiquitous in modern society and these probabilistic models have been applied to a whole range of networks beyond computer circuits as diverse as; image analysis [2], protein interaction [3], drug design [4] or electrical networks [5]. In this context, the design of optimal topologies is based on finding networks satisfying some requirements of their properties. The study of connectivity, fault tolerance, and/or reliability is of utmost importance in this area [6]–[8].

In this paper, we consider the problem of computing the probability that a network remains connected. More precisely, we focus on what is called, the *all-terminal network reliability problem*: given the probability p of an edge being operational in a network G, what is the probability that there exists an operational walk between every pair of vertices u and v of G.

†Department of Mathematics, University of Lleida, Lleida, 25001 Spain
*Corresponding author: P. Llagostera (email: pol.llagostera@udl.cat)

In the reliability context, networks are modeled by graphs. Our model is a stochastic network with perfect nodes and links failures: each link remains operational independently with probability $0 \leq p \leq 1$ and every link has the same probability of being operational. Moreover, no repair is allowed after a link connection fails.

From the computational point of view, the calculation of the reliability polynomial is an NP-complete problem [9]. Nevertheless, the reliability polynomial can be computed in polynomial time for a few families of networks, such as the serial-parallel networks [10], [11], or even for particular instances of two-terminal network reliability [12].

Our main contribution is to develop a powerful tool and methodologies to speed up the computation of the reliability polynomial of undirected networks. Moreover, the design of the tool is intuitive, easy to use, and modular. In section II we present a new version of the Deletion-Contraction algorithm (*DC* for short) that computes the reliability. Section III describes the implementation of the algorithm and further methodologies which we called 'modules' that work together with the *DC* algorithm to speed up the computation of the reliability. Each module section specifies its aim, the theory behind the methodology, its implementation, and use-case examples. The possible expansions of the tool are detailed in section V which includes examples and templates for new possible modules and methodologies. Finally, in the last sections, we talk about the conclusions and the possible future research lines.

A. Background

*1) **Hamiltonian networks**:* Hamiltonian networks [13] are characterized by containing a spanning cycle, that is, a cycle passing through all the nodes of the network. This cycle is called a *hamiltonian cycle*. The computational complexity of finding a hamiltonian cycle in a network is in general NP-complete. Hamiltonian networks can be drawn with all their nodes in a circular embedding (see Fig. 1).

*2) **Pathsets**:* A *pathset* of a network G is a subset of links that makes the network connected. Thus, the reliability of a network G having m links is computed using the following polynomial, known as the *reliability polynomial* of a network G:

$$\mathrm{R}_G(p) = \sum_{i=0}^{m} N_i p^i (1-p)^{m-i} \tag{1}$$

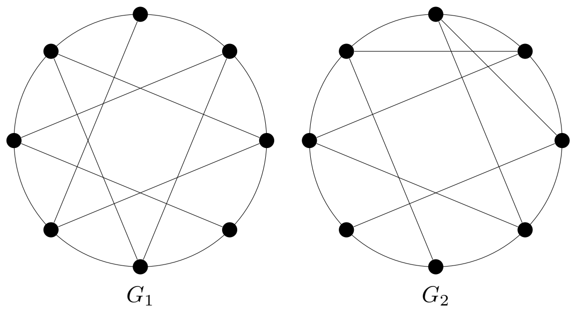


Fig. 1. Two hamiltonian networks with $n = 8$ nodes and $m = 15$ links.

where N_i denotes the number of pathsets of cardinality i.

3) ***Deletion-Contraction algorithm:*** The Deletion-contraction methodology (or *DC*) can compute the reliability of any graph using the combination of two network operations: link deletion, and link contraction. The operation of contracting a link e from a network G consists of deleting e and identifying the nodes of the link, while keeping all the other links incident to those nodes. Note that this operation may create multiple links and loops. We denote the operation of contracting an edge e by G/e, and the operation of deleting e by $G - e$. This methodology creates sub-graphs where the process is repeated again until the resulting sub-graph is trivial (trivial graph) which reliability is p. Then the resulting reliabilities of each sub-graph, are back-propagated in the algorithm while assembling together to create the reliability polynomial of the original graph. The *Deletion-Contraction algorithm [10], [14]* (*DC* for short) is based on the following recursive formula, also known as *the factoring theorem [15]–[17]*:

$$\mathrm{R}_G(p) = \begin{cases} \mathrm{R}_{G-e}(p) & \text{if } e \text{ is a loop,} \\ p \cdot \mathrm{R}_{G/e}(p) & \text{if } e \text{ is a cut-link,} \\ (1-p) \cdot \mathrm{R}_{G-e}(p) + & \\ p \cdot \mathrm{R}_{G/e}(p) & \text{otherwise.} \end{cases} \tag{2}$$

This formula is also graphically represented in Figure 2.

The algorithm's strongest point is its generality since it can compute any graph's reliability. On the other hand, its weakness is the computational time required for the calculations. Although the running time of the *DC* algorithm depends on several practical issues, such as the choice of the data structure, the link selection strategy, etc. the truth is that the running time of the *DC* algorithm may be exponential with the number of nodes. Nevertheless, there are a few collections of algorithms that compute $R_G(p)$ much faster than the *DC* for certain restricted families of networks. For instance, Satyanarayana [18] purposes a linear-time algorithm to compute reliability for series-parallel networks. In conclusion, the *DC* algorithm is useful when a graph cannot be categorized in any family or if its family doesn't have any algorithm that computes $R_G(p)$ faster than exponential time.

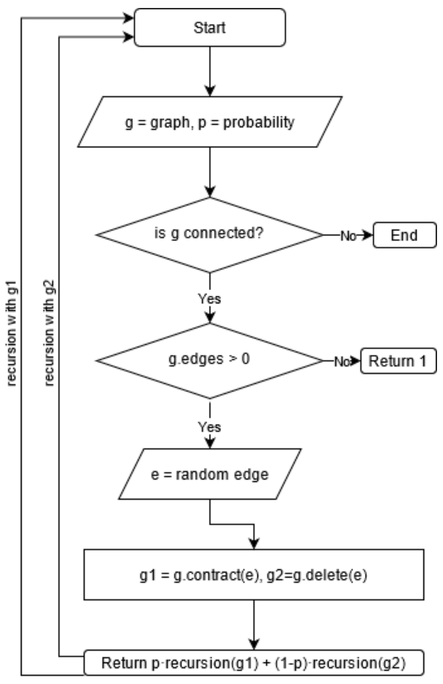

Start
g = graph, p = probability
is g connected?
No
End
Yes
g.edges > 0
No
Return 1
Yes
e = random edge
g1 = g.contract(e), g2=g.delete(e)
Return p·recursion(g1) + (1-p)·recursion(g2)
recursion with g1
recursion with g2

Fig. 2. Factoring theorem scheme

II. New *DC* algorithm

We developed a methodology that combines the generality of the *DC* algorithm with the computational speed of the algorithms designed to calculate the $R_G(p)$ for certain restricted families. Therefore, our algorithm effectively calculates the all-terminal reliability polynomial of any undirected graph faster than the original *DC*. We design the algorithm to be highly modular and dynamic. Its functionalities are encapsulated in what we called 'modules' which allows to treat or modify them separately without affecting the main algorithm. Moreover, due to this structure, it's easy to expand the main algorithm with new functionalities. The algorithm is also dynamic since it can be adapted to any given graph, for instance, one can select the modules that better fits the given graph. These selected modules will be used to calculate the reliability polynomial of the network, therefore, a fitter selection of modules, reduces the computational time.

A. Algorithm description

The idea behind the improved version of the *DC* algorithm, is to prevent 'dismantling' the network to its very basic components (trivial networks). If the graph or any sub-graph (produced by an edge contraction or deletion) is from a family in which there exists a faster algorithm that calculates its $R_G(p)$, then such algorithm is used. If the family cannot be found or doesn't exist any specific algorithm, then another iteration of the *DC* algorithm is performed.
We have observed that during the factorization process in the

classical version of the *DC* algorithm, many of the obtained subnetworks are either a multilink tree, multilink cycle, or a tree of cycle networks. Hence, the closed formulas given for this particular family of networks have been implemented and added in form of modules in our new version of the *DC* algorithm. Such modules contain two functionalities; determine if the graph fits the module specified graph family (identification) and calculate the $\mathrm{R}_G(p)$ of the graph using specific algorithms for that family.

B. Algorithm methodology

Previous to the calculation of the reliability, the algorithm can be fitted to the given graph by selecting which modules will be used, and, in case of edge contraction and deletion, what requirements would be followed in order to choose the edge to perform such operations. Once the features that fit better the given graph have been selected, the algorithm starts with the identification process. In this procedure, each of the chosen modules determines if the network belongs or not to its family. If the network can be categorized as part of a module family, then its reliability is calculated using the specific algorithm provided by the module. If the network doesn't fit in any of the categories present in the algorithm modules, then, an edge is chosen for its contraction and deletion which creates two sub-graphs (see previous subsection). In the case that any of the sub-graphs is trivial, its reliability is directly given, if not, all the process (since the identification point) will be applied again to that sub-graph. Each calculated reliability is assembled together to form the $R_G(p)$ of the original graph following the same procedure as the original *DC* algorithm described in section I-A3. Figure 3 contains a visual representation of the previously described methodology. In the next section we detail the implementation of the new *DC* algorithm which has been developed in form of a *python* package called ***atr***.

III. The **atr** package

We have created a python package called **atr** that calculates the all-terminal reliability of any undirected graph based on our improved *DC* algorithm described in the previous section. The package can be found online in the official python repository **PyPi** and in **GitHub** repository, which can contain more recent updates (that later are updated to PyPi). The url of these two repositories is presented below:

- **PyPi**: https://pypi.org/project/atr/
- **GitHub**: https://github.com/LlagosteraPol/ATR

The structure of the package is divided in one main function named *calculate_reliability()* two classes (*EdgeSelector* and *RelModule*) and four sub-classes (*ModuleTree*, *ModuleCycle*, *ModuleCycleTree* and, *ModuleCake*) which inherits from *RelModule*. Also, aditional methods have been implemented in the *graphtools* structure which provides functionalities to all the previous classes (Fig. 4 contains the UML diagram of the package). For its network inputs, uses the graph structure from the well known library *networkx* [19] which creates a model of the networks. On the other hand, for the outputs (resulting network reliability polynomials), uses the library

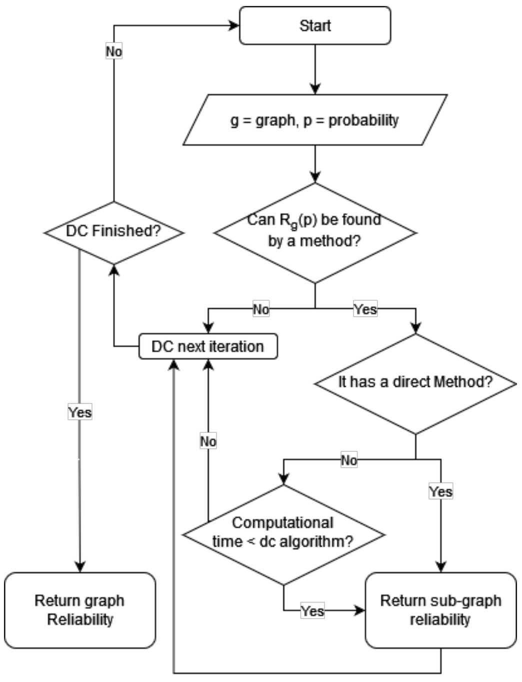

Start
g = graph, p = probability
Can R_g(p) be found by a method?
No
Yes
DC Finished?
No
Yes
DC next iteration
It has a direct Method?
No
Yes
No
Computational time < dc algorithm?
Yes
Return graph Reliability
Return sub-graph reliability

Fig. 3. Improved factoring theorem scheme

sympy [20].

Only with the *calculate_reliability()* function, the user can manage all the capabilities of the package. This function calculates the $\mathrm{R}_G(p)$ using two elements; the graph with class *networkx* and a list of strings containing the name of the module classes desired to be used. These modules are the functionalities that calculates the reliability for specific families of graphs (see II). Each module checks (identifies) if the given network belongs to its module graph family and calculates its reliability in that case. If it doesn't belong, then the next module provided in the list is executed. If the network doesn't belong to any of the provided module families, then an iteration of the basic *DC* is performed. Note that if one module positively identifies the network, the other modules will not perform the identification process.
In addition, two optional parameters can be provided to the function *calculate_reliability()* in order to customize even more the behavior of the function. These two parameters are; *graph_pruning* and, *edge_selector*, both are described below:

1) Graph pruning: The optional attribute *graph_pruning* requires a true/false boolean option (true by default) that, if set to true, tells the algorithm to separate the 'tree' parts of the graph and create sub-graphs with each connected component. The process is done in each iteration before the graph is evaluated by the modules. Then each sub-graph is treated as an input for a new iteration of the algorithm, where the modules evaluates the sub-graph and, if positively identified, calculates

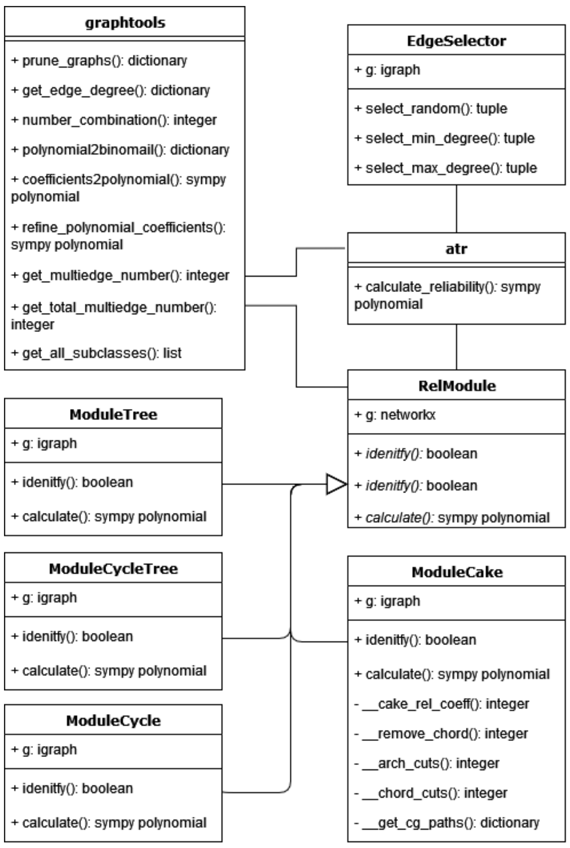

graphtools
+ prune_graphs(): dictionary
+ get_edge_degree(): dictionary
+ number_combination(): integer
+ polynomial2binomail(): dictionary
+ coefficients2polynomial(): sympy polynomial
+ refine_polynomial_coefficients(): sympy polynomial
+ get_multiedge_number(): integer
+ get_total_multiedge_number(): integer
+ get_all_subclasses(): list
EdgeSelector
+ g: igraph
+ select_random(): tuple
+ select_min_degree(): tuple
+ select_max_degree(): tuple
atr
+ calculate_reliability(): sympy polynomial
RelModule
+ g: networkx
+ idenitfy(): boolean
+ idenitfy(): boolean
+ calculate(): sympy polynomial
ModuleTree
+ g: igraph
+ idenitfy(): boolean
+ calculate(): sympy polynomial
ModuleCycleTree
+ g: igraph
+ idenitfy(): boolean
+ calculate(): sympy polynomial
ModuleCycle
+ g: igraph
+ idenitfy(): boolean
+ calculate(): sympy polynomial
ModuleCake
+ g: igraph
+ idenitfy(): boolean
+ calculate(): sympy polynomial
- __cake_rel_coeff(): integer
- __remove_chord(): integer
- __arch_cuts(): integer
- __chord_cuts(): integer
- __get_cg_paths(): dictionary

Fig. 4. UML class diagram of the inherent structured of the **atr** package indicating its classes, methods, and internal relations. Functions provided by the different classes use the **-** and **+** symbols to differentiate between private and public functions, respectively, and ***italic*** to indicate abstract functions. Associations between the classes are indicated by solid lines and inheritance relations by arcs ($\rightarrow$).

the $\mathrm{R}_G(p)$ of the graph or, if cannot be categorized, proceed with the contraction-deletion recursion of the *DC* algorithm (see section II). The pruning procedure is repeated in each iteration of the algorithm and for each sub-graph produced by the edge contraction and deletion.
This option works better in combination with the *TreeModule*, since the module calculates directly the reliability of all the tree components, the computational time is effectively reduced. It is optimal to use this option with graphs that contain a notorious amount of tree parts or if such parts are prone to appear in the sub-graphs. An example of this procedure is shown in Figure 5.

2) Edge selector: The attribute *edge_selector* specifies which methodology is used in order to select the edges to perform a contraction and deletion. This operation is performed in each iteration of the algorithm in the case that the graph (or sub-graph) is not identified. As an argument, it requires a string containing the name of the selector function of the class *EdgeSelector*. By default it selects a random edge using the

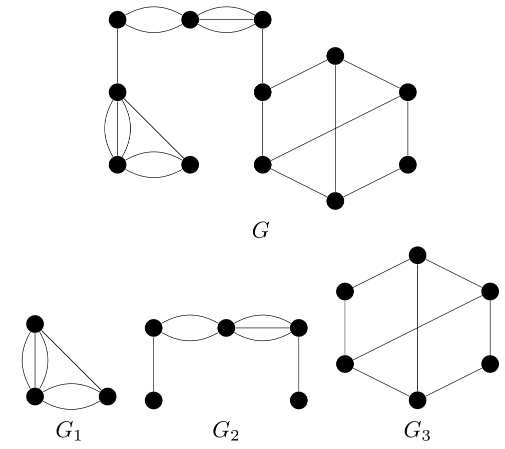


Fig. 5. A graph G divided into three sub-graphs G_1, G_2, and G_3 using the graph pruning methodology.

function *select_random()* from the class *EdgeSelector*. This class also contains two more functions, *select_min_degree()* and, *select_max_degree()*. These two functions select the edge where the sum of its conforming node degrees are maximum and minimum respectively.

A. Using the package

This section explains step by step how to use the package, that is; how to load it, create a network (or read it from a file), and calculate the all-terminal reliability polynomial using different configurations.

*1) **Loading the package**:* Since the package is included in the official Python repository, it can be installed as any other official Python package with the help of the standard package manager for Python **pip**:

```
# Unix/macOS
python3 -m pip install atr

# Windows
py -m pip install atr
```

The package can also be installed directly from the GitHub repository with **pip** using its URL:

```
# Unix/macOS
python3 -m pip install
        git+https://github.com/LlagosteraPol/ATR

# Windows
py -m pip install git+https://github.com/LlagosteraPol/ATR
```

Another option is to download the package either from the official repository PyPi or from GitHub and install it with the same commands shown previously but, instead of an URL, using the directory where the package is found.

*2) **Obtaining a network**:* The *atr* library works with *networkx* class networks. To create such graphs, the *networkx* library offers different options. One option is to create a network from an adjacency matrix with the function *from_numpy_matrix()*. As the name indicates, the matrix must be an object of class *NumPy* [21]. Moreover, this function also permits to create multi-edge networks with the attributes *parallel_edges* and *create_using*. The first attribute allows specifying directly in the adjacency matrix how many edges exist between each pair of nodes. The second attribute specifies the object type of the new network, for instance, a multi-edge graph will be of type *MultiGraph*. Following, we present an example:

```
import networkx as nx
import numpy as np
A = np.array([[0, 2, 0, 1],[2, 0, 3, 0],
        [0, 3, 0, 2],[1, 0, 2, 0]])
g = nx.from_numpy_matrix(A, parallel_edges = True,
        create_using=nx.MultiGraph)
```

Where the visual representation of the network is shown in Figure 7 and the adjacency matrix is:

$$\boldsymbol{A} = \begin{pmatrix} 0 & 2 & 0 & 1 \\ 2 & 0 & 3 & 0 \\ 0 & 3 & 0 & 2 \\ 1 & 0 & 2 & 0 \end{pmatrix}$$

Another option is to give the edges of the network. To this end, it's requested to specify the nodes that conform each edge. Below we exemplify this procedure by building a multi-edge tree network (see Fig. 6):

```
import networkx as nx
g = nx.MultiGraph([(1, 2), (1, 2), (1, 2),
        (2, 3), (2, 3), (3, 4)])
```

An interesting option to save and load simple undirected networks, is to use the **graph6** format. The networks in this format, are represented in **ASCII** characters which require little space in memory. Moreover, several networks can be written altogether in a single *.txt* file where each line represents a different graph. The *networkx* library provides the function *read_graph6()* to read a file containing graphs in **graph6** format and convert them into a list of *networkx* class objects. The library also gives the function *write_graph6()* to format any *networkx* graph into **graph6** and save it into a file. Below we present an example:

```
import networkx as nx

g = nx.Graph([(1, 2), (2, 3), (3, 4), (4, 1)])
nx.write_graph6(g, "test.txt", header=False)
g = nx.read_graph6("test.txt")
```

*3) **Calculating the reliability**:* As explained at the beginning of section III, the $R_G(p)$ is calculated using the function *calculate_reliability()* from the *atr* library. Such function admits different configurations to speed up the computational time depending on the given graph. Below we present an example of how to calculate the $R_G(p)$ from the network shown in Figure 5. To this end, we use different approaches, first; using only the pure *DC* algorithm (without modules and graph pruning), second, with all modules and default optional attributes, and, finally, with some modules, with graph pruning and specifying the selector. Note that the edge selection function (attribute *edge_selector*) will be set on random by default since we didn't specify it.

```
1: import networkx as nx
2: g = nx.MultiGraph( [(1,2), (1,2), (1,3), (2,3), (2,3), (2,3),
   (3,4), (4,5), (4,5), (5,6), (5,6), (5,6), (6,7), (7,8), (8,9),
   (9,10), (10,11), (11,12), (12,7), (8,11), (9,12)] )

3: rel1 = atr.calculate_reliability(g)

4: rel2 = atr.calculate_reliability(g,
        modules = ['ModuleTree', 'ModuleCycle',
        'ModuleCycleTree', 'ModuleCake'])

5: rel3 = atr.calculate_reliability(g, prune = True,
        modules = ['ModuleTree', 'ModuleCycleTree',
        'ModuleCake'])
```

Notice that the last configuration of the *calculate_reliability* function is the fittest for the given graph since, thanks to the pruning process, it separates the tree part of the network (also shown in Fig. 5) and creates three sub-networks. Moreover, the chosen modules fit exactly the family categories of these three sub-networks which are: multi-edge cycle, multi-edge tree, and cake graph (in the next sections detail each module). The computational times verifies the previous affirmation, where, the first computation (line 3) takes $\mathbf{4.978}$ seconds, the following (line 4) takes $\mathbf{1.278}$ seconds and the last computation (line 5) $\mathbf{0.024}$ seconds. Such results were calculated using a computer with a core *Intel i5* $8th$ *Gen* and 18GB of RAM. The resulting all-terminal reliability polynomial from the network in the example is:

$$R_G(p) = p^{21} + 19.0p^{20}(1-p) + 168p^{19}(1-p)^2 + 908p^{18}(1-p)^3 + 3309p^{17}(1-p)^4 + 8480p^{16}(1-p)^5 + 15487p^{15}(1-p)^6 + 19958p^{14}(1-p)^7 + 17442p^{13}(1-p)^8 + 9384p^{12}(1-p)^9 + 2376p^{11}(1-p)^{10}$$

IV. Modules

To enhance our new *DC* algorithm, we developed a series of methodologies that calculates the all-terminal reliability for several families of graphs. These methodologies had been designed to be modules of our *DC* algorithm (see section II). The main idea of a module is to be easily added or removed from the new *DC* algorithm, therefore to be 'modular'. Moreover, the modules also contain an algorithm that determines if the given graph is part of the graph family of the module, if so, the module can proceed to calculate the $R_G(p)$. Each module focuses on a specific graph family such as, multi-paths, multi-cycles, multi-cycle trees and the hamiltonian cake graphs (sections IV-A, IV-B, IV-C, and IV-D). Although the families belonging to the series-parallel networks already have known methodologies to calculate the $R_G(p)$ [18], in order to be incorporated into the new *DC* algorithm, we've created other compact algorithms that fit the requisites of the modules. We implemented into the *atr* library all the specific modules for the previously mentioned graph families. Each of them is represented by a sub-class which inherits form the *RelModule* class (see Fig. 4). All the sub-classes contain a constructor and at least two functions, the function *identify()* which determines if the network type matches the module, and the function *calculate()* which calculates the reliability of the given network. Although the main class of the *atr* library uses automatically the functionalities of the modules (see III), such functionalities can be directly used. For instance, this is convenient where the given network family is known and there is no need to identify the graph. In these cases, the specific module reliability calculator algorithm can be directly applied. Below we detail each module implementation by presenting its aim, the theory behind the algorithm and how to (directly) use its functionalities with examples. Note that all the examples assume the imports for the *networkx* library to create networks, the module classes, and the *graphtools* toolbox which contains extra methodologies such as converting a polynomial to a grouped sum of binomials (function *polynomial2binomial()*). Such imports are in the form:

```
import networkx as nx
import relmodules
import graphtools
```

A. Multi-Tree Module

A (simple) tree network of n nodes, denoted by $T(n)$, is any connected network with $n-1$ links where the deletion of any of the links produces a non-connected network. The reliability polynomial of $T(n)$ is therefore $\mathrm{R}_{T(n)}(p) = p^{n-1}$. We define the *multi-link tree* $MT(M)$ as a tree network on n nodes but allowing multiple links between nodes, where $M = \{m_1, m_2, ..., m_{n-1}\}$ is the set where each m_i counts the number of links between two given nodes, where $m_i \geq 1$, also known as link-cardinalities set. For instance, a simple tree network has $m_i = 1$ for all $1 \leq i \leq n-1$. Notice that $MT(M)$ is no longer connected when at least one whole set of links between two nodes is removed. If we generalize the classical reliability polynomial for simple trees $T(n)$ taking into account that $m_1 = m_2 = \cdots = m_{n-1} = 1$, we can determine the network reliability of Multi-Trees by combining m_i sets of disconnected links as shown in formula 3.

$$\mathrm{R}_{MT(M)}(p) = \prod_{i=1}^{n-1}[1-(1-p)^{m_i}] \tag{3}$$

The class *ModuleTree* from the *atr* package implements the functionalities of the Multi-Tree Module. This module uses the capabilities of the *networkx* package to determine if the given network is a Multi-Tree and implements the formula 3 to calculate its reliability. Following we demonstrate the usage of this module with an example.

First the network is created using the function *MultiGraph* from the *neworkx* library (such network is represented in Fig. 6). Once we have the graph, we use it to create an object instance from the class *ModuleTree*. Finally, if the network belongs to the Multi-Tree family, then we calculate its reliability. Note that we use the function *polynomial2binomial()* to represent the resulting polynomial in form of a grouped sum of binomials.

```
g = nx.MultiGraph([(1, 2), (1, 2), (1, 2),
        (2, 3), (2, 3), (3, 4)])

treemodule = atr.relmodules.ModuleTree(g)

if treemodule.identify():
    rel = treemodule.calculate()
    print(atr.graphtools.polynomial2binomial(rel))
```

The previous code prints the following result:

$$R_G(p) = p^6 + 5p^5(1-p) + 9p^4(1-p)^2 + 6p^3(1-p)^3$$

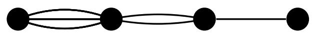


Fig. 6. A multilink Tree $T(M)$ on 4 nodes with set of link-cardinalities $M = \{3, 2, 1\}$. Its reliability is therefore $\mathrm{R}_{MT(M)}(p) = (1-(1-p)^3)(1-(1-p)^2)(1-(1-p))$

B. Multi-Cycle Module

As in the multi-link tree network, the *multilink-cycle* $MC(M)$ allows multiple links between pairs of nodes of the cycle graph on n nodes, where $M = \{m_1, m_2, ..., m_n\}$ is the set of link-cardinalities. In order to disconnect $MC(M)$ at least two whole sets of multiple links must be removed from the network. Therefore we can calculate all the possible ways to disconnect the network and retrieve all connected sub-networks (equation 4).

$$\mathrm{R}_{MC(M)}(p) = 1 - \left(\sum_{k=2}^{n}(-1)^k \sum_{i_1<\cdots<i_k}(k-1)(1-p)^{m_{i_1}+\cdots+m_{i_k}}\right) \tag{4}$$

where $m = \sum_{i=1}^{n} m_i$ is the total number of (multiple) links. In the particular case $m_1 = m_2 = \cdots = m_n = m'$, we can simplify the calculations of the previous equation:

$$\mathrm{R}_{MC(M)}(p) = 1 - \left(\sum_{k=2}^{n} (-1)^k (k-1) \binom{n}{k} (1-p)^{m' \cdot k} \right) \quad (5)$$

This module is implemented in the *atr* library by the class *ModuleCycle*. To determine if the family of the given graph is a cycle or multi-cycle, the class function *identify()*, temporarily converts the given graph into a simple network (without multi-edges), and determines its regularity. If the network is 2-regular (all the nodes have degree 2), then it's a cycle which indicates that its multi-link version must be also a cycle. To calculate the reliability and to improve its computational speed, the function *calculate()* counts the number of parallel edges between nodes, if all sets of parallel edges contains the same number of links, the function will use the equation 5, otherwise the equation 4 will be chosen. Below we exemplify the usage of this module with an example:

Like the multi-tree example presented previously, we first create the network (shown in Fig. 7) using the function *MultiGraph* from the *neworkx* library. We create then a *ModuleCycle* object using the previous network and finally if the network is detected as being part of the module family (either cycle or multi-cycle), then the reliability polynomial is calculated and printed (shown) in form of a grouped sum of binomials using the function *polynomial2binomial()*.

```
g = nx.MultiGraph([(1, 2), (1, 2), (1, 2), (2, 3),
        (2, 3), (3, 4), (4, 1), (4, 1)])

cyclemodule = atr.relmodules.ModuleCycle(g)

if cyclemodule.identify():
    rel = cyclemodule.calculate()
    print(atr.graphtools.polynomial2binomial(rel))
```

The resulting output of the code is the following:

$R_G(p) = p^8 + 8p^7(1-p) + 28p^6(1-p)^2 + 54p^5(1-p)^3 + 58p^4(1-p)^4 + 28p^3(1-p)^5$

C. Multi-CycleTree Module

Another closed formula can be given for a family of networks $MTC(M)$ which can be loosely defined as a tree of cycle networks (see Fig. IV-C). The $MTC(M)$ network depends on how the cycles are connected between them. To this end, a reduction of its links can be done, by 'collapsing' the links of the cycles to one link per cycle. The resulting network is the basis tree network. However, the reliability of $MTC(n)$ does not depend on the structure of the basis tree network but on the set $M = \{m_1, m_2, ..., m_s\}$ of the number of links of each cycle. Indeed, the coefficients N_i of

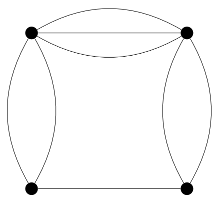


Fig. 7. A multilink Cycle G on 4 nodes with set of link-cardinalities $\{3, 2, 1, 2\}$. Its reliability is therefore $\mathrm{R}_G(p) = 1 - [1((1-p)^{3+2} + (1-p)^{3+2} + (1-p)^{3+1} + (1-p)^{2+2} + (1-p)^{2+1} + (1-p)^{2+1}) - 2((1-p)^{3+2+2} + (1-p)^{3+2+1} + (1-p)^{3+2+1} + (1-p)^{2+2+1}) + 3(1-p)^{3+2+2+1} = 1 - [(2(1-p)^5 + 2(1-p)^4 + 2(1-p)^3) - 2((1-p)^7 + 2(1-p)^6 + (1-p)^5) + 3(1-p)^8]$

the reliability polynomial $\mathrm{R}_G(p) = \sum_{i=0}^{m} N_i p^i (1-p)^{m-i}$ can be computed directly with the new formula:

$$\mathrm{R}_{MTC(M)}(p) = p^m + mp^{m-1}(1-p) + \sum_{k=2}^{s} \left(\sum_{i_1 < \cdots < i_k} m_{i_1} \cdots m_{i_k} \right) p^{m-k} (1-p)^k \quad (6)$$

where $m = \sum_{i=1}^{s} m_i$. We recall that the coefficient N_k in the reliability polynomial counts the number of different connected subnetworks. In this case, after the removal of k links of $MTC(M)$, the subnetwork remains connected only if these k links belong to different cycles each. This means that just one link can be removed from each cycle if we want to keep the network connected. For each k, there are $\sum_{i_1 < \cdots < i_k} m_{i_1} \cdots m_{i_k}$ different ways of removing these k edges.

The implementation of this module is done within the class *ModuleTreeCycle* from the *atr* library. The network identification process to determine if it belongs to the family of 'tree of cycles' (or Cycle-Tree), follows this reasoning; A Cycle-Tree network must contain only cycles without any shared edges. To know if the graph contains only cycles, we can search for the minimum set of cycles $C = \{c_1, c_2, \cdots, c_n\}$ such that any cycle in the graph can be written as a combination of these basis cycles, this is also known as the **cycle basis** of a network. With this set, we can compare the number of non-repeated nodes from the cycle basis with the order of the network. If the previous condition is asserted, then the network is formed only with cycles. Lastly, if none of the cycles contain shared edges, the following equality will hold true:

$$o(G) = \sum_{k=1}^{n} (o(c_k)) - n + 1$$

Where $o(G)$ is the order of the given network G and $o(c_k)$ is the order of a cycle from the cycle basis C. Therefore, if the graph only contains cycles and such cycles don't share any edges, then the network belongs to the Cycle-Tree family.
The calculation of the reliability depends if the given network is simple or Multi-Graph (with multiple edges), the function *calculate()*, uses a different approach for each case. If it's a simple graph, the previous equation 6 is chosen. If instead, the network contain multiple edges, the function takes the cycles

from the network cycle basis and, for each c_k applies the equation 4 (see previous section IV-B), therefore:

$$\mathrm{R}_{MTC(M)}(p) = \prod_{k=1}^{n} (\mathrm{R}_{MC(c_k)}(p)) \qquad (7)$$

As in the previous examples, using this module requires four steps; Step one, build the network. Step two, create an object instance of the module (in this case *ModuleCycleTree*) using the previous network. Step three, identify the network. Step four, if the network belongs to the module family, calculates its reliability. An optional last step is to transform the given polynomial to a grouped sum of binomials for better comprehension.

```
g = nx.MultiGraph([(1, 2), (1, 3), (2, 3), (3, 4), (3, 6),
        (4, 5), (5, 6), (6, 7), (7, 8), (8, 9),
        (9, 10), (10, 11), (11, 6)])

cycletreemodule = atr.relmodules.ModuleCycleTree(g)

if cycletreemodule.identify():
    rel = cycletreemodule.calculate()
    print(atr.graphtools.polynomial2binomial(rel))
```

Notice that even the network object is declared as MultiGraph, it doesn't contain any parallel edges, therefore the *calculate()* function will choose the equation for simple graphs (Equation 6). The resulting reliability from this example network is:

$$R_G(p) = p^{13} + 13p^{12}(1-p) + 54p^{11}(1-p)^2 + 72p^{10}(1-p)^3$$

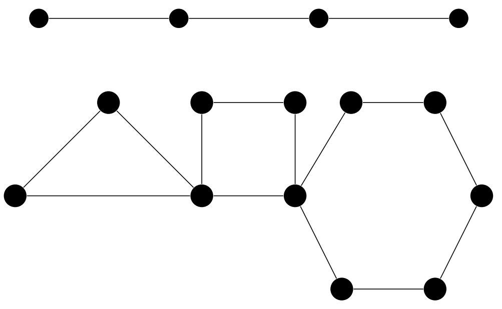


Fig. 8. The network $G = MTC(\{3,4,6\})$ (top) having the linear path on 4 vertices as its basis network (bottom) has reliability polynomial. $\mathrm{R}_G(p) = p^{13} + (3+4+6)p^{12}(1-p) + ((3\cdot 4) + (3\cdot 6) + (4\cdot 6))p^{11}(1-p)^2 + 3\cdot 4\cdot 6p^{10}(1-p)^3$

D. Cake Module

We developed a new method to speed up the computation of the reliability polynomial for hamiltonian networks [22] belonging to the cake graph family. These hamiltonian networks can be described as cycle networks C_n with extra diametral interior links that cross each other (like diametral cuts in a cake) known as *chords*. This type of networks are characterized by the *chord-path length vector* $(x_1, x_2, \ldots, x_c)$, where $x_i > 0$, that gives the lengths of every path defined by the end vertices of the chords given in clockwise order (see Fig. 9).

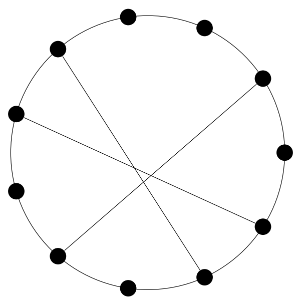


Fig. 9. A cake network with chord-path length vector $(3, 2, 1, 2, 2, 1)$.

The specific algorithm for cake networks calculates the probability of being disconnected. This procedure is divided into three parts; (a) Calculate all possible ways to disconnect the network by 'breaking' only links of the cycle. (b) Possible disconnections by 'extracting' the diametral chords, that is, a connected sub-network constituting of links of the cycle and only one chord. (c) Removing one chord and applying (a) and (b) on the resulting sub-network. Following, we give a detailed explanation of these steps.

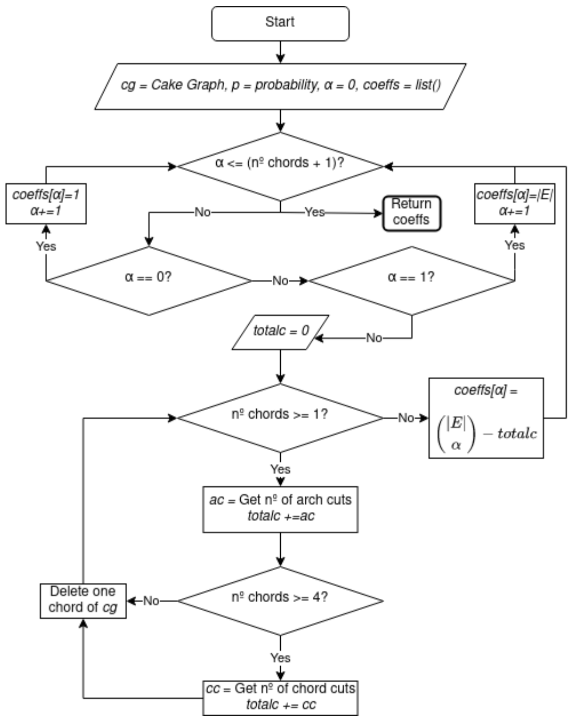


Fig. 10. Cake algorithm scheme

Given any α = number of disconnected links. For (a), the network is disconnected by breaking two links of the same c-path and any other $\alpha - 2$ links of the cycle c-paths $(x_1, x_2, \cdots, x_c)$. Therefore we need to consider different combinations of possibilities; only breaking links of the same path x_i, breaking 2 links of the same path and others only of a path $x_j > x_i$, same case but breaking links of a path where $x_j < x_i$ and, any possible combination of the last three cases. A pseudocode with this procedure is given in *algorithm 1*.

Regarding (b), the rule is we can only break one arc for path since the cases where more than one are broken, are already contemplated in (a). Then, in order to disconnect the network and following the previous rule, we need to break a minimum of four arcs that surrounds a chord or multiple chords and effectively 'extract' them as described previously. After we have the minimum set that disconnects the network (4 arcs), then we can break any combination of $\alpha - 4$ arcs that are not already in the minimum set. The algorithm is described into the pseudocode *algorithm 2*. Lastly, only the links that must be considered are the chords. Point (c) is the part of the algorithm that handle these cases. The idea behind (c), is to recursively delete a chord, applying (a) and (b) to the generated sub-network, and continue this procedure until no chords are left. A better look on this procedure is shown in *figure 10*. Notice that the procedures described in the scheme as 'Get number of arc cuts' and 'Get number of chord cuts' are referred to the previous nodes (a) and (b) respectively.

Algorithm 1 Pseudocode to calculate the number of all possible ways to disconnect a network with α 'cuts' by 'cutting' only the arcs.

```
procedure ArcCutsAlgorithm($x$, $\alpha$, $m$, $c$)
    // $x$: paths of length $x_i$
    // $\alpha$: maximum number of link cuts without disconnecting the network
    // $m$: number of links of the network
    // $c$: number of chords of the network

    int result = 0

    for $j \leftarrow 2, \alpha$ do
        for $i \leftarrow 2, |x|$ do
            if $j \leq x_{i-1}$ then
                current_path_comb = $\binom{x_{i-1}}{j}$
            else
                continue
            end if
            if $(\alpha - j) > 0$ then
                for $z \leftarrow 1, (\alpha - j)$ do
                    next_path_comb = $\binom{m-c-(|x_1|+|x_2|+...+|x_i|)}{\alpha-j-z}$
                    if $z \leq (i-1)$ then
                        prev_path_comb_lst = $P_z(x_1, x_2, ..., x_{i-1})$
                        int prev_path_comb = 0
                        for comb $\leftarrow$ prev_paths_comb_lst do
                            int mult = 1
                            for $w \leftarrow$ comb do
                                mult $\cdot =$ $|x_w|$
                            end for
                            prev_path_comb += mult
                        end for
                    end if
                    result += curr_path_comb $\cdot$ next_path_comb $\cdot$ prev_path_comb
                end for
            else
                result += curr_path_comb
            end if
        end for
    end for

    return result
end procedure
```

All the procedures to calculate the reliability of Cake graphs are implemented in the class *ModuleCake* from the *atr* library which also contains the functions described by the pseudocodes 1 and 2. Such functions are private, and are handled by the *calculate()* function of the module. The identification process to determine if the given network is from the Cake family, starts with some conditions. The graph must be simple (without multi-edges), the degree of its nodes $o(n_i)$ must range between $1 < deg(n_i) < 4$ and at least one of its degrees must be $deg(n_i) = 3$ (at least must contain one chord). If the graph meets the requirements, the algorithm proceeds to determine its hamiltonian cycle in order to identify the chords. Finally, the algorithm checks that all the chords are diametral. To know if a chord is diametral, the path in the extracted hamiltonian cycle in which its starting and ending nodes coincide with the chord endpoint nodes must have length $o(G)/2$ (either rounded up or down). As in the previous examples, using this module requires obtaining the network, creating an object instance of the module (*ModuleCake*), identifying the network, and if belongs to the module family, calculating its reliability.

```
g = nx.MultiGraph([(1, 2), (2, 3), (3, 4), (4, 5),
        (5, 6), (6, 7), (7, 8), (8, 1),
        (1, 5), (2, 6), (3, 7), (4, 8)])

cakemodule = atr.relmodules.ModuleCake(g)

if cakemodule.identify():
    rel = cakemodule.calculate()
    print(atr.graphtools.polynomial2binomial(rel))
```

The resulting reliability from the previous example is:

Algorithm 2 Pseudocode to calculate the number of all possible ways to disconnect a network with α 'cuts' by 'cutting' out ('extracting') the chords.

```
procedure ChordCutsAlgorithm($x$, $\alpha$, $c$)
    // $x$: paths of length $x_i$
    // $\alpha$: maximum number of link cuts without
        disconnecting the network
    // $c$: number of chords of the network

    list chord_combs = []
    list all_combs = []

    for $z \leftarrow 1, c$ do
        for $i_{(mod|x|)} \leftarrow 1, c$ do
            curr_comb = $[x_i, x_{i+z}, x_{i+c}, x_{i+c+z}]$
            chord_combs.append(curr_comb)
        end for
    end for
    if $(\alpha - 4) > 0$ then
        for ch_comb←chord_combs do
            rest_combs = $P_{(\alpha-4)}[(x_1, x_2, ..., x_{|x|})$−ch_comb ]
            for rs_comb←rest_combs do
                all_combs.append(ch_comb + rs_comb)
            end for
        end for
    else
        all_combs = chord_combs
    end if
    all_combs ← Delete all duplicate elements

    int result = 0
    for comb←all_combs do
        int mult = 1
        for e←comb do
            mult $\cdot = |x_e|$
        end for
        result $+ = mult$
    end for
    return result
end procedure
```

$R_G(p) = p^{12} + 12p^{11}(1-p) + 66p^{10}(1-p)^2 + 212p^9(1-p)^3 + 409p^8(1-p)^4 + 392p^7(1-p)^5$

*1) **Computational Performance**:* We ran several tests in order to compare the basic *DC* algorithm with our new algorithm in the computation of the reliability for Cake Graphs. As shown in table I, the Cake Module has a **polynomial** computational time compared with the original *DC* which has **exponential** time. For the tests, we used a computer with a core *Intel i5 8th Gen* and 18GB of RAM.

Firstly, to obtain a batch of these networks to compare the computational times, we developed an algorithm that constructs random Cake graphs. Basically random networks consist of a certain number of nodes connected with some probability model. For instance, the well known *Barabási-Albert model* produces random networks with a power law degree distribution (scale-free networks), that is, the probability that a node in the network interacts with k other vertices follows a power law. Nevertheless, most of the random network models produce non-hamiltonian networks. In order to construct a random hamiltonian network of the family of Cake graphs, we start with a cycle network and we add randomly a fixed number of chords as shown in *algorithm 3*. Note that the algorithm assumes that the number of chords its less or equal to half of the nodes ($c \leq n/2$).

Algorithm 3 Pseudocode to generate random hamiltonian Cake networks.

```
procedure RandomCakeAlgorithm($n$, $c$)
    // $n$: number of nodes of the network
    // $c$: number of chords of the network

    cycle = generate $C_n$
    mid = truncate($n/2$)
    node_list = cycle.nodes()[1:mid]

    while $c > 0$ do
        random_node = random_choice(node_list)
        cycle.add_edge(random_node, random_node + mid)
        node_list.remove(random_node)
        $c = c - 1$
    end while
    return cycle
end procedure
```

For our tests, we have created random cake graphs from 15 to 60 nodes with 3 and 6 chords. The time reduction between the original *DC* and the Cake module, has a mean with all the tests of 99.99%. This improvement increases as more nodes and links the network has (Table I).

V. Expanding the **ATR** library

The library is designed with high modularity to facilitate the implementation of new algorithms and methodologies without modifying the source code. Therefore, the new functionalities can be added directly in form of classes or functions and the library will automatically detect and manage them. In the next sub-sections, we detail the two main functionalities that can be added, which are: Methodologies that specify which edge is selected in each iteration of our new *DC* algorithm and, modules that identify and calculate the reliability of specific families of graphs.

A. Expanding Edge Selection Methodologies

The class *EdgeSelector* is responsible for providing the functions that choose the edges which will be contracted and deleted in each iteration of the *DC* algorithm. To ease the addition of new functions, the network is present in the class structure (is a global parameter) and can be accessed with the

TABLE I
RANDOM CAKE NETWORK COMPUTATIONAL TIMES (IN SECONDS) FOR DIFFERENT AMOUNT OF NODES. THE FIRST TABLE SHOWS THE CASE FOR 3 CHORDS, MEANWHILE THE SECOND ONE DEPICT RANDOM NETWORKS WITH 6 CHORDS.

	3 Chords Random Cake Networks		
n	**Original *DC***	**Cake Module**	**Time reduction**
15	3.1486	0.0105	99.9966%
20	10.5446	0.0136	99.8710%
25	27.9091	0.0171	99.9387%
30	61.7438	0.0214	99.9653%
35	130.2638	0.0289	99.9778%
40	235.9418	0.0299	99.9873%
45	426.1442	0.0362	99.9915%
50	710.5972	0.0429	99.9939%
55	1111.4745	0.0528	99.9952%
60	1729.5110	0.0590	99.9965%

	6 Chords Random Cake Networks		
n	**Original *DC***	**Cake Module**	**Time reduction**
15	40.7586	0.0412	99.8989%
20	205.4605	0.0520	99.9746%
25	854.5891	0.0740	99.9913%
30	2621.3500	0.0828	99.9968%
35	7652.6396	0.0948	99.9987%
40	17959.6568	0.1079	99.9993%
45	41305.4407	0.1266	99.9997%
50	100657.7801	0.1558	99.9998%
55	197461.1199	0.1687	99.9999%

python call *self.g*, therefore the new functions don't need to contain any arguments (apart of *self*). There are two ways to enhance this class with new methodologies: The first way is to add the new function directly in the class as shown in algorithm 4.

Algorithm 4 Modify EdgeSelector class to add a new method

```
class EdgeSelector:
    # Class Code

    def new_selector(self):
        graph = self.g
        # Code
        return edge_id
```

The other option is to add the new function outside the class, this means to create a function and, afterwards, add it to the class *EdgeSelector* with the python command *setattr()* as shown in algorithm 5

Algorithm 5 Dynamically add a methods to EdgeSelector class

```
def new_selector(self):
    graph = self.g
    # Code
    return edge_id

setattr(atr.selector.EdgeSelector, 'new_selector',
        new_selector)
```

Our package automatically detects all newly added functions to the class *EdgeSelector* (either added directly or dynamically). Where the user just needs to indicate which selector wants to use through a string in the arguments of a new *atr* object (see section III) as shown below:

```
atr.calculate_reliability(g, edge_selector = 'new_selector')
```

B. Adding New Modules

The library can be expanded by adding new modules that calculate the all-terminal reliability polynomial for specific families of networks. A module is a class that is derived (inherits) from the base class *RelModule*. All this classes must have a constructor and at least two functions which are *identify()* and *calculate()*. Where the function *identify()* contains the methodologies that identify if the given network belongs to the module family and, the function *calculate()*, calculates the given network reliability. Regarding the chosen module name, it hasn't any restrictions except to be different from the other module names. Note that the file containing the implementation of the module must be in the same folder as the other module files. The last step in the creation of the module is to index it in the *__init__.py* file from the folder *relmodules*. Next, we show the basic structure of a new module and how can be indexed.

```
from relmodules.reliability_modules import RelModule
import networkx as nx

class NewModule(RelModule):
    def __init__(self, g):
        self.g = g

    def identify(self):
        return False

    def calculate(self):
        return 0
```

```
# file 'relmodules/__init__.py'

from atr.relmodules.reliability_modules import
    RelModule
from atr.relmodules.new_module_file_name import
    NewModule
```

The *atr* function *calculate_reliability()* detects automatically any new modules and, in order to call them, the function only needs an string with the module name in the attribute *modules* as described in section III-A.

VI. CONCLUSIONS

In this paper, we present an improved new version of the *DC* algorithm along with a series of methodologies (modules) that work altogether to speed up the computation of the reliability polynomial of any undirected network. The algorithm has been implemented in form of a python package called *atr* which has been designed to be easy to use, modular, highly scalable, and flexible. We detail its implementation plus use-case examples. Moreover, we also show how the new *DC* algorithm can be expanded to include new modules and algorithms to reduce its computational time. Due to the flexibility of the algorithm, the variability of the computational time to calculate the reliability highly depends on the given network and the modules selected. Therefore, choosing or adding the right combination of modules can reduce significantly the computational time compared with the original *DC* algorithm. This said, we believe that our tool can be extended to all the areas which work with networks since the package can deal with general graphs and also be easily personalized and optimized to work with more specific types of networks.

VII. Future research lines

This work provides a base and methodologies to easily expand the new *DC* algorithm. For instance, due to the modular nature of the improved *DC*, it's easy to include new modules that work with specific families of graphs and new methods to selectively choose the best edges to contract/delete in each iteration of the algorithm. Therefore, it's a foundation that can be used by any field that works with network reliability and also can be expanded to fit the specific needs of the areas.

Acknowledgements: This work was partly supported by the Spanish Government through PID2020-115442RB-I00 from MCIN/AEI/10.13039/501100011033 project.

References

[1] E. Moore and C. Shannon, "Reliable circuits using less reliable relays," *Journal of the Franklin Institute*, vol. 262, no. 3, pp. 191 – 208, 1956.

[2] A. Majeed and I. Rauf, "Graph theory: A comprehensive survey about graph theory applications in computer science and social networks," *Inventions*, vol. 5, no. 1, 2020. [Online]. Available: https://www.mdpi.com/2411-5134/5/1/10

[3] O. Mason and M. Verwoerd, "Graph theory and networks in biology," *IET Systems Biology*, vol. 1, pp. 89–119(30), March 2007. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/iet-syb_20060038

[4] R. García-Domenech, J. Gálvez, J. V. de Julián-Ortiz, and L. Pogliani, "Some new trends in chemical graph theory," *Chemical Reviews*, vol. 108, no. 3, pp. 1127–1169, 2008, pMID: 18302420. [Online]. Available: https://doi.org/10.1021/cr0780006

[5] F. Dörfler, J. W. Simpson-Porco, and F. Bullo, "Electrical networks and algebraic graph theory: Models, properties, and applications," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 977–1005, 2018.

[6] D. Nguyen, B. Vo, and D.-L. Vu, "A parallel strategy for the logical-probabilistic calculus-based method to calculate two-terminal reliability," *Quality and Reliability Engineering International*, vol. 32, no. 7, pp. 2313–2327, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/qre.1937

[7] R. Mishra, M. A. Saifi, and S. K. Chaturvedi, "Enumeration of minimal cutsets for directed networks with comparative reliability study for paths or cuts," *Quality and Reliability Engineering International*, vol. 32, no. 2, pp. 555–565, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/qre.1772

[8] S. Rajkumar and N. K. Goyal, "Reliability analysis of multistage interconnection networks," *Quality and Reliability Engineering International*, vol. 32, no. 8, pp. 3051–3065, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/qre.1941

[9] H. Perez-Roses, "Sixty years of network reliability," *Mathematics in Computer Science*, vol. 12, no. 3, pp. 275–293, 6 2018.

[10] T. Politof and A. Satyanarayana, "Efficient algorithms for reliability analysis of planar networks - a survey," *IEEE Transactions on Reliability*, vol. 35, no. 3, pp. 252–259, 1986.

[11] A. Satyanarayana and R. Tindell, "Efficient algorithms for the evaluation of planar network reliability," DTIC Document, Tech. Rep., 1993. [Online]. Available: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA263602

[12] M. O. Ball and J. S. Provan, "Calculating bounds on reachability and connectedness in stochastic networks," *Networks*, vol. 13, no. 2, pp. 253–278, 1983.

[13] R. J. Gould, "Advances on the hamiltonian problem - a survey," *Graphs and Combinatorics*, vol. 19, pp. 7–52, 2003.

[14] T. Politof and A. Satyanarayana, "A linear-time algorithm to compute the reliability of planar cube-free networks," *IEEE Transactions on Reliability*, vol. 39, no. 5, pp. 557–563, 1990.

[15] A. Satyanarayana and M. K. Chang, "Network reliability and the factoring theorem," *Networks*, vol. 13, no. 1, pp. 107–120, 1983.

[16] J. M. Burgos and F. R. Amoza, "Factorization of network reliability with perfect nodes i: Introduction and statements," *Discrete Applied Mathematics*, vol. 198, pp. 82 – 90, 2016.

[17] J. M. Burgos, "Factorization of network reliability with perfect nodes ii: Connectivity matrix," *Discrete Applied Mathematics*, vol. 198, pp. 91 – 100, 2016.

[18] A. Satyanarayana and R. K. Wood, "A linear-time algorithm for computing k-terminal reliability in series-parallel networks," *SIAM Journal on Computing*, vol. 14, no. 4, pp. 818–832, 1985.

[19] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[20] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh *et al.*, "Sympy: symbolic computing in python," *PeerJ Computer Science*, vol. 3, p. e103, 2017.

[21] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[22] P. Llagostera, N. López, and C. Comas, "Network reliability in hamiltonian graphs," *Discrete Optimization*, vol. 41, p. 100645, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1572528621000244

Chapter 5

Spatial Networks with Point Processes

5.1 Overview

A spatial network (or geometric graph) is a representation where its vertices or edges model geometric elements, and therefore they are bound to a measurement system. Such systems can vary depending on the structure type that the network is modeling. For instance, in the case of a road structure (which is considered in this thesis), the used systems are the metric system and the geographic coordinate system. In this spatial linear structure, several events of interest can occur, for example, traffic accidents, crimes, and traffic jams. These collections of points are usually called point patterns and they are generated by a stochastic mechanism (usually called a spatial point process) on a linear networks. Several statistical tools can be considered to analyze the structure of such point occurrences and to visualize their spatial configurations; this can be crucial to understand such spatial configurations and predicting future occurrences. Moreover, based on these point occurrences, we can find optimal short paths between two points on a network (origin and destination), where some point occurrences can be chosen to be avoided or vice versa.

5.1.1 State of Art

The last decade has experienced an increase of interest in the spatial analysis of point patterns occurring on network structures. This raise of interest is due to, among other things, the creation of new network-based fields, the growing availability of datasets, and the necessity of interrelating and analyzing the processes occurring on networks. Among the various studies regarding this topic, we can note Okabe et al. [35] which proposed four statistical methods to analyze the distribution of points on a network, Okabe and Yamada [36] generalized the Ripley's K-function [44] to the network domain, Ang et al. [1] were the first to develop a geometry corrected Ripley's K function that provides a better understanding of network-based event configurations, while Rasmussen and Christensen [42] extended the analysis of point processes to directed linear networks. Recently, Baddeley et al. [2] presented a review of the literature related to the analysis of point patterns occurring over networks. Now we present two studies; the first work is devoted to the analysis of weighted networks based on wildlife-vehicle collisions (point patterns) to model traffic safety, while the second work details a tool in the form of an ***R*** package that provides tools to analyze the intensity of spatial point patterns occurring over different network structures.

5.2 Article: ‘Modeling road traffic safety based on point patterns of wildlife-vehicle collisions’

This article has been published in the journal Science of the Total Environment, belonging to the first quartile (Q1) in the area of environmental science as classified by Journal Citation Reports (JCR) and Scimago Journal Rank (SJR). See Table 5.1.

Table 5.1: Journal metrics corresponding to the journal Science of the Total Environment for the year 2021

Journal Metric	Value
Citescore	14.1
Impact Factor (JCR)	10.753
SJR	1.806
SNIP	2.17

The motivation of the article *‘Modeling road traffic safety based on point patterns of wildlife-vehicle collisions’* is to develop a general flexible framework to consider weighted graphs based on point patterns to obtain optical paths between points (origin and destination) on networks. Here, we consider the problem of finding optimal paths in terms of the intensity of wildlife-vehicle collisions (WVC) (a road safety problem). This approach simplifies real complex road structures into a mathematical linear network and considers different variables to calculate optimal routes between two points (origin and destination) assuming several restrictions. For this problem, we considered real data regarding a square area of 40 $\times$ 40 km around the city of Lleida, Spain. This work analyses the resulting graph model of the road network contained in this area (459050 km of roads), for several road categories, namely, highways, paved roads, and city roads. Then, based on this network structure, we obtain a weithed network structure based on the wildlife-vehicle collisions (as a point pattern) during the period $2010-2014$, the traffic density during the years $2014-2015$, the speed limits of the roads, and the spring vegetation density near the roads. Our new approach uses these variables (road types, WVCs, traffic density, speed limits, and vegetation density) to calculate weights for each road segment which is used to determine the safety of the road and the resulting optimal paths. The flexibility of the framework lies in the possibility of including or excluding variables, for example, centering the results on one or multiple variables, comparing different paths from different model restrictions, or the possibility to adapt the framework to other different problems. We have also developed an interactive application based on ***Shiny*** for the ***R*** language that provides a visual representation of the methodology used in this study. The main points of this research are:

- Create a flexible framework to calculate optimal paths depending on the network characteristics (weighed networks).
- Analysis of a real data set regarding an area of 40×40 km around the city of Lleida, Spain.
- Model the 459050 km of roads contained in this area.
- Identify different variables that affect the wildlife-vehicle collisions in a road network.
- Calculate optimal paths between pair of towns regarding variables such as road types, WVCs, traffic density, speed limits, and vegetation density.
- Produce an interactive app to visually see the optimal paths depending on the given covariates as well as the heatmaps generated by such covariates.

Where the ***Shiny*** application can be found following the next URL:

- GitHub: `https://github.com/LlagosteraPol/ATR`

Science of the Total Environment 846 (2022) 157237

Contents lists available at ScienceDirect

Science of the Total Environment

journal homepage: www.elsevier.com/locate/scitotenv

Modeling road traffic safety based on point patterns of wildlife-vehicle collisions

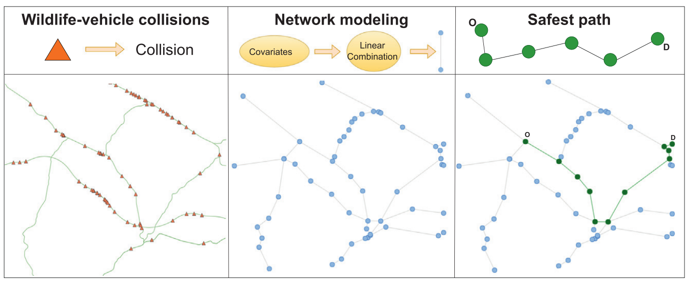
P. Llagostera [*], C. Comas, N. López

Department of Mathematics, Universitat de Lleida, St/Jaume II, 69, Lleida 25001, Spain

HIGHLIGHTS

- To reduce the impacts of wildlife-vehicle collisions a modeling framework is defined.
- Weights are based on the intensity of wildlife-vehicle collisions (point pattern).
- The optimal road path between two points ndeparture and destination) is based on a safety criterion.
- The model is tested on a real data set containing 491 wildlife-vehicle collisions.

GRAPHICAL ABSTRACT



ARTICLE INFO

Editor: Pavlos Kassomenos

Keywords:
Linear network
Road ecology
Road traffic safety
Road transport
Spatial models
Spatial point patterns

ABSTRACT

Wildlife-vehicle collisions represent one of the main coexistence problems that appear between human populations and the environment. In general terms, this affects road safety, wildlife management, and the building of road infrastructures. These accidents are a great danger to the life and safety of car drivers, cause property damage to vehicles, and affect wildlife populations. In this work, we develop a new approach based on algorithms used to obtain minimum paths between vertices in weighted networks to get the optimal (safest) route between two points (departure and destination points) in a road structure based on wildlife-vehicle collision point patterns together with other road variables such as traffic volume (traffic flow information), road speed limits, and vegetation density around roads. For this purpose, we have adapted the road structure into a mathematical linear network as described in the field of Graph Theory and added weights to each linear segment based on the intensity of accidents. Then, the resulting network structure allows us to consider some graph theory methodologies to manipulate and apply different calculations to analyze the network. This new approach has been illustrated with a real data set involving the locations of 491 wildlife-vehicle collisions in a square region (40 km × 40 km) around the city of Lleida, during the period 2010–2014, in the region of Catalonia, North-East of Spain. Our results show the usefulness of our new approach to model road traffic safety based on point patterns of wildlife-vehicle collisions. As such, optimal path selection on linear networks based on wildlife-vehicle collisions can be considered to find the safest path between two pairs of points, avoiding more dangerous routes and even routes containing hotspots of accidents.

1. Introduction

Wildlife-vehicle collisions (WVC) represent one of the main coexistence problems that appear between human populations and the environment (Mo et al., 2017). In general terms, this affects road safety, wildlife management and the building of road infrastructures (van der Ree et al., 2015).

* Corresponding author.
E-mail address: pol.llagostera@udl.cat (P. Llagostera).

http://dx.doi.org/10.1016/j.scitotenv.2022.157237
Received 9 February 2022; Received in revised form 16 June 2022; Accepted 4 July 2022
Available online 8 July 2022

These accidents are a great danger to the life and safety of car drivers, cause property damage to vehicles (Díaz-Varela et al., 2011; Groot Bruinderink and Hazebroek, 1996), and are a real peril to wildlife populations (Hilário et al., 2021; Coffin, 2007). For instance, in 2017 in Spain, traffic accidents were the fourth external cause of death behind suicides, drownings and accidental falls (Press release of the INE, October 2018). Moreover, in 2018 there were 102,299 traffic accidents with victims (1679 of them with fatalities) of which, at least 403, were caused by wildlife-vehicle collisions, with 6 fatalities (Anuario-DGT, 2018). These figures highlight the importance and the severity of these types of accidents. Sàenz-de Santa-María and Tellería (2015) found that 8.9 % of the accidents that happened in Spain between 2006 and 2012 (74,600 accidents in total) are related to fauna. These authors also pointed the spatial inhomogeneous distribution of these events, reaching $30 - 50\%$ in some of the northern mountainous regions. The expanding populations of wild boars (*Sus scrofa*) and roe deer (*Capreolus capreolus*), caused 79 % of the collisions. Therefore, it should be a priority to evaluate and describe the factors affecting these road collisions in order to determine effective measures to mitigate or eradicate them (Lord and Mannering, 2010).

As WVC do not occur randomly neither in space nor in time, as they have a clear space-time component, during the last few decades there has been a growing interest in the description and the modeling of such type of events (Gunson et al., 2011), especially for identifying areas with a high amount of accidents (hot spots) (Litvaitis and Tash, 2008; Ramp et al., 2005). This interest has been promoted by a greater availability of this type of data, the definition of new statistical techniques and methodologies based on geographic information systems (GIS) and the use of global positioning tools such as the GPS. In fact, the analysis of such space-time structures, implies the analysis of point locations (events) distributed on linear structures (roads). In this context, roadkills can be assumed as points (events) occurring on a linear structure and evolving through time. Other examples of similar linear spatial configurations include the spatial distribution of some invasive plant species (Spooner et al., 2004; Deckers et al., 2005), street crimes (Ang et al., 2012) and, in general, traffic accidents (Yamada and Thill, 2004; Xie and Yan, 2008). Note that for all these examples, point patterns occur on linear structures, and it is not expected that an event lies out of one of these linear configurations. As such, the analysis of these spatial configurations is focused on the description of the spatial configuration of points assuming that the whole point pattern is placed over the linear network. With this in mind, several counterpart versions of statistical tools derived from point process theory have been proposed to analyze such spatial point configurations. For instance, the (empirical) network K-function presented by Okabe and Yamada (2001) is a modification of Ripley's K-function (Ripley, 1977) to analyze the spatial structure of point patterns on a linear network. And the geometrically corrected version of this function proposed by Ang et al. (2012), given that the function defined by Okabe and Yamada (2001) depends on the geometry of the network. Baddeley et al. (2021) provide a review of methods and theory of point process on linear networks.

The occurrence of WVC are affected by several biological, ecological and meteorological covariates together with some structural road characteristics. For instance, Seo et al. (2013) analyzed the influence of landscape structure and seasonal changes on WVC, Keken et al. (2019) evaluate the roadside vegetation and the presence of hotspots of WVC, Borkovcová et al. (2012) analyze the effect of the type of road on roadkills, Al-harbi et al. (2012) considered the impact of meteorological conditions on road traffic accidents, whilst Ha and Shilling (2018) used the ecological model Maxent, described by Elith et al. (2011), to model potential WVC locations considering environmental factors and human population density. Moreover, the occurrence of WVC also depends on three important factors, i.e. the density of animals near the road, animal propensity to enter (cross) a road, and traffic volume. In fact, the risk of having a WVC depends on these three elements. A high traffic volume road, with a low density of animals near this road, results in a road with a low risk of having a WVC. Similarly, if the density of animals near the road is high, but the traffic volume is low, the risk of having a WVC continue being low. However, if the traffic volume is increased then the risk of having a WVC could also increase. Thus, it is clearly necessary to consider traffic flow information when identifying optimal roads in terms of the risk of having a WVC.

Moreover, we consider vegetation density as a good measure to discern the probability of animal road crossing and the density of animals near the roads. This is so since, one may expected high densities of animals near areas with abundant vegetation. Here we used the satellite 'normalized difference vegetation index (NDVI)' as an index of vegetation abundance. In particular, we consider data from the spring of 2017 provided by the Government of Catalonia (first year with available data). Note that this normalized index ranges in [0,1], providing 5 classes of vegetation densities. We can assume that the vegetation is scarce, low, medium, high and very abundant if this index ranges in [0, 0.2), [0.2, 0.4), [0.4, 0.6), [0.6, 0.8) and [0.8, 1], respectively. Finally, we also consider the travel speed limit of each road as a factor of road safety, assuming that on roads with higher speed limits are more likely to have a WVC. With this purpose, we added the speed limits for each road based on three categories; urban road (50 km/h), country road (90 km/h), and highway (120 km/h).

Consequently, distinct road sections will have distinct probabilities of having a WVC in terms of the covariates surrounding this road area (vegetation density, for instance), the traffic volume of this road section and its speed limit. Borrajo et al. (2021) model the point intensity of roadkills based on several landscape covariates. This suggests the possibility of considering weighted graphs to model the risk of WVC, and to define distinct path configurations to optimize this risk. Weighted graphs, based on WVC information, could be considered to find the safest road trip between two points (a departure and a destination point). Indeed, various network properties and indicators can be evaluated using Graph Theory methodologies. Several authors have already considered graph theory to model path configurations on linear networks. Quintero et al. (2012) used graph theory-based transit indicators to develop road safety models for transit infrastructure, transit network topology, transit route design, and transit performance and operations. Whilst, Xu and Chen (2003) used different shortest path algorithms to identify associations in criminal networks, and Chen et al. (2015) used a shortest path approach to discover potential tumor suppressor genes to help with the design of effective treatments against cancer.

Therefore, our main aim in this paper is to develop a new model framework adapting algorithms used to obtain minimum paths between vertices in weighted networks to model road traffic safety based on wildlife-vehicle collision point patterns, covariates related to vegetation density together with other road variables related to traffic flow information. In this new approach, we consider minimum-weight path finder algorithms, for instance, the Dijkstra algorithm, the Bellman-Ford algorithm, the Floyd-Warshall approach (Magzhan and Jani, 2013) and the Johnson algorithm (Johnson, 1977), to model road safety based on the intensity of roadkills. For this propose, we adapt the road structure into a mathematical linear network as described in the field of Graph Theory, adding weights to each linear segment based on the intensity of accidents. In this way, the resulting network structure allow us to consider some graph theory methodologies to manipulate and apply different calculations to analyze weighed network. Finally, we applied our new approach to analyze a dataset containing 491 wildlife-vehicle collisions occurred in a squared area (40 km × 40 km) during the period 2010–2014 around the city of Lleida, in the region of Catalonia, North-East of Spain.

The plan of the paper is the following. In Section 2 we present a real dataset of wildlife-vehicle collisions. Some definitions and preliminaries of spatial point processes on linear networks are provided in Section 3. In Section 4 we develop a model for road traffic safety based on point patterns over linear networks. We analyze the dataset and present some results in Section 5. And finally, the paper ends with a summary section.

2. Wildlife-vehicle collisions on a road network

We analyze the spatial structure of a dataset containing 491 wildlife-vehicle collisions occurred in a squared area (40 km × 40 km) around the city of Lleida in the region of Catalonia, North-East of Spain (see

Fig. 1). This study area involves 459.050 km of roads for three distinct road categories, namely, highways and paved roads during the period 2010–2014, together with the daily average traffic volume based on 365 days of the years 2014–2015 for each road section. Note that this squared area comprises a wide range of landscape structures, including riparian forests, crops fields and some densely populated areas (for instance, the metropolitan area of Lleida). Moreover, most of the wildlife-vehicle collisions involve ungulates (33.22 %), and non-identified mammals (66.78 %). For the ungulates, roadkills comprise only two species namely, wild boar (*Sus scrofa*) and roe deer (*Capreolus capreolus*), and no extra information is given for the non-identified mammals.

The Department of Territory and Sustainability of the Autonomic Government of Catalonia (https://web.gencat.cat) provided the roadkill records, the traffic flow information (in particular, daily average traffic volume and speed limit), the road structure for the study, and the vegetation density. The Mossos d'Esquadra, the autonomous police force of Catalonia, recorded all the wildlife-vehicle collisions including the location, the date and the type of animal involved in the accident.

A first inspection of Fig. 1 reveals that points are not distributed homogeneously on the network configuration, suggesting the presence of areas with a higher concentration of roadkills, and other areas with a scarce presence of accidents. So, this first visual analysis reinforces our idea that it is possible to find optimal road paths regarding the risk of having a wildlife-vehicle collision.

3. Linear networks and point processes

A road network is a complex structured formed by line segments and curves. A tentative way to represent these complex structures (geometric graph networks) is to consider linear networks. We define a line segment in the plane with endpoints u and v as $[u,v] = \{tu + (1 - t)v : 0 \leq t \leq 1\}$ (Ang et al., 2012). Then a linear network L is the union $L = \cup_{h=1}^{s} l_h$ of a finite collection of line segments $l_1, \ldots, l_s$ in the plane, with a total length $|L|$. We can also consider L as planar graph, defined by a set of vertices $v_1, \ldots, v_m$ and edges $e_{ij} = [v_i, v_j]$. Note that the intersection of two distinct edges contains at most one point.

Furthermore, a *path* between two points u and v is considered as a sequence $x_0, x_1, \ldots, x_m$ in L such that $x_0 = u$, $x_m = v$ and $[x_i, x_{i+1}] \subset L$, for $i = 0, \ldots, m-1$. Then, we define the length of this path $x_0, x_1, \ldots, x_m$ as $\sum_{i=0}^{m-1} \| x_{i+1} - x_i \|$ where $\| \cdot \|$ is the Euclidean distance. Finally, we define the ***shortest path distance*** $d_L(u,v)$ between two points u and v in L as the minimum of the lengths of all the paths from u to v. Note that for convenience, we set $d_L(u,v) = \infty$ if there no path between these two points, and so these points are placed in disconnected portions of L.

Now we consider a point process $\boldsymbol{X}$ on a linear network L to be a stochastic mechanism that generates a countable and finite set of events x_i, $i = 1, \ldots, n$, where n is not fixed in advance over a linear network L. Also, we assume that X is simple. In particular, we consider the Poisson point process with point intensity $\lambda > 0$, where the number of points falling in any line segment $B \in L$ has a Poisson distribution with mean $\lambda | B|$. Moreover, events falling in disjoint line segments $B_1, \ldots, B_s \in L$ are independent. Stochastic realizations of a point process X will be denoted as x.

3.1. Point intensity estimation on liner networks; modeling roadkills

This section is devoted to model the roadkill intensity based on kernel functions. Our intention is to work with the intensity of collisions to obtain weights related to each linear segment of the road network to assess the risk of having a wildlife-vehicle collision. These weights should be considered to determine paths on the network structure satisfying some conditions. The intensity or rate function $\lambda(u)$, for $u \in L$, of a point process related to roadkills is the spatially-varying expected number of random points per unit length of network, and it is defined via

$$E[N(\mathbf{X} \cap B)] = \int_B \lambda(u) du \qquad (1)$$

for all interval $B \in L$, where du denotes integration with respect to arc length along the network, and $N(\cdot)$ is a counting measure, which provides the number of points of X in B. Note that it would be possible to work also with the related probability density function $f(u)$, though for this work we shall just consider the rate or intensity function.

Let us now obtain an estimator of the rate or intensity function. A kernel estimator of this intensity function can be defined as

$$\hat{\lambda}(u) = \sum_{i=1}^{n} \kappa(u|x_i), u \in L$$

where $\kappa(u| v)$ is the kernel function. Various authors have considered a kernel estimator to estimate the intensity function on a network structure (Bailey and Gatrell, 1995; Okabe et al., 2009), and some of these kernel estimators have been also used to estimate the intensity of roadkills (Krisp and DurotJohnson, 2007; Morelle et al., 2013). Here we assume the diffusion estimator $\widehat{\lambda}^{H}(u)$ proposed by McSwiggan et al. (2016) and implemented in the Spatstat R package (Baddeley et al., 2015) as a tentative kernel point intensity estimator,

$$\hat{\lambda}_h(u) = \hat{\lambda}_h(u|\mathbf{x}, h) = \sum_{i=1}^{n} \kappa_t(u|x_i), u \in L \qquad (2)$$

where $\kappa_t(u| x_i)$ is the heat kernel, being the probability density at u of the location, at time t, of a particle which executes Brownian diffusion on the

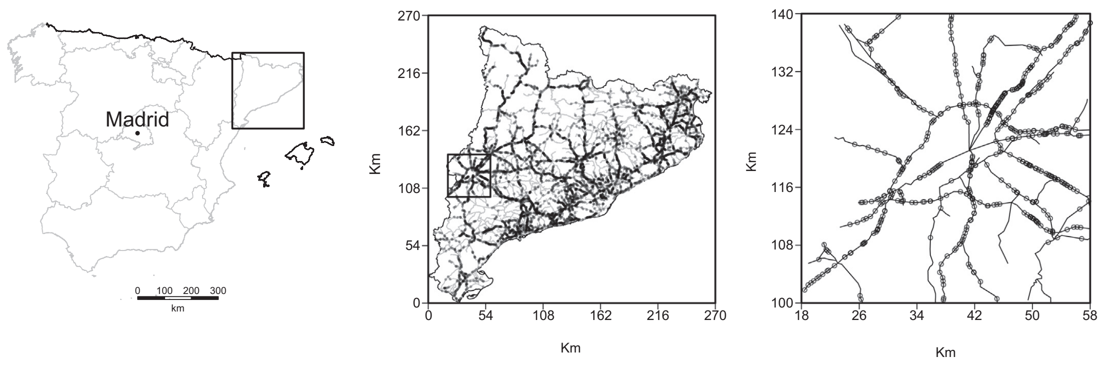


Fig. 1. Location of the study area (Left panel) together with the location of 6590 roadkills during the period 2010–2014 and the underlying road network in Catalonia (North-East of Spain), given in km. (Central panel), and a magnification of the study region around the city of Lleida (40 km × 40 km) with the location of 491 roadkills (Right panel).

network and which started at time $t = 0$ at location v. Here, $h > 0$ is the smoothing bandwidth and $t = h^2$.

3.2. The road network

A road is a complex structure formed by line segments and curves. Usually, this complex structure is defined by a set of line segments that represents all the complex configurations of a road, resulting in a configuration of vertices and edges to model this structure. Usually, the number of vertices and edges, and their structure is given and represent the linear road configuration. One problem that arises from the window of observation and the structure of the linear network are the crossing points between roads (line segments) and the edges of the observation window. In fact, these points are artificial endpoints of roads (one may expect that roads continue outside the observation window), and for convenience in our graph analysis, we shall consider them as vertices. Thus, the resulting undirected weighted graph is then defined by 410 vertices and 437 edges, by maintaining the inherent road structure (see Fig. 2).

4. Modeling road safety

A first step to model road safety based on point patterns over linear networks is to assign to each linear segment of the road network a weight related to the risk of having a wildlife-vehicle collision. These weight should be considered to determine paths on the network structure satisfying some conditions. Here, these segment weights are based on the intensity of roadkill collisions. Then we define the assigned weight value for each segment via

$$\Lambda(l_i) = E[N(\mathbf{X} \cap l_i)] = \int_{l_i} \lambda(u)du \tag{3}$$

where $\Lambda(l_i)$ is the intensity counting measure defined for the linear segment l_i, $i = 1, \ldots, s$. This value will be considered as the weight for each line segment to evaluate the risk of having a wildlife-vehicle collision. In the case of a constant point intensity, longer line segments are expected to have more accidents than shorter ones, and therefore, the shortest path between two

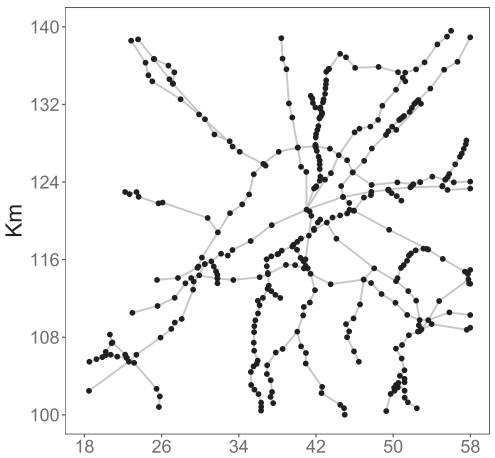


Fig. 2. Study region with the resulting network structure formed by 410 vertices (black points) and 437 edges.

prescribed points on L will result also in the safest one. Here, we consider the diffusion estimator $\hat{\lambda}^H(u)$ (2) as a tentative point intensity estimator.

4.1. Assuming several weight types

So far we have considered optimal paths, and therefore road safety, based on the point intensity of accidents. However, others criteria can be involved in finding optimal paths between points on L. Other important criteria to decide which path to choose between two points on L are the traffic flow information (daily average traffic volume and speed limit) and vegetation density which should be incorporated to full understand this decision analysis. In general, a tentative way to combine several distinct criteria (for instance, intensity of accidents, traffic volume along the network and road speed), is to consider a linear combination of these elements via

$$W(l_i) = a\frac{\Lambda(l_i) - \min(\Lambda(l))}{\max(\Lambda(l)) - \min(\Lambda(l))} + \sum_{j=1}^{\nu} b_j \frac{Z_j(l_i) - \min(Z_j(l))}{\max(Z_j(l)) - \min(Z_j(l))} \tag{4}$$

for $i = 1, \ldots, s$, where $W(l_i)$ is the global weight related to segment l_i, $Z_j(l_i)$ is the $j = 1, \ldots, \nu$ variable related to the same segment i, $a + b_1 + \ldots + b_\nu = 1$, and $\max(\Lambda(l))$ and $\min\Lambda(l))$ (say) is the maximum and the minimum value of all $\Lambda(l_i)$, for $i = 1, \ldots, s$. Note that this expression is not affected by the relative scale of each criterion. Then $0 \le W(l_i) \le 1$, for any $i = 1, \ldots, s$. The linear combination of several factors affecting WVC permits to have a better control between weight types, by changing the values of the a and b_j paramenters. Now if $a = 1$, $W(l_i) = (\Lambda(l_i) - \min(\Lambda(l)))/(\max(\Lambda(l)) - \min(\Lambda(l)))$, and therefore, the criterion used to path selection is based on the expected number of wildlife-vehicle collisions, independently of the Z variables. Similarly, if $b_1 = 1$, for $\nu = 1$, $W(l_i) = (Z(l_i) - \min(Z(l)))/(\max(Z(l)) - \min(Z(l)))$, and thus, the path selection is based on a single variable Z_1, independently of the expected number of collisions. Finally, it is possible to consider a criterion that combines several elements. For instance, if we take $a = 0.8$ and $b_1 = 0.2$, for $\nu = 1$, our path selection is mainly based on the intensity of accidents, although variable Z_1 on L is also taken into account.

Let us now consider the traffic volume as a tentative variable together with the intensity of accidents. Under two roads with the same intensity of accident, the road with higher traffic volume is, apparently, safer than that with a lower traffic volume. This is so, since for the same intensity of accidents the road with higher traffic volume results in less accidents per vehicle than that with lower traffic volume. Probably, either the road with higher traffic volume has less density of animals near the road, or else, animals have less propensity to enter (cross) the road. Thus, in our analysis, we shall consider that roads with higher traffic volumes to be safer than those with lower traffic volumes when having a similar intensity of accidents. As such, the traffic volume variable T should be considered in (4) as $Z_1(l_i) = max\,(T(l)) - T(l_i)$, where $T(l_i)$ is the daily average traffic volume (traffic flow information) of the road segment i. Note that now under this variable, for a given accident intensity, roads with higher traffic volume of vehicles are chosen to find the optimal (safer) road path. So we expect large weighted sum values for roads with high accident intensities and with low traffic volumes of vehicles. These roads should be avoid, for safety reasons, since even having a low traffic volume they have a high concentration of roadkills.

Moreover, in the case of road speed limit, we assume that roads with higher speed limits are more likely to have a WVC and so we assume $Z_2(l_i) = S(l_i)$ in (4), where $S(l_i)$ is the speed limit of the road segment i. The density of vegetation is considered in a similarly way, where high densities imply more animal density near and crossing roads, and thus we assume $Z_3(l_i) = V(l_i)$ in (4), where $V(l_i)$ is the vegetation density for segment i. Note that to obtain the vegetation density for each segment, we have crossed the information of the vegetation index (a pixel image) on the roads and obtained an average value of the index for each segment.

4.2. Avoiding paths with larger prescribed weight values (hotspots)

The methodology applied so far prioritizes possible paths between two vertices on L assuming the total sum of weights related to the edges that form each path. However, it might happen that this global criterion does not provide a satisfactory search for the optimal path between points. In fact, an optimal path between two points based on the total sum of weights might contain edges with weights larger than a prescribed weight value. For instance, in our application, this optimal path might contain an edge with a large expected value of wildlife-vehicle collisions (a hotspot). In this case, although the global path is the safest of all the possible paths, the presence of this hotspot of collisions might result in an unsafer path. To avoid these hotspots, we have adapted the DFS algorithm to filter paths that contain edges with weight values larger than a prescribed value.

4.3. Adapting algorithms used to obtain minimum paths between vertices in weighted networks

There are several algorithms used to calculate the minimum path between two vertices in a weighted network, and some of the most used are the Dijkstra algorithm, the Bellman-Ford algorithm, the Floyd-Warshall algorithm and the Johnson algorithm. Usually, shortest path algorithms for weighed graph explore networks in search of the path between two points (usually vertices) that has minimum cost. This cost is defined as the total sum of the weights associated with the edges of each path. Here a path is defined as a set of unique edges that connects distinct vertices starting from a vertex *origin* and ending with a vertex *destination*. The choice of the algorithm to be used depends on the network, for instance, the type of network (direct, undirected or mixed graph), its size (vertices and edges), the nature of the weights (positive, negative or mixed), and the problem under analysis, such as finding the shortest path between two points (the Dijkstra, and the Bellman-Ford and Johnson algorithms) or between each pair of vertices (the Floyd-Warshall algorithm).

In our analysis, we focus our attention on two problems. First, calculate the total number of possible paths between two points (vertices) and, second, rank the top K-best paths between them, based on a given criterion. For the first problem, we used a modified version of the well-known algorithm called Depth First Search (DFS) which has a computational cost of $O(|V| + |E|)$ from the *igraph R* library. This algorithm finds all possible paths by recursively expanding each node in a tree form and going through each branch of the tree until the iteration process reaches: (a) the *destination* vertex; (b) an already explored vertex by a previous iteration; or, (c) a dead-end (a vertex from which it is impossible to reach another one). Moreover, we used the Yen's algorithm (Yen, 1971) to work on the second problem. This is so since this algorithm works with non-negative edges and it is more efficient than the DFS algorithm when k (the rank position of all the possible paths) is less than the total number of possible paths. This algorithm has a cost of $O(kn^3)$ and uses the Dijkstra algorithm to find the best path between two nodes and then it calculates all k - 1 shortest paths.

Our new approach allows to analyze a combination of several variables based on a prescribed criterion to find optimal paths. This can be done by adding data on the edges as weights, and thus resulting in a multi-weight network. In our case, we used accident intensity and the traffic volume as weights but more variables can be considered. This implies to obtain weights related to each edge of the linear structure. If the variable under analysis is a point pattern, then we can consider expression (3) and (4). If the variable is an internal variable of the network, we can obtain its corresponding weights from the network geometry (for instance, the distance between vertices). Other variables such as those related to traffic flow information are usually provided by governmental agencies and should be considered directly as given for each road segment. Then, the DFS and Yen's algorithms are applied along the previous combined data to find the best optimal path (or the best k-optimals) between any two points on the network. Also, we can filter these paths by excluding edges with larger prescribed weight values (hotspots). We have implemented this hotspot filter assuming two different methodologies. First, our algorithm detects all the hotspot edges of the linear network, and then, avoiding these edges, search for the optimal path based on a given criterion. The second methodology filters the paths posterior to the optimal search and then avoids paths containing hotspot edges. Whilst the first methodology is faster, it is less informative than the second one, which is slower but it also gives a list of all the paths that should be avoided (the complete list of paths that contains a hotspot).

5. Analysing the wildlife-vehicle collision dataset

To illustrate the use of our new approach, we consider several scenarios assuming the wildlife-vehicle collision data. Fig. 3 (top, left panel) shows the resulting point density based on the diffusion estimator (2) with a bandwidth value around 750 m chosen to provide a good visual fitting to the point pattern. This figure highlights that visual inspection of this spatial structure reveals points forming aggregations on the road network, thereby suggesting the presence of clusters of wildlife-vehicle collisions probably due to a certain degree of inhomogeneity. The corresponding weighted network structure (see Fig. 3, top, right panel) shows the average number of wildlife-vehicle collisions for each linear segment (3). Moreover, we also consider the daily average traffic volume given by each network segment (Fig. 3, middle, left panel), vegetation density (spring 2017 NDVI) given by each network segment (Fig. 3, middle, right panel) and the road speed limits, also, for each segment (Fig. 3, bottom, left panel). The last panel (Fig. 3, bottom, right) shows the linear combination of the intensity of WVC, traffic volume, vegetation density and road speeds, assuming expression (4).

In particular, we adapt the DFS and the Yen's algorithm to find the safest paths between several pairs of points (origin/destination) based on the weighted graph defined by the average number of wildlife-vehicle collisions together with the traffic volume, vegetation density and road speeds based on a linear combination of these variables assuming expressions (4). To illustrate the use of our new approach, consider two tentative origins/destinations, in this case, real town locations (Vilanova de Segria/Soses, and Alcarras/Castelldans) (see Fig. 4, for town locations). Obviously, we could have considered any pair of vertices located on L.

5.1. Road safety based on the intensity of accidents and the traffic volume

Let now focus our attention on finding the optimal path between pairs of vertices on the road structure based on the linear combination of the intensity of accidents together with variables $Z_j(l_i)$, for $j = 1, \ldots, 3$ assuming (4) for $a = b_1 = b_2 = b_3 = 0.25$, since the four variables are considered to affect similarly the weighed sum. Here, we consider our new approach to find the safest paths between the two pairs of towns Vilanova de Segria/Soses and Alcarras/Castelldans, respectively. For completeness, we consider also the distance between vertices to find the shortest path between these two pairs of towns, and show how these optimal paths can differ when a safety criterion is taken into account. To simplify our notation, we call the criterion that combines intensity of accidents and the three covariates as the safety criterion, and the criterion based on the distance between towns as the path distance criterion. Fig. 4 shows the safest path (therefore based on the safety criterion) between these two pairs of origin/destination points together with the shortest paths between the same pair of points. The safest and the shortest paths are different under both examples (pair of town locations). Thus the safest path between these two pairs of towns is not the same as the resulting shortest paths. In fact, we have found 712.916 and 691.415 possible paths between these pair of towns, respectively. These represent the total number of possible combinations of edges that connects the source and destination nodes (towns) without repeating any node. Note that all these combinations do not depend on the weights associated to each edge, but only on the number of edges that form the graph. Table 1 shows the ranking values for the 10th safest path for these two pairs of origin/destination points. We see that the resulting weighted values for these 10th safest paths for a given pair of towns are very similar. In fact, these weighted values are around 13 and 8 under the

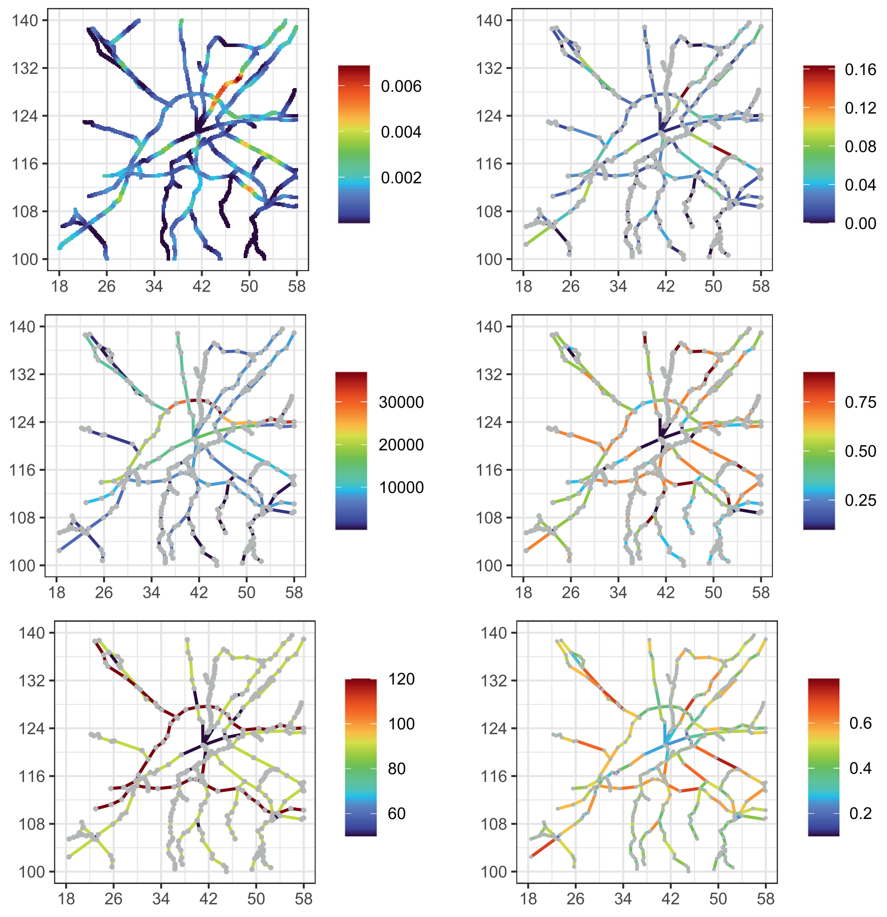


Fig. 3. Resulting point intensity for the roadkill point pattern based on the diffusion estimator (2) with a bandwidth value around 750 m (top, left panel), weighted network structure based on the average number of wildlife-vehicle collisions for each linear segment (top, right panel), daily average traffic volume for each road section (middle, left panel), vegetation density (spring 2017 NDVI) (middle, right panel), road speed limits (bottom, left panel), and the weights of the lineal combination of the variables based on expressions (4) (bottom, right panel) (the scales of the axes are given in km).

Vilanova de Segria/Soses, and the Alcarras/Castelldans pairs of towns, respectively. However, under both pair of towns the 10th shortest paths have a similar distance of around 28 km. thereby suggesting that the safest routes between Vilanova de Segria and Soses are far more dangerous than the safest routes between Alcarras and Castelldans.

5.2. Path selection avoiding wildlife-vehicle collision hot-spots

Let us now consider the road safety analysis avoiding collision hotspots. As such, we assume four distinct prescribed weight values by avoiding paths that contain edges with a weight larger than 20 %, 40 %, 60 % and 80 % of the largest weight value on L. In particular, we combine the intensity of accidents together with three covariates (daily average traffic volume, road speed limit, and vegetation density) assuming expression (4) (i.e. the safety criterion) to obtain the weight for each road segment. Table 2 shows the results of applying this filter for the two pairs of towns, and the four prescribed weight values. When we assume the more restrictive filter (20 %), we found that it is not possible to find a path with a maximum edge weight less than or equal to 20 % of the largest weight on L. So any of all the possible paths will contain an edge with a weight larger than the 20 % of the largest one. Similar results are obtained when we consider the 40 % and 60 % of the largest weight on L. If we relax this condition and we consider the prescribed weight value to be 80 % of the largest weight (we allow paths with larger weight per segment), we have found 1056 and 548 paths applying the safest criterion for Vilanova de Segria/Soses and Alcarras/Castelldans pairs of towns, respectively. Thus for prescribed values larger than 80 % it is possible to find several paths connecting both pairs of towns satisfying this filter condition. Thus under the Vilanova de Segria/Soses pair of towns (say), it is possible to find paths where all the line segments have a weight of less than or equal to $0.8 \times 0.7795114 = 0.6236091$, where 0.7795114 is the largest weight combining both variables for any segment on L. So any path with segments with weight values larger than 0.6236091 are not considered. For this specific case, hotspots are those sections of a road with a weight larger than 0.6236091, and it is possible to find several rout es to avoid such road segments. Note that large weight values usually are related to roads with high accident intensities, low traffic volumes, high road speeds, and high vegetation densities near roads. So we expect that roads with high accident intensities, low traffic volumes, high speeds, and abundant vegetation to be unsafer, and thus these roads (or road portions) should be avoid, for safety reasons.

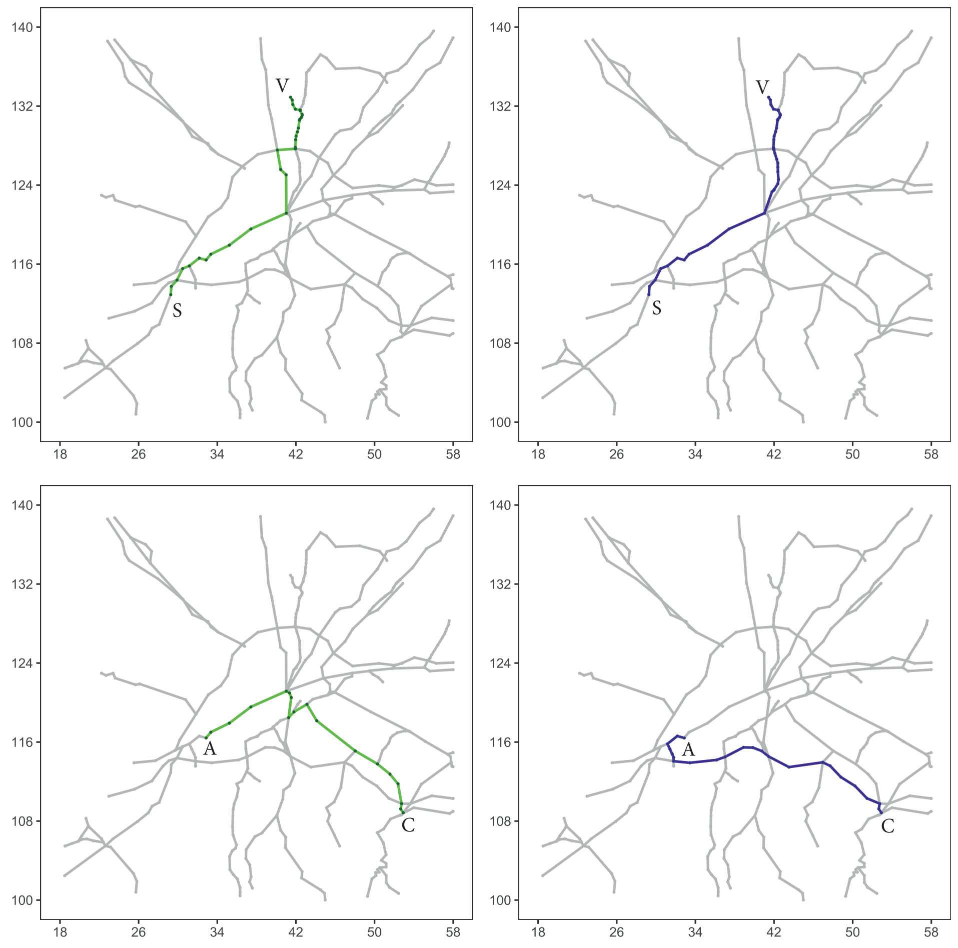


Fig. 4. Linear structure of the study region together with the resulting optimal path for the two pairs of origin/destination points, Vilanova de Segria (V)/Soses (S) (top panels), and Alcarras (A)/Castelldans (C) (bottom panel), under the safety criterion (safest path) (left panels and green lines) and the path distance criterion (shortest path) (right panels and blue lines) (the scales of the axes are given in km.). new figure.

Table 1
Top-10 rating of the safest paths between the towns of Vilanova de Segria and Soses (Vil-Sos), and Alcarras and Castelldans (Alc-Cas), with the total weighted sum based on the linear combination of the intensity of accidents together with the traffic volume (safety criterion) (see expression (4)) per path, and the shortest path distances (path distance criterion).

	Safest		Shortest (km)	
#	Vil-Sos	Alc-Cas	Vil-Sos	Alc-Cas
1	12.77587	7.059108	27.860	27.906
2	13.12468	7.192944	27.861	28.939
3	13.26935	7.611801	27.862	29.802
4	13.31218	7.745637	27.862	29.930
5	13.49028	8.151168	27.862	29.990
6	13.57224	8.366453	27.862	30.283
7	13.61815	8.398381	27.863	30.482
8	13.66098	8.400001	27.864	30.834
9	13.80565	8.48813	27.865	30.962
10	13.83909	8.532217	27.866	31.022

6. Summary

We have proposed a new approach to obtain minimum paths between vertices (optimal path finders) in weighted networks to obtain the safest route between two points (departure and destination points) in a road structure based on wildlife-vehicle collision point patterns, and on three covariates (daily average traffic volume, road speed limit and vegetation density).

Table 2
Number of paths satisfying that none of their segments has a weight value larger than the % of the largest segment weight value under the safety criterion, and for the two pairs of towns.

%	Vilanova de Segria/Soses	Alcarras/Castelldans
20	0	0
40	0	0
60	0	0
80	1056	548

In particular, we have adapted the DFS and the Yen's algorithms to find the optimal routes based on the intensity of roadkills together with these three covariates. For completeness, we have also found optimal routes based on distances between vertices. For this purpose, we have adapted the road structure into a mathematical linear network as described in the field of Graph Theory, adding weights to each linear segment based on a linear combination of the intensity of accidents, the daily average traffic volume, road speed and vegetation density. Then, the resulting network structure allows us to consider some graph theory methodologies to manipulate and apply different calculations to analyze weighted network. This new approach has been illustrated with a real data set involving the locations of 491 WVC in an agriculture square area (40 km × 40 km) around the city of Lleida, during the period 2010–2014, in the region of Catalonia, North-East of Spain, together with the underlying road network of this region.

To illustrate the use of our new approach, we have considered two tentative origins/destinations, in this case, real town locations. Our results show that our new approach ranks all possible paths between these pairs of towns in terms of the weight associated with each edge. We have considered four distinct types of weights associated with each edge, i.e. a linear combination of the average number of WVC together with the daily average traffic volume, road speed and vegetation density, and the edge length. For each type of weight, this new methodology ranks all possible paths between two pairs of vertices on *L*, and provides the optimal path for a given type of weight.

Moreover, we have shown that this methodology also permits to define optimal paths assuming the presence of hot-spots that should be avoid. Our new approach incorporates an algorithm to filter paths containing edges with weight values larger than a prescribed value. In this way, it is possible to optimize path selection avoiding hot-spots of WVC. More insight on optimal path selection on road networks would be definitely valuable for road traffic safety.

An important extension of this work is the definition of other functions, instead of a lineal relationship, to combine distinct variables of interest, and the incorporation of other variables in the optimization process. Although, this model is defined to analyze road traffic safety of vehicles, it would be interesting to adapt our new approach to model also safety of wildlife crossing the road. So in this case, an extension of our new approach might define road safety from two points of view, vehicles and passengers and wildlife crossing roads.

Mention that to support our research and make it accessible to other researchers, we developed an interactive application of our new approach in the R language. We can reproduce the results presented in this work using this new application, and the complete data set can be obtained from the application data files. Moreover, we designed the app to work with any given network if the provided data fills the application requisites. Detailed information about the usage of the app and the app itself, can be found in our GitHub repository https://github.com/LlagosteraPol/OptimalPathApp. This first version of our new application can help to analyze large data set and test our new approach in a real situation.

Finally, we conclude this paper by saying that the analysis of point patterns and linear networks opens up new promising lines of research to analyze point structures that depend on linear networks. In our case, focused on road traffic safety. In particular, the optimal path selection on linear networks based on WVC can be considered to find the safest path between two pairs of points, avoiding more dangerous routes and routes containing hot-spots of accidents.

Data availability

The dataset analyzed during the current study are available in the repository of Autonomic Government of Catalonia (https://web.gencat.cat), and it can be available from this Autonomic Government on a request.

CRediT authorship contribution statement

All authors contributed to the study conception and design. Data collection and data analysis were performed by P. llagostera, *C. Comas* and N. López. The methodological development and software implementation were performed by P. Llagostera and *C. Comas*. The first draft of the manuscript was written by P. Llagostera and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Declaration of competing interest

The authors declare that they have no conflict of interest.

Acknowledgements

The authors acknowledge the support of the MCIN/AEI/10.13039/501100011033 through Grant PID2020-115442RB-I00, from the Spanish Ministry of Science and Innovation.

References

Al-harbi, M., Yassin, M.F., Shams, M.B., 2012. Stochastic modeling of the impact of meteorological conditions on road traffic accidents. Stoch. Env. Res. Risk A. 26.
Ang, W., Baddeley, A., Nair, G., 2012. Geometrically corrected second order analysis of events on a linear network, with applications to ecology and criminology. Scand. J. Stat. 39, 591–617.
Anuario-DGT, 2018. Anuario estadistica de la dirección general de tráfico (dgt) para el año 2018. Ministerio del Interior, Gobierno de España.
Baddeley, A., Rubak, E., Turner, R., 2015. Spatial Point Patterns: Methodology And Applications With R. Chapman and Hall/CRC Press, London.
Baddeley, A., Nair, G., Rakshit, S., McSwiggan, G., Davies, T., 2021. Analysing point patterns on networks — a review. Spat.Stat. 42, 329–350.
Bailey, T., Gatrell, A., 1995. Interactive Spatial Data Analysis. Longman Scientific & Technical; J Wiley, Harlow Essex, England.
Borkovcová, M., Mrtka, J., Winkler, J., 2012. Factors affecting mortality of vertebrates on the roads in the Czech Republic. Transp. Res. D 17, 66–72.
Borrajo, M., Comas, C., Costafreda-Aumedes, S., Mateu, J., 2021. Stochastic smoothing of point processes for wildlife-vehicle collisions on road networks. Stoch. Env. Res. Risk A. https://doi.org/10.1007/s00477-021-02072-3.
Chen, L., Yang, J., Huang, T., Kong, X., Lu, L., Cai, Y., 2015. Mining for novel tumor suppressor genes using a shortest path approach. J. Biomol. Struct. Dyn. 34 (3), 64–75.
Coffin, A.W., 2007. From roadkill to road ecology: a review of the ecological effects of roads. J. Transp. Geogr. 15, 396–406.
Deckers, B., Verheyen, K., Hermy, M., Muys, B., 2005. Effects of landscape structure on the invasive spread of black cherry (Prunus serotina) in an agricultural landscape in Flanders, Belgium. Ecography 28, 99–109.
Díaz-Varela, E.R., Vazquez-Gonzalez, I., Marey-Perez, M., 2011. Assessing methods of mitigating wildlife-vehicle collisions by accident characterization and spatial analysis. Transp. Res.D 16, 281–287.
Elith, J., Philips, S., Hastie, T., Dudik, M., Chee, Y., Yates, C., 2011. Pa statistical explanation of maxent for ecologists. Divers. Distrib. 17, 43–57.
Groot Bruinderink, G.W.T.A., Hazebroek, E., 1996. Ungulate traffic collision in Europe. Conserv. Biol. 26, 1059–1067.
Gunson, K., Mountrakis, G., Quackenbush, L., 2011. Spatial wildlife-vehicle collision models: a review of current work and its application to transportation mitigation projects. J. Environ. Manag. 92, 1074–1082.
Ha, H., Shilling, F., 2018. Modelling potential wildlife-vehicle collisions (wvc) locations using environmental factors and human population density: a case-study from 3 state highways in Central California. Ecol.Informa. 43, 212–221.
Hilário, R., Carvalho, W., Gheler-Costa, C., Rosalino, L., Marques, T., Adania, C., Paulino, J., Almeida, P., Mustin, K., 2021. Drivers of humanwildlife impact events involving mammals in southeastern Brazil. Sci. Total Environ. 794, 148600.
Johnson, D., 1977. Efficient algorithms for shortest paths in sparse networks. J. ACM 24 (1), 1–13.
Keken, Z., Sedoník, J., Kusta, T., Andrásik, R., Bíl, M., 2019. Roadside vegetation influences clustering of ungulate vehicle collisions. Transp. Res. D 73, 381–390.
Krisp, J., DurotJohnson, S., 2007. Segmentation of lines based on point densities—an optimisation of wildlife warning sign placement in southern Finland. Accid. Anal. Prev. 39, 38–46.
Litvaitis, J., Tash, J., 2008. Tan approach toward understanding wildlifevehicle collisions. Environ. Manag. 42, 688–697.
Lord, D., Mannering, F., 2010. The statistical analysis of crash-frequency data: a review and assessment of methodological alternatives. Transp. Res. A Policy Pract. 44, 291–305.
Magzhan, K., Jani, H., 2013. A review and evaluations of shortest path algorithms. Int. J. Sci. Technol. Res. 2.
McSwiggan, G., Baddeley, A., Nair, G., 2016. Kernel density estimation on a linear network. Scand. J. Stat. 44, 324–345.
Mo, W., Wang, Y., Zhang, Y., Zhuang, D., 2017. Impacts of road network expansion on landscape ecological risk in a megacity, China: a case study of Beijing. Sci. Total Environ. 574, 1000–1011.
Morelle, K., Lehaire, F., Lejeun, P., 2013. Spatio-temporal patterns of wildlife-vehicle collisions in a region with a high-density road network. Nat.Conserv. 5, 53–73.

Okabe, A., Yamada, I., 2001. The k-function method on a network and its computational implementation. Geogr.Anal. 33, 271–290.
Okabe, A., Satoh, T., Sugihara, K., 2009. A kernel density estimation method for networks, its computational method and a GIS-based tool. Int. J. Geogr. Inf. Sci. 23, 7–32.
Quintero, L., Sayed, T., Wahaba, M.M., 2012. Safety models incorporating graph theory based transit indicators. Accid. Anal. Prev. 50, 635–644.
Ramp, D., CaldwellS, J., Kathryn, K., Croft, D.W.D., 2005. Modelling of wildlife fatality hotspots along the snowy mountain highway in New South Wales, Australia. Biol. Conserv. 126, 474–490.
van der Ree, R., Smith, D.J., Grilo, C. (Eds.), 2015. Handbook of Road Ecology. John Wiley & Sons.
Ripley, B.D., 1977. Modeling spatial patterns (with discussion). J.R.Stat.Soc.B 39, 172–212.
Sàenz-de Santa-María, A., Tellería, J.L., 2015. Wildlife-vehicle collisions in Spain. Eur. J. Wildl. Res. 61, 399–406.
Seo, C., Thorne, J., Choi, T., Kwon, H., Park, C., 2013. Disentangling roadkill: the influence of landscape and season on cumulative vertebrate mortality in South Korea. Landsc. Ecol. 11, 87–99.
Spooner, P., Lunt, I., Okabe, A., Shiode, S., 2004. Spatial analysis of roadside acacia populations on a road network using the network k-function. Landsc. Ecol. 19, 491–499.
Xie, Z., Yan, J., 2008. Kernel density estimation of traffic accidents in a network space. Comput. Environ. Urban. Syst. 32, 396–406.
Xu, J.J., Chen, H., 2003. Fighting organized crimes: using shortest-path algorithms to identify associations in criminal networks. Decis. Support. Syst. 38, 473–487.
Yamada, I., Thill, J.C., 2004. Comparison of planar and network k -functions in traffic accident analysis. J. Transp. Geogr. 12, 149–158.
Yen, J.Y., 1971. Finding the k shortest loopless paths in a network.Network. Manag. Sci. 17, 661–786.

5.3 Article: 'intensitynet: Intensity-based Analysis of Spatial Point Patterns Occurring on Complex Networks Structures in R'

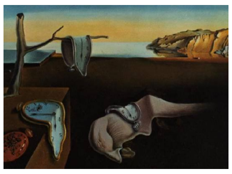
To properly analyze spatial point patterns that occur on a network, many steps are involved, for instance: data preparation, data visualization, statistical analysis of the data, and proper result display. Performing any of these steps may require time-consuming actions such as properly including all the information in the model, accessing part of the information regarding a set of vertices or nodes, performing statistical computations, or generating understandable plots to summarize results. Such operations are easily performed with the tools and functionalities provided by the package *intensitynet* presented in the paper *'intensitynet: Intensity-based Analysis of Spatial Point Patterns Occurring on Complex Networks Structures in R'*. This package is written in ***R*** language and it can be found in the Comprehensive ***R*** Archive Network (*CRAN*) as well as in the *GitHub* repository. The package is designed in a modular programming structure, with ease-to-use functions that require few data to perform operations and, with enhanced adaptability to other packages since it works with the widely known and used network modeling package *igraph*. The paper presents the structure of the *intensitynet* package, details each of its functions, and in each case presents examples using the *Chicago* dataset provided by the *intensitynet* package. These are the key points of this paper:

- The paper presents a general introduction to the *intensitynet* package.
- The introduction includes network modeling, data manipulation, intensity estimation, computation of local and global autocorrelation statistics, visualization, and extensions to marked point process scenarios.
- The package is written in ***R*** language and can be found in *CRAN* and *GitHub* repositories.
- The package can handle undirected, directed, and mixed networks.
- All the tools and methodologies provided by the package are accompanied by examples using a real dataset for the city of Chicago given by the *spatstat* package.

Where the package can be found following the next URLs:

- GitHub: `https://github.com/LlagosteraPol/intensitynet`
- CRAN: `https://cran.r-project.org/web/packages/intensitynet/index.html`

This paper has been submitted for publication and is not yet been published (changes might occur during the revision process).



Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

intensitynet: Intensity-based Analysis of Spatial Point Patterns Occurring on Complex Networks Structures in R

Pol Llagostera Universitat de Lleida

Carles Comas Universitat de Lleida

Matthias Eckardt Humboldt-Universität zu Berlin

Abstract

The statistical analysis of structured spatial point process data where the event locations are determined by an underlying spatially embedded relational system has become a vivid field of research. Despite a growing literature on different extensions of point process characteristics to linear network domains, most software implementations remain restricted to either directed or undirected network structures and are of limited use for the analysis of rather complex real-world systems consisting of both undirected and directed parts. Formalizing the network through a graph theoretic perspective, this paper discusses a complementary approach for the analysis of network-based event data through generic network intensity functions and gives a general introduction to the **intensitynet** package implemented in R covering both computational details and applications. By treating the edges as fundamental entities, the implemented approach allows the computation of intensities and other related values related to different graph structures containing undirected, directed, or a combination of both edges as special cases. The package includes characteristics for network modeling, data manipulation, intensity estimation, computation of local and global autocorrelation statistics, visualization, and extensions to marked point process scenarios. All functionalities are accompanied by reproducible code examples using the `chicago` data as toy example to illustrate the application of the package.

Keywords: Autocorrelation statistics, heatmaps, mixed network structures, visualization.

1. Introduction

The statistical investigation and characterization of point configurations in event-type data on relational objects, i.e. spatially embedded network structures, have attracted a lot of attention in recent years. Typical examples include the locations of accidents or crimes in traffic systems, insects on brick structures, and spines in neural networks. In any such data, the event locations are only observed on or along the individual structural entities (i.e. the edges) which, in turn, constitute the relational system of interest within a planar observations window. Dating back to the pioneering contributions on event-type data on linear networks of Okabe, Yomono, and Kitamura (1995) and Okabe and Yamada (2001), Ang, Baddeley, and Nair (2012) were the first to consider geometrically corrected summary characteristics for the analysis of network-based event patterns that account for the inherent structural properties of the relational system under study. Extending this idea further, a rich body of the literature has been established on network-adjusted extensions of classical spatial point process characteristics to spatially embedded structured domains (see Baddeley, Nair, Rakshit, McSwiggan, and Davies 2021, for a recent review of the literature). While extensions to directed linear networks (Rasmussen and Christensen 2021) and graphs with Euclidean edges (Anderes, Møller, and Rasmussen 2020; van Lieshout 2018) were proposed, most tools are of limited use for the analysis of more complex, real-world network structures where both directed and undirected edges coexist. In particular, the **spatstat** (Baddeley, Rubak, and Turner 2015; Baddeley and Turner 2005), **DRHotNet** (Briz-Redon 2021a), **geonet** (Schneble 2021), **spNetwork** (Gelb 2021a,b), **SpNetPrep**(Briz-Redon 2021b) and **stlnpp** (Moradi, Cronie, and Mateu 2020) packages in R (R Core Team 2021) do not help for the joint analysis of events on different types of edges nor the computation of point characteristics at different structural entities, e.g. set of neighbors or paths, of the network.

Addressing these limitations, Eckardt and Mateu (2018, 2021b) introduced a complementary approach that formalizes the network-based event data through different *network intensity functions* using a graph theoretic perspective. Treating the edges as fundamental entities, this approach allows for the computation of intensities and related quantities over different types and graph theoretic sets of (hyper-)structural entities including directed, undirected, or combinations of different, i.e. mixed, edges. All event-based characteristics derive directly from the intensities of the corresponding edges included where each edgewise intensity corresponds to the number of events, i.e. count per edge adjusted for its length. Although Eckardt and Mateu (2018, 2021b) derived a rich theoretical framework that allows for the analysis of event-type data on different network structures, no general implementation of the methodological toolbox exists. Summarizing their network intensity functions approach into an open-source solution in R, the **intensitynet** package is designed to fill this gap.

Implemented through `S3` classes in a modular programming structure, the package allows for easy computation of edgewise, nodewise, and pathwise characteristics for event-type data on (sets of) directed, undirected, and mixed-type network structures. To ease its handling, the package is built around one constructor function, i.e. `intensitynet()`, which automatically parses the underlying network structure to all computations. Using the capacities of the **igraph** (Csardi and Nepusz 2006) package, all results are provided as edge or node attributes which allow for easy manipulation and retrieval of the stored information. Apart from summary tables, the package includes flexible visualization tools which help to produce (i) a reliable plot of the network structure itself, i.e. the observed patterns on the paths between any two nodes (origin and destination), and (ii) outputs from the computed intensity functions, the corresponding mark proportions or averages, and autocorrelation statistics. Each plot can be personalized with a few arguments allowing, among other things, to show only certain nodes or edges, deciding to show or hide the event locations, and selecting its transparency. The package is available from the Comprehensive R Archive Network (CRAN) at `https://cran.r-project.org/package=intensitynet` and also as development version via the repository https://github.com/LlagosteraPol/IntensityNet/.

This paper provides a general introduction to the **intensitynet** package including data manipulation, intensity estimation, the computation of local and global autocorrelation statistics, visualization, and extensions to marked point process scenarios. The functionalities of the package are tested and exemplified using the `chicago` dataset provided by the **spatstat** package.

2. Structure and main functionalities

Embedded into a graph theoretic framework, the **intensitynet** package translates the observed network structure into a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V} = \{v_i\}_{i=1}^n$ and $\mathcal{E} = \{e_j\}_{j=1}^m \subseteq \mathcal{V} \times \mathcal{V}$ denoting the sets of vertices and edges, respectively. Each edge in $\mathcal{E}$ is associated with a pair of (not necessarily distinct) vertices, i.e. its endpoints, which constitute the relational structure of $\mathcal{G}$. For a traffic network, the edges could correspond to the set of distinct road segments and the vertices to the intersections of the edges, e.g. the crossings in the traffic system. Formally, the observed network structure is translated into a graph object through a $n \times n$ matrix $\mathbf{A}$ with element $a_{i,j}$. Each element $a_{i,j}$ numerically reflects the presence or absence of an edge joining v_i and v_j through zero and nonzero values with zeros indicating missing edges. Usually, a binary coding scheme is applied such that $\mathbf{A}$ only contains zeros and ones with ones corresponding to the edges contained in $\mathcal{E}$. We note that apart from this binary specification, more challenging coding schemes exist for weighted graphs in which weight ω, i.e. numerical attribute, is assigned to either the nodes or the edges, and multigraphs, where sets of vertices are joined by multiple edges. Under the present implementation of the **intensitynet** package, any such non-binary values need to be transformed into binary values in a preprocessing step.

If the road segments are direction preserving, i.e. if movement along a particular road is only possible in one way, the corresponding edges are called *directed*. In any such case, the associated vertices are defined by the ordered pair (v_i, v_j) and the corresponding edges are indicated by an arc with head v_j and tail v_i leading to asymmetric $\mathbf{A}$ as only $a_{i,j}$ but not $a_{j,i}$ is nonzero. In contrast, any edge which does not impose any movement restrictions is called *undirected* and related to the set $\{v_i, v_j\}$ corresponding to nonzero entries in both $a_{i,j}$ and $a_{j,i}$. Different from the directed case, undirected edges are represented by lines. Depending on the edges, $\mathcal{G}$ is called directed, if all edges are directed, undirected if all edges are undirected, and mixed if $\mathcal{G}$ consists of both directed and undirected edges. See Table 1 for a visual representation and corresponding specification of $\mathbf{A}$ for all three possible graph representations covered by the **intensitynet** package. To relate the spatial network structure to its graph theoretical representation and investigate the distributional characteristics of the event locations over the edges, both the graph and the events need to be augmented by additional geographical information. To this end, the sets of n nodes and p event locations are both reformulated into sets of coordinates $\{\mathbf{v}_1, \ldots, \mathbf{v}_n)$ and $\{\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_p\}$ where $\mathbf{v}_i = (x_i, y_i)$ and $\boldsymbol{\xi}_j = (u_i, w_j)$ are the exact coordinates of the i-th node and j-th event with respect to a given coordinate reference system (CRS), respectively. As such, only the set of n nodes $\{\mathbf{v}_1, \ldots, \mathbf{v}_n)$ is treated as fixed while the p events are considered as realizations of an unobserved stochastic mechanic which governs the locations on the network. Although both pairs of coordinates are treated differently, both sets of coordinates must be derived from a unique CRS system.

To explicitly control for the structured nature of the data and consider the points and net-

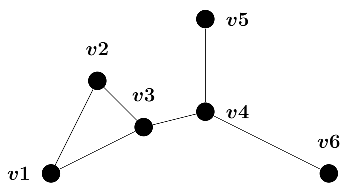
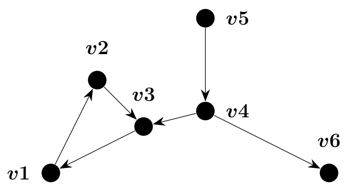
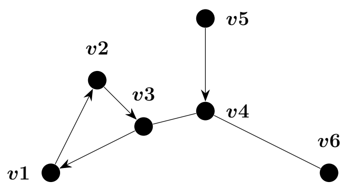
Graph Type	**Graph Structure**	**Adjacency Matrix**
Undirected		$\mathbf{A} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$
Directed		$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$
Mixed		$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$

Table 1: Examples of potential graph specifications and corresponding adjacency matrices covered by the **intensitynet** package

work structure simultaneously, all summaries and characteristics in the **intensitynet** package derive directly from local computations, treating the edges as core elements. To this end, the edges are considered as edge intervals with associated Euclidean length which takes $(\mathbf{v}_i, \mathbf{v}_j)$ as endpoints. The edgewise intensity itself can then be computed as the sum over all events $\{\boldsymbol{\xi}_p\}$ that fall onto the interval spanned between $(\mathbf{v}_i, \mathbf{v}_j)$ adjusted for its length. All alternative nodewise and pathwise summaries included in the **intensitynet** package derive directly from the edgewise characteristics including directed, undirected and mixed graph versions (see Eckardt and Mateu 2018, 2021b, for detailed discussion). To correct for small spatial deviations of the locations of the events $\{\boldsymbol{\xi}_p\}$ from the edges, the **intensitynet** packages allow to specify a maximal spatial error distance between the edges and events, i.e. a spatial buffer, that assigns each event to a distinct edge based on the distance argument specified by the user (see Section 3.4.1 for detailed description).

3. Operations on data

This section provides a detailed description of the different functionalities included in the **intensitynet** package and explains its application using the `chicago` data, originally provided by the **spatstat** (Baddeley *et al.* 2015; Baddeley and Turner 2005) package, as toy example. To highlight its use in different graph settings, the **intensitynet** package provides three adapted versions of the original data, i.e. `und_intnet_chicago`, `dir_intnet_chicago`, and `mix_intnet_chicago`, based on undirected, directed and mixed graph specifications.

3.1. Structure and classes

To introduce the basic operations and the interrelations among the functionalities of the **intensitynet** package, its internal structure needs to be described first. Conceptualized into two main and three subclasses, the **intensitynet** package provides several functions (computational tools) to calculate, manipulate and visualize structured event-type data observed on relational systems. Its internal structure and the relation among the distinct (sub)classes and functions are presented in Figure 1 in form of a Unified Modeling Language (UML) class diagram. At its core, the package creates a proper `intensitynet` object based on the given arguments of the `intensitynet()` function, and a constructor associated with the main class `intensitynet` of the package. All general functionalities of the main class are provided as visible (public) functions. Specific methods for undirected, directed and mixed graphs are given by the three subclasses `intensitynetUnd`, `intensitynetDir`, and `intensitynetMix` which inherit all methods from the main class. Associated with the `intensitynet` class, the package consists of a second main class, called `netTools`, which is constructed as a helper class with a set of functions used by the `intensitynet` class and corresponding graph specific subclasses. All methods of the helper class are implemented as invisible, i.e. private, functions and primarily designed for internal operations and should not be applied directly by the user. We note that apart from the main functionalities, both the `intensitynet` class and its related subclasses also contain private functions which are not designed for direct application. To model, manipulate and store the network and to externally visualize the results, all network-based information and computational results including distances, intensities, and correlations are stored as node and edge attributes using the capacities of the **igraph** (Csardi and Nepusz 2006) package.

3.2. The intensitynet function

To apply any operations on network-based event data, the event locations and corresponding information derived from the graph theoretic reformulation of the underlying relational system needs to the transformed into an `intensitynet` object. This object is initialized by the constructor `intensitynet()` which, in turn, requires the following three elements as mandatory attributes:

- `adjacency_mtx`: a binary adjacency matrix of dimension $n \times n$ which specifies the network structure as the relation (adjacency) between each pair of nodes according to Table 1.

- `node_coords`: a $n \times 2$ vector as `DataFrame` corresponding to the n pairs of exact coordinates provided by $\{\mathbf{v}_i\}_{i=1}^{n}$. These coordinates will be used to calculate the distances of edges and to draw the network in the desired spatial structure (shape).

- `event_data`: a $p \times 2$ vector as `DataFrame` corresponding to the p pairs of exact coordinates provided by $\{\boldsymbol{\xi}_j\}_{j=1}^{p}$. If additional marks, i.e. qualitative or quantitative point attributes, are available, `event_data` is required to be a `DataFrame` of dimension $p \times 3$ with the exact coordinates placed in the first two, and the mark information in the third columns. The mark column could contain mixtures of real-valued (numerical) and

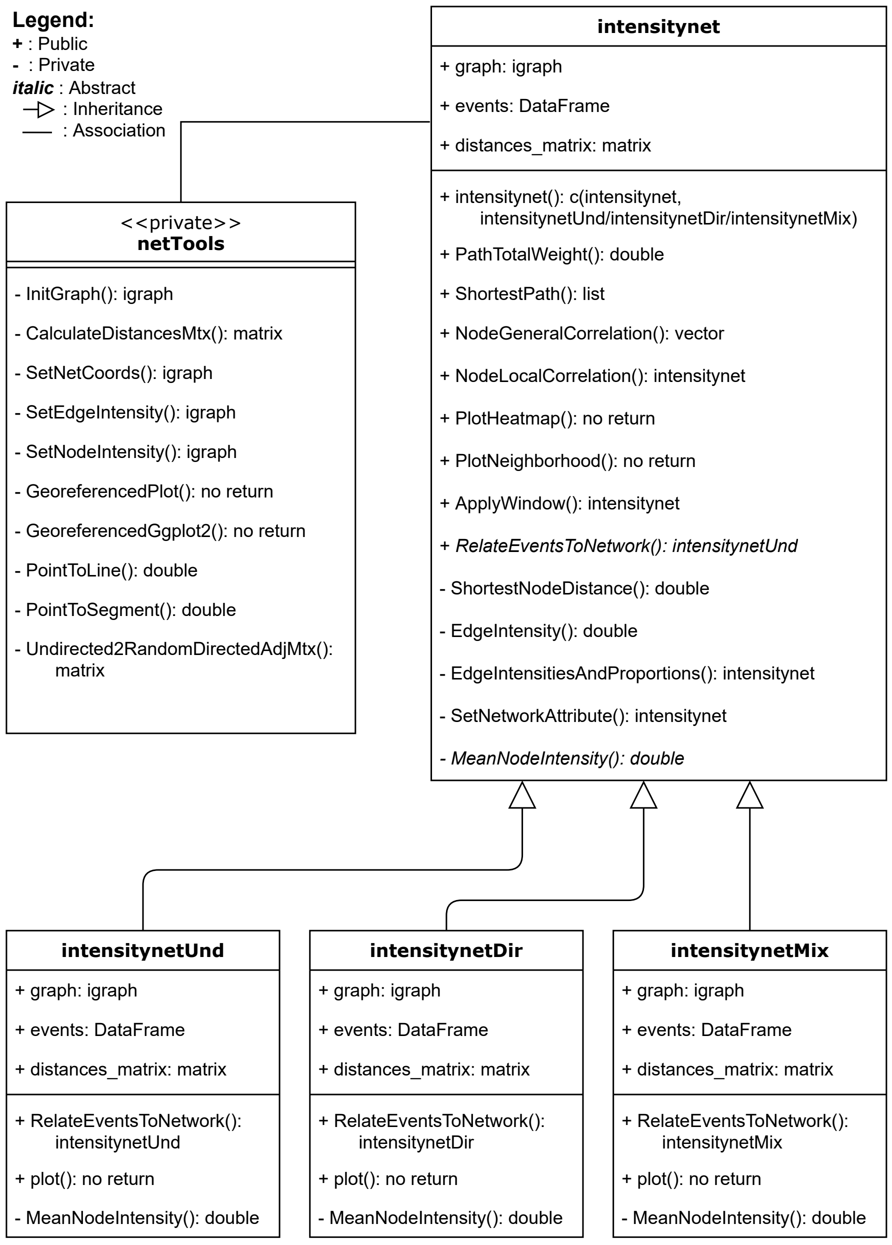

Legend:
+ : Public
- : Private
italic : Abstract
: Inheritance
: Association
intensitynet
+ graph: igraph
+ events: DataFrame
+ distances_matrix: matrix
+ intensitynet(): c(intensitynet, intensitynetUnd/intensitynetDir/intensitynetMix)
+ PathTotalWeight(): double
+ ShortestPath(): list
+ NodeGeneralCorrelation(): vector
+ NodeLocalCorrelation(): intensitynet
+ PlotHeatmap(): no return
+ PlotNeighborhood(): no return
+ ApplyWindow(): intensitynet
+ RelateEventsToNetwork(): intensitynetUnd
- ShortestNodeDistance(): double
- EdgeIntensity(): double
- EdgeIntensitiesAndProportions(): intensitynet
- SetNetworkAttribute(): intensitynet
- MeanNodeIntensity(): double
<<private>>
netTools
- InitGraph(): igraph
- CalculateDistancesMtx(): matrix
- SetNetCoords(): igraph
- SetEdgeIntensity(): igraph
- SetNodeIntensity(): igraph
- GeoreferencedPlot(): no return
- GeoreferencedGgplot2(): no return
- PointToLine(): double
- PointToSegment(): double
- Undirected2RandomDirectedAdjMtx(): matrix
intensitynetUnd
+ graph: igraph
+ events: DataFrame
+ distances_matrix: matrix
+ RelateEventsToNetwork(): intensitynetUnd
+ plot(): no return
- MeanNodeIntensity(): double
intensitynetDir
+ graph: igraph
+ events: DataFrame
+ distances_matrix: matrix
+ RelateEventsToNetwork(): intensitynetDir
+ plot(): no return
- MeanNodeIntensity(): double
intensitynetMix
+ graph: igraph
+ events: DataFrame
+ distances_matrix: matrix
+ RelateEventsToNetwork(): intensitynetMix
+ plot(): no return
- MeanNodeIntensity(): double

Figure 1: UML class diagram of the inherent structure of the **intensitynet** package indicating its classes, methods, and internal relations. Functions provided by the different classes use the **-** and + symbols to differentiate between private and public functions, respectively, and ***italic*** to indicate abstract functions. Associations between the classes are indicated by solid lines and inheritance relations by arcs (—▷).

categorical (string) entries. However, at each row, only one numerical or categorical mark is permitted.

For both `node_coords` and `event_data`, the coordinates are required to be numbers without any punctuation or spaces.

In addition, two optional arguments can be provided to the `intensitynet()` function to customize the network structure and event-on-edge alignment. By default, `intensitynet()` assumes an undirected graph. For alternative graph structures, specification of the `graph_type` is required to compute the corresponding set of suitable network intensity functions.

- `graph_type`: A string (text) indicating the specific network structure under study. Potential statements include `undirected`, treated as the default value, `directed`, and `mixed`.
- `event_correction`: Numerical value specifying the maximal distance in meters used to compute the event-to-edge alignment. By default, a maximal pointwise distance of 5 meters is applied using the world cartography representation WGS84 as the coordinate reference system. It is strongly advised to carefully adapt this value to the CRS of the node and event locations. Higher values of the `event_correction` argument potentially yield an increase of (mis-)assigned events while values close to 0 may increase the number of events that are not assigned to any edge in the network and, whence, excluded from any subsequent computations.

The constructor function returns a two-class object, i.e. an object with class `intensitynet` and corresponding subclass depending on the given `graph_type` argument according to Figure 1. The secondary classes are used internally to call the corresponding functions and operations for undirected, directed, or mixed network structures. The resulting object wraps 4 values into a list with the following information:

- `graph`: the graph representation of the observed network stored as `igraph` object with the Euclidean length placed as an edge attribute called *weight*.
- `events`: DataFrame of events as provided by the user in the `event_data` argument of `intensitynet` function.
- `graph_type`: A string corresponding to the specification of the graph type (`undirected`, `directed` or `mixed`).
- `distances_matrix`: A $n \times n$ dimensional matrix with the pairwise Euclidean distances between the node locations on the network specified as the length of the virtual line joining any pair of nodes in the network.
- `event_correction`: Value specified in the `event_correction` argument of the `intensitynet` function corresponding to the maximal distance used in the computation for the pointwise event-to-edge assignment.

All elements of the `intensitynet` object can directly be addressed, analogous to standard data operations in R, by using e.g. the '\$´ symbol.

3.3. Specifying the intensitynet objects from alternative network formats

Before providing a detailed treatment of available data manipulation operations in the **intensitynet** package, we briefly outline potential steps to compute the `intensitynet` function from (i) a linear point process (`lpp`) object as provided by the **spatstat** package and (ii) shapefile (`shp`) information. Although the aim of our package is not to read, adapt or convert different formats into an `intensitynet` object, all mandatory arguments of the `intensitynet()` function can be retrieved from different sources using a variety of available R packages.

Specifying the intensitynet object from linear point process object

In what follows, the main steps to gather all relevant information, i.e. the coordinates of the nodes, the corresponding adjacency matrix **A**, and the set of event locations, from a `lpp` object, using the capacities of the **igraph** package are described and illustrated using the `chicago` data of the **spatstat** package.

In the **spatstat** package, a `lpp` object is jointly defined from two objects: (i) a point pattern object (`ppp`) which specifies the observed event locations and (ii) a corresponding linear network object (`linnet`) which specifies the underlying network structure as a network of line segments. Having installed and loaded the **spatstat** and **intensitynet** packages and `chicago` data into the R environment, information on the edges can be extracted from the `from` and `to` vectors of the `domain` attribute of the `linnet` object which specifies the individual edges by the corresponding pairs of endpoints (`from`, `to`). The exact coordinate information for the vertices can then be retrieved from the `x` and `y` vectors of the `vertices` attribute of the `domain` object. Executing the following lines, both the edges and the vertices are stored as a `vector` and `DataFrame`, respectively.

```
R> library(intensitynet)
R> library(spatstat)
R> data(chicago)

R> edges <- cbind(chicago[["domain"]][["from"]], chicago[["domain"]][["to"]])
R> node_coords <- data.frame(xcoord=chicago[["domain"]][["vertices"]][["x"]],
+                            ycoord=chicago[["domain"]][["vertices"]][["y"]])
```

Given the coordinates of the nodes, the second mandatory argument of the `intensitynet()` function, i.e. the adjacency matrix **A**, can be computed directly from the `edges` object using subsequently the `graph_from_edgelist()` and `as_adjacency_matrix()` functions from the **igraph** package.

```
R> net <- igraph::graph_from_edgelist(edges)
R> adj_mtx <- as.matrix(igraph::as_adjacency_matrix(net))
```

Finally, ignoring any additional mark information provided by the `chicago` data, the event locations can be retrieved from the `x` and `y` columns of the `data` attribute of the `lpp` object. We note that identical coordinates information to `x` and `y` is also provided in the columns `seg` and `tp`, but using an alternative coordinate reference system. Collecting the required information, the `intensitynet` object can then be specified by executing the following lines.

```
R> chicago_df <- as.data.frame(chicago[["data"]][, -(3:4)])

R> intnet_chicago <- intensitynet(adjacency_mtx = adj_mtx,
+                                 node_coords = node_coords,
+                                 event_data = chicago_df)
R> attributes(intnet_chicago)
$names
[1] "graph"   "events"   "graph_type"   "distances_mtx"   "event_correction"

$class
[1] "intensitynet"    "intensitynetUnd"
```

Compared to the required operations of the **spatstat** package for network event data importation, the **intensitynet** appears to be highly flexible with eager object generability. Different from the simple network object specification through the `intensitynet()` function and essential data requirement, **spatstat** asks to perform several data transformations to define a linear network pattern. In detail, this process requires to subsequently create a point process object (`ppp`) and a `linnet` object which itself derives from an `ppp` object plus a two-column matrix specifying the edges and, finally, the linear point pattern object (`lpp`) with the `linnet` object plus the location of the points (coordinates). Moreover, if it is intended to create an `igraph` object using the linear point pattern, more conversions are required using the help of additional libraries such as `sf` and `sfnetworks`.

Specifying the intensitynet object from shapefile data

Shapefile (`shp`) event-type data emerge in different contexts, in particular in public data, and is commonly used in cartographic tools such as ArcGIS (Redlands 2011) or QGIS (QGIS Development Team 2022). As for the `lpp` object, any `shp` object can be transformed into the desired formats using the capacities of the **maptools** (Bivand and Lewin-Koh 2019) and **shp2graph** (Lu, Sun, Xu, Harris, and Charlton 2018) packages. All required steps are illustrated using the `ORN.nt` data on the Ontario road network (Ontario 2006) provided by the **shp2graph** package.

If required, any shapefile could initially be read into the R environment using the function `readShapeLines()` from **maptools** which stores the provided information as `SpatialLines-DataFrame` object, an object of class `sp`. Here, as the `ORN.nt` data is already provided in `sp` format, this initial step through the **maptools** package is omitted. Given a `sp` object as in our example, the object can be transformed into a list using the `readshpnw()` function of the **shp2graph**. By setting the argument `ELComputed` of the `readshpnw()` function to `TRUE`, the list will also contain the edge lengths.

```
R> library(intensitynet)
R> library(shp2graph)
R> data(ORN)

R> rtNEL <- shp2graph::readshpnw(ORN.nt, ELComputed = TRUE)
```

Executing the above code yields a list with the following elements: a Boolean stating if the list is in *Detailed* mode (`FALSE` by default), the list of nodes from the network, the list of edges, the lengths of the edges, a DataFrame of edge attributes extracted from the `sp` object, a vector with the `x` coordinates of all the nodes, and a vector containing the `y` coordinates. From this list, only information on the nodes, edges, and lengths of the edges are needed for further operations. Using the `nel2igraph()` from the **shp2graph** package, this selected information can finally be translated into an `igraph` object as outlined in the following lines.

```
R> nodes_orn <- rtNEL[[2]]
R> edges_orn <- rtNEL[[3]]
R> lenghts_orn <- rtNEL[[4]]

R> net_orn<-shp2graph::nel2igraph(nodelist = nodes_orn,
+                                 edgelist = edges_orn,
+                                 weight = lenghts_orn)
```

From the `net_orn` object, all required information to initialize the `intensitynet()` function can easily be obtained. Noting that `net_orn` object is a network in `igraph` format, its adjacency matrix **A** can directly be recovered using the `graph_from_edgelist()` and `as_adjacency_matrix()` functions as outlined in the previous section. The coordinates of the nodes can be extracted from the `nodes_orn` object using the `Nodes.coordinates()` function from the **shp2graph** package. Given the above information, the initialization of the `intensitynet` object is exemplified below. Notice that the argument `event_data` is an empty matrix with two columns (corresponding to the number of columns required) as no events are available in the original data.

```
R> adj_mtx_orn <- as.matrix(igraph::as_adjacency_matrix(net_orn))
R> node_coords_orn <- shp2graph::Nodes.coordinates(nodes_orn)

R> intnet_orn <- intensitynet(adjacency_mtx = adj_mtx_orn,
+                             node_coords = node_coords_orn,
+                             event_data = matrix(ncol = 2))
```

3.4. Event-to-edge alignment and network intensity function estimation

Event-to-edge alignment

The `intensitynet` object described in Section 3.2 summarizes the network-based event data together with information on the network as such including the node coordinates, its associated adjacency matrix, and the edge distances. Although the `intensitynet` object already allows for the direct computation of different graphical outputs and to retrieve the data or `igraph` object, it does not provide any event characteristics, i.e. intensity functions or autocorrelation statistics. To derive any further information on the event pattern from the `intensitynet` object, the individual event locations $\{\boldsymbol{\xi}_j\}_{j=1}^{p}$ need to be assigned to the edges in a subsequent action using the `RelateEventsToNetwork()` function. At its core, this function computes the distances $d_{j,k}$ of the j-th event to all k edges in the graph and assigns the event to exactly one edge based on the minimal distance $d_{j,k}$ which, in turn, depends on a pre-specified threshold distance τ stated by the user in the `event_correction` argument of the `intensitynet` object.

To illustrate the potential event-to-edge assignment scenarios, consider the three toy graphs $\mathcal{G}_1, \mathcal{G}_2$ and $\mathcal{G}_3$ with nodes $\mathbf{v}_1$ and $\mathbf{v}_2$ denoting the endpoints of an edge and $\boldsymbol{\xi}$ an artificial event location that calls for an event-to-edge assignment. Mathematically, any such assignment operation can be defined through the distance $\delta(\boldsymbol{\xi}, (\mathbf{v}_1, \mathbf{v}_2))$ between the event $\boldsymbol{\xi}$ and its projection to an edge with endpoints $(\mathbf{v}_1, \mathbf{v}_2)$. Using the distance δ as decision criteria, any event $\boldsymbol{\xi}$ is re-assigned to its closest edge. Note that each event-to-edge assignment only accounts for the subset of edges satisfying $\delta \leq \tau$. Theoretically, three potential cases might appear as illustrated in the toy examples $\mathcal{G}_1$ to $\mathcal{G}_3$ (compare Figure 2). While the event $\boldsymbol{\xi}$ falls into a rectangular area formed by both nodes $\mathbf{v}_1$ and $\mathbf{v}_2$ in scenario one ($\mathcal{G}_1$), $\boldsymbol{\xi}$ is in the exterior but close to the area spanned by $\mathbf{v}_1$ or $\mathbf{v}_2$ in the two alternative scenarios. To decide on the closest distance δ to the edge $(\mathbf{v}_1, \mathbf{v}_2)$, the **intensitynet** package performs a triangulation based on a vector projection and rejection approach (see Fox and Bolton 2002, for detailed discussion) to select the optimal event-to-edge assignment.

Formally, given the nodes $\mathbf{v}_1 = (x_1, y_1)$, $\mathbf{v}_2 = (x_2, y_2)$ and the event $\boldsymbol{\xi} = (u, w)$ on a network, simple calculation yield the vectors $\mathbf{a} = \mathbf{v}_2 - \mathbf{v}_1 = (x_2 - x_1, y_2 - y_1)$, $\mathbf{b} = \boldsymbol{\xi} - \mathbf{v}_1 = (u - x_1, w - y_1)$ and $\mathbf{c} = \boldsymbol{\xi} - \mathbf{v}_2 = (u - x_2, w - y_2)$. The definition of $\mathbf{a}$ and $\mathbf{b}$ allows to compute projection $\mathbf{b}_1$ of $\mathbf{b}$ onto the edge $(\mathbf{v}_1, \mathbf{v}_2)$ where $\mathbf{b}_1 = ((\mathbf{b} \cdot \mathbf{a})/(\mathbf{a} \cdot \mathbf{a})) \times \mathbf{a}$ and $\cdot$ denotes the dot product of the vectors $\mathbf{a}$ and $\mathbf{b}$. Finally, from $\mathbf{b}$ and its projection $\mathbf{b}_1$, the vector rejection $\mathbf{b}_2$ can be obtained from calculation of $\mathbf{b}_2 = \mathbf{b} - \mathbf{b}_1$ as $\mathbf{b} = \mathbf{b}_1 + \mathbf{b}_2$.

Under the above formulation and depending on the exact location of $\boldsymbol{\xi}$ relative to $\mathbf{a}$, the closest edge could be either $\mathbf{b}$, $\mathbf{b}_2$ or $\mathbf{c}$. To decide on which one is the closest, the vector distances of $\mathbf{b}_1$ and $\mathbf{a}$ can be used to determine whether $\boldsymbol{\xi}$ falls into the area spanned by $\mathbf{v}_1$ and $\mathbf{v}_2$ or an exterior region close to $\mathbf{v}_1$ or $\mathbf{v}_2$. If the location of $\boldsymbol{\xi}$ is between $\mathbf{v}_1$ and $\mathbf{v}_2$, the closest vector is $\mathbf{b}_2$, if it falls in an exterior area but close to $\mathbf{v}_2$, the closest vector is $\mathbf{c}$, and if it falls in an exterior area but is close to $\mathbf{v}_1$, the closest vector is $\mathbf{b}$.

The following decision criteria on the event-to-edge assignment can be derived - assuming either positive (left) or negative (right) directions of $\mathbf{a}$:

$$\begin{gathered}
0 < |\mathbf{b}_1| < |\mathbf{a}| \rightarrow \mathbf{b}_2 \text{ and } 0 > |\mathbf{b}_1| > |\mathbf{a}| \rightarrow \mathbf{b}_2 \\
0 \geq |\mathbf{b}_1| < |\mathbf{a}| \rightarrow \mathbf{b} \text{ and } 0 \leq |\mathbf{b}_1| > |\mathbf{a}| \rightarrow \mathbf{b} \\
0 < |\mathbf{b}_1| \geq |\mathbf{a}| \rightarrow \mathbf{c} \text{ and } 0 > |\mathbf{b}_1| \leq |\mathbf{a}| \rightarrow \mathbf{c}.
\end{gathered}$$

Based on the above decision criteria, the distance to the closest selected edge is calculated. This is the distance that will be used to determine whether the event is accounted into the edge events. The described procedure is implemented in the private function `PointToSegment()` of the class `netTools` and used by the `RelateEventsToNetwork()` function .

Network intensity function estimation

Having performed an event-to-edge assignment for all events of the `intensitynet` object, the `RelateEventsToNetwork()` function internally computes the edgewise and nodewise intensities of the network-based event pattern. As the core element of all network intensity function computations, the edgewise intensity which quantifies the number of events for each edge in

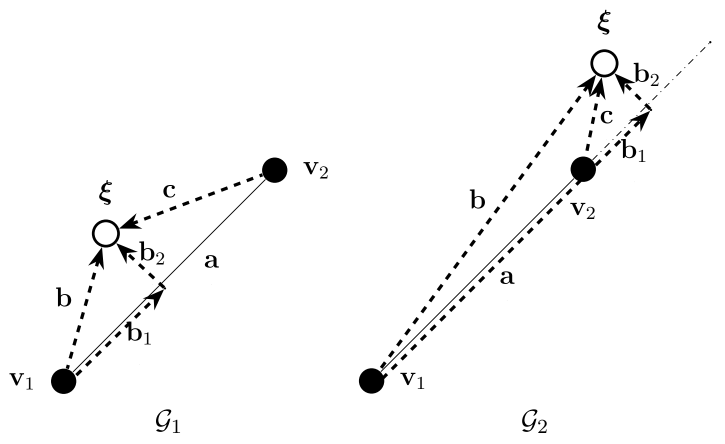


Figure 2: Three potential scenarios encountered in the event-to-edge alignment. The event $\boldsymbol{\xi}$ is located between the nodes $\mathbf{v}_1$ and $\mathbf{v}_2$ of $\mathcal{G}_1$ (left), the event $\boldsymbol{\xi}$ is not located between $\mathbf{v}_1$ and $\mathbf{v}_2$ but close to $\mathbf{v}_2$ of $\mathcal{G}_2$ (central), and the event $\boldsymbol{\xi}$ is not located between $\mathbf{v}_1$ and $\mathbf{v}_2$ but close to $\mathbf{v}_1$ of $\mathcal{G}_3$ (right).

the network adjusted for the length of the edge is computed in an initial step. We note that different from all alternative statistics, the edgewise intensity function is calculated for all edges regardless of the type and potential direction restrictions. This local quantity is then used in subsequent actions to compute the nodewise and also pathwise intensity functions. The nodewise intensity is computed as the average of the edgewise ones over the set of edges included in the hyperstructural set under study, i.e. the set of neighbors, the set of parents, or the set of children (see Eckardt and Mateu 2018, 2021b, for the formulation and mathematical details on different network intensity functions).

Internally, the edgewise and nodewise intensities are implemented in the private functions `EdgeIntensity()` and `MeanNodeIntensity()`, respectively. The `MeanNodeIntensity()` function is an abstract method from the `intensitynet` class and is implemented by its subclasses `intensitynetUnd`, `intensitynetDir`, and `intensitynetMix` (see Figure 1). The second class of `intensitynet` object allows to automatically compute the correct nodewise intensity depending on the network type (undirected, directed, or mixed). For undirected networks, the `MeanNodeIntensity()` function yields only one output which is stored under the name `intensity`. Different from the undirected case, application of the `MeanNodeIntensity()` function to directed graphs yields two different network intensity functions corresponding to the set of parent pa (where $pa(i) = j \to i$) and set of children ch (where $ch(i) = i \to j$) that are stored under the names `intensity_in` and `intensity_out`, respectively. Formally, the `intensity_in` object is computed by averaging the edgewise intensities over the set of incident edges with respect to a given vertex. Likewise, the `intensity_out` object reports the average intensity at node level constructed from the edgewise intensities of the set of dissident edges. Lastly, all these three outputs (`intensity_und`, `intensity_in`, `intensity_out`) and one additional object called `intensity_all` are provided in the case of mixed graphs, i.e. when the graph consists of both undirected and directed edges (see Diestel 2016, for a detailed review of Graph Theory concepts). The application of the `RelateEventsToNetwork()` function to empirical data is illustrated in the following lines.

Continuing with the `chicago` toy example, the internal event-to-edge assignment based on the `edge_correction` arguments and subsequent intensity estimations are initialized by directly applying the `RelateEventsToNetwork()` function to the `intensitynet` object. Recalling that all computations are internally stored as edge and node attributes, the graph object `g` needs to be addressed to extract the stored information from the results. Given the graph object, both the nodewise and edgewise intensities could be retrieved by specifying the `what` argument of the `igraph::as_data_frame(g, what)` command as `vertices` and `edges`, respectively.

```
R> intnet_chicago <- RelateEventsToNetwork(intnet_chicago)
Calculating edge intensities with event error distance of 5...
  |==============================================================| 100%
Calculating node intensities...
  |==============================================================| 100%

R> g <- intnet_chicago$graph
R> class(g)
[1] "igraph"

R> head(igraph::as_data_frame(g, what = "vertices"))
   name      xcoord   ycoord   intensity
V1   V1   0.3894739 1253.803 0.000000000
V2   V2 109.6830137 1251.771 0.000000000
V3   V3 111.1897363 1276.560 0.000000000
V4   V4 198.1486340 1276.560 0.000000000
V5   V5 197.9987626 1251.153 0.008242282
V6   V6 290.4787354 1276.560 0.000000000

R> head(igraph::as_data_frame(g, what = "edges"))[1:5]
  from  to    weight n_events  intensity
1   V1  V2 109.31241        0 0.00000000
2   V2  V3  24.83439        0 0.00000000
3   V2  V5  88.31791        0 0.00000000
4   V2 V24  54.86415        0 0.00000000
5   V4  V5  25.40732        0 0.00000000
6   V5  V7  90.99421        3 0.03296913
```

If the `intensitynet` object contains additional mark information, the `RelateEventsToNetwork()` function automatically calculates additional edgewise summaries in addition to the intensity computations. Accounting for the mark information provided, the `RelateEventsToNetwork()` function computes either edgewise averages (numerical marks) or proportions (categorical marks) of all events that fall into the edge interval considered. All computations including the edgewise and nodewise intensities, the mark means and proportions, and the number of events per edge are stored as node and edge attributes using the `igraph` class.

To give an example of the underlying computations, consider a set of network-based events with numerical mark *km* and categorical mark *animals* with levels *cat*, *dog*, and *cow*. For *km* and *animals*, suppose that we observed three events with marks $100, 20, 30$ and one *cat* recorded twice and *dog* appearing one time. Given the above specified data, application of the `RelateEventsToNetwork()` function results in the values 50 for *km*, 0.666 for *cat* and 0.333 for *dog*. As no event is labeled as *cow* in the toy data, the output will include a zero in the corresponding field.

Given any empirical network-based event data with additional mark information, the internal edge-to-event assignment and subsequent calculation can be implemented analogously to the unmarked case as illustrated above. From the output of the `RelateEventsToNetwork()` function, all results can be retrieved in subsequent actions directly from the graph object `g` as illustrated in the next lines.

```
R> head(igraph::as_data_frame(g, what = "edges"))[6:12]
  assault burglary cartheft    damage   robbery theft trespass
1       0        0        0 0.0000000 0.0000000     0        0
2       0        0        0 0.0000000 0.0000000     0        0
3       0        0        0 0.0000000 0.0000000     0        0
4       0        0        0 0.0000000 0.0000000     0        0
5       0        0        0 0.0000000 0.0000000     0        0
6       0        0        0 0.3333333 0.6666667     0        0
```

Apart from working on the complete network, the **intensitynet** package allows the selection of smaller areas from the network using the `ApplyWindow()` function. As before, this function only requires an `intensitynet` object as input and the pairs of (x, y) coordinates specifying the boundaries of the selection window. The function extracts the induced sub-network along with corresponding event locations and returns an object with the same class type as its origin. We note that apart from the pre-selection of the observation window, the function also allows for post-selection of sub-networks. In that case, all computations corresponding to the selected area will be inherited from the sub-network.

To illustrate the `ApplyWindow()` function, we outline its application for the `chicago` toy data example. Using the `intnet_chicago` object as input, an extract of the substructure with borders $x = (300, 900)$ and $y = (500, 1000)$ is computed executing the next lines of code.

```
R> sub_intnet_chicago <- ApplyWindow(intnet_chicago,
                                     x_coords = c(300, 900),
                                     y_coords = c(500, 1000))
```

Given any substructure, both the original and the selected network event patterns can be compared using e.g. the `gorder()` and `gsize()` functions which compute the number of nodes and edges in the corresponding networks, respectively, as outlined next.

```
R> c(igraph::gorder(intnet_chicago$graph),
      igraph::gsize(intnet_chicago$graph),
      nrow(intnet_chicago$events))
[1] 338 503 116

R> c(igraph::gorder(sub_intnet_chicago$graph),
      igraph::gsize(sub_intnet_chicago$graph),
      nrow(sub_intnet_chicago$events))
[1] 75 112 37
```

3.5. Exploring the network structure

Before proceeding with the estimation of alternative event-related characteristics, summaries of the network structure itself provided by **intensitynet** package are presented first.

Given any network, an important graph theoretic concept that helps to quantify the structural pairwise interrelations between the distinct nodes is the shortest path (Rahman 2017). From a graph theoretic perspective, a *path* connecting any two vertices (origin and destination) is defined as a sequence of distinct vertices $(v_1, v_2, \cdots, v_n)$ with a maximum degree of 2 and distinct edges $(e_1, e_2, \cdots e_{n-1})$ such that no edge and no vertex is traversed twice. A *shortest path* between an origin and a destination is the minimum number of edges (if unweighted) or the minimum total weight of the edges (if weighted) to move along a path from the origin to the destination. Several algorithms and methodologies can be found in the literature that computes the shortest path including the algorithms of Dijkstra, Bellman-Ford, Floyd-Warshall, (Magzhan and Jani 2013), and Johnson (Johnson 1977) or the Bread-First search algorithm to name just a few. Each of these algorithms serves better for certain networks with particular characteristics and, in turn, the choice of the algorithm used has a crucial impact on the computation and might yield an increase in the computational time or even intractable computations.
The function `ShortestPath()` of the **intensitynet** package is designed to simplify the shortest path detection on the network. Basically, this function calculates the shortest path between two nodes based on either the number of edges or their weights. To operate, the function only requires the network of class `intensitynet`, the origin and destination nodes (either its names as strings or IDs as integers) specifying the path under selection, and an optional parameter specifying the weight type as arguments. If no weight is provided, the function calculates the shortest path based on the number of edges. If weights are available, the shortest path detection returns the minimum total weight (sum of all path edge weights). Regarding the chosen algorithm that calculates the shortest path, the function will choose automatically the one that fits best for the given network. Internally, the function performs a selection based on the network characteristics. If the network is unweighted (or the weights are not considered) then the Bread-First search algorithm is used. If the user specifies a valid weight (that is present in the edge attributes), then Dijkstra's algorithm is chosen. For negative weights, the shortest path detection either uses Johnson's algorithm, if the network consists of more than 100 nodes, or Bellman-Ford's algorithm if the number of nodes is less than 100.

The use of the `ShortestPath()` function and the specification of its argument are demonstrated in the next lines of code using the `intnet_chicago` object. Selecting the nodes $\mathbf{v}_1$ (`V1`) and $\mathbf{v}_{300}$ (`V300`) from the `intnet_chicago` object, the following lines outline the shortest path detection with respect to the minimum number of edges, the minimum pathwise intensity, and the minimum number of car thefts taking additionally the mark information into account.

```
R> short_path <- ShortestPath(intnet_chicago,
                              node_id1 =  "V1" ,
                              node_id2 =  "V300")
R> short_path$path
+ 16/338 vertices, named, from d8a2c9e:
 [1]  V1   V2   V24  V46  V65  V83  V96  V115 V123 V125 V134 V219 V220
 [14] V221 V294 V300
```

```
R> short_path$total_weight
[1] 16

R> short_path_intensity <- ShortestPath(intnet_chicago,
                                         node_id1 =  "V1" ,
                                         node_id2 =  "V300" ,
                                         weight =  "intensity")
R> short_path_intensity$path
+ 44/338 vertices, named, from d8a2c9e:
 [1]  V1   V2   V24  V25  V26  V48  V63  V85  V102 V103 V144 V141 V140
 [14] V224 V223 V222 V221 V294 V300

R> short_path_intensity$total_weight
[1] 0.1826603

R> short_path_cartheft <- ShortestPath(intnet_chicago,
                                        node_id1 =  "V1" ,
                                        node_id2 =  "V300" ,
                                        weight =  "cartheft")
R> short_path_cartheft$path
+ 24/338 vertices, named, from d8a2c9e:
 [1]  V1   V2   V24  V25  V26  V48  V63  V85  V104 V103 V144 V141 V142
 [14] V135 V136 V137 V224 V223 V225 V282 V292 V293 V301 V300

R> short_path_cartheft$total_weight
[1] 0
```

Apart from the shortest path detection, the `PathTotalWeight()` function provided by the **intensitynet** package additionally allows the extraction of the weights of any given (not necessarily shortest) path. Requiring (i) the network as `intensitynet` object, (ii) a vector specifying the nodes along the path under selection, and (iii) an optional parameter with the type of weight to be computed as arguments, its use is illustrated in the following code. Note that if no weight type is provided, `PathTotalWeight()` returns the total amount of edges in the path, otherwise, returns the total sum of the specified weight.

```
R> path <- c("V89",  "V92",  "V111", "V162",  "V164")
R> PathTotalWeight(intnet_chicago, path = path)
[1] 3

R> PathTotalWeight(intnet_chicago, path = path, weight =  "intensity")
[1] 0.03296913

PathTotalWeight(intnet_chicago, path = path, weight =  "robbery")
[1] 0.6666667
```

Note that the path must be specified as an ordered list corresponding to the movement on the graph, i.e. a path with starting point $\mathbf{v}_{89}$ and destination $\mathbf{v}_{164}$ specified by $\mathbf{v}_{89} \rightarrow \mathbf{v}_{92} \rightarrow$

$\mathbf{v}_{111} \to \mathbf{v}_{162} \to \mathbf{v}_{164}$. The nodes on a path can either be stated by their names as strings or IDs as integers, or in a separate object as exemplified below.

```
R> path <- c(89, 92, 111, 162, 164)
R> PathTotalWeight(intnet_chicago, path = path)
[1] 3
```

3.6. Autocorrelation

To increase its applicability to different contexts and allow for a high level of flexibility within a unified framework, the **intensitynet** packages provide several correlation statistics in addition to the intensity function estimation including local and global versions of Moran I (Moran 1950), Geary's C (Geary 1954), and Getis and Ords G statistic (Getis and Ord 1992). While these global statistics help to quantify potential interrelations among the intensity functions over the complete network, their local counterparts quantify the contribution of the intensity functions for each element on the global one. As such, the local versions allow the identification of important hot- or coldspots on the network (see Anselin 2018, for detailed treatment and mathematical details).

Both global and local network correlation functions can directly be computed through the `NodeGeneralCorrelation()` and `NodeLocalCorrelation()` functions, respectively, from the `intensitynet` object, each of which requires three arguments including (i) an `intensitynet` object, (ii) the type of statistic (either `moran`, `getis`, or `geary`), and (iii) a vector containing the nodewise intensities. Recalling the internal computation for different types of graphs, this vector could correspond to different objects, i.e. `intensity_in`, `intensity_out`, `intensity_und`, or `intensity_all`. Using the adjacency matrix $\mathbf{A}$ to specify the neighborhood structure, i.e. the *lags* over the network, to compute the spatial autocorrelations among the distinct entities, any correlation function is only available at node-level. Formally, the underlying *lag structure* can be constructed for different orders, i.e. the k-order neighborhood, and includes either the partial or cumulative neighbors which derive directly from the corresponding adjacency matrices of and up to $\mathbf{A}^k$ of order k, respectively. We note that the implemented approach can also be used to include bivariate (Lee 2001) or partial and semi-partial (Eckardt and Mateu 2021a) autocorrelation statistics in a straightforward manner.

The `NodeLocalCorrelation()` functions return a list with the computational results stored as node attributes and an `intensitynet` object as arguments. The name of the node attribute corresponds to the type argument used in the computation, i.e. `moran`, `getis`, or `geary`. For the local Moran I, the function returns a $(n \times 5)$-dimensional table as output which contains the empirical local Moran I_i indices (`Ii`), the corresponding expected values $\mathbb{E}\left[i_i\right]$ (`E.Ii`) and variances $\mathbb{V}ar\left[i_i\right]$ (`Var.Ii`), Z-value (`Z.Ii`) and a p-valued (`Pr(z != E(Ii))`) based on a Normal distribution. This table is augmented by an `intensitynet` object (only storing the permutation data ($\mathbb{P}(z! = \mathbb{E}(I_i))$) from the node attributes). To illustrate its application, the next lines outline the computation of the `NodeLocalCorrelation()` using the `intnet_chicago` as a toy example.

Initially, to extract the nodewise intensity functions from the node attributes of the `intensitynet` object, the `vertex_attr()` function from the `igraph` package can be applied to the graph object `intnet_chicago$graph`. In addition to the mandatory `intensitynet` object, this information can then be used as an argument of the `NodeLocalCorrelation()` function to compute the desired local autocorrelation statistic. Depending on the `type` argument provided by the user, the function either computes the local Moran I (`moran`), Geary C (`geary`), or Getis G (`getis`).

```
R> intensity_vec <- igraph::vertex_attr(intnet_chicago$graph)$intensity

R> data_moran <- NodeLocalCorrelation(intnet_chicago,
+                                     dep_type =  "moran" ,
+                                     intensity = intensity_vec)
R> head(data_moran)
            Ii          E.Ii     Var.Ii       Z.Ii Pr(z != E(Ii))
V1  0.31576139 -0.0009369774 0.31640163  0.5630234      0.5734189
V2  0.12649811 -0.0009369774 0.07839415  0.4551423      0.6490069
V3  0.31576139 -0.0009369774 0.31640163  0.5630234      0.5734189
V4 -0.44129172 -0.0009369774 0.31640163 -0.7828586      0.4337102
V5 -0.04934006 -0.0018300486 0.15297793 -0.1214705      0.9033184
V6 -0.44129172 -0.0009369774 0.31640163 -0.7828586      0.4337102

R> intnet_chicago <- data_moran$intnet

R> data_geary <- NodeLocalCorrelation(intnet_chicago,
+                                     dep_type =  "geary" ,
+                                     intensity = intensity_vec)
R> head(data_geary$correlation)
[1] 0.0000000 0.4524253 0.0000000 1.8097012 1.0263252 1.8097012

R> intnet_chicago <- data_geary$intnet
R> data_getis <- NodeLocalCorrelation(intnet_chicago,
+                                     dep_type =  "getis" ,
+                                     intensity = intensity_vec)
R> head(data_getis$correlation)
[1] -0.5630234 -0.4551423 -0.5630234  0.7828586 -0.1214705  0.7828586

R> intnet_chicago <- data_getis$intnet
```

The correlation and covariance among the nodewise intensities, and the global Moran's I and Geary's C autocorrelation statistics provided by the `NodeGeneralCorrelation()` function are constructed as wrapper functions using the `nacf()` function from the package `sna` (Butts 2008) as the source. Internally, this function transforms the `intensitynet` object into a `sna` format to meet the desired input of the `nacf()` function. In addition to `intensitynet` object, the implemented function also requires (i) a type statement (`correlation`, `covariance`, `moran` or `geary`), (ii) a vector constructed from the nodewise intensities and (iii) a number specifying the *order* of $\mathbf{A}^k$. Apart from these mandatory arguments, the user could also specify if a partial neighborhood (default) or a cumulative lag structure up to order k should be used in the computations. The partial neighborhood for any given node v_i is defined as the set of nodes that are exactly k steps apart from v_i whereas the cumulative neighborhood of v_i subsumes all nodes whose distance to v_i is less than or equal to k. As output, the corresponding results are provided in form of a vector of length k, starting with zero neighbors where no lag information is used to the k-th neighbors as specified in the `order` statement. An application of the `NodeGeneralCorrelation()` function to the `intnet_chicago` is illustrated in the following lines.

Using the `intensity_vec` object from the previous example as input for our computations, execution of the following lines yields the partial covariances of the nodewise intensity functions for the sets of partial neighbors of orders 0 to 2.

```
R> NodeGeneralCorrelation(intnet_chicago,
+                         dep_type =  "covariance" ,
+                         lag_max = 2,
+                         intensity = intensity_vec)
           0            1            2
3.742840e-05 1.746413e-05 8.142177e-06
```

Likewise, the partial correlation for the sets of partial neighbors up to order $k = 5$ and its cumulative counterpart version (`partial_neighborhood= FALSE`) can be obtained by running the following code. Both quantities show a clear decrease in correlation from lag zero, corresponding to the correlation of the calculated nodewise intensity values with themselves, to lag five.

```
R>  NodeGeneralCorrelation(intnet_chicago,
+                          dep_type =  "correlation" ,
+                          lag_max = 5,
+                          intensity = intensity_vec)
         0          1          2          3          4          5
1.00000000 0.46522061 0.21689644 0.18078130 0.09940217 0.03468309

R>  NodeGeneralCorrelation(intnet_chicago,
+                          dep_type =  "correlation" ,
+                          lag_max = 5,
+                          intensity = intensity_vec,
+                          partial_neighborhood = FALSE)
        0         1         2         3         4         5
1.0000000 0.4652206 0.3012361 0.2402657 0.1827256 0.1317444
```

3.7. Data Visualization

Working with event data on relational systems, graphical outputs are of particular importance to describe the observed data and computed results. To allow for simple computations, different visualization tools are included in the **intensitynet** package. All main visual functionalities are provided by the `PlotHeatmap()` function which serves as a generic visualization tool and allows for the computation of graphically optimized heatmap representations of the network, either with or without any additional information on the event locations or estimated quantities. The function can directly be applied to any `intensitynet` object while the output can be specified through a wide range of additional arguments. The `PlotHeatmap()` function internally uses the `GeoreferencedGgplot2()` function from the `netTools` class which, in turn, is embedded into the **ggplot2** framework. As such, the function also allows for further arguments to manipulate the plot by customizing e.g. the edge or node size and color layers.

Without any further arguments, `PlotHeatmap()` yields a graphical output of the internally stored network structure including all edges and vertices (see Figure 3 for a visual representation of the underlying graph structure for the `intnet_orn` object presented in section 3.3.2). In addition to the basic visualization of the network structure itself, the precise event locations

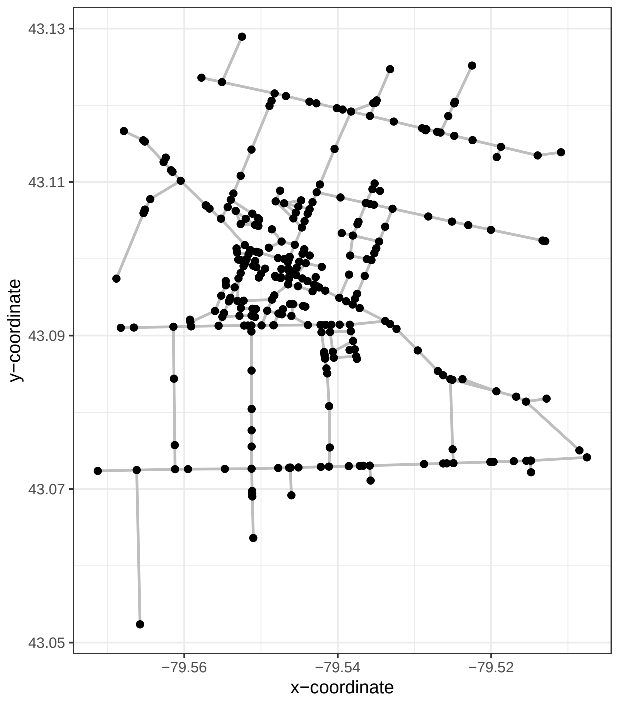


Figure 3: Imported network structured of the Ontario data example from section 3.3.2 which nodes represented as black dots and edges shown as solid gray lines

on the individual edges can be added to the output by setting the parameter `show_events` to `TRUE`. As a result, each event location is shown as orange squares. The transparency of the squares can be further customized by using the `alpha` argument.

Apart from the raw network and event information, the `PlotHeatmap()` function also allows for the representation of different results derived from the original input information including various local correlation functions, categorical marked point proportions or averages, and vertex or edge intensities as outlined in Sections 3.4 to 3.6. Any such intensity-based characteristics can be included in the output by specifying the `heat_type` argument. Examples of heatmap representations of the edgewise (`heat_type = e_intensity`) nodewise (`heat_type = v_intensity`) intensity functions, the mark proportions (`heat_type = trespass`), and Geary's C (`heat_type = geary`) are depicted in Figure 4.

In addition to the global representation of the network and all event-based computations over the complete relational system under study, the `PlotHeatmap()` function allows also the computation of graphical outputs for only some pre-selected vertices or edges. To construct a visualization of the localized network only, the selected vertices or edges need to be specified in advance either by using the vertex identifier or an `igraph` vertices or edges object, i.e. `igraph.vs` or `igraph.es`, respectively. The following lines of code outline the localized representation of the local Moran's I autocorrelation statistic for a list of pre-specified vertices (V66, V65, V64, V84, V98, V101, V116, V117, V118). The corresponding output is shown in the right bottom of Figure 4).

```
R> PlotHeatmap(intnet_chicago,
+             heat_type =  "moran" ,
+             net_vertices = c("V66",  "V65",  "V64",
+                              "V84",  "V98",  "V101",
+                              "V116",  "V117", "V118"))
```

In addition to the `PlotHeatmap()` function, the **intensitynet** package also provides a `plot()` function which is internally designed as a wrapper function of the `GeoreferencedPlot()` function provided in the auxiliary main class `netTools` and allows to uses the capabilities of the `plot.igraph()` function from the **igraph** package. As such, the `plot()` function allows to modify the nodes and edge labels of the network output and to highlight movements along the networks, i.e. paths, by specifying the corresponding vertex identifiers. Moreover, this function also allows for a proper spatial embedding of the network and the plotting of background grids, corresponding coordinates on the axes, and highlighting of event locations using simple flag arguments (`TRUE`, `FALSE`). As in the `PlotHeatmap()` function, the transparency of the events can be determined by using the `alpha` parameter. Finally, using the capacities of the `plot.igraph()` function, the `plot()` function allows to modify the output by including any graph analytic quantity from the **igraph** toolbox including communities or centrality measure based weights. Its applications to the `intnet_chicago` object is illustrated in the next lines of code.

```
R> plot(intnet_chicago, show_events = TRUE)
R> short_path <- ShortestPath(intnet_chicago,
+                            node_id1 =  "V1" ,
+                            node_id2 =  "V300" ,
+                            weight =  "intensity")
R> plot(intnet_chicago, show_events = TRUE, path = short_path$path)
```

Both the `PlotHeatmap()` and the `plot()` functions are augmented by the `PlotNeighborhood()` function which allows to highlight the first-order neighborhood for user-specified nodes and related event locations (highlighted by red circles). Execution of the code below yields the corresponding results for node v_{100} ($V100$), the resulting plot from the execution of this code is shown in Figure 5 on the right.

```
R> PlotNeighborhood(intnet_chicago, node_id =  "V100")
```

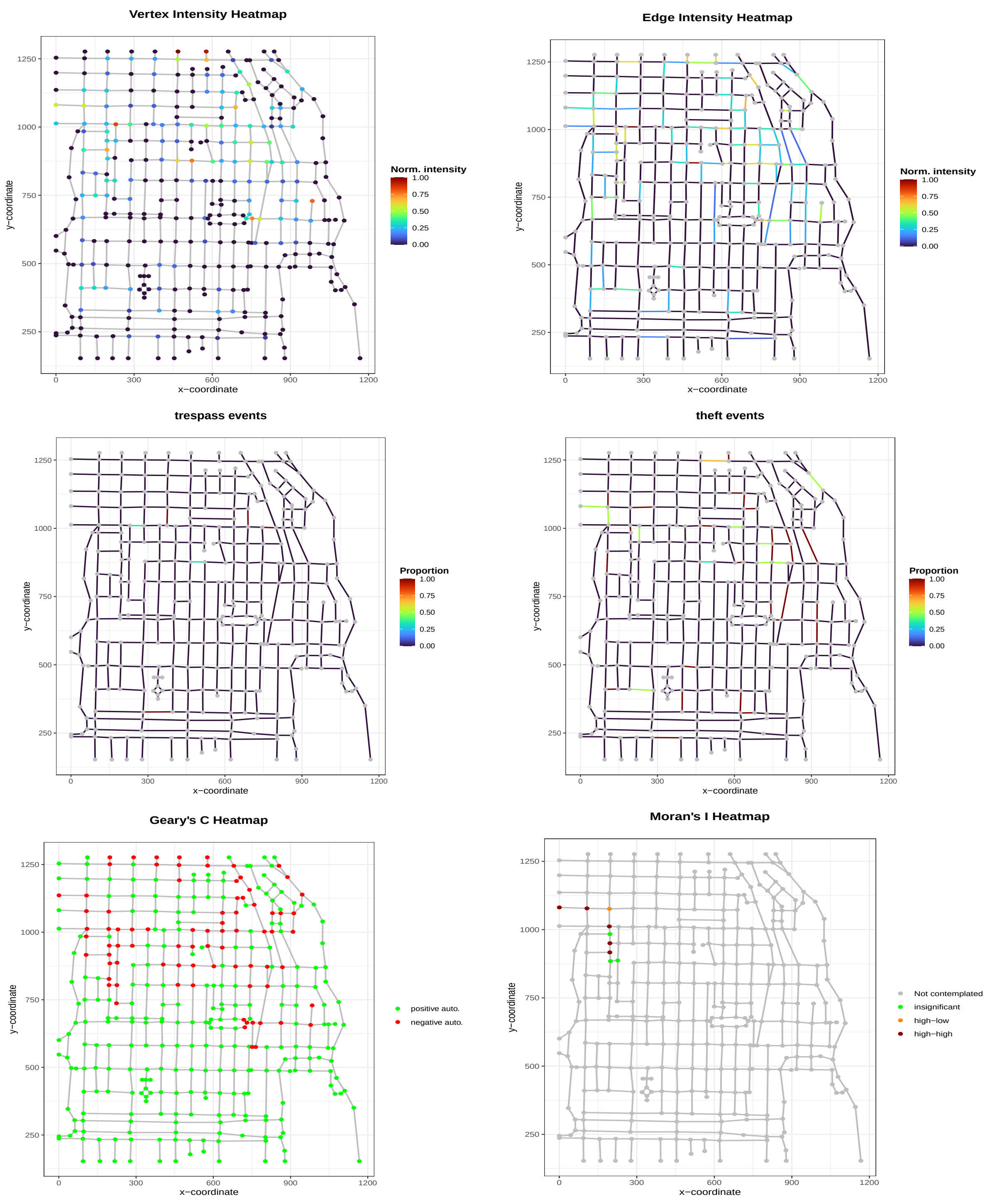


Figure 4: Heatmap representations of network intensity functions and derived characteristics computed from the `chicago` data with the rows defined as follows: The top row depicts the heatmaps for the nodewise (left) and edgewise intensity functions (right), the central row consists of the heatmaps constructed from the edgewise proportions of the crime categories *trespass* (left) and *theft* (right), and the bottom panels show the heatmap representations of the local Geary's C (left) and Moran's I (right) autocorrelation statistics. Positive and negative autocorrelations are highlighted in both Geary's C and Morans I heatmaps.

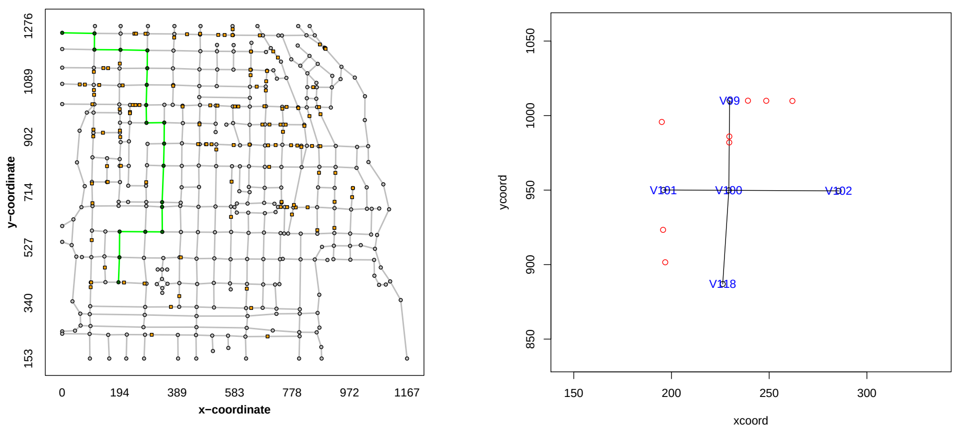


Figure 5: Selected outputs of the `plot()` and `PlotNeighborhood()` functions computed from the `intnet_chicago` object. Shortest path from v_1 to v_{300} along the network which avoids criminal areas highlighted in the color green (left), first-order neighborhood and incident edges for node v_{100} and closest raw event locations (red circles).

3.8. Going beyond points

Embedded into a graph theoretic formulation, the `intensitynet` package allows for additional investigations of the network event patterns that go well beyond the classic approaches in spatial point process analysis. The network assigned event locations or computed network intensity functions allow for the specification of advanced spatial (spatio-temporal) regression models for network-based responses and additional spatial covariates or mark information (see Eckadt, Klein, Greven, and Mateu 2022, for recent regression model for network-based response). Apart from regression specifications, the computed results can also be used to investigate the dependence or variation of different types of events, i.e. crimes, or to embed the observed network event patterns as graph-valued data into the framework of object-oriented data analysis. Finally, the inherent formulation of the network characteristics within the **igraph** framework allows using the well-established methodological toolbox and mathematical concepts of applied graph theory to detect communities and clusters, compute centrality and connectivity measures, and apply spectral analysis methods.

Using the `chicago` data as a toy example, the following code investigates the potential correlation between the nodewise intensities of trespass and robbery events.

```
R> chicago_trespass <- chicago_df[chicago_df$marks ==  "trespass" ,]
R> trespass_intnet <- intensitynet(adj_mtx,
+                                    node_coords = node_coords,
+                                    event_data = chicago_trespass)
R> trespass_intnet <- RelateEventsToNetwork(trespass_intnet)
Calculating edge intensities with event error distance of 5...
    |============================================================| 100%
```

```
Calculating node intensities...
    |=============================================================| 100%

R> chicago_robbery <- chicago_df[chicago_df$marks ==  "robbery" ,]
R> robbery_intnet <- intensitynet(chicago_adj_mtx,
+                                 node_coords = chicago_node_coords,
+                                 event_data = chicago_robbery)
R> robbery_intnet <- RelateEventsToNetwork(robbery_intnet)
Calculating edge intensities with event error distance of 5...
  |=============================================================| 100%

Calculating node intensities...
  |=============================================================| 100%
R> trespass_intensity <- igraph::vertex_attr(trespass_intnet$graph, "intensity")
R> robbery_intensity <- igraph::vertex_attr(robbery_intnet$graph, "intensity")
R > cor(trespass_intensity, robbery_intensity)
[1] 0.3559283
```

Extracting the `igraph` network structure from an `intensitynet` object, the next lines illustrate the computation of graph theoretic quantities from the `intnet_chicago` object.

```
R> g <- intnet_chicago$graph

R> degree <- igraph::degree(g)
R> head(degree)
V1 V2 V3 V4 V5 V6
 1  4  1  1  4  1
R> max(degree)
[1] 5

R> head(igraph::betweenness(g))
    V1     V2     V3     V4     V5     V6
  0.00 677.50   0.00   0.00 797.25   0.00

R> head(igraph::edge_betweenness(g))
[1] 337.00 337.00 461.75 556.25 337.00 527.00

R> head(igraph::closeness(g))
          V1           V2           V3           V4           V5           V6
2.850374e-06 3.183678e-06 3.101290e-06 3.303551e-06 3.399421e-06 3.481001e-06
```

4. Summary and discussion

This paper has introduced the general implementation and main functionalities of the **intensitynet** package including exploratory analysis tools, network adjusted summary and autocorrelation statistics, and visualization capacities. Bridging the general ideas from graph theory into the analysis of structured spatial event-type data, the package allows for the analysis of potential events on different types or combinations of edges and also disconnected subgraph structures. The package is build around a set of generic intensities functions which derive directly from edgewise computations and allow to address different graph theoretic entities and sets in the network under study and help to investigate the variation in numbers of events over the graph object. Embedded into the **igraph** framework, the **intensitynet** sets the ground for even further analysis based on graph theory methodologies and also easy importation into related R packages.

Although the package is designed to work with any network size and edge configuration, both the size of the network and the number of events under study affect the computing time. To address the computational burden and structure the necessary computational operations, the package is implemented in the two separate functions (`intensitynet()` and `RelateEventsToNetwork()`). This enables the user to decide to either obtain some intensity based statistics or work explicitly with the network representation in the form of an **igraph** class object.
To test the reliability of our package, we ran numerous unitary tests, and also general tests, using multiple datasets, some of them with dummy data and some involving real datasets such as the Chicago data presented in this work. Additionally, the package has passed all the CRAN repository policies and checks. Moreover, the package is in constant revision and development, to detect and amend unnoticed bugs and other possible code-related problems.

About future development, some of the features that could be implemented are (1) consider more input options to initialize the `intensitynet()`, (2) incorporate an occurrence legend in the plots `plot()` or `PlotHeatmap()` if the occurrences are chosen to be shown, (3) consider multiple output options to save the network model in different formats, and finally (4) develop an application to access interactively all the tools that our package offers, using the **Shiny** package.

Acknowledgments

The authors gratefully acknowledge the financial support of the Barcelona economic association *Amics del País* and the support of the MCIN/AEI/ 10.13039/501100011033 (Spanish Government) through Grant PID2020-115442RB-I00, Matthias Eckardt was funded by the Walter Benjamin grant 467634837 from the German Research Foundation.

References

Anderes E, Møller J, Rasmussen JG (2020). "Isotropic covariance functions on graphs and their edges." *The Annals of Statistics*, **48**(4), 2478 – 2503. `doi:10.1214/19-AOS1896`.

Ang Q, Baddeley A, Nair G (2012). "Geometrically Corrected Second Order Analysis of Events on a Linear Network, with Applications to Ecology and Criminology." *Scandinavian Journal of Statistics*, **39**(4), 591–617. `doi:10.1111/j.1467-9469.2011.00752.x`.

Anselin L (2018). "A Local Indicator of Multivariate Spatial Association: Extending Geary's C." *Geographical Analysis*, **51**(2), 133–150. `doi:10.1111/gean.12164`.

Baddeley A, Nair G, Rakshit S, McSwiggan G, Davies TM (2021). "Analysing point patterns on networks — A review." *Spatial Statistics*, **42**, 100435. doi:10.1016/j.spasta.2020.100435.

Baddeley A, Rubak E, Turner R (2015). *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press. doi:10.1007/s11004-016-9670-x.

Baddeley A, Turner R (2005). "spatstat: An R Package for Analyzing Spatial Point Patterns." *Journal of Statistical Software*, **12**(6), 1–42. doi:10.18637/jss.v012.i06.

Bivand R, Lewin-Koh N (2019). *maptools: Tools for Handling Spatial Objects*. R package version 0.9-9, URL https://CRAN.R-project.org/package=maptools.

Briz-Redon A (2021a). *DRHotNet: Differential Risk Hotspots in a Linear Network*. R package version 2.0, URL https://CRAN.R-project.org/package=DRHotNet.

Briz-Redon A (2021b). *SpNetPrep: Linear Network Preprocessing for Spatial Statistics*. R package version 1.2, URL https://CRAN.R-project.org/package=SpNetPrep.

Butts CT (2008). "Social Network Analysis with sna." *Journal of Statistical Software*, **6**(24), 1–51. doi:10.18637/jss.v024.i06.

Csardi G, Nepusz T (2006). "The igraph software package for complex network research." *InterJournal*, **Complex Systems**, 1695. URL http://igraph.org.

Diestel R (2016). *Graph Theory*. Springer. ISBN 978-3-662-53621-6.

Eckadt M, Klein N, Greven S, Mateu J (2022). "Structured additive distributional regression for spatial point patterns on network domains." *Working paper*.

Eckardt M, Mateu J (2018). "Point Patterns Occurring on Complex Structures in Space and Space-Time: An Alternative Network Approach." *Journal of Computational and Graphical Statistics*, **27**(2), 312–322. doi:10.1080/10618600.2017.1391695.

Eckardt M, Mateu J (2021a). "Partial and Semi-Partial Statistics of Spatial Associations for Multivariate Areal Data." *Geographical Analysis*, **53**(4), 818–835. doi:10.1111/gean.12266.

Eckardt M, Mateu J (2021b). "Second-order and local characteristics of network intensity functions." *TEST*, **30**(2), 318–340. doi:10.1007/s11749-020-00720-4.

Fox H, Bolton W (2002). *Mathematics for Engineers and Technologists*. Butterworth-Heinemann.

Geary RC (1954). "The contiguity ratio and statistical mapping." *The Incorporated Statistician*, **5**(3), 115–146. doi:10.2307/2986645.

Gelb J (2021a). "spNetwork, a package for network kernel density estimation." *The R Journal*. doi:10.32614/RJ-2021-102. URL https://journal.r-project.org/archive/2021/RJ-2021-102/index.html.

Gelb J (2021b). *spNetwork: Spatial Analysis on Network*. URL https://jeremygelb.github.io/spNetwork/.

Getis A, Ord JK (1992). "The Analysis of Spatial Association by Use of Distance Statistics." *Geographical Analysis*, **24**(3), 189–206. doi:10.1007/978-3-642-01976-0_10.

Johnson D (1977). "Efficient Algorithms for Shortest Paths in Sparse Networks." *Journal of the ACM*, **24**(1), 1–13. doi:10.1145/321992.321993.

Lee SI (2001). "Developing a bivariate spatial association measure: An integration of Pearson's r and Moran's I." *Journal of Geographical Systems*, **3**, 369–385. doi:10.1007/s101090100064.

Lu B, Sun H, Xu M, Harris P, Charlton M (2018). "Shp2graph: Tools to Convert a Spatial Network into an Igraph Graph in R." *ISPRS International Journal of Geo-Information*, **7**(8), 293. doi:10.3390/ijgi7080293.

Magzhan K, Jani H (2013). "A Review And Evaluations Of Shortest Path Algorithms." *International journal of scientific & technology research*, **2**(6). ISSN 2277-8616.

Moradi M, Cronie O, Mateu J (2020). *stlnpp: Spatio-temporal analysis of point patterns on linear networks.*

Moran PAP (1950). "Notes on continuous stochastic phenomena." *Biometrika*, **37**(1/2), 17–23. doi:10.2307/2332142.

Okabe A, Yamada I (2001). "The K-Function Method on a Network and Its Computational Implementation." *Geographical Analysis*, **33**(3), 271–290. doi:10.1111/j.1538-4632.2001.tb00448.x.

Okabe A, Yomono H, Kitamura M (1995). "Statistical Analysis of the Distribution of Points on a Network." *Geographical Analysis*, **27**(2), 152–175. doi:10.1111/j.1538-4632.1995.tb00341.x.

Ontario P (2006). "Ontario Road Network." Data retrieved from Ontario Ministry of Natural Resources: Peterborough, ON, Canada, http://www.geographynetwork.ca/website/orn/viewer.htm.

QGIS Development Team (2022). *QGIS Geographic Information System.* QGIS Association. URL https://www.qgis.org.

R Core Team (2021). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rahman MS (2017). *Paths, Cycles, and Connectivity*, pp. 31–46. Springer International Publishing, Cham. ISBN 978-3-319-49475-3. doi:10.1007/978-3-319-49475-3_3.

Rasmussen JG, Christensen HS (2021). "Point Processes on Directed Linear Networks." *Methodology and Computing in Applied Probability*, **23**(2), 647–667.

Redlands CESRI (2011). *ArcGIS Geographic Information System.* URL https://www.arcgis.com.

Schneble M (2021). *geonet: Intensity Estimation on Geometric Networks with Penalized Splines.* R package version 0.6.0, URL https://CRAN.R-project.org/package=geonet.

van Lieshout MNM (2018). “Nearest-neighbour Markov point processes on graphs with Euclidean edges.” *Advances in Applied Probability*, **50**(4), 1275–1293. doi:10.1017/apr.2018.60.

Affiliation:

Carles Comas
School of Agrifood and Forestry Science and Engineering
Universitat de Lleida
Campus Agroalimentari, Forestal i Veterinari
Av. de l'Alcalde Rovira Roure 191,
25198 Lleida, Spain
E-mail: carles.comas@udl.cat
URL: http://www.math.udl.cat/en/staff_CarlesComasRodriguez

Matthias Eckardt
School of Business and Economics
Chair of Statistics
Unter den Linden 6
10099 Berlin, Germany
E-mail: m.eckardt@hu-berlin.de
URL: http://hu.berlin/eckardtm

Pol Llagostera Blasco
Polytechnic School
Universitat de Lleida
Carrer Jaume II, 69
25001 Lleida, Spain
E-mail: pol.llagostera@udl.cat
URL: http://www.math.udl.cat/en/staff_PolLlagosteraBlasco

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd

Chapter 6

Other Research Activities

6.1 Conferences

The author of this thesis has contributed to the following conferences:

- P. Llagostera, C. Comas, N. López, On the Computation of the Reliability Polynomial: Graph Masters, Lleida, Spain, November 29-30, 2018.
- P. Llagostera, C. Comas, N. López, On the Computation of the Reliability Polynomial of Cycle Graphs with Ordered Path-Chords: International Workshop on Combinatorial and Computational Aspects of Optimization, Topology, and Algebra (ACCOTA), Merida, Yucatan, Mexico, December 2-7, 2018.
- P. Llagostera, N. López, J. Conde, C. Comas, Network Reliability Based on Spatial Point Patterns: Spatial Statistics, towards spatial data science, Sitges, Spain, July 10-13, 2019.
- P. Llagostera, C. Comas, C. Dalfó, N. López, Optimal Path Selection for Road Traffic Safety based on wildlife-vehicle collisions: 10th International Workshop on Spatio-Temporal Modelling (METMA), Lleida, Spain, June 1-3, 2022.
- P. Llagostera, C. Comas, N. López, Computing the Reliability Polynomial of Cake Graphs Efficiently: Graph Masters, Lleida, Spain, October 10 and 12, 2022.
- C. Comas, P. Llagostera, N. López, Riesgo de accidente de tráfico en networks lineales basado en patrones puntuales: SEIO, Granada, Spain, June 7-10, 2022.

6.1.1 Posters

The posters presented in this section had been presented during the development of this thesis:

On the computation of the realiability polynomial of cycle graphs with ordered path-chords

Pol Llagostera, Nacho López, Carles Comas.
{pol.llagostera, nlopez, carles.comas}@matematica.udl.es

Universitat de Lleida

Introduction

The reliability polynomial of a network gives the probability that the network remains operational when all its edges could fail independently with certain fixed probability. In general, the problem of computing the reliability polynomial becomes intractable when the size of the network increases. In this paper, we give some exact formulas for the realibility polynomial of certain networks. Combining these results with a new algorithm approach, we improve the computation of the realibility polynomial for networks satisfying certain conditions.

The Reliability Polynomial

Let $|G|$ denote the number of edges of a network $G = (V, E)$. Let us consider the set $\mathcal{G}$ of connected spanning subgraphs of G. Then, the probability that G is connected, as a function of p, is

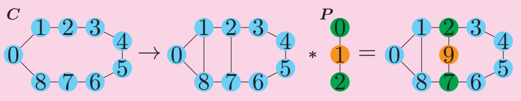
$$\sum_{G' \in \mathcal{G}} p^{|G'|}(1-p)^{m-|G'|} \tag{1}$$

This formula is known as *the reliability polynomial of* G, and it is denoted as $\text{Rel}(G, p)$. Formula (1) requieres the computation of all connected spanning subgraphs, so it is useful just for a few family of graphs, like Complete graphs, Cycle graphs or Trees. Of course $\text{Rel}(G, p) = 1$ if G is a single vertex and $\text{Rel}(G, p) = 0$ if G is disconnected, for the remaining cases the following recursive formula, also known as *the factoring theorem* is an alternative method for computing $\text{Rel}(G, p)$ based on the combination of two graph operations: edge deletion $G - e$, and edge contraction G/e:

$$\text{Rel}(G,p) = \begin{cases} \text{Rel}(G-e,p) & \text{if } e \text{ is a loop,} \\ p\text{Rel}(G/e,p) & \text{if } e \text{ is a cut-edge,} \\ (1-p)\text{Rel}(G-e,p) + p\text{Rel}(G/e,p) & \text{otherwise.} \end{cases} \tag{2}$$

Cycle graphs with ordered path-chords

A Cycle graph with ordereded path-chords G is obtained from a Cycle $C_n : 0, 1, \ldots n-1$ by adding some chords (x_i, y_i), $1 \le i \le t$, such that $x_1 < x_2 < \cdots < x_t < y_t < y_{t-1} < \cdots < y_2 < y_1$ Then, we replace every chord by a path graph in such a way that the endpoints of the path graph are the end points of the corresponding chord.



New Approach Formulae

Definitions:

$[A]^n$**:** Set of all the subsets of A with exactly n elements
k**:** Constants
m**:** Total edges
H**:** Set of graphs path
$H = [B_1, B_2, \cdots, B_t]$
H'**:** List containing the number of edges that has each element of H
$H' = [\tilde{Q}_1, \tilde{Q}_2, \cdots, \tilde{Q}_z]$
L**:** Set of cycles containded in the graph
$L = [C_1, C_2, \cdots, C_n]$
L'**:** List of cycle lengths without its shared edges
$L' = L - H$
$L' = [\tilde{F}_1, \tilde{F}_2, \cdots, \tilde{F}_j]$

$$Rel(G,p) = \sum_{n=0}^{|L|} (k_n \cdot p^{e-n}(1-p)^n) \tag{5}$$

Constants (k):

- k_0 **:** 1
- k_1 **:** e
- k_2 **:**

$$\sum_{i<j} |\tilde{F}_i| \cdot |\tilde{F}_j| + \sum_{i=1}^{|H|} |H_i| \cdot (e - |H|) + \sum_{i<z} |\tilde{Q}_i| \cdot |\tilde{Q}_z| \tag{6}$$

- $k_{(n \ge 3)}$**:**

$R = [H]^{|H|} = [r_1, r_2, \cdots, r_d]$
$G_i = G - r_i \rightarrow S = [G'_1, G'_2, \cdots, G'_d]$
$d \in h \in [H']^j$
$f \in l \in [L']^{(|L'|-j)}$

$$\sum_{i=1}^{|S|} G'_i = \sum_{j=1}^{|L'|} (\prod_{v=1}^{|h_j|} d_v \cdot \prod_{v=1}^{|l_b|} f_v) \tag{7}$$

New Formulas

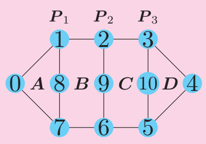
Let $G(l_1, \ldots, l_L)$ be a graph obtained from a Tree by replacing L edges by cycles of length $l_1, l_2, \ldots, l_L$, then

$$\text{Rel}(G(l_1,\ldots,l_L),p) = \\ p^m + mp^{m-1}(1-p) + \sum_{s=2}^{L} (\sum_{i_1 < \cdots < i_s} l_{i_1} \ldots l_{i_s} p^{m-s}(1-p)^s) \tag{3}$$

where m denotes the number of edges of $G(l_1, \ldots, l_L)$.
Let MT be a multitree graph, i.e. a graph obtained from a Tree by replacing L edges by a set of multiedges each of cardinality m_i, $1 \le i \le L$. Then

$$\text{Rel}(MT,p) = \prod_{i=1}^{L} (1 - (1-p)^{m_i}) \tag{4}$$

Example



$$p^{14} + \mathbf{14} \cdot p^{13}(1-p) + \mathbf{84} \cdot p^{12}(1-p)^2 \\ + \mathbf{256} \cdot p^{11}(1-p)^3 + \mathbf{336} \cdot p^{10}(1-p)^4 \tag{8}$$

Two broken edges (k_2)

A·B + A·C + A·D + B·C + B·D + C·D:
2·2 + 2·2 + 2·2 + 2·2 + 2·2 + 2·2 = 24

$\neg P_1 + \neg P_2 + \neg P_3$
2·8 + 2·8 + 2·8 = 48

$\neg(P_1P_2) + \neg(P_1P_3) + \neg(P_2P_3)$
2·2 + 2·2 + 2·2 = 12

Total = 24 + 48 + 12 = **84**

Three broken edges (k_3)

A·B·C + A·B·D + A·C·D + B·C·D:
2·2·2 + 2·2·2 + 2·2·2 + 2·2·2 = 32

$\neg P_1 + \neg P_2 + \neg P_3$
((2)·4·2 + (2)·4·2 + (2)·2·2) +
((2)·4·2 + (2)·4·2 + (2)·2·2) +
((2)·4·2 + (2)·4·2 + (2)·2·2) = 120

$\neg(P_1P_2) + \neg(P_1P_3) + \neg(P_2P_3)$
((2·2)·6 + (2·2)·2) +
((2·2)·4 + (2·2)·4) +
((2·2)·6 + (2·2)·2) = 96

$\neg(P_1P_2P_3)$
(2·2·2) = 8

Total = 32 + 20 + 96 + 8 = **256**

4 broken edges (k_4)

A·B·C·D:
2·2·2·2 = 16

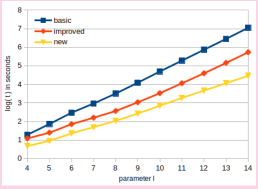
$\neg P_1 + \neg P_2 + \neg P_3$
(2)·4·2·2 +
(2)·4·2·2 +
(2)·4·2·2 = 96

$\neg(P_1P_2) + \neg(P_1P_3) + \neg(P_2P_3)$
(2·2)·6·2 +
(2·2)·4·4 +
(2·2)·6·2 = 160

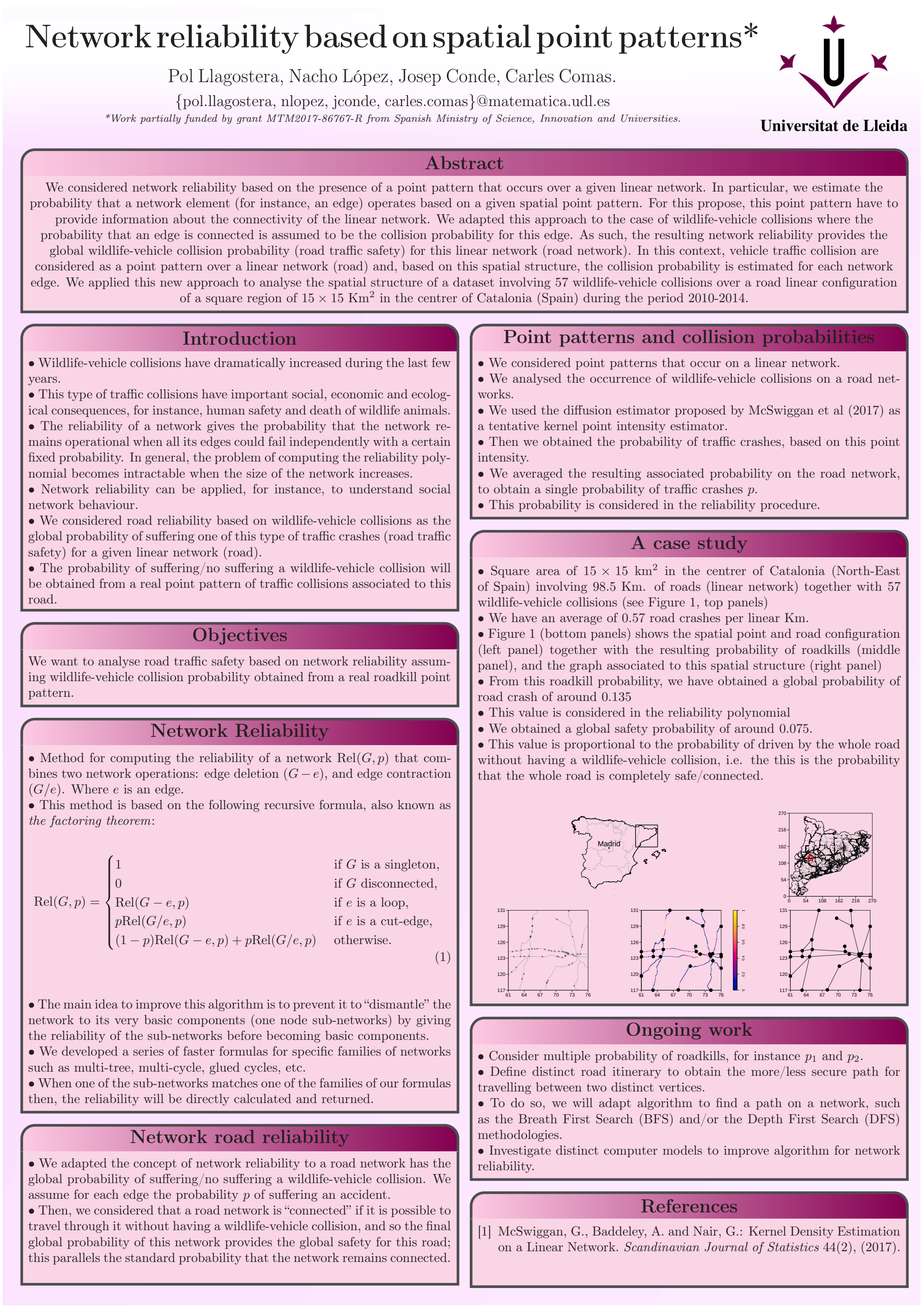
$\neg(P_1P_2P_3)$
(2·2·2)·8 = 64

Total = 16 + 96 + 160 + 64 = **336**

Computer Results for $P_2 \times P_l$



Computing times (s)			
Graph	**Old**	**Imp.**	**New**
P_2xP_4	0,19s	0,12s	0,05s
P_2xP_5	0,73s	0,25s	0,09s
P_2xP_6	2,97s	0,72s	0,23s
...	...	...	...
P_2xP_{12}	2,03h	6,55m	46,33s
P_2xP_{13}	7,82h	23,89m	1,92m
P_2xP_{14}	31,11h	1,49h	6,42m



Network reliability based on spatial point patterns*

Pol Llagostera, Nacho López, Josep Conde, Carles Comas.

{pol.llagostera, nlopez, jconde, carles.comas}@matematica.udl.es

*Work partially funded by grant MTM2017-86767-R from Spanish Ministry of Science, Innovation and Universities.

Universitat de Lleida

Abstract

We considered network reliability based on the presence of a point pattern that occurs over a given linear network. In particular, we estimate the probability that a network element (for instance, an edge) operates based on a given spatial point pattern. For this propose, this point pattern have to provide information about the connectivity of the linear network. We adapted this approach to the case of wildlife-vehicle collisions where the probability that an edge is connected is assumed to be the collision probability for this edge. As such, the resulting network reliability provides the global wildlife-vehicle collision probability (road traffic safety) for this linear network (road network). In this context, vehicle traffic collision are considered as a point pattern over a linear network (road) and, based on this spatial structure, the collision probability is estimated for each network edge. We applied this new approach to analyse the spatial structure of a dataset involving 57 wildlife-vehicle collisions over a road linear configuration of a square region of 15×15 Km^2 in the centre of Catalonia (Spain) during the period 2010-2014.

Introduction

• Wildlife-vehicle collisions have dramatically increased during the last few years.
• This type of traffic collisions have important social, economic and ecological consequences, for instance, human safety and death of wildlife animals.
• The reliability of a network gives the probability that the network remains operational when all its edges could fail independently with a certain fixed probability. In general, the problem of computing the reliability polynomial becomes intractable when the size of the network increases.
• Network reliability can be applied, for instance, to understand social network behaviour.
• We considered road reliability based on wildlife-vehicle collisions as the global probability of suffering one of this type of traffic crashes (road traffic safety) for a given linear network (road).
• The probability of suffering/no suffering a wildlife-vehicle collision will be obtained from a real point pattern of traffic collisions associated to this road.

Objectives

We want to analyse road traffic safety based on network reliability assuming wildlife-vehicle collision probability obtained from a real roadkill point pattern.

Network Reliability

• Method for computing the reliability of a network Rel(G, p) that combines two network operations: edge deletion $(G - e)$, and edge contraction (G/e). Where e is an edge.
• This method is based on the following recursive formula, also known as *the factoring theorem*:

$$\mathrm{Rel}(G,p) = \begin{cases} 1 & \text{if } G \text{ is a singleton,} \\ 0 & \text{if } G \text{ disconnected,} \\ \mathrm{Rel}(G-e,p) & \text{if } e \text{ is a loop,} \\ p\mathrm{Rel}(G/e,p) & \text{if } e \text{ is a cut-edge,} \\ (1-p)\mathrm{Rel}(G-e,p)+p\mathrm{Rel}(G/e,p) & \text{otherwise.} \end{cases} \tag{1}$$

• The main idea to improve this algorithm is to prevent it to "dismantle" the network to its very basic components (one node sub-networks) by giving the reliability of the sub-networks before becoming basic components.
• We developed a series of faster formulas for specific families of networks such as multi-tree, multi-cycle, glued cycles, etc.
• When one of the sub-networks matches one of the families of our formulas then, the reliability will be directly calculated and returned.

Network road reliability

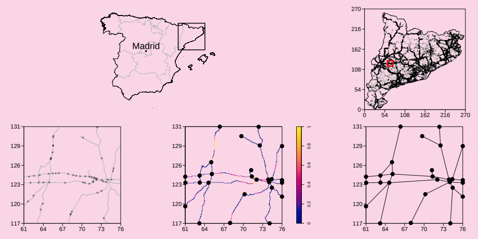
• We adapted the concept of network reliability to a road network has the global probability of suffering/no suffering a wildlife-vehicle collision. We assume for each edge the probability p of suffering an accident.
• Then, we considered that a road network is "connected" if it is possible to travel through it without having a wildlife-vehicle collision, and so the final global probability of this network provides the global safety for this road; this parallels the standard probability that the network remains connected.

Point patterns and collision probabilities

• We considered point patterns that occur on a linear network.
• We analysed the occurrence of wildlife-vehicle collisions on a road networks.
• We used the diffusion estimator proposed by McSwiggan et al (2017) as a tentative kernel point intensity estimator.
• Then we obtained the probability of traffic crashes, based on this point intensity.
• We averaged the resulting associated probability on the road network, to obtain a single probability of traffic crashes p.
• This probability is considered in the reliability procedure.

A case study

• Square area of 15×15 km^2 in the centre of Catalonia (North-East of Spain) involving 98.5 Km. of roads (linear network) together with 57 wildlife-vehicle collisions (see Figure 1, top panels)
• We have an average of 0.57 road crashes per linear Km.
• Figure 1 (bottom panels) shows the spatial point and road configuration (left panel) together with the resulting probability of roadkills (middle panel), and the graph associated to this spatial structure (right panel)
• From this roadkill probability, we have obtained a global probability of road crash of around 0.135
• This value is considered in the reliability polynomial
• We obtained a global safety probability of around 0.075.
• This value is proportional to the probability of driven by the whole road without having a wildlife-vehicle collision, i.e. the this is the probability that the whole road is completely safe/connected.



Ongoing work

• Consider multiple probability of roadkills, for instance p_1 and p_2.
• Define distinct road itinerary to obtain the more/less secure path for travelling between two distinct vertices.
• To do so, we will adapt algorithm to find a path on a network, such as the Breath First Search (BFS) and/or the Depth First Search (DFS) methodologies.
• Investigate distinct computer models to improve algorithm for network reliability.

References

[1] McSwiggan, G., Baddeley, A. and Nair, G.: Kernel Density Estimation on a Linear Network. *Scandinavian Journal of Statistics* 44(2), (2017).

Optimal path selection for road traffic safety*

P. Llagostera[1], C. Comas[1], C. Dalfó[1] and N. López[1,*]

{pol.llagostera, nacho.lopez, cristina.dalfo, carles.comas}@.udl.cat

*Work partially funded by grant PID2020-115442RB-I00 from MCIN/AEI/ 10.13039/501100011033.

Universitat de Lleida

Abstract

Wildlife-vehicle collisions (WVC) present an important coexisting problem between human populations and the environment. These type of accidents threatens the life and safety of car drivers, cause property damage to vehicles, and affect wildlife populations. We present a new approach based on algorithms used to obtain minimum paths between vertices in weighted networks to obtain the optimal (safest) route between two points (departure and destination points) in a road structure based on wildlife-vehicle collision point patterns. We have adapted the road structure into a mathematical linear network and analysed it using graph theory methodologies. This new approach has been illustrated with a case study in the region of Catalonia, North-East of Spain. This example shows the usefulness of our new approach to identify optimal path between pair of vertices based on weighted edges.

Introduction

- Growing interest in the analysis and modelling of WVC's since they do not occur randomly neither in space nor in time.
- Important to identify areas with high accident intensity (hotspots).
- Point patterns on linear network is a tentative way to investigate such space events.
- Occurrence of WVC are affected by several ecological, biological, and meteorological covariates along with structural road characteristics.
- This suggests that distinct road section will have distinct occurrence of such events.
- Weighted graphs can be considered to model the risk of WVC, and to define distinct path configurations to optimize this risk.
- Our main aim in this paper is to develop a new approach adapting algorithms used to obtain minimum paths between vertices in weighted networks to model road traffic safety based on WVC point patterns.

Linear network and point patterns

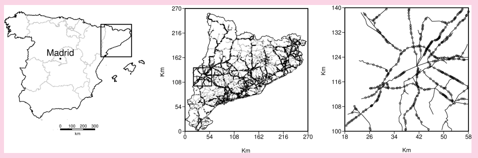
- Road configuration as a linear network L as defined in [1]
- Then we consider a point pattern on L
- We obtain an intensity estimator of this point pattern assuming the the diffusion estimator proposed by [2].
- We obtain the average intensity value for each edge of L as the integral of this intensity function over a edge
- Now each line segment has a weight that represents the average of WVC This value will be considered as the weight for each line

Minimum weighted path algorithms

- Several algorithms have been proposed to calculate the minimum path between two vertices in a weighted network
- These algorithms are defined to explore weighted networks in search of the path between two points that has minimum cost
- Cost is defined as the total sum of the weights associated with the edges of each path
- A path is a set of unique edges that connects distinct vertices starting from a vertex origin and ending with a vertex destination
- Two algothims are adapted; the Depth First Search (DFS) to calculate the total number of possible paths between two points, and, Yen's algorithm [3] to rank the top K-best paths between them.

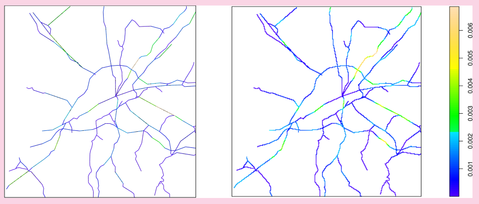
The Wildlife-vehicle collision dataset

- Case study; dataset containing 491 WVC occurred in a squared area ($40km \times 40km$) around the city of Lleida
- Road structure, 459.050 km of roads for three distinct road categories, namely, highways and paved roads, during the period $2010 - 2014$

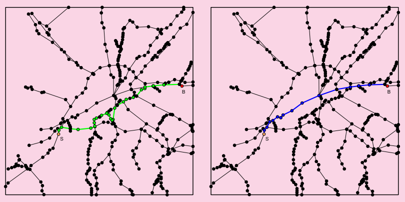


Analysing the dataset

- Next figure shows the resulting point density based on the diffusion estimator with a bandwidth value around 750 meter and the weighted network structure based on the average number of wildlife-vehicle collisions for each edge



- We consider a possible scenario to illustrate our approach where modified DFS and the Yen's algorithm are used to find the safest and the shortest path between two real town locations Soses and Bell-Lloc d'Urgell
- The figure below shows the safest path between this pair of origin/destination points together with the shortest path between these two locations.
- The safest path is not the shortest one between these two locations.
- Safest path is the best road route over 607.416 paths obtained by combining the 410 vertices and 437 edges of this linear network configuration.



Future work

- To consider traffic flow information in our optimization procedure
- For instance,raffic flow information, such as, vehicle traffic volume is crucial to full understand WVC.
- Consider other variables related with animal density and behaviour

References

[1] Ang, W., Baddeley, A., Nair, G., (2012). Geometrically corrected second order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591-617.

[2] McSwiggan, G., Baddeley, A., Nair, G., (2016). Kernel density estimation on a linear network. *Scandinavian Journal of Statistics* **44**, 324-345.

[3] Yen, J.Y., (1971). Finding the k shortest loopless paths in a network. *Network. Management Science* **17**, 661-786.

6.2 Abroad Research Stay

The candidate did a research stay from September 16, 2021, to December 21, 2021, at the Humboldt University of Berlin, Berlin, Germany, and has been supervised by Dr. Matthias Eckardt. This stay has been partially funded by the award from the Barcelona Economic Society ‘Amics del País’. In collaboration with the University of Berlin, this stay resulted in the development of the paper presented in section 5.3.

Chapter 7

Discussion and Conclusions

This Chapter summarizes the contents of this Thesis, discusses the methodology, results, limitations and implications, provides possible future research lines, and finalizes with some conclusions. The structure of this Chapter is presented as follows: Section 7.1 highlights the most important basic concepts and, summarizes the two aims of the thesis (Network Reliability and, Spatial Networks with Point Processes) by providing a brief introduction, state of art, methodology and, results for each aim. Section 7.2 analyzes and discusses the research presented in this Thesis as well as the results. Section 7.3 identifies and analyzes the limitations of the study along with its methodologies. Section 7.4 discusses the implications of the research and the results. Section 7.5 identifies possible future research lines based on the presented studies. Lastly, Section 7.6 presents the main concepts and highlights of this Thesis.

7.1 Summary

Many types of processes and relationships can be modeled into a network $G = (V, E)$ where V is a set of nodes and E is the set of edges. In combination with the branch of mathematics of Graph Theory, this model provides many useful advantages, for instance, to study optimization, networking, matching, and operational problems. This thesis is aimed to study the network's ability to remain connected, also known as network reliability, and the study of events occurring on spatial networks. To this end, the aims are divided into two sub-groups, network reliability and, spatial networks with point processes.

7.1.1 Network Reliability

The all-terminal reliability of a network refers to the ability of a network to remain operational when its edges have the same independent probability p to remain connected. In general, there are two main approaches to studying network reliability: the design of highly reliable networks and, efficient reliability calculation.

State of art

A network is uniformly most reliable (UMR), when there does not exist any other configuration with the same number of nodes n and edges e that produces a graph with higher reliability. The design of UMR networks has been studied in the literature by different authors, for instance, Boesch and Suffel [5] demonstrated that UMR networks exist when $e = n - 1$, n, $n + 1$, or $n + 2$, while Gross and Saccoman [24] extended these demonstrations to multi-edge networks, although Myrvold et al. [34] proved that UMR networks do not always exist. Regarding the construction of UMR networks, Smith and Lynne [51] show how to construct UMR regular graphs while Romero [45] provided a methodology to construct UMR through iterative augmentation. To the best of our knowledge, no research focused on the construction of UMR networks belonging to the hamiltonian family. To give more light on this topic, we studied this family, demonstrated the existence of UMR for hamiltonian networks with $m = n + 1$ and $m = n + 2$, and provided a computational approach for $m = n + 3$.

There exist several methodologies to calculate network reliability, for instance, Politof and Satyanarayana [39] and Satyanarayana and Tindell [48] developed efficient algorithms to analyze the reliability of planar networks, Satyanarayana and Chang [46], Burgos and Amoza [10] and Burgos [9] studied the use of the factoring theorem to calculate the network reliability, Srivaree-ratana et al. [52] estimated the reliability with neural networks and, Shafrat et al. [50] purposed a recursive truncation algorithm. This thesis presents a new modular methodology to calculate the all-terminal reliability of undirected networks that combines the generality of the contraction-deletion algorithm [37] with the computational speed of specific algorithms for certain network families. This methodology is implemented in form of a ***Python*** package and is designed to be easily expanded with new modules without requiring to modify the source code.

Methodology

A hamiltonian network can be thought of as a simple cycle network C_n with additional edges such that $m = n + 1, n + 2, n + 3 \ldots$ Therefore, to construct a UMR hamiltonian network, the number of edges is taken into account. With $m = n + 1$, the additional edge must connect two vertices of C_n at maximum distance, this creates a c-path length vector (x_1, x_2) such that $n = x_1 + x_2$, where $x_2 - 1 \leq x_1 \leq x_2 + 1$. With $m = n + 2$, the two additional edges must cross each other and generate a c-path length vector (x_1, x_2, x_3, x_4) where $x_i = x_{i+1}$ or $max(x_i) - 1 = min(x_i)$. This seems to be also the case with $m = n + 3$ where the c-path length vector is $(x_1, x_2, x_3, x_4, x_5, x_6)$ and $x_i = x_{i+1}$ or $max(x_i) - 1 = min(x_i)$. The presented UMR hamiltonian network configurations follow the structure from the Fair-Cake networks.

The given new methodology to calculate the reliability polynomial of undirected networks modifies the recursive Contraction-Deletion (DC) algorithm to reduce the computational time by including fast algorithms for specific families of graphs. Each iteration of the DC algorithm contracts produces two sub-graphs by contracting and deleting an edge, this process is recursively performed until the trivial network is reached, and from that point, the algorithm recursively constructs the reliability. The purposed modified version of the DC, adds modules that contain two operations, identification, and calculation. Each of these modules focuses on a specific family of networks, therefore the identification operation determines if the given network belongs to the module family and if so, the calculation method calculates the reliability of such network using the specific methodology for that family. For each iteration of the new DC algorithm, the network is evaluated by the modules. If a module can identify the network, the reliability is calculated and retrieved using the methodologies of that module. If not, another module is asked. If any of the modules could identify the network, another iteration of the DC algorithm is performed which generates two sub-networks. Then the algorithm starts again until the original network reliability is calculated.

Results

The research characterized the construction of UMR hamiltonian networks for $m = n + 1$ and $m = n + 2$. Although the case $m = n + 3$ could not be demonstrated, a computational approach had been given and conjectured how to construct UMR networks for this case. Moreover, the study demonstrated the non-existence of UMR for hamiltonian graphs with:

- $m = \binom{n}{2} - \frac{n+2}{2}$ for all $n \geq 6$ even;
- $m = \binom{n}{2} - \frac{n+5}{2}$ for all $n \geq 7$ odd.

To calculate the reliability of undirected graphs, the presented modular methodology proved to be effective to reduce the computational time with respect to the original DC in many cases. Such cases depend on the type of given graph and the included modules. Following is given an overview of the functionalities provided by this new methodology:

- Possibility to choose the functionality that selects the edge which is contracted/deleted. The currently implemented functionalities can randomly choose an edge, or select the edge with maximum order (sum of the order of its endpoints), or the edge with minimum order.
- Possibility to choose the active modules. The included modules work with the following families: multi-trees, multi-cycles, cycle-trees (trees where the edges are replaced by cycles), and cake networks.
- Possibility to separate all the ‘tree’ parts of the given network and generate sub-networks with all the components.
- The implemented algorithm structure allows to easily add new modules or methodologies to select the contracted/deleted edge without modifying the original code.

7.1.2 Spatial Network with Point Processes

The main characteristic of a geographically referred structure modeled into a spatial network is that the nodes and edges are located in the space and can be measured, i.e. with a metric system. A point process is any stochastic mechanism that generates a finit set of random events (points) located in a bounded region. For instance, if such events happen in a network-structured spatial space, then, the interrelation of the events with the network, can be analyzed using several methodologies proper from Graph Theory in combination with point process theory.

State of art

The studies of Okabe et al. [35], Okabe and Yamada [36], and Ang et al. [1], are notable contributions to the analysis of events occurring on linear spatial networks. Such events have space-time dependence and are restricted to linear structures which might represent, for instance, road accidents, street crimes and fires caused by high-voltage networks. This characteristic allows the creation of weights based on the events and the possibility to consider weighted networks to analyze the resulting network structure. In our case, we have considered the point intensity of wildlife-vehicle collisions (WVC) to obtain weighted networks. This type of accident is a great danger for both society and wildlife [53]. For this reason, in recent years, it has been a growing interest to develop models to analyze such road accidents [25], where the detection of hot-spot locations [27, 41] is a central issue. In Section 5.2, we analyzed several variables related to WVC and propose a new methodology to detect the most dangerous areas that are more likely to have a WVC. Moreover, our new approach also calculates the safest route between any pair of points (origin and destination) in the network.

Most of the ***R*** packages are limited in terms of modeling and analyzing spatial points occurring over complex real-world networks containing directed and undirected edges. In particular, the ***R*** packages ***spatstat*** [4, 3], ***DRHotNet*** [6], ***geonet*** [49], ***spNetwork*** [21], ***SpNetPrep*** [7] and ***stlnpp*** [32], are not helpful to analyze mixed networks nor to compute point attributes at distinct configurations such as the network paths or sets of neighbors. Section 5.3 provides a new tool in form of an ***R*** package that addresses these limitations. The new package is based on the works of Eckardt and Mateu [15, 16] which introduces a different approach to analyzing network-based event data through different network intensity functions.

Methodology

Section 5.2 presents a new modeling approach based on weighted networks to find optimal paths in the network to reduce the risk of having a WVC. In addition to WVC, we have considered road types, speed limits, traffic density, and vegetation density (NDVI) surrounding the roads for our analysis. We have adapted the road structure data into a connected mathematical network where its nodes and edges contain geographical information such as latitude and longitude. The road safety is modeled by assigning a weight to each linear segment of the network based on an estimator of the point intensity, and the values of the rest of the covariables associated with this linear segment.

The weights of the segments are calculated via the following linear combination

$$W(l_i) = a\frac{\Lambda(l_i) - \min(\Lambda(l))}{\max(\Lambda(l)) - \min(\Lambda(l))} + \sum_{j=1}^{v} b_j \frac{Z_j(l_i) - \min(Z_j(l))}{\max(Z_j(l)) - \min(Z_j(l))} \tag{7.1}$$

for $i = 1, \ldots, s$, where s is the total number of segments and $W(l_i)$ is the global weight related to segment l_i. Here $\Lambda(l_i)$ is the point intensity for segment l_i. The variable associated with this segment l_i is represented by $Z_j(l_i)$, where $j = 1, \ldots, v$ identifies the variable type from the total number of types v. Then, $a + b_1 + \ldots + b_v = 1$, and $\max(\Lambda(l))$ and $\min \Lambda(l))$ (say) is the maximum and the minimum value of all $\Lambda(l_i)$, for $i = 1, \ldots, s$. The values a and b_j allow us to have better control of the weights from the different factors that affect the WVCs. Then, the safest paths between any two points are calculated using algorithms such as Depth-First Search (DFS) and Dijkstra.

The ***R*** package ***intensitynet*** presented in Section 4.3, contains 3 S3 programming structure classes that work with undirected, directed, and mixed graphs respectively. The proper class for the given network is automatically chosen by the package. Since the package uses the well-known ***igraph R*** package, the output network model can be easily treated by other packages. The study also provides examples to easily import data sources in formats like shapefile (shp) commonly used in cartographic tools such as ArcGIS [43] or QGIS [40], and linear point process objects (lpp) generated by the ***spatstat*** package. Moreover, the presented ***intensitynet*** package provides methodologies to; manipulate data, estimate intensities, compute local and global autocorrelation statistics, data visualization, and other minor methodologies.

Results

Our new approach presented in Section 5.2 has been tested with a real dataset involving 491 WVCs located in a square area of 40 × 40 square km with a road extension of 459050 km around the city of Lleida, Spain. This study also presents an interactive program made in ***R*** language that can be found in the ***GitHub*** repository. This program graphically represents the network, calculate and shows the minimum paths, between two given points, provides information about the attributes stored in the nodes and edges, and plots heatmaps based on the selected variables.

All the methodologies offered by the ***intensitynet*** package had been tested with a real dataset from Chicago city provided by the ***spatstat*** package. This dataset is also found in the presented package and can be used to test it. An insight into each set of methodologies that the package offers is listed below:

- Manipulate data such as creating spatial network models, removing or adding nodes and edges, inserting data into the network model, and retrieving such data.
- Estimate intensities related to the point patterns occurring near nodes and edges.
- Compute local and global autocorrelation statistics, for example, Moran I [33], Geary's C [20], and Getis and Ords G [23].
- Data visualization, for instance, variable-related heatmaps, network paths, occurrences, and the creation of sub-windows focused on specific parts of the network.
- Minor methodologies such as calculating distances between points, providing the shortest paths, and retrieving the total weight from a path based on a selected variable.

7.2 Thesis Discussion

This thesis began with the analysis of network reliability where different network structures, methodologies, and results were studied. We developed several new methods to reduce the computational time required for the computation of network reliability such as the methodology shown in the poster *'On the computation of the reliability polynomial of cycle graphs with ordered path-chords'* presented in ACCOTA conference in Mexico, 2018 (see section 6.1.1). These methodologies were implemented in ***Python*** and used internally to computationally study the reliability of different network configurations. From these configurations, the network family named 'Cake' was found interesting due to the apparent relation between its symmetry and high reliability. Therefore, this family belonging to the hamiltonian networks became the focus of the research. We found that these networks are uniformly most reliable (UMR) when $m = n+1$, $m = n+2$, and $m = n+3$. The cases for $m = n + 1$, $m = n + 2$ were proved and the case $m = n + 3$ was conjectured based on computational proofs (see Section 4.2).

During the UMR networks study, we developed a new algorithm that combines the deletion-contraction methodology with specific algorithms to compute the reliability of specific graph families. Later, this new algorithm, together with previous algorithms purposed at the beginning, were reviewed, structured, and implemented into classes and sub-classes forming a ***Python*** package. This package was developed using dynamic functions and unitary methodologies to provide modularity and scalability. The package was named ***atr*** the paper *'Network reliability in hamiltonian graphs'* describes all the functionalities and provides use-case examples (see Section 4.3).

The research continued by studying the dependence structure of events occurring on networks and the possibility to consider the reliability of networks based on these events. For this purpose, a real dataset was considered involving road networks and WVCs. Since the nature of the roadkills is at random (a point pattern) the possible study of the all-terminal reliability of the network based on the intensity of accidents for each segment (as a weight) could not be done (by the time being). However, we found interesting to analyze the resulting weighted networks based on point patterns, to find optimal paths between points on the network, and other relations between point processes and spatial networks.

The motivation of the new study was to increase road safety either for animals or human lives. This objective was performed by optimizing paths between points on the network structure based on WVCs and other covariates affecting this type of accident. We developed a new approach that integrates all these variables and we tested the resulting algorithm with a real network model. We found different hotspots where more WVCs were concentrated, considered road heatmaps to display the previous variables (for instance, WVCs intensity, daily average traffic volume and vegetation density), and finally, we calculated optimal safest routes from an origin to a destination point (see Section 5.2).

The study of point processes on spatial networks continued in collaboration with the Humboldt University of Berlin. During the stay in Berlin, we started the development of a new ***R*** package named ***intensitynet*** to fill the gap with the other existent packages which do not work with complex networks that combines undirected and directed edges (see Section 5.3). In this package, we implemented several methodologies to analyze and visualize networks and their correlation with the occurrences happening over them. Such methodologies were also tested using a real data set provided by the package ***spatstat*** which contains the spatial locations of the crimes reported between 25 April to 8 May of 2002 near the University of

Chicago (Illinois, USA).

There were ups and downs during the development of this thesis but prevailed the will to give more light to blur areas and to provide tools and methodologies accessible to all. Although this research was not exempt from limitations and difficulties, interesting future research lines and possible upgrades were found.

7.3 Limitations and Difficulties

As in life, nothing is exempt from drawbacks, and neither is in research. This section is dedicated to describing the current limitations found in the presented research. The following sub-sections present the limitations and difficulties of each study.

7.3.1 Constructing more than $m = n + 2$ hamiltonian networks

Section 4.2 characterized the construction of UMR hamiltonian until $m = n+2$ and purposed a hypothesis based on a computational approach for the case of $m = n+3$. Although the results seem to point to the Fair Cake graph as the uniformly most reliable construction for any $m = n+c$ (where c is a chord), the study demonstrated that this statement does not hold for $m \geq n+4$. We also found the nonexistence of UMR in certain cases. Moreover, when the number of chords (c) is increased, the number of possible combinations grows exponentially. These findings suggest that generalizing a methodology is a difficult problem, and even for a specific number of chords.

7.3.2 Network identification

The new methodology presented in Section 4.3 that calculates the all-terminal reliability polynomial based on the Deletion-Contraction algorithm presents one drawback, the identification procedure. For instance, if an algorithm to calculate the reliability is directly applied to a graph with a computational time t, using our new methodology, also adds the identification time d, therefore being the total time (T) $T = t + d$. This additional cost is easily amortized if d is comparatively small, say $O(n)$, and the identification results are positive. If the identification cost is big or the identification is always negative in any of the sub-graphs generated by the methodology, the total cost to calculate the reliability of the original graph could be unnecessarily increased. To solve this issue, the time to identify the network must be reduced as much as possible, and the modules carefully selected depending on the given network.

7.3.3 Data preparation

To illustrate our new framework to find optimal paths (see Section 5.2), we consider a real dataset involving WVCs and some other covariates related to this type of accident. This process was very complex due to the nature of these types of data. Here, the data came in different data-sets from different sources and with different formats. To develop and illustrate our new approach, the data must be carefully chosen. To this end, we studied the interrelation between the distinct considered covariates and their connection with the number of accidents. Once the data was chosen, then it needed to be properly treated beforehand for its later usage. This implies cleaning and removing duplicates, inconsistencies and noise data, and formatting such that all the data-sources share the same structure. Also, note that the amount of data must be large enough to perform statistical analysis. Such procedures might imply some difficulties to add more variables into the model, even when the data is available.

7.3.4 Network size

The ***R*** package ***intensitynet*** detailed in Section 5.3 provides many methodologies to manipulate network data, estimate intensities, compute autocorrelation statistics, visualize data, and other minor functionalities. Although in the development of the package the computational times are intended to be as low as possible, the computing time highly depends on the network size. For instance, the functions *intensitynet()* and *RelateEventsToNetwork()* can be used separately to avoid unnecessary computations. This presents the challenge to optimize each functionality and computation which is complicated due to the size of the package and the interrelation of the methodologies.

7.4 Implications

Each of the following sub-sections analyzes the implications of the different studies presented in this Thesis.

7.4.1 Seminal work with UMR hamiltonian networks

The study presented in Section 4.2 is a seminal work in the area of uniform reliability since to the best of our knowledge is the first research that explicitly studies the UMR problems inside the hamiltonian family. Therefore this study prepares the ground for a new line of research.

7.4.2 An adaptive and scalable tool

The ***Python*** package ***atr*** introduced in Section 4.2 implements a methodology to calculate the all-terminal reliability from any undirected network. From its modular nature, it can be easily expanded and improved to reduce the computing time for more network families. Moreover, its adaptability to fit the requirements of any given network makes it a useful tool for any network-oriented research field. These characteristics imply that the usage of the tool directly affects its capabilities making it a dynamic tool. Therefore, the more the tool is used and upgraded to fit the necessities of different problems, the more extensive its benefits will be.

7.4.3 Spatial networks;

The study detailed in Section 5.2 purposes the use of weighted networks to obtain optimal paths in this spatial structure based on spatial point patterns. For this case, we considered WVCs. However, this methodology can be extended to analyze any spatial structure that occurs on linear networks, for instance, in electricity networks to analyze power cuts and in water pipeline systems to identify water quality based on possible contamination sources. Moreover, the study focused on WVCs already has direct implications, such as saving animal and human lives, avoiding property damages to vehicles, reducing the spent budget on special services (ambulance and police), and other indirect implications such as improving wildlife management and building road infrastructures.

7.4.4 Improve network analysis

Most of the present statistical tools to analyze networks with point patterns do not provide the capabilities to analyze complex network structures containing directed and undirected edges. The ***netintensity*** package presented in Section 5.3 addresses these limitations by treating the edges as fundamental entities. This perspective facilitates the calculation of intensities and related quantities over different types of network structures including directed, undirected, or a combination of both i.e. mixed edges. All event-based features are derived directly from edge-related intensities where each edge intensity is calculated as the sum of all occurrences that fall over the edge, adjusted (divided) for its spatial length. Moreover, the package eases the network statistical analysis by building its methodologies around one constructor function (*intensitynet()*) which automatically parses the network structure to all computations.

7.5 Possible future research lines and upgrades

The research presented in this Thesis opened several possibilities for expansion. This section summarizes such possibilities for each of the studies.

Network reliability in hamiltonian graphs

- Study the existence of uniformly most reliable hamiltonian networks for other values of n and m.
- Study other families of networks to find uniformly most reliable constructions.

Computing all-terminal network reliability: A new modular methodology

- Add new modules to calculate the reliability of different network families.
- Add new functionalities to select the edge for its deletion/contraction.
- Optimize the computational time of the identification methodology for each module.

Weighted spatial networks: Modeling road traffic safety based on point patterns of wildlife-vehicle collisions

- Define other functions to combine different key variables.
- Incorporate more variables into the methodology.

- Adapt our approach to other problems, for instance, to define road safety from two points of view, one regarding the wildlife crossing the road and the other regarding the vehicles and its passengers.

intensitynet: Intensity-based Analysis of Spatial Point Patterns Occurring on Complex Networks Structures in R

- Add more input possibilities to initialize an *intensitynet* object.
- Add an event legend to the resulting figures from *plot()* and *PlotHeatmap()* functions.
- Add multiple file options to export the network model generated by the package.
- Create an interactive app with *Shiny* package to visually access the functionalities of the ***intensitynet*** package.

7.6 Conclusion

Networks are present in many areas, and this trend is increasing exponentially in recent times. Therefore it is crucial to understand their structures, behavior, and interrelations with internal and external variables to create optimal networks, solve problems, and make predictions. To this end, the research presented in this thesis focuses on two main issues, one shared with all types of networks which is the reliability, and the second, spatial networks with point patterns, affecting a large set of existing networks. Although the size and complexity of networks significantly affect the difficulty of their analysis, the methodologies and tools presented in this thesis are aimed to ease and enhance their study and to provide base structures designed to be easily expandable. For instance, in Section 4.2 we present a new research line to study the uniformly-most reliable hamiltonian networks and provide a seminal work addressing this topic. In Section 4.3 we detail a new modular methodology implemented in a ***Python*** package to calculate the reliability polynomial of undirected networks which can be easily expanded through new modules without modifying the main structure. In Section 5.2 we present a general framework to model events that occur on a linear network, to model, for instance, road safety based on several variables. This flexible framework allows removing or adding variables to adapt network characteristics to the problem under analysis. And finally, Section 5.3, presents a series of methodologies to analyze spatial networks and the events occurring over them implemented in form of an ***R*** package designed in a modular programming structure.

This thesis has been possible thanks to all the research done by many fellow researchers throughout time and its intention is to facilitate the future research that is to come. As Newton rightly said ‘If I have seen further, it is by standing on the shoulders of giants.’

Bibliography

[1] Qi Wei Ang, Adrian Baddeley, and Gopalan Nair. Geometrically corrected second order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics*, 39(4):591–617, 2012.

[2] Adrian Baddeley, Gopalan Nair, Suman Rakshit, Greg McSwiggan, and Tilman M Davies. Analysing point patterns on networks — a review. *Spatial Statistics*, 42:100435, 2021.

[3] Adrian Baddeley, Ege Rubak, and Rolf Turner. *Spatial Point Patterns: Methodology and Applications with R.* Chapman and Hall/CRC Press, 2015.

[4] Adrian Baddeley and Rolf Turner. spatstat: An R package for analyzing spatial point patterns. *Journal of Statistical Software*, 12(6):1–42, 2005.

[5] F. T. Boesch, X. Li, and C. Suffel. On the existence of uniformly optimally reliable networks. *Networks*, 21(2):181–194, 1991.

[6] Alvaro Briz-Redon. *DRHotNet: Differential Risk Hotspots in a Linear Network*, 2021. R package version 2.0.

[7] Alvaro Briz-Redon. *SpNetPrep: Linear Network Preprocessing for Spatial Statistics*, 2021. R package version 1.2.

[8] D. Bulka and J.B. Dugan. A lower bound on the reliability of an n-dimensional hypercube. *Proceedings Ninth Symposium on Reliable Distributed Systems*, pages 44–53, 1990.

[9] Juan Manuel Burgos. Factorization of network reliability with perfect nodes ii: Connectivity matrix. *Discrete Applied Mathematics*, 198:91–100, 2016.

[10] Juan Manuel Burgos and Franco Robledo Amoza. Factorization of network reliability with perfect nodes i: Introduction and statements. *Discrete Applied Mathematics*, 198:82–90, 2016.

[11] Eduardo Canale, Héctor Cancela, Franco Robledo, Pablo Romero, and Pablo Sartor. Full complexity analysis of the diameter-constrained reliability. *International Transactions in Operational Research*, 22(5):811–821, 2015.

[12] C.J. Colbourn. Edge-packing of graphs and network reliability. *Discrete Math*, 72(1):49–61, 1988.

[13] Ioannis C. Drivas, Damianos P. Sakas, Georgios A. Giannakopoulos, and Daphne Kyriaki-Manessi. Big data analytics for search engine optimization. *Big Data and Cognitive Computing*, 4(2), 2020.

[14] Florian Dörfler, John W. Simpson-Porco, and Francesco Bullo. Electrical networks and algebraic graph theory: Models, properties, and applications. *Proceedings of the IEEE*, 106(5):977–1005, 2018.

[15] Matthias Eckardt and Jorge Mateu. Partial and semi-partial statistics of spatial associations for multivariate areal data. *Geographical Analysis*, 53(4):818–835, 2021.

[16] Matthias Eckardt and Jorge Mateu. Second-order and local characteristics of network intensity functions. *TEST*, 30(2):318–340, 2021.

[17] T. Evans and D. Smith. Optimally reliable graphs for both edge and vertex failures. *Networks*, 16(2):199–204, 1986.

[18] O. Frank and W. Gaul. On reliability in stochastic graphs. *Networks*, 12(2):119–126, 1986.

[19] Ramón García-Domenech, Jorge Gálvez, Jesus V. de Julián-Ortiz, and Lionello Pogliani. Some new trends in chemical graph theory. *Chemical Reviews*, 108(3):1127–1169, 2008. PMID: 18302420.

[20] R. C. Geary. The contiguity ratio and statistical mapping. *The Incorporated Statistician*, 5(3):115–146, 1954.

[21] Jeremy Gelb. *spNetwork: Spatial Analysis on Network*, 2021. R package version 0.2.1.

[22] Ilya Gertsbakh, Yoseph Shpungin, and Radislav Vaisman. *Networks with Ternary Components: Ternary Spectrum*, pages 1–23. Springer International Publishing, Cham, 2014.

[23] Arthur Getis and J. K. Ord. The analysis of spatial association by use of distance statistics. *Geographical Analysis*, 24(3):189–206, 1992.

[24] D. Gross and J. T. Saccoman. Uniformly optimally reliable graphs. *Networks*, 31(4):217–225, 1998.

[25] K.E. Gunson, G. Mountrakis, and L.J. Quackenbush. Spatial wildlife-vehicle collision models: A review of current work and its application to transportation mitigation projects. *Journal of Environmental Management*, 92:1074–1082, 2011.

[26] R.M. Karp and M. Luby. Monte-carlo algorithms for the planar multiterminal network reliability problem. *J. Complex.*, 1(1):45–64, 1985.

[27] J.A. Litvaitis and J.P. Tash. Tan approach toward understanding wildlife-vehicle collisions. *Environmental Management*, 42:688–697, 2008.

[28] Abdul Majeed and Ibtisam Rauf. Graph theory: A comprehensive survey about graph theory applications in computer science and social networks. *Inventions*, 5(1), 2020.

[29] Edward C. Malthouse and Hairong Li. Opportunities for and pitfalls of using big data in advertising research. *Journal of Advertising*, 46(2):227–235, 2017.

[30] Mohamed HS Mohamed, Yang Xiaozong, Liu Hongwei, and Wu Zhibo. An efficient evaluation for the reliability upper bound of distributed systems with unreliable nodes and edges. *Int. J. Eng. Technol*, 2(2):107–110, 2010.

[31] E.F. Moore and C.E. Shannon. Reliable circuits using less reliable relays. *Journal of the Franklin Institute*, 262(3):191 – 208, 1956.

[32] Mehdi Moradi, Ottmar Cronie, and Jorge Mateu. *stlnpp: Spatio-temporal analysis of point patterns on linear networks*, 2020.

[33] P. A. P. Moran. Notes on continuous stochastic phenomena. *Biometrika*, 37(1/2):17–23, 1950.

[34] Wendy Myrvold, Kim H. Cheung, Lavon B. Page, and Jo Ellen Perry. Uniformly-most reliable networks do not always exist. *Networks*, 21(4):417–419, 1991.

[35] A. Okabe, H. Yomono, and M. Kitamura. Statistical analysis of the distribution of points on a network. *Geographical Analysis*, 27(2):152–175, 1995.

[36] Atsuyuki Okabe and Ikuho Yamada. The k-function method on a network and its computational implementation. *Geographical Analysis*, 33(3):271–290, 2001.

[37] Hebert Perez-Roses. Sixty years of network reliability. *Mathematics in Computer Science*, 12(3):275–293, 6 2018.

[38] T. Politof and A. Satyanarayana. Efficient algorithms for reliability analysis of planar networks - a survey. *IEEE Transactions on Reliability*, 35(3):252–259, 1986.

[39] T. Politof and A. Satyanarayana. Efficient algorithms for reliability analysis of planar networks - a survey. *IEEE Transactions on Reliability*, 35(3):252–259, 1986.

[40] QGIS Development Team. *QGIS Geographic Information System.* QGIS Association, 2022.

[41] D. Ramp, J. CaldwellS, K.A. Kathryn, and D. Warton D.B Croft. Modelling of wildlife fatality hotspots along the snowy mountain highway in new south wales, australia. *Biological Conservation*, 126:474–490, 2005.

[42] Jakob G. Rasmussen and Heidi S. Christensen. Point processes on directed linear networks. *Methodology and Computing in Applied Probability*, 23(2):647–667, 2021.

[43] CA: Environmental Systems Research Institute Redlands. *ArcGIS Geographic Information System*, 2011.

[44] B. D. Ripley. Modelling spatial patterns. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(2):172–192, 1977.

[45] P. Romero. Building uniformly most-reliable networks by iterative augmentation. In *2017 9th International Workshop on Resilient Networks Design and Modeling (RNDM)*, pages 1–7, Sep. 2017.

[46] A. Satyanarayana and Mark K. Chang. Network reliability and the factoring theorem. *Networks*, 13(1):107–120, 1983.

[47] A. Satyanarayana and R. KevinWood. A linear-time algorithm for computing k-terminal reliability in series-parallel networks. *SIAM J. Comput.*, 14(4):818–832, 1985.

[48] A Satyanarayana and R Tindell. Efficient algorithms for the evaluation of planar network reliability. Technical report, DTIC Document, 1993.

[49] Marc Schneble. *geonet: Intensity Estimation on Geometric Networks with Penalized Splines*, 2021. R package version 0.6.0.

[50] Ahmad R. Sharafat and Omid R. Ma'rouzi. All-terminal network reliability using recursive truncation algorithm. *IEEE Transactions on Reliability*, 58(2):338–347, 2009.

[51] Derek H. Smith and Lynne L. Doty. On the construction of optimally reliable graphs. *Networks*, 20(6):723–729, 1990.

[52] Chat Srivaree-ratana, Abdullah Konak, and Alice E. Smith. Estimation of all-terminal network reliability using an artificial neural network. *Computers & Operations Research*, 29(7):849–868, 2002.

[53] Rodney van der Ree, Daniel J Smith, Clara Grilo, et al. The ecological effects of linear infrastructure and traffic: challenges and opportunities of rapid global growth. *Handbook of road ecology*, pages 1–9, 2015.

[54] R.M. Van Slyke and H. Frank. Network reliability analysis: part i. *Networks*, 1:279–290, 1971.

[55] Takashi Washio and Hiroshi Motoda. State of the art of graph-based data mining. *SIGKDD Explor. Newsl.*, 5(1):59–68, jul 2003.

[56] Wenchao Xu, Haibo Zhou, Nan Cheng, Feng Lyu, Weisen Shi, Jiayin Chen, and Xuemin Shen. Internet of vehicles in big data era. *IEEE/CAA Journal of Automatica Sinica*, 5(1):19–35, 2018.

[57] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, 48(1):1–36, 2022.

[58] Qing Zhou, Long Gao, Ruifang Liu, and Shuguang Cui. *Network Robustness under Large-Scale Attacks*. Springer Science & Business Media, 2012.